Брюс Шнайер

Прикладная криптография

2-е издание

Протоколы, алгоритмы и исходные тексты на языке C

СОДЕРЖАНИЕ

Глава 3
Основные протоколы

3.1 Обмен ключами
3.2 Удостоверение подлинности
3.3 Удостоверение подлинности и обмен ключами
3.4 Формальный анализ протоколов проверки подлинности и обмена ключами
3.5 Криптография с несколькими открытыми ключами
3.6 Разделение секрета
3.7 Совместное использование секрета
3.8 Криптографическая защита баз данных

Глава 4
Промежуточные протоколы

4.1 Службы меток времени
4.2 Подсознательный канал
4.3 Неотрицаемые цифровые подписи
4.4 Подписи уполномоченного свидетеля
4.5 Подписи по доверенности
4.6 Групповые подписи
4.7 Подписи с обнаружением подделки
4.8 Вычисления с зашифрованными данными
4.9 Вручение битов
4.10 Подбрасывание "честной" монеты
4.11 Мысленный покер
4.12 Однонаправленные сумматоры
4.13 Раскрытие секретов "все или ничего"
4.14 Условное вручение ключей

Глава 5
Развитые протоколы

5.1 Доказательства с нулевым знанием
5.2 Использование доказательства с нулевым знанием для идентификации
5.3 Слепые подписи
5.4 Личностная криптография с открытыми ключами
5.5 Рассеянная передача
5.6 Рассеянные подписи
5.7 Одновременная подпись контракта
5.8 Электронная почта с подтверждением
5.9 Одновременный обмен секретами

Глава 6
Эзотерические протоколы

6.1 Безопасные выборы
6.2 Безопасные вычисления с несколькими участниками
6.3 Анонимная широковещательная передача сообщений
6.4 Электронные наличные

Часть II Криптографические методы

Глава 7
Длина ключа

Глава 8
Управление ключами

Глава 9
Типы алгоритмов и криптографические режимы

Глава 10 (Текст главы на английском, sorry. Переводчик, похоже, устал :-)
Использование алгоритмов

Часть III Криптографические алгоритмы

Глава 11
Математические основы

Глава 12
Стандарт шифрования данных DES

Глава 13
Другие блочные шифры

Глава 14
И еще о блочных шифрах

Глава 15
Объединение блочных шифров

Глава 16
Генераторы псевдослучайных последовательностей и потоковые шифры

Глава 17
Другие потоковые шифры и генераторы настоящих случайных последовательностей

Глава 18
Однонаправленные хэш-функции

Глава 19
Алгоритмы с открытыми ключами

Глава 20
Алгоритмы цифровой подписи с открытым ключом

Глава 21
Схемы идентификации

Глава 22
Алгоритмы обмена ключами

Глава 23
Специальные алгоритмы для протоколов

Часть IV Реальный мир

Глава 24
Примеры реализаций

Глава 25
Политика

Мэтт Блейз. Послесловие

Часть V Исходные коды

1. DES
2. LOKI91
3. IDEA
4. GOST
5. BLOWFISH
6. 3-WAY
7. RC5
8. A5
9. SEAL

Библиография

Прикладная криптография

2-е издание

Протоколы, алгоритмы и исходные тексты на языке С

Брюс Шнайер

Предисловие
Уитфилд Диффи

История литературы по криптографии довольно любопытна . Секретность, конечно же, всегда играла важную роль, но до Первой мировой войны о важных разработках время от времени сообщалось в печати и криптогр афия развивалась также, как и другие специализированные дисциплины. В 1918 году в виде научного отчета ч астной Лаборатории Ривербэнк вышла в свет монография Вильяма Ф. Фридмана *Показатель совпадений и его применения в криптографии* (*Index of Coincidence and Its Applications in Cryptography*) [577], одна из определяющих работ 20-го столетия. И это несмотря на военный заказ, по которому была сделана эта работа. В том же году Эдвард Х. Хеберн из Окленда, Калифорния, получил первый патент [710] на роторную машину, устройство, на котором основывалась военная криптография в течение почти 50 лет.

После Первой мировой войны, однако, все изменилось. Организации армии и флота Соединенных Штатов, полностью засекретив свои работы, добились фундаментальных успехов в криптографии. В течение 30 -х и 40-х годов в открытой литературе по данному предмету появлялись только отдельные основные работы и моногр афии, но чем дальше, тем меньше они соответствовали реальному положению дел. К концу войны переход по лностью завершился. Открытая литература умерла за исключением одного заметного исключения, работы Клода Шэннона "The Communication Theory of Secrecy systems" (*Теория связи между секретными системами*), напечатанной в 1949 году в *Bell System Technical Journal* [1432]. Эта статья, как и работа Фридмана в 1918 году, явилась результатом исследований Шэннона во время войны. После окончания Второй мировой войны она была рассекречена, возможно по ошибке.

С 1949 по 1967 литература по криптографии была бессодержательной. В 1967 году она пополнилась работой другого типа, историей Дэвида Кана *Дешифровщики* (*The Codebreakers*) [794]. В этой книге не было новых идей, но она содержала достаточно полную историю предмета, включая упоминание о некоторых вещах, все еще засекреченных правительством. Значение *Дешифровщиков* заключалось не только в значительном охвате предмета, книга имела заметный коммерческий успех и познакомила с криптографией тысячи людей, раньше и не задумывавшихся о ее существовании. Тоненьким ручейком начали появляться новые работы по криптогр афии.

Почти в то же время Хорста Фейстела, ранее работавшего над прибором "свой/чужой" для ВВС, на всю дальнейшую жизнь охватила страсть к криптографии, и он перешел в Уотсоновскую Лабораторию фирмы IBM, расположенную в Йорктаун Хайтс, Нью-Йорк. Там он начал разработку того, что затем стало стандартом DES (U.S. Data Encryption Standard, Стандарт шифрования данных Соединенных Штатов). В начале 70-х годов IBM опубликовала ряд технических отчетов по криптографии, выполненных Фейстелом и его коллегами [1482, 1484, 552].

Таково было положение, когда в конце 1972 года я начал работать в этой области. Литература по крипт ографии обильной не была, но в ней можно было найти ряд сверкающих самородков.

В криптографической науке есть особенность, отсутствующая в обычных академических дисциплинах: нео бходимость взаимодействия криптографии и криптоанализа. Причиной этого является отсутствие требований к передаче реальной информации, следовательно, нетрудно предложить систему, которая кажется непогрешимой. Многие академические разработки настолько сложны, что будущий криптоаналитик не знает с чего начать. О бнаружить дыры в этих проектах намного сложнее, чем разработать их. В результате невозможно соревнование, являющееся одним из сильнейших мотивов в академических исследований.

Когда Мартин Хеллман и я в 1975 году предложили криптографию с открытыми ключами [496], одним из косвенных аспектов нашего предложения было появление проблемы, решение которой не кажется простым. Теперь честолюбивый проектировщик мог создать что-то - вполне разумную криптосистему, решающую более обширные задачи, чем простое превращение значимого текста в чепуху. В результате значительно возросло число людей, занимающихся криптографией, число проводимых встреч и число опубликованных книг и статей.

В речи по поводу присуждения мне совместно с Мартином Хеллманом премии Дональда Е. Финка (присуждаемой за лучшую пояснительную статью в журнале IEEE) я сказал, что, написав "Privacy and Authentication" ("Секретность и удостоверение подлинности"), я получил опыт, который необычен даже для выдающи хся ученых, получивших премии IEEE. Я написал статью, которую я хотел бы изучить, когда я впервые серьезно заинтересовался криптографией, и которую не смог найти. Если бы я сегодня отправился в Стэнфордскую би блиотеку и собрал бы современные работы по криптографии, я, возможно, получил бы представление о предмете гораздо раньше. Но осенью 1972 года были доступны только несколько классических работ и ряд туманных технических отчетов.

У сегодняшнего исследователя нет такой проблемы. Сегодня основная сложность состоит в выборе, с чего

начать среди тысяч статей и десятков книг. А сегодняшние программисты и инженеры, которые просто хотят использовать криптографию? К каким источникам им обращаться? До сих пор необходимо было проводить долгие часы, выискивая научную литературу и изучая ее, прежде чем удавалось начать разработку криптогр афических приложений, так гладко описанных в популярных статьях.

Именно этот промежуток и призвана заполнить *Прикладная криптография* Брюса Шнайера. Начав с целей засекречивания передачи данных и элементарных примеров программ для достижения этих целей, Шнайер ра зворачивает перед нами панораму результатов 20 лет открытых исследований. Содержание книги полностью определяется ее названием, вы найдете в ней описание различных приложений, от засекречивания телефонного разговора до электронных денег и криптографического обеспечения выборов.

Не удовлетворенный простым изложением алгоритмов и описанием кода, Шнайер включил в книгу обсу ждение различных мировых организаций, связанных с разработкой и применением криптографических средств, от Международной ассоциации криптологических исследований до NSA (National Security Agency, Агентство национальной безопасности).

Когда на рубеже 70-х и 80-х годов возрос общественный интерес к криптографии, NSA, официальный крип тографичекий орган США, предприняло ряд попыток подавить этот интерес. Первой такой попыткой было письмо старого сотрудника NSA, по видимому действовавшего по своему усмотрению. Письмо было послано в IEEE и предупреждало, что публикация материалов по криптографии является нарушением Правил междун ародной продажи оружия (International Traffic in Arms Regulations, ITAR). Эта точка зрения, как оказалось, не поддерживаемая самими правилами, в явном виде содержащими льготы для публикуемых материалов, создала неожиданную рекламу использованию криптографии и Семинару по теории информации 1977 года.

Более серьезная попытка была предпринята в 1980 году, когда NSA финансировало изучение вопроса Ам ериканским советом по образованию с целью убедить Конгресс узаконить контроль над публикациями в области криптографии. Результаты, оказавшиеся далекими от ожиданий NSA, привели к программе добровольного р ецензирования работ по криптографии. От исследователей потребовали перед публикацией запрашивать мнение NSA, не принесет ли раскрытие результатов исследований вред национальным интересам.

К середине 80-х годов основным объектом внимания стала не теория, а практика криптографии. Сущес т-вующие законы дают NSA право с помощью Госдепартамента регулировать экспорт криптографического об орудования. Так как бизнес все больше и больше принимает международный характер и американская часть м ирового рынка уменьшается, возрастает желание использовать единый продукт и для внутреннего, и для внешн его рынка. Такие продукты являются субъектами контроля над экспортом, и поэтому NSA получило возможность контролировать не только экспортируемые криптографические продукты, но и продаваемые в Соединенных Штатах.

В то время, когда писались эти строки, возникло новое препятствие для общественного использования кри птографии. Правительство дополнило широко опубликованный и используемый алгоритм DES засекреченным алгоритмом, реализованным в микросхемах памяти, независящей от времени. Эти микросхемы будут содержать кодифицированный механизм правительственного контроля. Отрицательные аспекты такой програ м-мы-троянского коня простираются от потенциально губительного раскрытия тайны лич ности до высокой сто имости аппаратной модернизации продуктов, ранее реализованных программно. Таким образом, предлагаемое нововведение не вызвало энтузиазма и подверглось широкой критике, особенно со стороны независимых кри птографов. Ряд людей, однако, видят свое будущее в программировании, а не в политике и удваивают свои ус илия, стремясь представить миру мощные средства криптографии.

Значительное отступление от возможности того, что закон о контроле над экспортом отменит Первую п оправку[1], казалось было сделан в 1980 году, когда в опубликованные в *Federal Register* исправления ITAR вошло следующее положение: "...положение было добавлено с целью показать, что регулирование экспорта технич еских данных не приведет к конфликту с правами личности, определяемыми Первой поправкой". Но то, что ко нфликт между Первой поправкой и законами о контроле над экспортом не разрешен окончательно, должно быть очевидно из заявлений, сделанных на конференции, проводимой RSA Data Security. Представитель NSA из отдела контроля над экспортом выразил мнение, что люди, публикующие криптографические программы, нах одятся " в серой зоне" по отношению к закону. Если это так, то именно эту "серую зону" немного осветило первое издание этой книги. Экспорт приложений для этой книги был разрешен с подтверждением того, что опублик ованные материалы не попадают под юрисдикцию Совета по контролю над вооружением. Однако, экспортир овать опубликованные программы на диске было запрещено.

Изменение стратегии NSA от попыток контролировать криптографические исследования к усилению регул ирования в области разработки и развертывания криптографических продуктов по видимому обусловлено осо знанием того, что все величайшие криптографические работы не защитили ни одного бита информации. Будучи

---

[1] К конституции США

поставлен в шкаф, этот том не сделает ничего нового по сравнению с предшествующими книгами и работами, но использование его содержания на рабочей станции, где пишется криптографический код, может привести к иному результату.

Уитфилд Диффи

Маунтэйн Вью, Калифорния.

Введение

Криптография бывает двух типов: криптография, которая помешает читать ваши файлы вашей младшей с е-стре, и криптография, которая помешает читать ваши файлы дядям из правительства. Эта книга о втором типе криптографии.

Если я беру письмо, кладу его в сейф где-нибудь в Нью-Йорке, затем велю Вам прочитать это письмо, то это не безопасность. Это непонятно что. С другой стороны, если я беру письмо и кладу его в сейф, затем передаю этот сейф Вам вместе с детальным описанием, передаю также сотню подобных сейфов с их комбинациями, чт обы Вы и лучшие "медвежатники" мира могли изучить систему замков, а вы все равно не сможете открыть сейф и прочитать письмо - вот это и есть безопасность.

В течение многих лет этот тип криптографии использовался исключительно в военных целях. Агентство н ациональной безопасности Соединенных Штатов Америки (National Security Agency, NSA) и его аналоги в быв-шем Советском Союзе, Англии, Франции, Израиле и прочих странах тратили миллиарды долларов на очень серьезную игру в обеспечение безопасности собственных линий связи, одновременно пытаясь взломать все о с-тальные. Отдельные личности, обладающие значительно меньшими средствами и опытом, были беспомощны защитить свои секреты от правительств.

В течение последних 20 лет значительно вырос объем открытых академических исследований. Пока обычные граждане использовали классическую криптографию, со времен Второй мировой войны компьютерная крипт о-графия во всем мире применялась исключительно в военной области. Сегодня искусство компьютерной крипт о-графии вырвалось из стен военных ведомств. Непрофессионалы получили возможность средства, позволяющие им обезопасить себя от могущественнейших противников, средства, обеспечивающие защиту от военных в е-домств.

А нужна ли обычному человеку такая криптография? Да. Люди могут планировать политическую кампанию, обсуждать налоги, вести незаконные действия. Они могут разрабатывать новые изделия, обсуждать рыночную политику или планировать захват конкурирующей фирмы. Они могут жить в стране, которая не соблюдает з а-прета на вторжение в личную жизнь своих граждан. Они могут делать что-либо, что не кажется им незаконным, хотя таковым и является. По многим причинам данные и линии связи должны быть личными, тайными и з а-крытыми от постороннего доступа.

Эта книга выходит в свет в беспокойное время. В 1994 году администрация Клинтона приняла Стандарт у с-ловного шифрования (Escrowed Encryption Standard), включая микросхему Clipper и плату Fortezza, и превра-тило Билль о Цифровой телефонии в закон. Эти инициативы пытаются увеличить возможности правительства проводить электронный контроль.

Вступают в силу некоторые опаснейшие домыслы Оруэлла: правительство получает право прослушивать личные переговоры, а с человеком, пытающимся скрыть свои секреты от правительства, может что-нибудь сл у-читься. Законодательство всегда разрешало слежку по решению суда, но впервые люди сами должны предпр и-нимать какие-то шаги, чтобы *сделаться доступными* для слежки. Эти инициативы не просто предложения пр а-вительства в некой туманной сфере, это упреждающая и односторонняя попытка присвоить прежде принадл е-жащие людям права.

Законопроекты о микросхеме Clipper и Цифровой телефонии не способствуют сохранению тайны, но беспо ч-венно заставляют людей считать, что правительство уважает их тайны. Те же самые власти, которые незаконно записывали телефоны Мартина Лютера Кинга, могут легко прослушать телефон, защищенный микросхемой Clipper. В недавнем прошлом полицейские власти на местах были привлечены к гражданской или уголовной ответственности за незаконное прослушивание во многих судах - в Мэриленде, Коннектикуте, Вермонте, Джорджии, Миссури и Неваде. Идея развернуть технологию, которая может привести к появлению полице й-ского государства - это плохая идея.

Дело в том, что недостаточно защитить себя законами, нам нужно защитить себя математикой. Шифров а-ние имеет слишком большое значение, чтобы оставить ее использование только правительствам .

Эта книга снабдит Вас инструментарием, позволяющим защитить ваши тайны. Передача криптографич е-ских продуктов может быть объявлена незаконной, передача информации - никогда .

Как читать эту книгу

Я написал *Прикладную криптографию* как живое введение в криптографию и как всеобъемлющий спр а-вочник. Я пытался сочетать читаемость текста с жертвенной точностью, но эта книга писалась не как мат е-матическая работа. Хотя я не искажал информацию умышленно, торопясь, я опускал теорию . Для интере-сующихся теоретическими выкладками приведены обширные ссылки на академическую литературу .

Глава 1 представляет собой введение в криптографию, описывает множество терминов, в ней кратко ра с-

сматривается докомпьютерная криптография.

Главы со 2 по 6 (Часть I) описывают криптографические протоколы - что люди могут сделать с помощью криптографии - от простых (передача шифрованных сообщений от одного человека другому) до сложных (щелканье монетой по телефону) и тайных (секретное и анонимное обращение электронных денег). Некоторые из этих протоколов очевидны, другие - удивительны. Множество людей и не представляет многие из проблем, которые может решить криптография.

Главы с 7 по 10 (Часть II) содержат обсуждение методов криптографии. Все эти четыре главы важны для самых распространенных применений криптографии. В главах 7 и 8 рассказывается о ключах: какова должна быть длина безопасного ключа, как генерировать, хранить и распределять ключи, и т.д. Управление ключами представляет собой труднейшую часть криптографии и часто является ахиллесовой пятой систем, безопасных во всем остальном. В главе 9 рассматриваются различные способы использования криптографических алгоритмов, а глава 10 описывает особенности и цели использования этих алгоритмов - как их выбирать, реализовывать и применять.

Главы с 11 по 23 (Часть III) описывают эти алгоритмы. Глава 11 представляет собой математическую базу и является обязательной только, если вы интересуетесь алгоритмами с открытыми ключами. Если вы собираетесь использовать DES (или что-то похожее), ее можно пропустить. В главе 12 обсуждается алгоритм DES, его история, безопасность и разновидности. В главах 13, 14 и 15 рассказывается о других блочных алгоритмах. Если вам нужно что-то более надежное чем DES, сразу переходите к разделам о IDEA и тройном DES. При желании узнать о группе алгоритмов, некоторые из которых могут быть безопаснее DES, прочитайте всю главу. В главах 16 и 17 обсуждаются потоковые алгоритмы. В главе 18 подробно рассматриваются однонаправленные хэш-функции, среди которых самыми являются MDS и SHA, хотя я останавливаюсь и на многих других. В главе 19 рассматриваются алгоритмы шифрования с открытым ключом, а в главе 20 - алгоритмы цифровой подписи с открытым ключом. В главе 21 обсуждаются алгоритмы идентификации с открытым ключом, а в главе 22 - алгоритмы обмена с открытым ключом. Самыми важными являются алгоритмы RSA, DSA, Фиат-Шамира (Fiat-Shamir) и Диффи-Хелмана (Diffie-Hellman). Глава 23 содержит ряд эзотерических алгоритмов и протоколов с открытым ключом, математика в этой главе достаточно сложна, так что пристегните ремни.

Главы 24 и 25 (Часть IV) переносят вас в реальный мир криптографии. В главе 24 обсуждаются некоторые современные применения алгоритмов и протоколов, в то время как глава 25 касается некоторых политических аспектов криптографии. Несомненно, эти главы не являются всеохватывающими.

В книгу также включены исходные коды 10 алгоритмов, рассмотренных в Части III. Я не смог включить весь код, который хотел, из-за его большого объема, кроме того, криптографические коды в любом случае нельзя экспортировать. (Любопытно, что Госдепартамент разрешил экспортировать первое издание этой книги с исходным кодом, но не разрешил экспортировать компьютерный диск с теми же исходными кодами. Смотри рисунок.) Соответствующий набор дисков с исходным кодом содержит существенно больше исходных кодов, чем я смог включить в эту книгу, возможно, это самая большая подборка криптографических исходных кодов, появившаяся за пределами военных ведомств. Сейчас я могу переслать эти диски с исходным кодом только гражданам США и Канады, живущим в этих странах, но, возможно, когда-нибудь все изменится. Если вы собираетесь использовать или попробовать эти алгоритмы, добудьте диск. Подробности на последней странице книги..

К недостаткам этой книги относится то, что из-за ее энциклопедической природы пострадала читаемость книги. Я хотел написать единый справочник для тех, кто мог встретиться с каким-либо алгоритмом в академической литературе или при использовании какого-то продукта, и заранее извиняюсь перед теми, кто разыскивает учебное пособие. Впервые все множество сделанного в криптографии собрано под одной обложкой. Несмотря на это, соображения объема заставили меня оставить многое за пределами этой книги, я включил те темы, которые мне показались важными, практическими или интересными. Если я не мог полностью охватить тему, я приводил ссылки на соответствующие работы и статьи.

Я сделал все, что мог, пытаясь выловить и исправить все ошибки в книге, но многие люди уверяли меня, что это все равно невозможно. Конечно, во втором издании ошибок меньше, чем в первом. Перечень ошибок можно получить у меня, он также периодически рассылается в телеконференции Usenet sci.crypt. Если кто-нибудь из читателей обнаружит ошибку, пожалуйста, пусть сообщит мне об этом. Каждому, кто первый обнаружит данную ошибку в книге, я бесплатно пошлю диск с исходным кодом.

Благодарности

Перечень людей, приложивших руку к созданию этой книги, может показаться бесконечным, но все они достойны упоминания. Мне хотелось бы поблагодарить Дона Альвареса (Don Alvarez), Росса Андерсона (Ross Anderson), Дэйва Бейленсона (Dave Balenson), Карла Бармса (Karl Barms), Стива Белловина (Steve Bellovin), Дэна Бернстайна (Dan Bernstein), Эли Байем (Ell Biham), Джоан Бояр (Joan Boyar), Карен Купер (Karen Cooper), Вита Диффи (Whit Diffie), Джоан Фейгенбаум (Joan Feigenbaum), Фила Кана (Phil Karn), Нила Коблица (Neal Koblitz), Ксуейа Лай (Xuejia Lai), Тома Леранта (Tom Leranth), Майка Марковица (Mike Markowitz), Ральфа Меркла (Ralph Merkle), Билла Паттена (Bill Patten), Питера Пирсона (Peter Pearson), Чарльза Пфлегера (Charles

Pfleeger), Кена Пиццини (Ken Pizzini), Барта Пренела (Bart Preneel), Марка Риордана (Mark Riordan), Йоахима Шурмана (Joachim Schurman) и Марка Шварца (Marc Schwartz) за чтение и редактирование всего первого издания или его частей; Марка Воклера (Marc Vauclair) за перевод первого издания на французский; Эйба Абрахама (Abe Abraham), Росса Андерсона (Ross Anderson), Дэйва Бенисара (Dave Banisar), Стива Белловина (Steve Bellovin), Эли Байем (Ell Biham), Мэтта Бишопа (Matt Bishop), Мэтта Блэйза (Matt Blaze), Гэри Картера (Gary Carter), Жана Комениша (Jan Comenisch), Клода Крепо (Claude Crepeau), Джоан Дэймон (Joan Daemon), Хорхе Давила (Jorge Davila), Эда Доусона (Ed Dawson), Вита Диффи (Whit Diffie), Карла Эллисона (Carl Ellison), Джоан Фейгенбаум (Joan Feigenbaum), Нильса Фергюсона (Niels Ferguson), Матта Франклина (Matt Franklin), Розарио Сеннаро (Rosario Cennaro), Дитера Колмана (Dieter Collmann), Марка Горески (Mark Goresky), Ричарда Грэйвмана (Richard Graveman), Стюарта Хабера (Stuart Haber), Джингмана Хе (Jingman He), Боба Хэйга (Bob Hague), Кеннета Айверсона (Kenneth Iversen), Маркуса Джекобсона (Markus Jakobsson), Берта Калиски (Burt Kaliski), Фила Кана (Phil Karn), Джона Келси (John Kelsey), Джона Кеннеди (John Kennedy), Ларса Кнудсена (Lars Knudsen), Пола Кочера (Paul Kocher), Джона Лэдвига (John Ladwig), Ксуейа Лай (Xuejia Lai), Аджена Ленстры (Arjen Lenstra), Пола Лейланда (Paul Leyland), Майка Марковица (Mike Markowitz), Джима Мэсси (Jim Massey), Брюса МакНейра (Bruce McNair), Вильяма Хью Мюррея (William Hugh Murray), Роджера Нидхэма (Roger Needham), Клифа Неймана (Clif Neuman), Кейсу Найберг (Kaisa Nyberg), Люка О'Коннора (Luke O'Connor), Питера Пирсона (Peter Pearson), Рене Перальта (Rene Peralta), Барта Пренела (Bart Preneel), Израиля Радай (Yisrael Radai), Мэтта Робшоу (Matt Robshaw), Майкла Роу (Michael Roe), Фила Рогуэя (Phil Rogaway), Эви Рубина (Avi Rubin), Пола Рубина (Paul Rubin), Селвина Рассела (Selwyn Russell), Казуе Сако (Kazue Sako), Махмуда Салмасизадеха (Mahmoud Salmasizadeh), Маркуса Стадлера (Markus Stadler), Дмитрия Титова (Dmitry Titov), Джимми Аптона (Jimmy Upton), Марка Воклера (Marc Vauclair), Сержа Воденея (Serge Vaudenay), Гидеона Ювала (Gideon Yuval), Глена Зорна (Glen Zorn) и многих безымянных правительственных служащих за чтение и редактирование всего второго издания или его частей; Лори Брауна (Lawrie Brown), Лизу Кэндл (Leisa Candle), Джоан Дэймон (Joan Daemon), Питера Гутмана (Peter Gutmann), Алана Инсли (Alan Insley), Криса Джонстона (Chris Johnston), Джона Келси (John Kelsey), Ксуейа Лай (Xuejia Lai), Билла Лейнингера (Bill Leininger), Майка Марковица (Mike Markowitz), Ричарда Аутбриджа (Richard Outerbridge), Питера Пирсона (Peter Pearson), Кена Пиццини (Ken Pizzini), Кэлма Пламба (Calm Plumb), RSA Data Security, Inc., Майкла Роу (Michael Roe), Майкла Вуда (Michael Wood) и Фила Циммермана (Phil Zimmermann) за предоставленные исходные коды; Пола МакНерланда (Paul MacNerland) за создание рисунков к первому издания; Карен Купер (Karen Cooper) за редактирование второго издания; Бота Фридмана (Both Friedman) за сверку второго издания; Кэрол Кеннеди (Кэрол Kennedy) за работу над предметным указателем для второго издания; читателей sci.crypt и почтового списка Cypherpunks за комментирование идей, ответы на вопросы и поиск ошибок первого издания; Рэнди Сюсс (Randy Seuss) за предоставление доступа к Internet; Джеффа Дантермана (Jeff Duntemann) и Джона Эриксона (Jon Erickson) за то, что помогли мне начать; семью Insley (в произвольном порядке) за стимуляцию, воодушевление, поддержку, беседы, дружбу и обеды; и AT&T Bell Labs, зажегшей меня и сделавшей возможным все это. Все эти люди помогли создать гораздо лучшую книгу, чем я бы смог создать в одиночку.

Брюс Шнайер
Оак Парк, Иллинойс
schneier@counterpane.com

Об авторе

БРЮС ШНАЙЕР - президент Counterpane Systems, Оак Парк, Иллинойс, фирма-консультант, специализирующаяся в криптографии и компьютерной безопасности. Брюс также написал *E-Mail Security*, John Wiley & Sons, 1995, (*Безопасность электронной почты*) и *Protect Your Macintosh*, Peachpit Press, 1994, (*Защити свой Макинтош*). Он является автором дюжин статей по криптографии в основных журналах. Он также соредактор *Dr. Dobb's Journal* (*Журнал доктора Добба*), где он редактирует колонку "Аллея алгоритмов", и соредактор *Computer and Communications Security Reviews* (Обзор безопасности компьютеров и линий связи). Брюс входит в совет директоров Международной Ассоциации Криптологических Исследований (International Association for Cryptologic Research), является членом Консультационного совета Центра Секретности Электронной Информации (Electronic Privacy Information Center) и входит в комитет программы Семинара по Новым парадигмам Безопасности (New Security Paradigms Workshop). К тому же, он находит время для частых лекций по криптографии, компьютерной безопасности и секретности.

Глава 1
Основные понятия

1.1 Терминология

Отправитель и получатель

Предположим, что отправитель хочет послать сообщение получателю. Более того, этот отправитель хочет послать свое сообщение безопасно: он хочет быть уверен, что перехвативший это сообщение не сможет его пр очесть.

Сообщения и шифрование

Само сообщение называется **открытым текстом** (иногда используется термин клер). Изменение вида сообщения так, чтобы спрятать его суть называется **шифрованием**. Шифрованное сообщение называется **шифротекстом**. Процесс преобразования шифротекста в открытый текст называется **дешифрированием**. Эта последовательность показана на 0th.

(Если вы хотите следовать стандарту ISO 7498-2, то в английских текстах используйте термины "enchipher" вместо " encrypt" ("зашифровывать") и "dechipher" вместо " decrypt" ("дешифровывать")).

Искусство и наука безопасных сообщений, называемая **криптографией**, воплощается в жизнь **криптографами**. **Криптоаналитиками** называются те, кто постоянно используют **криптоанализ**, искусство и науку взламывать шифротекст, то есть, раскрывать, что находится под маской. Отрасль математики, охватывающая криптографию и криптоанализ, называется криптологией, а люди, которые ей занимаются, - **криптологами**. Современным криптологам приходится неплохо знать математику.

Рис. 1-1. Шифрование и дешифрирование

Обозначим открытый текст как *M* (от *message*, сообщение), или *P* (от *plaintext*, открытый текст). Это может быть поток битов, текстовый файл, битовое изображение, оцифрованный звук, цифровое видеоизображение… да что угодно. Для компьютера *M* - это просто двоичные данные. (Во всех следующих главах этой книги рассматриваются только двоичные данные и компьютерная криптография.) Открытый текст может быть создан для хранения или передачи. В любом случае, *M* - это сообщение, которое должно быть зашифровано.

Обозначим шифротекст как *C* (от *ciphertext*). Это тоже двоичные данные, иногда того же размера, что и *M*, иногда больше. (Если шифрование сопровождается сжатием, *C* может быть меньше чем *M*. Однако, само шифрование не обеспечивает сжатие информации.) Функция шифрования *E* действует на *M*, создавая *C*. Или, в математической записи:

$E(M) = C$

В обратном процессе функция дешифрирования *D* действует на *C*, восстанавливая *M:*

$D(C) = M$

Поскольку смыслом шифрования и последующего дешифрирования сообщения является восстановление первоначального открытого текста, должно выполняться следующее равенство:

$D(E(M)) = M$

Проверка подлинности, целостность и неотрицание авторства

Кроме обеспечения конфиденциальности криптография часто используется для других функций:

— **Проверка подлинности**. Получатель сообщения может проверить его источник, злоумышленник не сможет замаскироваться под кого-либо.

— **Целостность. Получатель сообщения может проверить, не было ли сообщение изменено в процессе доставки, злоумышленник не сможет подменить правильное сообщение ложным.**

— **Неотрицание авторства. Отправитель не сможет ложно отрицать отправку сообщения.**

Существуют жизненно важные требования к общению при помощи компьютеров, также как существуют ана-

логичные требования при общении лицом к лицу. То, что кто-то является именно тем, за кого он себя выдает … что чьи-то документы - водительские права, медицинская степень или паспорт - настоящие … что документ, полученный от кого-то, получен именно от этого человека… Как раз это обеспечивают проверка подлинности, целостность и неотрицание авторства.

Алгоритмы и ключи

Криптографический алгоритм, также называемый **шифром**, представляет собой математическую функцию, используемую для шифрования и дешифрирования. (Обычно это две связанных функции: одна для шифрования, а другая для дешифрирования.)

Если безопасность алгоритма основана на сохранении самого алгоритма в тайне, это **ограниченный** алгоритм. Ограниченные алгоритмы представляют только исторический интерес, но они совершенно не соответствуют сегодняшним стандартам. Большая или изменяющаяся группа пользователей не может использовать такие алгоритмы, так как всякий раз, когда пользователь покидает группу, ее члены должны переходить на другой алгоритм. Алгоритм должен быть заменен и, если кто-нибудь извне случайно узнает секрет.

Что еще хуже, ограниченные алгоритмы не допускают качественного контроля или стандартизации. У каждой группы пользователей должен быть свой уникальный алгоритм. Такие группы не могут использовать открытые аппаратные или программные продукты - злоумышленник может купить такой же продукт и раскрыть алгоритм. Им приходится разрабатывать и реализовывать собственные алгоритмы. Если в группе нет хорошего криптографа, то как ее члены проверят, что они пользуются безопасным алгоритмом?

Несмотря на эти основные недостатки ограниченные алгоритмы необычайно популярны для приложений с низким уровнем безопасности. Пользователи либо не понимают проблем, связанных с безопасностью своих систем, либо не заботятся о них.

Современная криптография решает эти проблемы с помощью **ключа** K. Такой ключ может быть любым значением, выбранным из большого множества. Множество возможных ключей называют **пространством ключей**. И шифрование, и дешифрирование этот ключ (то есть, они зависят от ключа, что обозначается индексом K), и теперь эти функции выглядят как:

$E_K(M)=C$

$D_K(C)=M$

При этом выполняется следующее равенство (см -1-й):

$D_K(E_K(M))=M$

Для некоторых алгоритмов при шифровании и дешифрировании используются различные ключи (см -2-й). То есть ключ шифрования, K_1, отличается от соответствующего ключа дешифрирования, K_2. В этом случае:

$$E_{K_1}(M)=C$$

$$D_{K_2}(C)=M$$

$$D_{K_2}(E_{K_1}(M))=M$$

Безопасность этих алгоритмов полностью основана на ключах, а не на деталях алгоритмов. Это значит, что алгоритм может быть опубликован и проанализирован. Продукты, использующие этот алгоритм, могут широко тиражироваться. Не имеет значения, что злоумышленнику известен ваш алгоритм, если ему не известен конкретный ключ, то он не сможет прочесть ваши сообщения.

Криптосистема представляет собой алгоритм плюс все возможные открытые тексты, шифротексты и ключи.

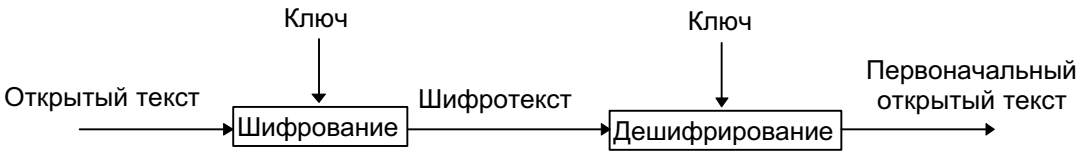


Рис. 1-2. Шифрование и дешифрирование с ключом

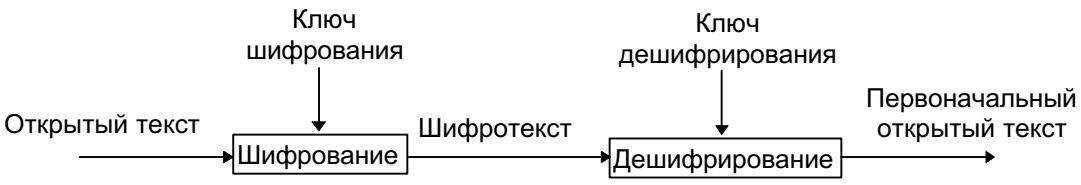


Рис. 1-3. Шифрование и дешифрирование с двумя различными ключами

Симметричные алгоритмы

Существует два основных типа алгоритмов, основанных на ключах: симметричные и с открытым ключом . **Симметричные алгоритмы**, иногда называемые условными алгоритмами , представляют собой алгоритмы, в которых ключ шифрования может быть рассчитан по ключу дешифрирования и наоборот . В большинстве симметричных алгоритмов кличи шифрования и дешифрирования одни и те же . Эти алгоритмы, также называемые алгоритмами с секретным ключом или алгоритмами с одним ключом, требуют, чтобы отправитель и получатель согласовали используемый ключ перед началом безопасной передачи сообщений . Безопасность симметричного алгоритма определяется ключом, раскрытие ключа означает, что кто угодно сможет шифровать и дешифрир овать сообщения. Пока передаваемые сообщения должны быть тайными, ключ должен храниться в секрете . Шифрование и дешифрирование с использованием симметричного алгоритма обозначается как :

$E_K(M)=C$

$D_K(C)=M$

Симметричные алгоритмы делятся на две категории . Одни алгоритмы обрабатывают открытый текст побитно (иногда побайтно), они называются **потоковыми алгоритмами** или **потоковыми шифрами**. Другие работаю с группами битов открытого текста. Группы битов называются блоками, а алгоритмы - **блочными алгоритмами** или **блочными шифрами**. Для алгоритмов, используемых в компьютерных модемах, типичный размер блока составляет 64 бита - достаточно большое значение, чтобы помешать анализу, и достаточно небольшое и удобное для работы. (До появления компьютеров алгоритмы обычно обрабатывали открытый текст посимвольно . Такой вариант может рассматриваться как потоковый алгоритм, обрабатывающий поток символов .)

Алгоритмы с открытым ключом

Алгоритмы с открытым ключом (называемые асимметричными алгоритмами) разработаны таким образом, что ключ, используемый для шифрования, отличается от ключа дешифрирования . Более того, ключ дешифрирования не может быть (по крайней мере в течение разумного интервала времени) рассчитан по ключу шифрования. Алгоритмы называются "с открытым ключом", потому что ключ шифрования может быть открытым: кто угодно может использовать ключ шифрования для шифрования сообщения, но только конкретный ч еловек с соответствующим ключом дешифрирования может расшифровать сообщение. В этих системах ключ шифрования часто называется **открытым** ключом, а ключ дешифрирования - **закрытым**. Закрытый ключ иногда называется секретным ключом, но чтобы не было путаницы с симметричными алгоритмами, этот термин не используется в данной книге. Шифрование с открытым ключом *K* обозначается как:

$E_K(M)=C$

Хотя открытый и закрытый ключи различны, дешифрирование с соответствующим закрытым ключом об означается как:

$D_K(C)=M$

Иногда сообщения шифруются закрытым ключом, а дешифрируются открытым, что используется для ци фровой подписи (см. раздел 2.6). Несмотря на возможную путаницу эти операции, соответственно, обозначаю тся как:

$E_K(M)=C$

$D_K(C)=M$

Криптоанализ

Смысл криптографии - в сохранении открытого текста (или ключа, или и того, и другого) в тайне от зл оумышленников (также называемых взломщиками , соперниками, врагами, перехватчиками). Предполагается, что злоумышленники полностью контролируют линии связи между отправителем и получателем .

Криптоанализ - это наука получения открытого текста, не имея ключа . Успешно проведенный криптоанализ может раскрыть открытый текст или ключ . Он также может обнаружить слабые места в криптосистемах, что в конце концов приведет к предыдущему результату . (Раскрытие ключа не криптологическими способами наз ы-

вается **компрометацией**.)

Попытка криптоанализа называется **вскрытием**. Основное предположение криптоанализа, впервые сформулированное в девятнадцатом веке Датчманом А. Керкхофсом (Dutchman A. Kerckhoffs), icостоит в том, что безопасность полностью определяется ключом [794]. Керкхофс предполагает, что у криптоаналитика есть полное описание алгоритма и его реализации. (Конечно же, у ЦРУ не в обычае сообщать Моссад о своих криптографических алгоритмах, но Моссад возможно все равно добудет их .) Хотя в реальном мире криптоаналитики не всегда обладают подробной информацией, такое предположение является хорошей рабочей гипотезой . Если противник не сможет взломать алгоритм, даже зная, как он работает, то тем более враг не сможет вскрыть а лгоритм без этого знания.

Существует четыре основных типа криптоаналитического вскрытия . Для каждого из них, конечно, предполагается, что криптоаналитик обладает всей полнотой знания об используемом алгоритме шифрования :

1. **Вскрытие с использованием только шифротекста** У криптоаналитика есть шифротексты нескольких сообщений, зашифрованных одним и тем же алгоритмом шифрования . Задача криптоаналитика состоит в раскрытии открытого текста как можно большего числа сообщений или, что лучше, получении ключа (ключей), использованного для шифрования сообщений, для дешифрировании других сообщений, зашифрованных теми же ключами.

   Дано: $C_1=E_k(P_1)$, $C_2=E_k(P_2)$, . . . $C_i=E_k(P_i)$

   Получить: Либо P_1, P_2, . . . P_i; k; либо алгоритм, как получать P_{i+1} из $C_{i+1}=E_k(P_{i+1})$

2. **Вскрытие с использованием открытого текста** У криптоаналитика есть доступ не только к шифротекстам нескольких сообщений, но и к открытому тексту этих сообщений . Его задача состоит в получении ключа (или ключей), использованного для шифрования сообщений, для дешифрировании других сообщений, зашифрованных тем же ключом (ключами) .

   Дано: P_1, $C_1=E_k(P_1)$, P_2, $C_2=E_k(P_2)$, . . . P_i, $C_i=E_k(P_i)$

   Получить: Либо k; либо алгоритм, как получать P_{i+1} из $C_{i+1}=E_k(P_{i+1})$

3. **Вскрытие с использованием выбранного открытого текста** У криптоаналитика не только есть доступ к шифротекстам и открытым текстам нескольких сообщений, но и возможность выбирать открытый текст для шифрования. Это предоставляет больше вариантов чем вскрытие с использованием открытого текста, так как криптоаналитик может выбирать шифруемые блоки открытого текста, что может дать больше информации о ключе. Его задача состоит в получении ключа (или ключей), использованного для шифрования сообщений, или алгоритма, позволяющего дешифрировать новые сообщения, зашифрованные тем же ключом (или ключами) .

   Дано: P_1, $C_1=E_k(P_1)$, P_2, $C_2=E_k(P_2)$, . . . P_i, $C_i=E_k(P_i)$
   где криптоаналитик может выбирать P_1, P_2, . . . P_i

   Получить: Либо k; либо алгоритм, как получать P_{i+1} из $C_{i+1}=E_k(P_{i+1})$

4. **Адаптивное вскрытие с использованием открытого текста** Это частный случай вскрытия с использованием выбранного открытого текста. Криптоаналитик не только может выбирать шифруемый текст, но также может строить свой последующий выбор на базе полученных результатов шифрования. При вскрытии с использованием выбранного открытого текста криптоаналитик мог выбрать для шифрования только один большой блок открытого текста, при адаптивном вскрытии с использованием выбранного открытого текста он может выбрать меньший блок открытого текста, затем выбрать следующий блок, используя результаты первого выбора и так далее .

Существует по крайней мере еше три типа криптоаналитической вскрытия .

5. **Вскрытие с использованием выбранного шифротекста** Криптоаналитик может выбрать различные шифротексты для дешифрирования и имеет доступ к дешифрированным открытым текстам . Например, у криптоаналитика есть доступ к "черному ящику", который выполняет автоматическое дешифрирование. Его задача состоит в получении ключа.

   Дано: C_1, $P_1=D_k(C_1)$, C_2, $P_2=D_k(C_2)$, . . . C_i, $P_i=D_k(C_i)$

   Получить: k

   Такой тип вскрытия обычно применим к алгоритмам с открытым ключом и обсуждается в разделе 19.3. Вскрытие с использование выбранного шифротекста иногда также эффективно против симметричных алгоритмов. (Иногда вскрытие с использованием выбранного открытого текста и вскрытие с использованием выбранного шифротекста вместе называют вскрытием с использованием выбранного текста.)

6. **Вскрытие с использованием выбранного ключа**. Такой тип вскрытия означает не то, что криптоаналитик может выбирать ключ, а что у него есть некоторая информация о связи между различными ключами. Этот странный, запутанный и не очень практичный тип вскрытия обсуждается в разделе 12.4.

7. **Бандитский криптоанализ**. Криптоаналитик угрожает, шантажирует или пытает кого-нибудь, пока не получит ключ. Взяточничество иногда называется **вскрытием с покупкой ключа**. Это очень мощные способы вскрытия, часто являющиеся наилучшим путем взломать алгоритм.

Вскрытия с известным открытым текстом и с использованием выбранного открытого текста встречаются чаще, чем можно подумать. Не является невозможным для криптоаналитика добыть открытый текст шифрованного сообщения или подкупить кого-нибудь, кто зашифрует выбранное сообщение. Может и не потребоваться никого подкупать - передав письмо послу, вы, возможно, обнаружите, что письмо будет зашифровано и отправлено в его страну для изучения. Многие сообщения имеют стандартные начало и окончание, что может быть известно криптоаналитику. Особенно уязвим шифрованный исходный код из-за частого использования ключевых слов: #define, struct, else, return. Те же проблемы и у шифрованного исполнимого кода: функции, циклические структуры и так далее. Вскрытия с известным открытым текстом (и вскрытия с выбранным шифротекстом) успешно использовались в борьбе с немцами и японцами в ходе Второй мировой войны. Исторические примеры вскрытий такого типа можно найти в книгах Дэвида Кана [794,795,796].

И не забывайте о предположении Керкхофса: если мощь вашей новой криптосистемы опирается на то, что взломщик не знает, как работает алгоритм, вы пропали. Если вы считаете, что хранение принципа работы алгоритма в секрете лучше защитит вашу криптосистему, чем предложение академическому сообществу проанализировать алгоритм, вы ошибаетесь. А если вы думаете, что кто-то не сможет дезассемблировать ваш исходный код и восстановить ваш алгоритм, вы наивны. (В 1994 году такое произошло с алгоритмом RC4, см. раздел 17.1.) Нашими лучшими алгоритмами являются те, которые были разработаны открыто, годами взламывались лучшими криптографами мира и все еще несокрушимы. (Агентство Национальной Безопасности хранит свои алгоритмы в секрете, но у них работают лучшие криптографы мира, а у вас - нет. Кроме того, они обсуждают свои алгоритмы друг с другом, полагаясь на способность товарища обнаружить все слабости в своей работе.)

У криптоаналитиков не всегда есть доступ к алгоритмам (например, вскрытие в ходе Второй мировой войны Соединенными Штатами японского дипломатического кода PURPLE [794]), но часто они его получают. Если алгоритм используется в коммерческой программе безопасности, то это просто вопрос времени и денег, удастся ли дезассемблировать программу и раскрыть алгоритм. Если же алгоритм используется в военной системе связи, то это просто вопрос времени и денег купить (или украсть) аппаратуру и реконструировать алгоритм.

Те, кто стремится получить нераскрываемый шифр, считая этот шифр таковым только потому, что они сами не смогли его взломать, либо гении, либо дураки. К несчастью, последних в мире достаточно много. Остерегайтесь людей, расхваливающих надежность своих алгоритмов, но отказывающихся их опубликовать. Доверять таким алгоритмам нельзя.

Хорошие криптографы опираются на мнение других, отделяя хорошие алгоритмы от плохих.

Безопасность алгоритмов

Различные алгоритмы предоставляют различные степени безопасности в зависимости от того, насколько трудно взломать алгоритм. Если стоимость взлома алгоритма выше, чем стоимость зашифрованных данных, вы, скорее всего, в безопасности. Если время взлома алгоритма больше, чем время, в течение которого зашифрованные данные должны сохраняться в секрете, то вы также, скорее всего, в безопасности. Если объем данных, зашифрованных одним ключом, меньше, чем объем данных, необходимый для взлома алгоритма, и тогда вы, скорее всего, в безопасности.

Я говорю "скорее всего", потому что существует вероятность новых прорывов в криптоанализе. С другой стороны, значимость большинства данных падает со временем. Важно, чтобы значимость данных всегда оставалась меньше, чем стоимость взлома системы безопасности, защищающей данные.

Ларс Кнудсен (Lars Knudsen) разбил вскрытия алгоритмов по следующим категориям, приведенным в порядке убывания значимости [858]:

1. **Полное вскрытие**. Криптоаналитик получил ключ, K, такой, что $D_K(C) = P$.

2. **Глобальная дедукция**. Криптоаналитик получил альтернативный алгоритм, A, эквивалентный $D_K(C)$ без знания K.

3. **Местная (или локальная) дедукция**. Криптоаналитик получил открытый текст для перехваченного шифротекста.

4. **Информационная дедукция**. Криптоаналитик получил некоторую информацию о ключе или открытом тексте. Такой информацией могут быть несколько бит ключа, сведения о форме открытого текста и так далее.

Алгоритм является **безусловно безопасным**, если, независимо от объема шифротекстов у криптоаналитика, информации для получения открытого текста недостаточно. По сути, только шифрование одноразовыми блокнотами (см. раздел 1.5) невозможно вскрыть при бесконечных ресурсах. Все остальные криптосистемы подвержены вскрытию с использованием только шифротекста простым перебором возможных ключей и проверкой осмысленности полученного открытого текста. Это называется вскрытием **грубой силой** (см. раздел 7.1).

Криптография больше интересуется криптосистемами, которые тяжело взломать вычислительным способом. Алгоритм считается **вычислительно безопасным** (или, как иногда называют, сильным), если он не может быть взломан с использованием доступных ресурсов сейчас или в будущем. Термин "доступные ресурсы" является достаточно расплывчатым. Сложность вскрытия можно измерить (см раздел 11.1) различными способами:

1. **Сложность данных**. Объем данных, используемых на входе операции вскрытия.
2. **Сложность обработки**. Время, нужное для проведения вскрытия. Часто называется **коэффициентом работы**.
3. **Требования к памяти**. Объем памяти, необходимый для вскрытия.

В качестве эмпирического метода сложность вскрытия определяется по максимальному из этих трех коэффициентов. Ряд операций вскрытия предполагают взаимосвязь коэффициентов: более быстрое вскрытие возможно за счет увеличения требований к памяти.

Сложность выражается порядком величины. Если сложность обработки для данного алгоритма составляет 2^{128}, то 2^{128} операций требуется для вскрытия алгоритма. (Эти операции могут быть сложными и длительными.) Так, если предполагается, что ваши вычислительные мощности способны выполнять миллион операций в секунду, и вы используете для решения задачи миллион параллельных процессоров, получение ключа займет у вас свыше 10^{19} лет, что в миллиард раз превышает время существования вселенной.

В то время, как сложность вскрытия остается постоянной (пока какой-нибудь криптоаналитик не придумает лучшего способа вскрытия), мощь компьютеров растет. За последние полвека вычислительные мощности феноменально выросли, и нет никаких причин подозревать, что эта тенденция не будет продолжена. Многие криптографические взломы пригодны для параллельных компьютеров: задача разбивается на миллиарды маленьких кусочков, решение которых не требует межпроцессорного взаимодействия. Объявление алгоритма безопасным просто потому, что его нелегко взломать, используя современную технику, в лучшем случае ненадежно. Хорошие криптосистемы проектируются устойчивыми к взлому с учетом развития вычислительных средств на много лет вперед.

Исторические термины

Исторически термин код относится к криптосистеме, связанной с лингвистическими единицами: словами, фразами, предложениями и так далее. Например, слово "ОЦЕЛОТ" может кодировать целую фразу "ПОВОРОТ НАЛЕВО НА 90 ГРАДУСОВ", слово "ЛЕДЕНЕЦ" - фразу "ПОВОРОТ НАПРАВО НА 90 ГРАДУСОВ", а слова "ПОДСТАВЬ УХО" могут кодировать слово "ГАУБИЦА". Коды такого типа не рассматриваются в данной книге, см. [794,795]. Коды полезны только при определенных обстоятельствах. Если у вас нет кода для "МУРАВЬЕДЫ", вы не сможете передать это понятие. А используя шифр можно сказать все.

1.2 Стеганография

Стеганография служит для передачи секретов в других сообщениях, так что спрятано само существование секрета. Как правило отправитель пишет какое-нибудь неприметное сообщение, а затем прячет секретное сообщение на том же листе бумаги. Исторические приемы включают невидимые чернила, невидимые простому глазу пометки у букв, плохо заметные отличия в написании букв, пометки карандашом машинописных символов, решетки, покрывающие большую часть сообщения кроме нескольких символов и тому подобное.

Ближе к сегодняшнему дню люди начали прятать секреты в графических изображениях, заменяя младший значащий бит изображения битом сообщения. Графическое изображение при этом менялось совсем незаметно - большинство графических стандартов определяют больше цветовых градаций, чем способен различить человеческий глаз - и сообщение извлекалось на противоположном конце. Так в черно-белой картинке 1024x1024 пиксела можно спрятать можно спрятать сообщение в 64 Кбайт. Многие общедоступные программы могут проделывать подобный фокус.

Имитационные функции Питера Уэйнера (Peter Wayner) маскируют сообщения. Эти функции изменяют сообщение так, что его статистический профиль становится похожим на что-нибудь еще: раздел *The New York*

Times, а пьесу Шекспира или телеконференцию в Internet [1584,1585]. Этот тип стеганографии не одурачит человека, но может обмануть большой компьютер, ищущий нужную информацию в Internet.

1.3 Подстановочные и перестановочные шифры

До появления компьютеров криптография состояла из алгоритмов на символьной основе . Различные криптографические алгоритмы либо заменяли одни символы другими, либо переставляли символы. Лучшие алгоритмы делали и то, и другое, и по много раз.

Сегодня все значительно сложнее, но философия остается прежней. Первое изменение заключается в том, что алгоритмы стали работать с битами, а не символами. Это важно хотя бы с точки зрения размера алфавита - с 26 элементов до двух. Большинство хороших криптографических алгоритмов до сих пор комбинирует подстановки и перестановки.

Подстановочные шифры

Подстановочным шифром называется шифр, который каждый символ открытого текста в шифротексте заменяет другим символом. Получатель инвертирует подстановку шифротекста, восстанавливая открытый текст . В классической криптографии существует четыре типа подстановочных шифров :

— **Простой подстановочный шифр**, или **моноалфавитный шифр**, - это шифр, который каждый символ открытого текста заменяет соответствующим символом шифротекста . Простыми подстановочными шифрами являются криптограммы в газетах.

— **Однозвучный подстановочный шифр** похож на простую подстановочную криптосистему за исключением того, что один символ открытого текста отображается на несколько символов шифротекста . Например, "A" может соответствовать 5, 13, 25 или 56, "B" - 7, 19, 31 или 42 и так далее.

— **Полиграмный подстановочный шифр** - это шифр, который блоки символов шифрует по группам . Например, "ABA" может соответствовать "RTQ", "ABB" может соответствовать "SLL" и так далее.

— **Полиалфавитный подстановочный шифр** состоит из нескольких простых подстановочных шифров . Например, могут быть использованы пять различных простых подстановочных фильтров ; каждый символ открытого текста заменяется с использованием одного конкретного шифра .

Знаменитый **шифр Цезаря**, в котором каждый символ открытого текста заменяется символом, находящегося тремя символами правее по модулю 26 ("A" заменяется на "D," "B" - на "E", ... "W" - на " Z ", "X" - на "A", "Y" - на "B", "Z" - на "C"), представляет собой простой подстановочный фильтр . Он действительно очень прост, так как алфавит шифротекста представляет собой смещенный, а не случайно распределенный алфавит открытого текста.

ROTI3 - это простая шифровальная программа, обычно поставляемая с системами UNIX. Она также является простым подстановочным шифром. В этом шифре "A" заменяется на "N," "B" - на "O" и так далее. Каждая буква смещается на 13 мест. Шифрование файла программой ROTI3 дважды восстанавливает первоначальный файл.

P = ROT13 (ROT13 (P))

ROTI3 не используется для безопасности, она часто применяется в почте, закрывая потенциально неприятный текст, решение головоломки и тому подобное .

Простые подстановочные шифры легко раскрываются, так как шифр не прячет частоты использования различных символов в открытом тексте. Чтобы восстановить открытый текст, хорошему криптоаналитику требуется только знать 26 символов английского алфавита [1434]. Алгоритм вскрытия таких шифров можно найти в [578, 587, 1600, 78, 1475, 1236, 880]. Хороший компьютерный алгоритм приведен в [703].

Однозвучные подстановочные шифры использовались уже в 1401 году в герцогстве Мантуа [794]. Они более сложны для вскрытия, чем простые подстановочные шифры, хотя и они не скрывают всех статистических свойств языка открытого текста. При помощи вскрытия с известным открытым текстом эти шифры раскрываются тривиально. Вскрытие с использованием только шифротекста более трудоемко, но и оно занимает на компьютере лишь несколько секунд. Подробности приведены в [1261].

Полиграмные подстановочные шифры - это шифры, которые кодируют сразу группы символов . Шифр Playfair ("Честная игра"), изобретенный в 1854 году, использовался англичанами в Первой мировой войне [794]. Он шифрует пары символов, и его криптоанализ обсуждается в [587,1475,880]. Другим примером полиграмного подстановочного шифра является шифр Хилла (Hill) [732]. Иногда можно видеть как вместо шифра используется кодирование по Хаффману (Huffman), это небезопасный полиграмный подстановочный шифр .

Полиалфавитные подстановочные шифры были изобретены Лином Баттистой (Lean Battista) в 1568 году

[794]. Они использовались армией Соединенных Штатов в ходе Гражданской войны в Америке . Несмотря на то, что они легко могут быть взломаны [819, 577, 587, 794] (особенно с помощью компьютеров), многие коммерческие продукты компьютерной безопасности используют такие шифры [1387,1390, 1502]. (Подробности того, как вскрыть эту схему шифрования, используемую программой WordPerfect, можно найти в [135,139].) Шифр Вигенера (Vigenere), впервые опубликованный в 1586 году, и шифр Бофора (Beaufort) также являются примерами полиалфавитных подстановочных шифров.

У полиалфавитных подстановочных шифров множественные однобуквенные ключи, каждый из которых используется для шифрования одного символа открытого текста . Первым ключом шифруется первый символ открытого текста, вторым ключом - второй символ, и так далее . После использования всех ключей они повторяются циклически. Если применяется 20 однобуквенных ключей, то каждая двадцатая буква шифруется тем же ключом. Этот параметр называется **периодом** шифра. В классической криптографии шифры с длинным периодом было труднее раскрыть, чем шифры с коротким периодом. Использование компьютеров позволяет легко раскрыть подстановочные шифры с очень длинным периодом .

Шифр с бегущим ключом (иногда называемый книжным шифром), использующий один текст для шифрования другого текста, представляет собой другой пример подобного шифра . И хотя период этого шифра равен длине текста, он также может быть легко взломан [576,794].

Перестановочные шифры

В **перестановочном шифре** меняется не открытый текст, а порядок символов. В **простом столбцовом перестановочном шифре** открытый текст пишется горизонтально на разграфленном листе бумаги фиксированной ширины, а шифротекст считывается по вертикали (см. -3-й). Дешифрирование представляет собой запись шифротекста вертикально на листе разграфленной бумаги фиксированной ширины и затем считывание открытого текста горизонтально.

Криптоанализ этих шифров обсуждается в [587,1475]. Так как символы шифротекста те же, что и в открытом тексте, частотный анализ шифротекста покажет, что каждая буква встречается приблизительно с той же частотой, что и обычно. Это даст криптоаналитику возможность применить различные методы, определяя правильный порядок символов для получения открытого текста . Применение к шифротексту второго перестановочного фильтра значительно повысит безопасность . Существуют и еще более сложные перестановочные фильтры, но компьютеры могут раскрыть почти все из них .

Немецкий шифр ADFCVX, использованный в ходе Первой мировой войны , представлял собой перестановочный фильтр в сочетании с простой подстановкой . Этот для своего времени очень сложный алгоритм был раскрыт Жоржем Пенвэном (Georges Painvin), французским криптоаналитиком [794].

Хотя многие современные алгоритмы используют перестановку, с этим связана проблема использования большого объема памяти, а также иногда требуется работа с сообщениями определенного размера . Подстановка более обычна.

Роторные машины

В 1920-х годах для автоматизации процесса шифрования были изобретены различные механические устройства. Большинство использовало понятие **ротора**, механического колеса, используемого для выполнения подстановки.

Роторная машина, включающая клавиатуру и набор роторов , реализует вариант шифра Вигенера. Каждый ротор представляет собой произвольное размещение алфавита, имеет 26 позиций и выполняет простую подстановку. Например, ротор может быть использован для замены "A" на " F", "B" на "U", "C" на "I" и так далее. Выходные штыри одного ротора соединены с входными штырями следующего ротора.

Открытый текст:COMPUTER GRAPHICS MAY BE SLOW BUT AT LEAST IT'S EXPENSIVE.

```
COMPUTERGR
APHICSMAYB
ESLOWBUTAT
LEASTITSEX
PENSIVE
```

Шифротекст:CAELP OPSEE MHLAN PIOSS UCWTI TCBIV EMUTE RATSG YAERB TX

Рис. 1-4. Столбцовый перестановочный фильтр.

Например, в четырехроторной машине первый ротор может заменять "A" на " F", второй - "F" на "Y", третий - "Y" на "E" и четвертый - "E" на "C", "C" и будет конечным шифротекстом. Затем некоторые роторы смещаются, и в следующий раз подстановки будут другими .

Именно комбинация нескольких роторов и механизмов, движущих роторами, и обеспечивает безопасность машины. Так как роторы вращаются с различной скоростью, период для n-роторной машины равен 26^n. Некоторые роторные машины также могут иметь различные положения для каждого ротора, что делает криптоанализ еще более бессмысленным.

Самым известным роторным устройство является Энигма (Enigma). Энигма использовалась немцами во Второй мировой войне. Сама идея пришла в голову Артуру Шербиусу (Arthur Scherbius) и Арвиду Герхарду Дамму (Arvid Gerhard Damm) в Европе. В Соединенных Штатах она была запатентована Артуром Шербиусом [1383]. Немцы значительно усовершенствовали базовый проект для использования во время войны.

У немецкой Энигмы было три ротора, котроые можно было выбрать из пяти возможных, коммутатор, который слегка тасовал открытый текст, и отражающий ротор, который заставлял каждый ротор обрабатывать открытый текст каждого письма дважды. Несмотря на сложность Энигмы, она была взломана в течение Второй мировой войны. Сначала группа польских криптографов взломала немецкую Энигму и объяснила раскрытый алгоритм англичанам. В ходе войны немцы модифицировали Энигму, а англичане продолжали криптоанализ новых версий. Объяснение работы роторных шифров и способов их раскрытия можно найти в [794, 86, 448, 498, 446, 880, 1315, 1587, 690]. В двух следующих отчетах увлекательно рассказывается о взломе Энигмы [735, 796].

Для дальнейшего чтения

Данная книга не является книгой по классической криптографии, поэтому далее я не буду подробно останавливаться на этих предметах. Прекрасными книгами по докомпьютерной криптологии являются [587, 1475]. [448] содержит современный криптоанализ шифровальных машин. Дороти Деннинг (Dorothy Denning) рассматривает многие из этих шифров в [456], а [880] содержит беспристрастный сложный математический анализ тех же самых шифров. Другим описанием старой криптографии, описывающим аналоговую криптографию, является [99]. Прекрасный обзор выполнен в статье [579]. Великолепны также книги по исторической криптографии Дэвида Кана [794, 795, 796].

1.4 Простое XOR

XOR представляет собой операцию "исключающее или": '^' в языке C или Q в математической нотации. Это обычная операция над битами:

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Также заметим, что:

$$a \oplus a = 0$$

$$a \oplus b \oplus b = a$$

Казалось бы, запутанный алгоритм простого XOR по сути является ничем иным, как полиалфавитным шифром Вигенера. Здесь он упоминается только из-за распространенности в коммерческих программных продуктах, по крайней мере в мире MS-DOS и Macintosh [1502, 1387]. К сожалению, если о программе компьютерной безопасности заявляется, что это "патентованный" алгоритм шифрования, значительно более быстрый, чем DES, то скорее всего используется какой-то вариант следующего.

```
/* Использование: crypto key input_file output_file */
void main (int argc, char *argv[])
{
      FILE *fl, *fo;
      char *cp;
      int c;

      if ((cp = argv[l]) && *cp!= '\0') {
            if ((fi = fopen(argvl[2], "rb")) != NULL) {
                  if ((fo = fopen(argv[3], "wb")) != NULL) {
                        while ((c = getc(fi)) != EOF) {
                              if (!*cp) cp = argv[1];
                              c^= *(cp++);
                              putc(c,fo);
                        }
                        fclose(fo);
                  }
                  fclose(fi);
            }
```

```
            }
        }
```

Это симметричный алгоритм. Открытый текст подвергается операции "исключающее или" вместе с ключевым текстом для получения шифротекста. Так как повторное применение операции XOR восстанавливает оригинал для шифрования и дешифрирования используется одна и та же программа :

$$P \oplus K = C$$

$$C \oplus K = P$$

Настоящей безопасности здесь никогда не было. Этот тип шифрования легко вскрывается, даже без компьютера [587, 1475]. Его взлом на компьютере занимает несколько секунд.

Предположим, что открытый текст использует английский язык. Более того, пусть длина ключа любое небольшое число байт. Ниже описано, как взломать этот шифр:

1. Определим длину ключа с помощью процедуры, известной как **подсчет совпадений** [577]. Применим операцию XOR к шифротексту, используя в качестве ключа сам шифротекст с различными смещениями, и подсчитаем совпадающие байты. Если величина смещения кратна длине ключа, то совпадет свыше 6 процентов байтов. Если нет, то будут совпадать меньше чем 0.4 процента (считая, что обычный ASCII текст кодируется случайным ключом, для других типов открытых текстов числа будут другими). Это называется **показателем совпадений**. Минимальное смещение от одного значения, кратного длине ключа, к другому и есть длина ключа.
2. Сместим шифротекст на эту длину и проведем операцию XOR для смещенного и оригинального шифротекстов. Результатом операции будет удаления ключа и получение открытого текста, подвергнутого операции XOR с самим собой, смещенным на длину ключа. Так как в английском языке на один байт приходится 1.3 бита действительной информации (см раздел 11.1), существующая значительная избыточность позволяет определить способ шифрования.

Несмотря на это, количество поставщиков программного обеспечения, навязывающих этот игрушечный алгоритм в качестве "почти такого же безопасного как DES", впечатляет [1387]. Именно этот алгоритм (с 160-битным повторяющимся "ключом") NSA в конце концов разрешило использовать в цифровых телефонных сотовых сетях для закрытия голоса. XOR может защитить ваши файлы от младшей сестры, но настоящего криптоаналитика задержит лишь на считанные секунды.

1.5 Одноразовые блокноты

Поверите или нет, но идеальный способ шифрования существует. Он называется **одноразовым блокнотом** и был изобретен в 1917 году Мэйджором Джозефом Моборном (Major Joseph Mauborgne) и Гилбертом Вернамом (Gilbert Vernam) из AT&T [794]. (Фактически одноразовый блокнот представляет собой особый случай пороговой схемы, см. раздел 3.7.) В классическом понимании одноразовый блокнот является большой неповторяющейся последовательностью символов ключа, распределенных случайным образом, написанных на кусочках бумаги и приклеенных к листу блокнота. Первоначально это была одноразовая лента для телетайпов. Отправитель использовал каждый символ ключа блокнота для шифрования только одного символа открытого текста. Шифрование представляет собой сложение по модулю 26 символа открытого текста и символа ключа из одноразового блокнота.

Каждый символ ключа используется только единожды и для единственного сообщения . Отправитель шифрует сообщения и уничтожает использованные страницы блокнота или использованную часть ленты . Получатель, в свою очередь, используя точно такой же блокнот, дешифрирует каждый символ шифротекста . Расшифровав сообщение, получатель уничтожает соответствующие страницы блокнота или часть ленты . Новое сообщение - новые символы ключа. Например, если сообщением является:

ONETIMEPAD

а ключевая последовательность в блокноте:

TBFRGFARFM

то шифротекст будет выглядеть как:

IPKLPSFHGQ

так как

Q + T mod 26 = I

N + B mod 26 = P

E + F mod 26 = K

и т.д.

В предположении, что злоумышленник не сможет получить доступ к одноразовому блокноту, использова нному для шифрования сообщения, эта схема совершенно безопасна. Данное шифрованное сообщение на вид соответствует любому открытому сообщению того же размера.

Так как все ключевые последовательности совершенно одинаковы (помните, символы ключа генерируются случайным образом), у противника отсутствует информация, позволяющая подвергнуть шифротекст криптоан ализу. Кусочек шифротекста может быть похож на:

POYYAEAAZX

что дешифрируется как:

SALMONEGGS

или на:

BXEGBMTMXM

что дешифрируется как:

GREENFLUID

Повторю еще раз: так как все открытые тексты равновероятны, у криптоаналитика нет возможности опред елить, какой из открытых текстов является правильным. Случайная ключевая последовательность, сложенная с неслучайным открытым текстом, дает совершенно случайный шифротекст, и никакие вычислительные мощн ости не смогут это изменить.

Необходимо напомнить, что символы ключа должны генерироваться случайным образом. Любые попытки вскрыть такую схему сталкиваются со способом, которым создается последовательность символов ключа. Использование генераторов псевдослучайных чисел не считается, у них всегда неслучайные свойства. Если вы используете действительно случайный источник - это намного труднее, чем кажется на первый взгляд, см. ра здел 17.14 - это совершенно безопасно.

Другой важный момент: ключевую последовательность никогда нельзя использовать второй раз. Даже если вы используете блокнот размером в несколько гигабайт, то если криптоаналитик получит несколько текстов с перекрывающимися ключами, он сможет восстановить открытый текст. Он сдвинет каждую пару шифротекстов относительно друг друга и подсчитает число совпадений в каждой позиции. Если шифротексты смещены пр авильно, соотношение совпадений резко возрастет - точное значение зависит от языка открытого текста. С этой точки зрения криптоанализ не представляет труда. Это похоже на показатель совпадений, но сравниваются два различных "периода" [904]. Не используйте ключевую последовательность повторно.

Идея одноразового блокнота легко расширяется на двоичные данные. Вместо одноразового блокнота, с остоящего из букв, используется одноразовый блокнот из битов. Вместо сложения открытого текста с ключом одноразового блокнота используйте XOR. Для дешифрирования примените XOR к шифротексту с тем же одноразовым блокнотом. Все остальное не меняется, и безопасность остается такой же совершенной.

Все это хорошо, но существует несколько проблем. Так как ключевые биты должны быть случайными и не могут использоваться снова, длина ключевой последовательности должна равняться длине сообщения. Одноразовый блокнот удобен для нескольких небольших сообщений, но его нельзя использовать для работы по каналу связи с пропускной способностью 1.544 Мбит/с. Вы можете хранить 650 Мбайт случайных данных на CD-ROM, но и тут есть проблемы. Во первых, вам нужно только две копии случайных битов, но CD-ROM экономичны только при больших тиражах. И во вторых, вам нужно уничтожать использованные биты. Для CD-ROM нет другой возможности удалить информацию кроме как физически разрушить весь диск. Гораздо больше подходит цифровая лента.

Даже если проблемы распределения и хранения ключей решены, вам придется точно синхронизировать р аботу отправителя и получателя. Если получатель пропустит бит (или несколько бит пропадут при передаче), сообщение потеряет всякий смысл. С другой стороны, если несколько бит изменятся при передаче (и ни один бит не будет удален или добавлен - что гораздо больше похоже на влияние случайного шума), то лишь эти биты будут расшифрованы неправильно. Но одноразовый блокнот не обеспечивает проверку подлинности.

Одноразовые блокноты используются и сегодня, главным образом для сверхсекретных каналов связи с ни зкой пропускной способностью. По слухам "горячая линия" между Соединенными Штатами и бывшим Совет ским Союзом (а действует ли она сейчас?) шифруется с помощью одноразового блокнота. Многие сообщения советских шпионов зашифрованы с использованием одноразовых блокнотов. Эти сообщения нераскрыты сего дня и навсегда останутся нераскрытыми. На этот факт не повлияет время работы суперкомпьютеров над этой

проблемой. Даже когда враги из созвездия Андромеды приземлят свои тяжелые корабли с компьютерами н емыслимой мощности, и они не смогут прочесть сообщения советских шпионов, зашифрованные с помощью о дноразовых (если, конечно, они не смогут вернуться в прошлое и добыть нужные одноразовые блокноты).

1.6 Компьютерные алгоритмы

Существует множество компьютерных алгоритмов. Следующие три используются чаще всего :

— DES (Data Encryption Standard, стандарт шифрования данных) - самый популярный компьютерный алгоритм шифрования, является американским и международным стандартом . Это симметричный алгоритм, один и тот же ключ используется для шифрования и дешифрирования .

— RSA (назван в честь создателей - Ривеста (Rivest), Шамира (Sharnir) и Эдлмана (Adleman)) - самый популярный алгоритм с открытым ключом. Используется и для шифрования, и для цифровой подписи .

— DSA (Digital Signature Algorithm, алгоритм цифровой подписи, используется как часть стандарта цифровой подписи, Digital Signature Standard) - другой алгоритм с открытым ключом. Используется только для цифровой подписи, не может быть использован для шифрования .

Именно эти и подобные алгоритмы описываются в этой книге .

1.7 Большие числа

На протяжении всей книги я использую различные большие числа для описания различных вещей в крипт ографии. Так как легко заблудиться в этих числах и их значениях, физические аналоги некоторых чисел прив едены в 0-й.

Эти числа оцениваются по порядку величины и были отобраны из различных источников. Многие астроф изические значения объясняются в работе Фримана Дайсона (Freeman Dyson), "Время без конца: физика и биология в открытой Вселенной" ("Time Without End: Physics and Biology in an Open Universe") в *Reviews of Modem Physics,* v. 52, n. 3, July 1979, pp. 447-460. Смертность в результате автокатастроф рассчитана с помощью стат истики Министерства транспорта (163 смерти миллион человек в 1993 году и для средней продолжительности жизни 69.7 года.

Табл. 1-1. Большие числа

Физический аналог	Число
Вероятность быть убитым молнией (в течение дня)	1 из 9 миллиардов (2^{33})
Вероятность выиграть главный приз в государственной лотерее США	1 из 4000000 (2^{22})
Вероятность выиграть главный приз в государственной лотерее США и быть убитым молнией в течение того же дня	1 из2^{61}
Вероятность утонуть (в США в течение года)	1 из 59000 (2^{16})
Вероятность погибнуть в автокатастрофе (в США в году)	1 из 6100 (2^{13})
Вероятность погибнуть в автокатастрофе (в США в течение времени жизни)	1 из 88 (2^{7})
Время до следующего оледенения	14000 (2^{14}) лет
Время до превращения Солнца в сверхновую звезду	10^{9} (2^{30}) лет
Возраст планеты	10^{9} (2^{30}) лет
Возраст Вселенной	10^{10} (2^{34}) лет
Число атомов планеты	10^{51} (2^{170})
Число атомов Солнца	10^{57} (2^{190})
Число атомов галактики	10^{67} (2^{223})
Число атомов Вселенной	10^{77} (2^{265})
Объем Вселенной	10^{84} (2^{280}) см ^{3}

Если Вселенная конечна:

Полное время жизни вселенной	10^{11} (2^{37}) лет 10^{18} (2^{61}) секунд
Если Вселенная бесконечна:	
Время до остывания легких звезд	10^{14} (2^{47}) лет
Время до отрыва планет от звезд	10^{15} (2^{50}) лет
Время до отрыва звезд от галактик	10^{19} (2^{64}) лет
Время до разрушения орбит гравитационной радиацией	10^{20} (2^{67}) лет
Время до разрушения черных дыр процессами Хокинга	10^{64} (2^{213}) лет
Время до превращения материи в жидкость при нулевой температуре	10^{65} (2^{216}) лет
Время до превращения материи в твердое тело	$10^{10^{26}}$ лет
Время до превращения материи в черную дыру	$10^{10^{76}}$ лет

Часть 1

КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ

Глава 2
Элементы протоколов

2.1 Введение в протоколы

Смысл криптографии - в решении проблем. (По сути, в этом состоит и смысл использования компьютеров, о чем многие пытаются забыть.) Криптография решает проблемы секретности, проверки подлинности, целостности и человеческой нечестности. Вы можете выучить все о криптографических алгоритмах и методах, но они представляют только академический интерес, если не используются для решения какой-нибудь проблемы. Именно поэтому мы собираемся сначала взглянуть на протоколы.

Протокол - это порядок действий, предпринимаемых двумя или более сторонами, предназначенный для решения определенной задачи. Это важное определение. "Порядок действий" означает, протокол выполняется в определенной последовательности, с начала до конца. Каждое действие должно выполняться в свою очередь и только после окончания предыдущего. "Предпринимаемых двумя или более сторонами" означает, что для реализации протокола требуется по крайней мере два человека, один человек не сможет реализовать протокол. Человек в одиночку может выполнить некоторые действия, решая задачу (например, покупая торт), но это не протокол. (Для того, чтобы получился настоящий протокол, кто-то должен съесть торт.) Наконец, "предназначенный для решения определенной задачи" означает, что протокол должен приводить к какому-то результату. Что-то, похожее на протокол, но не решающее никакой задачи - это не протокол, это потеря времени. У протоколов есть также и другие характеристики:

— Каждый участник протокола должен знать протокол и последовательность составляющих его действий.

— Каждый участник протокола должен согласиться следовать протоколу.

— Протокол должен быть непротиворечивым, каждое действие должно быть определено так, чтобы не было возможности непонимания.

— Протокол должен быть полным, каждой возможной ситуации должно соответствовать определенное действие.

В этой книге каждый протокол организован как некоторый порядок действий. Выполнение протокола происходит по действиям, линейно, пока не будет команды перейти к следующему действию. Каждое действие включает по крайней мере одно из двух: вычисления, выполняемые одной или несколькими сторонами, или сообщения, которыми обмениваются стороны.

Криптографический протокол - это протокол, использующий криптографию. Стороны могут быть друзьями и слепо доверять друг другу или врагами и не верить друг другу даже при сообщении времени суток. Криптографический протокол включает некоторый криптографический алгоритм, но, вообще говоря, предназначение протокола выходит за рамки простой безопасности. Участники протокола могут захотеть поделиться секретом друг с другом, совместно генерировать случайную последовательность, подтвердить друг другу свою подлинность или подписать контракт в один и тот же момент времени. Смысл использования криптографии в протоколе - в предотвращении или обнаружении вредительства и мошенничества. Если вы никогда не сталкивались с подобными протоколами, они могут радикально изменить ваше представление о том, что недоверяющие друг другу стороны могут выполнить, используя компьютерную сеть. Общее правило можно сформулировать следующим образом:

— Невозможно сделать или узнать больше, чем определено в протоколе.

Это гораздо сложнее, чем кажется. В следующих нескольких главах я рассматриваю множество протоколов. В некоторых из них один из участников может обмануть другого. В других, злоумышленник может взломать протокол или узнать секретную информацию. Ряд протоколов проваливаются, так как их разработчики недостаточно тщательно определяли требования. Другие проваливаются из-за того, что их разработчики недостаточно тщательно анализировали свои протоколы. Как и для алгоритмов, гораздо легче доказать возможную небезопасность протокола, чем его полную безопасность.

Смысл протоколов

В повседневной жизни почти для всего существуют неформальные протоколы: заказ товаров по телефону, игра в покер, голосование на выборах. Никто не задумывается об этих протоколах, они вырабатывались в течение длительного времени, все знают, как ими пользоваться и они работают достаточно хорошо.

Сегодня все больше и больше людей общаются не лично, а используя компьютерную сеть. Для тех же вещей, которые люди делают не задумываясь, компьютерам нужны формальные протоколы. Когда вы переезжаете из государства в государство и обнаруживаете кабинку, совершенно отличающуюся от той, к которой вы при-

выкли, вы легко адаптируетесь. Компьютеры далеко не так гибки .

Честность и безопасность многих протоколов человеческого общения основаны на личном присутствии . Разве вы дадите незнакомцу кучу денег, чтобы он купил для вас что-нибудь в бакалее ? Сядете ли вы играть в покер с тем, кто жульничает, сдавая карты? Пошлете ли вы свой избирательный бюллетень правительству, не будучи уверенным в тайности такого голосования?

Наивно считать, что пользователи компьютерных сетей всегда честны. Также наивно считать, что всегда честны разработчики компьютерных сетей. Для большинства из них это именно так, но даже несколько жуликов могут принести много вреда. Формализируя протоколы, можно проверить способы, используемые жуликами для взлома протоколов. Так мы можем разработать протоколы, устойчивые к взлому.

Кроме формализации действий, протоколы позволяют абстрагироваться при решении задачи от способа решения. Протокол связи один и тот же и на PC, и на VAX. Можно проверить протокол, не вдаваясь в детали его реализации. Когда мы убедимся в надежности протокола, его можно будет реализовать где угодно от компьютеров до телефонов и интеллектуальных тостеров .

Игроки

Для демонстрации работы протоколов я использую несколько игроков (см. 1-й). Первые двое - это Алиса и Боб. Они участвуют во всех двусторонних протоколах . Как правило, Алиса (Alice) начинает все протоколы, а Боб (Bob) отвечает. Если для протокола нужна третья или четвертая сторона, в игру вступают Кэрол (Êýðîë) и Дэйв (Dave). Другие игроки играют специальные вспомогательные роли, они будут представлены по зже.

Протоколы с посредником

Посредник - это незаинтересованная третья сторона, которой доверено завершение протокола (см. 1-й (a)). Незаинтересованность означает, что у посредника нет заинтересованности в результате работы протокола и склонности к одной из сторон . "Доверено" означает, что все участники протокола принимают все, что скажет посредник за истину, все его действия - как правильные, и уверены в том, что посредник выполнит свою часть протокола. Посредники помогают реализовать работу протоколов взаимодействия недоверяющих друг другу сторон.

В реальном мире в качестве посредников часто выступают юристы . Например, Алиса продает незнакомому ей Бобу машину. Боб хочет заплатить чеком, но у Алисы нет способа проверить, действителен ли чек . Алиса хочет, чтобы расчет по чеку был произведен прежде, чем право собственности перейдет к Бобу . Боб, который верит Алисе не больше, чем она ему, не хочет передавать чек, не получив права собственности .

Табл. 2-1. Действующие лица

Алиса	Первый участник всех протоколов
Боб	Второй участник всех протоколов
Кэрол	Третий участник в протоколах с участием трех и четырех сторон
Дэйв	Четвертый участник в протоколах с участием трех и четырех сторон
Ева	Злоумышленник (eavesdropper)
Мэллори	Взломщик протоколов
Трент	Заслуживающий доверия посредник
Уолтер	Контролер, защищает Алису и Боба в ряде протоколов
Пегги	Свидетель
Виктор	Проверяет подлинность

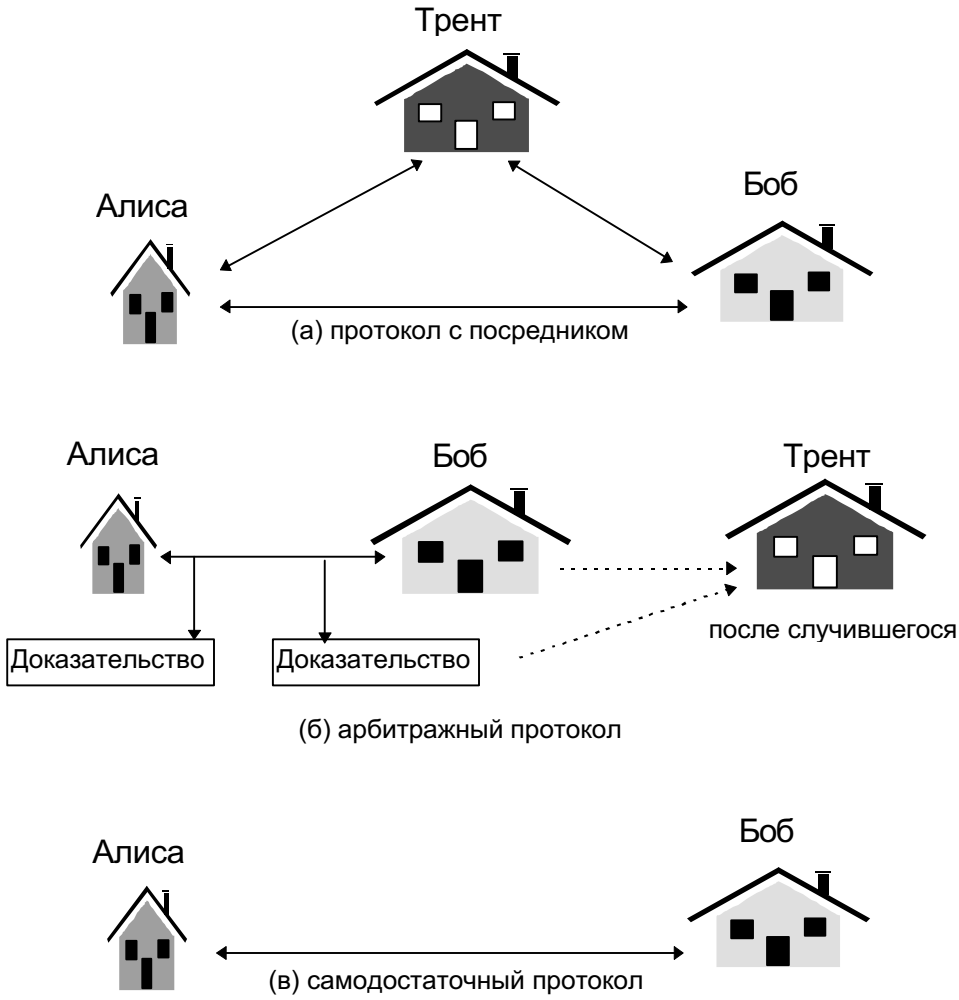


Рис. 2-1. Типы протоколов

Посредничество юриста устроит обоих. С его помощью Алиса и Боб могут выполнить следующий протокол, чтобы защитить себя от обмана:

(1) Алиса передает право собственности юристу.

(2) Боб передает чек юристу.

(3) Алиса депонирует чек.

(4) Дождавшись оплаты чека юрист передает право собственности Бобу. Если чек не оплачен в течение определенного времени, Алиса доказывает этот факт юристу, и тот возвращает право собственности Алисе.

В этом протоколе Алиса верит, что юрист не передаст Бобу право собственности до тех пор, пока чек не будет оплачен, и вернет право собственности Алисе, если чек оплачен не будет. Боб верит, что юрист будет обладать правом собственности до тех пор, пока чек не будет оплачен, и передаст право собственности Бобу сразу же после оплаты чека. Юрист не заботится об оплате чека. Он в любом случае выполнит свою часть протокола, ведь ему заплатят в любом случае.

В этом примере юрист играет роль посредника. Юристы часто выступают в роли посредников при завещаниях и иногда при переговорах о контракте. Различные биржи выступают в качестве посредников между покупателями и продавцами.

В качестве посредника может выступить и банк - для покупки машины:

(1) Боб заполняет чек и передает его в банк.

(2) Если на счету Боба достаточно денег для покрытия чека, банк заверяет чек и возвращает его Бобу.

(3) Алиса передает Бобу право собственности, а Боб передает Алисе заверенный чек.

(4) Алиса депонирует чек.

Этот протокол работает, потому что Алиса верит банковскому свидетельству. Алиса верит, что банк сохранит деньги Боба для нее и не использует их для финансирования сомнительных операций с недвижимостью в

банановых республиках.

Другим общепринятым посредником является нотариус. Когда Боб получает от Алисы заверенный нотариусом документ, он убежден, что Алиса подписала документ по своему желанию и собственноручно. При необходимости нотариус может выступить в суде и засвидетельствовать этот факт.

Понятие посредника старо как мир. Всегда существовали определенные люди - вожди, жрецы и тому подобное - обладавшие влиянием, позволяющим им действовать справедливо. Посредники играют определенную роль в нашем обществе, обман доверия подорвал бы занимаемое ими положение. Юристы-посредники, нарушающие правила игра, подвергаются наказанию - например, исключению из коллегии адвокатов. Это идеальная картина, в реальном мире положение, к сожалению, может отличаться от нее.

Этот идеал можно перенести на мир компьютеров, но с компьютерными посредниками существует ряд проблем:

— Легко найти нейтральную третью сторону, которой можно доверять, если вы знаете посредника и можете лично увидеть его. Две стороны, относящиеся друг к другу с подозрением, с тем же подозрением отнесутся и к безликому посреднику, затерянному где-то в сети.

— Компьютерная сеть должна обеспечить поддержку посредника. Занятость юристов общеизвестна, на кого в сети лягут дополнительные накладные расходы?

— Существует задержка, присущая всем протоколам с посредником.

— Посредник должен принимать участие в каждой транзакции, являясь узким местом в крупномасштабных реализациях любого протокола. Рост числа посредников смягчит эту проблему, но вырастет и цена этой услуги.

— Так как каждый в сети должен доверять посреднику, то посредник представляет собой слабое место сети при попытке ее взлома.

Несмотря на это посредничество все еще активно используется. В протоколах с использованием посредника эту роль будет играть Трент.

Арбитражные протоколы

Используемый из-за высокой стоимости найма посредников арбитражные протокол может быть разбит на два **подпротокола** нижнего уровня. Первый представляет собой протокол без посредника, используемый при желании сторон выполнить протокол. Другой представляет собой протокол с посредником, приглашаемым в исключительных обстоятельствах - при наличии разногласий между сторонами. Соответствующий специальный посредник называется **арбитром** (см. 1-й(б)).

Арбитр, как и посредник, представляет собой незаинтересованного участника протокола, которому доверяют обе стороны. В отличие от посредника он непосредственно не принимает участия в каждой отдельной реализации протокола и приглашается только для проверки честности выполнения протокола сторонами.

Профессиональными арбитрами являются судьи. В отличие от нотариусов к судьям обращаются только при появлении разногласий. Алиса и Боб могут заключить контракт без участия судьи. Судья никогда не узнает о контракте, если одна из сторон не подаст на другую в суд. Протокол подписания контракта можно формализовать следующим образом:

Подпротокол без посредника (выполняется всегда):

(1) Алиса и Боб договариваются об условиях контракта.

(2) Алиса подписывает контракт.

(3) Боб подписывает контракт.

Подпротокол с использованием арбитра (выполняется при наличии разногласий):

(4) Алиса и Боб предстают перед судьей.

(5) Алиса предоставляет свои доказательства.

(6) Боб предоставляет свои доказательства.

(7) Судья принимает решение на основании доказательств.

Различие используемых в этой книге понятий посредника и арбитра состоит в том, что участие арбитра происходит не всегда. Стороны обращаются к судье только при разногласиях. Если разногласий нет, судья не нужен.

Существуют арбитражные компьютерные протоколы. Они предполагают, что участвующие стороны честны, но при подозрении о возможном мошенничестве по существующему набору данных третья сторона, которой доверяют участники, сможет обнаружить факт мошенничества. Хороший арбитражный протокол позволяет арбитру установить и личность мошенника. Арбитражные протоколы обнаруживают, а не предупреждают мошенничество. Неотвратимость обнаружения выступает в качестве предупредительной меры, предотвращая мошенничество.

Самодостаточные протоколы

Самодостаточный протокол является лучшим типом протокола. Он полностью обеспечивает честность сторон (см. 1-й(в)). Для выполнения протокола не нужен ни посредник, не решающий споры арбитр. Само построение протокола обеспечивает отсутствие споров. Если одна из сторон попытается смошенничать, мошенничество будет немедленно обнаружено другой стороной, и протокол прекратит выполняться. Чего бы не пыталась добиться мошенничающая сторона, этому не суждено случиться.

В лучшем мире любой протокол должен быть самодостаточным, но, к несчастью, не существует самодостаточных протоколов для каждой ситуации.

Попытки вскрытия протоколов

Криптографические попытки взлома могут быть направлены против криптографических алгоритмов, используемых в протоколах, против криптографических методов, используемых для реализации алгоритмов и протоколов или непосредственно против протоколов. Поскольку в этом разделе книги обсуждаются именно протоколы, я предполагаю, что криптографические алгоритмы и методы безопасны, и рассматриваю только попытки вскрытия протоколов.

Люди могут использовать множество способов взломать протокол. Некоторые, не являясь участниками протокола, могут "подслушивать" какую-то часть или весь протокол. Это называется **пассивным вскрытием**, так как взломщик не воздействует на протокол. Все, что он может сделать - это проследить за протоколом и попытаться добыть информацию. Этот тип вскрытия соответствует вскрытию с использованием только шифротекста, обсуждавшемуся в разделе 1.1. Так как пассивные вскрытия трудно обнаружить, протоколы стремятся предотвращать, а не обнаруживать их. В этих протоколах роль злоумышленника будет играть Ева.

В другом случае взломщик может попытаться изменить протокол для собственной выгоды. Он может выдать себя за другого, ввести новые сообщения в протокол, заменить одно сообщение другим, повторно передать старые сообщения, разорвать канал связи или изменить хранящуюся в компьютере информацию. Такие действия называются активным вскрытием, так как они требуют активного вмешательства. Эти формы вскрытия зависят от вида сети.

Пассивные взломщики стараются получить информацию об участниках протокола. Они собирают сообщения, переданные различными сторонами, и пытаются криптоанализировать их. Попытки активного вскрытия, с другой стороны, преследуют более широкий набор целей. Взломщик может быть заинтересован в получении информации, ухудшении работы системы или получении несанкционированного доступа к ресурсам.

Активные вскрытия более серьезны, особенно в отношении протоколов, в которых стороны не обязательно доверяют друг другу. Взломщик не обязательно кто-то совсем посторонний, он может быть зарегистрированным пользователем системы и даже системным администратором. Может быть даже несколько активных взломщиков, работающих вместе. В этой книге роль злонамеренного активного взломщика будет играть Мэллори.

Взломщиком может быть и один из участников протокола. Он может обманывать, выполняя протокол, или вовсе не следовать правилам протокола. Такой взломщик называется **мошенником**. **Пассивные мошенники** выполняют правила протокола, но стараются получить больше информации, чем предусмотрено протоколом. **Активные мошенники** нарушают работу протокола, пытаясь смошенничать.

Очень трудно поддерживать безопасность протокола, если большинство его участников - активные мошенники, но иногда активное мошенничество может быть обнаружено законными участниками. Конечно, протоколы должны быть защищены и от пассивного мошенничества.

2.2 Передача информации с использованием симметричной криптографии

Как двум сторонам безопасно обмениваться информацией? Конечно же, шифрую свои сообщения. Посмотрим, что должно произойти, когда Алиса посылает шифрованное сообщение Бобу (полный протокол гораздо сложнее).

(1) Алиса и Боб выбирают систему шифрования.

(2) Алиса и Боб выбирают ключ.

(3) Алиса шифрует открытый текст своего сообщения с использованием алгоритма шифрования и ключа, п олучая шифрованное сообщение.

(4) Алиса посылает шифрованное сообщение Бобу.

(5) Боб дешифрирует шифротекст сообщения с использованием алгоритма шифрования и ключа, получая о ткрытый текст сообщения.

Что может Ева, находясь между Алисой и Бобом, узнать, подслушивая этот протокол ? Если она может подслушать только передачу на этапе (4), ей придется подвергнуть шифротекст криптоанализу . Это пассивное вскрытие представляет собой вскрытие с использованием только шифротекста, применяемые алгоритмы усто йчивы (насколько нам известно) по отношению к любым вычислительным мощностям, который может запол учить Ева для решения проблемы.

Ева, однако, не глупа. Она может также подслушать и этапы (1) и (2). Тогда ей станут известны алгоритм и ключ - также как и Бобу. Когда она перехватит сообщение на этапе (4), то ей останется только дешифровать его самостоятельно.

В хорошей криптосистеме безопасность полностью зависит от знания ключа и абсолютно не зависит от зн ания алгоритма. Именно поэтому управление ключами так важно в криптографии . Используя симметричный алгоритм, Алиса и Боб могут открыто выполнить этап (1), но этап (2) они должны сохранить в тайне . Ключ должен оставаться в секрете перед, после и в течение работы протокола - до тех пор, пока должно оставаться в тайне передаваемое сообщение - в противном случае сообщение тут же будет раскрыто . (О криптографии с открытыми ключами, решающей эту проблему иначе, рассказывается в разделе 2.5 .)

Мэллори, активный взломщик, может сделать кое-что другое. Он может попытаться нарушить линию связи не этапе (4), сделав так, что Алиса вообще не сможет передавать информацию Бобу . Мэллори также может перехватить сообщение Алисы и заменить его своим собственным . Если ему удалось узнать ключ (перехватив обмен информацией на этапе (2) или взломав криптосистему), он сможет зашифровать свое сообщение и отпр авить его Бобу вместо перехваченного, и Боб не сможет узнать, что сообщение отправлено не Алисой. Если Мэ ллори не знает ключа, он может только создать сообщение, превращающееся при дешифровке в бессмыслицу . Боб, считая, что сообщение отправлено Алисой, может решить, что либо у Алисы, либо в сети возникли серье зные проблемы.

А Алиса? Что она может сделать, чтобы испортить протокол? Она может передать копию ключа Еве, и тогда Ева сможет читать все, что говорит Боб, и напечатать его слова в *Нью-Йорк Таймс*. Это серьезно, но проблема не в протоколе. Алиса и так может передавать Еве любые открытые тексты, передаваемые с использованием протокола. Конечно, то же самое может сделать и Боб. Протокол предполагает, что Алиса и Боб доверяют друг другу. Итак, симметричным криптосистемам присущи следующие проблемы :

— Распределение ключей должно проводиться в секрете . Ключи столь же важны, как и все сообщения, з ашифрованные этими ключами, так как знание ключа позволяет раскрыть все сообщения . Для распространенных систем шифрования задача распределения ключей - серьезнейшая задача . Часто курьеры лично доставляют ключи по назначению.

— Если ключ скомпрометирован (украден, разгадан, выпытан, получен за взятку и т.д.), то Ева сможет расшифровать все сообщения, зашифрованные этим ключом . Она сможет также выступить в качестве одной из сторон и создавать ложные сообщения, дурача другую сторону .

— В предположении, что каждая пара пользователей сети использует отдельный ключ, общее число ключей быстро возрастает с ростом числа пользователей . Сеть из *n* пользователей требует *n{n - l)/2* ключей. Например, для общения 10 пользователей между собой нужно 45 различных ключей, для 100 пользователей потребуется 4950 ключей. Решение проблемы - в уменьшении числа пользователей, но это не всегда во зможно.

2.3 Однонаправленные функции

Понятие **однонаправленной функции** является центральным в криптографии с открытыми ключами . Не являясь протоколами непосредственно однонаправленные функции представляют собой краеугольный камень большинства протоколов, обсуждаемых в этой книге .

Однонаправленные функции относительно легко вычисляются, но инвертируются с большим трудом. То есть, зная *x* просто рассчитать *f(x)*, но по известному *f(x)* нелегко вычислить *x.* Здесь, "нелегко" означает, что для вычисления *x* по *f(x)* могут потребоваться миллионы лет, даже если над этой проблемой будут биться все компь ютеры мира.

Хорошим примером однонаправленной функции служит разбитая тарелка . Легко разбить тарелку на тысячу крошечных кусочков.. Однако, нелегко снова сложить тарелку из этих кусочков .

Это звучит красиво, но туманно и непонятно. Математически строгого доказательства существования однонаправленных функций нет, нет и реальных свидетельств возможности их построения [230, 530, 600, 661]. Несмотря на это, многие функции выглядят в точности как однонаправленные: мы можем рассчитать их и, до сих пор, не знаем простого способа инвертировать их. Например, в ограниченной окрестности легко вычислить x^2, но намного сложнее $x^{1/2}$. В оставшейся части раздела я собираюсь притвориться, что однонаправленные функции существуют. Мы поговорим об этом в еще разделе 11.2.

Итак, что же хорошего в однонаправленных функциях? Непосредственно их нельзя использовать для шифрования. Сообщение, зашифрованное однонаправленной функцией бесполезно - его невозможно дешифровать. (Упражнение: напишите на тарелке что-нибудь, разбейте тарелку на крошечные осколки и затем отдайте их приятелю. Попросите его прочитать сообщение. Посмотрите, как он будет озадачен однонаправленной функцией.) Для криптографии с открытыми ключами нам нужно что-то другое (хотя существуют и непосредственные криптографические применения однонаправленных функций - см. раздел 3.2).

Однонаправленная функция с люком - это особый тип однонаправленной функции, с секретной лазейкой. Ее легко вычислить в одном направлении и трудно - в обратном. Но если вам известен секрет, вы можете легко рассчитать и обратную функцию. То есть, легко вычислить *f(x)* по заданному *x*, но трудно по известному *f(x)* вычислить *x*. Однако, существует небольшая секретная информация, *y*, позволяющая, при знании *f(x)* и *y*, легко вычислить *x*.

В качестве хорошего примера однонаправленной функции с люком рассмотрим часы. Легко разобрать часы на сотни малюсеньких кусочков и трудно снова собрать из этих деталей работающие часы. Но, с секретной информацией - инструкцией по сборке - намного легче решить эту задачу.

2.4 Однонаправленные хэш-функции

У **однонаправленной хэш-функции** может быть множество имен: функция сжатия, функция сокращения contraction function, краткое изложение, характерный признак, криптографическая контрольная сумма, код целостности сообщения (message integrity check, MIC) и код обнаружения манипуляции (manipulation detection code, MDC). Как бы она не называлась эта функция является центральной в современной криптографии. Однонаправленные хэш-функции - это другая часть фундамента многих протоколов.

Хэш-функции, долгое время использующиеся в компьютерных науках, представляют собой функции, математические или иные, которые получают на вход строку переменной длины (называемую **прообразом**) и преобразуют ее в строку фиксированной, обычно меньшей, длины (называемую значением хэш-функции). В качестве простой хэш-функции можно рассматривать функцию, которая получает прообраз и возвращает байт, представляющий собой XOR всех входных байтов.

Смысл хэш-функции состоит в получении характерного признака прообраза - значения, по которому анализируются различные прообразы при решении обратной задачи. Так как обычно хэш-функция представляет собой соотношение "многие к одному", невозможно со всей определенностью сказать, что две строки совпадают, но их можно использовать, получая приемлемую оценку точности.

Однонаправленная хэш-функция - это хэш-функция, которая работает только в одном направлении: легко вычислить значение хэш-функции по прообразу, но трудно создать прообраз, значение хэш-функции которого равно заданной величине. Упоминавшиеся ранее хэш-функции, вообще говоря, не являются однонаправленными: задав конкретный байт, не представляет труда создать строку байтов, XOR которых дает заданное значение. С однонаправленной хэш-функцией такого не выйдет. Хорошей однонаправленной хэш-функцией является хэш-функция **без столкновений** - трудно создать два прообраза с одинаковым значением хэш-функции.

Хэш-функция является открытой, тайны ее расчета не существует. Безопасность однонаправленной хэш-функцией заключается именно в ее однонаправленности. У выхода нет видимой зависимости от входа. Изменение одного бита прообраза приводи к изменению, в среднем, половины битов значения хэш-функции. Вычислительно невозможно найти прообраз, соответствующий заданному значению хэш-функции.

Посмотрите на это как на способ получить характерные признаки файлов. Если вы хотите проверить, что у кого-то есть тот же файл, что и у вас, но вы не хотите, чтобы этот файл был передан вам, попросите послать вам значение хэш-функции. Если присланное значение хэш-функции совпадет с рассчитанным вами, то почти наверняка чужой файл совпадает с вашим. Это особенно полезно при финансовых транзакциях, когда вы не хотите где-то в сети превратить снятие со счета $100 в снятие $1000. В обычных условиях вы можете использовать однонаправленную хэш-функцию без ключа, так что кто угодно может проверить значение хэш-функции. Если нужно, чтобы проверить значение хэш-функции мог только один получатель, прочтите следующий раздел.

Коды проверки подлинности сообщения

Код проверки подлинности сообщения (message authentication code, MAC), известный также как код про-

верки подлинности данных (data authentication code, DAG), представляет собой однонаправленную хэш-функцию с добавлением секретного ключа (см. раздел 18.14). Значение хэш-функции является функцией и прообраза, и ключа. Теория остается той же, что и для хэш-функций, но только тот, кто знает ключ, может проверить значение хэш-функции. MAC можно создать с помощью хэш-функции или блочного алгоритма шифрования, существуют также и специализированные MAC.

2.5 Передача информации с использованием криптографии с открытыми ключами

Взгляните на симметричный алгоритм как на сейф. Ключ является комбинацией. Знающий комбинацию человек может открыть сейф, положить в него документ и снова закрыть. Кто-то другой при помощи той же комбинации может открыть сейф и забрать документ. Тем, кто не знает комбинации, придется научиться взламывать сейфы.

В 1976 году Уитфилд Диффи и Мартин Хеллман навсегда изменили эту парадигму криптографии [496]. (NSA заявило, что знало о такой возможности еще в 1966 году, но доказательств не представило.) Они описали **криптографию с открытыми ключами**, используя два различных ключа - один открытый и один закрытый. Определение закрытого ключа по открытому требует огромных вычислительных затрат. Кто угодно, используя открытый ключ может зашифровать сообщение, но не расшифровать его. Расшифровать сообщение может только владелец закрытого ключа. Это похоже на превращение криптографического сейфа в почтовый ящик. Шифрование с открытым ключом аналогично опусканию письма в почтовый ящик, любой может сделать это, опустив письмо в прорезь почтового ящика. Дешифрирование с закрытым ключом напоминает извлечение почты из почтового ящика. Обычно это гораздо сложнее - вам может понадобиться сварочный агрегат. Однако, если вы знаете секрет (у вас есть ключ от почтового ящика), вы без труда достанете вашу почту.

Математической основой процесса являются ранее обсуждавшиеся однонаправленные хэш-функции с люком. Шифрование выполняется в прямом направлении. Указания по шифрованию открыты, каждый может зашифровать сообщение. Дешифрирование выполняется в обратном направлении. Оно настолько трудоемко, что, не зная секрета, даже на компьютерах Cray за тысячи (и миллионы) лет невозможно расшифровать сообщение. Секретом, или люком, и служит закрытый ключ, он делает дешифрирование таким же простым, как и шифрование. Вот как, используя криптографию с открытыми ключами, Алиса может послать сообщение Бобу:

(1) Алиса и Боб согласовывают криптосистему с открытыми ключами.

(2) Боб посылает Алисе свой открытый ключ.

(3) Алиса шифрует свое сообщение и отправляет его Бобу.

(4) Боб расшифровывает сообщение Алисы с помощью своего закрытого ключа.

Обратите внимание, что криптография с открытыми ключами устраняет проблему распределения ключей, присущую симметричным криптосистемам. Раньше Алиса и Боб должны были тайно договориться о ключе. Алиса могла выбрать любой ключ, но ей нужно было передать его Бобу. Она могла сделать это заранее, но это требует от нее определенной предусмотрительности. Она могла бы послать ключ с секретным курьером, но для этого нужно время. Криптография с открытыми ключами все упрощает. Алиса может отправить Бобу секретное сообщение без каких-либо предварительных действий. У Евы, подслушивающей абсолютно все, есть открытый ключ Боба и сообщение, зашифрованное этим ключом, но она не сможет получить ни закрытый ключ Боба, ни текст сообщения.

Обычно целая сеть пользователей согласовывает используемую криптосистему. У каждого из них есть открытый и закрытый ключ, открытые ключи помещаются в общедоступной базе данных. Теперь протокол выглядит еще проще:

(1) Алиса извлекает открытый ключ Боба из базы данных.

(2) Алиса шифрует свое сообщение с помощью открытого ключа Боба и посылает его Бобу.

(3) Боб расшифровывает сообщение Алисы с помощью своего закрытого ключа.

В первом протоколе Боб должен был послать Алисе ее открытый ключ прежде, чем она могла отправить ему сообщение. Второй протокол больше похож на обычную почту. Боб не участвует в протоколе до тех пор, пока он не начнет читать сообщение.

Смешанные криптосистемы

Первые алгоритмы с открытым ключом стали известны в то же время, когда проходило DES обсуждение как предполагаемого стандарта. Это привело к известной партизанщине в криптографическом сообществе. Как это описывал Диффи [494]:

Прекрасные криптосистемы с открытым ключом, обсуждаемые в популярной и научной печати, тем не менее, не нашли соответствующего отклика среди криптографических чиновников. В том же году, когда была открыта криптография с открытыми ключами, Агентство национальной безопасности (NSA) предложило удобную криптографическую систему, разработанную фирмой IBM, в качестве федерального *Стандарта шифрования данных* (Data Encryption Standard, DES). Марти Хеллман и я критиковали это предложение из-за недостаточной длины ключа, но производители подготовились поддержать стандарт, и наша критика была воспринята многими как попытка помешать введению стандарта ради продвижения нашей собственной работы. Криптография с открытым ключом, в свою очередь, также подвергалась критике в популярной литературе [1125] и технических статьях [849, 1159], словно это был конкурирующий продукт, а не недавнее научное открытие. Это, однако, не помешало NSA объявить о своих заслугах в этой области. Его директор в одной из статей *Encyclopedia Britannica* [1461] указал, что "двухключевая криптография была открыта в Агентстве на десять лет раньше", хотя доказательства этого утверждения не были публично представлены.

В реальном мире алгоритмы с открытыми ключами не заменяют симметричные алгоритмы и используются не для шифрования сообщений, а для шифрования ключей по следующим двум причинам:

1. Алгоритмы с открытыми ключами работают медленно. Симметричные алгоритмы по крайней мере в 1000 раз быстрее, чем алгоритмы с открытыми ключами. Да, компьютеры становятся все быстрее и быстрее и лет через 15 криптография с открытыми ключами достигнет скоростей, сравнимых с сегодняшней скоростью симметричной криптографии. Но требования к объему передаваемой информации также возрастают, и всегда будет требоваться шифровать данные быстрее, чем это сможет сделать криптография с открытыми ключами.
2. Криптосистемы с открытыми ключами уязвимы по отношению к вскрытию с выбранным открытым текстом. Если $C = E(P)$, где P - открытый текст из n возможных открытых текстов, то криптоаналитику нужно только зашифровать все n возможных открытых текстов и сравнить результаты с C (помните, ключ шифрования общедоступен). Он не сможет раскрыть ключ дешифрирования, но он сможет определить P.

Вскрытие с выбранным открытым текстом может быть особенно эффективным, если число возможных шифрованных сообщений относительно мало. Например, если P - это денежная сумма в долларах, меньшая чем $1000000, то такое вскрытие сработает, криптоаналитик переберет весь миллион значений. (Эта проблема решается с помощью вероятностного шифрования, см. раздел 23.15.) Даже если P не так хорошо определено, такое вскрытие может быть очень эффективно. Полезным может быть простое знание, что шифротекст не соответствует конкретному открытому тексту. Симметричные криптосистемы не чувствительны к вскрытиям такого типа, так как криптоаналитик не может выполнить тестовых дешифровок с неизвестным ключом.

В большинстве реализаций криптография с открытыми ключами используется для засекречивания и распространения сеансовых ключей, которые используются симметричными алгоритмами для закрытия потока сообщений [879]. Иногда такие реализации называются смешанными (гибридными) криптосистемами

(1) Боб посылает Алисе свой открытый ключ

(2) Алиса создает случайный сеансовый ключ, шифрует его с помощью открытого ключа Боба и передает его Бобу.

$E_B(K)$

(3) Боб расшифровывает сообщение Алисы, используя свой закрытый ключ, для получения сеансового ключа.

$D_B(E_B(K))=K$

(4) Оба участника шифруют свои сообщения с помощью одного сеансового ключа.

Использование криптографии с открытыми ключами для распределения ключей решает очень важную проблему распределения ключей. В симметричной криптографии ключ шифрования данных, если он не используется, валяется без дела. Если Ева заполучит его, она сможет расшифровать все закрытые этим ключом сообщения. С помощью приведенного протокола при необходимости зашифровать сообщения создается сеансовый ключ, который уничтожается по окончании сеанса связи. Это значительно уменьшает риск компрометации сеансового ключа. Конечно, к компрометации чувствителен и закрытый ключ, но риска значительно меньше, так как в течение сеанса этот ключ используется только один раз для шифрования сеансового ключа. Подробно связанные с этим вопросы обсуждаются в разделе 3.1.

Головоломки Меркла

Ральф Меркл (Ralph Merkle) изобрел первую схему криптографии с открытыми ключами. В 1974 году он записался на курс по компьютерной безопасности в Калифорнийском университете, Беркли, который вел Ланс Хоффман (Lance Hoffman). Темой его курсовой работы, поданной раньше срока, была "Безопасная передача данных по небезопасным каналам" [1064]. Хоффман не понял предложения Меркла, и в конце концов Меркл прекратил занятия. Он продолжал работать над проблемой несмотря на продолжающееся непонимание его результатов.

Техника Меркла основывалась на головоломках ("puzzle"), которые отправителю и получателю решить лег-

че чем злоумышленнику. Вот как Алиса может послать шифрованное сообщение Бобу, не обмениваясь с ним ключом до того.

(1) Боб создает 2^{20} (другими словами, больше миллиона) сообщений типа: "Это головоломка номер x. Это секретный ключ номер y.", где x - случайное число, а y - случайный секретный ключ. И x, и y отличаются в каждом сообщении. Используя симметричный алгоритм, он шифрует каждое сообщение своим 20 би т-ным ключом и все их отправляет Алисе.

(2) Алиса выбирает одно сообщение и приступает к вскрытию **грубой силой**, пытаясь получить открытый текст. Эта работа является объемной, но не невозможной.

(3) Алиса шифрует свое секретное сообщение при помощи некоторого симметричного алгоритма полученным ею ключом и посылает это сообщение Бобу вместе с x.

(4) Боб знает, какой секретный ключ y он использовал в сообщении x, следовательно он может расшифровать сообщение Алисы.

Ева может взломать эту систему, но ей придется выполнить гораздо больше работы чем Алисе и Бобу . Для раскрытия сообщения на этапе (3) она должна будет вскрыть грубой силой каждое из 2^{20} сообщений, отправленных Бобом на этапе (1). Сложность этого вскрытия составит 2^{40}. Значения x также не помогут Еве, ведь они на этапе (1) присвоены случайным образом. В общем случае, вычислительные затраты Евы будут равны возв е-денным в квадрат вычислительным затратам Алисы.

Это выигрыш (n по отношению к n^2) невелик по криптографическим стандартам, но при определенных усл о-виях может быть достаточен. Если Алиса и Боб могут проверить десять тысяч ключей в секунду, каждому из них потребуется минута для выполнения своих действий и еще одна минута для передачи головоломок от Боба к Алисе по линии связи 1.544 Мбит/с. Если вычислительные возможности Евы сравнимы с приведенными, ей потребуется около года для взлома системы. Другие алгоритмы еще более устойчивы к вскрытию .

2.6 Цифровые подписи

Рукописные подписи издавна используются как доказательство авторства документа или, по крайней мере, согласия с ним. Что же так притягательно в подписи [1392]?

1. Подпись достоверна. Она убеждает получателя документа в том, что подписавший сознательно подп и-сал документ.
2. Подпись неподдельна. Она доказывает, что именно подписавший, и никто иной, сознательно подписал документ.
3. Подпись не может быть использована повторно. Она является частью документа, жулик не сможет перенести подпись на другой документ.
4. Подписанный документ нельзя изменить. После того, как документ подписан, его невозможно изм е-нить.
5. От подписи не возможно отречься. Подпись и документ материальны. Подписавший не сможет вп о-следствии утверждать, что он не подписывал документ.

В действительности, ни одно из этих утверждений не является полностью справедливым . Подписи можно подделать, свести с одного листа бумаги на другой, документы могут быть изменены после подписания . Однако, мы миримся с этими проблемами из-за того, что мошенничество затруднительно и может быть обнар ужено.

Хотелось бы реализовать что-нибудь подобное и на компьютерах, но есть ряд проблем . Во первых, компьютерные файлы скопировать не просто, а очень просто. Даже если подпись человека трудно подделать (например, графическое изображение рукописной подписи), можно легко вырезать правильную подпись из одного докуме н-та и вставить в другой. Простое наличие такой подписи ничего не означает. Во вторых, компьютерные файлы очень легко можно изменить после того, как они подписаны, не оставляя ни малейшего следа изменения .

Подпись документа с помощью симметричных криптосистем и посредника

Алиса хочет подписать цифровое сообщение и отправить его Бобу. Она может это сделать с помощью Тре н-та и симметричной криптосистемы.

Трент - это обладающий властью посредник, которому доверяют . Он может связываться и с Алисой, и с Б о-бом (и со всеми другими желающими подписывать цифровые документы). Он выдает секретный ключ, K_A, Али-се и другой секретный ключ, K_B, - Бобу. Эти ключи определяются задолго до начала действия протокола и могут быть использованы многократно для многих подписей.

(1) Алиса шифрует свое сообщение Бобу ключом K_A и посылает его Тренту.

(2) Трент, зная ключ K_A, расшифровывает сообщение.

(3) Трент добавляет к расшифрованному сообщению утверждение, что он получил это сообщение от Алисы, и шифрует это новое сообщение ключом K_B.

(4) Трент посылает новое сообщение Бобу.

(5) Боб расшифровывает сообщение ключом K_B. Он может прочитать и сообщение Алисы, и подтверждение Трента, что сообщение отправлено именно Алисой.

Откуда Трент узнает, что сообщение пришло именно от Алисы, а не от какого-то самозванца ? Он делает этот вывод из шифрования сообщения.

Также ли это хорошо, как подпись на бумаге? Посмотрим на требуемые свойства:

1. Эта подпись достоверна. Трент - это заслуживающий доверия посредник, и Трент знает, что сообщ е-ние получено от Алисы. Подтверждение Трента служит доказательством для Боба.
2. Эта подпись неподдельна. Только Алиса (и Трент, но ему все верят) знает K_A, поэтому только Алиса могла послать Тренту сообщение, зашифрованное ключом K_A. Если кто-нибудь попытается выдать с е-бя за Алису, Трент сразу заметит это на этапе (2) и не заверит подлинность.
3. Эту подпись нельзя использовать повторно. Если Боб попытается взять подтверждение Трента и пр и-соединить его к другому сообщению, Алиса закричит "Караул!" Посредник (Трент или кто-то совсем другой, имеющий доступ к той же информации) попросит Боба предъявить его сообщение и шифр о-ванное сообщение Алисы. Затем посредник зашифрует сообщение ключом K_A и увидит, что оно не со-ответствует шифрованному сообщению, переданному Бобом . Боб, конечно же, не сможет создать пр а-вильное шифрованное сообщение, потому что он не знает кл юча K_A.
4. Подписанный документ нельзя изменить. Если Боб попытается, получив документ, изменить его, Трент обнаружит мошенничество уже описанным способом.
5. От подписи невозможно отказаться. Если впоследствии Алиса заявит, что она никогда не посылала сообщение, подтверждение Трента докажет обратное. Помните, все доверяют Тренту, все, сказанное им - истина.

Если Боб захочет показать Кэрол документ, подписанный Алисой, он не сможет раскрыть ей свой секретный ключ. Ему придется снова о братиться к Тренту:

(1) Боб берет сообщение и утверждение Трента, что сообщение получено от Алисы, шифрует их ключом K_B и посылает обратно Тренту.

(2) Трент расшифровывает полученный пакет с помощью ключа K_B.

(3) Трент проверяет свою базу данных и подтверждает, что отправителем оригинального сообщения была Алиса.

(4) Трент шифрует полученный от Боба пакет ключом K_C, который он выделил для Кэрол, и посылает Кэрол шифрованный пакет.

(5) Трент расшифровывает полученный пакет с помощью ключа K_C. Теперь она может прочитать и сообщ е-ние, и подтверждение Трента, что сообщение отправлено Алисой.

Эти протоколы работают, но они требуют от Трента немалых затрат времени. Он должен целыми днями расшифровывать и шифровать сообщения, посредничая между каждой парой людей, которые хотят обмен и-ваться подписанными документами. Он должен хранить сообщения в базе данных (хотя этого можно избежать, посылая получателю копию шифрованного сообщения отправителя). Он будет узким местом любой системы связи, даже если он - просто бесчувственная компьютерная программа. .

Такого посредника как Трент, которому будут доверять все корреспонденты, тяжело найти и тяжело сохр а-нить. Трент должен быть непогрешим, если он сделает хотя бы одну ошибку на миллион подписей, никто не будет верить ему. Трент должен быть абсолютно безопасен . Если его база данных с секретными ключами когда-нибудь раскроется, или кто-нибудь сможет перепрограммировать его, все подписи станут бесполезными . Поя-вятся документы будто бы подписанные годы назад . Это приведет к хаосу. Правительства падут, и станет пр а-вить анархия. Такая схема теоретически может работать, но она недостаточно хороша для практического пр и-менения.

Деревья цифровых подписей

Ральф Меркл предложил систему цифровых подписей, основанную на криптографии с секретным ключом, создающей бесконечное количество одноразовых подписей, используя древовидную структуру [1067,1068]. Ос-новной идеей этой схемы является поместить корень дерева в некий открытый файл , удостоверяя его таким о б-

разом. Корень подписывает одно сообщение и удостоверяет подузлы дерева . Каждый из этих узлов подписывает одно сообщение и удостоверяет свои подузлы, и так далее .

Подпись документа с помощью криптографии с открытыми ключами

Существуют алгоритмы с открытыми ключами, которые можно использовать для цифровых подписей. В н е-которых алгоритмах - примером является RSA (см. раздел 19.3) - для шифрования может быть использован или открытый, или закрытый ключ. Зашифруйте документ своим закрытым ключом, и вы получите надежную ци ф-ровую подпись. В других случаях - примером является DSA (см. раздел 20.1) - для цифровых подписей исполь-зуется отдельный алгоритм, который невозможно использовать для шифрования . Эта идея впервые была изо-бретена Диффи и Хеллманом [496] и в дальнейшем была расширена и углублена в других работах [1282, 1328, 1024, 1283, 426]. Хороший обзор этой области приведен в [1099]. Основной протокол прост:

(1) Алиса шифрует документ своим закрытым ключом, таким образом подписывая его.

(2) Алиса посылает подписанный документ Бобу.

(3) Боб расшифровывает документ, используя открытый ключ Алисы, таким образом проверяя подпись.

Этот протокол гораздо лучше предыдущего. Трент не нужен ни для подписи документов, ни для ее проверки. (Он нужен для подтверждения, что открытый ключ принадлежит именно Алисе .) Трент не нужен сторонам даже для разрешения споров: Если Боб не смог осуществить этап (3), то он знает, что подпись неправильна. Такая подпись соответствует всем требованиям:

1. Эта подпись достоверна. Когда Боб расшифровывает сообщение с помощью открытого ключа Алисы, он знает что она подписала это сообщение.
2. Эта подпись неподдельна. Только Алиса знает свой закрытый ключ.
3. Эту подпись нельзя использовать повторно. Подпись является функцией документа и не может быть перенесена на другой документ.
4. Подписанный документ нельзя изменить. После любого изменения документа подпись не сможет больше подтверждаться открытым ключом Алисы.
5. От подписи невозможно отказаться. Бобу не требуется помощь Алисы при проверке ее подписи.

Подпись документа и метки времени

На самом деле, при определенных условиях Боб сможет смошенничать . Он может повторно использовать документ и подпись совместно. Это не имеет значения, если Алиса подписала контракт (одной копией подпи-санного контракта больше, одной меньше), но что если Алиса поставила цифровую подпись под чеком?

Предположим, что Алиса послала Бобу подписанный чек на $100. Боб отнес чек в банк, который проверил подпись и перевел деньги с одного счета на другой. Боб, выступающий в роли жулика, сохранил копию эле к-тронного чека. На следующей неделе он снова отнес его в этот или другой банк . Банк подтвердил подпись и пе-ревел деньги с одного счета на другой. Если Алиса не проверяет свою чековую книжку, Боб сможет продел ы-вать это годами.

Поэтому в цифровые подписи часто включают метки времени . Дата и время подписания документа доба в-ляются к документу и подписываются вместе со всем содержанием сообщения . Банк сохраняет эту метку врем е-ни в базе данных. Теперь, если Боб попытается получить наличные по чеку Алисы во второй раз, банк проверит метку времени по своей базе данных. Так как банк уже оплатил чек Алисы с той же меткой времени, то будет вызвана полиция. Затем Боб проведет лет 15 в тюрьме Ливенворт, изучая криптографические протоколы .

Подпись документа с помощью криптографии с открытыми ключами и однонаправленных хэш-функций

На практике алгоритмы с открытыми ключами часто недостаточно эффективны для подписи больших док у-ментов. Для экономии времени протоколы цифровой подписи нередко используют вместе с однонаправленными хэш-функциями [432, 433]. Алиса подписывает не документ, а значение хэш-функции для данного документа . В этом протоколе однонаправленная хэш-функция и алгоритм цифровой подписи согласовываются заранее .

(1) Алиса получает значение однонаправленной хэш-функции для документа.

(2) Алиса шифрует это значение своим закрытым ключом, таким образом подписывая документ.

(3) Алиса посылает Бобу документ и подписанное значение хэш-функции.

(4) Боб получает значение однонаправленной хэш-функции для документа, присланного Алисой. Затем, и с-пользуя алгоритм цифровой подписи, он расшифровывает подписанное значение хэш-функции с помощью

открытого ключа Алисы. Если подписанное значение хэш-функции совпадает с рассчитанным, подпись правильна.

Скорость заметно возрастает и, так как вероятность получить для двух различных документов одинаковое 160-битное значение хэш-функции составляет только один шанс из 2^{160}, можно безопасно приравнять подпись значения хэш-функции и подпись документа. Должна использоваться только однонаправленная хэш-функция, иначе создать разные документы с одним и тем же значением хэш-функции нетрудно, и подпись одного документа приведет к ошибочной подписи сразу многих документов.

У протокола есть и другие выгоды. Во первых, подпись может быть отделена от документа. Во вторых, значительно уменьшаются требования к объему памяти получателя, в котором хранятся документы и подписи. Архивная система может использовать этот протокол для подтверждения существования документов, не храня их содержания. В центральной базе данных могут храниться лишь значения хэш-функции для файлов. Вовсе не нужно просматривать файлы, пользователи помещают свои значения хэш-функции в базу данных, а база данных хранит эти значения, помечая их временем получения документа. Если в будущем возникнет какое-нибудь разногласие по поводу автора и времени создания документа, база данных сможет разрешить его при помощи хранящегося в ней значения хэш-функции. Подобная система имеет большое значение при хранении секретной информации: Алиса может подписать документ и сохранить его в секрете. Ей понадобится опубликовать документ, только если она захочет доказать свое авторство. (См. раздел 4.1).

Алгоритмы и терминология

Существует множество алгоритмов цифровой подписи. Все они представляют собой алгоритмы с открытыми ключами с закрытой частью для подписи документов и с открытой - для проверки подписи. Иногда процесс подписи называют **шифрованием с закрытым ключом**, а процесс проверки подписи - **дешифрированием с открытым ключом**. Это может ввести в заблуждение, являясь справедливым только для одного алгоритма, RSA. У других алгоритмов - другие реализации. Например, использование однонаправленных хэш-функций и меток времени иногда приводит к появлению дополнительных этапов при подписании и проверке подписи. Многие алгоритмы можно использовать для цифровой подписи, но нельзя для шифрования.

В общем случае я буду ссылаться на процессы подписи и проверки, не вдаваясь в подробности алгоритмов. Подпись сообщения с закрытым ключом K будет обозначаться как:

$S_K(M)$

а проверка подписи с помощью соответствующего открытого ключа как:

$V_K(M)$

Строку битов, присоединенную к документу после его подписания (в предыдущем примере, значение однонаправленной хэш-функции документа, зашифрованное зарытым ключом), будем называть **цифровой подписью** или просто **подписью**. Весь протокол, с помощью которого получатель сообщения проверяет личность отправителя и целостность сообщения, называется удостоверением подлинности. Более подробно эти протоколы рассматриваются в разделе 3.2.

Несколько подписей

Как Алисе и Бобу одновременно подписать один и тот же документ? В отсутствие однонаправленных хэш-функций существует две возможности. Алиса и Боб могут подписать различные копии одного и того же документа. Полученное сообщение будет в два раза длиннее первоначального документа. Или Алиса подписывает документ, а затем Боб подписывает подпись Алисы. Этот способ работает, но проверить подпись Алисы, не проверяя при этом подписи Боба, невозможно.

С помощью однонаправленных реализовать несколько подписей просто:

(1) Алиса подписывает значение хэш-функции документа.

(2) Боб подписывает значение хэш-функции документа.

(3) Боб посылает свою подпись Алисе.

(4) Алиса посылает Кэрол документ, свою подпись и подпись Боба.

(5) Кэрол проверяет подписи Алисы и Боба.

Алиса и Боб могут выполнить этапа (1) и (2) или параллельно, или последовательно. На этапе (5) Кэрол может проверить любую подпись независимо от другой.

Невозможность отказаться от цифровой подписи

Алиса может смошенничать с цифровыми подписями, и с этим ничего нельзя поделать. Она может подписать документ и затем утверждать, что она этого не делала. Сначала она, как обычно, подписывает письмо. Затем она анонимно раскрывает свой закрытый ключ или теряет в людном месте. Теперь Алиса утверждает, что ее подпись была скомпрометирована и использована кем-то другим, выдающим себя за нее. Она дезавуирует свою подпись под всеми документами, подписанными с помощью этого закрытого ключа. Это называется отказ от подписи.

Метки времени могут снизить эффект такого мошенничества, но Алиса всегда может заявить, что ее ключ был скомпрометирован раньше. Если Алиса правильно рассчитает время, она сможет подписать документ и затем успешно заявить, что она этого не делала. Поэтому так много говорится о хранении закрытых ключей в надежных местах - чтобы Алиса не могла добраться до своего и злоупотребить им.

Хотя с подобным злоупотреблением ничего нельзя сделать, можно предпринять некоторые действия, гарантирующие то, что старые подписи не будут признаны недостоверными из-за разногласий по новым подписям. (Например, Алиса может "потерять" свой ключ, чтобы не платить Бобу за подержанную машину, которую он вчера ей продал и, в результате, сделает недействительным свой банковский счет.) Получателю нужно проставлять метки времени для полученных документов [453]. Общая схема протокола приведена в [2, 8]:

(1) Алиса подписывает сообщение.

(2) Алиса создает заголовок, содержащий некоторую идентификационную информацию. Она присоединяет к заголовку подписанное сообщение, подписывает все вместе и посылает Тренту.

(3) Трент проверяет внешнюю подпись и подтверждает идентификационную информацию. Он добавляет метку времени к подписанному сообщению Алисы и идентификационной информации. Затем он подписывает все вместе и посылает пакет Алисе и Бобу.

(4) Боб проверяет подпись Трента, идентификационную информацию и подпись Алисы.

(5) Алиса проверяет сообщение, которое Трент послал Бобу. Если она не признает свое авторство, она быстро заявляет об этом.

В другой схеме Трент используется в качестве арбитра [209]. Получив подписанное сообщение, Боб посылает копию Тренту для проверки. Трент может подтвердить подпись Алисы.

Использование цифровых подписей

Одним из самых ранних предложенных применений цифровых подписей было упрощение проверки соблюдения договоров о ядерных испытаниях [1454, 1467]. Соединенные Штаты и Советский Союз (кто-нибудь помнит Советский Союз?) разрешили друг другу разместить на чужой территории сейсмографы для слежения за ядерными испытаниями. Проблема была в том, что каждая из сторон должна была уверена в том, что другая сторона не подделала данные этих сейсмографов. Одновременно, другая сторона должна была быть уверена, что эти датчики посылают только ту информацию, которая нужна для слежения за ядерными испытаниями.

Метод условного удостоверения подлинности может решить первую проблему, но только цифровые подписи могут решить обе проблемы. Сторона, на территории которой стоит сейсмограф, может прочесть, но не изменить данные сейсмографа, а следящая сторона знает, что данные не были подделаны.

2.7 Цифровые подписи и шифрование

Объединив цифровые подписи и криптографию с открытыми ключами, мы разрабатываем протокол, комбинирующий безопасность шифрования и достоверность цифровых подписей. Сравните с письмом от вашей маиы. Подпись удостоверяет авторство а конверт обеспечивает тайну.

(1) Алиса подписывает сообщение с помощью своего закрытого ключа.

$S_A(M)$

(2) Алиса шифрует подписанное сообщение открытым ключом Боба и посылает его Бобу.

$E_B(S_A(M))$

(3) Боб расшифровывает сообщение с помощью своего закрытого ключа.

$D_B(E_B(S_A(M))) = S_A(M)$

(4) Боб проверяет подпись с помощью открытого ключа Алисы и восстанавливает сообщение.

$V_A(S_A(M)) = M$

Подпись перед шифрованием выглядит естественно. Когда Алиса пишет письмо, она подписывает его и затем кладет в конверт. Если она положит письмо в конверт неподписанным, то Боб может забеспокоиться, вдруг письмо было тайно подменено. Если Боб покажет Кэрол письмо Алисы и конверт, Кэрол может обвинить Боба, что он врет о том, какое письмо в каком конверте пришло.

В электронной корреспонденции точно также является разумным использование подписи перед шифрованием [48]. Это не только более безопасно - враг не сможет удалить подпись из шифрованного сообщения и добавить свою собственную - но существуют и юридические соображения: если подписываемый текст не виден подписывающему, когда он ставит подпись, то юридическая сила подписи невелика [1312]. Существуют также некоторые криптографические способы вскрытия такой последовательности действий, использующей подписи RSA (см. раздел 19.3).

Для Алисы не существует причин использовать одну пару ключей - открытый/закрытый - для шифрования и подписи. У нее может быть две пары ключей: одна для шифрования и одна для подписи. У такого разделения есть свои преимущества: Алиса может передать свой ключ шифрования полиции, не компрометируя свою подпись, один ключ может быть условно передан (см. раздел 4.13), не влияя на другой. У ключей могут быть различные длины и сроки действия.

Конечно же, для предотвращения повторного использования сообщений с этим протоколом должны быть использованы метки времени. Метки времени также могут защитить от других возможных ловушек, пример одной из которых приведен ниже.

Возвращение сообщения при приеме

Рассмотрим реализацию этого протокола с дополнительной возможностью подтверждения сообщений - получив сообщение, Боб обязательно возвращает подтверждение приема.

(1) Алиса подписывает сообщение с помощью своего закрытого ключа, шифрует подписанное сообщение открытым ключом Боба и посылает его Бобу.

$E_B(S_A(M))$

(2) Боб расшифровывает сообщение с помощью своего закрытого ключа, проверяет подпись с помощью открытого ключа Алисы и восстанавливает сообщение.

$V_A(D_B(E_B(S_A(M)))) = M$

(3) Боб подписывает сообщение с помощью своего закрытого ключа, шифрует подписанное сообщение открытым ключом Алисы и посылает его Алисе обратно.

$E_A(S_B(M))$

(4) Алиса расшифровывает сообщение с помощью своего закрытого ключа и проверяет подпись с помощью открытого ключа Боба. Если полученное сообщение совпадает с отправленным, она знает, что Боб получил правильное сообщение.

Если для шифрования и проверки цифровой подписи используется один и тот же алгоритм, то существует возможность вскрытия [506]. В таких случаях операция цифровой подписи - противоположность операции шифрования: $V_X = E_X$ и $S_X = D_X$.

Пусть Мэллори - зарегистрированный пользователь со своей парой ключей: открытым и закрытым. Теперь посмотрим, как он сможет читать почту Боба. Сначала он запишет сообщение Алисы Бобу - этап (1). Затем, немного погодя, он пошлет это сообщение Бобу, утверждая, что оно отправлено самим Мэллори. Боб, думая, что это обычное сообщение от Мэллори, дешифрирует это сообщение своим закрытым ключом и пытается проверить подпись Мэллори, дешифрируя ее с помощью открытого ключа Мэллори. В результате получается полная чепуха:

$E_A(D_B(E_B(D_A(M)))) = E_M(D_A(M))$

Даже в этом случае, следуя протоколу, Боб посылает Мэллори полученное сообщение:

$E_M(D_B(E_M(D_A(M))))$

Теперь Мэллори остается только расшифровать сообщение с помощью своего закрытого ключа, зашифровать его открытым ключом Боба, расшифровать снова с помощью своего закрытого ключа и зашифровать открытым ключом Алисы. *Voilà!* Мэллори получает *M*.

Отнюдь не глупо предположить, что Боб может автоматически посылать Мэллори квитанцию. Этот протокол, например, может быть встроен в его коммуникационное программное обеспечение и посылать квитанции автоматически. Именно готовность сообщить о приеме чепухи и нарушает безопасность. Если Боб проверит сообщение на осмысленность перед отправкой квитанции, он сможет избежать таких проблем с безопасностью.

Существуют модернизации этого способа вскрытия, предполагающие, что Мэллори пошлет Бобу сообщение, отличное от того, которое он желает перехватить. Никогда не подписывайте произвольных сообщений от других людей и не передавайте результаты дешифровки произвольных сообщений иным людям.

Обнаружение вскрытия, основанного на возвращении сообщения

Только что описанное вскрытие работает потому, что операция шифрования совпадает с операцией проверки подписи, а операция дешифрирования - с операцией подписи. Операции шифрования и цифровой подписи в безопасном протоколе должны хотя бы слегка отличаться. Проблему решает использование различных ключей для каждой операции, или использование для каждой операции различных алгоритмов, или применение меток времени, которые делают различными принятое и отправляемое сообщения, или цифровая подпись с помощью однонаправленной хэш-функции (см. раздел 2.6). Тогда, в общем случае, следующий протокол, использующий алгоритм с открытым ключом, является безопасным:

(1) Алиса подписывает сообщение.

(2) Алиса шифрует подписанное сообщение открытым ключом Боба (используя алгоритм, отличающийся от алгоритма цифровой подписи) и посылает его Бобу.

(3) Боб расшифровывает сообщение с помощью своего закрытого ключа

(4) Боб проверяет подпись Алисы.

Вскрытия криптографии с открытыми ключами

Во всех подобных протоколах криптографии с открытыми ключами я не рассказал, как Алиса получает открытый ключ Боба. Подробно этот вопрос описан в разделе 3.1, но о нем стоит упомянуть и здесь.

Проще всего узнать чей-то открытый ключ, считав его откуда-то из безопасной базы данных. Эта база данных должна быть общедоступна, чтобы каждый мог получить нужный ему ключ. База данных должна быть защищена от несанкционированной записи, в противном случае Мэллори сможет подменить открытый ключ Боба. После этого Боб уже не сумеет читать адресованные ему сообщения, зато это сможет сделать Мэллори.

Даже если открытые ключи хранятся в надежной базе данных, Мэллори может подменить их при передаче. Чтобы воспрепятствовать этому, Трент должен подписывать каждый открытый ключ, используя свой собственный закрытый ключ. Трента, который действует подобным образом, часто называют **Органом сертификации ключей** или **Центром распределения ключей** (Key Distribution Center, KDC). На практике KDC подписывает сложное сообщение, состоящее из имени пользователя, его открытого ключа и другой информации о пользователе. Это подписанное сложное сообщение и хранится в базе данных KDC. Когда Алиса получает ключ Боба, она проверяет подпись KDC, удостоверяясь в правильности ключа.

При окончательном анализе видно, что и это только затрудняет, но не делает невозможным мошенничество Мэллори. Алиса же должна откуда-то получить открытый ключ KDC. Мэллори нужно подменить этот ключ своим открытым ключом, испортить базу данных и заменить правильные ключи своими (подписанными его закрытым ключом, как если бы он и был KDC), и его дело сделано. Но, даже подписи на бумаге могут быть подделаны, если Мэллори всерьез возьмется за дело. Подробно обмен ключами рассматривается в разделе 3.1.

2.8. Генерация случайных и псевдослучайных последовательностей

Почему даже в книге по криптографии снова эти докучливые рассуждения о генерации случайных чисел? Генератор случайных чисел встроен в каждый компилятор, обычный вызов функции. Почему бы не использовать его? К сожалению, эти генераторы случайных чисел почти наверняка недостаточно безопасны для криптографии и, возможно, даже не совсем случайны. Большинство из них весьма плохи.

Генераторы случайных чисел на самом деле совсем не случайны, потому что им и не нужно быть такими. Для большинства приложений, например, компьютерных игр, требуется так мало случайных чисел, что их неслучайность вряд ли будет заметна. Однако, криптография очень чувствительна к свойствам генераторов случайных чисел. Примените плохой генератор, и у вас появятся таинственные корреляции и странные результаты [1231, 1238]. Если ваша безопасность зависит от генератора случайных чисел, таинственные корреляции и странные результаты являются абсолютно не тем, чего бы вы желали добиться.

Проблема в том, что генератор случайных чисел не создает случайной последовательности. Он, возможно, не выдает ничего даже отдаленно напоминающего случайную последовательность. Конечно, невозможно создавать на компьютере что-то по настоящему случайное. Дональд Кнут приписывал фон Нейману следующие слова: "Каждый, кто занимается арифметическими методами получения случайных чисел, определенно грешит" [863]. Компьютеры - это детерминированные бестии: закладывается известный материал, выполняются полностью предсказуемые действия, и что-то отличное выползает с другого конца. Подача одного и того же на вход в двух различных случаях приведет к одному и тому же результату. Заложите одинаковые исходные данные в два

идентичных компьютера, и оба они подсчитают одно и то же. Компьютер может находиться только в ограниченном числе состояний (очень большом, но все же ограниченном), и выдаваемый результат всегда будет строго определяться исходными данными и текущим состоянием компьютера. Это значит, что любой генератор случайных чисел на компьютере (по меньшей мере, на конечном автомате), по определению, периодичен. А все, что периодично, по определению, предсказуемо. А все, что предсказуемо, не может быть случайным. Для настоящего генератора случайных чисел нужно подавать на вход что-нибудь случайное, компьютер же не может обеспечить это требование.

Псевдослучайные последовательности

Лучшее, что может сделать компьютер - это **генератор псевдослучайных последовательностей** Что это такое? Многие пытались дать его формальное определение, но я уклонюсь от этого. Псевдослучайная последовательность - это что-то, выглядящее как случайное. Период последовательности должен быть достаточно велик, поэтому конечная последовательность разумной длины - которая в действительности и используется - не периодична. Если вам нужен миллиард случайных бит, не пользуйтесь генератором последовательности, повторяющейся каждые шестнадцать тысяч бит. Эти относительно короткие непериодические подпоследовательности должны быть, насколько это возможно, неотличимы от случайных последовательностей. Например, в них должно быть примерно одинаковое количество единиц и нулей, около половины серий (последовательностей одинаковых бит) должны быть единичной длины, четверть - состоять из двух бит, восьмая часть - из трех, и т.д. Эти последовательности должны быть несжимаемы. Распределение длин серий для нулей и единиц должно быть одинаковым [643, 863, 99, 1357]. Эти свойства могут быть измерены опытным путем и затем сравнены с ожидаемыми статистически с помощью статистики хи-квадрат. Для наших целей генератор последовательности считается псевдослучайным, если он обладает следующим свойством:

1. Он выглядит случайно. Это означает, что он проходит все тесты на случайность, которые нам удалось найти. (Начните с приведенных в [863].)

Множество усилий было затрачено на создание хороших псевдослучайных последовательностей на компьютере. Обсуждение генераторов в большом количестве можно найти в академической литературе вместе с различными тестами на случайность. Все эти генераторы периодичны (этого невозможно избежать), но, если их период 2^{256} и выше, они могут быть использованы в самых серьезных приложениях.

Проблема именно в этих таинственных корреляциях и странных результатах. Каждый генератор псевдослучайных последовательностей создает такие странности, если вы используете его определенным образом. А это именно то, что нужно криптоаналитику для взлома системы.

Криптографически безопасные псевдослучайные последовательности

Криптографические приложения предъявляют к генератору псевдослучайных последовательностей более высокие требования по сравнению с другими приложениями. Криптографическая случайность не ограничивается статистической случайностью, хотя и включает ее. Чтобы последовательность была **криптографически безопасной псевдослучайной** последовательностью, она должна обладать следующим свойством:

2. Она непредсказуема. Должно быть очень трудно (с точки зрения применения вычислительных мощностей) предсказать, каким будет следующий случайный бит, даже если полностью известен алгоритм или устройство, генерирующее последовательность, и все предыдущие биты потока.

Криптографически безопасные псевдослучайные последовательности не должны сжиматься..., если вам неизвестен ключ. Ключом обычно является заданное начальное состояние генератора.

Как и любой криптографический алгоритм, генераторы криптографически безопасных псевдослучайных последовательностей представляют собой предмет вскрытия. Так же как криптографический алгоритм, может быть взломан и генератор криптографически безопасных псевдослучайных последовательностей. Создание устойчивых к вскрытию генераторов является основой криптографии.

Настоящие случайные последовательности

Теперь мы вторгаемся в область, принадлежащую философам. Существует ли такая вещь как случайность? Что такое случайная последовательность? Как узнать, что последовательность случайна? Является ли "101110100" более случайной чем "l01010101"? Квантовая механика убеждает нас в том, что в реальном мире существует настоящая случайность. Но как сохранить эту случайность в предопределенном мире компьютерных микросхем и конечных автоматов?

В сторону философию, с нашей точки зрения генератор последовательности **действительно случаен**, если он обладает третьим свойством:

3. Создаваемая им последовательность не может быть уверенно воспроизведена. Если вы запускаете г е-

нератор случайных чисел дважды с одним и тем же входом (по крайней мере, насколько это в челов е-ческих силах), то вы получите две совершенно независимые случайные последовательности.

Выход генератора, удовлетворяющего всем трем приведенным требованиям, будет достаточно хорош для одноразового блокнота, генерации ключа и других криптографических применений, требующих генерации де й-ствительно случайных последовательностей. Трудность в том, чтобы понять, действительно ли последовател ь-ность случайна? Если я повторно зашифрую строку, используя DES и заданный ключ, я получу хороший, вы-глядящий случайным образом результат, вы не сможете сказать, что он не случаен, пока вы не наймете взло м-щика DES из NSA.

Глава 3
Основные протоколы

3.1 Обмен ключами

Общепринятой криптографической техникой является шифрование каждого индивидуального обмена сообщениями отдельным ключом. Такой ключ называется сеансовым, так как он используется для единственного отдельного сеанса обмена информацией. В разделе 8.5 говорится о том, что сеансовые ключи полезны, так как время их существования определяется длительностью сеанса связи. Передача этого общего сеансового ключа в руки обменивающихся информацией представляет собой сложную проблему.

Обмен ключами с помощью симметричной криптографии

Этот протокол предполагает, что пользователи сети, Алиса и Боб, получают секретный ключ от Центра распределения ключей (Key Distribution Center, KDC) [1260] - Трента наших протоколов. Перед началом протокола эти ключи уже должны быть у пользователей. (Протокол игнорирует очень насущную проблему доставки этих секретных ключей, предполагается, что ключи уже у пользователей, и Мэллори не имеет о них никакой информации.)

(1) Алиса обращается к Тренту и запрашивает сеансовый ключ для связи с Бобом.

(2) Трент генерирует случайный сеансовый ключ. Он зашифровывает две копии ключа: одну для Алисы, а другую - для Боба. Затем Трент посылает обе копии Алисе.

(3) Алиса расшифровывает свою копию сеансового ключа.

(4) Алиса посылает Бобу его копию сеансового ключа.

(5) Боб расшифровывает свою копию сеансового ключа.

(6) Алиса и Боб используют этот сеансовый ключ для безопасного обмена информацией.

Этот протокол основан на абсолютной надежности Трента, для роли которого больше подходит заслуживающая доверия компьютерная программа, чем заслуживающий доверия человек. Если Мэллори получит доступ к Тренту, скомпрометированной окажется вся сеть. В его руках окажутся все секретные ключи, выделенные пользователям Трентом, он сможет прочесть все переданные сообщения, которые ему удалось перехватить, и все будущие сообщения. Ему останется только подключиться к линиям связи и подслушивать зашифрованный поток сообщений.

Другой проблемой такой системы является то, что Трент потенциально является ее узким местом. Он должен участвовать в каждом обмене ключами. Если с ним что-то случится, это разрушит всю систему.

Обмен ключами, используя криптографию с открытыми ключами

Базовая смешанная криптосистема обсуждалась в разделе 1.5. Для согласования сеансового ключа Алиса и Боб применяют криптографию с открытыми ключами, а затем используют этот сеансовый ключ для шифрования данных. В некоторых реализациях подписанные ключи Алисы и Боба доступны в некоторой базе данных. Это значительно облегчает протокол, теперь Алиса, даже если Боб о ней никогда не слышал, может безопасно послать Бобу сообщение:

(1) Алиса получает открытый ключ Боба из KDC.

(2) Алиса генерирует случайный сеансовый ключ, зашифровывает его открытым ключом Боба и посылает его Бобу.

(3) Боб расшифровывает сообщение Алисы с помощью своего закрытого ключа.

(4) Алиса и Боб шифруют свой обмен информацией этим сеансовым ключом.

Вскрытие "человек-в-середине"

В то время, как Ева не может сделать ничего лучшего, чем пытаться взломать алгоритм с открытыми ключами или выполнить вскрытие с использованием только шифротекста, у Мэллори гораздо больше возможностей. Он не только может подслушать сообщения Алисы и Боба, но и изменить сообщения, удалить сообщения и создать совершенно новые. Мэллори может выдать себя за Боба, сообщающего что-то Алисе, или за Алису, сообщающую что-то Бобу. Вот как будет выполнено вскрытие:

(1) Алиса посылает Бобу свой открытый ключ. Мэллори перехватывает его и посылает Бобу свой собственный открытый ключ.

(2) Боб посылает Алисе свой открытый ключ. Мэллори перехватывает его и посылает Алисе Бобу собстве нный открытый ключ.

(3) Когда Алиса посылает сообщение Бобу, зашифрованное открытым ключом "Боба", Мэллори перехватыв ает его. Так как сообщение в действительности зашифровано его собственным открытым ключом, он ра сшифровывает его, снова зашифровывает открытым ключом Боба и посылает Бобу.

(4) Когда Боб посылает сообщение Алисе, зашифрованное открытым ключом "Алисы", Мэллори перехват ывает его. Так как сообщение в действительности зашифровано его собственным открытым ключом, он расшифровывает его, снова зашифровывает открытым ключом Алисы и посылает Алисе.

Это вскрытие будет работать, даже если открытые ключи Алисы и Боба хранятся в базе данных . Мэллори может перехватить запрос Алисы к базе данных и подменить открытый ключ Боба своим собственным . То же самое он может сделать и с открытым ключом Алисы . Или, еще лучше, он может исподтишка взломать базу данных и подменить открытые ключи Бобо и Алисы своим . Теперь он может преуспеть, просто дождавшись, пока Алиса и Боб начнут обмениваться сообщениями, и начав перехватывать и изменять эти сообщения .

Такое **вскрытие "человек-в-середине"** работает, так как у Алисы и Боба нет способа проверить, действ ительно ли они общаются именно друг с другом . Если вмешательство Мэллори не приводит к заметным задер жкам в сети, оба корреспондента и не подумают, что кто-то, сидящий между ними, читает всю их секретную по чту.

Протокол "держась за руки"

Протокол "держась за руки", изобретенный Роном Ривестом (Ron Rivest) и Эди Шамиром (Adi Shamir) [1327], предоставляет неплохую возможность избежать вскрытия "человек-в-середине" . Вот как он работает:

(1) Алиса посылает Бобу свой открытый ключ.

(2) Боб посылает Алисе свой открытый ключ.

(3) Алиса зашифровывает свое сообщение открытым ключом Боба. Половину зашифрованного сообщения она отправляет Бобу.

(4) Боб зашифровывает свое сообщение открытым ключом Алисы. Половину зашифрованного сообщения он отправляет Алисе.

(5) Алиса отправляет Бобу вторую половину зашифрованного сообщения.

(6) Боб складывает две части сообщения Алисы и расшифровывает его с помощью своего закрытого ключа. Боб отправляет Алисе вторую половину своего зашифрованного сообщения.

(7) Алиса складывает две части сообщения Боба и расшифровывает его с помощью своего закрытого ключа.

Идея в том, что половина зашифрованного сообщения бесполезна без второй половины, она не может быть дешифрирована. Боб не сможет прочитать ни одной части сообщения Алисы до этапа (6), а Алиса не сможет прочитать ни одной части сообщения Боба до этапа (7). Существует множество способов разбить сообщение на части:

— Если используется блочный алгоритм шифрования, половина каждого блока (например, каждый второй бит) может быть передана в каждой половине сообщения.

— Дешифрирование сообщения может зависеть от вектора инициализации (см. раздел 9.3), который может быть передан во второй части сообщения.

— Первая половина сообщения может быть однонаправленной хэш -функцией шифрованного сообщения (см. раздел 2.4), а во вторая половина - собственно шифрованным сообщением.

Чтобы понять, как такой протокол помешает Мэллори, давайте рассмотрим его попытку нарушить протокол . Как и раньше, он может подменить открытые ключи Алисы и Боба своим на этапах (1) и (2) . Но теперь, перехватив половину сообщения Алисы на этапе (3), он не сможет расшифровать ее своим закрытым ключом и сн ова зашифровать открытым ключом Боба . Он может создать совершенно новое сообщение и отправить половину его Бобу. Перехватив половину сообщения Боба Алисе на этапе (4), Мэллори столкнется с этой же проблемой . Он не сможет расшифровать ее своим закрытым ключом и снова зашифровать открытым ключом Алисы . Ему придется создать совершенно новое сообщение и отправить половину его Алисе . К тому времени, когда он пе рехватит вторые половины настоящих сообщений на этапах (5) и (6), подменять созданные им новые сообщения будет слишком поздно . Обмен данными между Алисой и Бобом изменится радикально .

Мэллори может попытаться избежать такого результата. Если он достаточно хорошо знает обоих корреспо ндентов, чтобы сымитировать их при обмене данными , они могут никогда не заметить подмены . Но все-таки это сложнее, чем просто сидеть между корреспондентами, перехватывая и читая их сообщения .

Обмен ключами с помощью цифровых подписей

Использование цифровой подписи в протоколе обмена сеансовым ключом также позволяет избежать вскр ытия "человек-в-середине". Трент подписывает открытые ключи Алисы и Боба. Подписанные ключи включают подписанное заверение подлинности. Получив ключи, и Алиса, и Боб проверяют подпись Трента. Теперь они уверены, что присланный открытый ключ принадлежит именно указанному корреспонденту. Затем выполняется протокол обмена ключами.

Мэллори сталкивается с серьезными проблемами. Он не может выдать себя за Алису или Боба, ведь он не знает их закрытых ключей. Он не может подменить их открытые ключи своим, потому что при подписи его ключа Трент указал, что это ключ Мэллори. Все, что ему остается - это прослушивать зашифрованный поток сообщений или испортить линии связи, мешая обмену информации Алисы и Боба.

Трент выступает участником этого протокола, но риск компрометации KDC меньше, чем в первом протоколе. Если Мэллори компрометирует Трента (взламывает KDC), он получает только закрытый ключ Тре нта. Этот ключ позволит ему только подписывать новые ключи, а не расшифровывать сеансовые ключи ил ч итать произвольный поток сообщений. Для чтения сообщений Мэллори придется выдать себя за пользователя сети и обманывать честных пользователей, шифруя сообщения своим поддельным открытым ключом.

Мэллори может предпринять такое вскрытие. Используя закрытый ключ Трента, он может создать поддел ьные подписанные ключи, чтобы обмануть Алису и Боба. Затем он может либо подменить этими ключами н астоящие ключи в базе данных, либо перехватывать запросы пользователей к базе данных и посылать в ответ поддельные ключи. Это позволит ему осуществить вскрытие "человек-в-середине" и читать сообщения польз ователей.

Такое вскрытие будет работать, но помните, что для этого Мэллори должен уметь перехватывать и изменять сообщения. В ряде сетей это намного сложнее, чем просто пассивно сидеть, просматривая сообщения в сети по мере их поступления. В широковещательных каналах, таких как радиосеть, почти невозможно подменить одно сообщение другим - хотя можно забить всю сеть. В компьютерных сетях это менее сложно и, кажется, с каждым днем становится проще и проще. Обратите внимание на подмену IP-адреса, вскрытие маршрутизатора и т.п. Активное вскрытие не обязательно означает, что кто-то засовывает зонд в люк, да и выполнять их теперь могут не только правительственные агентства.

Передача ключей и сообщений

Алисе и Бобу не обязательно выполнять протокол обмена ключами перед обменом сообщениями. В этом протоколе Алиса отправляет Бобу сообщение без предварительного протокола обмена ключами:

(1) Алиса генерирует случайный сеансовый ключ, *K*, и зашифровывает *M* этим ключом.

$E_K(M)$

(2) Алиса получает открытый ключ Боба из базы данных.

(3) Алиса шифрует *K* открытым ключом Боба.

$E_B(K)$

(4) Алиса посылает Бобу шифрованные сообщение и сеансовый ключ.

$E_K(M)$, $E_B(K)$

Для дополнительной защиты от вскрытия "человек-в-середине" Алиса подписывает передачу.

(5) Боб расшифровывает сеансовый ключ Алисы, *K*, используя свой закрытый ключ.

(6) Боб, используя сеансовый ключ, расшифровывает сообщение Алисы.

Подобная смешанная система и употребляется чаще всего в системах связи. Ее можно соединить с цифровыми подписями, метками времени и другими протоколами обеспечения безопасн ости.

Широковещательная рассылка ключей и сообщений

Не существует причин, запрещающих Алисе посылать шифрованное сообщение нескольким людям. В следующем примере Алиса посылает шифрованное сообщение Бобу, Кэрол и Дэйву:

(1) Алиса генерирует случайный сеансовый ключ, *K*, и зашифровывает *M* этим ключом.

$E_K(M)$

(2) Алиса получает из базы данных открытые ключи Боба, Кэрол и Дэйва.

(3) Алиса шифрует *K* открытыми ключами Боба, Кэрол и Дэйва.

$E_B(K)$, $E_C(K)$, $E_D(K)$

(4) Алиса широковещательно посылает шифрованное сообщение и все шифрованные ключи своим корреспондентам.

$E_K(M)$, $E_B(K)$, $E_C(K)$, $E_D(K)$

(5) Только Боб, Кэрол и Дэйв могут, каждый при помощи своего закрытого ключа, расшифровать ключ K.

(6) Только Боб, Кэрол и Дэйв могут расшифровать сообщение Алисы, используя K.

Этот протокол может быть реализован для сетей электронной почты. Центральный сервер может отправить сообщение Алисы Бобу, Кэрол и Дэйву вместе с конкретным шифрованным ключом. Сервер не должен быть надежным и безопасным, так как он не может расшифровать ни одно из сообщений.

3.2 Удостоверение подлинности

Когда Алиса подключается к главному компьютеру (или к автоматическому, или к телефонной банковской системе, или к какому-нибудь другому терминалу), как главный компьютер узнает, кто она? Откуда главный компьютер узнает, что это не Ева, пытающаяся выдать себя за Алису? Обычно эта проблема решается с помощью паролей. Алиса вводит свой пароль, и главный компьютер проверяет его правильность. Таким образом, и Алисе, и главному компьютеру известна некоторая секретная информация, которую главный компьютер запрашивает всякий раз, когда Алиса пытается подключится.

Удостоверение подлинности с помощью однонаправленных функций

Роджер Needхэм (Roger Needham) и Майк Гай (Mike Guy) показали, что главному компьютеру не нужно знать сами пароли, вполне достаточно, чтобы главный компьютер мог отличать правильные пароли от неправильных. Этого легко достичь с помощью однонаправленных функций [1599, 526,1274, 1121]. При этом на главном компьютере хранятся значения однонаправленных функций паролей, а не сами пароли.

(1) Алиса посылает главному компьютеру свой пароль.

(2) Главный компьютер вычисляет однонаправленную функцию пароля.

(3) Главный компьютер сравнивает полученное значение с хранящимся.

Раз главный компьютер больше не хранит таблицу правильных паролей всех пользователей, снижается угроза того, что кто-то проникнет в главный компьютер и выкрадет таблицу паролей. Список паролей, обработанный однонаправленной функцией, бесполезен, так как однонаправленную функцию не удастся инвертировать для получения паролей.

Вскрытия с помощью словаря и "соль"

Файл паролей, зашифрованных однонаправленной функцией, тем не менее, уязвим. Имея запас времени, Мэллори может составить список из миллиона наиболее часто встречающихся паролей. Он обработает весь миллион однонаправленной функцией и сохранит результат. Если каждый пароль состоит из восьми байт, размер получившегося файла не превысит 8 Мбайт, и этот файл может быть размещен всего на нескольких на дискетах. Теперь Мэллори добывает шифрованный файл паролей. Он сравнивает этот файл со своим файлом шифрованных возможных паролей и ищет совпадения.

Это **вскрытие с помощью словаря** может быть удивительно успешным (см. раздел 8.1). "**Соль**" - это способ затруднить его. "Соль" представляет собой случайную строку, добавляемую к паролям перед обработкой их однонаправленной функцией. Затем в базе данных главного компьютера сохраняются и значение "соли", и результат однонаправленной функции. Использование достаточно большого числа возможных значений "соли" практически устраняет возможность вскрытия с помощью словаря, так как Мэллори придется вычислять значение однонаправленной хэш-функции для каждого возможного значения "соли". Это простейший пример использование вектора инициализации (см. раздел 9.3).

Идея состоит в том, чтобы заставить Мэллори выполнить пробное шифрование каждого пароля из его словаря при каждой попытке узнать чей-то чужой пароль вместо одноразовой обработки всех возможных паролей.

Для этого нужно много "соли". Большинство UNIX-систем используют для "соли" 12 бит. Несмотря на это Дэниел Кляйн (Daniel Klein) написал программу разгадывания паролей, которая в некоторых системах за неделю часто вскрывала 40 процентов паролей [847,848] (см. раздел 8.1). Дэвид Фельдмайер (David Feldmeier) и Филип Кан (Philip Karn) составили список из 732000 наиболее часто используемых паролей, присоединив к каждому из них 4096 возможных значений "соли". По их оценкам 30 процентов паролей на любом главном компьютере могут быть взломаны с помощью этого списка [561].

"Соль" не является панацеей, увеличение числа бит "соли" не решит всех проблем. "Соль" предохраняет

только от самых обычных вскрытий файла паролей с использованием словаря, а не от согласованной атаке о дного пароля. Она защищает людей, использующих один и тот же пароль на различных машинах, но не делает лучше плохо выбранный пароль.

SKEY

SKEY - это программа удостоверения подлинности, обеспечивающая безопасность с помощью однонапра вленной функции. Это легко объяснить.

Регистрируясь в системе, Алиса задает случайное число, R. Компьютер вычисляет $f(R)$, $f(f(R))$, $f(f(f(R)))$, и так далее, около сотни раз. Обозначим эти значения как x_1, x_2, x_3, ..., x_{100}. Компьютер печатает список этих чисел, и Алиса прячет его в безопасное место. Компьютер также открытым текстом ставит в базе данных подключения к системе в соответствие Алисе число x_{101}.

Подключаясь впервые, Алиса вводит свое имя и x_{100}. Компьютер рассчитывает $f(x_{100})$ и сравнивает его с x_{101}, если значения совпадают, права Алисы подтверждаются. Затем компьютер заменяет в базе данных x_{101} на x_{100}. Алиса вычеркивает x_{100} из своего списка.

Алиса, при каждом подключении к системе, вводит последнее невычеркнутое число из своего списка: x_i. Компьютер рассчитывает $f(x_i)$ и сравнивает его с x_{i+1}, хранившемся в базе данных. Так как каждый номер используется только один раз, Ева не сможет добыть никакой полезной информации. Аналогично, база данных бесполезна и для взломщика. Конечно же, как только список Алисы исчерпается ей придется перерегистрир оваться в системе.

Удостоверение подлинности с помощью криптографии с открытыми ключами

Даже с использованием "соли" у первого протокола есть серьезные проблемы с безопасностью. Когда Алиса посылает свой пароль главному компьютеру, любой, у кого есть доступ пути передачи ее данных, может пр очесть пароль. Она может получить доступ к своему главному компьютеру посредством запутанного пути пер едачи информации, проложив его через четырех промышленных конкурентов, три других страны и два перед овых университета. Ева может находиться в любой из этих точек, подслушивая передаваемую Алисой послед овательность. Если у Евы есть доступ к оперативной памяти главного компьютера, она сможет подсмотреть п ароль до того, как главный компьютер сможет его хэшировать.

Криптография с открытыми ключами может решить эту проблему. Главный компьютер хранит файл откр ытых ключей всех пользователей, а все пользователи хранят свои закрытые ключи. Вот как выглядит упрощенная попытка организовать протокол подключения:

(1) Главный компьютер посылает Алисе случайную строку.

(2) Алиса шифрует эту строку своим закрытым ключом и посылает ее обратно главному компьютеру вместе со своим именем.

(3) Главный компьютер находит в базе данных открытый ключ Алисы и дешифрирует сообщение, используя этот открытый ключ.

(4) Если отправленная сначала и расшифрованная строки совпадают, главный компьютер предоставляет Алисе доступ к системе

Никто другой не может воспользоваться закрытым ключом Алисы, следовательно никто не сможет выдать себя за нее. Что более важно, Алиса никогда не посылает на компьютер свой закрытый ключ. Ева, подслушивая взаимодействие, не получит никаких сведений, которые позволили бы ей вычислить закрытый ключ Алисы и выдать себя за нее.

Закрытый ключ должен быть достаточно длинным и не должен быть мнемоническим. Он будет автоматич ески обрабатываться аппаратурой пользователя или программным обеспечением связи. Это требует использования "умного" терминала, которому Алиса доверяет, но не главный компьютер, ни линии связи не обязаны быть безопасными.

Глупо шифровать произвольные строки - не только посланные подозрительным авторов, но и вообще любые. Иначе может быть использовано схема вскрытия, обсуждаемая в разделе 19.3. Безопасные идентификационные протоколы имеют следующую, более сложную форму:

(1) Алиса выполняет вычисление, основанное на некоторых случайных числах и своем закрытом ключе, и п осылает результат на главный компьютер.

(2) Главный компьютер посылает другое случайное число.

(3) Алиса выполняет некоторое вычисление, основанное на случайных числах (как созданном ею, так и пол ученном от главного компьютера) и своем закрытом ключе, и посылает результат на главный компьютер.

(4) Главный компьютер выполняет некоторое вычисление для различных чисел, полученных от Алисы, и ее открытого ключа, проверяя, что ей известен ее закрытый ключ.

(5) Если проверка завершается успешно, личность Алисы подтверждается.

Если Алисы доверяет главному компьютеру не в большей степени, чем тот доверяет Алисе, то она должна потребовать подтверждения подлинности главного компьютера аналогичным образом .

Этап (1) может показаться ненужным и запутанным, но он необходим для защиты протокола от вскрытия . Различные протоколы и алгоритмы подтверждения подлинности математически подробно описываются в ра зделах 21.1 и 21.2. См. также [935].

Обоюдное удостоверение подлинности с использованием протокола "держась за руки"

Пусть два пользователя, Алиса и Боб, хотят проверить подлинность друг друга . У каждого из них есть па роль, известный другому пользователю: P_A у Алисы и P_B у Боба. Вот как выглядит протокол, который *не* будет работать:

(1) Алиса и Боб обмениваются открытыми ключами.

(2) Алиса шифрует P_A открытым ключом Боба и посылает его ему.

(3) Боб шифрует P_B открытым ключом Алисы и посылает его ей.

(4) Алиса расшифровывает полученное на этапе (3) и подтверждает правильность пароля.

(5) Боб расшифровывает полученное на этапе (2) и подтверждает правильность пароля.

Мэллори может предпринять успешное вскрытие "человек-в-середине" (см. раздел 3.1):

(1) Алиса и Боб обмениваются открытыми ключами. Мэллори перехватывает оба сообщения, и посылает обоим корреспондентам свой собственный открытый ключ, подменив им их ключи.

(2) Алиса шифрует P_A открытым ключом "Боба" и посылает его ему. Мэллори перехватывает сообщение, расшифровывает P_A с помощью своего закрытого ключа, снова шифрует P_A открытым ключом Боба и по сылает его ему.

(3) Боб шифрует P_B открытым ключом "Алисы" и посылает его ей. Мэллори перехватывает сообщение, ра с шифровывает P_B с помощью своего закрытого ключа, снова шифрует P_B открытым ключом Алисы и по сылает его ей.

(4) Алиса расшифровывает P_B и подтверждает его правильность.

(5) Боб расшифровывает P_A и подтверждает его правильность.

Для Алисы и Боба ничего не изменилось. Однако, Мэллори знает и P_A, и P_B. Дональд Дэвис (Donald Davies) и Вильям Прайс (William Price) описывают, как протокол "держась-за-руки" (см. раздел 3.1) противодействует такому вскрытию [435]. Стив Белловин (Steve Bellovin) и Майкл Мерритт (Michael Merritt) рассматривают спо собы вскрытия этого протокола в [110]. Если Алиса - это пользователь, а Боб - хост-компьютер , Мэллори может предпочесть быть Бобом, выполнить первые этапа протокола с Алисой и разорвать соединение . Симулирование шума на линии или сетевого отказа потребует от Мэллори настоящего артистизма, но в результате Мэллори получит пароль Алисы. Затем он сможет соединиться с Бобом и завершить протокол, получая и пароль Боба .

Протокол можно изменить так, чтобы Боб передавал свой пароль перед Алисой в предположении, что п ароль пользователя более важен чем пароль главного компьютера . Это приведет к усложнению способа вскр ытия, также описанного в [110].

SKID

SKID2 и SKID3 - это симметричные криптографические протоколы идентификации, разработанные для пр оекта RACE RIPE [1305] (см. раздел 25.7). Они используют MAC (см. раздел 2.4) для обеспечения безопасности и предполагают, что Алиса и Боб используют общий секретный ключ , K. SKID2 позволяет Бобу доказать свою подлинность Алисе. Вот этот протокол:

(1) Алиса выбирает случайное число, R_A. (Документами RIPE определяется 64-битовое число). Она посылает это число Бобу.

(2) Боб выбирает случайное число, R_B. (Документами RIPE определяется 64-битовое число). Он посылает Алисе.

R_B, $H_K(R_A, R_B, B)$

H_K - это MAC. (В документах RIPE предлагается функция RIPE-MAC, см. раздел 18.4.) B - это имя Боба.

(3) Алиса рассчитывает $H_K(R_A, R_B, B)$ и сравнивает результат со значением, полученным от Боба. Если результаты совпадают, Алиса убеждается в том, что она соединилась именно с Бобом.

SKID3 обеспечивает совместную проверку подлинности Алисой и Бобом. Этапы (1) - (3) совпадают с протоколом SKID2, а затем выполняются следующие действия:

(4) Алиса посылает Бобу:

$H_K(R_B, A)$

A - это имя Алисы.

(5) Боб рассчитывает $H_K(R_B, A)$ и сравнивает результат со значением, полученным от Алисы. Если результаты совпадают, Боб убеждается в том, что она соединилась именно с Алисой.

Этот протокол неустойчив к вскрытию "человек-в-середине". В общем случае, вскрытие "человек-в-середине" может угрожать любому протоколу, в который не входит какой-нибудь секрет.

Удостоверение подлинности сообщений

Когда Боб получает сообщение от Алисы, как ему узнать, что это сообщение подлинно? Если Алиса подписала свое сообщение, то все просто. Цифровая подпись Алисы достаточна, чтобы подтвердить кому угодно подлинность ее сообщения.

Некоторую проверку подлинности предоставляют и симметричные алгоритмы. Когда Боб получает сообщение от Алисы, шифрованное их общим ключом, он знает, что это сообщение от Алисы. Никто больше не знает их ключа. Однако, у Боба нет возможности убедить в этом кого-то еще. Боб не может показать сообщение Тренту и убедить его, что оно отправлено Алисой. Трент может сделать вывод, что сообщение отправлено или Алисой, или Бобом (так как их секретный ключ никому больше не принадлежит), но у него нет способа определить, кто же конкретно автор сообщения.

Если сообщение не шифровано, Алиса может также использовать MAC. Это также убедит Боба в подлинности сообщения, но возникнут те же проблемы, что и для решений симметричной криптографии.

3.3 Удостоверение подлинности и обмен ключами

Эти протоколы объединяют удостоверение подлинности и обмен ключами для решения основной компьютерной проблемы: Алиса и Боб хотят безопасно обмениваться сообщениями, находясь на различных концах сети. Как могут Алиса и Боб обменяться секретным ключом, при этом сохраняя уверенность, что они обмениваются сообщениями друг с другом, а не с Мэллори? В большинстве протоколов предполагается, что каждому пользователю Трент выделяет отдельный секретный ключ, и перед началом работы протокола все ключи уже находятся у пользователей. Символы, используемые в этих протоколах, сведены в 2-й.

Табл. 3-1.
Символы, используемые в протоколах удостоверения подлинности и обмена ключами

A	Имя Алисы
B	Имя Боба
E_A	Шифрование ключом, выделенном Трентом Алисе
E_B	Шифрование ключом, выделенном Трентом Бобу
I	Порядковый номер
K	Случайное сеансовое число
L	Время жизни
T_A, T_B	Метки времени
R_A, R_B	Случайные числа, выбранные Алисой и Бобом, соответственно

Лягушка с широким ртом

Протокол "Лягушка с широким ртом" (Wide-Mouth Frog) [283,284], возможно, является простейшим симметричным протоколом управления ключами, в котором используется заслуживающий доверия сервер. Алиса и Боб делят свой секретный ключ с Трентом. Эти ключи используются только для распределения ключей, а не для шифрования пользовательских сообщений. Вот как, используя два сообщения, Алиса передает Бобу сеансовый

ключ:

(1) Алиса объединяет метку времени, имя Боба и случайный сеансовый ключ, затем шифрует созданное с о-общение общим с Трентом ключом и посылает его Тренту вместе со своим именем.

$A, E_A(T_A, B, K)$

(2) Трент расшифровывает сообщение от Алисы. Затем он добавляет новую метку времени, имя Алисы и сл у-чайный сеансовый ключ, шифрует полученное сообщение общим с Бобом ключом. Трент посылает Бобу:

$E_B(T_B, B, K)$

Наибольшим допущением, сделанным в этом протоколе, является то, что Алиса обладает достаточной ко м-петентностью для генерации хороших сеансовых ключей . Вспомните, что случайные числа генерировать совсем не просто, для этого может потребоваться кто-нибудь понадежнее Алисы .

Yahalom

В этом протоколе Алисы и Боб делят с Трентом секретный ключ [283,284].

(1) Алиса объединяет свое имя и случайное число, и отправляет созданное сообщение Бобу.

A, R_A

(2) Боб объединяет имя Алисы, ее случайное число, свое случайное число, шифрует созданное сообщение о б-щим с Трентом ключом и посылает его Тренту, добавляя свое имя:

$B, E_B(A, R_A, R_B)$

(3) Трент создает два сообщения. Первое включает имя Боба, случайный сеансовый ключ, случайные числа Боба и Алисы и шифруется ключом, общим для Трента и Алисы. Второе состоит из имени Алисы, сл у-чайного сеансового ключа и шифруется ключом, общим для Трента и Боба. Трент посылает оба сообщ е-ния Алисе:

$E_A(B, K, R_A, R_B), E_B(A, K)$

(4) Алиса расшифровывает первое сообщение, извлекает K и убеждается, что R_A совпадает со значением, от-правленным на этапе (1). Алиса посылает Бобу два сообщения. Одним является сообщение Трента, з а-шифрованное ключом Боба. Второе - это R_B, зашифрованное сеансовым ключом.

$E_B(A, K), E_K(R_B),$

(5) Боб расшифровывает первое сообщение, извлекает K и убеждается, что R_B совпадает с отправленным на этапе (2).

В результате Алиса и Боб убеждены, что они общаются именно друг с другом, а не с третьей стороной . Нововведение состоит в том, что именно Боб первым обращается к Тренту, который только посылает одно соо б-щение Алисе.

Needham-Schroeder

В этом протоколе, изобретенном Роджером Неедхэмом (Roger Needham) и Майклом Шредером (Michael Schroeder) [1159], также используются симметричная криптография и Трент .

(1) Алиса посылает Тренту сообщение, содержащее ее имя, имя Боба и случайное число.

A, B, R_A

(2) Трент генерирует случайный сеансовый ключ. Он шифрует сообщение, содержащее случайный сеансовый ключ и имя Алисы, секретным ключом, общим для него и Боба. Затем он шифрует случайное число Ал и-сы, имя Боба, ключ, и шифрованное сообщение секретным ключом, общим для него и Алисы. Наконец, он отправляет шифрованное сообщение Алисе:

$E_A(R_A, B, K, E_B(K, A))$

(3) Алиса расшифровывает сообщение и извлекает K. Она убеждается, что R_A совпадает со значением, от-правленным Тренту на этапе (1). Затем она посылает Бобу сообщение, зашифрованное Трентом ключом Боба.

$E_B(K, A)$

(4) Боб расшифровывает сообщение и извлекает K. Затем он генерирует другое случайное число, R_B. Он шифрует это число ключом K и отправляет его Алисе.

$E_K(R_B)$

(5) Алиса расшифровывает сообщение с помощью ключа K. Она создает число R_B-1 и шифрует это число ключом K. Затем она посылает это сообщение обратно Бобу.

$E_K(R_B\text{-}1)$

(6) Боб расшифровывает сообщение с помощью ключа K и проверяет значение R_B-1.

Вся эта возня с R_A, R_B, и R_B-1 служит для предотвращения **вскрытия с повторной передачей**. При таком способе вскрытия Мэллори может записать старые сообщения и впоследствии использовать их при попытке взломать протокол. Присутствие R_A на этапе (2) убеждает Алису, что сообщение Трента достоверно и не является повторной передачей отклика, использованного при одном из предыдущих применений протокола. Когда Алиса успешно расшифрует R_B и передает Бобу R_B-1 на этапе (5), Боб убеждается, что сообщения Алисы не является повторной передачей сообщений, использованных при одном из предыдущих применений протокола.

Главной прорехой этого протокола является важность использованных сеансовых ключей. Если Мэллори получит доступ к старому K, он сможет предпринять успешное вскрытие [461]. Ему нужно только записать сообщения Алисы Бобу на этапе (3). Тогда, имея K, он может выдать себя за Алису:

(1) Мэллори посылает Бобу следующее сообщение:

$E_B(K, A)$

(2) Боб извлекает K, генерирует R_B и отправляет Алисе:

$E_K(R_B)$

(3) Мэллори перехватывает сообщение, расшифровывает его с помощью ключа K и посылает Бобу:

$E_K(R_B\text{-}1)$

(4) Боб убеждается, что сообщение "Алисы" состоит из R_B-1.

Теперь Мэллори убедил Боб, что он и есть "Алиса". Более защищенный протокол, использующий метки времени, может противостоять этому вскрытию [461,456]. Метки времени добавляются к сообщению Трента на этапе (2) и шифруются ключом Боба: $E_B(K, A, T)$. Метки времени требуют надежной и точной системы единого времени, что само по себе нетривиальная проблема.

Если ключ, общий для Трента и Алисы будет скомпрометирован, последствия будут драматичны. Мэллори сможет использовать его, для получения сеансовых ключей для обмена сообщениями с Бобом (или с кем-нибудь еще). Даже хуже, Мэллори продолжать подобные действия даже после замены ключа Алисы [90].

Неедхэм и Шредер пытались исправить эти проблемы в модифицированной версии своего протокола [1160]. Их новый протокол по существу совпадает с протоколом Отуэя- Риса (Otway-Rees), опубликованном в том же выпуске того же журнала.

Otway-Rees

Этот протокол также использует симметричную криптографию [1224].

(1) Алиса создает сообщение, состоящее из порядкового номера, ее имени, имени Боба и случайного числа. Сообщение шифруется ключом, общим для Алисы и Трента. Она посылает это сообщение Бобу вместе с порядковым номером, ее и его именами:

$I, A, B, E_A(R_A, I, A, B)$

(2) Боб создает сообщение, состоящее из нового случайного числа, порядкового номера, имени Алисы и имени Боба. Сообщение шифруется ключом, общим для Алисы и Боба. Он посылает это сообщение Тренту вместе шифрованным сообщением Алисы, порядковым номером, ее и его именами:

$I, A, B, E_A(R_A, I, A, B), E_B(R_B, I, A, B)$

(3) Трент генерирует случайный сеансовый ключ. Затем он создает два сообщения. Одно, состоящее из случайного числа Алисы и сеансового ключа, шифруется ключом, общим для него и Алисы. Другое, состоящее из случайного числа Боба и сеансового ключа, шифруется ключом, общим для него и Боба. Он отправляет два этих сообщения вместе с порядковым номером Бобу:

$I, E_A(R_A, K), E_B(R_B, K)$

(4) Боб отправляет Алисе сообщение, шифрованное ее ключом, и порядковый номер:

$I, E_A(R_A, K)$

(5) Алиса расшифровывает сообщение, получая свои ключ и случайное число. Алиса убеждается, что при выполнении протокола они не изменились Боб отправляет Алисе сообщение, шифрованное ее ключом, и по-

рядковый номер.

Если все случайные числа правильны, а порядковый номер не изменился при выполнении протокола, Алиса и Боб убеждаются в подлинности друг друга и получают секретный ключ для обмена сообщениями .

Kerberos

Kerberos - вариант протокола Needham-Schroeder - подробно обсуждается в разделе 24.5. В базовом протоколе Kerberos Version 5 у Алисы и Боба общие ключи с Трентом. Алиса хочет генерировать сеансовый ключ для сеанса связи с Бобом.

(1) Алиса посылает Тренту сообщение со своим именем и именем Боба:

A, B

(2) Трент создает сообщение, состоящее из метки времени, время жизни, L, случайного сеансового ключа и имени Алисы. Он шифрует сообщение ключом, общим для него и Боба. Затем он объединяет метку времени, время жизни, сеансовый ключ, имя Боба, и шифрует полученное сообщение ключом, общим для него и Алисы. Оба шифрованных сообщения он отправляет Алисе.

$E_A(T, L, K, B), E_B(T, L, K, A)$

(3) Алиса создает сообщение, состоящее из ее имени и метки времени, шифрует его ключом K и отправляет Бобу. Алиса также посылает Бобу сообщение от Трента, шифрованное ключом Боба:

$E_A(A, T), E_B(T, L, K, A)$

(4) Боб создает сообщение, состоящее из метки времени плюс единица, шифрует его ключом K и отправляет Алисе:

$E_K(T+1)$

Этот протокол работает, но только если часы каждого пользователя синхронизированы с часами Трента . На практике эффект достигается синхронизацией с надежным сервером времени с точностью в несколько минут и обнаружением повторной передачи в течение определенного интервала времени .

Neuman-Stubblebine

Из-за недостатков системы или саботажа синхронизация часов может быть нарушена . Если часы сбиваются, против большинства протоколов может быть использован определенный способ вскрытия [644]. Если часы отправителя опережают часы получателя, Мэллори может перехватить сообщение отправителя и повторно использовать его позднее, когда метка времени станет текущей в месте нахождения получателя . Этот способ, называющийся вскрытием с **подавлением повторной передачи**, может привести к неприятным последствиям.

Этот протокол, впервые опубликованный в [820] и исправлен в [1162], пытается противостоять вскрытию с подавлением повторной передачи. Этот отличный протокол является улучшением Yahalom.

(1) Алиса объединяет свое имя и случайное число, и отправляет созданное сообщение Бобу.

A, R_A

(2) Боб объединяет имя Алисы, ее случайное число и метку времени, шифрует созданное сообщение общим с Трентом ключом и посылает его Тренту, добавляя свое имя и новое случайное число:

$B, R_B, E_B(A, R_A, T_B)$

(3) Трент генерирует случайный сеансовый ключ. Затем он создает два сообщения. Первое включает имя Боба, случайное число Алисы, случайный сеансовый ключ, метку времени и шифруется ключом, общим для Трента и Алисы. Второе состоит из имени Алисы, сеансового ключа, метки времени и шифруется ключом, общим для Трента и Боба. Трент посылает оба сообщения Алисе вместе со случайным числом Боба:

$E_A(B, R_A, K, T_B), E_B(A, K, T_B), R_B$

(4) Алиса расшифровывает сообщение, зашифрованное ее ключом, извлекает K и убеждается, что R_A совпадает со значением, отправленным на этапе (1). Алиса посылает Бобу два сообщения. Одним является сообщение Трента, зашифрованное ключом Боба. Второе - это R_B, зашифрованное сеансовым ключом.

$E_B(A, K), E_K(R_B),$

(5) Боб расшифровывает сообщение, зашифрованное его ключом, извлекает K и убеждается, что значения T_B и R_B те же, что и отправленные на этапе (2).

Если оба случайных числа и метка времени совпадают, Алиса и Боб убеждаются в подлинности дуг друга и получают секретный ключ. Синхронизация часов не требуется, так как метка времени определяется только по

часам Боба, и только Боб проверяет созданную им метку времени .

У этого протокола есть еще одно полезное свойство - Алиса может использовать полученное от Трента с о-общение для последующей проверки подлинности Боба в пределах некоторого времени . Предположим, что Алиса и Боб выполнили приведенный выше протокол, провели и завершили сеанс связи . Алиса и Боб могут повторно проверить подлинность друг друга, не обращаясь к Тренту .

(1) Алиса посылает Бобу сообщение, присланное ей Трентом на этапе (3) и новое случайное число.

$E_B(A, K, T_B), R'_A$

(2) Боб посылает Алисе другое новое случайное число и случайное число, присланное Алисой, шифруя их с е-ансовым ключом связи.

$R'_B, E_K(R'_A)$

(3) Алиса посылает Бобу его новое случайное число, шифруя его сеансовым ключом связи.

$E_K(R'_B)$

Новые случайные числа защищают от вскрытия с повторно передачей .

DASS

Протоколы Распределенной служба безопасности и проверки подлинности (Distributed Authentication Sec u-rity Service, DASS), созданные в Digital Equipment Corporation, также обеспечивают обоюдную проверку по д-линности и обмен ключами [604, 1519, 1518]. В отличие от предыдущего протокола DASS использует как крип-тографию с открытыми ключами, так и симметричную криптографию . И у Алисы, и у Боба есть свой закрытый ключ. Трент подписывает копии их открытых кл ючей.

(1) Алиса посылает Тренту сообщение, состоящее из имени Боба.

B

(2) Трент посылает Алисе открытый ключ Боба, K_B, подписанный закрытым ключом Трента, T. Подписанное сообщение содержит имя Боба.

$S_T(B, K_B)$

(3) Алиса проверяет подпись Трента, убеждаясь, что она действительно получила открытый ключ Боба. Она генерирует случайный сеансовый ключ, K, и случайную пару ключей открытый/закрытый, K_p. Она шиф-рует метку времени ключом K, а затем подписывает время жизни, L, свое имя и своим закрытым ключом, K_A. Наконец, она зашифровывает K открытым ключом Боба и подписывает его с помощью K_p. Все это она отправляет Бобу.

$E_K(T_A), S_{K_A}(L, A, K_p), S_{K_p}(E_{K_B}(K))$

(4) Боб посылает Тренту (это может быть другой Трент) сообщение, состоящее из имени Алисы.

A

(5) Трент посылает Бобу открытый ключ Алисы, K_A, подписанный закрытым ключом Трента. Подписанное сообщение содержит имя Алисы.

$S_T(A, K_A)$

(6) Боб проверяет подпись Трента, убеждаясь, что он действительно получила открытый ключ Алисы. Затем он проверяет подпись Алисы и извлекает K_p. Боб использует свой закрытый ключ, извлекая K. Затем он расшифровывает T_A, проверяя, что это сообщение - текущее.

(7) Если требуется обоюдная проверка подлинности, Боб шифрует новую метку времени ключом K и посыла-ет ее Алисе.

$E_K(T_B)$

(8) Алиса расшифровывает T_B ключом K, проверяя, что это сообщение - текущее.

SPX, продукт DEC, основан на DASS. Дополнительную информацию можно найти в [34].

Denning-Sacco

В этом протоколе также используется криптография с открытыми ключами [461]. Трент ведет базу данных, хранящую открытые ключи всех пользователей .

(1) Алиса посылает Тренту сообщение, состоящее из ее имени и имени Боба.

A, B

(2) Трент посылает Алисе открытый ключ Боба, K_B, подписанный закрытым ключом Трента, T. Трент также посылает Алисе ее собственный открытый ключ, K_A, подписанный закрытым ключом Трента.

$S_T(B, K_B), S_T(A, K_A)$

(3) Алиса посылает Бобу случайный сеансовый ключ и метку времени, подписав их своим закрытым ключом и зашифровав открытым ключом Боба, вместе с обоими подписанными ключами.

$E_B(S_A(K, T_A)), S_T(A, K_A), S_T(B, K_B)$

(4) Боб расшифровывает сообщение Алисы с помощью своего закрытого ключа и проверяет подпись Алисы с помощью ее открытого ключа. Он также убеждается, что метка времени правильна.

С этого момента Алиса и Боб получили K и могут провести безопасный сеанс связи. Это выгляди красиво, но есть одна тонкость - выполнив протокол с Алисой, Боб сможет выдать себя за Алису [5]. Смотрите:

(1) Боб посылает Тренту свое имя и имя Кэрол.

B, C

(2) Трент посылает Бобу подписанные открытые ключи Боба и Кэрол.

$S_T(B, K_B), S_T(C, K_C)$

(3) Боб посылает Кэрол подписанный случайный сеансовый ключ и метку времени, ранее полученные от Алисы, зашифровав их открытым ключом Кэрол, вместе с подтверждением Алисы и подтверждением К эрол.

$E_C(S_A(K, T_A)), S_T(A, K_A), S_T(C, K_C)$

(4) Кэрол расшифровывает сообщение Алисы с помощью своего закрытого ключа и проверяет подпись Ал исы с помощью ее открытого ключа. Он также убеждается, что метка времени правильна.

Теперь Кэрол считает, что она соединилась с Алисой, Боб успешно одурачил ее . Действительно, Боб сможет дурачить любого пользователя сети, пока не закончится срок действия метки времени . Но это легко можно исправить. Просто вставьте имена в шифрованное сообщение на этапе (3) :

$E_B(S_A(A, B, K, T_A)), S_T(A, K_A), S_T(B, K_B)$

Теперь Боб не сможет повторно послать Кэрол старое сообщение, потому что оно явно предназначено для сеанса связи между Алисой и Бобом .

Woo-Lam

В этом протоколе также используется криптография с открытыми ключами [1610, 1611]:

(1) Алиса посылает Тренту сообщение, состоящее из ее имени и имени Боба.

A, B

(2) Трент посылает Алисе открытый ключ Боба, K_B, подписанный закрытым ключом Трента, T.

$S_T(K_B)$

(3) Алиса проверяет подпись Трента. Затем она посылает Бобу свое имя и случайное число, шифрованное о ткрытым ключом Боба.

$A, E_B(R_B)$

(4) Боб посылает Тренту свое имя, имя Алисы и случайное число Алисы, шифрованное открытым ключом Трента, K_T.

$A, B, E_{K_T}(R_A)$

(5) Трент посылает Бобу открытый ключ Алисы, K_A, подписанный закрытым ключом Трента. Он также посылает Бобу случайное число Алисы, случайный сеансовый ключ, имена Алисы и Боба, подписав все это закрытым ключом Трента и зашифровав открытым ключом Боба.

$S_T(K_A), E_{K_B}(S_T(R_A, K, A, B))$

(6) Боб проверяет подписи Трента. Затем он посылает Алисе вторую часть сообщения Трента, полученного на этапе (5), и новое случайное число, зашифровав все открытым ключом Алисы.

$E_{K_A}(S_T(R_A, K, A, B), R_B)$

(7) Алиса проверяет подпись Трента и свое случайное число. Затем она посылает Бобу второе случайное чи сло, шифрованное сеансовым ключом.

$E_K(R_B)$

(8) Боб расшифровывает свое случайное число и проверяет, что оно не изменилось.

Другие протоколы

В литературе описано множество протоколов. Протоколы X.509 рассматриваются в разделе 24.9, KryptoKnight - в разделе 24.6, а Шифрованный обмен ключами (Encrypted Key Exchange) - в разделе 22.5.

Другим новым протоколом с открытыми ключами является Kuperee [694]. Ведется работа на протоколами, использующими **маяки** - заслуживающие доверия узлы сети, которые непрерывно и широковещательно пер едают достоверные метки времени [783].

Выводы

Из приведенных протоколов, как из тех, которые вскрываются, так и из надежных, можно извлечь ряд ва жных уроков:

— Многие протоколы терпят неудачу, так как их разработчики пытались быть слишком умными. Они опт имизировали протоколы, убирая важные элементы - имена, случайные числа и т.п. - и пытаясь сделать протоколы как можно более прозрачными [43, 44].

— Оптимизация - эта манящая ловушка - сильно зависит от сделанных предположений. Пример: наличие достоверного времени позволяет вам реализовать многие вещи, невозможные в проти вном случае.

— Выбираемый протокол зависит от архитектуры используемых средств связи. Хотите ли вы минимизир овать размер сообщений или их количество? Могут ли сторона взаимодействовать каждый с каждым или круг их общения будет ограничен?

Именно подобные вопросы и привели к созданию формальных методов анализа протоколов .

3.4 Формальный анализ протоколов проверки подлинности и обмена ключами

Проблема выделения безопасного сеансового ключа для пары компьютеров (или людей) в сети настолько фундаментальна, что стала причиной многих исследований. Некоторые исследования заключались в разработке протоколов, подобных рассматриваемым в разделах 3.1, 3.2 и 3.3. Это, в свою очередь, привело к появлению более важной и интересной задачи: формальному анализу протоколов проверки подлинности и обмена ключами. Иногда прорехи в протоколах, кажущихся вполне надежными, обнаруживались спустя много лет п осле их разработки, и разработчикам потребовались средства, позволяющие сразу же проверять безопасность протокола. Хотя большая часть этого инструментария применима и к более общим криптографическим прот околам, особое внимание уделялось проверке подлинности и обмену ключами. Существует четыре основных подхода к анализу криптографических протоколов [1045]:

1. Моделирование и проверка работы протокола с использованием языков описания и средств проверки, не разработанных специально для анализа криптографических протоколов.
2. Создание экспертных систем, позволяющих конструктору протокола разрабатывать и исследовать различные сценарии.
3. Выработка требований к семейству протоколов, используя некую логику для анализа понятий "знание" и "доверие".
4. Разработка формальных методов, основанных на записи свойств криптографических систем в алге браическом виде.

Полное описание этих четырех подходов и связанных с ними исследований выходит за рамки данной книги. Хорошее введение в эту тему дано в [1047, 1355], я же собираюсь коснуться только основных вопросов.

Первый из подходов пытается доказать правильность протокола, рассматривая его как обычную компьюте рную программу. Ряд исследователей представляют протокол как конечный автомат [1449, 1565], другие используют расширения методов исчисления предиката первого порядка [822], а третьи для анализа протоколов используют языки описания [1566]. Однако, доказательство правильности отнюдь не является доказательством безопасности, и этот подход потерпел неудачу при анализе многих "дырявых" протоколов. И хотя его применение поначалу широко изучалось, с ростом популярности третьего из подходов работы в этой области были п ереориентированы.

Во втором подходе для определения того, может ли протокол перейти в нежелательное состояние (например, потеря ключа), используются экспертные системы. Хотя этот подход дает лучшие результаты при поиске "дыр", он не гарантирует безопасности и не предоставляет методик разработки вскрытий. Он хорош для проверки того, содержит ли протокол конкретную "дыру", но вряд ли способен обнаружить неизвестные "дыры" в протоколе. Примеры такого подхода можно найти в [987,1521], а в [1092] обсуждается экспертная система, разработанная армией США и названная Следователем (Interrogator).

Третий подход гораздо популярнее. Он был впервые введен Майклом Бэрроузом (Michael Burrows), Мартином Абэди (Martin Abadi) и Роджером Needхэмом. Они разработали формальную логическую модель для анализа знания и доверия, названную **БАН-логикой** [283, 284]. БАН-логика является наиболее широко распространена при анализе протоколов проверки подлинности. Она рассматривает подлинность как функцию от целостности и новизны, используя логические правила для отслеживания состояния этих атрибутов на протяжении всего протокола. Хотя были предложены различные варианты и расширения, большинство разработчиков протоколов до сих пор обращаются к оригинальной работе.

БАН-логика не предоставляет доказательство безопасности, она может только рассуждать о проверке подлинности. Она является простой, прямолинейной логикой, легкой в применении и полезной при поиске "дыр". Вот некоторые предложения БАН-логики:

Алиса считает X. (Алиса действует, как если бы X являлось истиной.)

Алиса видит X. (Кто-то послал сообщение, содержащее X, Алисе, которая может прочитать и снова передать X - возможно после дешифрирования.)

Алиса сказала X. (В некоторый момент времени Алиса послала сообщение, которое содержит предложение X. Не известно, как давно было послано сообщение, и было ли оно послано в течении текущего выполнения протокола. Известно, что Алиса считала X, когда говорила X.)

X ново. (X никогда не было послано в сообщении до текущего выполнения протокола.)

И так далее. БАН-логика также предоставляет правила для рассуждения о доверии протоколу. Для доказательства чего-либо в протоколе или для ответа на какие-то вопросы к логическим предложениям о протоколе можно применить эти правила. Например, одним из правил является правило о значении сообщения:

ЕСЛИ Алиса считает, что у Алисы и Боба общий секретный ключ, *K,* и Алиса видит *X*, шифрованное *K,* и Алиса не шифровала *X* с помощью *K*, ТО Алиса считает, что Боб сказал *X.*

Другим является правило подтверждения метки времени:

ЕСЛИ Алиса считает, что *X* могло быть сказано только недавно, и, что Боб *X* когда-то сказал *X*, ТО Алиса считает, что Боб считает *X.*

БАН-анализ делится на четыре этапа:

(1) Преобразуйте протокол к идеальной форме, используя описанные выше предложения.

(2) Добавьте все предположения о начальном состоянии протокола.

(3) Присоедините логические формулы к предложениям, получая утверждения о состоянии системы после каждого предложения.

(4) Примените логические постулаты к утверждениям и предположениям, чтобы раскрыть состояние доверия участников протокола.

Авторы БАН-логики "рассматривают идеализированные протоколы как более ясные и полные описания, чем традиционные, найденные в литературе..." [283, 284]. Другие исследователи не так оптимистичны и подвергают это действие критике, так как при этом реальный протокол может быть искажен [1161, 1612]. Дальнейшие споры отражены в [221, 1557]. Ряд критиков пытается показать, что БАН-логика может и получить очевидно неправильные характеристики протоколов [1161] - см. контрдоводы в [285, 1509] - и что БАН-логика занимается только доверием, а не безопасностью [1509]. Подробное обсуждение приведено в [1488, 706, 1002].

Несмотря на эту критику БАН-логика достигла определенных успехов. Ей удалось обнаружить "дыры" в нескольких протоколах, включая Needham-Schroeder и раннюю черновую версию протокола CCITT X.509 [303]. Она обнаружила избыточность во многих протоколах, включая Yahalom, Needham-Schroeder и Kerberos. Во многих опубликованных работах БАН-логика используется для заявления претензии о безопасности описываемых протоколов [40, 1162, 73].

Были опубликованы и другие логические системы, некоторые из них разрабатывались как расширения БАН-логики [645, 586, 1556, 828], а другие основывались на БАН-логике для исправления ощутимых слабостей [1488, 1002]. Из них наиболее успешной оказалась CNY [645], хотя у нее есть ряд изъянов [40]. В [292,474] к БАН-логике с переменным успехом были добавлены вероятностные доверия. Другие формальные логики описаны в [156, 798,288]. [1514] пытается объединить черты нескольких логик. А в [1124, 1511] представлены логики, в которых доверия изменяются со временем.

Четвертый подход к анализу криптографических протоколов предлагает моделировать протокол как алгебраическую систему, выразить состояние знания участников о протоколе и затем проанализировать достиж-

мость определенных состояний. Этот подход пока не привлек столько внимания, как формальная логика, но состояние дел меняется. Он впервые был использован Майклом Мерриттом [1076], который показал, что для анализа криптографических протоколов можно использовать алгебраическую модель. Другие подходы рассмотрены в [473, 1508, 1530, 1531, 1532, 1510, 1612].

Анализатор протоколов Исследовательской лаборатория ВМС (Navy Research Laboratory, NRL), возможно, является наиболее успешным применением этих методов [1512, 823, 1046, 1513]. Он был использован для поиска как новых, так и известных "дыр" во множестве протоколов [1044, 1045, 1047]. Анализатор протоколов определяет следующие действия:

— Принять (Боб, Алиса, *M*, *N*). (Боб принимает сообщение *M* как пришедшее от Алисы в течение локального раунда Боба *N*.)

— Узнать (Ева, *M*). (Ева узнает *M*.)

— Послать (Алиса, Боб, *Q*, *M*). (Алиса посылает *M* Бобу в ответ на запрос, *Q*.)

— Запросить (Боб, Алиса, *Q*, *N*). (Боб посылает *Q* Алисе в течение локального раунда Боба *N*.)

Используя эти действия, можно задать требования. Например:

— Если Боб принял сообщение *M* от Алисы в какой-то прошедший момент времени, то Ева не знает *M* в какой-то прошедший момент времени.

— Если Боб принял сообщение *M* от Алисы в течение локального раунда Боба *N*, то Алиса послала *M* Бобу в ответ на запрос Боба в локальном раунде Боба *N*.

Для анализа Анализатором протоколов NRL исследуемый протокол должен быть описан с помощью приведенных конструкций. Затем выполняются четыре фазы анализа: определение правил перехода для честных участников, описание операций, возможных и для полностью честных, и для нечестных участников, описание базовых блоков протокола и описание правил преобразования. Смысл всего этого в том, чтобы показать, что данный протокол удовлетворяет необходимым требованиями. Использование инструментов, подобных Анализатору протоколов NRL, в итоге могло бы привести к созданию протокола, который был бы обоснованно признан безопасным.

Хотя формальные методы в основном применяются к уже существующим протоколам, сегодня есть тенденция использовать их и при проектировании протоколов. Ряд предварительных шагов в этом направлении сделан в [711]. Это же пытается сделать и анализатор протоколов NRL [1512, 222, 1513].

Применение формальных методов к криптографическим протоколам представляет собой качественно новую идею, и трудно обрисовать, к чему может привести ее реализация. С этой точки зрения слабейшим звеном кажется процесс формализации.

3.5 Криптография с несколькими открытыми ключами

Обычная криптография с открытыми ключами использует два ключа. Сообщение, зашифрованное одним ключом, может быть расшифровано другим. Обычно один ключ является закрытым, а другой - открытым. Пусть, один ключ находится у Алисы, а другой - у Боба. Мы хотим реализовать следующую схему: Алиса может зашифровать сообщение так, что только Боб сможет расшифровать его, а Боб может зашифровать сообщение так, что только Алиса сможет прочитать его.

Эта концепция была обобщены Конном Бойдом (Conn Boyd) [217]. Представьте себе вариант криптографии с открытыми ключами, использующий три ключа: K_A, K_B и K_C, распределение которых показано в 1-й.

Алиса может зашифровать сообщение ключом K_A так, что Эллен может расшифровать его, используя ключи K_B и K_C. То же самое, сговорившись, могут сделать Боб и Кэрол. Боб может зашифровать сообщение так, что Фрэнк сможет прочесть его а Кэрол сможет зашифровать сообщение так, что его сможет прочесть Дэйв. Дэйв может зашифровать сообщение ключом K_A так, что Эллен сможет прочесть его, ключом K_B так, что его сможет прочесть Фрэнк, или обоими ключами, K_A и K_B, так, что сообщение сможет прочесть Кэрол. Аналогично, Эллен может зашифровать сообщение так, что Алиса, или Дэйв, или Фрэнк сможет прочесть его. Все возможные комбинации показаны в , других не существует.

Табл. 3-2.
Распределение ключей в трехключевой системе.

Алиса	K_A
Боб	K_B

Кэрол	K_C
Дэйв	K_A и K_B
Эллен	K_B и Kc
Франк	K_C и K_A

Такая схема может быть расширена на n ключей. Если для шифрования сообщения используется заданное подмножество ключей, то для дешифрирования сообщения потребуются оставшиеся ключи .

Широковещательная передача сообщения

Представьте, что в некоей операции занято 100 ваших тайных агентов . Вы хотите иметь возможность посылать сообщения группам агентов, но вы не знаете заранее состав групп . Можно либо шифровать сообщение отдельно для каждого корреспондента, либо распределить ключи для всех возможных комбинаций агентов . Для реализации первого способа потребуется множество сообщений, для второго - множество ключей .

Криптография с несколькими ключами позволяет решить эту задачу намного проще. Мы будем использовать трех агентов: Алису, Боба и Кэрол. Вы выдадите Алисе ключ K_A и K_B, Бобу - K_B и K_C, Кэрол - K_C и K_A. Теперь вы сможете говорить с любым нужным подмножеством агентов. Если вы хотите, чтобы сообщение могла прочитать только Алиса, зашифруйте его ключом K_C. Когда Алиса получит сообщение, она расшифрует его, последовательно используя ключи K_A и K_B. Если вы хотите послать сообщение только Бобу, зашифруйте его ключом K_A, а сообщение для Кэрол - ключом K_B. Если вы хотите, чтобы посланное сообщение могли прочитать Алиса и Боб, зашифруйте его ключами K_A и K_C.

Для трех агентов это не слишком впечатляет, но для 100 преимущество достаточно ощутимо. Индивидуальные сообщения означают использование отдельного ключа для каждого агента (всего 100 ключей) и каждого сообщения. Передача сообщений всем возможным подмножествам означает использование 2^{100}-2 различных ключей (исключены случаи сообщения всем агентам и никому из них) . Для схемы, использующий криптографию с несколькими открытыми ключами, нужно только одно шифрованное сообщение и сто различных ключей . Недостатком этой схемы является то, что вам также придется широковещательно передавать, какое подмножество агентов может читать сообщение, иначе каждому их них придется перебирать все возможные комбинации ключей в поисках подходящей. Даже только перечисление имен получателей может быть весьма внушительным. Кроме того, каждому агенту придется хранить немаленький объем информации о ключах, по крайней мере при прямолинейной реализации этой схемы .

Существуют и другие способы широковещательной передачи , ряд из них позволяет избежать описанной проблемы. Эти способы обсуждаются в разделе 22.7.

Табл. 3-3.
Шифрование сообщения в трехключевой системе.

Шифруется ключами	Должно быть расшифровано ключами
K_A	K_B и Kc
K_B	K_A и K_C
K_C	K_A и K_B
K_A и K_B	K_C
K_A и K_C	K_B
K_B и Kc	K_A

3.6 Разделение секрета

Вообразите, что вы изобрели новый, сверхлипкую, сверхсладкую сливочную тянучку или соус для гамбургеров, который еще безвкуснее, чем у ваших конкурентов. Это очень важно, и вы хотите сохранить изобретение в секрете. Только самым надежным работникам вы можете сообщить точный состав ингредиентов , но вдруг и кто-то из них подкуплен конкурентами? Секрет выкрадут, и немного погодя каждый в квартале будет делать гамбургеры с таким же безвкусным соусом, как ваш.

Предлагаемая схема называется **разделением секрета**. Есть способы взять сообщение и разделить его на части [551]. Каждая часть сама по себе ничего не значит, но сложите их - и вы получите сообщение . Если это

рецепт, и у каждого работника находится только его часть, то лишь собравшись все вместе ваши служащие см огут сделать соус. Если кто-нибудь из работников уволится, прихватив с собой свою часть рецепта, раскрытая информация по себе будет бесполезной.

По простейшей схеме сообщение делится между двумя людьми. Вот протокол, используя который Трент д елит сообщение между Алисой и Бобом:

(1) Трент генерирует строку случайных битов, R, такой же длины, что и сообщение, M.

(2) Трент выполняет "исключающее или" (XOR) над M и R, создавая S.

$$R \oplus M = S$$

(3) Трент передает Алисе R, а Бобу - S.

Чтобы получить сообщение, Алисе и Бобу нужно выполнить единственное действие:

(4) Алиса и Боб выполняют операцию над имеющимися у них частями, восстанавливая сообщение.

$$R \oplus S = M$$

Этот метод при правильном выполнении абсолютно безопасен. Каждая часть в отдельности абсолютно бе ссмысленна. Что существенно, Трент шифрует сообщение одноразовым блокнотом и дает шифротекст одному человеку, а блокнот - другому. Одноразовые блокноты, обладающие абсолютной безопасностью, обсуждаются в разделе 1.5. Никакие вычислительные средства не смогут восстановить сообщение только по одной его части.

Эту схему легко расширить на большее число людей. Чтобы разделить сообщение между более чем двумя людьми, выполните операцию XOR с большим числом строк случайных битов. В следующем примере Трент делит сообщение на четыре части:

(1) Трент генерирует три строки случайных битов, R, S и T, такой же длины, что и сообщение, M.

(2) Трент выполняет "исключающее или" (XOR) над M и созданными тремя строками, создавая U.

$$M \oplus R \oplus S \oplus T = U$$

(3) Трент передает Алисе R, Бобу - S, Кэрол - T, а Дэйву - U.

Вместе Алиса, Боб, Кэрол и Дэйв могут восстановить сообщение:

(4) Алиса, Боб, Кэрол и Дэйв собираются вместе и вычисляют:

$$R \oplus S \oplus T \oplus U = M$$

Это арбитражный протокол. Трент обладает абсолютной властью и может делать все, что он хочет. Он может раздать чепуху и утверждать, что это настоящие части секретной информации, никто не сможет это пров ерить, пока, собравшись вместе, участники протокола не попробуют прочитать письмо. Он может выдать части секрета Алисе, Бобу, Кэрол и Дэйву и позже заявить всем, что только Алиса, Кэрол и Дэйв нужны для восст ановления секрета, застрелив при этом Боба. Но это не является проблемой, так как делимый секрет принадл ежит Тренту.

Однако, одна проблема у этого протокола существует. Если любая из частей будет потеряна, а Трента не б удет поблизости, пропадет и все сообщение. Если Кэрол, обладая частью рецепта соуса, перейдет работать к ко нкуренту, оставив свою часть секрета у себя, значит, остальным не повезло. Она не сможет восстановить рецепт, но не смогут, собравшись, и Алиса, Боб и Дэйв. Ее часть также критична для восстановления сообщения, как и любая другая. Все, что известно Алисе, Бобу и Дэйву - это длина сообщения, и ничего больше. Это истинно, так как у R, S, T, U и M одинаковая длина, следовательно, каждому из участников известна длина M. Помните, сообщение M делится не в обычном смысле этого слова, а подвергается операции XOR со случайными величинами.

3.7 Совместное использование секрета

Вы вводите программу запуска ядерной ракеты и хотите быть уверенным, что никакой псих в одиночку не сможет вызвать пуск. Вы хотите быть уверенным, что и два психа не смогут вызвать пуск. Вы хотите, чтобы пуск произошел только, если не меньше трех из пяти офицеров будут психами.

Эта проблема легко может быть решена. Сделайте механическое устройство контроля запуска. Выдайте ключ каждому из пяти офицеров и потребуйте, чтобы по меньше мере три офицера вставили свои ключи в соо тветствующие гнезда, прежде чем вы разрешите им взорвать того, кого мы взрываем на этой неделе. (Если вы действительно волнуетесь, сделайте гнезда подальше друг от друга и потребуйте, чтобы офицеры вставляли ключи одновременно - вы ведь не хотели бы, чтобы офицер, выкравший недостающую пару ключей, смог бы испепелить Толедо.)

Можно сделать еще сложнее. Пусть только генерал у и паре полковников разрешено запустить ракету, но е сли генерал занят игрой в гольф, то запустить ракету имеют право только пять полковников . Сделайте контрольное устройство с пятью ключами. Выдайте генералу три ключа, а полковникам по одному. Генерал вместе с двумя полковниками или пять полковников смогут запустить ракету. Однако ни генерал в одиночку, ни четыре полковника не смогут этого сделать.

Более сложная схема совместного использования, называемая **пороговой схемой**, может решить и эти задачи, и более сложные - математически. На ее простейшем уровне вы можете взять любое сообщение (секретный рецепт, коды запуска, ваш список для прачечной и т.п.) и разделить его на *n* частей, называемых **тенями** или долями, так, что по любым *m* из них можно восстановить сообщение. Более точно, это называется **(m,n)-пороговой схемой**.

Используя (3,4)-пороговую схему, Трент может разделить свой секретный рецепт между Алисой, Бобом, К эрол и Дэйвом так, что любые трое из них могут сложить свои тени вместе и восстановить сообщение. Если Кэрол в отпуске, то Алиса, Боб и Дэйв смогут восстановить сообщение. Если Боб попал под автобус, то сообщение смогут восстановить Алиса, Кэрол и Дэйв. Но если Боб попал под автобус, а Кэрол в отпуске, то Алиса и Дэйв самостоятельно не смогут восстановить сообщение.

Вообще, пороговые схемы могут быть еще более гибкими. Можно отмоделировать любые сценарии совм естного использования, которые вы только сможете вообразить. Можно разделить сообщение между людьми в вашем здании так, что для его восстановления, если нет никого с третьего этажа, потребуется семь человек с первого этажа и пять со второго, в противном случае достаточно представителя третьего этажа вместе с тремя человеками с первого этажа и двумя со второго. Если же есть кто-то с четвертого этажа, то для восстановления сообщения достаточно этого человека и одного с третьего этажа или этого человека вместе с двумя с первого этажа и одного со второго. Если же ... ну вы уловили идею.

Эта идея была независимо выдвинута Ади Шамиром [1414] и Джорджем Блэкли (George Blakley) [182] и интенсивно была изучена Гусом Симмонсом (Gus Simmons) [1466]. Множество различных алгоритмов обсуждается в разделе 23.2.

Совместное использование с мошенниками

Существует множество способов обмануть пороговую схему. Вот только несколько из них. Сценарий 1 : Полковники Алиса, Боб и Кэрол сидят в изолированном бункере где-то глубоко под землей. Однажды они получают закодированное сообщение от президента: "Запустить ракеты. Мы собираемся стереть с лица Земли любые сл еды исследований противника в области нейронных сетей". Алиса, Боб и Кэрол открывают свои тени, но Кэрол вводит случайное число. Он на самом деле пацифист и не хочет, чтобы ракеты были запущены. Поскольку Кэрол не ввела правильное тени, секретная информация, которую они хотели получить, оказалась неправильной. Ракеты остались в своих шахтах. И самое плохое, никто не знает почему. Даже объединившись Алиса и Боб не смогут доказать, что тень Кэрол неправильна.

Сценарий 2: Полковники Алиса и Боб сидят в бункере вместе с Мэллори. Мэллори ложно выдает себя за полковника. От президента приходит то же самое сообщение и все открывают свои тени. "Ха-ха-ха!" кричит Мэллори. "Я подделал это сообщение президента. Теперь я знаю обе ваших доли." Он убегает вверх по лестнице и исчезает прежде, чем его успеют поймать.

Сценарий 3: Полковники Алиса, Боб и Кэрол сидят в бункере вместе с Мэллори, который снова замаскировался. (Помните, у Мэллори нет правильной тени.) От президента приходит то же самое сообщение и все о ткрывают свои тени. Мэллори открывает свою тень, только услышав все остальные. Так как для восстановления секрета требуется только три тени, он может быстро создать правильную тень и открыть ее. Итак, он не только заполучил секрет, но и никто не догадался, что он не является частью этой системы. Некоторые протоколы, которые позволяют бороться с подобными мошенниками, рассматриваются в разделе 23.2.

Совместное использование секрета без Трента

Банк хочет, чтобы его подвал могли открыть трое из пяти офицеров, введя свои ключи. Это выглядит как типичная (3,5)-пороговая схема, но с одной тонкостью. Никто не знает секрета целиком. Трента, которые делит секрет на пять частей, нет. Существуют протоколы, используя которые пять офицеров могут создать секрет и поделить его на части так, что никто из офицеров не узнает секрета, пока он не будет восстановлен . В этой книге я не рассматриваю эти протоколы, подробности см. в [756].

Совместное использование секрета без раскрытия долей

У этих схем есть одна проблема. Когда участники протокола собираются, чтобы восстановить секрет, они открывают свои части. Но раскрытие секрета не всегда желательно. Если разделяемый секрет является закр ытым ключом (например, к цифровой подписи), то каждый из *n* участников может выполнить частичную подпись

документа. После *n*-ой частичной подписи документ оказывается подписан совместно используемым закрытым ключом, а ни один из участников не может узнать содержания части, используемой другим участником . Смысл в том, что вы можете повторно использовать секрет, и для работы с ним вам не понадобится надежный посре д-ник. Дальнейшее развитие эта идея получила в работах Иво Десмедта (Yvo Desmedt) и Йера Френкеля (Yair Frankel) [483, 484].

Подтверждаемое совместное использование секрета

Трент передает Алисе, Бобу, Кэрол и Дэйву часть секрета (или, по крайней мере, заявляет, что он это делает). Единственный способ убедиться, что их части правильны - это попытаться восстановить секрет . Может быть Трент послал Бобу поддельную часть, или часть Боба случайно испортилась при передаче по линиям связи. Подтверждаемое совместное использование секрета позволяет каждому из участников лично убедиться, что их часть правильна, без необходимости восстанавливать секрет [558, 1235].

Схемы совместного использования секрета с мерами предохранения

Секрет делится среди 50 человек так, чтобы любые 10 могли собраться вместе и восстановить секрет . Это нетрудно. Но, можем ли мы реализовать ту же схему совместного использования секрета, добавив требование, чтобы 20 человек могли собраться вместе и *помешать* остальным, независимо от их числа, восстановить секрет? Оказывается, что да [153].

Математика достаточно сложна, но основная идея в том, что каждый получает две части : часть "да" и часть "нет". Когда приходит время восстановить секрет, люди предоставляют одну из своих частей . Какую конкретно зависит от того, хотят ли они, чтобы секрет был раскрыт. Если предоставлено *m* или больше долей "да" и меньше чем *n* долей "нет", то секрет может быть восстановлен. В противном случае, это невозможно.

Конечно же, ничего не мешает достаточному числу людей "да" отойти в уголок, уединившись от людей "нет" (если они знают, кто есть кто) и восстановить секрет. Но при условии, что все передают свои части в централ ь-ный компьютер эта схема будет работать.

Совместное использование секрета с вычеркиванием из списка

Вы создали систему совместного использования секрета и теперь хотите застрелить одного из владельцев части секрета. Вы могли бы создать новую схему, исключив этого несчастного, но время поджимает. Для п о-добной системы существуют способы копирования. Они позволяют активизировать новую схему совместного использования секрета сразу же после того, как вы перестали доверять одному из участников [1004].

3.8 Криптографическая защита баз данных

База данных членов организации - это весьма важная вещь. С одной стороны вы хотите предоставить к не доступ всем членам, желая, чтобы они общались друг с другом, обменивались идеями и делились друг с другом бутербродами. С другой стороны, если вы пустите в вашу базу данных кого угодно, сведения обязательно поп а-дут в руки надоедливых страховых агентов и докучливых поставщиков всякого хлама по почте .

Криптография может облегчить эту проблему. Можно зашифровать базу данных так, чтобы получить адрес одного человека было легко, а извлечь список почтовых адресов всех членов - трудно .

Схема, предложенная в [550, 549], прямолинейна. Выберите однонаправленную хэш-функцию и симметри ч-ный алгоритм шифрования. У каждой записи в базе данных два поля. Индексным полем является фамилия чл е-на, и именно оно обрабатывается однонаправленной хэш-функцией . Поле данных, в котором хранится полное имя и адрес члена, шифруется с помощью используемой в качестве ключа фамилии . Если вы не знаете фам и-лии, вы никогда не сможете расшифровать поле данных .

Поиск по конкретной фамилии прост. Сначала хэшируется фамилия, и выполняется поиск значения хэш-функции в базе данных. Наличие нескольких совпадений означает, что база данных содержит информацию о нескольких людях с такой фамилией.

В [550] авторы используют эту систему для защиты словаря из 6000 испанских слов. Они сообщают о том, что потеря производительности, вызванная шифрованием, минимальна . В более сложной схеме [549] использ у-ется поиск по нескольким индексам, но идея остается той же. Основная проблема, связанная с этой системой, состоит в том, что вы не сможете найти человека, не зная, как пишется его фамилия . Можно попробовать не-сколько вариантов, пока не будет найден правильный, но неудобно перебирать всех, чьи фамилии начинаются на "Sch" при поиске "Schneier."

Эта защита несовершенна. Очень назойливый страховой агент восстановит базу данных членов организации с помощью грубого взлома, перебирая все возможные фамилии . Если у него есть телефонная база данных, он может использовать имеющийся в ней список фамилий. Это пережевывание номеров может занять несколько

недель, но дело будет сделано. Тем не менее такая схема усложнит работу взломщика (в мире продаже всякой чепухи по почте "усложнит" быстро превращается в "сделает слишком дорогой". Другой подход, предложенный в [185], предлагает набирать статистику по шифрованным данным .

Глава 4
Промежуточные протоколы

4.1 Службы меток времени

Во многих ситуациях людям нужно убедиться, что определенный документ уже существовал в определенный момент времени. Примером является спор об авторских правах или патенте. Дело выигрывает сторона, которая представит более раннюю копию спорной работы. Бумажные документы заверяются нотариусами и хранятся у юристов. Если возникает спор, нотариус или юрист свидетельствует, что письмо существовало в определенный момент времени.

В цифровом мире все гораздо сложнее. Нет способов обнаружить признаки подделки электронного докуме нта. Его можно бесконечно копировать и изменять, не оставляя никаких следов. Несложно и изменить время со здания компьютерного файла. Никто не может взглянуть на документ и с полной уверенностью сказать: "Да, этот документ был создан раньше 4 ноября 1952 года"

Этой проблемой задались Стюарт Хабер (Stuart Haber) и В. Скотт Сторнетта (W. Scott Stornetta) из Bellcore [682, 683, 92]. Им потребовался протокол цифровых меток времени со следующими свойствами:

— Метка времени должна существовать сама по себе, независ ия от физической среды, используемой для ее хранения.

— Должно быть невозможно тайно изменить ни единого бита документа.

— Должно быть невозможно задать для документа метку времени, отличного от текущего.

Решение с посредником

В этом протоколе участвуют Трент, обладающий надежной службой меток времени, и Алиса, которая хочет задать метку времени для документа.

(1) Алиса передает копию документа Тренту.

(2) Трент записывает время и дату получения документа, оставляя у себя копию для безопасного хранения.

Теперь, если кто-нибудь усомнится в заявленном Алисой времени создания документа, то Алисе просто нужно обратиться к Тренту. Он предоставит свою копию документа и подтвердит, что он получил документ в указанный день и час.

Этот протокол работает, но есть ряд очевидных проблем. Во первых, невозможно сохранить тайну - Алиса должна предоставить копию документа Тренту. Кто-то, подслушивающий линию связи, сможет прочесть док умент. Она может зашифровать документ при передаче, но ведь он должен будет храниться в базе данных Тре нта. Насколько эта база безопасна?

Во вторых, самой базе данных придется быть очень большой. Велики будут требования и к пропускной сп особности линии связи.

Третья проблема связана с возможными ошибками. Ошибки при передаче или электромагнитная бомба, взорванная где-то в центральном компьютере Трента могут полностью свести на нет заявление Алисы о метке времени.

И в четвертых, может оказаться невозможным найти такого честного Трента для ведения службы меток времени. Может быть, Алиса использует метку времени Боба. Ничто не остановит Алису и Боба от сговора и пометки документа тем временем, которое им нужно.

Улучшенное решение с посредником

Большинство этих проблем легко снимаются при использовании однонаправленной хэш-функции и цифр овых подписей:

(1) Алиса вычисляет значение однонаправленной хэш-функции для документа.

(2) Алиса передает это значение Тренту.

(3) Трент добавляет время и дату получения этого значения и затем подписывает результат цифровой подп исью.

(4) Трент отправляет подписанное значение хэш-функции вместе с меткой времени Алисе.

Это решает все проблемы, кроме последней. Алисе больше не нужно беспокоиться о раскрытии содержания

документа, использование значения хэш-функции вполне достаточно. Тренту больше не нужно хранить копии документов (и даже значения хэш-функции), поэтому снимаются проблемы безопасности и объема сохраняемых данных (помните, у однонаправленных хэш-функций нет ключа). Алиса может немедленно проверить подписанную метку времени, полученную на этапе (4), и немедленно обнаружить любые ошибки передачи. Единственной оставшейся проблемой остается сговор Алисы и Трента с целью создания поддельной метки времени.

Протокол связи

Одним из путей решения этой проблемы является установление связи между меткой времени Алисы и метками времени, ранее созданными Трентом. Весьма вероятно, что эти метки были созданы не для Алисы, а для других людей. Так как порядок, в котором Трент получает различные запросы о метках времени не может быть известен заранее, перед меткой времени для Алисы должна была появиться другая метка времени. И так как запрос, пришедший позже, связан с меткой времени Алисы, то ее метка должна была появиться раньше. Эти две метки содержат между собой запрос Алисы как будто в сэндвиче.

Если A - это имя Алисы, H_n - значение хэш-функции, для которого Алиса хочет зафиксировать время, а T_{n-1} - предыдущая метка времени, то протокол имеет следующий вид:

(1) Алиса посылает Тренту H_n и A.

(2) Трент посылает Алисе обратно:

$T_n=S_K(n,A,H_n,T_n,I_{n-1},H_{n-1},T_{n-1},L_n)$

где состоит L_n - это информация о следующей хэшированной связи:

$L_n =H(I_{n-1},H_{n-1},T_{n-1},L_{n-1})$

S_K указывает, что сообщение подписано открытым ключом Трента. Имя Алисы определяет ее как отправителя запроса. Параметр n указывает последовательность запросов. Это n-ая метка времени, которую создал Трент. Параметр T_n - это время. Дополнительно используется информация об идентификаторе, оригинального значения хэш-функции, времени и хэшированной метка предыдущего документа, помеченного Трентом.

(3) Когда Трент помечает следующий документ, он посылает Алисе идентификатор отправителя этого документа: I_{n-1}.

Если кто-то оспаривает метку времени Алисы, ей надо только связаться с отправителями предыдущего и следующего документов: I_{n-1} и I_{n+1}. Если и их свидетельство под вопросом, можно обратиться к авторам документов I_{n-1} и I_{n+1} и т.д. Любой может показать, что его документ был помечен после одного документа и перед другим.

Этот протокол мешает Алисе и Тренту договориться и создать документ с временем создания, отличным от времени создания настоящего документа. Трент не может изменить дату документа Алисы на более раннюю, так как для этого нужно знать заранее, для какого документа перед данным будет проставляться метка времени. Даже если он сможет подделать предыдущий документ, ему придется знать, какой документ предшествовал предыдущему и так далее. Трент не может изменить дату документа Алисы и на более позднюю, потому что метка времени должна быть вставлена перед меткой времени документа, заверяемого сразу же после данного, а этот документ уже существует. Единственный возможный способ сломать эту схему - это ввести фиктивную цепочку документов перед и после документа Алисы, достаточно длинную, чтобы лишить терпения того, кто проверяет метку времени документа Алисы.

Распределенный протокол

Люди умирают, метки времени теряются. Между пометкой документа и его оспариванием может произойти многое, что помешает Алисе получить копию метки времени I_{n-1}. Эта проблема может быть частично снята вставкой меток времени предыдущих 10 человек в метку Алисы и последующей передаче Алисе имен следующих 10 человек. Так у Алисы появится гораздо больше возможностей найти людей, все еще хранящих свои метки времени.

Развивая эту идею, следующий протокол позволяет обойтись и без Трента:

(1) Используя в качестве входа H_n, Алиса генерирует последовательность случайных чисел с помощью криптографически безопасного генератора случайных чисел.

$V_1, V_2, V_3, \ldots V_k$

(2) Алиса рассматривает каждое из этих чисел как идентификатор, I, другого человека и посылает каждому из этих людей H_n.

(3) Каждый из них добавляет время и дату к значению хэш-функции, подписывает результат и отправляет его

обратно Алисе.

(4) Алиса собирает и хранит все подписи как метку времени.

Криптографически безопасный генератор случайных чисел, используемый на этапе (1), позволяет Алисе избежать преднамеренного выбора коррумпированных I в качестве свидетелей. Даже если она сделает простейшие изменения в своем документе, пытаясь создать набор коррумпированных I, ее шансы добиться этого пренебрежимо малы. Хэш-функция рандомизирует значения, и Алиса не может на них воздействовать.

Этот протокол работает, потому что подделать метку времени Алиса может, только договорившись о сотрудничестве со всеми k людьми. Так как на этапе (1) она выбирала их случайным образом, вероятность этого очень низка. Чем коррумпированнее общество, тем больше должно быть число k.

Кроме того, должен использоваться некоторый механизм, учитывающий то, что ряд людей не смогут вовремя возвратить метку времени. Все, что нужно для правильной метки времени - это некоторое подмножество k. Детали зависят от реализации.

Дальнейшая работа

Дальнейшие улучшения протоколов метки времени описаны в [92]. Авторы используют двоичные деревья для увеличения количества меток времени, зависящих от данной метки, уменьшая вероятность создания цепочки фальшивых меток времени. Они также рекомендуют публиковать список значений хэш-функций за прошедший день в некотором общедоступном источнике, например газете. Это работает как отправка значения хэш-функции случайным людям в распределенном протоколе. Действительно, метка времени появляется в каждом номере воскресной *Нью-Йорк Таймс* с 1992 года.

Эти протоколы меток времени запатентованы [684, 685, 686]. Патенты принадлежат дочерней компании Bellcore, названной Surety Technologies, которая продает Систему цифрового нотариата, поддерживающую эти протоколы. В первой версии клиенты посылали запросы о "заверении" на центральный координирующий центр. Следуя методике Меркла по использованию хэш-функций для построения деревьев [1066], сервер строит дерево значений хэш-функции, листья которого представляют собой все запросы, полученные в течение данной секунды, и посылает каждому автору запроса список значений хэш-функции, описывающий путь от его листа до корня. Клиентская часть программного обеспечения сохраняет этот список и может выдать "сертификат" Цифрового нотариата для любого файла, который был сертифицирован. Последовательность корней этих деревьев образует "Запись универсального подтверждения" ("Universal Validation Record"), которая будет доступна в электронном виде во многих хранилищах (и также выпущена на CD-ROM). Клиентская часть также содержит функцию "подтверждения", позволяющую пользователю проверить, был ли заверена именно текущая форма файла (запросив из хранилища корень соответствующего дерева и сравнив его со значением хэш-функции, соответствующим образом рассчитанным для файла, и сертификатом). За дальнейшей информацией обращайтесь в Surety Technologies, 1 Main St., Chatham, NJ, 07928; (201) 701-0600; Fax: (201) 701-0601.

4.2 Подсознательный канал

Алиса и Боб были арестованы и отправлены в тюрьму, он - в мужскую, а она - в женскую. Уолтер, надзиратель, разрешает Алисе и Бобу обмениваться сообщениями, но он не разрешает шифровать сообщения. Уолтер считает, что они планируют бегство, поэтому он хочет читать все, что они пишут.

Уолтер надеется также суметь обмануть Алису или Боба. Он хочет, чтобы один из них посчитал принятое им ложное сообщение настоящим. Алиса и Боб мирятся с риском возможного обмана, иначе они вообще не смогут общаться, но им нужно согласовать свои планы. Для этого им необходимо обмануть надзирателя и найти способ передавать секретную информацию. Им нужно создать подсознательный канал, скрытый канал связи в открытых сообщениях, хотя сообщения сами по себе не содержат секретной информации. С помощью обмена совершенно безобидными подписанными сообщениями они обменяются секретной информацией и одурачат Уолтера, даже если он просматривает все сообщения.

Простым подсознательным каналом может быть число слов в предложении. Нечетное число слов в предложении может соответствовать "1", а четное число слов -"0". Так, пока вы читаете этот самый обычный абзац, я передал вам сообщение "110". Проблематичность этого метода в том, что он является обычной стеганографией (см. раздел 1.2), ключ не используется и безопасность зависит от секретности алгоритма.

Густавус Симмонс придумал идею организации подсознательного канала с помощью обычного алгоритма цифровой подписи [1458, 1473]. Так как подсознательные сообщения спрятаны в том, что выглядит нормальными цифровыми подписями, это форма маскировки. Уолтер видит, как подписанные безобидные сообщения передаются туда и обратно, но реальная передаваемая информация проходит незаметно для него по подсознательному каналу. В действительности, алгоритм подсознательного канала в подписях не отличим от нормального алгоритма в подписях, по крайней мере для Уолтера. Ер не только не может прочитать сообщение, передаваемое по подсознательному каналу, но у него вообще нет ни малейшего представления о существовании такого

сообщения. В общем случае протокол выглядит примерно так:

(1) Алиса создает безобидное сообщение, все равно какое.

(2) Используя общий с Бобом ключ, Алиса подписывает безобидное сообщение, пряча свое подсознательное сообщение в подписи. (Это суть подсознательного протокола, см. раздел 23.3).

(3) Алиса посылает подписанное сообщение Бобу через Уолтера.

(4) Уолтер читает безобидное сообщение и проверяет подпись. Не обнаружив ничего подозрительного, он передает подписанное сообщение Бобу.

(5) Боб проверяет подпись под безобидным сообщением, убеждаясь, что сообщение получено от Алисы.

(6) Боб игнорирует безобидное сообщение и, используя общий с Алисой секретный ключ, извлекает подсознательное сообщение.

А мошенничество? Уолтер не верит никому, и никто не верит Уолтеру. Он всегда может помешать передаче сообщений, но у него нет возможности подделать сообщение. Так как Уолтер не может создать правильной подписи, Боб обнаружит подделку на этапе (5). Уолтер не может читать подсознательные сообщения - у него нет нужного ключа. Что еще важнее, у него нет ни малейшего представления, что подсознательные сообщения существуют. Подписанные сообщения, использующие алгоритм цифровой подписи на вид ничем не отличаются от подписанных сообщений, содержащих подсознательные сообщения в подписи.

Более проблематичен обман своего партнера Алисой или Бобом. В некоторых реализациях подсознательного канала секретная информация, нужная Бобу для чтения подсознательного сообщения, совпадает с информацией, нужной Алисе для подписи безобидного сообщения. Если это так, Боб может выдать себя за Алису. Он может подписать сообщения, выдав их за посланные Алисой, и Алиса ничего не сможет с этим поделать. Если ей необходимо отправить ему подсознательное сообщение, она должна верить, что он не будет мошенничать с ее закрытым ключом.

В других реализациях подсознательного канала такой проблемы нет. Секретный ключ, общий для Алисы и Боба, позволяет Алисе отправлять Бобу подсознательные сообщения, но закрытый ключ Алисы не передается, и Боб не может подписывать сообщения ее подписью. Алисе не нужно верить, что Боб не будет мошенничать с ее закрытым ключом.

Применения подсознательного канала

Наиболее очевидным применением подсознательного канала является шпионская сеть. Если кто-то посылает и принимает сообщения, то передача сообщений по подсознательному каналу в подписанных документах не будет вызывать подозрений. Конечно же, вражеские шпионы могут делать то же самое.

Используя подсознательный канал, Алиса может, даже если ей угрожают, безопасно подписать документ. Подписывая документ, она может вставить подсознательное сообщение, написав: "Я арестована ". Иные применения не так бросаются в глаза. Компания может подписать документы и вставить подсознательные сообщения для отслеживания времени действия документов. Правительство может "пометить" электронные деньги. Мошенническая программа для подписи документов может использовать подсознательные сообщения в создаваемых подписях для организации утечки секретной информации. Возможности бесконечны.

Подписи, свободные от подсознательного канала

Алиса и Боб обмениваются подписанными сообщениями, обговаривая сроки контракта. Они используют протокол цифровой подписи. Однако, эти переговоры на самом деле маскируют шпионскую деятельность Алисы и Боба. Используя алгоритм цифровой подписи, они не волнуются о подписываемых ими сообщениях. Для обмена секретной информацией они используют подсознательный канал в подписях под документами. Контрразведка, однако, не знает, что переговоры о контракте и используемые подписанные сообщения являются только прикрытием. Для противодействия подобной схеме были разработаны схемы подписи, свободной от подсознательного канала. Используемые в этих схемах цифровые подписи невозможно изменить для организации подсознательного канала. Подробности см. в [480, 481].

4.3 Неотрицаемые цифровые подписи

Обычные цифровые подписи могут быть точно скопированы. Иногда это свойство полезно, например, при распространении публичных заявлений. В другой раз это может оказаться проблемой. Вообразите личное или деловое письмо, подписанное цифровой подписью. Если распространяется множество копий этого документа, каждая из которых может быть проверена кем угодно, то это может привести к замешательству или шантажу. Лучшим решением является цифровая подпись, правильность которой может быть доказана получателю, но которая не позволит получателю показать третьей стороне полученное сообщение без согласия разрешения ли-

ца, подписавшего сообщение.

Alice Software Company (Компания программного обеспечения Алисы) распространяет продукт DEW (Do-Everything-Word, Делая со словом что угодно). Для гарантии отсутствия вирусов каждая копия содержит цифровую подпись. Однако, создатели хотят, чтобы только легальные покупатели продукта, а не компьютерные пираты могли проверить подпись. В то же время, если обнаруживаются копии DEW, содержащие вирус, у Alice Software Company не должно быть возможности отрицать правильную подпись.

Неотрицаемые подписи [343,327] удобны для решения подобных задач. Как и обычная цифровая подпись, неотрицаемая цифровая подпись зависит от подписанного документа и закрытого ключа человека, подписавшего документ. Но, в отличие от обычных цифровых подписей, неотрицаемая подпись не может быть проверена без разрешения подписавшего. Хотя для этих подписей можно было бы подобрать название получше, например, "непередаваемые подписи", существующее название обусловлено тем обстоятельством, что если Алисе придется либо подтвердить, либо отрицать подпись - может быть в суде - она не сможет ложно отрицать свою настоящую подпись. Несмотря на сложность математики основная идея проста:

(1) Алиса предъявляет Бобу подпись.

(2) Боб создает случайное число и посылает его Алисе.

(3) Алиса выполняет вычисления, используя случайное число и свой закрытый ключ, и посылает Бобу результат. Алиса может выполнить эти вычисления только, если подпись правильна.

(4) Боб проверяет это.

Также существует дополнительный протокол, позволяющий Алисе доказать, что она не подписывала документ, и не допускающий возможности ложно отказаться от подписи.

Боб не может повернуться и убедить Кэрол, что подпись Алисы правильна, потому что Кэрол не знает, что числа Боба случайны. Он может легко без помощи Алисы изложить протокол на бумаге и послать результат Кэрол. Кэрол может удостовериться в правильности подписи Алисы только, если она сама выполнит этот протокол с Алисой. Сейчас кажется, что в этом немного смысла, но он появится, когда вы взглянете на математику раздела 23.4.

Это решение не совершенно. Иво Десмедт и Моти Юнг (Moti Yung) показали, что в некоторых случаях Боб может убедить Кэрол в правильности подписи Алисы [489].

Например, Боб покупает легальную копию DEW. Он может подтвердить подпись под программным продуктом, когда захочет. Тогда, Боб может убедить Кэрол, что он работает на Alice Software Company, и продать ей пиратскую копию DEW. Когда Кэрол попытается подтвердить подпись Боба, он одновременно подтверждает подпись у Алисы. Когда Кэрол посылает ему случайное число, он отправляет его Алисе. Ответ Алисы он пересылает Кэрол. Кэрол убеждается в том, что она - легальный покупатель, хотя она таковым не является. Такое вскрытие является прмером проблемы великого гроссмейстера и подробно рассматривается в разделе 5.2.

Несмотря на это у неотрицаемых подписей множество применений, во многих случаях Алиса не хочет, чтобы кто угодно мог проверить ее подпись. Она может не хотеть, чтобы подпись под ее личной корреспонденцией могла быть проверена журналистами, чтобы ее письма были опубликованы и подтверждены независимо от контекста, или просто, чтобы нельзя было обнаружить изменения в письмах, сделанные ею позже. Если она подписывает информацию, которую она продает, то она не хочет, чтобы кто-то, не заплатив за информацию, мог подтвердить ее достоверность. Защитить свои права Алиса может контролируя тех, кто проверяет ее подпись.

Ряд вариантов неотрицаемых подписей отделяет связь между подписавшим и сообщением от связи между подписавшим и подписью [910]. В одной схеме кто угодно может проверить, что подпись действительно была создана ее автором, а для проверки правильности подписи для данного сообщения требуется сотрудничество подписавшего.

Близким понятием является **доверительная неотрицаемая подпись** [1229]. Представьте, что Алиса работает на Toxins, Inc., и передает обличающие документы в газету, используя протокол неотрицаемой подписи. Алиса может подтвердить свою подпись только репортеру газеты и никому больше. Однако, негодяй Боб подозревает, что источником документов является Алиса. Он требует, чтобы Алиса использовала протокол снятия подписи, чтобы очистить свое имя, а Алиса отказывается. Боб настаивает, что единственной причиной отказа Алисы является ее виновность, и убивает ее.

Доверительные неотрицаемые подписи похожи на обычные неотрицаемые подписи за исключением протокола снятия подписи, который может быть запущен только Трентом. Только Трент, а не Боб может потребовать от Алисы использовать протокол снятия. И если Трент представляет судебную систему, то он использует этот протокол только для разрешения формального спора.

4.4 Подписи уполномоченного свидетеля

Alice Software Company добилась бурного роста продаж, продавая DEW - такого, что Алиса большую часть времени посвящает подтверждению неотрицаемых подписей, а не работе над новыми возможностями.

Алисе хотелось бы назначить некоего человека в компании ответственным за подтверждение подписи. Ал иса, ил любой другой программист, сможет подписывать документы с помощью неотрицаемого протокола. Но все подтверждения будут проводиться только Кэрол.

Оказывается, это возможно с использованием **подписи уполномоченного свидетеля** [333,1213]. Алиса может подписать документ, так что Боб убедится, что подпись правильна, но не сможет убедить в этом третье л ицо. В то же время Алиса назначает Кэрол на должность будущего свидетеля своей . Алисе даже не нужно заранее просить разрешения у Кэрол, ей только нужно использовать открытый ключ Кэрол. И Кэрол сможет подтвердить подпись Алисы, если Алиса уехала из города, уволилась, была повышена или умерла .

Подписи уполномоченного свидетеля представляют собой некий компромисс между обычными цифров ыми подписями и неотрицаемыми подписями . Определенно существуют случаи, когда Алиса захочет ограничить число тех, кто может подтвердить ее подпись . С другой стороны, предоставление Алисе полного контроля по дрывает сам институт подписей - Алиса может отказаться сотрудничать и в подтверждении, и в отрицании, она может заявить о потере ключей для подтверждения или отрицания, наконец, она может быть просто недоступна . Подписи уполномоченного свидетеля могут предоставить Алисе защиту, создаваемую неотрицаемой подписью, одновременно не позволяя ей злоупотреблять этой защитой . Алиса даже может предпочесть этот способ : подписи уполномоченного свидетеля могут помешать ложным применениям, защитить ее, если она действительно потеряла свой ключ, выручить, если она в отпуске, в больнице или даже умерла .

Эта идея может иметь различные применения. Например, Кэрол может сделаться государственным нотари усом. Она опубликует в каком-то каталоге свой открытый ключ, и люди получают возможность назначать ее свидетелем своих подписей. Получая небольшую плату за подтверждение подписей, она может жить припева ючи.

Кэрол может быть агентством по охране авторских прав, правительственным агентством или еще какой-нибудь организацией . Этот протокол позволяет отделить людей, подписывающих документы, от людей, которые помогают подтверждать по дписи.

4.5 Подписи по доверенности

Подписи уполномоченного свидетеля позволяют подписавшему назначить кого-то другого для подтвержд ения подписи . Пусть Алиса хочет поехать в деловую поездку в некое место, где нет хорошей компьютерной сети - в африканские джунгли, например. Или она не дееспособна после тяжелой операции . Она ожидает получения важной электронной почты и инструктирует своего секретаря Боба ответить соответствующим образом . Как Алиса может передать Бобу полномочия подписывать сообщения за нее, не передавая ему своего закрытого ключа?

Решением этого являются **подписи по доверенности** [1001]. Алиса передает Бобу полномочия так, чтобы имели место следующие свойства :

— ***Различимость***. Кто угодно может отличить подписи по доверенности от обычных подписей.

— ***Неподдельность***. Только сам подписывающий и назначенный им подписывающий по доверенности м ожет создать правильную подпись по доверенности.

— ***Отличие подписи*** по доверенности. подписывающий по доверенности не может создать правильную подпись по доверенности, которую можно выдать за оригинальную подпись.

— ***Подтверждаемость***. По подписи по доверенности контролер должен убедиться в согласии первон ачального подписывающего с подписанным сообщением.

— ***Идентифицируемость***. Первоначальный подписывающий может определить личность подписывающ его по доверенности по подписи по доверенности.

— ***Неотрицаемость***. Подписывающий по доверенности не может снять им подпись по доверенности, п олученную пользователем

В некоторых случаях требуется строгая форма идентифицируемости - кто угодно должен иметь возможность определить личность подписывающего по доверенности по подписи по доверенности . Схемы подписи по доверенности, основанные на различных схемах цифровой подписи приведены в [1001].

4.6 Групповые подписи

Эта проблема была введена Дэвидом Чаумом (David Chaum) в [330]:

> У компании есть несколько компьютеров, подсоединенных к локальной сети . В каждом отделе компании есть свой принтер (также присоединенный к сети), и только один человек в отделе имеет право печатать на принтере своего отдела. Перед печатью, следовательно, принтер должен проверять, что данный сотрудник работает в этом отделе В то же время, компания хочет обеспечить тайну, имя пользователя не должно раскрываться. Если, однако, кто-то в конце дня обнаружит, что принтер используется слишком часто, у директора должна быть возможность найти того, кто использует принтер не по назначению и послать ему чек.

Решение этой проблемы называется групповой подписью. Групповые подписи обладают следующими свойствами:

— Только члены группы могут подписывать сообщения.

— Получатель подписи может убедиться, что это - правильная подпись группы.

— Получатель подписи не может определить, кто именно из членов группы подписал документ.

— при споре подпись будет раскрыта для определения личности подписавшего.

Групповые подписи с надежным посредником

Следующий протокол использует заслуживающего посредника:

(1) Трент создает большую кучу пар открытый ключ/закрытый ключ и выдает каждому члену группы индивидуальный список уникальных закрытых ключей. Одинаковых ключей в списках нет. (Если в группе n членов, и каждый из них получает m пар ключей, то общее число пар ключей составит $n*m$.)

(2) Трент публикует главный список всех открытых ключей для группы в случайном порядке, сохраняя в секрете, какой ключ кому принадлежит.

(3) Когда член группы хочет подписать документ, он случайным образом выбирает ключ из своего списка.

(4) Когда кто-то хочет убедиться, что подпись принадлежит члену данной группы, он перебирает главный список в поисках подходящего открытого ключа и проверяет подпись.

(5) В случае споров обращаются к Тренту, который знает, какие ключи использует каждый член группы.

Проблема протокола состоит в том, что для него необходим надежный посредник. Трент знает закрытые ключи каждого и может подделывать подписи. Кроме того, должно быть достаточно велико, чтобы помешать попыткам анализа с целью поиска владельца каждого ключа.

Чаум [330] перечислил ряд других протоколов, в некоторых из них Трент не может подделать подписи, а в других от и не нужен вовсе. Еще один протокол [348] не только прячет личность подписывающего, но и позволяет добавлять новых членов в группу . И еще один протокол можно найти в [1230].

4.7 Подписи с обнаружением подделки

Пусть Ева является могучим противником. У нее есть обширные компьютерные сети и залы, набитые компьютерами Крэй, на много порядков более мощных, чем доступные Алисе . Все эти компьютеры днем и ночью пыхтят, пытаясь взломать закрытый ключ Алисы. Наконец - успех. Теперь Ева может выдавать себя за Алису, при желании подделывая ее подпись под документами.

Подписи с обнаружением подделки, введенные Биржитом Пфицманом (Birgit Pfitzmann) и Майклом Уэйднером (Michael Waidner) [1240] предотвращают подобное мошенничество. Если после грубого взлома Ева подделывает подписи Алисы, Алиса сможет доказать подлог . Если Алиса подпишет документ, а потом объявит свою подпись подложной, правда может быть доказана судом .

Основная идея, стоящая за подписями с обнаружением подделки, состоит в том, что для каждому возможному открытому ключу соответствует множество возможных закрытых ключей . Каждый из этих закрытых ключей дает множество различных цифровых подписей . Однако, у Алисы есть только один закрытый ключ, и она может рассчитать только одну подпись. Другие закрытые ключи ей неизвестны.

Ева хочет взломать закрытый ключ Алисы. (Ева также сможет быть Алисой, вычислив для себя второй закрытый ключ.) Она собирает подписанные сообщения и, используя множество своих суперкомпьютеров, пытается раскрыть ключ Алисы. Даже если ей удастся раскрыть подходящий закрытый ключ, таких ключей настолько много, что, скорее всего, она получит иной, чем у Алисы, ключ . Вероятность раскрытия ключа, принадлежащего именно Алисе, настолько мала, что ею можно пренебречь.

Теперь, когда Ева подделает подпись под документом, используя найденный закрытый ключ, подделанная подпись будет отличаться от той подписи, которую поставила бы сама Алиса. При обращении в суд Алиса

предъявит две различных подписи под одним и тем же сообщением и открытый ключ (соответствующий ее закрытому ключу и закрытому ключу, найденному Евой), чтобы доказать подлог. С другой стороны, если Алиса не может предъявить две различные подписи, то подлога не было и Алиса должна отвечать за свою подпись.

Эта схема подписей противостоит взлому Евой подписи Алисы с помощью необычайно мощных вычислительных средств. Она ничего не сможет сделать с более вероятной попыткой Мэллори вломиться в дом Алисы и стащить ее закрытый ключ или с попыткой Алисы подписать документ, а затем "случайно" потерять свой закрытый ключ. Чтобы защититься от упомянутой попытки Мэллори, Алисе стоит купить себе хорошую сторожевую собаку, но подобные рекомендации выходят за рамки криптографии.

Дополнительную теорию и применения подписей с обнаружением подделки можно найти в [1239, 1241, 730, 731].

4.8 Вычисления с зашифрованными данными

Алиса хочет знать решение для некоторой функции $f(x)$ для некоторого конкретного значения x. К несчастью, ее компьютер сломан. Боб хочет вычислить для нее значение $f(x)$, но Алиса не хочет, чтобы Боб знал ее x. Как Алисе позволить Бобу провести вычисление $f(x)$и не сообщить ему x?

Это обычная проблема **вычислений с зашифрованными данными**, также известных как **тайная информация прорицателя**. (Прорицателем является Боб - он отвечает на вопрос.) Для некоторых функций существуют способы решить эту задачу, они обсуждаются в разделе 23.6.

4.9 Вручение битов

Алиса Великолепная, выдающаяся волшебница, сейчас продемонстрирует мощь своего искусства. Она угадает карту, которую выберет Боб до того, как он ее выберет! Следите за тем, как Алиса записывает свое предсказание на кусочке бумаги. Восхищайтесь тем, как Алиса кладет этот кусочек бумаги в конверт и запечатывает его. Дрожите от того, как Алиса отдает запечатанный конверт случайному зрителю. "Выбери карту, Боб, любую карту." Он глядит на нее и показывает карту Алисе и зрителям. Это семерка бубен. Теперь Алиса забирает конверт у зрителя и открывает его. Предсказание, записанное до того, как Боб выбрал карту, сообщает "семерка бубен"! Аплодисменты.

Для успеха этого трюка Алисе нужно подменить конверт в конце фокуса. Однако, криптографические протоколы могут обеспечить защиту от любой ловкости рук. А какая в этом польза? Вот более приземленная история.

Биржевой брокер Алиса хочет убедить инвестора Боба, что ее метод определять перспективные акции заслуживает внимания.

Боб: "Подберите-ка для меня пяток акций. Если на них удастся заработать, я передам свой бизнес вам."

Алиса: "Если я подберу пять акций для вас, вы сможете вложить в них деньги, не заплатив мне. Почему бы мне не показать вам пять акций, которые я подобрал в прошлом месяце?"

Боб: "Откуда я знаю, что вы не подменили результаты вашего выбора, узнав настоящие. Если вы сообщите мне о своем выборе сейчас, я буду уверен, что вы не подмените результат. Я не буду вкладывать деньги в эти акции, пока я не оплачу ваши услуги. Поверьте мне."

Алиса: "Я лучше покажу вам свою подборку акций за прошлый месяц. Я не подменяла их. Поверьте мне."

Алиса хочет передать свое предсказание (т.е., бит или последовательность битов), но не хочет раскрывать свое предсказание до некоторого времени. Боб, с другой стороны, хочет удостовериться, что Алиса не сможет изменить свое мнение после того, как она сделала предсказание.

Вручение битов с помощью симметричной криптографии

Этот протокол вручения битов использует симметричную криптографию:

(1) Боб генерирует строку случайных битов, R, и посылает ее Алисе.

(2) Алиса создает сообщение, состоящее из своего бита, который она хочет вручить, b (в действительности, это может быть и несколько битов), и случайную строку Боба. Она шифрует сообщение некоторым случайным ключом, K, и посылает его обратно Бобу.

$E_K(R,b)$

Эта часть протокола представляет собой процедуру вручения. Боб не может расшифровать сообщение, поэтому он не знает, что за бит прислала Алиса.

Когда для Алисы придет время раскрыть свой бит, протокол продолжается:

(3) Алиса передает Бобу ключ.

(4) Боб расшифровывает сообщения, узнавая бит. Он проверяет свою случайную строку, убеждаясь в правильности бита.

Без случайной строки Боба Алиса может тайно расшифровывать сообщение, посланное Бобу, используя множество ключей, подбирая тот, который позволит при дешифрировании отправленного сообщения изменить врученный бит. Так как у бита только два возможных значения, ее поиски наверняка увенчаются успехом после нескольких попыток. Случайная строка Боба не дает ей использовать этот способ вскрытия, ей придется искать новый ключ, который не только инвертирует врученный бит, но и сохранит нетронутой случайную строку Боба. Если используется достаточно хороший алгоритм шифрования, вероятность удачного поиска чрезвычайно мала. Алиса не может изменить свой бит после его вручения.

Вручение бита с помощью однонаправленных функций

Этот протокол использует однонаправленные функции:

(1) Алиса создает две случайных строки битов, R_1 и R_2.

R_1, R_2

(2) Алиса создает сообщение, состоящее из ее случайных строк и бита, который она хочет вручить (в действительности, это может быть и несколько битов).

(R_1, R_2, b)

(3) Алиса вычисляет однонаправленную функцию для сообщения и посылает результат вместе с одной из случайных строк Бобу.

$H(R_1, R_2, b), R_1$

Это сообщение Алисы является доказательством вручения. Использование однонаправленной функции на этапе (3) мешает Бобу, инвертируя функцию, определить бит.

Когда для Алисы придет время раскрыть свой бит, протокол продолжается:

(4) Алиса отправляет Бобу первоначальное сообщение.

(R_1, R_2, b)

(5) Боб вычисляет однонаправленную функцию для сообщения и сравнивает его и R_1 со значением однонаправленной функции и случайной строкой, полученными на этапе (3). Если они совпадают, то бит правилен.

Преимущество этого протокола перед предыдущим в том, что Бобу не нужно посылать никаких сообщений. Алиса посылает Бобу одно сообщение для вручения бита, а другое - для его раскрытия.

Не нужна и случайная строка Боба, так как результат Алисиного вручения - это сообщение, обработанное однонаправленной функцией. Алиса не может смошенничать и подобрать другое сообщение (R_1, R_2, b'), для которого $H(R_1, R_2, b') = H(R_1, R_2, b)$. Посылая Бобу R_1, она вручает значение b. Если Алиса не сохранит в секрете R_2, то Боб получит возможность вычислить $H(R_1, R_2, b')$ и $H(R_1, R_2, b)$, получая возможность увидеть, что же он получил от Алисы.

Вручение бита с помощью генератора псевдослучайной последовательности

Этот протокол даже проще [1137]:

(1) Боб создает строку случайных битов и посылает ее Алисе.

R_B

(2) Алиса создает стартовую последовательность для генератора псевдослучайных битов. Затем для каждого бита в строке случайных битов Боба она посылает Бобу либо:

(a) выход генератора, если бит Боба равен 0, или

(b) XOR выхода генератора и бита Боба, если Бит Боба равен 1.

Когда для Алисы придет время раскрыть свой бит, протокол продолжается:

(3) Алиса посылает Бобу свою стартовую последовательность.

(4) Боб выполняет этап (2), убеждаясь, что Алиса действует честно.

Если строка случайных битов достаточно длинна, а генератор псевдослучайных битов непредсказуем, мошенничество Алисы практически невозможно.

Blob-объекты

Строки, которые Алиса посылает Бобу для вручения бита, иногда называют **blob-объектами**. Blob-объект - это последовательность битов, хотя протоколы этого и не требуют. Как сказал Жиль Брассар (Gilles Brassard), "Они могли бы быть сделаны и из волшебной пыли, если бы это было полезным " [236]. Blob-объекты обладают следующими четырьмя свойствами:

1. Алиса может вручить blob-объекты. Вручая blob-объект, она вручает бит.
2. Алиса может открыть любой blob-объект, который она вручила. Когда она открывает blob-объект, она может убедить Боба в значении бита, который был вручен вместе с blob-объектом. Следовательно, она не может открыть произвольный blob-объект, например, ноль или единицу.
3. Боб не может знать, каким образом Алиса может открыть blob-объект, который она вручила. Это остается справедливым, даже когда Алиса откроет другие blob-объекты.
4. Blob-объекты не несут никакой информации, кроме вручаемого Алисой бита. Сами по себе blob-объекты, также как и процесс, с помощью которого Алиса вручает и открывает их, не связаны не с чем другим, что Алиса хотела бы сохранить в секрете от Боба.

4.10 Подбрасывание "честной" монеты

Настало время процитировать Джо Килиана (Joe Kilian) [831]:

> Алиса и Боб хотели сыграть в "орла и решку", но монеты у них не было. Алиса предложила простой способ подбрасывать монетку мысленно.
>
> "Сначала вы задумываете случайный бит, затем я задумаю случайный бит. Затем мы выполняем над битами "исключающее или", - предложила она.
>
> "Но если один из нас не будет задумывать биты случайным образом?", - спросил Боб.
>
> "Это не важно. Если хотя бы один из битов действительно случаен, то и "исключающее или" битов должно быть действительно случайным", - ответила Алиса, и после минутного раздумья Боб согласился.
>
> Немного спустя Алиса и Боб наткнулись на книгу по искусственному интеллекту, лежащую на обочине дороги. Алиса, добропорядочная гражданка, сказала: "Один из нас должен подобрать эту книгу и сдать ее в бюро находок". Боб согласился, и предложил использовать их протокол подбрасывания монетки, что бы определить, кто унесет книгу.
>
> "Если полученный бит будет 0, то ты возьмешь книгу, а если 1 - то я", - сказала Алиса. " Какой у тебя бит?"
>
> Боб ответил: "1".
>
> "Ну вот, и у меня такой же", - лукаво заметила Алиса. - "Я думаю, у тебя сегодня неудачный день".
>
> Очевидно, у протокола подбрасывания монетки есть серьезный дефект. Хотя это правда, что "исключающее или" действительно случайного бита, x, и любого независимо распределенного бита, y, дает в результате действительно случайный бит, протокол Алисы не гарантирует, что два бита будут распределены независимо. На самом деле нетрудно убедиться, что не существует мысленного протокола, который позволит двум независимым сторонам подбрасывать "честную" монетку. Алиса и Боб горевали, пока не получили письмо от неизвестного студента с дипломом по криптографии. Информация в письме была слишком теоретической, чтобы ее можно было применить для чего-то земного, но конверт, в котором пришло письмо, оказался чрезвычайно полезным.
>
> Когда Алиса и Боб в следующий раз захотели подбросить монетку, они изменили первоначальный протокол. Сначала бит задумал Боб, но вместо того, чтобы открыть его немедленно, он записывает свой бит на листке бумаги и кладет листок в конверт. Затем Алиса объявляет свой бит. Наконец, Алиса и Боб достают бит Боба из конверта и вычисляют случайный бит. Этот бит уже действительно случаен, независимо от честности играющих. Алиса и Боб получили работающий протокол, с социально значимая мечта криптографов осуществилась, и все они жили долго и счастливо.

Эти конверты выглядят весьма похожими на blob-объекты вручения бита. Когда Мануэль Блам (Manuel Blum) столкнулся с проблемой подбрасывания "честной" монеты по модему [194], он решил ее, используя протокол вручения бита:

(1) Алиса вручает случайный бит, используя любую из схем вручения бита, описанную в разделе 4.9.

(2) Боб загадывает свой бит.

(3) Алиса раскрывает бит Бобу. Боб выигрывает бросок, если он правильно загадал бит.

В общем случае, нам нужен протокол со следующими свойствами:

— Алиса должна "бросить монету" до того, как Боб загадает свой бит.

— Алиса не должна иметь возможности изменить результаты своего броска, узнав бит Боба.

— У Боба не должно быть возможности узнать результат броска перед тем, как он сделает свое предположение.

Существует несколько возможностей выполнить это.

Бросок монеты с помощью однонаправленных функций

Если Алиса и Боб договорятся об однонаправленной функции, протокол прост:

(1) Алиса выбирает случайное число, x. Она вычисляет $y=f(x)$, где $f(x)$ - однонаправленная функция.

(2) Алиса посылает *y* Бобу.

(3) Боб предполагает, что *x* четно или нечетно, и посылает свое предположение Алисе.

(4) Если предположение Боба правильно, результатом броска является "орел", если неправильно - то "решка". Алиса объявляет результат броска монеты и посылает *x* Бобу.

(5) Боб проверяет, что *y=f(x)*.

Безопасность этого протокола обеспечивается однонаправленной функцией . Если Алиса сможет найти *x* и *x′*, такие что *x* - четно, а *x′* - нечетно, и *y=f(x)= f(x′)*, то она каждый раз сможет обманывать Боба. Кроме того, наименьший значащий бит *f(x)* должен быть некоррелирован с *x*. В противном случае Боб сможет обманывать Алису, по крайней мере иногда. Например, если *f(x)* в 75 процентах случаев четна, если *x*, у Боба будет преимущество. (Иногда наименьший значащий бит не является лучшим выбором для использования в приложении, потому что его вычисление может оказаться слишком простым .)

Бросок монеты с помощью криптографии с открытыми ключами

Этот протокол работает как с криптографией с открытыми ключами, так и с симметричной криптографией . Единственное условие - переключение алгоритма. То есть :

$$D_{K_1}(E_{K_2}(E_{K_1}(M))) = E_{K_2}(M)$$

В общем случае это свойство не выполняется для симметричных алгоритмов, но справедливо для некоторых алгоритмов с открытыми ключами (например, RSA с идентичными модулями). Этот протокол:

(1) И Алиса, и Боб создают пары открытый ключ/закрытый ключ.

(2) Алиса создает два сообщения, одно для "орла", а второе - для "решки". Эти сообщения должны включать некоторую случайную строку, чтобы она могла подтвердить их подлинность на последующих этапах протокола. Алиса шифрует оба сообщения своим открытым ключом и посылает их Бобу в произвольном порядке.

$E_A(M_1), E_A(M_2)$

(3) Боб, которые не может прочитать не одно сообщение, случайным образом выбирает одно из них. (Он может посчитать их с помощью "Эники-беники ели вареники", воспользоваться компьютером для взлома протоколы или обратиться к цыганке.) Он шифрует выбранное сообщение своим открытым ключом и посылает его обратно Алисе.

$E_B(E_A(M))$

где M - M_1 или M_2.

(4) Алиса, которая не может прочитать полученное сообщение, расшифровывает его своим закрытым ключом и посылает обратно Бобу.

$D_A(E_B(E_A(M)))= E_B(M_1),$ если $M = M_1$, или $E_B(M_2)$, если $M = M_2$.

(5) Боб расшифровывает сообщение своим закрытым ключом, раскрывая результат броска монеты, и посылает расшифрованное сообщение Алисе.

$D_B(E_B(M_1))$ или $D_B(E_B(M_2))$

(6) Алиса читает результат броска монеты и проверяет, что случайная строка правильна.

(7) Алиса и Боб раскрывают пары своих ключей, чтобы каждый из сторон могла убедиться в отсутствии мошенничества.

Этот протокол самодостаточен. Любая сторона может немедленно обнаружить мошенничество другой, и не требуется третья сторона ни для участия в протоколе, ни в качестве арбитра после завершения протокола . Чтобы посмотреть, как это работает, давайте попытаемся смошенничать .

Если выиграть, смошенничав, хочет Алиса, у нее есть три возможных пути повлиять на результат. Во первых, она может зашифровать два сообщения для "орла" на этапе (2). Боб обнаружит это, когда Алиса раскроет свои ключи на этапе (7). Во вторых, она может использовать какой-то другой ключ для расшифровывания сообщения на этапе (4). Это приведет к бессмыслице, которую Боб и обнаружит на этапе (5). В третьих, она может объявить неправильным сообщение на этапе (6). Боб также обнаружит это на этапе (7), когда Алиса не сможет доказать, неправильность сообщения. Конечно, Алиса может отказаться от участия в протоколе на любом этапе, когда жульничество Алисы станет для Боба очевидным.

Если Боб захочет мошеннически выиграть, его положение ничуть не лучше. Он может неправильно зашифровать сообщение на этапе (3), но Алиса обнаружит обман, взглянув на заключительное сообщение на этапе (6).

Он может заявить, что неправильно выполнил этап (5) из-за какого-то мошенничества со стороны Алисы, но эта форма жульничества вскроется на этапе (7). Наконец, он может послать Алиса сообщение о "решке" на этапе (5), независимо от расшифрованного сообщения, но Алиса сможет немедленно проверить достоверность сообщения на этапе (6).

Бросок монеты в колодец

Интересно отметить, что во всех этих протоколах Алиса и Боб узнают результат броска не одновременно. В каждом протоколе есть момент, когда одна из сторон (Алиса в первых двух протоколах и Боб в последнем) узнает результат броска, но не может изменить его. Эта сторона может, однако, задержать раскрытие результата для второй стороны. Это называется **броском монет в колодец**. Представьте себе высохший колодец. Алиса стоит рядом с колодцем, А Боб - немного подальше. Боб бросает монету, и она падает в колодец. Алиса может теперь заглянуть в колодец и увидеть результат, но она не может спуститься вниз и изменить его. Боб не сможет увидеть результат, пока Алиса не позволит ему подойти и заглянуть в колодец.

Генерация ключей с помощью броска монеты

Реальным применением этого протокола служит генерация сеансового ключа. Протоколы броска монеты позволяют Алисе и Бобу создать случайный сеансовый ключ так, что никто из них не сможет повлиять на то, каким будет этот ключ. Если Алиса и Боб зашифруют свои сообщения, процедура генерации ключа к тому же станет безопасной от злоумышленника.

4.11 Мысленный покер

Протокол, аналогичный протоколу броска монеты с помощью открытых ключей, позволяет Алисе и Бобу играть друг с другом в покер по электронной почте. Алиса вместо создания и шифрования двух сообщений, одного для "орла", а другого - для "решки", создает 52 сообщения M_1, M_2, ..., M_{52}, по числу карт в колоде. Боб случайным образом выбирает пять из них, шифрует своим открытым ключом и посылает обратно Алисе. Алиса расшифровывает сообщения и посылает их обратно Бобу, который расшифровывает их для определения своей "руки". Затем он случайным образом выбирает еще пять сообщений и, не изменяя их, посылает Алисе. Она расшифровывает их, и эти соответствующие карты становятся ее "рукой". В течение игры эта же процедура применяется для сдачи игрокам дополнительных карт. В конце игры Алиса и Боб раскрывают свои карты и пары ключей, чтобы каждый мог убедиться в отсутствии мошенничества.

Мысленный покер с тремя игроками

Покер интереснее, если в игре участвуют несколько человек. Базовый протокол мысленного покера легко может быть распространен на трех и более игроков. В этом случае криптографический алгоритм также должен быть коммутативным.

(1) Алиса, Боб и и Кэрол создают пары открытый ключ/закрытый ключ.

(2) Алиса создает 52 сообщения, по одному для каждой карты колоды. Эти сообщения должны включать некоторую уникальную случайную строку, чтобы Алиса могла проверить их подлинность на последующих этапах протокола. Алиса шифрует все сообщения своим открытым ключом и посылает их Бобу в произвольном порядке.

$E_A(M_n)$

(3) Боб, который не может прочитать не одно сообщение, случайным образом выбирает пять из них. Он шифрует их своим открытым ключом и посылает обратно Алисе.

$E_B(E_A(M_n))$

(4) Боб отправляет Кэрол оставшиеся 47 сообщений.

$E_A(M_n)$

(5) Кэрол, которая не может прочитать не одно сообщение, случайным образом выбирает пять из них. Она шифрует их своим открытым ключом и посылает Алисе.

$E_C(E_A(M_n))$

(6) Алиса, которая не может прочитать ни одно из полученных сообщений, расшифровывает их своим закрытым ключом и посылает обратно Бобу или Кэрол (в соответствии с тем, от кого она их получила).

$D_A(E_B(E_A(M_n)))= E_B(M_n)$

$D_A(E_C(E_A(M_n)))= E_C(M_n)$

(7) Боб и Кэрол расшифровывают сообщения своими ключами, чтобы узнать свои карты

$D_B(E_B(M_n))$

$D_C(E_C(M_n))$

(8) Кэрол случайным образом выбирает пять из оставшихся 42 сообщений и п осылает Алисе.

$E_A(M_n)$

(9) Алиса расшифровывает сообщения, чтобы узнать свои карты.

$D_A(E_A(M_n))$

(10) В конце игры Алиса, Боб и Кэрол раскрывают свои карты и пары ключей, чтобы каждый мог убедиться в отсутствии мошенничества.

Дополнительные карты раздаются подобным же образом. Если карта нужна Бобу или Кэрол, любой из них берет зашифрованную колоду и повторяет протокол с Алисой, Если карта нужна Алисе, то тот, у кого сейчас находится зашифрованная колода, посылает ей случайную карту.

В идеале, этап (10) является обязательным. Свои "руки" в конце протокола должны открывать не все игроки, а только те, которые не спасовали. Так как этап (10) в протокол только для контроля мошенничества, возможны какие-нибудь улучшения.

В покере интересно только, не смошенничал ли победитель. Все остальные могут мошенничать сколько влезет, раз уж они все равно проигрывают. (В действительности это не совсем верно. Кто-то, проигрывая, может собирать данные о стиле игры в покер других игроков.) Итак, взглянем на случаи выигрыша различных игроков.

Если выигрывает Алиса, она раскрывает свою "руку" и свои ключи. Боб может использовать закрытый ключ Алисы для проверки правильности действий Алисы на этапе (2), то есть проверить, что каждое из 52 сообщений соответствует отдельной карте. Кэрол может проверить, что Алиса не лжет о своей "руке", шифруя карты открытым ключом Алисы и проверяя, что они соответствуют шифрованным сообщениям, которые она послала Алисе на этапе (8).

Если выигрывают Боб или Кэрол, победитель раскрывает свои карты и ключи. Алиса может убедиться в правильности карт, проверив свои случайные строки. Она может также убедиться, что сданы были именно эти карты, шифруя их открытым ключом победителя и проверяя, что они совпадают с зашифрованными сообщениями, полученными на этапах (3) или (5).

Этот протокол не защищен от сговора игроков-мошенников. Алиса и другой игрок могут объединиться и безнаказанно вместе надувать третьего игрока. Следовательно, важно проверять все ключи и случайные строки каждый раз, когда игроки раскрывают свои карты. И если вы сидите за виртуальным столом с двумя игроками, которые никогда одновременно не раскрывают свои карты, причем один из них сдает (в предыдущем протоколе это Алиса) кончайте игру.

Все это красиво в теории, но реализовать все это на компьютере весьма непросто. В реализации для трех игроков на трех различных Sparc-станциях восемь часов требуется только для тасования колоды, так что лучше поиграть в настоящий покер [513].

Вскрытия мысленного покера

Криптографы показали, что при использовании этими протоколами покера алгоритма с открытыми ключами RSA происходит небольшая утечка информации [453, 573]. Конкретно, если двоичное представление карт является квадратичным остатком (см раздел 11.3), то зашифрованные карты также являются квадратичным остатком. Это свойство может быть использовано для "крапления" некоторых карт - например, всех тузов. Это даст не много информации о сдачах, но в такой игре как покер даже чуть-чуть информации даст преимущество при длительной игре.

Шафи Голдвассер (Shafi Goldwasser) и Сильвия Микали (Silvia Micali) [624] разработали протокол умственного покера для двух игроков, который решает эту проблему, хотя из-за своей сложности он скорее имеет только теоретическое значение. Обобщенный протокол покера для n игроков, устраняющий проблему утечки информации, был разработан в [389].

Результаты других исследований протоколов игры в покер можно найти в [573, 1634, 389]. Усложненный протокол, позволяющий игрокам не раскрывать своих "рук", приведен в [390]. Дон Копперсмит (Don Coppersmith) рассматривает два способа мошенничества в умственном покере, использующем алгоритм RSA [370].

Анонимное распределение ключей

Хотя непохоже, чтобы кто-нибудь собирался использовать этот протокол для игры в покер по модему, Чарльз Пфлегер (Charles Pfleeger) рассматривает ситуацию, в которой этот тип протокола может оказаться полезным [1244].

Рассмотрим проблему распределения ключей. Если предположить, что люди не могут сами генерировать свои ключи(ключи должны иметь определенную форму, или должны быть подписаны некоторой организацией, или еще что-нибудь подобное), то для генерации и рассылки ключей придется создать Центр распределения ключей (Key Distribution Center, KDC). Проблема в том, что нужно найти такой способ распределения ключей, что никто, включая сервер, не сможет понять, кому какой ключ достался. Следующий протокол решает эту проблему:

(1) Алиса создает пару открытый ключ/закрытый ключ. В этом протоколе она сохраняет в секрете оба ключа.

(2) KDC генерирует непрерывный поток ключей.

(3) KDC шифрует ключи, один за другим, своим открытым ключом.

(4) KDC передает зашифрованные ключи, один за другим, по сети.

(5) Алиса случайным образом выбирает ключ.

(6) Алиса шифрует выбранный ключ своим открытым ключом.

(7) Алиса ждет какое-то время (достаточно большое, чтобы сервер не мог определить, какой ключ она выбрала) и посылает дважды зашифрованный ключ в KDC.

(8) KDC расшифровывает дважды зашифрованный ключ с помощью своего закрытого ключа, получая ключ, зашифрованный открытым ключом Алисы.

(9) Сервер посылает шифрованный ключ обратно Алисе.

(10) Алиса расшифровывает ключ с помощью своего закрытого ключа.

У находящейся где-то в середине протокола Евы нет ни малейшего представления о выбранном Алисой ключе. Она видит непрерывный поток ключей, создаваемых на этапе (4). Когда Алиса посылает ключ серверу на этапе (7), он шифруется ее открытым ключом, который также для этого протокола хранится в секрете. Способа связать это сообщение с потоком ключей у Евы нет. Когда ключ возвращается Алисе сервером на этапе (9), он также зашифрован открытым ключом Алисы. Ключ становится известным, только когда Алиса расшифровывает его на этапе (10).

Если вы используете RSA, в этом протоколе происходит утечка информации со скоростью, по меньшей мере, один бит на сообщение. Причиной этого снова являются квадратичные остатки. Если вы собираетесь использовать этот способ для распределения ключей, убедитесь, что эта утечка не приведет к каким-либо последствиям. Кроме того, поток ключей, создаваемый KDC должен быть достаточно большим, чтобы противостоять вскрытию грубым взломом. Конечно же, если Алиса не может верить KDC, то она не должна пользоваться его ключами. Мошенничающий KDC может предусмотрительно записывать все создаваемые им ключи. Тогда он сможет найти среди них ключ, выбранный Алисой.

Этот протокол также предполагает, что Алиса будет действовать честно. При использовании RSA существует ряд действий, которые может предпринять Алиса, чтобы получить больше информации, чем ей удалось бы при другом методе шифрования. В нашем сценарии эта проблема не существенна, но при других обстоятельствах она может стать важной.

4.12 Однонаправленные сумматоры

Алиса является членом организации "Заговорщики". Иногда ей приходится встречаться с другими членами в плохо освещенных ресторанах и шептать секреты налево и направо. Беда в том, что рестораны настолько плохо освещены, что она не может быть уверена, что человек, сидящий напротив нее за столом, тоже член организации.

"Заговорщики" могут выбирать из нескольких решений. Каждый может носить с собой список членов организации. Это влечет за собой две следующих проблемы. Во первых, теперь каждый должен носить с собой большую базу данных, и, во вторых, им придется как следует охранять этот список членов. Другим способом является использование идентификационных карт, выпущенных надежным секретарем. Дополнительным преимуществом этого способа является то, что и посторонние смогут проверять членов организации (всякие скидки в местной бакалейной лавке), до для этого нужен надежный секретарь. Никому из "заговорщиков" нельзя доверять до такой степени.

Новым решением является использование **однонаправленного сумматора** [116]. Это что-то похожее на од-

нонаправленные хэш-функции, для которых выполняется требование коммутативности. То есть, можно хэш ировать базы данных членов организации в произвольном порядке и получать одно и то же значение . Более того, можно добавлять новых членов в хэш-таблицу и получать новое хэш-значение, снова не зависящее от порядка .

Итак, вот что делает Алиса. Она выполняет расчет, используя множество всех имен членов организации, о тличных от нее. Затем она сохраняет это полученное значение вместе со своим именем. Боб и другие члены д елают то же самое. Теперь, когда Алиса и Боб встречаются в плохо освещенном ресторане, они просто обмен иваются друг с другом вычисленными значениями и именами. Алиса убеждается, что результат, получаемый при добавлении имени Боба к значению Алисы, совпадает с результатом, получаемым при добавлении имени Ал исы к значению равно значению Боба. Боб делает то же самое. Теперь они оба знают, что собеседник - также член организации. И в то же время никто не сможет определить личности других членов организации .

Более того, рассчитанные значения каждого члена могут быть выданы посторонним. Тогда Алиса сможет подтвердить свое членство постороннему (возможно, для членской скидки в буфете местной контрразведки), не показывая ему весь список членов.

Новых членов можно добавить просто послав по кругу новые имена. К несчастью, удалить члена можно только единственным путем: всем членам рассылается новый список и они пересчитывают свои значения . Но "заговорщикам" придется выполнять это действие только при отставке кого-то из членов, мертвые члены могут остаться в списке. (Странно, но это не создает проблемы.)

Это разумная идея применяется в ряде приложений, когда вы хотите достичь эффекта цифровой подписи без использования централизованной системы подписей.

4.13 Раскрытие секретов "все или ничего"

Представьте себе, что Алиса - бывший агент бывшего Советского Союза, а теперь безработная . Чтобы заработать, она продает секреты. Каждый, готовый заплатить названную цену, может купить секрет . У нее даже есть каталог. Все ее секреты с аппетитными названиями упорядочены по номерам: "Где Джимми Хоффа?", "Кто тайно контролирует Трехстороннюю комиссию?", "Почему Борис Ельцин всегда выглядит, как будто он прогл отил живую лягушку?", и т.д.

Алиса не хочет отдавать два секрета по цене одного и не показывает даже части информации, касающейся любого из секретов. Боб, потенциальный покупатель, не хочет платить за кота в мешке . Он также не хочет сообщать Алисе, какие из секретов ему нужны. Это не ее дело, и, кроме того, тогда Алиса сможет добавить в свой каталог пункт "Секреты, которыми интересуется Боб".

Протокол покера не работает в этом случае, так как в конце этого протокола Алиса и Боб должны раскрыть свои карты друг другу. К тому же, существуют трюки, с помощью которых Боб может узнать сразу несколько секретов.

Решение называется **раскрытием секретов "все или ничего"** (all-or-nothing disclosure of secrets, **ANDOS**) [246], потому что если Боб получил любую информацию о любом из секретов Алисы, то он потерял возможность узнать что-либо еще о других ее секретах.

В криптографической литературе можно найти различные протоколы ANDOS. Некоторые из них обсуждаются в разделе 23.9.

4.14 Условное вручение ключей

Вот отрывок из введения в тему Сильвио Микали [1084]:

> Сегодня подслушивание с разрешения суда является эффективным методом отдавать преступников в руки правосудия . По нашему мнению еще более важно, что это также предотвращает дальнейшее распространение преступления, удерживая от использования обычных сетей связи с незаконными целями. Следовательно, существует обоснованное беспокойство, что ра спространение криптографии с открытыми ключами может быть на руку преступным и террористическим организациям . Действительно, во многих законах предполагается, что соответствующие правительственные ведомства при определенных усл овиях, оговоренных законом, должны иметь возможность получить открытый текст любого обмена информацией по общедо ступным сетям. В настоящее время many это может быть трактоваться, как требование к законопослушным гражданам либо (1) *использовать слабые криптосистемы* - т.е., криптосистемы, которые соответствующие власти (а также кто угодно!) см огут вскрыть с помощью умеренных усилий, или (2) заранее *сообщать свои секреты* властям. Не удивительно, что такая альтернатива законно встревожила многих заинтересованных граждан, создавая в результате мнение, что тайна личности дол жна стоять над национальной безопасностью и отправлением закона .

Условное вручение ключей является сутью продвигаемых правительством США программы Clipper и Стандарта условного шифрования (Escrowed Encryption Standard). Проблема в том, чтобы и обеспечить тайну личн ости, и в то же время позволить разрешенное судом подслушивание .

Escrowed Encryption Standard обеспечивает безопасность с помощью защищенного оборудования. У каждой микросхемы шифрования уникальный идентификационный номер (ID) и секретный ключ. Этот ключ делится на

две части и хранится, вместе с ID, двумя различными организациями условного вручения. Всякий раз, когда микросхема шифрует файл данных, она сначала шифрует сеансовый ключ уникальным секретным ключом. Затем она передает зашифрованный сеансовый ключ и свой ID по каналу связи. Когда правоохранительные органы хотят расшифровать поток информации, зашифрованной одной из этих микросхем, они извлекают из потока ID, получают соответствующие ключи из организаций условного вручения, объединяют их с помощью операции XOR, расшифровывают сеансовый ключ и затем используют его для дешифрирования потока сообщений. Для защиты от мошенников в эту схему введены дополнительные усложнения, подробно описанные в разделе 24.16. Аналогичная схема может быть реализована и программно с использованием криптографии с открытыми ключами [77, 1579, 1580, 1581].

Микали называет свою идею **честной криптосистемой** [1084,10851. (Говорят, что правительство США заплатило Микали $1000000 за использование его патентов [1086, 1087] в своем стандарте Escrowed Encryption Standard, затем патент Микали купил Банковский трест.) В таких криптосистемах закрытый ключ делится на части и распределяется среди различных организаций. Как и схема с совместным использованием секрета, эти организации могут объединиться и восстановить закрытый ключ. Однако, части ключа обладают дополнительным свойством - их правильность может быть проверена независимо без восстановления закрытого ключа.

Алиса может создать свой собственный закрытый ключ и распределить его части среди n доверительных собственников. Ни один из них не может восстановить закрытый ключ Алисы. Однако каждый может проверить, что его часть - это правильная часть закрытого ключа. Алиса не может послать кому-то из доверительных собственников строку случайных битов и надеяться улизнуть. Если судебные власти разрешат подслушивание, соответствующие правоохранительные органы смогут воспользоваться постановлением суда для того, чтобы n доверительных собственников выдали свои части. Собрав все n частей, власти восстановят закрытый ключ и смогут подслушивать линии связи Алисы. С другой стороны, чтобы получить возможность восстановить ключ Алисы и нарушить ее тайну личности, Мэллори придется купить всех n доверительных собственников.

Вот как работает этот протокол:

(1) Алиса создает пару закрытый ключ/открытый ключ. Она разбивает закрытый ключ на несколько открытых и закрытых частей.

(2) Алиса посылает открытую часть и соответствующую закрытую часть каждому из доверительных собственников. Эти сообщения должны быть зашифрованы. Она также посылает открытый ключ в KDC.

(3) Каждый из доверительных собственников независимо выполняет вычисления над своими закрытой и открытой частями, чтобы убедиться в их правильности. Каждый доверительный собственник хранит закрытую часть в каком-нибудь надежном месте и отправляет открытую часть в KDC.

(4) KDC выполняет иное вычисление для открытых частей и открытого ключа. Убедившись, что все правильно, он подписывает публичный ключ и отправляет его обратно Алисе или помещает в какую-нибудь базу данных.

При наличии постановления суда о подслушивании каждый из доверительных собственников передает свою часть в KDC, и KDC получает возможность восстановить закрытый ключ. До этой передачи ни KDC, ни кто-либо из доверительных собственников не может самостоятельно восстановить закрытый ключ, для восстановления ключа нужны все доверительные собственники.

Любой алгоритм с открытыми ключами можно сделать "честным" подобным образом. Ряд конкретных алгоритмов рассматривается в разделе 23.10. В работах Микали [1084, 1085] обсуждаются пути объединения описанного с пороговой схемой, чтобы для восстановления закрытого ключа требовалось некоторое подмножество доверительных собственников (например, трое из пяти). Он также показывает, как объединить это с рассеянной передачей (см. раздел 5.5) так, чтобы доверительные собственники не знали, чей закрытый ключ восстанавливается.

"Честные" криптосистемы несовершенны. Преступник может использовать такую систему, применяя подсознательный канал (см. раздел 4.2.), чтобы вставить другой секретный ключ в свою информацию. Таким образом он может безопасно обмениваться информацией с кем-нибудь еще, используя подсознательный ключ и совершенно не волнуясь по поводу разрешенного судом подслушивания. Данная проблема решается другим протоколом, который называется **отказоустойчивым условным вручением ключей** [946, 833]. Этот алгоритм и протокол описывается в разделе 23.10.

Политика условного вручения ключей

Помимо правительственных планов относительно условного вручения ключей распространяются и коммерческие системы с условным вручением ключей. Возникает очевидный вопрос: какое преимущество от условного вручения ключей получает пользователь?

Ну, на самом деле никакого. Пользователь не получает от условного вручения ключей ничего такого, чего он

и сам не смог бы обеспечить. Он и сам может создать резервную копию ключей, если захочет (см. раздел 8.8). Условное вручение ключей гарантирует, что полиция сможет подслушивать его разговоры или читать файлы данных, даже когда они шифрованы. Оно гарантирует, что NSA сможет подслушивать его международные звонки - без всякого ордера - хотя они и шифрованы. Может ему будет разрешено использовать такую криптографию с теми странами, для которых сейчас установлены запреты, но это сомнительное преимущество.

Недостатки условного вручения ключей весьма ощутимы. Пользователю приходится верить в безопасность действия организаций, занятых условным вручением ключей также, как и в честность занятых этим людей. Ему придется верить, что политика соответствующих организаций останется неизменной, правительство не поменяет законы, и те, кто имеет полномочия вскрыть его ключ, будут делать это по закону и с полной ответственностью. Вообразите нападение террористов на Нью-Йорк, какие бы ограничения не были бы сметены полицией, чтобы остановить последствия?

Трудно представить себе, что эти условные схемы шифрования, как говорят их защитники, будут использоваться без принуждения извне. Следующим очевидным шагом будет запрет на использование всех других способов шифрования. Это, вероятно, единственный способ добиться коммерческого успеха этой системы, и это, определенно, единственный способ заставить технически грамотных преступников и террористов использовать ее. Пока не ясно, насколько трудно будет объявить не-условную криптографию вне закона, или как это повлияет на криптографию как на академическую дисциплину. Как я могу исследовать программно ориентированные алгоритмы криптографии, не имея доступа к программному обеспечению устройств не-условного шифрования, нужна ли мне будет специальная лицензия?

И другие законные вопросы. Как условно врученные ключи повлияют на ответственность пользователей, должна ли становиться известной зашифрованная информация? Если правительство США пытается защитить органы условного вручения, не будет ли это косвенным свидетельством того, что если секрет скомпрометирован либо пользователем, либо органами условного вручения, то виновником будет признан пользователь?

Что если база данных главной службы условного вручения ключей, все равно государственной или коммерческой, будет украдена? Что, если правительство США попытается ненадолго скрыть этот факт? Ясно, что все эти вопросы повлияют на желание пользователей пользоваться условным вручением ключей. Если использование не будет добровольным, то пара скандалов вызовет рост политического давления с целью либо сделать использование подобных систем добровольным, либо ввести новые сложные правила в этой отрасли.

Еще более опасным будет скандал, когда выяснится, что годами под наблюдением находился политический оппонент текущей администрации или некий громкоголосый критик спецслужб и полицейских ведомств. Это сильно настроит общественное мнение против условного шифрования.

Если ключи подписей будут шифроваться тем же способом, что и ключи шифрования, возникнут дополнительные моменты. Допустимо ли для властей использовать ключи подписей для проведения операции против подозреваемого преступника? Будет ли признана судом подлинность подписей, основанных на ключах с условным вручением? Чем в действительности будут владеть пользователи, если власти действительно используют их ключи пользователей для подписи какого-то невыгодного контракта, для поддержки определенных отраслей промышленности, или просто, чтобы украсть деньги?

Глобальное распространение криптографии рождает дополнительные вопросы. Будут ли схемы условного вручения ключей совместимы в различных странах? Захотят ли транснациональные корпорации смириться с существованием в каждой стране своих условно врученных ключей, совместимых с различным местным законодательством? Без обеспечения совместимости исчезает одно из пропагандируемых преимуществ схемы с условным вручением ключей (международное использование мощных средств криптографии).

Что если ряд стран не примет на веру надежность организаций, связанных с условным вручением ключей? Как будут пользователи вести свои дела в этих странах? Будут ли признаны судами их электронные контракты, или тот факт, что ключи их подписей условно хранятся в США, позволит им утверждать где-нибудь в Швейцарии, что этот электронный контракт мог подписать кто-то другой? Или для людей, которые ведут дела в подобных странах, будут специальные исключения?

А что делать с промышленным шпионажем? Где гарантии, что страны, занимающиеся сейчас промышленным шпионажем для своих важнейших или государственных предприятия, не воспользуются для этого системами с условным вручением ключей? В самом деле, так как ни одна страна не собирается позволять другим странам следить за своими разведывательными операциями, распространение условного шифрования возможно приведет к увеличению подслушивания.

Даже если страны, в которых соблюдаются гражданские права, будут использовать условность такого шифрования только для законного преследования преступников и террористов, где-нибудь этим обязательно воспользуются для выслеживания диссидентов, шантажа политических оппонентов, и т.п. Цифровые линии связи предоставляют возможность гораздо более тщательно, чем это было возможно в аналоговом мире, контролировать действия граждан, их мнения, Digital communications offer the opportunity to do a much more thorough lob of

monitoring citizens' actions, opinions, доходы и объединения.

Не ясно, не будет ли через 20 лет продажа системы с условным вручением ключей Турции или Китаю пох одить на продажу электрических дубинок Южной Африке в 1970 году или на строительство химического завода в Ираке в 1980 году. Даже хуже, легкое и незаметное подслушивание линий связи может искусить многие пр авительства, которые раньше, возможно, этим и не занимались, следить за корреспонденцией своих граждан . И нет гарантии, что либеральные демократии устоят перед подобным искушением .

Глава 5
Развитые протоколы

5.1 Доказательства с нулевым знанием

А вот другая история:

Алиса: "Я знаю пароль компьютера Федеральной Резервной Системы, компоненты секретного соуса МакДональдс и оде р-жание 4-го тома Дональда Кнута".

Боб: "Нет, ты не знаешь".

Алиса: "Нет, я знаю".

Боб: "Не знаешь".

Алиса: "Нет, знаю".

Боб: "Докажи".

Алиса: "Хорошо, я скажу тебе". Она шепчет Бобу на ухо.

Боб: "Это интересно. Теперь я тоже это знаю и собираюсь рассказать это все *Вашингтон Пост*".

Алиса: "Ооооой".

К несчастью, обычно Алиса может доказать что-нибудь Бобу, только рассказав ему все. Но тогда он тоже получит все сведения. Затем Боб может выложить полученные сведения кому угодно, и Алиса ничего не сможет с этим поделать. (В литературе для описания этих протоколов часто используются различные персонажи . Пегги обычно доказывает, а Виктор проверяет. Именно эти имена появляются в используемых примерах вместо Ал и-сы и Боба.)

Используя однонаправленные функции, Пегги сможет провести **доказательство с нулевым знанием** [626]. Этот протокол доказывает Виктору, что у Пегги действительно есть информация, но не дает Виктору не мале й-шей возможности узнать, что это за информация.

Эти доказательства принимают форму интерактивного протокола. Виктор задает Пегги ряд вопросов. Если Пегги знает секрет, то она ответит на все вопросы правильно. Если секрет ей неизвестен, у нее есть некоторая вероятность - 50 процентов в следующих примерах - ответить правильно . После примерно 10 вопросов Виктор убедится, что Пегги знает секрет. Но ни один из вопросов или ответов не даст Виктору ни малейших сведений об информации Пегги, но докажет знание Пегги этой информации .

Базовый протокол с нулевым знанием

Жан-Жак Кискатер (Jean-Jacques Quisquater) и Луи Гилу (Louis Guillou) поясняют нулевое знание историей о пещере [1281]. У пещеры, показанной на 4-й, есть секрет. Тот, кто знает волшебные слова может открыть п о-тайную дверь между C и D. Для всех остальных оба прохода ведут в тупик.

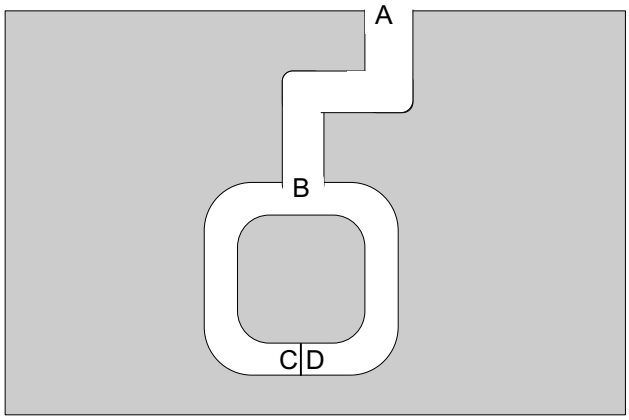


Рис. 5-1. Пещера нулевого знания

Пегги знает секрет пещеры. Она хочет доказать свое знание Виктору, но не хочет раскрывать волшебных слов. Вот как она убеждает его:

(1) Виктор находится в точке A.

(2) Пегги проходит весь путь по пещере, либо до точки C, либо до точки D.

(3) После того, как Пегги исчезнет в пещере, Виктор переходит в точку B.

(4) Виктор кричит Пегги, спрашивая ее либо о:

(a) или выйти из левого прохода

(b) выйти из правого прохода.

(5) Пегги исполняет его просьбу, при необходимости используя волшебные слова, чтобы отпереть дверь.

(6) Пегги и Виктор повторяют этапы (1) - (5) *n* раз.

Предположим, что у Виктора есть видеокамера, и он записывает все, что видит. Он записывает, как Пегги исчезает в пещере, записывает, как он сам кричит, указывая, где Пегги должна появиться, записывает как Пегги появляется. Он записывает все *n* тестов. Если он покажет эту видеозапись Кэрол, поверит ли она, что Пегги знает волшебные слова, отпирающие дверь? Нет. А что если Пегги и Виктор заранее договорились, что Виктор будет кричать, а Пегги будет делать вид, что она прошла весь путь. Тогда она будет каждый раз выходить из указанного Виктором места, не зная волшебных слов. Или они могли сделать по другому. Пегги входит в один из проходов и Виктор случайным образом выкрикивает свои просьбы. Если Виктор угадывает правильно, хорошо, если нет - они вырежут эту попытку из видеозаписи. В любом случае Виктор может получить видеозапись, показывающую в точности ту последовательность, которая получилась бы, если бы Пегги знала волшебные слова.

Этот опыт показывает две вещи. Во первых, Виктор не может убедить третью сторону в правильности доказательства. И во вторых, данный протокол является протоколом с нулевым знанием. Если Пегги не знает волшебных слов, то очевидно, что Виктор не узнает ничего из просмотра видеозаписи. Но так как нет способа отличить реальную видеозапись от подделанной, то Виктор не может ничего узнать из реального доказательства - это и есть нулевое знание.

Методика, используемая в этом протоколе, называется **разрезать и выбрать** из-за того, что она похож на классический протокол честного деления чего-либо:

(1) Алиса делит некую вещь пополам.

(2) Боб выбирает одну из половин себе.

(3) Алиса забирает оставшуюся половину.

В интересах Алисы честно разделить на этапе (1), потому что Боб выберет на этапе (2) ту половину, которая ему больше нравится. Майкл Рабин (Michael Rabin) первым использовал в криптографии технику "разрезать и выбрать" [1282]. Понятия **интерактивного протокола** и нулевого знания были формализованы позже [626, 627].

Протокол "разрезать и выбрать" работает, потому что Пегги не может несколько раз подряд угадывать, откуда Виктор попросит ее выйти. Если Пегги не знает секрета, он может выйти только из того прохода, в который она зашла. В каждом раундепротокола ее вероятность (иногда называемая **аккредитацией**) угадать, с какой стороны Виктор попросит ее выйти, составляет 50 процентов, поэтому ее вероятность обмануть Виктора также равна 50 процентам. Вероятность обмануть его в двух раундах составит 25 процентов, а во всех *n* раундах - один шанс из 2^n. После 16 раундов у Пегги 1 шанс из 65536 обмануть Виктора. Виктор может уверенно предположить, что если все 16 доказательств Пегги правильны, то она действительно знает тайные слова, открывающие дверь между точками C и D. (Аналогия с пещерой несовершенна. Пегги может просто входить с одной стороны и выходить с другой, протокол "разрезать и выбрать" не нужен. Однако, он необходим с для нулевого знания с математической точки зрения.)

Предположим, что Пегги известна некоторая информация, которая является решением трудной проблемы. Базовый протокол нулевого знания состоит из нескольких раундов.

(1) Пегги использует свою информацию и случайное число для преобразования одной трудной проблемы в другую, изоморфную оригинальной проблеме. Затем она использует свою информацию и случайное число для решения новой трудной проблемы.

(2) Пегги вручает решение новой проблемы, используя схему вручения бита.

(3) Пегги раскрывает Виктору новый экземпляр проблемы. Виктор не может использовать эту новую проблему для получения информации о первоначальной проблеме или ее решении.

(4) Виктор просит Пегги либо

(a) доказать ему, что новая и старая проблема изоморфны (т.е., два различных решения для двух связанных проблем), либо

(b) открыть решение, полученное на этапе (2) и доказать, что это решение новой проблемы.

(5) Пегги исполняет его просьбу.

(6) Пегги и Виктор повторяют этапы (1) - (5) n раз.

Помните видеокамеру в протоколе для пещеры? Здесь вы можете сделать то же самое . Виктор может записать обмен между ним и Пегги. Он не сможет использовать эту запись для убеждения Кэрол, но он всегда может сговориться с Пегги с целью создать имитатор, который подделывает информацию Пегги . Этот аргумент может быть использован, чтобы доказать, что используется доказательство с нулевым знанием .

Математическая основа доказательства этого типа сложна. Проблемы и случайное преобразование должны выбираться осторожно, чтобы Виктор не получил никакой информации о решении оригинальной проблемы, даже после многих повторений протокола . Не все трудные проблемы можно использовать для доказательств с нулевым знанием, но большинство из них .

Изоморфизм графа

Объяснение этого понятия, пришедшего из теории графов [619, 622], может занять некоторое время. Граф представляет собой сеть линий ,соединяющих различные точки. Если два графа идентичны во всем, кроме имен точек, они называются **изоморфными**. Для очень больших графов доказательство их изоморфности может потребовать веков компьютерного времени , это одна из так называемых **NP-полных** проблем, рассматриваемых в разделе 11.1.

Предположим, что Пегги знает об изоморфности двух графов , G_1 и G_2. Следующий протокол докажет Виктору знание Пегги:

(1) Пегги случайным образом тасует G_1, получая другой граф, H, который изоморфен G_1. Так как Пегги знает об изоморфизме H и G_1, то ей также известен изоморфизм между H и G_2. Для любого другого поиск изоморфизма между H и G_1 или H и G_2 является такой же трудной задачей, как и поиск изоморфизма между G_1 и G_2.

(2) Пегги посылает H Виктору.

(3) Виктор просит Пегги либо

(a) доказать, что H и G_1 изоморфны, либо

(b) доказать, что H и G_2 изоморфны.

(4) Пегги исполняет его просьбу. Она либо:

(a) доказывает, что H и G_1 изоморфны, не доказывая, что H и G_2 изоморфны, либо

(b) доказывает, что H и G_2 изоморфны, не доказывая, что H и G_1 изоморфны.

(5) Пегги и Виктор повторяют этапы (1) - (4) n раз.

Если Пегги не знает об изоморфизме между G_1 и G_2, она не сможет создать граф H, изоморфный обоим графам. Она может создать либо граф, который изоморфен G_1, либо граф, который изоморфен G_2. Как и в предыдущем примере у нее только 50 шансов из 100 угадать, какое доказательство потребует от нее Виктор на этапе (3).

Этот протокол не дает Виктору никакой полезной информации, помогающей ему из ответов Пегги установить изоморфизм между G_1 и G_2. Так как Пегги для каждого нового раунда протокола генерирует новый граф H, Виктор не сможет получить информацию независимо от того, из скольких раундов будет состоять их протокол . Он не сможет из ответов Пегги установить изоморфизм между G_1 и G_2.

В каждом раунде Виктор получает новое случайное преобразование H, вместе с изоморфизмом между H и G_1 или G_2. Виктор может также создать что-то подобное самостоятельно. Так как Виктор может создать имитацию протокола, это действительно доказательство с нулевым знанием .

Гамильтоновы циклы

Вариант этого примера был впервые представлен Мануэлем Блюмом (Manuel Blum) [196]. Пегги знает кружной, продолжительный путь вдоль линий графа, который проходит через каждую точку только один раз . Этот путь называется **гамильтоновым циклом**. Поиск гамильтонова цикла является другой тяжелой задачей . У Пегги есть эта информация - она, возможно, получила ее создав граф с конкретным гамильтоновым циклом - и она хочет доказать Виктору, что эта информация ей известна .

Пегги знает гамильтонов цикл графа , G. Виктору известен G, но не его гамильтонов цикл . Пегги хочет доказать Виктору, что она знает гамильтонов цикл, не раскрывая самого цикла . Вот как она должна действовать:

(1) Пегги случайным образом преобразовывает G. Она передвигает точки и изменяет их метки, создавая но-

вый граф, *H*. Поскольку *G* и *H* топологически изоморфны (т.е., это один и тот же граф), если ей известен гамильтонов цикл *G*, то она легко может найти гамильтонов цикл *H*. Если она не сама создает *H*, определение изоморфизма между двумя графами будет являться другой сложной проблемой, решение которой также потребует веков компьютерного времени. Затем она шифрует *H*, получая *H'*. (Должно использоваться вероятностное шифрование каждой строчки *H*, то есть, шифрованный 0 или шифрованная 1 для каждой линии *H*.)

(2) Пегги передает Виктору копию *H*.

(3) Виктор просит Пегги либо:

(a) доказать ему, что *H'* - это зашифрованная изоморфная копия *G*, либо

(b) показать ему гамильтонов цикл для *H*.

(4) Пегги исполняет его просьбу. Она либо:

(a) доказывает, что *H'* - это зашифрованная изоморфная копия *G*, раскрывая преобразования и расшифровывая все, не показывая гамильтонов цикл для *G* или *H*, либо

(b) показывает гамильтонов цикл для *H*, расшифровывая только те строки, которые образуют гамильтонов цикл, не доказывая, что *H* и *G* топологически изоморфны.

(5) Пегги и Виктор повторяют этапы (1) - (4) *n* раз.

Если Пегги не обманывает, она сможет предъявить Виктору одно из доказательств на этапе (3). Однако, если гамильтонов цикл для *G* ей неизвестен, она не сможет создать зашифрованный граф *H'*, который удовлетворяет обоим доказательствам. Лучшее, что она может сделать - это создать или граф, изоморфный *G*, или граф с таким же числом точек и линий и правильным гамильтоновым циклом. Хотя ее шансы угадать, какое доказательство потребует Виктор на этапе (3), составляют 50 процентов, Виктор может повторить протокол достаточное число раз, убеждаясь, что Пегги знает гамильтонов цикл для *G*.

Параллельные доказательства с нулевым знанием

В базовом протоколе с нулевым знанием используется *n* обменов информацией между Пегги и Виктором. Почему бы не выполнить их параллельно:

(1) Пегги использует свою информацию и *n* случайных чисел для преобразования трудной проблемы в *n* различных изоморфных проблем. Затем она с помощью своей информации и случайных чисел решает *n* новых трудных проблем.

(2) Пегги вручает решение *n* новых трудных проблем.

(3) Пегги раскрывает Виктору эти *n* новых трудных проблем. Виктор не может воспользоваться этими новыми проблемами для получения информации об оригинальных проблемах или их решении.

(4) Для каждой новой трудной проблемы Виктор просит Пегги либо

(a) доказать ему, что старая и новая проблемы изоморфны, либо

(b) раскрыть решение, врученное на этапе (2), и доказать, что оно является решением данной новой проблемы.

(5) Пегги исполняет его просьбу для каждой новой проблемы.

К несчастью, все не так просто. Этот протокол, в отличие от предыдущего, не обладает такими же свойствами нулевого знания. На этапе (4) Виктор может потребовать, чтобы доказательство было представлено в виде значения однонаправленной хэш-функции всех значений, врученных на первом этапе, делая невозможным имитацию записи протокола. Это тоже нулевое знание, но другого рода. На практике оно представляется безопасным, но никто не знает, как это доказать. Мы действительно знаем только то, что при определенных условиях определенные протоколы для определенных проблем могут быть выполнены параллельно без потери свойства нулевого знания [247, 106, 546, 616].

Неинтерактивные доказательства с нулевым знанием

Кэрол невозможно убедить, потому что она не участвует в интерактивном процессе протокол. Для убеждения Кэрол и других заинтересованных лиц нам нужен неинтерактивный протокол.

Для неинтерактивных доказательств с нулевым знанием был придуман ряд протоколов [477, 198, 478, 197], которые не требуют непосредственного взаимодействия. Пегги может опубликовать их и, таким образом, доказать свое знание всем, у кого найдется время это проверить

Базовый протокол похож на параллельное доказательство с нулевым знанием, но место Виктора занимает

однонаправленная хэш-функция:

(1) Пегги использует свою информацию и n случайных чисел для преобразования трудной проблемы в n различных изоморфных проблем. Затем она с помощью своей информации и случайных чисел решает n новых трудных проблем.

(2) Пегги вручает решение n новых трудных проблем.

(3) Пегги использует все эти вручения в качестве входа для однонаправленной хэш-функции. (В конце концов эти вручения - не что иное, как строки битов.) Затем она сохраняет первые n битов полученного значения однонаправленной хэш-функции.

(4) Пегги берет n битов, полученных на этапе (3). По очереди для каждой n-ой трудной проблемы она берет n-ый бит и

(a) если бит равен 0, доказывает, что старая и новая проблемы изоморфны, либо

(b) если бит равен 1, раскрывает решение, врученное на этапе (2), и доказывает, что оно является решением данной новой проблемы.

(5) Пегги опубликовывает все решения, врученные на этапе (2), и все доказательства, полученные на этапе (4).

(6) Виктор, Кэрол и все остальные заинтересованные лица проверяют, что этапы (1)-(5) выполнены правильно.

Это впечатляет: Пегги может опубликовать некоторые данные, которые не содержат никакой информации о ее секрете, но могут кого угодно убедить в существовании самого секрета. Этот протокол может быть использован проверка определена как вычисление однонаправленной хэш-функции первоначальных сообщений и подписываемого сообщения.

Эта схема работает, потому что однонаправленная хэш-функция действует как беспристрастный генератор случайных битов. Чтобы мошенничать, Пегги нужно уметь предсказывать результат однонаправленной хэш-функции. (Помните, если решение трудной проблемы ей неизвестно, она может сделать на этапе (4) либо (a), либо (b), но не оба действия одновременно.) Если она каким-то образом узнает, выполнение какого действия потребует от нее однонаправленная хэш-функция, то она сможет смошенничать. Однако, Пегги не сможет заставить однонаправленную хэш-функцию выдать определенный бит или догадаться, какой бит будет получен. Однонаправленная хэш-функция по сути является заменителем Виктора в случайном выборе одного из двух доказательств на этапе (4).

В неинтерактивном протоколе должно быть гораздо больше итераций в последовательности запрос/ответ. Пегги, а не Виктор, отбирает трудные проблемы с помощью случайных чисел. Она может подбирать различные проблемы, следовательно, и различные векторы вручения, до тех пор, пока хэш-функция не выдаст что-то, нужное Пегги. В интерактивном протоколе 10 итераций - вероятность мошенничества Пегги составит 1 шанс из 2^{10} (1 из 1024) - может быть достаточно. Однако, для неинтерактивных доказательств с нулевым знанием этого не хватит. Помните, что Мэллори всегда может выполнить на этапе (4) либо (a), либо (b). Он может, выполняя этапы (1)-(3), попытаться догадаться, что его попросят сделать, и посмотреть, правильно ли его предположение. Если нет, он попробует снова и снова. Сделать 1024 предположения на компьютере нетрудно. Для предотвращения такого вскрытия грубым взломом для неинтерактивных протоколов нужно 64 или даже 128 итераций.

Главная идея состоит в использовании однонаправленной хэш-функции - Пегги не может предсказать выход хэш-функции, потому что она не может предсказать ее вход. Вручения, используемые на входе, становятся известны только после решения новых проблем.

Общие замечания

Блюм (Blum) доказал, что любая математическая теорема может быть преобразована в граф, такой, что доказательство теоремы будет эквивалентно доказательству существования гамильтонова цикла для этого графа. В общем виде то, что для любого NP-полного утверждения есть доказательство с нулевым знанием, использующее однонаправленные функции и, следовательно, хорошие алгоритмы шифрования, доказано в [620]. Любое математическое доказательство может быть преобразовано в доказательство с нулевым знанием. Используя эту методику, исследователь может доказать миру, что ему известно доказательство конкретной теоремы, не раскрывая самого решения. Блюм мог опубликовать свои результаты, не раскрывая их.

Также существуют **доказательства с минимальным раскрытием** [590]. Для доказательства с минимальным раскрытием выполняются следующие свойства:

1. Пегги не может обмануть Виктора. Если Пегги не знает доказательства, ее шансы убедить Виктора в

том, что доказательство ей известно, пренебрежимо малы.

2. Виктор не может обмануть Пегги. Он не получает ни малейшего намека на доказательство кроме того факта, что доказательство известно Пегги. В частности, Виктор не может продемонстрировать доказательство никому другому, не доказав все сам с самого начала.

У доказательств с нулевым знанием есть дополнительное условие :

3. Виктор не узнает от Пегги ничего такого, чего он не смог бы узнать и самостоятельно кроме того факта, что доказательство известно Пегги.

Существует заметная математическая разница между доказательствами с минимальным раскрытием и доказательствами с нулевым знанием. Это различие находится вне рамок данной книги, но более искушенные читатели могут проштудировать другую литературу. Понятия изложены в in [626, 619, 622]. Дальнейшая проработка этих идей, основанная на различных математических предположениях, выполнена в [240, 319, 239].

Существуют различные типы доказательств с нулевым знанием :

— **Совершенное**. Существует имитатор, который создает стенограммы, полностью соответствующие реальным стенограммам (примеры с гамильтоновым циклом и изоморфизмом графов).

— **Статистическое**. Существует имитатор, который создает стенограммы, полностью соответствующие реальным стенограммам, кроме фиксированного числа исключений.

— **Вычислительное**. Существует имитатор, который создает стенограммы, неотличимые от реальных.

— **Неиспользующее**. Имитатора может и не быть, но мы можем доказать, что Виктор не узнает никакой информации из доказательства (параллельный пример)

Годы тяжелой работы, как теоретической, так и прикладной, присели к появлению доказательств с минимальным раскрытием и нулевым знанием. Майк Берместер (Mike Burmester) и Иво Десмедт изобрели широковещательно интерактивное доказательство, где владелец секрета может широковещательно передавать большой группе контролеров интерактивное доказательство с нулевым знанием [280]. Криптографы доказали, что *все*, что может быть доказано с помощью интерактивного доказательства, может быть доказано и с помощью интерактивного доказательства с нулевым знанием [753, 137].

Хорошей обзорной статьей по данной теме является [548]. Дополнительные математические подробности, варианты, протоколы и приложения ищите в [590, 619, 240, 319, 620, 113,241, 152, 8, 660, 238, 591, 617, 510, 592, 214, 104, 216, 832, 97, 939, 622, 482, 615, 618, 215, 476, 71]. *Много чего* было написано по этому вопросу.

5.2 Использование доказательства с нулевым знанием для идентификации

В реальном мире для доказательств подлинности часто используются физические символы: паспорта, водительские права, кредитные карточки и т.д. Эти символы содержат что-то, связывающее их с конкретным человеком: обычно фотографию или подпись, но с той же вероятностью это может быть отпечаток пальца, снимок сетчатки глаза или рентгеновский снимок челюсти. Как было бы здорово делать что-то подобное цифровым образом?

Использовать доказательства с нулевым знанием для доказательства идентичности было впервые предложено Уриелем Файгом (Uriel Feige), Амосом Фиатом (Amos Fiat) и Ади Шамиром [566, 567]. Закрытый ключ Алисы становится функцией ее "идентичности". Используя доказательство с нулевым знанием, она доказывает, что она знает свой закрытый ключ и, таким образом, свою идентичность. Соответствующие алгоритмы можно найти в разделе 23.11.

Это очень многообещающая идея. Она позволяет человеку доказать свою личность без использования физических символов. Однако, она не совершенна. Вот примеры возможных злоупотреблений.

Проблема гроссмейстера

Вот как Алиса, даже не зная правил шахмат, может обыграть гроссмейстера. (Иногда это называется проблемой гроссмейстера.) Она посылает вызов Гарри Каспарову и Анатолию Карпову, предлагая играть в одно время, в одном и том же месте, но в раздельных комнатах. Она играет белыми против Каспарова и черными против Карпова. Ни один гроссмейстер не знает о другом.

Карпов, играя белыми, делает свой ход первым. Алиса записывает ход и идет в комнату к Каспарову. Играя белыми, она делает тот же ход на доске Каспарова. Каспаров делает свой первый ход черными. Алиса записывает ход, идет в комнату к Карпову и делает тот же ход. Это продолжается, пока она не выигрывает одну из партий, проигрывая другую, или обе партии кончаются вничью.

На самом деле Каспаров играет с Карповым, а Алиса просто посредник, повторяющий ходы одного гросс-

мейстера на доске другого. Однако, если Карпов и Каспаров не знают о присутствии друг друга, каждый из них будет поражен игрой Алисы.

Этот способ мошенничества может быть использовать против доказательства личности с нулевым знанием [485, 120]. Когда Алиса доказывает свою личность Мэллори, Мэллори может одновременно доказать Бобу, что он-то и есть Алиса.

Обман, выполненный мафией

Обсуждая свой протокол идентификации с нулевым знанием, Ади Шамир сказал [1424]: "Я могу ходить в принадлежащий мафии магазин хоть миллион раз подряд, а они все еще не смогут выдать себя за меня."

Вот как мафия сможет это сделать. Алиса ест в ресторанчике Боба, принадлежащем мафии. Кэрол делает покупки в дорогом ювелирном магазине Дэйва. Боб и Кэрол - мафиози, переговаривающиеся по потайному радиоканалу. Алиса и Дэйв не подозревают о мошенничестве.

Когда Алиса поела и собралась платить и доказывать свою личность Бобу, Боб подает сигнал Кэрол, что пора начинать. Кэрол выбирает бриллианты подороже и собирается доказывать свою личность Дэйву. Теперь, пока Алиса доказывает свою личность Бобу, тот подает сигнал Кэрол, и та выполняет тот же протокол с Дэйвом. Когда Дэйв задает вопрос по протоколу, Кэрол сообщает этот вопрос Бобу, а Боб задает его Алисе. Когда Алиса отвечает, Боб передает правильный ответ Кэрол. По сути, Алиса просто доказывает свою личность Дэйву, а Боб и Кэрол просто, находясь внутри протокола, передают сообщения туда-сюда. Когда протокол завершается, Алиса доказала свою личность Дэйву и заплатила за дорогие бриллианты (с которыми Кэрол теперь и исчезнет).

Обман, выполненный террористами

Если Алиса хочет объединиться с Кэрол, то они также могут провести Дэйва. В этом протоколе Кэрол - известная террористка. Алиса помогает ей въехать в страну. Дэйв - офицер-пограничник, Алиса и Кэрол общаются по тайному радиоканалу.

Когда Дэйв задает Кэрол вопросы в соответствии по протоколу с нулевым знанием, Кэрол передает их Алисе, которая и отвечает на вопросы. Кэрол повторяет эти ответы Дэйву. Carol recites these answers to Dave. По сути, Свою личность Дэйву доказывает Алиса, а Кэрол выступает в роли линии связи. Когда протокол завершается, Дэйв считает, что Кэрол - это Алиса, и разрешает ей въехать в страну. Спустя три дня Кэрол всплывает у правительственного здания вместе с микроавтобусом, набитом взрывчаткой.

Предлагаемые решения

Оба описанных мошенничества возможны, так как заговорщики используют тайный радиоканал. Одним из способов предотвратить мошенничество является проведение процедуры идентификации в клетке Фарадея, блокирующей электромагнитное излучение. В примере с террористом это гарантирует, что Кэрол не получит ответов от Алисы. В примере с мафией Боб может построить фальшивую клетку Фарадея в своем ресторане, но у ювелира-то клетка будет работать, и Боб и Кэрол не смогут обмениваться сообщениями. Для решения проблемы гроссмейстера Алиса должна сидеть на своем стуле до конца игры.

Тотас Бот (Thomas Both) и Иво Десмедт предложили другое решение, использующее точные часы [148]. Если каждый этап протокола должен происходить в заданное время, у мошенников не останется времени для обмена информацией. В случае с проблемой гроссмейстера это соответствует предложению ограничить время обдумывания хода одной минутой - у Алисы не останется времени бегать из комнаты в комнату. В истории с мафией у Боб и Кэрол не хватит времени передавать друг другу ответы и вопросы.

Обман с несколькими личностями

Существуют и другие способы злоупотребить доказательствами идентичности с нулевым знанием, также рассматриваемые в [485, 120]. В ряде реализаций проверка при регистрации человеком своего ключа не производится. Следовательно, у Алисы может быть несколько закрытых ключей и, таким образом, несколько личностей. Это может здорово помочь ей, если она захочет мошенничать с налогами. Алиса также может совершить преступление и скрыться. В первых, она создает несколько личностей, одна из которых не используется. Затем, она использует эту личность для совершения преступления так, чтобы свидетель идентифицировал ее как эту личность. Затем, она немедленно прекращает пользоваться этой личностью. Свидетель знает личность преступника, но Алиса никогда больше не будет использовать эту личность - ее невозможно проследить.

Для защиты от этого нужны механизмы, обеспечивающие, чтобы у каждого человека была только одна личность. В [120] авторами предлагается причудливая идея защищенных от воровства детей, которые не могут быть клонированы, и у которых есть уникальный номер, являющийся частью их генетического кода. Они также предложили присваивать каждому ребенку личность при рождении. (Действительно, родителям придется сде-

лать это, так как иначе ребенок может быть украден.) Этим тоже легко злоупотребить - родители могут создать для родившегося ребенка несколько личностей. В конце концов, уникальность личности основана на доверии.

Прокат паспортов

Алиса хочет поехать в Заир, но правительство этой страны не дает ей визы. Кэрол предлагает сдать свою личность Алисе "напрокат". (Первым это предложил Боб, но возник ряд очевидных проблем.) Кэрол продает Алисе свой закрытый ключ и Алиса едет в Заир, выдавая себя за Кэрол.

Кэрол получает не только плату за свою личность, но и идеальное алиби. Она совершает преступление, пока Алиса находится в Заире. "Кэрол" доказала свою личность в Заире, как она могла совершить преступление дома?

Конечно, развязаны руки и у Алисы. Она может совершить преступление либо перед отъездом, либо сразу же после возвращения, около дома Кэрол. Сначала она покажет, что она - Кэрол (имея закрытый ключ Кэрол, ей не составит труда сделать это), затем она совершит преступление и убежит. Полиция будет искать Кэрол. Кэрол будет утверждать, что сдала свою личность напрокат Алисе, но кто поверит в такую невероятную историю?

Проблема в том, что Алиса доказывает не свою личность, а то, что ей известна некоторая секретная информация. Именно связь между этой информацией и личностью и служит предметом злоупотребления. Решение защищенных от воровства детей защитило бы от такого мошенничества, как и создание полицейского государства, в котором все граждане должны очень часто доказывать свою личность (в конце дня, на каждом углу и т.д.). Помочь решить эту проблему могут биометрические методы - отпечатки пальцев, снимки сетчатки глаза, запись голоса и т.п.

Доказательство членства

Алиса хочет доказать Бобу, что она является членом суперсекретной организации, но не хочет раскрывать свою личность. Эта проблема, близкая проблеме доказательства личности, но отличающаяся от нее, была изучена в [887, 906, 907, 1201, 1445]. Ряд решений связан с проблемой групповых подписей (см. раздел 4.6).

5.3 Слепые подписи

Важным свойством протоколов цифровой подписи является знание подписывающим содержания подписываемого документа. Это хорошее свойство, даже когда хочется обратного.

Мы можем пожелать, чтобы люди подписывали документы, даже не зная их содержания. Существуют способы, когда подписывающий может не точно, но *почти* знать, что он подписывает. Но все по порядку.

Полностью слепые подписи

Боб - государственный нотариус. Алиса хочет, чтобы он подписал документ, не имея ни малейшего представления о его содержании. Боб не отвечает за содержание документа, он только заверяет, что нотариально засвидетельствовал его в определенное время. Он собирается действовать по следующему плану:

(1) Алиса берет документ и умножает его на случайное число. Это случайное число называется **маскирующим множителем**.

(2) Алиса посылает замаскированный документ Бобу.

(3) Боб подписывает замаскированный документ.

(4) Алиса удаляет маскирующий множитель, получая оригинальный документ, подписанный Бобом.

Этот протокол работает только, если функция подписи и умножение коммутативны. Если нет, то помимо умножения существуют и другие способы изменить документ. Несколько подходящих алгоритмов приведены в разделе 23.12. А сейчас, для простоты математики остановимся на умножении.

Может ли боб смошенничать? Может ли он получить какую-нибудь информацию о том, что пописывает? Если множитель осторожности действительно случаен и делает замаскированный документ действительно случайным, то нет. Замаскированный документ, подписываемый Бобом на этапе, (2) ничем не похож на оригинальный документ Алисы. Замаскированный документ с подписью Боба на нем на этапе (3) ничем не похож на подписанный документ этапа (4). Даже если Боб заполучит документ со своей подписью после окончания протокола, он не сможет доказать (себе или кому-то другому), что он подписал его в этом конкретном протоколе. Он знает, что его подпись правильна. Он может, как и любой другой, проверить свою подпись. Однако, у него нет никакой возможности связать подписанный документ и любую информацию, полученную при выполнении протокола. Если он подписал, используя этот протокол, миллион документов, у него не будет способа узнать когда какой документ он подписал. Полностью слепые подписи обладают следующими свойствами:

1. Подпись Боба под документом правильна и служит доказательством того, что Боб подписал этот д окумент. Она убедит Боба в том, что он подписал этот документ, когда документ будет впоследствии показан Бобу. Она также обладает всеми свойствами цифровых подписей, обсуждаемых в разделе 2.6.

2. Боб не может связать подписанный документ с процессом подписания документа. Даже если у него хранятся записи обо всех сделанных им слепых подписях, он не сможет определить, когда он подп исал конкретный документ.

У Евы, находящейся между Алисой и Бобом и следящей за протоколом, информации еще меньше, чем у Б оба.

Слепые подписи

С помощью протокола полностью слепых подписей Алиса может заставить Боба подписать что-нибудь вр оде: "Боб должен Алисе миллион долларов", "Боб должен Алисе своего первого ребенка", "Боб должен Алисе ящик шоколада". Возможности бесконечны, и поэтому во многих приложениях этот протокол бесполезен..

Однако, существует способ, с помощью которого Боб может узнать, что он подписывает, вместе с тем сохр аняя полезные свойства слепых подписей. Центральным моментом этого протокола является техника "разрезать и выбрать". Рассмотрим следующий пример. Множество людей каждый день въезжают в некую страну, и Д епартамент иммиграции хочет удостовериться, что они не ввозят кокаин. Служащие могут обыскивать каждого, но вместо этого используется вероятностное решение - обыскивается каждый десятый въезжающий. Подвергается досмотру имущество одного человека из десяти, остальные девять пропускаются беспрепятственно. Постоянные контрабандисты в большинстве случаев проскакивают незамеченными, но с вероятностью 10 процентов их ловят. И если судебная система работает эффективно, наказание за единственную поимку на месте престу пления более чем перевешивает выгоды девяти удачных попыток.

Если Департамент иммиграции захочет повысить вероятность поимки контрабандистов, служащим придется обыскивать больше людей, захочет понизить вероятность - можно будет обыскивать меньше людей. Управляя вероятностями, можно контролировать эффективность протокола при поимке контрабандистов.

Протокол слепой подписи работает аналогичным образом. Боб получает большую пачку различных замаск ированных документов. Он **откроет**, например, все кроме одного и затем подпишет последний.

Посмотрите на замаскированный документ как на лежащий в конверте. Процесс маскировки документа можно рассматривать как помещение документа в конверт, а процесс удаления множителя маскировки - как вскрытие конверта. Когда документ спрятан в конверт, никто не сможет его прочитать. Документ подписывается с помощью кусочка копировальной бумаги, помещенной в конверт: Когда подписывающий ставит свою по дпись на конверте, с помощью кусочка копировальной бумаги эта подпись ставится и под документом.

В следующем сценарии действует группа агентов контрразведки. Их личности засекречены, даже само Управление контрразведки не знает, кто они такие. Директора Управления хочет выдать каждому агенту подп исанный документ следующего содержания: "Податель этого подписанного документа, (вставьте имя, под кот орым действует агент), обладает полной дипломатической неприкосновенностью". У каждого из агентов есть свой список псевдонимов, поэтому Управление не может просто раздать подписанные документы. Агенты не хотят передавать свои псевдонимы в Управление, так как враг мог вскрыть компьютер Управления. С другой стороны, Управление не хочет слепо подписывать документы, предоставленные агентом. Хитрый агент может представить сообщение, подобное следующему: "Агент (имя) вышел в отставку, и ему назначена ежегодная пенсия в миллион долларов. Подписано, Президент". В этом случае могут быть полезны слепые подписи.

Предположим, что у каждого агента по 10 псевдонимов, выбранных ими самими и больше никому неизвес тных. Предположим также, что агентам все равно, под каким именем они получат дипломатическую неприко сновенность. Также предположим, компьютер управления называется Agency's Large Intelligent Computing Engine (Большая Интеллектуальная Вычислительная Машина Управления), или ALICE, а нашим конкретным агентом является Bogota Operations Branch (Сектор операций в Боготе): BOB.

(1) BOB готовит *n* документов, каждый из которых использует различный псевдоним, дающих ему дипломатическую неприкосновенность.

(2) BOB маскирует каждый из документов отличным маскирующим множителем.

(3) BOB отправляет *n* документов ALICE.

(4) ALICE случайным образом выбирает *n-1* документ и просит BOB'а прислать маскирующий множитель для каждого из выбранных документов.

(5) BOB посылает ALICE соответствующие маскирующие множители.

(6) ALICE открывает (т.е., удаляет маскирующий множитель) *n-1* документ и убеждается в том, что они ко р-

ректны - и не являются разрешением на выплату пенсии.

(7) ALICE подписывает оставшийся документ и посылает его BOB'у.

(8) Агент удаляет маскирующий множитель и читает свой новый псевдоним: "Малиновая полоса." Подписанный документ дает ему дипломатическую неприкосновенность под этим именем.

Этот протокол надежно защищен от мошенничества BOB'а. Чтобы смошенничать, он должен точно угадать, какой документ ALICE не будет проверять. Вероятность этого - 1 шанс из *n* - не слишком велика. ALICE знает это и чувствует себя уверенно, подписывая документ, который она не сможет проверить. Для этого документа рассматриваемый протокол полностью совпадает с предыдущим протоколом полностью слепой подписи, сохраняя все свойства анонимности.

Существует трюк, который еще больше уменьшает вероятность мошенничества BOB'а. На этапе (4) ALICE случайным образом выбирает *n/2* документов для проверки, и BOB присылает ей соответствующий маскирующие множители на этапе (5). На этапе (7) ALICE перемножает все непроверенные документы и подписывает получившийся мегадокумент. На этапе (8) BOB удаляет все маскировочные множители. Подпись ALICE будет правильной, только если ею подписано произведение *n/2* идентичных документов. Чтобы смошенничать, BOB'у нужно точно угадать, какое подмножество документов будет проверять ALICE. Вероятность этого гораздо ниже, чем вероятность угадать единственный документ, который ALICE не проверяла.

BOB может смошенничать по другому. Он может создать два различных документа, один из которых ALICE согласна подписать, а другой - нет. Затем он может попытаться найти два различных маскирующих множителя, которые преобразуют указанные документы к одинаковому виду. Таким образом, если ALICE захочет проверить документ, BOB передаст ей маскирующий множитель, преобразующий документ к невинному виду. Если ALICE не захочет просмотреть документ и подпишет его, он применит тот маскирующий множитель, который преобразует замаскированный подписанный документ в документ, являющийся целью мошенничества. Хотя теоретически это и возможно, математика конкретных алгоритмов делает пренебрежимо малой вероятность для BOB'а найти такую пару. Действительно, она может быть столь низкой, как и вероятность Боба создать необходимую подпись под произвольным документом самостоятельно. Этот вопрос обсуждается ниже в разделе 23.12.

Патенты. Владельцем патентов на ряд особенностей слепых подписей является Чаум (Chaum) (см. 4-й).

Табл. 5-1. Патенты Чаума на слепые подписи

№ патента США	Дата	Название
4759063	19.07.88	Blind Signature Systems [323] (Системы слепых подписей)
4759064	19.07.88	Blind Unanticipated Signature Systems [324] (Системы слепых неожиданных подписей)
4914698	03.03.90	One-Show Blind Signature Systems [326] (Системы слепых подписей, показываемых один раз)
4949380	14.08.90	Returned-Value Blind Signature Systems [328] (Системы слепых подписей с возвращаемым значением)
4991210	05.02.91	Unpredictable Blind Signature Systems [331] (Системы непредсказуемых слепых подписей)

5.4 Личностная криптография с открытыми ключами

Алиса хочет отправить Бобу безопасное сообщение. Она не хочет получать свой открытый ключ с сервера ключей, она не хочет проверять подпись некоторой заслуживающей доверия третьей стороны на сертификате своего открытого ключа, и она даже не хочет хранить открытый ключ Боба в своем компьютере. Она хочет просто послать ему безопасное сообщение.

Эту проблему решают **личностные** криптосистемы, иногда называемые системами с Неинтерактивным разделением ключей (Non-Interactive Key Sharing, NIKS) [1422]. Открытый ключ Боба основывается на его имени и сетевом адресе (телефонном номере, почтовом адресе или чем-то подобном). В обычной криптографии с открытыми ключами Алисе нужен подписанный сертификат, связывающий личность Боба и его открытый ключ. В личностной криптографии открытый ключ Боба *и есть* его личность. Это действительно свежая идея является почти совершенной для почтовой системы - Если Алиса знает адрес Боба, она может безопасно посылать ему почту, что делает криптографию прозрачной, насколько это вообще возможно.

Система основана на выдаче Трентом ключей пользователям в зависимости от их личности. Если закрытый

ключ Алисы будет скомпрометирован, ей придется изменить одно из свойств, определяющих ее личность. Серьезной проблемой является проектирование системы таким образом, чтобы сговор нечестных пользователей не мог привести к подделке ключа.

При разработке математики таких схем, обеспечение безопасности которых оказалось зверски сложным, был выполнен большой объем работы - главным образом в Японии. Многие предложенные решения содержат выбор Трентом случайного числа для каждого пользователя - по моему, это угрожает самой идее таких систем. Ряд алгоритмов, рассматриваемых в главах 19 и 20, могут быть личностными. Подробности алгоритмов и криптосистем можно найти в [191, 1422, 891, 1022, 1515, 1202, 1196, 908, 692, 674, 1131, 1023, 1516, 1536, 1544, 63, 1210, 314, 313, 1545, 1539, 1543, 933, 1517, 748, 1228]. Алгоритм, который не использует случайных чисел, описан в [1035]. Система, обсуждаемая в [1546, 1547, 1507], ненадежна против вскрытия с использованием выбранного открытого ключа, то же самое можно сказать и о системе, предложенной как NIKS-TAS [1542, 1540, 1541, 993, 375, 1538]. По правде говоря, среди предложенного нет ничего одновременно практичного и безопасного.

5.5 Рассеянная передача

Криптограф Боб безнадежно пытается разложить на множители 500-битовое число n. Он знает, что оно является произведением пяти 100-битовых чисел, и ничего больше. (Вот проблема. Если он не восстановит ключ, ему придется работать сверхурочно, и он не попадет на еженедельную игру с Алисой в мысленный покер.) Что же делать? И вот появляется Алиса:

> "Мне посчастливилось узнать один из множителей числа", - говорит она, - "и я продам его тебе за 100 долларов. По доллару за бит." Показывая свою серьезность, она собирается использовать схему вручения бита, вручая каждый бит отдельно.
>
> Боб заинтересован, но только за 50 долларов. Алиса не хочет сбрасывать цену и предлагает продать Бобу половину битов за половину стоимости. "Это заметно сократит тебе работу", -.
>
> "Но как я узнаю, что твое число действительно является множителем n. Если ты покажешь мне число и позволишь мне убедиться, что оно действительно является множителем, я соглашусь с твоими условиями", - говорит Боб.
>
> Они в патовой ситуации. Алиса не может убедить Боба в том, что она знает сомножитель n, не раскрыв его, а Боб не хочет покупать 50 битов, которые вполне могут оказаться бесполезными.

Эта история, утащенная у Джо Килиана [831], вводит понятие **рассеянной передачи**. Алиса передает Бобу группу сообщений. Боб получает некоторое подмножество этих сообщений, но Алиса не знает, какие из сообщений Боб получил. Однако, это не полностью решает проблему. Когда Боб получит случайную половину битов, Алисе придется убеждать его, используя доказательство с нулевым знанием, что она послала часть множителя n.

В следующем протоколе Алиса посылает Бобу одно из двух сообщений. Боб получает сообщение, но какое - Алиса не знает.

(1) Алиса генерирует две пары открытый ключ/закрытый ключ, всего четыре ключа. Она посылает оба открытых ключа Бобу.

(2) Боб выбирает ключ симметричного алгоритма (например, DES). Он выбирает один из открытых ключей Алисы и шифрует с его помощью свой ключ DES. Он посылает шифрованный ключ Алисе, не сообщая, какой из ее открытых ключей он использовал для шифрования.

(3) Алиса дважды расшифровывает ключ Боба, используя оба своих закрытых ключа. В одном из случаев она использует правильный ключ и успешно расшифровывает ключ DES, присланный Бобом. В другом случае она использует неправильный ключ и получает бессмысленную последовательность битов, которая, тем не менее, похожа на случайный ключ DES. Так как ей неизвестен правильный открытый текст, она не может узнать, какой из ключей правилен.

(4) Алиса зашифровывает каждое из своих сообщений каждым из ключей, полученных ею на предыдущем этапе (один из которых - настоящий, а другой - бессмысленный), и посылает оба сообщения Бобу.

(5) Боб получает сообщения Алисы, одно из которых зашифровано правильным ключом DES, а другое - бессмысленным ключом DES. Когда Боб расшифровывает каждое из этих сообщений своим ключом DES, он может прочитать одно из них, а другое будет для него выглядеть полной бессмыслицей.

Теперь у Боба есть два сообщения Алисы, и Алиса не знает, какое из них Бобу удалось успешно расшифровать. К несчастью, если протокол остановится на этом этапе, Алиса сможет смошенничать. Необходим еще один этап.

(6) Когда протокол завершится, и станут известны оба возможных результата передачи, Алиса должна передать Бобу свои закрытые ключи, чтобы он убедиться в отсутствии мошенничества. В конце концов, она могла зашифровать на этапе (4) обоими ключами одно и то же сообщение.

В этот момент, конечно же Боб сможет узнать и второе сообщение.

Протокол надежно защищен от взлома со стороны Алисы, потому что у нее нет возможности узнать, какой из двух ключей DES является настоящим. Оба из них она использует для шифрования своих сообщений, но Боб может успешно расшифровать только одно из них - до этапа (6). Протокол защищен и от взлома со стороны Боба, потому что до этапа (6) он не сможет получить закрытый ключ Алисы, чтобы определить ключ DES, которым зашифровано другое сообщение. На вид этот протокол может показаться просто усложненным способом бросать "честную" монету по модему, но он интенсивно используется во многих сложных протоколах.

Конечно же, ничто не может помешать Алисе послать Бобу два совершенно бессмысленных сообщения: "Мяу-мяу" и "Ты молокосос". Этот протокол гарантирует, что Алиса передаст Бобу одно из двух сообщений, но нет никакой гарантии, что Боб захочет получить любое из них.

В литературе можно найти и другие протоколы рассеянной передачи. Некоторые из них неинтерактивны, т.е. Алиса публикует свои два сообщения, а Боб может прочесть только одно из них. Он может сделать это, когда захочет, ему не нужно для этого связываться с Алисой [105].

В действительности на практике никто не использует протокол рассеянной передачи, но это понятие является важным блоком при построении других протоколов. Хотя существует много типов рассеянной передачи - у меня есть два секрета, а вы получаете один, у меня есть n секретов, а вы получаете один, у меня есть один секрет, который вы получаете с вероятностью 1/2 и так далее - все они эквивалентны [245, 391, 395].

5.6 Рассеянные подписи

Честно говоря, я не могу придумать, чего их можно использовать, но существует два типа рассеянных подписей [346]:

1. У Алисы есть *n* различных сообщений. Боб может выбрать одно из них, чтобы Алиса его подписала, и у Алисы не будет способа узнать, что же она подписала.
2. У Алисы есть единственное сообщение. Боб может выбрать один из *n* ключей, которым Алиса подпишет сообщение, и Алиса не сможет узнать, какой ключ она использовала.

Идея изящна, я уверен, что где-нибудь она найдет применение.

5.7 Одновременная подпись контракта

Подпись контракта с помощью посредника

Алиса и Боб хотят заключить контракт. Они достигли согласия на словах, но никто не хочет ставить свою подпись, пока не поставлена подпись другого. При личной встрече это не вызывает затруднений - оба подписывают вместе. На расстоянии они могут обратиться к посреднику.

(1) Алиса подписывает копию контракта и посылает ее Тренту.

(2) Боб подписывает копию контракта и посылает ее Тренту.

(3) Трент посылает сообщение и Алисе, и Бобу, сообщающее, что другой партнер подписал контракт.

(4) Алиса подписывает две копии контракта и посылает их Бобу.

(5) Боб подписывает обе копии контракта, сохраняет одну для себя, и посылает другую Алисе.

(6) Алиса и Боб сообщают Тренту, что у каждого из них есть подписанная обоими партнерами копия контракта.

(7) Трент уничтожает свои две копии контракта, с единственной подписью под каждым.

Этот протокол работает, потому что Трент защищает любую из сторон от мошенничества другой. Если Боб попытается отказаться от подписи под контрактом на этапе (5), Алиса может обратиться к Тренту за копией контракта, уже подписанного Бобом. Если Алиса попытается отказаться от подписи под контрактом на этапе (4), Боб может сделать то же самое. Когда Трент сообщает, что он получил оба контракта на этапе (3), Алиса и Боб узнают, что другой партнер уже подписал контракт. Если Трент не получит оба контракта на этапах (1) и (2), он уничтожает имеющуюся у него копию, и ни одна из сторон не связана более обязательствами контракта.

Одновременная подпись контракта без посредника (лицом к лицу)

Если Алиса и Боб встречаются лицом к лицу, они могут подписать контракт следующим образом [1244]:

(1) Алиса пишет первую букву своего имени и передает контракт Бобу.

(2) Боб пишет первую букву своего имени и передает контракт Алисе.

(3) Алиса пишет вторую букву своего имени и передает контракт Бобу.

(4) Боб пишет вторую букву своего имени и передает контракт Алисе.

(5) Это продолжается до тех пор, пока Алиса и Боб не напишут свои имена полностью.

Если пренебречь очевидной проблемой протокола (имя Алисы длиннее имени Боба), то он работает дост аточно хорошо. Написав только одну букву из подписи, Алиса знает, что никакой судья не станет заставлять ее выполнять условия контракта. Но написанная буква - это акт доброй воли, и Боб отвечает аналогичным дейс твием.

Когда каждая из сторон напишет несколько букв подписи, судья вероятно сможет убедиться, что обе стороны подписали контракт. Хотя, если вглядеться, ситуация весьма туманна. Конечно же, то, что контракт не вступает в силу после написания первых букв, так же очевидно как и то, что контракт становится действующим после того, как стороны напишут свои имена. В каком месте протокола стороны оказываются связанными контрактом? Написав половину своих имен? Две трети? Три четверти?

Так как ни Алиса, ни Боб не знают точно, с какого момента начинает действовать контракт, то у каждого из них на протяжении всего протокола есть опасение, что для него или для нее контракт уже вступил в силу . Не существует этапа протокола, на котором Боб смог бы сказать : "Вы написали четыре буквы подписи, а я только три. Вы связаны контрактом, а я нет." У Боба нет причин прекращать выполнение протокола. Более того, чем дольше стороны выполняют протокол, тем больше вероятность того, что судья решит, что контракт вступил в силу. И снова, нет причины прерывать протокол. В конце концов, они оба хотели подписать контракт, они пр осто не хотели подписывать его раньше другого партнера.

Одновременная подпись контракта без посредника (без личной встречи)

В этом протоколе используется такая же неопределенность [138]. Алиса и Боб по очереди движутся детскими шажками к подписанию протокола.

В этом протоколе Алиса и Боб обмениваются рядом подписанных сообщений вида : "Я согласен, что с вероятностью p я связан условиями контракта."

Получатель сообщения может предъявить его судье и, с вероятностью p, судья признает контракт подписа нным.

(1) Алиса и Боб согласовывают дату окончания подписания контракта .

(2) Алиса и Боб договариваются о различии вероятностей, которым они собираются пользоваться. Например, Алиса может решить, что ее вероятность быть связанной условиями контракта не должна превышать в е роятность Боба больше, чем на 2 процента. Обозначим различие Алисы как a, различие Боба - как b.

(3) Алиса посылает Бобу подписанное сообщение с вероятностью $p = a$.

(4) Боб посылает Алисе подписанное сообщение с $p = a + b$.

(5) Пусть p - это вероятность из сообщения, полученного Алисой от Боба на предыдущем этапе . Алиса посылает Бобу подписанное сообщение с $p' = p + a$ или 1, смотря что меньше.

(6) Пусть p - это вероятность из сообщения, полученного Бобом от Алисы на предыдущем этапе . Боб посылает Алисе подписанное сообщение с $p' = p + b$ или 1, смотря что меньше.

(7) Алиса и Боб продолжают выполнять этапы (5) и (6) до тех пор, пока они оба не получат сообщения с $p = 1$ или пока не наступит согласованная на этапе (1) дата .

По мере выполнения протокола и Алиса, и Боб соглашаются, что они оказываются связанными контрактом со все большей и большей вероятностью. Например, Алиса может определить свое a как 2 процента, а Боб свое b - как 1 процент. (Лучше бы они выбрали большие приращения, а то мы застрянем на этом месте.) В первом сообщении Алиса сообщает, что она связана контрактом с вероятностью 2 процента . Боб может ответить, что он связан контрактом с вероятностью 3 процента. Следующее сообщение Алисы может утверждать, что она связ ана контрактом с вероятностью 5 процента и так далее, пока вероятности обоих не достигнут 100 процентов.

Если и Алиса, и Боб завершили протокол к оговоренной дате , то все прекрасно. В противном случае, любая из сторон может предъявить контракт судье вместе с подписанным последним сообщением другой стороны. Прежде, чем просмотреть контракт, судья случайным образом выбирает число между 0 и 1. Если это число меньше вероятности, подписанной второй стороной, то обе стороны связаны контрактом. Если это число больше вероятности, подписанной второй стороной, то обе стороны не связаны контрактом. (Затем судья сохраняет использованное число на случай решения другого вопроса, касающегося того же контракта .) Именно это и означает "быть связанным контрактом с вероятностью p".

Это базовый протокол, но могут использоваться и дополнительные усложнения . Судья может принимать ре-

шение в отсутствие одной из сторон. Решение судьи связывает контрактом либо обе стороны, либо ни одну из них. Не существует ситуации, когда одна из сторон связана контрактом, а другая - нет. Более того, протокол завершится, как только одна из сторон захочет иметь хоть немного большую, чем другая, вероятность быть связанной контрактом.

Одновременная подпись контракта без посредника (с помощью криптографии)

Этот криптографический протокол использует тот же принцип детских шажков [529]. Для определенности используется DES, хотя вместо него может быть любой симметричный алгоритм.

(1) И Алиса, и Боб случайным образом выбирают $2n$ ключей DES, сгруппированных попарно. В парах нет ничего особенного, это просто способ группировки для данного протокола.

(2) И Алиса, и Боб создают n пар сообщений, L_i и R_i, например, "Это левая половина моей i-ой подписи" и "Это правая половина моей i-ой подписи". Идентификатор, i, меняется от 1 до n. В каждое сообщение, вероятно, также будет входить цифровая подпись контракта и метка времени. Контракт считается подписанным, если другая сторона может предъявить обе половины, L_i и R_i, одной пары подписей.

(3) И Алиса, и Боб шифруют свои пары сообщений парами ключей DES, левое сообщение - левым ключом в паре, а правое - правым.

(4) Алиса и Боб посылают друг другу свои пачки из $2n$ шифрованных сообщений, поясняя, какие сообщения какими половинами каких пар являются.

(5) Алиса и Боб посылают друг другу все пары ключей, используя протокол рассеянной передачи для каждой пары. То есть, Алиса посылает Бобу независимо для каждой из n пар ключей либо ключ, использованный для шифрования левого сообщения, либо ключ, использованный для шифрования правого сообщения. Боб делает то же самое. Они могут посылать свои половинки по очереди, или сначала один может послать все 100, а потом другой - это не имеет значения. Теперь и у Алисы, и у Боба есть по одному ключу из каждой пары, но никто не знает, какие из половинок получил партнер.

(6) Алиса и Боб, используя полученные ключи, расшифровывают те половинки сообщений, которые они могут расшифровать. Они убеждаются, что расшифрованные сообщения правильны.

(7) Алиса и Боб посылают друг другу первые биты всех $2n$ ключей DES.

(8) Алиса и Боб повторяют этап (7) для вторых битов всех $2n$ ключей DES, затем третьих битов и так далее, пока все биты всех ключей DES не будут переданы.

(9) Алиса и Боб расшифровывают оставшиеся половинки сообщений, и контракт подписан.

(10) Алиса и Боб обмениваются закрытыми ключами, использованными для протокола рассеянной передачи на этапе (5), и каждый из них убеждается в отсутствии мошенничества.

Почему Алисе и Бобу нужно выполнить всю эту нудную последовательность действий? Предположим, что Алиса хочет смошенничать, и посмотрим, что получится. На этапах (4) и (5) Алиса могла бы разрушить протокол, послав Бобу ничего не значащие строки. Боб обнаружил бы это на этапе (6) при попытке расшифровать полученные половинки. Боб с полной безопасностью может остановиться до того, как Алиса сможет расшифровать любую из пар сообщений Боба.

Если бы Алиса была очень хитрой, она могла бы разрушить только половину протокола. Она могла бы послать одну половинку из каждой пары правильно, а вместо второй отправить бессмысленные строки. Вероятность Боба получить правильную половинку составит только 50 процентов, поэтому в половине случаев мошенничество Алисы удастся, но только для одной пары. Если бы использовались только две пары, этот способ мошенничества удастся в 25 процентах случаев. Вот почему n должно быть велико. Алисе необходимо точно угадать результат n протоколов рассеянной передачи, ее вероятность добиться этого составляет 1 шанс из 2^n. Если $n = 10$, у Алисы 1 шанс из 1024 обмануть Боба.

Алиса также может отправить Бобу случайные биты на этапе (8). Возможно, Боб не узнает, что она послала ему случайные биты, пока не получит весь ключ и не попытается расшифровать половинки сообщения. Но снова вероятность на стороне Боба. Он уже получил половину ключей, и Алиса не знает какую. Если n достаточно велико, Алиса наверняка пришлет ему бессмысленный бит для ключа, который у него уже есть, и он немедленно узнает, что она пытается его обмануть.

Возможно, Алиса будет выполнять этап (8) до тех пор, пока она не получит достаточно битов ключей для вскрытия грубым взломом, и затем прекратит передачу битов. Длина ключа DES - 56 битов. Если она получила 40 из 56 битов, ей останется перебрать 2^{16}, или 65536, ключей для дешифровки сообщения, а эта задача, определенно, по силам современным компьютерам. Но Боб получит ровно столько же битов ее ключей (или, в худшем случае, на один бит меньше), следовательно, он сможет сделать то же самое. У Алисы нет другого выбора, кроме как продолжать следовать протоколу.

Идея в том, что Алисе придется играть честно, потому что вероятность обмануть Боба слишком мала. В конце протокола у обеих сторон есть n подписанных пар сообщений, любое из которых достаточно для правильной подписи.

У Алисы есть только один способ смошенничать - она может послать Бобу одинаковые сообщения на этапе (5). Боб не сможет обнаружить этого до окончания протокола, но он сможет использовать стенограмму протокола, чтобы убедить судью в двуличности Алисы.

Протоколы этого типа имеют два слабых места [138]. Во первых, проблема возникает, если вычислительная мощь одной стороны гораздо больше, чем у другой. Если, например, Алиса может выполнить вскрытие грубым взломом быстрее Боба, то она рано прекратит передачу битов на этапе (8) и раскроет ключи Боба самостоятельно. Бобу, которому для таких же действий просто не хватит времени, не повезет.

Во вторых, проблема возникает, если одна из сторон прекращает протокол раньше времени. Если Алиса оборвет выполнение протокола, оба столкнутся с одинаковыми вычислительными проблемами, но у не хватит ресурсов завершить вычисления к нужному сроку. Проблема появляется, к примеру, если контракт определяет, что Алиса должна сделать что-то через неделю, а она прерывает протокол в тот момент, когда Бобу для вычисления ее подписи потребуется целый год расчетов. Реальная сложность при этом заключается в близкой дате предмета контракта, к которой процесс не будет завершен одной или обеими подписывающими сторонами.

Эти проблемы существуют также для протоколов разделов 5.8 и 5.9.

5.8 Электронная почта с подтверждением

Такой же протокол одновременной рассеянной передачи, использованный для подписания контракта, с небольшими изменениями используется для электронной почты с подтверждением [529]. Пусть Алиса хочет послать сообщение Бобу, но не хочет, чтобы он прочитал его, не расписавшись в получении. В реальной жизни это обеспечивается неприветливыми почтовыми служащими, но то же самое может быть сделано при помощи криптографии. Эту проблему первым рассмотрел Уитфилд Диффи в [490].

На первый взгляд, эту проблему мог бы решить протокол одновременного подписания контракта. Алиса просто копирует свое сообщение ключом DES. Ее половина протокола выглядит примерно так: "Это левая половина ключа DES: 32f5", а половина протокола Боба - так: "Это левая половина моей квитанции." Все остальное не меняется.

Чтобы понять, почему это не работает, вспомните, что протокол опирается на то, что рассеянная передача на этапе (5) предохраняет от мошенничества обе стороны. Оба партнера знают, что они послали другой стороне правильную половину, но никто не знает какую. Они не мошенничают на этапе (8), потому что вероятность выйти сухим из воды чрезвычайно мала. Если Алиса посылает Бобу не сообщение, а половину ключа DES, то Боб не может проверить правильность ключа DES на этапе (6). Алиса же может проверить правильность квитанции Боба, поэтому Бобу придется быть честным. Алиса легко может отправить Бобу неправильный ключ а когда он обнаружит это, его квитанция уже будет у Алисы. Вот невезуха, Боб.

Решение этой проблемы потребует некоторой коррекции протокола:

(1) Алиса шифрует свое сообщение случайным ключом DES и посылает его Бобу.

(2) Алиса создает n пар ключей DES. Первый ключ каждой пары генерируется случайным образом, а второй представляет собой XOR первого ключа и ключа шифрования сообщения.

(3) Алиса шифрует сообщение-заглушку каждым из своих $2n$ ключей.

(4) Алиса посылает Бобу всю пачку шифрованных сообщений, проверяя, что он знает, какие сообщения какими половинами каких пар являются.

(5) Боб создает n пар случайных ключей DES.

(6) Боб создает пару сообщений, образующих правильную квитанцию. Хорошим вариантами могут служить "Это левая половина моей квитанции" и "Это левая половина моей квитанции" с добавлением какой-нибудь строки случайных битов. Он создает n пар квитанций, нумеруя каждую. Как и в предыдущем протоколе квитанция считается правильной, если Алиса может предъявить обе половины квитанции (с одним и тем же номером) и все ее ключи шифрования.

(7) Боб шифрует каждую свою пару сообщений парами ключей DES, i-ую пару сообщений - i-ой парой ключей, левое сообщение - левым ключом в паре, а правое - правым. в паре.

(8) Боб посылает Алисе свою пачку шифрованных сообщений, проверяя, что она знает, какие сообщения какими половинами каких пар являются.

(9) Алиса и Боб посылают друг другу все пары ключей, используя протокол рассеянной передачи. То есть,

Алиса посылает Бобу независимо для каждой из n пар ключей либо ключ, использованный для шифрования левого сообщения, либо ключ, использованный для шифрования правого сообщения. Боб делает то же самое. Они могут посылать свои половинки по очереди, или сначала один может послать все n, а потом другой - это не имеет значения. Теперь и у Алисы, и у Боба есть по одному ключу из каждой пары, но никто не знает, какие из половинок получил партнер.

(10) Алиса и Боб расшифровывают те половинки сообщений, которые могут и убеждаются, что расшифрованные сообщения правильны.

(11) Алиса и Боб посылают друг другу первые биты всех $2n$ ключей DES. (Если они беспокоятся, что Ева сможет прочитать эти почтовые сообщения, то они должны шифровать свой обмен битами).

(12) Алиса и Боб повторяют этап (11) для вторых битов всех $2n$ ключей DES, затем третьих битов и так далее, пока все биты всех ключей DES не будут переданы.

(13) Алиса и Боб расшифровывают оставшиеся половинки сообщений. Алиса получает правильную квитанцию от Боба, а Боб может выполнить "исключающее или" для любой пары ключей и получить ключ, которым зашифровано оригинальное сообщение.

(14) Алиса и Боб обмениваются закрытыми ключами, использованными для протокола рассеянной передачи, и каждый из них убеждается в отсутствии мошенничества.

Этапы (5)-(8) для Боба и (9)-(12) для обеих сторон не меняются по сравнению с протоколом подписания контракта. Отличие - в сообщениях-заглушках Алисы. Они предоставляют Бобу возможность проверить правильность ее рассеянной передачи на этапе (10), что заставляет ее оставаться честной на этапах (11)-(13). И, как и для протокола одновременного подписания контракта, для выполнения протокола требуются обе половины одного из сообщений Алисы.

5.9 Одновременный обмен секретами

Алиса знает секрет A, а Боб - секрет B. Алиса собирается сообщить Бобу A, если он расскажет ей B. Боб хочет сообщить Алисе B, если она расскажет ему A. Следующий протокол, подслушанный на школьном дворе, работать не будет:

(1) Алиса: "Я скажу, если ты скажешь мне первым."

(2) Боб: "Я скажу, если ты скажешь мне первой."

(3) Алиса: "Нет, ты первый."

(4) Боб: "Ну, хорошо." Боб шепчет Алисе.

(5) Алиса: "Ха, а я тебе не скажу."

(6) Боб: "Это не честно."

Честным это может сделать криптография. Предыдущие два протокола являются реализациями более общего протокола, который и позволит Алисе и Бобу одновременно обменяться секретами [529]. Чтобы не повторять полностью весь протокол, я набросаю необходимые изменения протокола почты с подтверждением.

Алиса выполняет этапы (1)-(4), используя в качестве сообщения A. Боб выполняет эти же действия, используя в качестве своего сообщения B. Алиса и Боб выполняют рассеянную передачу на этапе (9), расшифровывают на этапе (10) те половинки, которые могут, и выполняют необходимые итерации на этапах (11) и (12). Чтобы защититься от Евы, они должны шифровать свои сообщения. Наконец, и Алиса, и Боб расшифровывают оставшиеся половины пар сообщения и выполняют XOR для любой пары ключей, чтобы получить ключи, которыми зашифрованы оригинальные сообщения.

Этот протокол позволяет Алисе и Бобу одновременно обмениваться секретами, но не гарантирует качества переданных секретов. Алиса может пообещать Бобу план лабиринта Минотавра и прислать ему схему Бостонского метро. Боб получит только тот секрет, который Алиса пришлет ему. Другие протоколы описаны в [1286, 195, 991, 1524, 705, 753, 259, 358, 415].

Глава 6
Эзотерические протоколы

6.1 Безопасные выборы

Компьютерное голосование никогда не будет использовано для обычных выборов, пока не появится прот о-кол, который одновременно предохраняет от мошенничества и защищает тайну личности. Идеальный протокол должен обладать, по меньшей мере, следующими шестью свойствами :

1. Голосовать могут только те, кто имеет на это право .
2. Каждый может голосовать не более одного раза .
3. Никто не может узнать, за кого проголосовал конкретный избиратель .
4. Никто не может проголосовать вместо другого. (Это оказывается самым тяжелым требованием .)
5. Никто не может тайно изменить чей-то голос .
6. Каждый голосующий может проверить, что его голос учитывался при подведении итогов голосования .

Кроме того, для некоторых схем голосования может понадобиться следующее требование :

7. Каждый знает, кто голосовал, а кто нет.

Прежде чем описывать сложные протоколы, имеющие приведенные характеристики, давайте взглянем на рад протоколов попроще.

Упрощенный протокол голосования №1

(1) Каждый голосующий шифрует свой бюллетень открытым ключом Центральной избирательной комиссии (ЦИК).

(2) Каждый голосующий посылает свой бюллетень в ЦИК .

(3) ЦИК расшифровывает бюллетени, подводит итоги и опубликовывает результаты голосования .

Этот протокол просто кишит проблемами. ЦИК не может узнать, откуда получены бюллетени, и даже, пр и-надлежат ли присланные бюллетени правомочным избирателям . У нее нет ни малейшего представления о том, не голосовали ли правомочные избиратели больше одного раза . Положительной стороной является невозмо ж-ность изменить бюллетень другого человека , но никто и не будет пытаться это сделать, потому что гораздо г о-лосовать повторно, добиваясь нужных результатов выборов .

Упрощенный протокол голосования №2

(1) Каждый голосующий подписывает свой бюллетень своим закрытым ключом .

(2) Каждый голосующий шифрует свой бюллетень открытым ключом ЦИК .

(3) Каждый голосующий посылает свой бюллетень в ЦИК .

(4) ЦИК расшифровывает бюллетени, проверяет подписи, подводит итоги и опубликовывает результаты г о-лосования.

Этот протокол обладает свойствами 1 и 2 : Только правомочные избиратели могут голосовать, и никто не может голосовать более одного раза - ЦИК может записывать бюллетени, полученные на этапе (3). Каждый бюллетень подписан закрытым ключом голосующего, поэтому ЦИК знает, кто голосовал, а кто нет, и, как гол о-совал каждый избиратель. Если получен бюллетень, который не подписан правомочным пользователем, или бюллетень, подписанный избирателем, который уже проголосовал , то такой бюллетень игнорируется комиссией. Кроме того, из-за цифровой подписи никто не может изменить бюллетень другого избирателя, даже если сумеет перехватить его на этапе (2).

Проблема этого протокола в том, что подпись добавляется к бюллетеню, ЦИК знает, кто за кого голосовал . Шифрование бюллетеней открытым ключом ЦИК мешает посторонним злоупотреблять протоколом и узнавать, кто за кого голосовал, но вам придется полностью доверять ЦИК Это как будто в кабинке для голосования вам через плечо заглядывает электронный судья .

Два следующих примера показывают, как трудно обеспечить хотя бы первые три требования к протоколу безопасного голосования.

Голосование со слепыми подписями

Нам нужно как-то отделить бюллетень от голосующего, сохранив процедуру идентификации личности. Именно это можно сделать с помощью протокола слепой подписи.

(1) Каждый избиратель создает 10 наборов сообщений, каждый набор содержит правильный бюллетень для каждого возможного результата (например, если бюллетенем является один из ответов "да"-"нет", то каждый набор состоит из двух бюллетеней, одного для "да", а другого для "нет"). Каждое сообщение содержит также случайным образом созданный идентификационный номер, достаточно большой, чтобы избежать путаницы с другими избирателями.

(2) Каждый избиратель лично маскирует все сообщения (см. раздел 5.3) и посылает их в ЦИК вместе с маскирующим множителями.

(3) ЦИК по своей базе данных проверяет, что пользователь не присылал раньше для подписания свои замаскированные бюллетени. ЦИК открывает 9 из наборов, проверяя, что они правильно сформированы. Затем она индивидуально подписывает каждое сообщение набора и посылает их обратно избирателю, сохраняя имя избирателя в своей базе данных.

(4) Избиратель снимает маскировку с сообщений и получает набор бюллетеней, подписанных ЦИК. (Эти бюллетени подписаны, но не зашифрованы, поэтому избиратель легко увидит, какой из бюллетеней - "да", а какой - "нет".)

(5) Каждый избиратель выбирает один из бюллетеней (о, демократия!) и шифрует его открытым ключом ЦИК.

(6) Избиратель отправляет свой бюллетень.

(7) ЦИК расшифровывает бюллетени, проверяет подписи, проверяет по базе данных уникальность идентификационного номера, сохраняет последовательный номер и подводит итоги. Она опубликовывает результаты голосования вместе с каждым последовательным номером и соответствующим бюллетенем.

Мэллори, избиратель-жулик, не может обмануть эту систему. Протокол слепой подписи обеспечивает единственность его бюллетени. Если он попытается отправить тот же бюллетень дважды, ЦИК обнаружит дублирование последовательных номеров на этапе (7) и не будет учитывать второй бюллетень. Если он попытается получить несколько бюллетеней на этапе (2), ЦИК обнаружит это на этапе (3). Мэллори не может создать свои собственные бюллетени, потому что он не знает закрытого ключа комиссии. По той же причине он не может перехватить и изменить чужие бюллетени.

Протокол "разрезать и выбрать" на этапе (3) должен обеспечить уникальность бюллетеней. Без этого этапа Мэллори мог бы создать точно такой же, за исключением идентификационного номера, набор бюллетеней и заверить их все в ЦИК.

Мошенническая ЦИК не сможет узнать, как голосовал конкретный избиратель. Так как протокол слепой подписи маскирует последовательные номера бюллетеней до момента подведения итогов, ЦИК не сможет установить связь между подписанным ею замаскированным бюллетенем и подытоживаемым бюллетенем. Опубликование перечня последовательных номеров и связанных с ними бюллетеней позволяет пользователям убедиться, что их бюллетени были правильно учтены.

Но проблемы все еще остаются. Если этап (6) не анонимен, и ЦИК может записать, кто какой бюллетень прислал, то она сможет узнать, кто за кого голосовал. Однако, это невозможно, если комиссия получает бюллетени в запечатанной урне для голосования и считает их позже. Хотя ЦИК и не сможет установить связь между избирателями и их бюллетенями, она сможет создать большое количество подписанных и правильных бюллетеней и смошенничать, прислав их сама себе. И если Алиса обнаружит, что ЦИК подменила ее бюллетень, она не сможет доказать этого. Аналогичный протокол, пытающийся устранить эти проблемы, описан в [1195, 1370].

Голосование с двумя Центральными комиссиями

Одним из решений является разделить ЦИК пополам. Ни у одной из них не будет достаточно власти, чтобы смошенничать по своему усмотрению.

В следующем протоколе используется Центральное управление регистрации (ЦУР), занимающееся проверкой пользователей, и отдельная ЦИК для подсчета бюллетеней [1373].

(1) Каждый избиратель отправляет письмо в ЦУР, запрашивая регистрационный номер.

(2) ЦУР возвращает избирателю случайный регистрационный номер. ЦУР ведет список регистрационных номеров. Кроме того, ЦУР хранит список получателей регистрационных номеров на случай, если кто-то попытается проголосовать дважды.

(3) ЦУР отправляет список регистрационных номеров в ЦИК.

(4) Каждый избиратель выбирает случайный идентификационный номер. Он создает сообщение с этим номером, регистрационным номером, полученным в ЦУР, и своим бюллетенем. Он посылает это сообщение в ЦИК.

(5) ЦИК проверяет регистрационные номера по списку, полученному от ЦУР на этапе (3). Если регистрационный номер есть в списке, ЦИК вычеркивает его (чтобы избежать повторного голосования). ЦИК добавляет идентификационный номер к списку тех, кто проголосовал за определенного кандидата, и прибавляет единичку к соответствующему итоговому числу.

(6) После того, как все бюллетени будут получены, ЦИК публикует результаты вместе со списками, содержащими идентификационные номера и соответствующие бюллетени.

Как и в предыдущем протоколе каждый избиратель может увидеть список идентификационных номеров и найти в нем свой собственный. Так он может убедиться, что его бюллетень учтен. Конечно, все сообщения, которыми обмениваются участники протокола должны быть зашифрованы и подписаны, чтобы помешать кому-нибудь выдать себя за другого или перехватить сообщения.

ЦИК не может изменить бюллетени, потому что каждый избиратель будет искать свой регистрационный номер. Если избиратель не находит свой регистрационный номер или находит его в итоговом списке с другим результатом голосования, он немедленно узнает, что произошел обман. ЦИК не может добавить бюллетень в урну, которая находится под наблюдением ЦУР. ЦУР знает, сколько избирателей зарегистрировалось, их регистрационные номера и обнаружит любые изменения.

Мэллори, не обладающий избирательными правами, может попытаться смошенничать, угадав правильный регистрационный номер. Угроза этого может быть минимизирована, если множество возможных регистрационных номеров намного больше, чем множество реальных регистрационных номеров: 100-битовое число для миллиона избирателей. Конечно же, регистрационные номера должны генерироваться случайным образом.

Несмотря на это, ЦУР должна быть заслуживающим доверия органом власти - ведь она может зарегистрировать неправомочных избирателей. Она также может зарегистрировать правомочных избирателей несколько раз. Этот риск может быть сведен к минимуму, если ЦУР опубликует список зарегистрировавшихся избирателей (но без их регистрационных номеров). Если число избирателей в этом списке меньше, чем число подсчитанных бюллетеней, то что-то не так. Однако, если зарегистрировалось больше избирателей, чем было прислано бюллетеней, то это, возможно, означает, что ряд зарегистрировавшихся избирателей не проголосовал. Многие, зарегистрировавшись, не утруждаются бросить в урну свой бюллетень.

Этот протокол беззащитен перед сговором ЦИК и ЦУР. Если они действуют вместе, они могут объединить свои базы данных и узнать, кто за кого голосует.

Голосование с одной Центральной комиссией

Чтобы избежать опасности сговора между ЦУР и ЦИК можно использовать более сложный протокол [1373]. Этот протокол идентичен предыдущему с двумя изменениями:

— ЦУР и ЦИК являются единой организацией, и

— для анонимного распределения регистрационных номеров на этапе (2) используется ANDOS (см. раздел 4.13).

Так как протокол анонимного распределения ключей не позволяет ЦИК узнать, у какого избирателя какой регистрационный номер, У ЦИК нет способа связать регистрационные номера и полученные бюллетени. Но ЦИК должна быть надежным органом, чтобы не выдавать регистрационных номеров неправомочным избирателям. Эту проблему также можно решить с помощью слепых подписей.

Улучшенное голосование с одной Центральной комиссией

В этом протоколе также используется ANDOS [1175]. Он удовлетворяет всем шести требованиям хорошего протокола голосования. Он не удовлетворяет седьмому требованию, но обладает двумя свойствами, дополняющими перечисленные в начале раздела шесть свойств:

7. Избиратель может изменить свое мнение (т.е., аннулировать свой бюллетень и проголосовать заново) в течение заданного периода времени.

8. Если избиратель обнаруживает, что его бюллетень посчитан неправильно, он может установить и и справить проблему, не рискуя безопасностью своего бюллетеня.

Вот этот протокол:

(1) ЦИК публикует список всех правомочных избирателей.

(2) В течение определенного срока каждый избиратель сообщает в ЦИК, собирается ли он голосовать.

(3) ЦИК публикует список избирателей, участвующих в выборах .

(4) Каждый избиратель получает идентификационный номер , *I*, с помощью протокола ANDOS.

(5) Каждый избиратель генерирует пару открытый ключ/закрытый ключ : *k*, *d*. If Если *v* - это бюллетень, то избиратель создает и посылает в ЦИК следующее сообщение :

$I, E_k(I, v)$

Это сообщение должно быть послано анонимно .

(6) ЦИК подтверждает получение бюллетеня, публикуя :

$E_k(I, v)$

(7) Каждый избиратель посылает ЦИК :

I, d

(8) ЦИК расшифровывает бюллетени. В конце выборов она публикует их результаты и, для каждого варианта выбора, список соответствующий значений $E_k(I, v)$.

(9) Если избиратель обнаруживает, что его бюллетень подсчитан неправильно, он протестует, посылая ЦИК :

$I, E_k(I, v), d$

(10) Если избиратель хочет изменить свой бюллетень с *v* на *v′*, он посылает ЦИК:

$I, E_k(I, v'), d$

Другой протокол голосования использует вместо ANDOS слепые подписи, но по сути мало чем отличается [585]. Этапы (1) - (3) являются предварительными. Их цель состоит в том, чтобы узнать и опубликовать всех действительных избирателей. Хотя некоторые из них, вероятно, не примут участи в голосовании, это уменьшает возможность ЦИК добавить поддельные бюллетени .

На этапе (4) два избирателя могут получить один и тот же идентификационный номер . Эта возможность может быть минимизирована, если число возможных идентификационных номеров будет гораздо больше, чем число реальных избирателей. Если два избирателя присылают бюллетени с одинаковым идентификатором, ЦИК генерирует новый идентификационный номер, *I′*, выбирает одного из избирателей и публикует :

$I', E_k(I, v)$

Владелец этого бюллетеня узнает о произошедшей путанице и посылает свой бюллетень снова, повторяя этап (5) с новым идентификационным номером .

Этап (6) дает каждому избирателю возможность проверить, что ЦИК правильно получила его бюллетень . Если его бюллетень неправильно подсчитан, он может доказать это на этапе (9). Предполагая, что бюллетень избирателя на этапе (6) правилен, сообщение, которое он посылает на этапе (9) доказывает, что его бюллетень был неправильно подсчитан.

Одной из проблем этого протокола является то, что жульническая ЦИК сможет воспользоваться людей, к оторые сообщили о намерении голосовать на этапе (2), но не голосовали в действительности . Другой проблемой является сложность протокола ANDOS. Авторы рекомендуют разбивать избирателей на меньшие группы, н апример избирательные округа .

Еще одной, более серьезной проблемой является то, что ЦИК может не подсчитать какой-нибудь бюллетень . Эта проблема неразрешима: Алиса утверждает, что ЦИК намеренно пренебрег ее бюллетенем, а ЦИК утве рждает, что Алиса никогда не голосовала .

Голосование без Центральной избирательной комиссии

В следующем протоколе ЦИК не используется, избиратели следят друг за другом . Этот протокол, созданный Майклом Мерриттом [452, 1076, 453], настолько громоздок, что возможность его реализации больше чем для пяти человек сомнительна, но все же познакомиться с ним будет полезно .

Алиса, Боб, Кэрол и Дэйв голосуют (да/нет или 0/1) по какому-то вопросу . Пусть у каждого избирателя есть открытый и закрытый ключи. Пусть также все открытые ключи известны всем .

(1) Каждый голосующий решает, как голосовать, и делает следующее:

(a) Добавляет случайную строку к своему бюллетеню.

(b) Шифрует результат этапа (a) открытым ключом Дэйва.

(c) Шифрует результат этапа (b) открытым ключом Кэрол.

(d) Шифрует результат этапа (c) открытым ключом Боба.

(e) Шифрует результат этапа (d) открытым ключом Алисы.

(f) Добавляет новую случайную строку к результату этапа (e) и шифрует получившееся открытым ключом Дэйва. Он записывает значение случайной строки.

(g) Добавляет новую случайную строку к результату этапа (f) и шифрует получившееся открытым ключом Кэрол. Он записывает значение случайной строки.

(h) Добавляет новую случайную строку к результату этапа (g) и шифрует получившееся открытым ключом Боба. Он записывает значение случайной строки.

(i) Добавляет новую случайную строку к результату этапа (g) и шифрует получившееся открытым ключом Алисы. Он записывает значение случайной строки.

Если E - это функция шифрования, R_i - случайная строка, а V - бюллетень , то его сообщение будет выглядеть следующим образом:

$E_A(R_5,E_B(R_4,E_C(R_3,E_D(R_2,E_A(E_B(E_C(E_D(V,R_1)))))))$

Каждый голосующий сохраняет промежуточные результаты на каждом этапе вычислений. Эти результаты будут использоваться на дальнейших этапах протокола для подтверждения, что бюллетень данного избирателя будет учтен.

(2) Каждый голосующий отправляет сообщение Алисе.

(3) Алиса расшифровывает все бюллетени, используя свой закрытый ключ, и удаляет все случайные строки на этом уровне.

(4) Алиса перетасовывает все бюллетени и посылает результат Бобу.

Теперь каждый бюллетень будет выглядеть следующим образом:

$E_B(R_4,E_C(R_3,E_D(R_2,E_A(E_B(E_C(E_D(V,R_1)))))))$

(5) Боб расшифровывает все бюллетени, используя свой закрытый ключ, проверяет, есть ли его бюллетень среди присланных бюллетеней, удаляет все случайные строки на этом уровне, тасует бюллетени и посылает результат Кэрол.

Теперь каждый бюллетень будет выглядеть следующим образом:

$E_C(R_3,E_D(R_2,E_A(E_B(E_C(E_D(V,R_1))))))$

(6) Кэрол расшифровывает все бюллетени, используя свой закрытый ключ, проверяет, есть ли ее бюллетень среди присланных бюллетеней, удаляет все случайные строки на этом уровне, тасует бюллетени и посылает результат Дэйву.

Теперь каждый бюллетень будет выглядеть следующим образом:

$E_D(R_2,E_A(E_B(E_C(E_D(V,R_1)))))$

(7) Дэйв расшифровывает все бюллетени, используя свой закрытый ключ, проверяет, есть ли его бюллетень среди присланных бюллетеней, удаляет все случайные строки на этом уровне, тасует бюллетени и посылает результат Алисе.

Теперь каждый бюллетень будет выглядеть следующим образом:

$E_A(E_B(E_C(E_D(V,R_1))))$

(8) Алиса расшифровывает все бюллетени, используя свой закрытый ключ, проверяет, есть ли ее бюллетень среди присланных бюллетеней, подписывает все бюллетени и посылает результат Бобу, Кэрол и Дэйву.

Теперь каждый бюллетень будет выглядеть следующим образом:

$S_A\ (E_B(E_C(E_D(V,R_1))))$

(9) Боб проверяет и удаляет подписи Алисы. Он расшифровывает все бюллетени, используя свой закрытый ключ, проверяет, есть ли его бюллетень среди присланных бюллетеней, подписывает все бюллетени и посылает результат Алисе, Кэрол и Дэйву.

Теперь каждый бюллетень будет выглядеть следующим образом:

$S_B(E_C(E_D(V,R_1)))$

(10) Кэрол проверяет и удаляет подписи Боба. Она расшифровывает все бюллетени, используя свой закрытый

ключ, проверяет, есть ли ее бюллетень среди присланных бюллетеней, подписывает все бюллетени и посылает результат Алисе, Бобу и Дэйву.

Теперь каждый бюллетень будет выглядеть следующим образом:

$S_C(E_D(V,R_1))$

(11) Дэйв проверяет и удаляет подписи Кэрол. Он расшифровывает все бюллетени, используя свой закрытый ключ, проверяет, есть ли его бюллетень среди присланных бюллетеней, подписывает все бюллетени и посылает результат Алисе, Бобу и Кэрол.

Теперь каждый бюллетень будет выглядеть следующим образом:

$S_D(V,R_1)$

(12) Все проверяют и удаляют подпись Дэйва. Они убеждаются, что их бюллетени находятся среди полученных (находя свою случайную строку).

(13) Все удаляют случайные строки из каждого бюллетеня и суммирует бюллетени.

Этот протокол не только работает, он сам является своим арбитром. Алиса, Боб, Кэрол и Дэйв немедленно узнают, если кто-нибудь из них попытается мошенничать. Не нужно никаких ЦИК и ЦУР. Чтобы увидеть, как это работает, попытаемся смошенничать.

Если кто-нибудь пытается добавить бюллетень, Алиса обнаружит эту попытку на этапе (3), когда она получит бюллетеней больше чем количество людей, участвующих в голосовании. Если Алиса попытается добавить бюллетень, Боб обнаружит это на этапе (4).

Более ловкой является подмена одного бюллетеня другим. Так как бюллетени шифруются различными открытыми ключами, каждый может создать столько правильных бюллетеней, сколько нужно. Протокол дешифрирования состоит из двух частей: первая включает этапы (3)-(7), а вторая - этапы (8)-(11). Подмена голоса на различных этапах обнаруживается по разному.

Если кто-нибудь заменит один бюллетень другим во второй части, его действия будут обнаружены неменленно. На каждом этапе бюллетени подписываются и посылаются всем избирателям. Если один избиратель (или несколько) обнаруживает, что его бюллетеня больше нет среди набора бюллетеней, он немедленно прекращает выполнение протокола. Так как бюллетени подписываются на каждом этапе, и так как каждый может вернуться во второй части протокола на несколько шагов назад, то обнаружить мошенника, подменившего бюллетени, легко.

Замена одного бюллетеня другим в первой части протокола более тонка. Алиса не может сделать замену на этапе (3), потому что Боб, Кэрол и Дэйв обнаружат это на этапах (5), (6) или (7). Боб может попробовать на этапе (5). Если он заменит бюллетени Кэрол и Дэйва (помните, он не знает, какой бюллетень чей), Кэрол или Дэйв заметят это на этапах (6) или (7). Они не будут знать, кто подменил их бюллетени (хотя это должен быть кто-то, уже обработавший бюллетени), но они будут знать, что их голоса подменены. Если Бобу повезло, и ему удалось подменить бюллетень Алисы, она не заметит этого до второй части протокола. Тогда она обнаружит исчезновение своего голоса на этапе (8), но не сможет узнать, кто подменил бюллетень. В первой части бюллетени перетасовываются на каждом этапе и не подписываются, поэтому никто не сможет отработать протокол обратно и определить, кто подменил бюллетени.

Другой формой мошенничества является попытка узнать, кто за кого проголосовал. Из-за перетасовки бюллетеней в первой части никто не сможет отработать протокол обратно и связать бюллетени и голосующих. Удаление случайных строк в первой части также является решающим для сохранения анонимности. Если строки не удаляются, перемешивание голосов может быть инвертировано при помощи повторного шифрования получаемых голосов открытым ключом того, кто их тасовал. Когда протокол остановится, конфиденциальность бюллетеней сохранится.

Более того, из-за начальной случайной строки, R_1, даже одинаковые бюллетени шифруются по разному на каждом этапе протокола. Никто не может узнать значение бюллетеня до этапа (11).

Каковы проблемы этого протокола? Во первых, для выполнения протокола нужны грандиозные вычисления. В приведенном примере в голосовании принимают участие только четверо, но и он *уже* сложен. Такой протокол не сможет работать при реальных выборах с десятками тысяч голосующих. Во вторых, Дэйв узнает результаты выборов раньше остальных. Хотя он и не может повлиять на результат, он получает определенное преимущество. С другой стороны такое также возможно и при централизованной схеме голосования.

Третья проблема заключается в том, что Алиса может скопировать бюллетень другого участника, даже не зная его содержания заранее. Чтобы понять, почему это может стать проблемой, рассмотрим выборы для трех голосующих - Алисы, Боба и Евы. Еве не важны результаты выборов, но она хочет знать, как голосовала Алиса. Поэтому она копирует бюллетень Алисы, и результат выборов будет соответствовать бюллетеню Алисы.

Другие схемы голосования

Было предложено много сложных безопасных протоколов выборов. Их можно разделить на два типа. Существуют протоколы с перемешиванием, как "Голосование без Центральной избирательной комиссии", в которых все бюллетени перемешиваются, чтобы никто не мог связать бюллетень и избирателя.

Также существуют протоколы с разделением, в которых личные бюллетени делятся между различными счетными комиссиями так, что ни одна из них не сможет обмануть избирателей [360, 359, 118, 115]. Эти протоколы Эти протоколы защищают анонимность избирателей только, если различные "части" правительства (или кто бы не проводил голосование) не сговариваются против избирателя. (Идея разбить центральный орган на несколько частей, которые пользуются доверием, только когда они действуют параллельно, пришла из [316].)

Один из протоколов с разделением предложен в [1371]. Основная идея состоит в том, что каждый избиратель делит свой бюллетень на несколько частей. Например, если бы бюллетень содержал "да" или "нет", 1 обозначала бы "да", а 0 - "нет", избиратель мог бы создать несколько чисел, которые в сумме давали бы 0 или 1. Эти доли посылаются счетным комиссиям, каждой по одной, и также шифруются и сохраняются. Каждый центр суммирует полученные доли (существуют протоколы, обеспечивающие правильность итога), и окончательный итог является суммой всех промежуточных итогов. Существуют также протоколы, гарантирующие, что доли каждого избирателя будут сложены для получения 0 или 1.

Другой протокол, предложенный Дэвидом Чаумом [322], позволяет проследить избирателя, который пытается мошенничать. Однако, выборы придется проводить повторно, исключив мешающего пользователя. Этот подход не применим на практике для выборов с большим числом избирателей.

Еще один, более сложный протокол, решающий некоторые из этих проблем можно найти в [770, 771]. Существует даже протокол, использующий шифры со многими ключами [219]. Другой протокол, который, как утверждается, подходит для крупномасштабных выборов, приведен в [585]. А [347] позволяет избирателям не голосовать.

Протоколы голосования работают, они даже облегчают продажу и покупку голосов. Когда покупатель может быть уверен, что продавец проголосует, как обещал, стимул купить голоса становится еще сильнее. Ряд протоколов были спроектированы **без подтверждения**, не позволяя избирателю доказать кому-либо еще, что он проголосовал определенным образом [117, 1170, 1372].

6.2 Безопасные вычисления с несколькими участниками

Безопасные вычисления с несколькими участниками представляют собой протокол, с помощью которого группа людей может определенным образом вычислить функцию многих переменных. Каждый в группе обеспечивает одну или несколько переменных. Результат вычислений становится известным каждому в группе, но никому не известны значения, предоставленные другими членами группы, если это не является очевидным из результата вычислений. Ниже приведено несколько примеров:

Протокол №l

Как может группа людей вычислить свою среднюю зарплату без того, чтобы зарплата одного стала известна другому?

(1) Алиса добавляет секретное случайное число к сумме своей зарплаты, шифрует результат открытым ключом Боба и посылает его Бобу.

(2) Боб расшифровывает результат своим закрытым ключом. Он добавляет сумму своей зарплаты к полученному от Алисы значению, шифрует результат открытым ключом Кэрол и посылает его Кэрол.

(3) Кэрол расшифровывает результат своим закрытым ключом. Она добавляет сумму своей зарплаты к полученному от Боба значению, шифрует результат открытым ключом Дэйва и посылает его Дэйву.

(4) Дэйв расшифровывает результат своим закрытым ключом. Он добавляет сумму своей зарплаты к полученному от Кэрол значению, шифрует результат открытым ключом Алисы и посылает его Алисе.

(5) Алиса расшифровывает результат своим закрытым ключом. Она вычитает случайное число, прибавленное на этапе (1), получая сумму всех зарплат.

(6) Алиса делит результат на число людей (в данном случае на четыре) и объявляет результат.

Этот протокол подразумевает, что каждый участник честен - они хотя и могут любопытствовать, но следуют протоколу. Если любой из участников солжет о своей зарплате, средняя зарплата будет рассчитана неправильно. Более серьезная проблема состоит в том, что Алиса может искажать итоговый результат. Она может вычесть на этапе (5) любое число, которое ее устраивает, и никто об этом не узнает. Помешать Алисе сделать это можно, потребовав от нее вручить ее случайное число с помощью одной из схем вручения бита из раздела 4.9, но когда

она откроет свое случайное число в конце протокола Боб сможет узнать ее зарплату.

Протокол №2

Алиса и Боб вместе в ресторане и спорят о том, кто старше. Никто, однако, не хочет сообщить другому свой возраст. Каждый из них мог бы прошептать свой возраст на ушко доверенной нейтральной стороне (например, официанту), кто мог бы сравнить числа в уме и объявить результат и Алисе, и Бобу.

У приведенного протокола есть две проблемы. Во первых, вычислительные способности среднего официант не позволяю ему обработать ситуации более сложный чем определение большего из двух чисел. И во вторых, если бы Алиса и Боб действительно заботились о сохранении своей информации в тайне, им пришлось бы ут опить официанта в ванне с минеральной водой, чтобы он ничего не разболтал буфетчику.

Криптография с открытыми ключами предлагает существенно менее жесткое решение. Существует прот окол, в соответствии с которым Алиса, зная значение a, и Боб, зная b, могут совместно определить верно ли, что $a<b$, так, чтобы Алиса не получила информации о b, а Боб - об a. Кроме того, и Алиса, и Боб убеждены в пр оверке правильности вычислений. Так как используемый криптографический алгоритм является существенной частью протокола, подробности можно найти в ра зделе 23.14.

Конечно, этот протокол не защитит от активных мошенников. Ничто не сможет помешать Алисе (или Бобу, какая разница) солгать о своем возрасте. Если бы Боб был компьютерной программой, которая слепо следовала бы протоколу, Алиса могла бы узнать его возраст (является ли возрастом компьютерной программы отрезок времени с момента ее написания или с момента ее запуска?), повторно выполняя протокол. Алиса могла бы ук азать, что ее возраст - 60 лет. Узнав, что она старше, она могла бы выполнить протокол снова, указав, что ее во зраст - 30 лет. Узнав, что Боб старше, она могла бы снова выполнить протокол, указав, что ее возраст - 45 лет, и так далее, пока Алиса не узнает возраст Боба с любой нужной ей степенью точности.

При условии, что участники не обманывают специально, этот протокол легко расширить для нескольких участников. Любое количество людей может определить порядок их возрастов с помощью последовательных честных применений прот окола, и никакой участник не сможет узнать возраст другого.

Протокол №3

Алиса нравится забавляться с плюшевыми медведями. В эротических фантазиях Боба важное место зан имают мраморные столы. Оба весьма стесняются своих привычек, но с удовольствием нашли бы кого-нибудь, кто разделил бы с ними их... гм... образ жи зни.

В Службе безопасных вычислений с несколькими участниками мы спроектировали протокол для подобных людей. Мы занумеровали впечатляющий список их пристрастий от "африканских муравьедов" до "яблочных пирогов". Разделенные модемной линией связи, Алиса и Боб могут участвовать в безопасном протоколе с н есколькими участниками. Они вместе могут определить, есть ли у них общие привычки. Если есть, они могли бы устремиться к обоюдному счастью. Если нет, то они могли бы безопасно расстаться, сохраняя уверенность, что их привычки остались в тайне. Никто, даже Служба безопасных вычислений с несколькими участниками, ник огда не узнает об их пристрастиях.

Вот как это работает:

(1) С помощью однонаправленной функции Алиса хэширует свою привычку как семизначную строку.

(2) Используя эту семизначную строку как телефонный номер, Алиса звонит по этому номеру и оставляет с ообщение Бобу. Если никто не отвечает, или номер не обслуживается. Алиса применяет однонаправленную функцию к телефонному номеру до тех пор, пока не найдется кто-нибудь, кто подхватит протокол.

(3) Алиса сообщает Бобу, сколько раз ей пришлось применять однонаправленную функцию к своей привы чке.

(4) Боб хэширует свою привычку столько же раз. Он также использует семизначную строку как телефонный номер и спрашивает человека на другом конце провода, нет ли для него сообщений.

Обратите внимание, что у Боба есть возможность вскрытия с использованием выбранного открытого текста. Он может хэшировать распространенные привычки и позвонить по получившемуся телефону, разыскивая соо бщения для него. Это протокол реально работает только такое вскрытие непрактично из-за достаточного числа возможных открытых текстов сообщений.

Также существует математический протокол, похожий на Протокол № 2. Алиса знает a, Боб знает b, и они вместе пытаются определить, верно ли, что $a = b$, причем так, чтобы Боб ничего не узнал об a, а Алиса - о b. Подробности можно найти в разделе 23.14.

Протокол №4

Вот другая проблема для безопасных вычислений со многими участниками [1373]: совет семи регулярно

встречается, чтобы тайно проголосовать по некоторым вопросам. (Все в порядке, они управляют миром - не говорите никому, что я вам проговорился.) Все члены совета могут голосовать "да" или "нет". Кроме того, две стороны обладают "супер-бюллетенями": 5-да и 5-нет. Они не обязаны использовать эти "супер-бюллетени" и, если хотят, могут воспользоваться обычными бюллетенями. Если никто не использует "супер-бюллетени", то вопрос решается простым большинством голосов. В случае применения одного или двух эквивалентных "супер-бюллетеней" все обычные голоса игнорируются. В случае двух противоречащих вопрос решается большинством обычных голосов. Нам нужен протокол, который надежно осуществляет такую форму голосования.

Следующие два примера иллюстрируют процесс голосования. Пусть участвуют пять обычных избирателей, от N_1 до N_5, и два суперизбирателя: S_1 и S_2. Вот голосование по вопросу №1:

S_1	S_2	N_1	N_2	N_3	N_4	N_5
Супер-да	нет	нет	нет	нет	да	да

В этом примере действует только один "супер-бюллетень" S_1, и результат голосования - "да". А вот голосование по вопросу №2:

S_1	S_2	N_1	N_2	N_3	N_4	N_5
Супер-да	Супер-нет	нет	нет	нет	да	да

В этом примере два "супер-бюллетеня" нейтрализуют друг друга, и вопрос решается большинством обычных "нет".

Если не требуется скрыть информацию о том, обычный или супербюллетель был решающим, то это простое применение безопасного протокола голосования. Сокрытие этой информации потребует более сложного безопасного протокола вычислений с несколькими участниками.

Этот вид голосования может произойти в реальной жизни. Это может быть часть организационной структуры корпорации, где некоторые люди обладают большей властью чем другие, или часть процедуры ООН, где одни государства имеют большее значение, чем другие.

Безусловные безопасные протоколы с несколькими участниками

Это только частный случай общей теоремы: любая функция с *n* входами может быть вычислена *n* игроками способом, который позволит всем узнать значение функции, но любое количество игроков, меньшее, чем *n/2*, не сможет получить никакой дополнительной информации, не следующей из их собственных входов и результата вычислений. Подробности можно найти в [136, 334, 1288, 621].

Безопасная оценка схемы

Вход Алисы - *a*, а Боба - *b*. Они вместе хотят вычислить некоторую функцию *f(a,b)* так, чтобы Алиса не смогла ничего узнать о входе Боба, а Боб - о входе Алисы. Главная проблема безопасных вычислений с несколькими участниками также называется **безопасной оценкой схемы**. Алиса и Боб могут создать произвольную логическую схему. Эта схема получает на вход значения Алисы и Боба и выдает результат. Безопасная оценка схемы является протоколом, который реализует следующие три требования:

1. Алиса может ввести свое значение так, что Боб не сможет его узнать.
2. Боб может ввести свое значение так, что Алиса не сможет его узнать.
3. И Алиса, и Боб могут вычислить результат, причем обе стороны будут убеждены в том, что результат правилен и не подтасован ни одной стороной.

Детали протокола безопасной оценки схемы можно найти в [831].

6.3 Анонимная широковещательная передача сообщений

Вам не удастся пообедать с компанией криптографов и не оказаться среди ожесточенной перепалки. В [321] Дэвид Чаум вводит Проблему обедающих криптографов:

> Три криптографа сидят за обедом в своем любимом трехзвездочном ресторане. Их официант сообщает им, что метрдотель принял необходимые меры, чтобы счет можно было бы оплатить анонимно. За обед мог бы заплатить один из криптографов или NSA. Три криптографа очень уважают право каждого из них заплатить анонимно, но им хотелось бы знать, заплатит ли NSA.

Как криптографам Алисе, Бобу и Кэрол узнать, не заплатил ли за обед кто-нибудь из них, и в то же время не

нарушить анонимность плательщика? Чаум решает проблему:

> Каждый криптограф бросает несмещенную монету, прикрывшись своим меню, между ним и криптографом справа от н е-го так, что только они двое могут видеть результат. Затем каждый криптограф громко объявляет, упаали ли две монеты - одна его и одна его левого соседа - на одну или на различные стороны. Если плательщик - один из криптографов, то его утвержд ение противоположно тому, что он видит. Нечетное число заявленных различий за столом указывает, что обед оплачивает криптограф; четное число различий - что NSA (при условии, что обед может быть оплачен только один раз). Однако, если обед оплачивает криптограф, никто из двух других не узнает из сделанных заявлений, кто же конкретно опл атил обед.

Чтобы увидеть, как это работает, вообразите, что Алиса пытается понять, кто из двух других криптографов заплатил за обед (при условии, что не она и не NSA). Если она видит две различных монеты, то либо оба других криптографа (Боб и Кэрол) сказали, "одинаковые" или оба сказали, "разные". (Помните, нечетное число кри п-тографов, говорящих "разные" указывает, что оплатил кто-то из них.). Если оба сказали "разные", то плател ь-щик - криптограф, сидящий ближе всего к монете, результат броска которой тот же, что и у скрытой монеты (брошенной между Бобом и Кэрол). Если оба сказали "одинаковые", то плательщик - криптограф, сидящий ближе всего к монете, результат броска которой отличается от результата броска скрытой монеты. Однако, если Алиса видит две одинаковых монеты, то или Боб сказал, "одинаковые", а Кэрол - "разные", или Боб сказал "разные", а Кэрол - "одинаковые". Если ли скрытая монета - такая же как и видимые ей две монеты, то пл а-тельщик - криптограф, который сказал, "разные". Если скрытая монета отлична от видимых ей двух монет, то плательщик - криптограф, который сказал "одинаковые". Чтобы определить, кто платил, во всех этих случаях Алиса должна знать результат броска монеты между Бобом и Кэрол.

Этот протокол может быть обобщен на любое количество криптографов, которые сидят по кругу и бросают монеты между собой. Каждая пара криптографов выполняет протокол. Конечно, они знают, кто платит, но кто-то, наблюдающий за протоколом может сказать только, что заплатил один из криптографов или NSA, но не сожет указать, какой из криптографов платил.

Применение этого протокола выходит далеко за пределы обеденного стола. Вот пример **безусловного отправителя и неотслеживаемого отправителя**. Группа пользователей сети может использовать этот протокол для отправления анонимных сообщений.

(1) Пользователи упорядочиваются по кругу.

(2) Через регулярные интервалы времени соседние пары пользователей бросают между собой монету, испол ь-зуя какой-нибудь безопасный от злоумышленников протокол бросания "честной" монеты.

(3) После каждого броска каждый пользователь объявляет либо "одинаковые", либо "разные".

Если Алиса хочет передать широковещательное сообщение, она просто начинает инвертировать свое утве р-ждение в тех раундах, которые соответствуют 1 в двоичном представлении ее сообщения. Например, если ее сообщение было "1001", она инвертирует ее утверждение, сообщит правду, сообщит правду, и затем инвертир у-ет снова утверждение. При условии, что результатом ее бросков было были "разные", "одинаковые", "одинаковые", "одинаковые", она будет говорить "одинаковые", "одинаковые ", "одинаковые", "разные".

Если Алиса замечает, что полный результат протокола не соответствует сообщению, которое она пробует п осылать, она понимает, что в это же время кто-то еще пытается посылать сообщение. Тогда она прекращает п е-редачу сообщения и выжидает случайное количество раундов перед очередной попыткой. Точные параметры должны быть выработаны на основе трафика сообщений в сети, но идея достаточно понятна.

Чтобы сделать дело еще более интересным, эти сообщения могут быть зашифрованы открытым ключом др у-гого пользователя. Затем, когда каждый принимает сообщение (практическая реализация должна включать стандартные заголовки и окончания сообщений), только определенный получатель сможет расшифровать и прочесть сообщение. Никто другой ничего не узнает про автора сообщения и не сможет определить получателя сообщения. Даже если удастся расшифровать сами сообщения, то анализ трафика, отслеживающий и собира ю-щий формы межпользовательского обмена, бесполе зен.

Альтернативой бросанию монет между соседними сторонами могло бы быть использование файла случа й-ных битов. Возможно, стороны могли бы хранить файл на CD-ROM, или один член пары мог бы генерировать пачку битов и посылать их другой стороне (конечно, в зашифрованном виде). Или, они могли бы договориться использовать совместно криптографически безопасный генератор псевдослучайных чисел, и каждый из них смог бы генерировать для протокола ту же самую последовательность псевдослучайных битов.

Проблемой этого протокола является то, что хотя мошенничающий участник и не сможет читать никаких с о-общений, он может незаметно испортить всю систему, постоянно обманывая на этапе (3). Существует модификация предыдущего протокола, позволяющая обнаружить вредительство [1578, 1242]. Эта проблема называется "Обедающие криптографы в дискотеке".

6.4 Электронные наличные

Наличные деньги - это проблема. Раздражает их носить, они способствуют распространению микробов, л юди могут красть их у Вас. Чеки и кредитные карточки уменьшили количество наличных денег, оборачивающи хся в обществе, но полное удаление наличных денег фактически невозможно. Этого никогда не произойдет; то рговцы наркотиками и политические деятели никогда этого не допустят. Чеки и кредитные карточки можно пр оследить, вы не можете скрыться от того, кому дали деньги.

С другой стороны, чеки и кредитные карточки позволяют людям вторгаться в вашу личную жизнь как ник огда прежде. Вы никогда не допустили бы, чтобы полиция всю жизнь ходила за вами по пятам, но полицейские могут проследить ваши финансовые операции. Они могут видеть, где вы покупаете газ, где вы покупаете еду, кому вы звоните по телефону, и все это не отрываясь от своих компьютерных терминалов. Люди должны уметь защитить свою анонимность, чтобы защитить свои личные тайны.

К счастью, существует сложный протокол, который разрешает использование заверенных, но неотслежива емых сообщений. Лоббист Алиса может передать **электронные деньги** конгрессмену Бобу так, чтобы газетный репортер Ева ничего не узнала бы об Алисе. Боб может затем вносить эти электронные деньги на свой банко вский счет, даже если банк не имеет об Алисе никакого представления. Но если Алиса пробует покупать кокаин на ту же самую порцию электронных денег, которую она использовала для подкупа Боба, она будет обнаружена банком. И если Боб пробует вносить порцию электронных денег на два различных счета, это будет обнаружено, но Боб, как и Алиса, останется он анонимным. Иногда это называется **анонимными электронными деньгами**, чтобы можно было отличить их от отслеживаемых электронных денег, типа креди тных карточек.

В подобных вещах существует большая общественная необходимость. С ростом использования Internet для коммерческих операций растет и потребность в секретности передаваемой по сети информации и анонимности при ведении дел. (Имеется немало причин для того, чтобы люди отказывались посылать номер их кредитной карточки по Internet.) С другой стороны, банки и правительства, по видимому, не пожелают уступить контроль над современными банковскими системами. Хотя им придется это сделать. Все, что потребуется, чтобы эле ктронные деньги вошли в моду, - это появление некоторого заслуживающего доверия учреждения, желающего преобразовывать цифры в реальные деньги.

Протоколы электронных денег очень сложны. Дальше мы шаг за шагом построим один из них. Более подробно об этом протоколе можно прочитать в [318, 339, 325, 335, 340]. Но помните, это только один из протоколов электронных денег, существуют и другие.

Протокол №1

Первые несколько протоколов представляют собой физические аналоги криптографических протоколов. Следующий протокол является упрощенным физическим протоколом для анонимных денежных чеков:

(1) Алиса готовит 100 анонимных денежных чеков по $1000 каждый.

(2) Алиса вкладывает каждый из них и листок копировальной бумаги в 100 различных конвертов и относит все конверты в банк.

(3) Банк открывает 99 конвертов и убеждается, что каждый чек выписан на $1000.

(4) Банк подписывает единственный оставшийся нераспечатанным конверт. С помощью копировальной бумаги подпись переводится на чек. Банк возвращает нераспечатанный конверт Алисе и списывает $1000 с ее счета.

(5) Алиса вскрывает конверт и отдает денежный чек продавцу.

(6) Продавец проверяет банковскую подпись, убеждаясь в законности денежного чека.

(7) Продавец относит денежный чек в банк.

(8) Банк проверяет свою подпись и начисляет $1000 на счет продавца.

Этот протокол работает. Банк не видит денежный чек, который он подписывает, поэтому, когда продавец принесет чек в банк, банк никогда не узнает, что это чек Алисы. Благодаря подписи банк убежден в законности чека. А из-за протокола "разрезать и выбрать" (см. раздел 5.1) банк уверен, что нераспечатанный денежный чек - на сумму $1000 (а не $100000 или $100000000). Он проверяет остальные 99 конвертов, поэтому вероятность обмана банка Алисой составляет только 1 процент. Конечно, банк назначит за обман достаточно большой штраф, такой, чтобы не стоило мошенничать. Ведь если банк просто откажется подписать последний чек (если Алиса поймана на обмане), не штрафуя Алису, она продолжит свои попытки, пока ей не повезет. Лучшее сре дство устрашения - это тюремное заключение.

Протокол №2

Предыдущий протокол не дает Алисе написать чек на сумму, отличную от заявленной, но он не мешает ей откс ерокопировать чек и использовать его дважды. Это называется **проблемой повторной оплаты**; для ее решение придется усложнить протокол:

(1) Алиса готовит 100 анонимных денежных чеков по $1000 каждый. К каждому денежному чеку она добавляет уникальную строку, выбранную случайным образом и достаточно длинную, чтобы вероятность использования этой строки другим человеком была пренебрежимо мала.

(2) Алиса вкладывает каждый из них и листок копировальной бумаги в 100 различных конвертов и относит все конверты в банк.

(3) Банк открывает 99 конвертов и убеждается, что каждый чек выписан на $1000.

(4) Банк подписывает единственный оставшийся нераспечатанным конверт. С помощью копировальной бумаги подпись переводится на чек. Банк возвращает нераспечатанный конверт Алисе и списывает $1000 с ее счета.

(5) Алиса вскрывает конверт и отдает денежный чек продавцу.

(6) Продавец проверяет банковскую подпись, убеждаясь в законности денежного чека.

(7) Продавец относит денежный чек в банк.

(8) Банк проверяет свою подпись и по своей базе данных убеждается, что денежный чек с такой уникальной строкой ранее не депонировался. Если это так, банк начисляет $1000 на счет продавца и записывает уникальную строку в базу данных.

(9) Если денежный чек уже был депонирован ранее, банк отказывается принять его.

Теперь, если Алиса попытается расплатиться ксерокопией денежного чека или продавец попытается депонировать денежный чек повторно, используя ксерокопию, банк узнает об этом.

Протокол №3

Предыдущий протокол защищает банк от мошенников, но не устанавливает их личность. Банк не знает, попытался ли человек, который получил чек (банк ничего не знает об Алисе), обмануть продавца, или продавец пытается обмануть банк. Эта неоднозначность исправляется следующим протоколом:

(1) Алиса готовит 100 анонимных денежных чеков по $1000 каждый. К каждому денежному чеку она добавляет уникальную строку, выбранную случайным образом и достаточно длинную, чтобы вероятность использования этой строки другим человеком была пренебрежимо мала.

(2) Алиса вкладывает каждый из них и листок копировальной бумаги в 100 различных конвертов и относит все конверты в банк.

(3) Банк открывает 99 конвертов и убеждается, что каждый чек выписан на $1000, и что все случайные строки различны.

(4) Банк подписывает единственный оставшийся нераспечатанным конверт. С помощью копировальной бумаги подпись переводится на чек. Банк возвращает нераспечатанный конверт Алисе и списывает $1000 с ее счета.

(5) Алиса вскрывает конверт и отдает денежный чек продавцу.

(6) Продавец проверяет банковскую подпись, убеждаясь в законности денежного чека.

(7) Продавец просит Алису написать случайную идентификационную строку на денежном чеке.

(8) Алиса выполняет это.

(9) Продавец относит денежный чек в банк.

(10) Банк проверяет свою подпись и по своей базе данных убеждается, что денежный чек с такой уникальной строкой ранее не депонировался. Если это так, банк начисляет $1000 на счет продавца и записывает уникальную строку в базу данных.

(11) Если уникальная строка уже есть в базе данных, банк отказывается принять денежный чек и сравнивает идентификационную строку на денежном чеке с хранимой в базе данных. Если они совпадают, то банк убеждается, что копия была снята с чека продавцом. Если идентификационные строки различны, то банк знает, что чек был скопирован человеком, который его готовил.

В этом протоколе предполагается, что продавец не может изменить идентификационную строку после того,

как Алиса напишет ее на денежном чеке. На денежном чеке мог бы находиться ряд небольших квадратов, которые по требованию торговца Алиса должна заполнить крестиками или ноликами. Денежный чек мог бы быть сделан из бумаги, которая рвется при исправлениях.

Так как продавец и банк взаимодействуют после того, как Алиса потратит деньги, продавцу могут всучить плохой денежный чек. Практические реализации этого протокола могли бы потребовать от Алисы подождать у кассового аппарата, пока продавец будет разбираться с банком, точно также, как это происходит сегодня при обработке платежей с использованием кредитных карточек.

Алиса также может приспособиться и к этому. Она может потратить копию денежного чека второй раз, написав ту же самую идентификационную строку на этапе (7). Если продавец не ведет базу данных уже полученных денежных чеков, он будет введен в заблуждение. Эту проблему устраняет следующий протокол.

Протокол №4

Если окажется, что человек, выписавший банковский чек, попытался обмануть продавца, то банк может захотеть личность этого человека. Чтобы сделать это, придется вернуться от физических аналогий в мир криптографии.

Чтобы спрятать имя Алисы в электронном чеке, можно воспользоваться методикой разделения секрета .

(1) Алиса готовит *n* анонимных денежных чеков на заданную сумму.

Каждый из чеков содержит уникальную строку, *X*, полученную случайным образом и достаточно длинную, чтобы вероятность появления двух одинаковых строк была пренебрежимо мала .

На каждом чеке есть также *n* пар битовых строк идентификации, I_1, I_2, ..., I_n. (Именно так, *n* различных пар на *каждом* чеке.) Каждая из этих пар генерируется следующим образом: Алиса создает строку, содержащую ее имя, адрес и прочие сведения, требуемые банком. Затем она делит эту строку на две части, используя протокол деления секрета (см. раздел 3.6) и вручает каждую часть, используя протокол вручения битов.

Например, I_{37} состоит из двух частей: I_{37_L} и I_{37_R}. Каждая часть представляет собой пакет врученных битов, который Алису могут попросить открыть, и чье открытое содержание может быть мгновенно проверено. Любая пара (например, I_{37_L} и I_{37_R}, но не I_{37_L} и I_{38_R}), раскрывает личность Алисы. Каждый из чеков выглядит следующим образом:

Сумма

Уникальная строка: *X*

Строки идентификации: $I_1 = (I_{1_L}, I_{1_R})$

$I_2 = (I_{2_L}, I_{2_R})$

....

$I_n = (I_{n_L}, I_{n_R})$

(2) Алиса маскирует все *n* чеков с помощью протокола слепой подписи и относит чеки в банк.

(3) Банк просит Алису снять маскировку с *n-1* денежных чеков и убеждается, что все они правильно оформлены. Банк проверяет сумму, уникальную строку и просит Алису раскрыть все строки идентификации.

(4) Если банк удовлетворен, не обнаружив попыток мошенничества, он подписывает оставшийся замаскированный денежный чек. Банк возвращает замаскированный чек Алисе и списывает сумму с ее счета.

(5) Алиса снимает маскировку с чека и тратит его у продавца.

(6) Продавец проверяет банковскую подпись, убеждаясь в законности денежного чека .

(7) Продавец случайным образом просит Алису раскрыть либо левые, либо правые половины всех строк идентификации на чеке. По сути, продавец выдает Алисе случайную *n*-битовую **строку-селектор**, b_1, b_2, ..., b_n. Левую или правую половину I_i откроет Алиса, зависит от значения b_i, 0 или 1.

(8) Алиса выполняет это.

(9) Продавец относит денежный чек в банк.

(10) Банк проверяет свою подпись и по своей базе данных убеждается, что денежный чек с такой уникальной строкой ранее не депонировался. Если это так, банк начисляет указанную сумму на счет продавца и запи-

сывает уникальную строку в базу данных.

(11) Если уникальная строка уже есть в базе данных, банк отказывается принять денежный чек и сравнивает идентификационную строку на денежном чеке с хранимой в базе данных. Если они совпадают, то банк убеждается, что чек был скопирован продавцом. Если идентификационные строки различны, то банк знает, что чек был скопирован человеком, который готовил этот денежный чек. Так как второй продавец, получивший чек, выдал Алисе другую, чем первый, строку-селектор, банк обнаружит, что для какой-то из позиций Алиса открыла левую половину одному продавцу, а правую - другому. Выполнив над этими половинами строки идентификации операцию XOR, банк определит личность Алисы.

Это весьма интересный протокол, поэтому посмотрим на него с разных сторон.

Может ли Алиса смошенничать? Ее электронные деньги представляют собой просто строку битов, которую она легко может скопировать. Потратить их в первый раз - не проблема, она просто выполнит протокол, и все пройдет без проблем. Продавец выдаст ей на этапе (7) случайную n-битовую строку-селектор, и Алиса откроет либо левую, либо правую половину каждой I_i на этапе (8). На этапе (10) банк запишет все эти данные вместе с уникальной строкой денежного чека.

Когда она попытается использовать те же электронные деньги второй раз, продавец (тот же или иной) выдаст ей на этапе (7) другую случайную n-битовую строку-селектор. Алиса должна выполнить этап (8), ее отказ немедленно встревожит продавца. Теперь, когда продавец приносит деньги в банк на этапе (10), банк немедленно заметит, что денежный чек с этой уникальной строкой уже был депонирован. Банк сравнивает открытые половины строк идентификации. Вероятность совпадения двух случайных строк-селекторов составляет один шанс из 2^n, этого не случится до следующего оледенения. Теперь банк находит пару, первая половина которой была открыта в первый раз, а вторая - во второй, выполняет над этими половинами операцию XOR и извлекает имя Алисы. Так банк узнает, кто попытался воспользоваться чеком дважды.

Что этот протокол не мешает Алисе мошенничать, но ее мошенничество почти наверняка будет обнаружено. Смошенничав, Алиса не сможет сохранить в тайне свою личность. Она не может изменить ни уникальную строку, ни какую-нибудь из строк идентификации, иначе испортится банковская подпись, и продавец немедленно заметит это на этапе (6).

Алиса могла бы попытаться подсунуть банку плохой денежный чек, такой, на котором строки идентификации не раскрывают ее имени, или, еще лучше, раскрывают имя кого-то еще. Вероятность, что такая уловка проскочит мимо банка на этапе (3), составляет 1 из n. Это не невозможно, но если штраф за мошенничество достаточно суров, Алиса не будет испытывать судьбу. Или вы можете увеличить число избыточных чеков, предъявляемых Алисой на этапе (1).

Может ли смошенничать продавец? Его шансы даже хуже. Он не может депонировать денежный чек два жды, банк заметит повторное использование строки-селектора. Он не сможет мошенничать, обвиняя Алису, так как только она может открыть любую строку идентификации.

Не поможет обмануть банк и любой сговор между Алисой и продавцом. Если банк подписал денежный чек с уникальной строкой, он может быть уверен в том, что этот чек будет оплачен только один раз.

А как насчет банка? Может ли он вычислить, что денежный чек, полученный от продавца, это и есть тот самый чек, который был подписан для Алисы? На этапах (2)-(5) Алиса защищена протоколом слепой подписи. Банк не сможет связать Алису и чек, даже если он полностью сохраняет запись каждой транзакции. Более того, даже объединившись, банк и продавец не смогут установить личность Алисы. Алиса может пройтись по магазину и, оставаясь полностью анонимной, купить то, что ей надо.

Может смошенничать Ева. Если она сможет подслушать линию связи между Алисой и продавцом, и если она сможет добраться до банка раньше продавца, она сможет первой депонировать чек. Банк примет его и, что хуже, когда продавец попытается депонировать свой чек, то он будет обвинен в мошенничестве. Если Ева украдет электронные деньги Алисы и успеет потратить их прежде Алисы, то в мошенничестве будет обвинена Алиса. Не существует способа помешать этому, и это является прямым следствием анонимности наличных. И Алиса, и продавец должны защищать свои биты так, как они защищали бы свои деньги.

Место этого протокола где-то между протоколом с посредником и самодостаточным протоколом. И Алиса, и продавец доверяют банку в том, что касается денег, но Алиса не должна доверять банку сведения о своих покупках.

Электронные наличные и идеальное приведение

У электронных наличных есть и своя темная сторона. Иногда людям не нужно так много секретности. Смотрите, как Алиса совершает идеальное преступление [1575]:

(1) Алиса крадет ребенка.

(2) Алиса готовит 10000 анонимных денежных чеков по $1000 (или другое количество чеков нужного ей достоинства).

(3) Алиса маскирует все 10000 денежных чеков, используя протокол слепой подписи. Она посылает их властям с угрозой убить ребенка, если не будут выполнены следующие инструкции :

(a) Все 10000 денежных чеков должны быть подписаны банком.

(b) Результаты должны быть опубликованы в газете.

(4) Власти соглашаются.

(5) Алиса покупает газету, снимает маскировку с денежных чеков и начинает тратить их . Не смогут найти ее, проследив за денежными чеками.

(6) Алиса освобождает ребенка.

Заметьте, что эта ситуация гораздо хуже чем при использовании любых физических носителей, например, наличных. Без физического контакта у полиции гораздо меньше шансов задержать похитителя .

Однако, в общем случае электронные наличные не слишком удобны для преступников . Проблема в том, что анонимность работает только для одной стороны - покупатель анонимен, а продавец нет . Более того, продавец не сможет скрыть факт получения денег. Электронные наличные помогут правительству определить, сколько денег вы зарабатываете, но определить, как вы их тратите, останется невозможным .

Реальные электронные наличные

Голландская компания, DigiCash, владеет большей частью патентов в области электронных наличных и реализовала протоколы электронных наличных в работающих продуктах owns. Если вы заинтересовались этим, обратитесь в DigiCash BV, Kruislaan 419, 1098 VA Amsterdam, Nethe rlands.

Другие протоколы электронных наличных

Существуют и другие протоколы электронных наличных, см. [707, 1554, 734, 1633, 973]. Ряд из них использует весьма изощренную математику. Различные протоколы электронных наличных можно разделить на различные категории. **Диалоговые** системы требуют, чтобы продавец связывался с банком при каждой продаже, что очень похоже на сегодняшний протокол для кредитных карточек . Если возникает какая-нибудь проблема, банк не принимает наличные, и Алиса не может смошенничать.

Автономные системы, подобные протоколу №4, не требуют соединения между продавцом и банком до окончания транзакции между продавцом и покупателем. Эти системы не помешают Алисе мошенничать, но вместо этого обнаружат ее мошенничество. Протокол №4 обнаруживает мошенничество Алисы, раскрывая ее личность при попытке мошенничать. Алиса знает о последствиях и, поэтому, не мошенничает .

Другой путь состоит в создании специальной интеллектуальной карты (см. Раздел 24.13), содержащей защищенную микросхему, называемую **наблюдателем** [332, 341, 387]. Микросхема-наблюдатель хранит мини-базу данных всех частей электронных наличных, потраченных этой интеллектуальной платой. Если Алиса попытается скопировать какие-то электронные наличные и потратить их дважды, внедренная микросхема-наблюдатель обнаружила бы такую попытку и не разрешила транзакцию. Так как микросхема-наблюдатель защищена от вмешательства извне, Алиса не сможет стереть мини-базу данных без разрушения интеллектуальной карты. Наличные деньги могут оборачиваться в экономике, когда они, наконец будут депонированы, банк может проверить наличные и определить мошенника, если произошел обман.

Протоколы электронных наличных можно разделить и по другому признаку . Номинал **электронных монет** фиксирован, людям, использующим такую систему, нужен ряд монет различных номиналов . **Электронные чеки** могут быть использованы для любых сумм до заданного максимума, а непотраченный остаток может быть возвращен на счет.

Двумя отличными совершенно отличающимися друг от друга автономными протоколами электронных монет являются [225, 226, 227] и [563, 564, 565]. Также можно предложить система NetCash (Сетевые наличные) с более слабыми свойствами [1048, 1049]. Другой новой системой является [289].

В [1211] Тацуаки Окамото (Tatsuaki Okamoto) и Казуо Охта (Kazuo Ohta) перечислили шесть свойств идеальной системы электронных наличных:

1. Независимость. Безопасность электронных наличных не зависит от местонахождения . Наличные могут быть переданы по компьютерным сетям.
2. Безопасность. Электронные наличные нельзя скопировать и повторно использовать .
3. Тайна личности (Неотслеживаемость). Тайна личности пользователя защищена, связь между пользо-

вателем и его покупками обнаружить невозможно.

4. Автономный платеж. Когда пользователь расплачивается за покупку электронными наличными, протокол между пользователем и продавцом выполняется автономно. То есть, магазину не нужно соединяться с центральным компьютером для обработки платежа пользователя.
5. Перемещаемость. Наличные могут передаваться другим пользователям.
6. Делимость. Заданная сумма электронных наличных может быть поделена на части меньшей суммы. (Конечно, общая сумма в конце должна сойтись.)

Ранее приведенные протоколы удовлетворяют требованиям 1, 2, 3 и 4, но не удовлетворяют требованиям 5 и 6. Ряд диалоговых систем электронных наличных удовлетворяет всем требованиям кроме 4 [318, 413, 1243]. Первая автономная система, удовлетворяющая требованиям 1, 2, 3 и 4, похожая на одну из описанных, была предложена [339]. Окамото и Охта предложили систему, удовлетворяющая требованиям с 1 по 5 [1209], они также предложили систему, удовлетворяющую требованиям с 1 по 6, объем данных для одного платежа составил приблизительно 200 мегабайт. Другая автономная система электронных монет с возможностью деления описана в [522].

Схема электронных наличных, предложенная теми же авторами [1211], удовлетворяет требованиям с1 по 6 без необходимости такого огромного объема данных. Общий объем данных для одного электронного платежа составляет около 20 килобайт, и протокол может быть выполнен за несколько секунд. Авторы рассматривают эту схему как первую идеальную систему неотслеживаемых электронных наличных.

Анонимные кредитные карточки

Этот протокол [988] для защиты личности клиента использует несколько различных банков. Каждый клиент имеет счет в двух различных банках. Первый банк, которому известна личность человека, может зачислять деньги на его счет. Второй банк знает клиента только под псевдонимом (подобно номерному счету в швейцарском банке).

Клиент может брать деньги из второго банка, доказывая, что он является владельцем счета. Но, этот банк не знает личности человека и не может зачислять деньги на его счет. Первый банк знает клиента и перечисляет деньги во второй банк, не зная псевдонима. Затем клиент анонимно тратит эти деньги. В конце месяца второй банк выставляет счет первому банку, веря, что он его оплатит. Первый банк передает счет клиенту, веря, что тот его оплатит. Когда клиент оплачивает счет, первый банк перечисляет дополнительные деньги во второй банк. Все транзакции проводятся через посредника, который действует подобно электронному Федеральному Резерву: оплачивает банковские счета, регистрирует сообщения и создает контрольный след.

Обмены между клиентом, продавцом и различными банками особо выделены в [988]. Если все не сговариваются против клиента, его анонимность гарантирована. Однако, это не электронные наличные, банк слишком легко может мошенничать. Протокол позволяет клиентам пользоваться преимуществами кредитных карточек, не раскрывая своей личности.

Часть 2

Криптографические методы

Глава 7
Длина ключа

7.1 Длина симметричного ключа

Безопасность симметричной криптосистемы является функцией двух факторов: надежности алгоритма и длины ключа. Первый более важен, но роль второго легче продемонстрировать.

Пусть надежность алгоритма совершенна. На практике этого чрезвычайно трудно достигнуть, но в примере - достаточно легко. По совершенством я подразумеваю отсутствие лучшего пути взлома криптосистемы, чем вскрытие грубой силой с помощью перебора всех возможных ключей.

Для выполнения такого вскрытия криптоаналитику требуется кусочек шифротекста и соответствующего о т-крытого текста, вскрытие грубой силой представляет собой вскрытие с известным открытым текстом. Для блочного шифра криптоаналитику понадобится блок шифротекста и соответствующий открытый текст: обычно 64 бита. Заполучить такие кусочки открытого текста и шифротекста легче, чем можно себе представить. Криптоаналитик может получить каким-то образом копию открытого текста сообщения и перехватить соответствующий шифротекст. Он может знать что-то о формате шифротекста: например, что это файл в формате WordPerfect, или у него есть стандартный заголовок сообщения электронной почты, или файл каталога UNIX, или изображение в формате TIFF, или стандартная запись в базе данных клиентов. Все эти форматы содержат некоторые предопределенные байты. Криптоаналитику для такого вскрытия не нужно много открытого текста.

Рассчитать сложность вскрытия грубой силой нетрудно. Если используется 8-битовый ключ, то существует 2^8, или 256, возможных ключей. Следовательно, для обнаружения правильного ключа потребуется, самое бол ьшее, 256 попыток, с 50-процентной вероятностью найти нужный ключ после половины попыток. Если длина ключа равна 56 битам, то существует 2^{56} возможных ключей. Если компьютер может проверить миллион кл ючей в секунду, поиск нужного ключа займет в среднем 2285 лет. Если используется 64-битовый ключ, то тому же суперкомпьютеру понадобится около 585000 лет, чтобы найти правильный ключ среди 2^{64} возможных ключей. Если длина ключа равна 128 битам поиск ключа займет 10^{25} лет. Возраст вселенной составляет всего 10^{10} лет, поэтому 10^{25} лет - это большое время. При 2048-битовом ключе миллион компьютеров, работая параллел ьно и проверяя миллион ключей в секунду, потратят 10^{587} лет в поисках ключа. К этому времени вселенная давно расширится, превратившись в ничто или сожмется.

Прежде чем кидаться изобретать криптосистему с 8-килобайтным ключом, вспомните, что другой стороной является надежность: алгоритм должен быть настолько безопасен, чтобы лучшего способа, чем вскрывать его грубой силой, не существовало. Это не так просто, как может показаться. Криптография - это тонкое искусство. Выглядящие совершенными криптосистемы часто оказываются чрезвычайно слабыми. Пара изменений, внесенных в сильные криптосистемы, может резко ослабить их. Криптографам-любителям следует подвергать по чти параноидальному сомнению каждый новый алгоритм. Лучше доверять алгоритмам, над которыми годами бились профессиональные криптографы, не сумев взломать их, и не обольщаться утверждениями конструкторов алгоритмов об их грандиозной безопасности.

Вспомните важный момент из раздела 1.1: безопасность криптосистем должна основываться на ключе, а не особенностях алгоритма. Предположим, что криптоаналитику известны все подробности вашего алгоритма. Предположим, что у него есть столько шифротекста, сколько ему нужно, и что он попытается выполнить инте нсивное вскрытие с использованием только шифротекста. Предположим, что он попытается выполнить вскрытие с использованием открытого текста, имея в своем распоряжении столько данных, сколько ему нужно. Предположим даже, что он попытается выполнить вскрытие с использованием выбранного открытого текста. Если ваша криптосистема останется безопасной даже перед лицом всех подобных опасностей, то... у вас действительно что-то есть.

Несмотря на это предупреждение пространство, предоставляемое криптографией для маневра, достаточно велико. В действительности, безопасность такого типа во многих практических ситуациях не нужна. У большинства врагов нет таких знаний и вычислительных средств, как у больших правительств, а тем, кто обладает такими возможностями, может оказаться ненужным взламывать вашу криптосистему. Если вы организуете заговор с целью свергнуть большое правительство, проверенные и правильные алгоритмы, приведенные в конце этой книги, будут для вас жизненно необходимы. А все остальные пусть просто получат удовольс твие.

Оценки времени и стоимости вскрытия грубой силой

Вспомните, что вскрытие грубой силой обычно является вскрытием с использованием известного открытого текста, для этого нужно немного шифротекста и соответствующего открытого текста. Если вы предполагаете, что наиболее эффективным способа взлома алгоритма является вскрытие грубой силой - большое допущение - то ключ должен быть достаточно длинным, чтобы сделать вскрытие невозможным. Насколько длинным?

Скорость вскрытия грубой силой определяется двумя параметрами: количеством проверяемых ключей и скоростью проверки одного ключа. Большинство симметричных алгоритмов в качестве ключа могут использовать в качестве ключа любую битовую последовательность фиксированной длины. Длина ключа DES составляет 56 бит, всего может быть 2^{56} возможных ключей. Длина ключей для ряда алгоритмов, обсуждаемых в этой книге, равны 64 битам, всего может быть 2^{64} возможных ключей. Другие алгоритмы используют 128-битовые ключи.

Скорость, с которой может быть проверен каждый ключ, имеет менее важное значение. Для проводимого анализа я предполагаю, что скорость проверки ключа для каждого алгоритма примерно одинакова. В действительности скорость проверки одного алгоритма может быть в два, три или даже десять раз выше чем другого. Но так как для тех длин ключей, для которых мы проводим поиск, время поиска в миллионы раз больше, чем время проверки одного ключа, небольшие отличия в скорости проверки не имеют значения.

В криптологической среде большинство споров по поводу вскрытия грубой силой сконцентрированы вокруг алгоритма DES. В 1977 году Уитфилд Диффи и Мартин Хеллман [497] сформулировали условия существования специализированной машины по взлому DES. Эта машина состоит из миллионов микросхем, каждая из которых проверяет миллион ключей в секунду. Такая машина за два часа сможет проверить 2^{56} за 20 часов. При вскрытии алгоритма с 64-битовым ключом проверка всех 2^{64} потребует 214 дней.

Задача вскрытия грубой силой как будто специально придумана для параллельных процессоров. Каждый процессор проверяет подмножество пространства ключей. Процессорам не нужно обмениваться между собой информацией, единственным используемым сообщением будет сообщение, сигнализирующее об успехе. Не требуется и доступ к одному участку памяти. Сконструировать машину с миллионом процессоров, каждый из которых работает независимо от других, нетрудно.

Сконструировать машину для взлома грубой силой Майкл Винер решил [1597, 1598]. (Он сконструировал машину для DES, но анализ может быть выполнен почти для всех алгоритмов.) Он разработал специализированные микросхемы, платы и стойки, оценил затраты и сделал вывод, что за миллион долларов можно построить машину, которая сможет взломать 56-битный ключ DES key в среднем за 3.5 часа (и наверняка за 7 часов). Соотношение стоимость/скорость является линейным. Для ряда длин ключей эти значения обобщены в 6-й. Вспомните о законе Мура: мощь вычислительных средств приблизительно каждые 18 месяцев. Это означает, что затраты будут уменьшаться на порядок каждые пять лет, и то, что в 1995 году стоит миллион долларов, в 2000 году будет стоить около 100000 долларов. Еще более упростить процесс вычислений могла бы конвейеризация [724].

Для 56-битовых ключей эти суммы оказываются вполне по карману большинству крупных корпораций и многим криминальным организациям. Военные бюджеты большинства промышленно развитых стран могут позволить взламывать и 64-битные ключи. Вскрытие 80-битного ключа все еще за пределами возможного, но если текущая тенденция сохранится, то через каких-нибудь тридцать лет все может измениться.

Конечно, нелепо прогнозировать компьютерную мощь на 35 лет вперед. Технологические прорывы, популярные в научной фантастике, могут сделать эти прогнозы смешными. С другой стороны, неизвестные в настоящее время физические ограничения могут сделать эти прогнозы нереально оптимистичными. В криптографии умнее быть пессимистом. Применение в алгоритме 80-битного ключа кажется недостаточно дальновидным. Используйте ключ, длина которого, по меньшей мере, 112 бит.

Табл. 7-1.
Оценки среднего времени для аппаратного вскрытия грубой силой в 1995 году.

	Длина ключей в битах					
Стоимость	40	56	64	80	112	128
$100 К	2 секунды	35 часов	1 год	70000 лет	10^{14} лет	10^{19} лет
$1 М	0.2 секунды	3.5 часа	37 дней	7000 лет	10^{13} лет	10^{18} лет
$10 М	0.02 секунды	21 минута	4 дня	700 лет	10^{12} лет	10^{17} лет
$100 М	2 миллисекунды	2 минуты	9 часов	70 лет	10^{11} лет	10^{16} лет
$1 Г	0.2 миллисекунды	13	1 час	7 лет	10^{10} лет	10^{15} лет
$10 Г	0.02. миллисекунды	1 секунда	5.4 минуты	245 дней	10^{9} лет	10^{14} лет
$100 Г	2 микросекунды	0.1 секунды	32 секунд	24 дня	10^{8} лет	10^{13} лет
$1 Т	0.2 микросекунды	0.01 секунды	3 секунды	2.4 дня	10^{7} лет	10^{12} лет
$10 Т	0.02 микросекунды	1 миллисекунда	0.3 секунды	6 часов	10^{6} лет	10^{11} лет

Если взломщик очень сильно хочет взломать ключ, все, что ему нужно, это потратить деньги. Следовательно, стоит попытаться определить минимальную "цену" ключа: в пределах какой стоимости сведений можно пользоваться одним ключом прежде, чем его вскрытие станет экономически выгодным ? Крайний случай: если шифрованное сообщение стоит $1.39, то нет финансового смысла устанавливать аппаратуру стоимостью 10 миллионов долларов для взлома этого ключа. С другой стороны, если стоимость открытого текста - 100 миллионов долларов, то дешифрирование этого одиночного сообщения вполне окупит стоимость аппаратуры взлома. Кроме того, стоимость многих сообщений со временем очень быстро падает.

Программное вскрытие

Без специализированной аппаратуры и огромных параллельных машин вскрытие грубой силой намного сложнее. Программное вскрытие в тысячи раз медленнее, чем аппаратное.

Реальная угроза программного вскрытия грубой силой страшна не своей неизбежностью, а тем, что такое вскрытие "свободно". Ничего не стоит загрузить простаивающий микрокомпьютер проверкой возможных ключей. Если правильный ключ будет найден - замечательно, если нет - ничего не потеряно. Ничего не стоит использовать для этого целую сеть микрокомпьютеров. В недавних экспериментах с DES 40 рабочих станций в течение одного дня сумели проверить 2^{34} ключей [603]. При этой скорости для проверки всех ключей потребуется четыре миллиона дней, но если попытки вскрытия будут предприняты достаточным количеством людей, то кому-нибудь где-нибудь повезет. Как было сказано в [603]:

> Основной угрозой программного вскрытия является слепое везение. Представьте себе университетскую сеть из 512 объединенных в сеть рабочих станций. Для некоторых университетских городков это сеть весьма среднего размера. Такие сети могут даже расползтись по всему миру, координируя свою деятельность по электронной почте. Пусть каждая рабочая станция способна работать (с алгоритмом) со скоростью 15000 шифрований в секунду. ... С учетом накладных расходов на проверку и смену ключей уменьшим скорость до . . . 8192 проверок в секунду на машину. Чтобы, используя описанную систему, исчерпать пространство (56-битовых) ключей потребуется 545 лет (в предположении, что сеть тратит на эту задачу 24 часа в сутки). Заметим, однако, что с помощью таких вычислений сторонники нашего студента получают один шанс из 200000 раскрыть ключ в течение одного дня. За долгий уикенд их шансы возрастают до одного из шестидесяти шести тысяч. Чем быстрее их аппаратура, или чем больше задействовано машин, тем лучше становятся их шансы. Вероятность заработать на жизнь, выигрывая на скачках, невысока, но разве не эти выигрыши заполняют собой пресс-релизы. К примеру, это гораздо большая вероятность, чем возможность выигрыша в правительственных лотереях. "Один на миллион"? "Один раз за тысячу лет "? Больше невозможно с полной ответственностью делать такие заявления. Является ли приемлемым этот продолжающийся риск?

Использование алгоритма с 64-битовым ключом вместо 56-битового ключа делает это вскрытие в 256 раз сложнее. А 40-битовый ключ делает картину просто безрадостной. Сеть из 400 компьютеров с производительностью 32000 шифрований в секунду может за день выполнить вскрытие грубым взломом 40-битового ключа. (В 1992 году алгоритмы RC2 и RC4 было разрешено экспортировать с 40- битовым ключом - см. раздел 13.8.)

128-битовый ключ делает нелепой даже мысль о вскрытии грубым взломом. По оценке промышленных экспертов к 1996 году в мире будет использоваться 200 миллионов компьютеров. Эта оценка включает все - лт гигантского мэйнфрейма Cray до блокнотных компьютеров. Даже если все эти компьютеры будут брошены на вскрытие грубой силой, и каждый из них будет выполнять миллион шифрований в секунду, время раскрытия ключа все равно будет в миллион раз больше времени существования вселенной.

Нейронные сети

Нейронные сети не слишком пригодны для криптоанализа, в первую очередь из-за формы пространства решений. Лучше всего нейронные сети работают с проблемами, имеющими непрерывное множество решений, одни из которых лучше других. Это позволяет нейронным сетям обучаться, предлагая все лучшее и лучшие решения. Отсутствие непрерывности в алгоритме почти не оставляет места обучению : вы либо раскроете ключ, либо нет. (По крайней мере, это верно при использовании любого хорошего алгоритма .) Нейронные сети хорошо работают в структурированных средах, где обучение возможно, но не в высокоэнтропийном, кажущемся случайным мире криптографии.

Вирусы

Самая большая трудность в получении миллионов компьютеров для вскрытия грубым взломом - это убедить миллионы компьютерных владельцев принять участие во вскрытии. Вы могли бы вежливо попросить, но это требует много времени, и они могут сказать нет. Вы могли бы пробовать силой ворваться в их компьютеры, но это потребует еще больше времени и может закончиться вашим арестом. Вы могли бы также использовать компьютерный вирус, чтобы распространить программу взлома среди как можно большего количества компьютеров.

Эта особенно коварная идея впервые появилась в [1593]. Взломщик пишет и выпускает на волю компьютерный вирус. Этот вирус не переформатирует жесткий диск, не удаляет файлы, но во время простоя компьютера он работает на криптоаналитической проблемой грубого взлома. Различные исследования показали, что комп ь-

ютер простаивает от 70 до 90 процентов времени, так что у вируса не будет проблем с временем для решения этой задачи. Если он нетребователен и в других отношениях, то его работа даже не будет заметна.

В конце концов, одна из машина наткнется на правильный ключ. В этот момент имеются два варианта пр одолжения. Во первых, вирус мог бы породить другой вирус. Он не делал бы ничего, кроме самовоспроизведения и удаления всех найденных копий вскрывающего вируса, но содержал бы информацию о правильном ключе. Этот новый вирус просто распространялся бы среди компьютеров, пока не добрался бы до компьютера челов ека, который написал первоначальный вирус.

Другим, трусливым подходом бал бы вывод на экран следующего сообщения :

> В этом компьютере есть серьезная ошибка. Пожалуйста позвоните 1-8001234567 и продиктуйте оператору следующее 64-битовое число:
>
> xxxx xxxx xxxx xxxx
>
> Первому, кто сообщит об этой ошибке будет выплачено вознаграждение 100 долларов.

Насколько эффективно такое вскрытие? Пусть типичный зараженный компьютер проверяет тысячу ключей в секунду. Эта скорость намного меньше потенциальных возможностей компьютера, ведь мы полагаем, что он иногда будет делать и другие вещи. Предположим также, что типичный вирус инфицирует 10 миллионов машин. Этот вирус может вскрыть 56-битовый ключ за 83 дня, а 64 битовый - за 58 лет. Вам возможно пришлось бы подкупить разработчиков антивирусного программного обеспечения, но это уже ваши проблемы . Любое увеличение скорости компьютеров или распространения вируса, конечно, сделало бы это нападение более эффективным.

Китайская лотерея

Китайская Лотерея - эклектический, но возможный способ создания громадной параллельной машины для криптоанализа [1278]. Вообразите, что микросхема, вскрывающая алгоритм грубой силой со скоростью милл ион проверок в секунду, встроена в каждый проданный радиоприемник и телевизор. Каждая микросхема запр ограммирована для автоматической проверки различного набора ключей после получения пары открытый текст/шифротекст по эфиру. Каждый раз когда китайское правительство хочет раскрыть ключ, оно передает исходные данные по радио. Все радиоприемники и телевизоры в стране начинают пыхтеть. В конечном счете, правильный ключ появляется на чьем-нибудь дисплее. Китайское правительство платит приз тому человеку - это гарантирует, что результат будет сообщен быстро и правильно, и также способствует рыночному успеху р адиоприемников и телевизоров с микросхемами вскрытия.

Если у каждого человека в Китае, будь то мужчина, женщина или ребенок, есть радиоприемник или телев изор, то правильное значение 56-битового ключа появится через 61 секунду. Если радиоприемник или телевизор есть только у каждого десятого китайца(что близко к действительности), то правильный ключ появится через 10 минут. Правильный 64-битовый ключ будет раскрыт через 4.3 часа (43 часа, если радиоприемник или телевизор есть только у каждого десятого китайца).

Чтобы сделать такое вскрытие возможным на практике, необходимо сделать ряд модификаций. Во первых, проще, чтобы каждая микросхема проверяла случайные, а не уникальные ключи . Это сделает вскрытие на 39% медленнее, что не очень важно для чисел такого масштаба. Затем, Китайская коммунистическая партия должна принять решение, что каждый должен включать свой приемник или телевизор в определенное время, чтобы гарантировать работу всех приемных устройств во время передачи пары открытый текст/шифротекст . Наконец, каждому должно быть приказано позвонить в Центр - или как он там называется - когда ключ появляется у него на экране и зачитать строку чисел, появившуюся на экране .

Эффективность Китайской лотереи для различных стран и различных длин ключа показана в 5-й. Ясно, что Китай оказался бы в лучшем положении, если бы у каждого китайца - мужчины, женщины или ребенка - бал свой приемник или телевизор. В Соединенных штатах живет меньше людей, но гораздо больше аппаратуры . Штат Вайоминг самостоятельно сможет взломать 56-битовый ключ меньше, чем за день.

Табл. 7-2.
Оценки среднего времени вскрытия грубой силой при китайской лотерее

(Все данные взяты из *World Almanac and Book of Facts* за 1995 год.)

Страна	Население	Количество телевизоров/радиоприемников	Время взлома	
			56 бит	64 бита
Китай	1190431000	257000000	280 секунд	20 часов
США	260714000	739000000	97 секунд	6.9 часа
Ирак	19890000	4730000	4.2 часа	44 дня
Израиль	5051000	3640000	5.5 часа	58 дней
Вайоминг	470000	1330000	15 часов	160 дней
Виннемукка, Невада	6100	17300	48 дней	34 года

Биотехнология

Если возможно создание биомикросхем, то было бы глупо не попытаться использовать их в инструмента криптоанализа вскрытием грубой. Рассмотрим гипотетическое животное, называемое "DESозавром" [1278]. Оно состоит из биологических клеток, умеющих проверять возможные ключи. Пары "открытый текст/шифротекст" поступают в клетки по некоторому оптическому каналу (видите ли, все эти клетки прозрачны). Решения доставляются к органам речи DESозавра с помощью специальных клеток, путешествующих по кровеносной системе животного.

Типичный динозавр состоит из 10^{14} клеток (без бактерий). Если каждая из них выполняет миллион шифрований в секунду (неплохой результат), вскрытие 56-битового ключа займет семь десятитысячных секунды. Вскрытие 64-битового ключа потребует меньше, чем две десятых секунды. Вскрытие 8-битового ключа все же продлится 10^{11} лет.

Другой биологический подход состоит в использовании генетически проектируемых криптоаналитических морских водорослей, которые умеют выполнять вскрытие криптографических алгоритмов грубой силой [1278]. Такие организмы, покрыв большую область, позволили бы создать распределенную машину с большим количеством процессоров. Пара "открытый текст/шифротекст" могла бы передаваться по радио через спутник. Обнаружение результата организмом могло бы стимулировать близлежащие ячейки изменить цвет, сообщая решение обратно на спутник.

Предположим, что типичная клетка морской водоросли - это кубик со стороной 10 микрон (возможно, это оценка сверху, следовательно 10^{15} клеток заполнят кубический метр. Выплесните их в океан, покрывая 200 квадратных миль (518 квадратных километров) на глубину один метр (это ваши проблемы, как осуществить это - я подаю только идею), и у вас будет 10^{23} водорослей (более чем сотней миллиарда галлонов), плавающих в океане. (Для сравнения, из танкера *Valdez* вытекло 10 миллионов галлонов нефти.) Если каждая из них может проверять миллион ключей в секунду, то для 128-битового алгоритма они раскроют ключ в только спустя 100 лет. (Возникшее при этом цветение морских водорослей - это ваша проблема.) Крупные достижения в быстродействии морских водорослей, их диаметр или даже размеры пятна в океане могут заметно уменьшить эти значения. Даже не спрашивайте меня о нанотехнологии.

Термодинамические ограничения

Одним из следствий закона второго термодинамики является то, что для представления информации необходимо некоторое количество энергии. Запись одиночного бита, изменяющая состояние системы, требует количества энергии не меньше чем kT; где T - абсолютная температура системы и k - постоянная Больцмана. (Не волнуйтесь, урок физики уже почти закончен.)

Приняв, что $k = 1.38*10^{-16}$ эрг/K, и что температура окружающей вселенной 3.2K, идеальный компьютер, работая при 3.2K, потреблял бы $4.4*10^{-16}$ эрга всякий раз, когда он устанавливает или сбрасывает бит. Работа компьютера при температуре более низкой, чем температура космического пространства, потребовала бы дополнительных расходов энергии для отвода тепла.

Далее, энергия, излучаемая нашим Солнцем за год, составляет около $1.21*10^{41}$ эргов. Это достаточно для выполнения $2*10^{56}$ перемен бита в нашем идеальном компьютере, а этого, в свою очередь, хватит для того, чтобы 187-битовый счетчик пробежал все свои значения. Если мы построим вокруг Солнца сферу Дайсона и перехватим без всяких потерь всю его энергию за 32 года, мы сможем получить компьютер для вычисления 2^{192} чисел. Конечно, энергии для проведения каких-нибудь полезных вычислений с этим счетчиком уже не

останется.

Но это только одна жалкая звезда. При взрыве типичной сверхновой выделяется около 10^{51} эргов. (В сто раз больше энергии выделяется в виде нейтрино, но пусть они пока летают.) Если всю эту энергию удастся бросить на одну вычислительную оргию, то все свои значения сможет принять 219-битовый счетчик.

Эти числа не имеют ничего общего с самой аппаратурой, они просто показывают максимальные значения, обусловленные термодинамикой. Кроме того, эти числа наглядно демонстрируют, что вскрытие грубой силой 256-битового ключа будет невозможно, пока компьютеры построены из обычной материи и располагаются в обычном пространстве.

7.2 Длина открытого ключа

Однонаправленные функции обсуждались в разделе 2.3. Однонаправленной функцией является умножение двух больших простых чисел, получить произведение, перемножив числа, нетрудно, но нелегко разложить произведение на множители и получить два больших простых числа (см. раздел 11.3). Криптография с открытыми ключами использует эту идею для создания однонаправленной функции с люком. На самом деле, это ложь, *не доказано*, что разложение на множители является тяжелой проблемой (см. раздел 11.4). Насколько сегодня известно, это похоже на правду. И даже если это так, никто не может доказать, что трудные проблемы действительно трудны. Все считают, что разложение на множители является трудной задачей, но это никогда не было доказано математически.

На этом стоит остановиться поподробнее. Легко представить, что лет через 50 мы соберемся вместе, вспоминая старое доброе время, когда все люди считали, что разложение на множители было трудным и лежало в основе криптографии, а различные компании делали из этого деньги. Легко вообразить, что будущие достижения в теории чисел упростят разложение на множители или достижения теории сложности сделают разложение на множители тривиальным. Нет причин верить в это - и большинство людей, знающих достаточно, чтобы иметь собственное мнение, скажет вам, что подобное развитие событий является маловероятным - но нет и причин верить, что такого прорыва не случится.

В любом случае, доминирующие сегодня алгоритмы шифрования с открытым ключом основаны на трудности разложения на множители больших чисел, которые являются произведением двух больших простых чисел. (Другие алгоритмы основаны на так называемой Дискретной проблемой логарифма, но пока предположим, что к ней применимы те же рассуждения.) Эти алгоритмы также восприимчивы к вскрытию грубой силой, но по разному. Взлом этих алгоритмов состоит не из перебора всех возможных ключей, а из попыток разложения больших чисел на множители (или взятия дискретных логарифмов в очень большом конечной области - точно такая же проблема). Если число слишком мало, вы никак не защищены. Если число достаточно велико, то вы надежно защищены против всей вычислительной мощи мира, если она будет биться над этой задачей с настоящего времени до тех пор, пока Солнце не станет сверхновой - таково сегодняшнее понимание математики этой проблемы. В разделе 11.3 разложение на множители рассматривается математически подробно, а здесь я ограничусь оценкой времени разложения на множители чисел различной длины.

Разлагать большие числа на множители нелегко, но, к несчастью для проектировщиков алгоритмов, этот процесс становится все легче. Что еще хуже, он становится легче ч большей скоростью, чем предсказывалось математиками. В 1976 году Ричард Гай (Richard Guy) писал: "Я был бы немало удивлен, если бы кто-нибудь научился разлагать на множители произвольные числа порядка 10^{80} в течение данного столетия" [680]. В 1977 году Рон Ривест (Ron Rivest) заявил, что разложение на множители 125-разрядного числа потребует 40 квадриллионов лет [599]. В 1994 году было разложено на множители 129-разрядное число [66]. Если из этого и можно сделать какие-то выводы, то только то, что предсказывать глупо.

В 4-й приведены результаты разложения на множители за последнюю дюжину лет. Самым быстрым алгоритмом разложения на множители является квадратичное решето (см. раздел 11.3).

Эти числа сильно пугают. Сегодня 512-битовые числа уже используются в операционных системах. Разложение их на множители, и полная компрометация, таким образом, системы защиты, вполне реально. Червь в Internet мог бы сделать это в течение уикенда.

Табл. 7-3.
Разложение на множителя с помощью "квадратичного решета"

Год	Число десятичных разрядов в разложенном числе	Во сколько раз сложнее разложить на множители 512-битовое число
1983	71	>20 миллионов
1985	80	>2 миллионов
1988	90	250000
1989	100	30000
1993	120	500
1994	129	100

Вычислительная мощь обычно измеряется в mips-годах: годовая работа компьютера, выполняющего миллион операций в секунду (one-million-instruction-per-second, mips), или около $3*10^{13}$ операций. Принято, что машина с производительностью 1 mips-год эквивалентна VAX 11/780 компании DEC. То есть, mips-год - это год работы компьютера VAX 11/780 или эквивалентного. (100 МГц Pentium - это машина в 50 mips, а Intel Paragon из 1800 узлов - примерно 50000 mips.)

В 1983 году разложение на множители 71-разрядного числа требовало 0.1 mips-года, в 1994 году разложение на множители 129-разрядного числа потребовало 5000 mips-лет. Такой взлет вычислительной мощности обусловлен, в основном, введением распределенных вычислений, использующих время простоя сети рабочих станций. Этот подход был предложен Бобом Силверманом (Bob Silverman) и полностью разработан Аржаном Ленстрой (Arjen Lenstra) и Марком Манассом (Mark Manasse). В 1983 году разложение на множители использовало 9.5 часов процессорного времени на единственном компьютере Cray X-MP, в 1994 году разложение на множители заняло 5000 mips-лет и использовало время простоя 1600 компьютеров во всем мире в течение приблизительно восьми месяцев. Современные методы разложения на множители позволяют использовать подобные распределенные вычисления.

Картина даже продолжает ухудшаться. Новый алгоритм разложения на множители - решето общего поля чисел - заменил квадратичное решето. В 1989 году математики сказали бы вам, что решето общего поля чисел никогда не будет иметь практического значения. В 1992 году они сообщили бы, что оно реализуемо, но дает выигрыш по сравнению с квадратичным решетом только для чисел со 130-150 десятичными разрядами и больших. Сегодня известно, что этот новый алгоритм быстрее, чем квадратичное решето, для чисел значительно меньших, чем 116-разрядные [472, 635]. Решето общего поля чисел может разложить на множители 512-битовое число в 10 раз быстрее, чем квадратичное решето. На 1800-узловом компьютере Intel Paragon выполнение этого алгоритма заняло бы меньше года. В 3rd показано количество mips-лет, которое требуется для разложения чисел различных размеров при использовании современных реализаций решета общего поля чисел [1190].

Табл. 7-4.
Разложение на множители с помощью решета общего поля чисел

Количество бит	Сколько mips-лет нужно для разложения
512	30000
768	$2*10^{8}$
1024	$3*10^{11}$
1280	$1*10^{14}$
1536	$3*10^{16}$
2048	$3*10^{20}$

Кроме того, решето общего поля чисел становится все быстрее и быстрее. Математики изобретают новые трюки, оптимизации, методы, и нет причин считать, что эта тенденция оборвется. Близкий алгоритм, решето специального поля чисел, уже может разлагать на множители числа определенной специализированной формы - обычно не используемые в криптографии - гораздо быстрее, чем решето общего поля чисел может разложить на

множители любые числа того же размера. Разумно предположить, что решето общего поля чисел может быть оптимизировано, чтобы достичь такой же скорости [1190], возможно, что NSA уже знает, как это сделать. В 2-й показано количество mips-лет, требуемое для разложения чисел различной длины при помощи решета специального поля чисел [1190].

Табл. 7-5.
Разложение на множители с помощью решета специального поля чисел

Количество бит	Сколько mips-лет нужно для разложения
512	<200
768	100000
1024	$3*10^7$
1280	$3*10^9$
1536	$2*10^{11}$
2048	$4*10^{14}$

В 1991 году участники семинара Европейского института безопасности систем (European Institute for System Security) согласились, что 1024-битовых модулей будет достаточно для длительного хранения секретов до 2002 года [150]. Однако, они предупреждали: "Хотя участники этого семинара являются лучшими специалистами в соответствующих областях, это заявление (по поводу срока безопасности) должно быть воспринято с осторожностью." Это хороший совет.

Умный криптограф сверхконсервативен при выборе длин открытых ключей. Чтобы понять, насколько длинный ключ вам нужен, вам придется оценить нужную безопасность и время жизни ключа, не забывая о текущем состоянии искусства разлагать на множители. Чтобы получить тот же уровень безопасности, который давало 512-битовое число в начале восьмидесятых, сегодня вам понадобится 1024-битовое число. Если же вы хотите, чтобы ваши ключи оставались безопасными в ближайшие 20 лет, 1024-битовое число, по видимому, слишком коротко.

Даже если ваши конкретные секреты не стоят усилий, нужных для разложения вашего модуля, вы можете оказаться в опасности. Представьте себе автоматическую банковскую систему, использующую для безопасности RSA. Мэллори может предстать перед судом и заявить: "Читали ли вы в газете за 1994 год, что RSA-129 был взломан, и что 512-битовые числа могут быть разложены на множители любой организацией, которая может потратить несколько миллионов долларов и подождать несколько месяцев ? Мой банк использует для безопасности 512-битовые числа и, между прочим, эти семь изъятий сделаны не мной." Даже если Мэллори лжет, судья, вероятно, может потребовать, чтобы банк это доказал.

Почему не использовать 10000-битовые ключи? Конечно, можно, но чем длиннее ваши ключи, тем больше стоимость вычислений. Вам нужен ключ, который был бы достаточно длинным для обеспечения безопасности, но достаточно коротким, чтобы его можно было использовать.

Ранее в этом разделе я называл предсказания глупостью. Теперь я сам попытаюсь предсказать кое-что. В 1-й приведены мои рекомендации по выбору длин открытых ключей в зависимости от того, какой срок безопасности ключа вам нужен. Для каждого года приведены три длины ключа, одна для частного лица, одна для крупной корпорации и одна для правительства большого государства.

Вот некоторые соображения из [66]:

> Мы считаем, что сможем получить доступ к 100 тысячам компьютеров без сверхчеловеческих усилий и неэтичных действий. То есть, мы *не собираемся* выпускать в Internet "червя" или разрабатывать вирус, который бы предоставил бы нам вычислительные ресурсы. Во многих организациях многие тысячи машин подключены к сети. Доступ к их возможностям потребует искусной дипломатии, но не является невозможным. Приняв среднюю производительность машины равной 5 mips и время работы 1 год, вполне возможно осуществить проект, который требует полмиллиона mips-лет.

Проект разложения на множители 129-разрядного числа без значительных усилий смог задействовать 0.03 процента оценочной полной вычислительной мощности Internet [1190]. Разумно предположить, что хорошо разрекламированный проект получит на год 2 процента всемирной вычислительной мощности.

Предположим, что увлеченный криптоаналитик сможет получить в свое распоряжение 10000 mips-лет, большая корпорация - 10^7 mips-лет, а правительство большой страны - 10^9 mips-лет. Предположим также, что вычислительная мощь будет возрастать на порядок каждые пять лет. И , наконец, предположим также, что ус-

пехи в математике разложения на множители позволят нам раскладывать любые числа со скоростью, сравнимой с той, которую обеспечивает решето специального поля чисел. (Это пока невозможно, но прорыв может случиться в любой момент.) 1st рекомендует для различных лет использовать с целью обеспечения безопасности различные длины ключей.

Табл. 7-6.
Рекомендованные длины открытых ключей в (битах)

Год	Частное лицо	Корпорация	Правительство
1995	768	1280	1536
2000	1024	1280	1536
2005	1280	1536	2048
2010	1280	1536	2048
2015	1536	2048	2048

Не забывайте учитывать значимость ключа. Открытые ключи часто используются для длительной обеспечения безопасности важной информации: главный ключ банка для системы электронных наличных, ключ, используемый правительством для подтверждения паспортов, ключ цифровой подписи государственного нотариуса. Возможно, не стоит тратить месяцы машинного времени на вскрытие какого-то личного ключа, но если можете с помощью добытого ключа напечатать собственные деньги, то идея становится весьма захватывающей. Длина 1024-битовогой ключа достаточна для подписи чего-нибудь, что будет проверено в течение недели, месяца, даже нескольких лет. Но вы же не хотите, представ перед судом лет 20 спустя с подписанным электронным образом документом, смотреть, как противоположная сторона показывает, как подделать документы, используя эту же подпись.

Предсказывать более далекое будущее еще глупее. Кто может знать, каких успехов к 2020 году достигнет вычислительная техника, сетевые вычисления и математика? Однако, если окинуть взглядом всю картину, можно заметить, что в каждом следующем десятилетии мы получаем возможность разлагать на множители вдвое более длинные числа, чем в предыдущем. Это позволяет построить 0-й.

С другой стороны, техника разложения на множители может достичь предела своих возможностей задолго до 2045. Хотя я думаю, что это маловероятно.

Не все согласятся с моими рекомендациями. NSA установило для своего Стандарта цифровой подписи (Digital Signature Standard, см. раздел 20.1) длину ключей от 512 до 1024 бит - намного меньше, чем я рекомендую для длительной безопасности. У Pretty Good Privacy ("Вполне надежный секрет", см. раздел 24.12) максимальная длина ключа RSA составляет 2047 бит. Аржан Ленстра, лучший в мире раскладыватель на множители, в течение последних 10 лет отказывается делать предсказания [949]. В -1-й приведены рекомендации Рона Ривеста для длины ключей, которые сделаны в 1990 году и кажутся мне слишком оптимистичными [1323]. Хотя его анализ на бумаге выглядит хорошо, в недавней истории можно найти примеры регулярно происходящих сюрпризов. Чтобы предохранить себя от последствия этих сюрпризов, есть смысл выбирать ключи с запасом.

Табл. 7-7.
Долгосрочный прогноз разложения на множители

Год	Длина ключа (в битах)
1995	1024
2005	2048
2015	4096
2025	8192
2035	16384
2045	32768

Минимальные оценки предполагают бюджет $25000, алгоритм "квадратичное решето" и скорость технического прогресса 20 процентов в год. Средние оценки предполагают бюджет 25 миллионов долларов, алгоритм "решето общего поля чисел" и скорость технического прогресса 33 процента в год. Максимальные оценки предполагают бюджет 25 миллиардов долларов, алгоритм "решето общего поля чисел", работающий со скоростью

решета специального поля чисел и скорость технического прогресса 45 процентов в год .

Всегда есть вероятность того, что успехи в разложении на множители будут поразительны и для меня, но я попытался учесть этот множитель в своих прогнозах . Но почему мне нужно верить? Я лишь продемонстрировал собственную глупость, занимаясь предсказаниями .

Табл. 7-8.
Оптимистичные рекомендации Ривеста для длины ключей (в битах)

Год	Минимальная	Средняя	Максимальная
1990	398	515	1289
1995	405	542	1399
2000	422	572	1512
2005	439	602	1628
2010	455	631	1754
2015	472	661	1884
2020	489	677	2017

Вычисление с помощью ДНК

Это похоже на волшебство. В 1994 году Леонард Эдлман (Leonard M. Adleman) продемонстрировал метод решения задачи **NP-полноты** (см. раздел 11.2) в биохимической лаборатории, используя молекулы ДНК для представления возможных решений задачи [17]. Задача, решенная Эдлманом, была частным случаем задачи направленного гамильтонова пути: дана карта городов, соединенных однонаправленными дорогами, нужно найти путь из города A в город Z, который проходит через каждый город на карте только один раз . Каждый город был представлен своей случайной цепочкой ДНК с 20 основаниями. С помощью обычных методов молекулярной биологии Эдлман синтезировал 50 пикомолей (30 миллионов миллионов молекул) цепочки ДНК, представляющей каждый город. Каждая дорога была представлена цепочкой ДНК с 20 основаниями, но эти цепочки выбирались не случайным образом: они умело выбирались так, чтобы "начало" цепочки ДНК, представляющей дорогу от города P к городу K ("Дорога PK") стремилась бы соединиться со цепочкой ДНК, представляющей город P, а конец Дороги PK стремился бы соединиться с городом K.

Эдлман синтезировал 50 пикомолей ДНК, представляющих каждую дорогу, смешал их вместе с ДНК гор одами, представляющей города, и добавил фермент лигазу, которая связывает вместе концы молекул ДНК. Правильно подобранная связь между цепочками ДНК для дорог и цепочками ДНК для городов приводит к тому, что лигаза соединяет цепочки ДНК для дорог вместе правильным образом . То есть, "Выход" дороги PK всегда будет соединен со "входом" какой-либо дороги, начинающейся в городе K, но никогда с "выходом" любой дороги или со "входом" дороги, которая начинается не в городе K. После тщательно ограниченного времени реакции лигаза создала большое количество цепочек ДНК, представляющих возможные, но все равно случайные объединения путей карты.

В этом супе из случайных путей Эдлман может найти малейший след - может быть единственной молекулы - ДНК, которая является ответом задачи. Используя обычные методы молекулярной биологии, он удалил все цепочки ДНК, представлявшие слишком короткие или слишком длинные пути . (Число дорог в нужном пути должно равняться числу городов минус один .) Затем он удалил все цепочки ДНК, которые не проходили через город A, затем те, которые шли мимо города B, и так далее. Молекула ДНК, прошедшая через это сито, и представляет собой нужную последовательность дорог, являясь решением задачи направленного гамильтонова пути .

По определению частный случай задачи **NP-полноты** может быть преобразован за полиномиальное время к частному случаю любой другой задачи **NP-полноты**, и, следовательно, к задаче о направленном гамильтоновом пути. С семидесятых годов криптологи пытались использовать задачи **NP-полноты** для криптографии с открытыми ключами.

Хотя частный случай, решенный Эдлманом, весьма прост (семь городов на карте, простым наблюдением задача моежт быть решена за несколько минут), это направление только начало развиваться, и не существует никаких препятствий для расширения данной методики на более сложные задачи . Таким образом, рассуждения о безопасности криптографических протоколов, основанных на задачах **NP-полноты**, рассуждения, которые до сих пор начинались словами, "Предположим, что у врага есть миллион процессоров, каждый из которых выполняет миллион проверок каждую секунду", скоро, может быть, будут начинаться словами, "Предположим, у врага есть тысяча ферментных ванн, емкостью по 20000 литров каждая ".

Квантовые вычисления

А теперь еще большая фантастика. В основе квантовых вычислений используется двойственная природа м атерии (и волна, и частица). Фотон может одновременно находиться в большом количестве состояний. Классич еским примером является то, что фотон ведет себя как волна, встречая частично прозрачное зеркало. Он одн овременно и отражается и проходит через зеркало подобно тому, как морская волна, ударяясь о волнолом с н ебольшим отверстием в нем, одновременно отразится от стены и пройдет сквозь нее . Однако, при измерении фотон ведет себя подобно частице, и только одно состояние может быть обнаружено .

В [1443] Питер Шор (Peter Shor) очертил принципы построения машины для разложения на множители, о сновanной на законах квантовой механики. В отличие от обычного компьютера, который можно представить как машину, имеющее в каждый момент времени единственное фиксированное состояние, квантовый компьютер обладает внутренней волновой функцией, являющейся суперпозицией комбинаций возможных основных с остояний. Вычисления преобразуют волновую функцию, меняя весь набор состояний единым действием . Таким образом, квантовый компьютер имеет преимущество над классическим конечным автоматом : он использует квантовые свойства для числа разложения на множители за полиномиальное время, теоретически позволяя взломать криптосистемы, основанные на разложении на множители или задаче дискретного логарифма .

Общепризнанно, что квантовый компьютер не противоречит фундаментальным законам квантовой механ ики. Однако, непохоже, что квантовая машина для разложения на множители будет построена в обозримом б удущем ... если вообще будет построена. Одним из главных препятствий является проблема некогерентности , которая является причиной потери отчетливости волновыми огибающими и приводит к сбою компьютера . Из-за некогерентности квантовый компьютер, работающий при 1К, будет сбиваться каждую наносекунду . Кроме того, для построения квантового устройства для разложения на множители потребуется огромное количество вент илей, а это может не дать построить машину. Для проекта Шора нужно совершенное устройство для возведения в степень. Внутренние часы не используются, поэтому для разложения на множители криптографически знач имых чисел могут потребоваться миллионы или, возможно, миллиарды индивидуальных вентилей . Если мини мальная вероятность отказа каждого из n квантовых вентилей равна p, то среднее количество испытаний, нео бходимое для достижения успеха, составит $(1/(1-p))^n$. Число нужных вентилей, по видимому, растет полином иально с ростом длины числа (в битах), поэтому число требуемых попыток будет расти с увеличением длины используемых чисел сверхэкспоненциально - хуже чем при разложении делением !

Поэтому, хотя квантовое разложение на множители вызывает восхищение в академических кругах, малов ероятно, что оно будет иметь практическое значение в обозримом будущем . Но не говорите потом, что я вас не предупреждал.

7.3 Сравнение длин симметричных и открытых ключей

Система взламывается обычно в ее слабейшем месте. Если вы проектируете систему, которая использует и симметричную криптографию, и криптографию с открытыми ключами, то длины ключей для криптографии каждого типа должны выбираться так, чтобы вскрыть любой из компонентов системы было одинаково трудно . Бессмысленно использовать симметричный алгоритм со 128-битовым ключом вместе с алгоритмом с открыт ыми ключами, использующим 386-битовый ключ. Точно так же бессмысленно использовать в одной системе симметричный алгоритм с 56-битовым ключом и алгоритм с открытыми ключами, применяющий 1024-битовый ключ.

В -2-й перечислены длины модулей открытых ключей, трудность разложения которых на множители сра внима со сложностью вскрытием грубой силой сопоставленных длин популярных симметричных ключей .

Табл. 7-9.
Длины симметричных и открытых ключей с аналогичной устойчивостью к вскрытию грубой силой

Длина симметричного ключа (в битах)	Длина открытого ключа (в битах)
56	384
64	512
80	768
112	1792
128	2304

Из этой таблица можно сделать вывод, что если вы достаточно беспокоитесь о своей безопасности, чтобы выбрать симметричный алгоритм со 112-битовым ключом, вам следует выбрать длину модуля в вашем алгоритме с открытыми ключами порядка 1792 бит. Однако, в общем случае следует выбирать длину открытого ключа более безопасную, чем длина вашего симметричного ключа. Открытые ключи обычно используются дольше и применяются для защиты большего количества информации.

7.4 Вскрытие в день рождения против однонаправленных хэш-функций

Существует два способа вскрытия однонаправленных хэш-функций грубой силой. Первый наиболее очевиден: дано значение хэш-функции сообщения, *H(M)*, врагу хотелось бы суметь создать другой документ, *M'*, такой, что *H(M')=H(M)*. Второй способ более тонок: врагу хотелось бы найти два случайных сообщения, *M* и *M'*, таких, что *H(M')=H(M)*. Такой способ называется **столкновением** и является более простым, чем первый, способом вскрытия.

Парадокс дня рождения является стандартной статистической проблемой. Сколько человек должно собраться в одной комнате, чтобы с вероятностью 1/2 хотя бы у кого-нибудь из них был бы общий с вами день рождения? Ответ - 183. Хорошо, а сколько людей должно собраться, чтобы с вероятностью 1/2 хотя бы у двоих из них был бы общий день рождения? Ответ удивителен - 23. 23 человека, находящихся в комнате, образуют 253 различных *пары*.

Найти кого-нибудь с тем же днем рождения - аналогия с первым способом вскрытия, найти двух человек с произвольным одинаковым днем рождения - аналогия со вторым способом. Второй способ широко известен как **вскрытие в день рождения**.

Предположим, что однонаправленная хэш-функция безопасна, и лучшим способом ее вскрытия является вскрытие грубой силой. Результатом функции является *m*-битовое число. Поиск сообщения, хэш-значение которого совпадает с заданным, в среднем потребовал бы хэширования 2^m случайных сообщений. А для обнаружения двух сообщений с одинаковым хэш-значением потребуется только $2^{m/2}$ случайных сообщений. Компьютеру, который хэширует миллион сообщений в секунду, потребовалось бы 600000 лет, чтобы найти второе сообщение с тем же 64-битовым хэш-значением. Тот же компьютер сможет найти пару сообщений с общим хэш-значением примерно за час

Это значит, что, если вы опасаетесь вскрытия в день рождения, вы должны выбирать длину хэш-значения в два раза длиннее, чем вам потребовалось бы в противном случае. Например, если вы хотите уменьшить вероятность взлома вашей системы до 1 шанса из 2^{80}, используйте 160-битовую однонаправленную хэш-функцию.

7.5 Каков должны быть длина ключа?

На этот вопрос нет единого ответа, ответ этот зависит от ситуации. Чтобы понять, какая степень безопасности вам нужна, вы должны задать себе несколько вопросов. Сколько стоит ваша информация? Как долго она должна безопасно храниться? Каковы ресурсы ваших врагов?

Список клиентов может стоить $1000. Финансовая информация при неожиданном разводе могла бы стоить $10000. Реклама и данные маркетинга для большой корпорации могли бы стоить 1 миллион долларов. Главный ключ для системы электронных наличных может стоить миллиарды.

В мире торговли предметами потребления секреты должны только сохраняться в течение нескольких минут. В газетном бизнесе сегодняшние секреты - это завтрашние заголовки. Информация о разработке какого-то продукта, возможно, должна будет храниться в секрете в течение года или двух Изделия(программы) могла бы была бы должна остаться секретом в течение года или два. Данные переписи США в соответствии с законом должны храниться в секрете в течение 100 лет.

Список гостей, приглашенных на вечер-сюрприз в честь дня рождения вашей сестры, интересен только в ашим любопытным родственникам. Торговые секреты корпорации представляют интерес для конкурирующих компаний. Военные секреты интересны вражеским военным.

В этих терминах даже можно определить требования к безопасности Можно. Например:

> Длина ключа должна быть такой, чтобы взломщик, готовый потратить 100 миллионов долларов, мог взломать систему в течение года с вероятностью не более, чем $1/2^{32}$, даже с учетом скорости технического прогресса 30 процентов в год.

В -3-й, частично взятой из [150], приведены оценки требований к безопасности для различной информ ации:

Будущую вычислительную мощь оценить нелегко, но вот разумное эмпирическое правило: эффективность вычислительных средств удваивается каждые 18 месяцев и возрастает на порядок каждые 5 лет. Следовательно, через 50 лет самые быстрые компьютеры будут в 10 миллиардов быстрее, чем сегодня! Кроме того, не забывайте, что эти числа относятся только к универсальным компьютерам, кто знает, какие специализированные ус тройства для вскрытия криптосистем будут разработаны в следующие 50 лет?

Предполагая, что криптографический алгоритм будет использоваться в ближайшие 30 лет, вы можете пре д-ставить себе, насколько он должен быть безопасен. Алгоритм, созданный сегодня, возможно не станет широко использоваться до 2000 года, и все еще будет использоваться в 2025 для шифрования сообщений, которые должны оставаться в секрете до 2075 года и позже.

Табл. 7-10.
Требования к безопасности различной информации

Типы трафика	Время жизни	Минимальная длина ключа (в битах)
Тактическая военная информация	минуты/часы	56-64
Объявления о продуктах, слиянии компаний, процен т-ных ставках	дни/недели	64
Долговременны бизнес-планы	годы	64
Торговые секреты (например, рецепт кока-колы)	десятилетия	112
Секреты водородной бомбы	>40 лет	128
Личности шпионов	>50 лет	128
Личные дела	>50 лет	128
Дипломатические конфликты	>65 лет	128
Данные переписи США	100 лет	по меньшей мере 128

7.6 Caveat emptor[1]

Вся эта глава - просто много чепухи. This entire chapter is a whole lot of nonsense. Смешно говорить даже о самом понятии предсказания вычислительной мощи на 10, а тем более на 50 лет вперед. Эти расчеты приведены только для ориентировки, ни для чего больше. Экстраполируя прошлое, мы получаем будущее, которое, возможно, будет иметь мало общего с грядущей реальностью.

Будьте консерваторами. Если ваши ключи длиннее, чем вам кажется необходимым, то меньшее количество технологических сюрпризов сможет повредить вам.

[1] Да будет осмотрителен покупатель (латин.)

Глава 8
Управление ключами

У Алисы и Боба есть безопасная система связи. Они играют в мысленный покер, одновременно подписыв аю т контракты и даже меняют цифровые наличные. Их протоколы безопасны. Их алгоритмы -самые лучшие. К несчастью, они покупают свои ключи от "Keys-R-Us" Евы, чей лозунг - "Вы можете доверять нам: Безопасность - среднее имя человека, которого туристический агент нашей бывшей тещи встретил в "Kwik-E-Mart".

Ева не нужно вскрывать алгоритмы. Ей не нужно полагаться на тонкие дефекты протоколов. Она может и с пользовать их ключи для чтения всех сообщений Алисы и Боба, не прикладывая никаких криптоаналитических усилий.

В реальном мире управление ключами представляет собой самую трудную часть криптографии. Проектир о вать безопасные криптографические алгоритмы и протоколы не просто, но Вы можете положиться на большой объем академических исследований. Сохранить секрет ключей намного труднее.

Криптоаналитики часто вскрывают и симметричные криптосистемы, и криптосистемы с открытыми ключ а ми через распределение ключей. Зачем Еве беспокоиться об проблеме вскрытия криптографического алгоритма целиком, если она может восстановить ключ из-за неаккуратного хранения ключа? Зачем ей тратить 10 ми л лионов долларов на создание машина для криптоанализа, если она может подкупить клерка за 1000 долларов? Миллион долларов за клерка связи на хорошем месте в дипломатическом посольстве может быть выгодной сделкой. Уолкеры годами продавали Советам ключи шифрования ВМС США. Руководитель контрразведки ЦРУ стоил меньше 2 миллионов долларов, включая жену. Это намного дешевле, чем строить огромные маш и ны вскрытия и нанимать блестящих криптоаналитиков. Ева может выкрасть ключи. Она может арестовать или похищать кого-то, кто знает ключи. Она может совращать кого-то и получать ключи таким образом. (Морские пехотинцы, охранявшие посольство США в Москве не устояли перед подобной атакой.) Намного проще нах о дить дефекты в людях, чем в криптосист емах.

Алиса и Боб должны защищать и свой ключ, и в той степени шифруемые им данные. Если ключ не изменять регулярно, то количество данных может быть огромно. К сожалению, многие коммерческие продукты просто объявляют "Мы используем DES" и забывают обо всем остальном. Результаты не слишком впечатляют.

Например, программа DiskLock для Macintosh (версия 2.l), продававшаяся в большинстве магазинов пр о граммного обеспечения, претендует на безопасное шифрование DES. Она шифрует файлы, используя DES. Ре а лизация DES алгоритма правильна. Однако, DiskLock сохраняет ключ DES вместе с зашифрованным файлом. Если вы знаете, где искать ключ, и хотеть прочитать файл, шифрованный DiskLock с помощью DES, восстан о вите ключ из шифрованного файла и затем расшифровывать файл. Не имеет значение, что программа использ у ет шифрование DES - реализация абсолютно небезопасна.

Дальнейшую информацию относительно управления ключами можно найти в [457, 98, 1273, 1225, 775, 357]. В следующих разделах обсуждаются некоторые из вопросов и решений.

8.1 Генерация ключей

The security of an algorithm rests in the key. If you're using a cryptographically weak process to generate keys, then your whole system is weak. Eve need not cryptanalyze your encryption algorithm; she can cryptanalyze your key generation algorithm.

Безопасность алгоритма сосредоточена в ключе. Если вы используете криптографически слабый процесс для генерации ключей, то ваша система в целом слаба. Еве не нужно криптоанализировать ваш алгоритм шифров а ния, она может криптоанализировать ваш алгоритм генерации ключей.

Уменьшенные пространства ключей

DES использует 56-битовый ключ с битами. Любая правильно заданная 56-битовая строка может быть кл ю чом, существует 2^{56} (10^{16}) возможных ключей. Norton Discreet for MS-DOS (версии 8.0 и более ранние) разр е шает пользоваться только ключам ASCII, делая старший бит каждого байта нолем. Программа также преобр а зует символы нижнего регистра в верхний регистр (так что пятый бит каждого байта всегда противоположен шестому бита) и игнорирует бит младшего разряда каждого байта, что приводит к пространству в 2^{40} возможных ключей. Эти ущербные процедуры генерации ключей сделали свою реализацию DES в десять тысяч раз проще для вскрытия.

7-й содержит число возможных ключей для различных ограничений на входные строки. В 6-й приведено время, требуемое для исчерпывающего перебора всех возможных ключей при миллионе попыток в секу нду.

Могут быть использованы для вскрытия грубой силой любые специализированные аппаратные и параллел ь-

ные реализации. При проверке миллиона ключей в секунду (одной машиной или несколькими параллельно) физически возможно расколоть ключи из символов нижнего регистра и ключи из цифр и символов нижнего регистра длиной до 8 байтов, алфавитно-цифровые ключи - длиной до 7 байтов, ключи из печатаемых символов и ASCII-символов - длиной до 6 байтов, в ключи из 8-битовых ASCII-символов - длиной до 5 байтов.

Табл. 8-1.
Количество возможных ключей в различных пространствах ключей

	4 байта	5 байтов	6 байтов	7 байтов	8 байтов
Строчные буквы (26)	460000	$1.2*10^{7}$	$3.1*10^{8}$	$8.0*10^{9}$	$2.1*10^{11}$
Строчные буквы и цифры (36)	1700000	$6.0*10^{7}$	$2.2*10^{9}$	$7.8*10^{10}$	$2.8*10^{12}$
Алфавитные и цифровые символы (62)	$1.5*10^{7}$	$9.2*10^{8}$	$5.7*10^{10}$	$3.5*10^{12}$	$2.2*10^{14}$
Печатаемые символы (95)	$8.1*10^{7}$	$7.7*10^{9}$	$7.4*10^{11}$	$7.0*10^{13}$	$6.6*10^{15}$
Символы ASCII (128)	$2.7*10^{8}$	$3.4*10^{10}$	$4.4*10^{12}$	$5.6*10^{14}$	$7.2*10^{16}$
8-битовые ASCII символы (256)	$4.3*10^{9}$	$1.1*10^{12}$	$2.8*10^{14}$	$7.2*10^{16}$	$1.8*10^{19}$

Табл. 8-2.
Время исчерпывающего поиска различных пространства ключей (при одном миллионе проверок в секунду)

	4 байта	5 байтов	6 байтов	7 байтов	8 байтов
Строчные буквы (26)	0.5 секунды	12 секунд	5 минут	2.2 часа	2.4 дня
Строчные буквы и цифры (36)	1.7 секунды	1 минута	36 минут	22 часа	33 дня
Алфавитные и цифровые символы (62)	15 секунд	15 минут	16 часов	41 день	6.9 года
Печатаемые символы (95)	1.4 минуты	2.1 часа	8.5 дня	2.2 года	210 лет
Символы ASCII (128)	4.5 минуты	9.5 часа	51 день	18 лет	2300 лет
8-битовые ASCII символы (256)	1.2 часа	13 дней	8.9 года	2300 лет	580000 лет

И помните, вычислительная мощь удваивается каждые 18 месяцев. Если вы хотите, чтобы ваши ключи были устойчивы к вскрытию грубой силой в течение 10 лет, вы должны соответствующим образом планировать и с-пользование ключей.

Обедненный выбор ключей

Когда люди сами выбирают ключи, они выбирают ущербные ключи. Они с большей вероятностью выберут "Barney", чем "*9 (hH/A". Это не всегда происходит из-за плохой практики, просто "Barney" легче запомнить чем "*9 (hH/A". Самый безопасный алгоритм в мире не сильно поможет, если пользователи по привычке выб-ирают имена своих жен (мужей) для ключей или пишут свои ключи на небольших листочках в бумажниках. И н-теллектуальное вскрытие грубой силой не перебирает все возможные ключи в числовом порядке, но пробует сначала очевидные ключи.

Это называется **вскрытием со словарем**, потому что нападющий использует словарь общих ключей. Дэн-ел Кляйн (Daniel Klein) смог расколоть 40 процентов паролей на среднем компьютере, используя этот способ вскрытия [847, 848]. Нет, он не перебирал один пароль за другим, пытаясь войти в систему. Он скопировал з-ашифрованный файл паролей и предпринял вскрытие автономно. Вот, что он пробовал:

1. В качестве возможного пароля имя пользователя, инициалы, имя бюджета и другую связанную с ч-еловеком информацию. В целом, на основе такой информации пробовалось до 130 различных паролей. Вот некоторые из паролей, проверявшихся для имени бюджета **klone** и пользователя "Daniel V. Klein": klone, klone0, klonel, klonel23, dvk, dvkdvk, dklein, Dklein, leinad, nielk, dvklein, danielk, DvkkvD, DANIEL-KLEIN, (klone), KleinD, и так далее.
2. Слова из различных баз данных. Использовались списки мужских и женских имен (всего около 16000), названия мест (включая изменения, поэтому рассматривались и "spain", " Spanish", и "Spaniard"), имена известных людей, мультфильмы и мультипликационные герои, заголовки, герои и места из фильмов и научной фантастики, мифические существа (добытые из *Bullfinch's Mythology* и

словарей мифических животных), спорт (включая названия команд, прозвища и специальные терм ины), числа (записанные как цифрами - '2001", так и буквами " twelve"), строки символов и чисел ("a", "aa", "aaa", "aaaa" и т.д.), китайские слоги (из Piny in Romanization of Chinese, международного стан дарта письма по китайски на англоязычной клавиатуре), Библия короля Джеймса; биологические те рмины, разговорные и вульгарные выражения (типа "fuckyou", "ibmsux" и "deadhead"), стандарты кл авиатуры (типа "qwerty", "asdf" и "zxcvbn"), сокращения (типа "roygbiv" - первые буквы названий цв етов радуги по английски - и "ooottafagvah" - мнемоническая схема для запоминания 12 черепных не рвов), имена компьютеров (полученные из /etc/hosts), герои, пьесы и места действия у Шекспира, с амые распространенные слова языка Идиш, названия астероидов, совокупность слов из различных те хнических статей, опубликованных ранее Кляйном. Итого, для пользователя рассматривалось более чем 60000 отдельных слов (с отбрасыванием дубликатов в различных словарях).

3. Вариации слов из пункта 2. Это включало перевод первого символа в верхний регистр или его замену управляющим символом, перевод всего слова в верхний регистр, инверсию регистра слова (с и без вышеупомянутого изменения регистра первой буквы), замену буквы "o" на цифру "0" (так, чтобы сл ово "scholar" было также проверено как "sch0lar"), замену буквы "l" на цифру "1" (так, чтобы слово "scholar" было бы также проверено как "scho1ar") и выполнение аналогичных манипуляций с буквой "z" и цифрой "2", а также с буквой "s" и цифрой "5". Другая проверка состояла из перевода слова во множественное число (независимо от того, было ли слово существительным) с учетом необходимых правил, чтобы "dress" заменилось на "dresses", "house" - на "houses", а "daisy" - на "daisies". Хотя Кляйн не жестко придерживался правил преобразования ко множественному числу, поэтому "datum" стала "datums" (а не "data"), "sphynx" - "sphynxs" (а не "sphynges"). Аналогично, для преобразования слов добавлялись суфиксы "-ed", "-er" и "-ing", подобно "phase" в "phased," "phaser" и "phasing". Эти дополнительные проверки добавили еще 1000000 слов к списку возможных паролей, которые пров ерялись для каждого пользователя.

4. Различные варианты преобразования к верхнему регистру слов пункта 2, не рассматривавшихся в пункте 3. Сюда вошло преобразование к верхнему регистру одиночных символов (так, чтобы "michael" был также проверен как "m Ichael", "miChael", "micHael", "michAel", и т.д.), преобразование к верхнему регистру пары символов ("MIchael", "MiChael", "MicHael", ..., "mIChael", "mIcHael", и т.д.), преобразование к верхнему регистру трех символов, и т.д. Изменения одиночного символа доб авили к проверяемым примерно еще 400000 слов, а изменения пары символов - 1500000 слов. Измен ения трех символов добавляли по крайней мере еще 3000000 слов для каждого пользователя, если для завершения тестирования хватало времени. Проверка изменения четырех, пяти и шести символов была признана непрактичной, так как для их проверки не хватало вычислительных мощностей.

5. 5. Иностранные слова для иностранных пользователей. Специфический тест, который был выполнен, проверял пароли из китайского языка для пользователей с китайскими именами. Для китайских сл огов использовался стандарт Pinyin Romanization, слоги объединялись вместе в одно-, двух- и тре хсложные слова. Так как не было выполнено предварительной проверки слов на значимость, использ овался исчерпывающий перебор. Так как в системе Pinyin существует 298 китайских слогов, то имеется 158404 слов с двумя слогами, и немного больше 16000000 слов с тремя слогами. Подобный способ вскрытия мог бы быть легко использован и для английского языка, с учетом правил образования пр оизносимых ничего не значащих слов.

6. Пары слов. Объем такого исчерпывающего теста колеблется. Чтобы упростить тест, из */usr/dict/words* использовались только слова длиной три или четыре символа. Даже при этом, число пар слов сост авило приблизительно десять миллионов.

Вскрытие со словарем намного мощнее, когда оно используется против файла ключей, а не против одного ключа. Одиночный пользователь может быть достаточно разумен и выбрать хорошие ключи. Если из тысячи людей каждый выбирает собственный ключ как пароль компьютерной системы, то велика вероятность того, что по крайней мере один человек выберет ключ, имеющийся в словаре взломщика.

Случайные ключи

Хорошими ключами являются строки случайных битов, созданные некоторым автоматическим процессом. Если длина ключа составляет 64 бита, то все возможные 64-битовые ключи должны быть равновероятны. Ген ерируйте биты ключей, пользуясь либо надежным источником случайных чисел (см. раздел 17.14), либо крипт ографически безопасным генератором псевдослучайных битов (см. главы 16 и 17.) Если такие автоматические процессы недоступны, бросайте монетку или кости.

Это важно, но не увлекайтесь обсуждением того, является ли шум из звуковых источников более случайным, чем шум из радиоактивного распада. Ни один из этих источников случайного шума не совершенен, но все они, скорее всего, будут достаточно хороши. Для генерации ключей важно использовать хороший генератор случа йных чисел, но гораздо важнее использовать хорошие алгоритмы шифрования и процедуры управления ключ а-

ми. Если вы беспокоитесь о случайности ваших ключей, используйте описанную ниже методику перемалывания ключа.

Некоторые алгоритмы шифрования имеют слабые ключи - специфические ключи, менее безопасные чем другие ключи. Я советую проверять слабость ключа ключей и, обнаружив ее, генерировать новый. У DES только 16 слабых ключей в пространстве 2^{56}, так что вероятность получить один из этих ключей невероятно мала. Заявлялось, что криптоаналитик не будет знать о том, что используется слабый ключ, и, следовательно, не сможет получить никакой выгоды из их случайного использования. Также заявлялось, что информацию криптоаналитику дает совсем не использование слабых ключей. Однако, проверка немногих слабых ключей настолько проста, что кажется глупым пренебречь ею.

Генерация ключей для систем криптографии с открытыми ключами тяжелее, потому что часто ключи должны обладать определенными математическими свойствами (возможно, они должны быть простыми числами, квадратичным остатком, и т.д.). Методы генерации больших случайных простых чисел рассматриваются в разделе 11.5. Важно помнить, что с точки зрения управления ключами случайные стартовые последовательности для таких генераторов должны быть действительно случайны.

Генерация случайного ключа возможна не всегда. Иногда вам нужно помнить ваш ключ. (Интересно, сколько времени вам понадобится, чтобы запомнить 25e8 56f2 e8ba c820?). Если вам надо генерировать простой для запоминания ключ, замаскируйте его. Идеалом является то, что легко запомнить, но трудно угадать. Вот несколько предложений:

— Пары слов, разделенные символом пунктуации, например, " turtle*moose" или "zorch!splat"

— Строки букв, являющиеся акронимами длинных фраз, например, "Mein Luftkissenfahrzeug ist voller Aale!" служит для запоминания ключа "MLivA!"

Ключевые фразы

Лучшим решением является использование вместо слова целой фразы и преобразование этой фразы в ключ. Такие фразы называются **ключевыми фразами**. Методика с названием **перемалывание ключа** преобразует легко запоминающиеся фразы в случайные ключи. Для преобразования текстовой строки произвольной длины в строку псевдослучайных бит используте однонаправленную хэш-функцию. Например, легко запоминающаяся текстовая строка:

```
My name is Ozymandias, king of kings. Look on my works, ye mighty, and despair. [1]
```

может "перемолоться" в такой 64-битовый ключ:

```
e6c1 4398 5ae9 0a9b
```

Конечно, может быть нелегко ввести в компьютер целую фразу, если вводимые символы не отображаются на экране. Разумные предложения по решению этой проблемы будут оценены.

Если фраза достаточно длинна, то полученный ключ будет случаен. Вопрос о точном смысле выражения "достаточно длинна" остается открытым. Теория информации утверждает, что информационная значимость стандартного английского языка составляет около 1.3 бита на символ (см. раздел 11.1). Для 64-битового ключа достаточной будет ключевая фраза, состоящая примерно из 49 символов, или 10 обычных английских слов. В качестве эмпирического правила используйте пять слов для каждых 4 байтов ключа. Это предложение работает с запасом, ведь в нем не учитываются регистр, пробелы и знаки пунктуации.

Этот метод также можно использовать для генерации закрытых ключей в криптографических системах с открытыми ключами: текстовая строка преобразуется в случайную стартовую последовательность, а эта последовательность может быть использована в детерминированной системе, генерирующей пары открытый ключ/закрытый ключ.

Выбирая ключевую фразу, используйте что-нибудь уникальное и легко запоминающееся. Не выбирайте фразы из книг - пример с "Ozymandias" в этом смысле плох. Легко доступны и могут быть использованы для вскрытия со словарем и собрание сочинений Шекспира, и диалоги из *Звездных войн*. Выберите что-нибудь туманное и личное. Не забудьте о пунктуации и преобразовании регистра, если возможно включите числа и неалфавитные символы. Плохой или искаженный английский, или даже любой иностранный язык, делает ключевую фразу более устойчивой к вскрытию со словарем. Одним из предложений является использование фразы, которая является "потрясающей ерундой", чем-то таким, что вы вряд ли запомните и вряд ли запишете.

Несмотря на все написанное здесь маскировка не заменяет истинную случайность. Лучшими являются случайные ключи, которые так тяжело запомнить.

---

[1] Я Озимандиас, царь царей. Вы, сильные мира сего, смотрите на мои труды и трепещите.

Стандарт генерации ключей X9.17

Стандарт ANSI X9.17 определяет способ Генерации ключей (см. 7th) [55]. Он не создает легко запоминающиеся ключи, и больше подходит для генерации сеансовых ключей или псевдослучайных чисел в системе . Для генерации ключей используется криптографический алгоритм DES, но он может быть легко заменен любым другим алгоритмом.

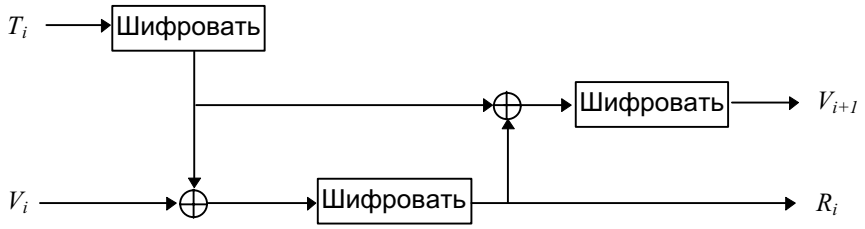


Рис. 8-1. Генерация ключей ANSI X9.17

Пусть $E_K(X)$ - это X, зашифрованный DES ключом K, специальным ключом, предусмотренным для генерации секретных ключей. V_0 - это секретная 64-битовая стартовая последовательность. T - это метка времени. Для генерации случайного ключа R_i вычислим:

$R_i = E_K(E_K(T_i) \oplus V_i)$

Для генерации V_{i+1}, вычислим:

$V_{i+1} = E_K(E_K(T_i) \oplus R_i)$

Для превращения R_i в ключ DES, просто удалите каждый восьмой бит. Если вам нужен 64-битовый ключ, используйте ключ без изменения. Если вам нужен 128-битовый ключ, создайте пару ключей и объедините их.

Генерация ключей в министерстве обороны США

Министерство обороны США для генерации случайных ключей рекомендует использовать DES в режиме OFB (см. раздел 9.8) [1144]. Создавайте ключи DES, используя системные вектора прерывания, регистры состояния системы и системные счетчики. Создавайте вектор инициализации, используя системные часы, идентификатор системы, с также дату и время. Для открытого текста используйте 64-битовые величины, созданные кем-то другим, например, 8 символов, введенных системным администратором . Используйте в качестве своего ключа результат.

8.2 Нелинейные пространства ключей

Вообразите, что вы - это военная криптографическая организация, создающая криптографический модуль для ваших войск. Вы хотите использовать безопасный алгоритм, но что будет, если аппаратура попадет во вражеские руки? Ведь вы не хотите, чтобы ваши приборы использовались для защиты *вражеских* секретов.

Если вы можете поместить ваш алгоритм в защищенный модуль, то вот, что вы можете сделать. Потребуйте, чтобы модуль правильно работал только с ключами специальной и секретной формы, а со всеми другими ключами для шифрования использовался сильно ослабленный алгоритм. Можно сделать так, чтобы вероятность того, что кто-то, не знающий этой специальной формы, случайно наткнется на правильный ключ, была исчезающе малой.

Получившееся пространство ключей называется **нелинейным**, потому что ключи не являются одинаково сильными. (Противоположным является линейное, или плоское, пространство ключей.) Простым способом добиться этого можно, создавая ключ, состоящий из двух частей: непосредственно ключа и некоторой фиксированной строки, шифрованной этим ключом. Модуль расшифровывает строку, используя ключ. Если результатом оказывается фиксированная строка, то ключ используется как обычно, если нет, то используется другой, слабый алгоритм. Если алгоритм имеет 128-битовый ключ и 64-битовый размер блока, то длина полного ключа - 192 бита. Таким образом, у алгоритма 2^{128} эффективных ключа, но вероятность случайно выбрать правильный составляет один шанс из 2^{64}.

Вы можете сделать еще хитрее. Можно разработать такой алгоритм, что некоторые ключи будут сильнее других. У алгоритма не будет слабых ключей - ключей, которые с очевидностью являются недостаточно защищенными - и тем не менее у него будет нелинейное пространство ключей.

Это работает только, если используется секретный алгоритм, который враг не может перепроектировать, или если различие в силе ключей достаточно тонко, чтобы враг не смог о нем догадаться. NSA проделывало это с секретными алгоритмами в своих модулях Overtake (см. раздел 25.1). Делали ли они то же самое с Skipjack (см. раздел 13.12)? Неизвестно.

8.3 Передача ключей

Алиса и Боб собираются для безопасной связи использовать симметричный криптографический алгоритм, им нужен общий ключ. Алиса генерирует ключ, используя генератор случайных ключей. Теперь она должна безопасно передать его Бобу. Если Алиса сможет где-то встретить Боба (какие-нибудь задворки, комната без окон или одна из лун Юпитера), то она сможет передать ему копию ключа. В противном случае, у них есть пр облема. Криптография с открытыми ключами решает проблему легко и с минимумом предварительных соглаш ений, но эти методы не всегда доступны (см. раздел 3.1). Некоторые системы используют альтернативные кан алы, считающиеся безопасными. Алиса могла бы посылать Бобу ключ с доверенным посыльным. Она могла бы послать ключ заказной почтой или ночной службой доставки. Она могла бы устанавливать другой канал связи с Бобом и надеяться, что его то никто не подслушивает.

Алиса могла бы послать Бобу симметричный ключ по их каналу связи - тот, который они собираются ши фровать. Но глупо передавать ключ шифрования канала по этому же каналу в открытом виде, кто-то, подслуш ивающий канал, наверняка сможет расшифр овывать все сообщения.

Стандарт X9.17 [55] определяет два типа ключей: **ключи шифрования ключей** и **ключи данных**. Ключами шифрования ключей при распределении шифруются другие ключи. Ключи данных шифруют сами сообщ ения. Ключи шифрования ключей должны распределяться вручную, (хотя они могут быть в безопасности в з ащищенном от взлома устройстве, таком как кредитная карточка), но достаточно редко. Ключи данных распр еделяются гораздо чаще. Подробности можно найти в [75]. Эта идей двухсвязных ключей часто используется при распределении ключей.

Другим решением проблемы распределения является разбиение ключа на несколько различных частей (см. раздел 3.6) и передача их по различным каналам. Одна часть может быть послана телефоном, другая - почтой, третья - службой ночной доставки, четвертая - почтовым голубем, и так далее, (см. 6-й). Так противник может собрать все части, кроме одной, и все равно ничего не узнает про ключ. Этот метод будет работать во всех сл учаях, кроме крайних. В разделе 3.6 обсуждаются схемы разбиения ключа на несколько частей. Алиса могла бы даже применить схему совместно используемого секрета, (см. раздел 3.7), что даст возможность Бобу восст анавливать ключ, если некоторые из частей потеряны при перед аче.

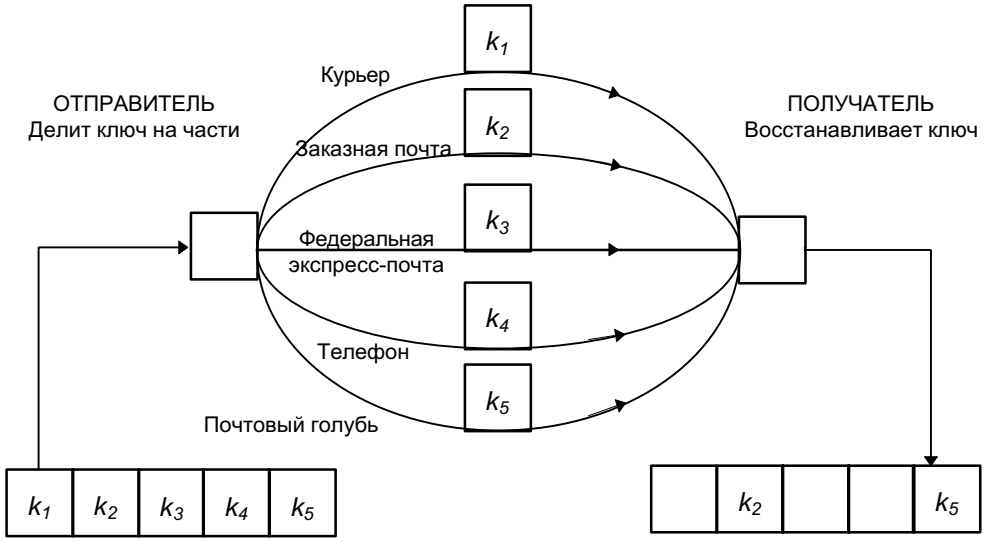


Рис. 8-2. Распределение ключей по параллельным каналам.

Алиса безопасно передает Бобу ключ шифрования ключей или при личной встрече, или с помощью только что рассмотренной методики разбиения. Как только и у Алисы, и у Боба будет ключ шифрования ключей, Ал иса сможет посылать Бобу ключи данных на день по тому же самому каналу связи, шифруя при этом каждый ключ данных ключом шифрования ключей. Так как трафика, шифруемый ключом шифрования ключей незн ачителен, то этот ключ часто менять не нужно. Однако, так как компрометация ключа шифрования ключей м ожет скомпрометировать все сообщения, шифрованное использованными ключами данных, которые были з ашифрован этим ключом шифрования ключей, этот ключ должен храниться в безопасности.

Распределение ключей в больших сетях

Ключи шифрования ключей, общие для пары пользователей, хорошо использовать в небольших сетях, но с увеличением сети такая система быстро становится громоздкой. Так как каждая пара пользователей должна

обменяться ключами, общее число обменов ключами в сети из n человек равно n(n - l)/2.

В сети с шестью пользователями потребуется 15 обменов ключами. В сети из 1000 пользователей понадобится уже около 500000 обменов ключами. В этих случаях работа сети гораздо более эффективна при использовании центрального сервера (или серверов) ключей.

Кроме того, любой из протоколов симметричной криптографии или криптографии с открытыми ключами, приведенных в разделе 3.1, подходит для безопасного распределения ключей.

8.4 Проверка ключей

Как Боб узнает, получив ключ, что ключ передан Алисой, а не кем-то другим, кто выдает себя за Алису? Все просто, если Алиса передает ему ключ при личной встрече. Если Алиса посылает свой ключ через доверенного курьера, то курьеру должен доверять и Боб. Если ключ зашифрован ключом шифрования ключей, то Боб должен доверять тому, что этот ключ шифрования ключей есть только у Алисы. Если для подписи ключа Алиса использует протокол электронной подписи, Боб при проверке подписи должен доверять базе данных открытых ключей,. (Ему также придется считать, что Алиса сохранила свой ключ в безопасности.) Если Центр распределения ключей (Key Distribution Center, KDC) подписывает открытый ключ Алисы, Боб должен считать, что его копия открытого ключа KDC не была подменена.

Наконец, тот, кто управляет всей сетью вокруг Боба, может заставить его думать все, что ему хочется. Мэллори может послать шифрованное и подписанное сообщение, выдавая себя за Алису. Когда Боб, проверяя подпись Алисы, обратится к базе данных открытых ключей, Мэллори может возвратить ему собственный открытый ключ. Мэллори может создать свой собственный поддельный KDC и подменить открытый ключ настоящего KDC ключом своего собственного изделия. Боб никак не сможет это обнаружить.

Некоторые люди использовали этот аргумент, утверждая, что криптография с открытыми ключами бесполезна. Так как единственный способ Алисе и Бобу знать наверняка, что никто не взломал их ключи, - это личная встреча, то криптография с открытыми ключами вообще не обеспечивает безопасность.

Эта точка зрения наивна. Теоретически все правильно, но действительность гораздо сложнее. Криптография с открытыми ключами, используемая вместе с электронными подписями и надежными KDC, сильно усложняет подмену одним ключом другого. Боб никогда не может быть абсолютно уверен, что Мэллори не контролирует его реальность полностью, но Боб может знать наверняка, что такая подмена реальности потребует гораздо больше ресурсов, чем сможет заполучить реальный Мэллори.

Боб мог бы также проверять ключ Алисы по телефону, получив возможность услышать ее голос. Распознавание голоса действительно является хорошей схемой идентификации личности. Если речь идет об открытом ключе, он может безопасно его повторить его даже при угрозе подслушивания. Если это секретный ключ, он может использовать для проверки ключа одностороннюю хэш-функцию. Оба TSD PGP (см. раздел 24.12.) и AT$T (см. Раздел 24.18) используют этот способ проверки ключей.

Иногда может даже не важно точно проверять, кому принадлежит открытый ключ. Может понадобиться проверить, что он принадлежит тому же человеку, что и год назад. Если кто-то посылает банку подписанное сообщение о переводе денег, банк волнует не то, кто конкретно снимает деньги, а только то, чтобы этот человек был тем, кто внес деньги в первый раз.

Обнаружение ошибок при передаче ключей

Иногда ключи искажаются при передаче. Это является проблемой, так как искаженный ключ может привести к мегабайтам нерасшифрованного шифротекста. Все ключи должны передаваться с обнаружением ошибок и исправлением битов. Таким образом ошибки при передаче могут быть легко обнаружены и, если потребуется, ключ может быть послан еще раз.

Одним из наиболее широко используемых методов является шифрование ключом некоторой постоянной величины и передача первых 2-4 байт этого шифротекста вместе с ключом. У получателя делается то же самое. Если шифрованные константы совпадают, то ключ был передан без ошибки. Вероятность ошибки находится в диапазоне от $1/2^{16}$ до $1/2^{32}$.

Обнаружение ошибок при дешифрировании

Иногда получатель хочет проверить, является ли его конкретный ключ правильным ключом симметричного дешифрирования. Если открытый текст сообщения представляет собой что-то похожее на ASCII, он может попытаться расшифровать и прочитать сообщение. Если открытый текст случаен, то существуют другие приемы.

Наивным подходом явилось бы присоединение к открытому тексту до шифрования **проверочного блока** - известного заголовка. Получатель Боб расшифровывает заголовок и проверяет, что он правилен. Это работает, но дает Еве известный кусочек открытого текста, что помогает ей криптоанализировать систему. Это также об-

легчает вскрытие шифров с коротким ключом, таких как DES и все экспортируемые шифры. Рассчитайте заранее один раз для каждого ключа проверочную сумму, затем используйте эту проверочную сумму для определ ения ключа в любом сообщении, которое вы перехватили после этого. *Любая* проверочная сумма ключа, в которую не включены случайные или, по крайней мере, различные данные, обладает этим свойством. По идее это очень похоже на генерацию ключей по ключ евым фразам.

Вот для этого способ получше [821]:

(1) Сгенерите вектор идентификации (отличный от используемого в сообщении).

(2) Используйте этот вектор идентификации для генерации большого блока битов: скажем, 512.

(3) Хэшируйте результат.

(4) Используйте те же фиксированные биты хэш-значения, скажем, 32, для контрольной суммы ключа.

Это тоже дает Еве какую-то информацию, но очень небольшую. Если она попытается использовать младшие 32 бита конечного хэш-значения для вскрытия грубой силой, ей придется для каждого вероятного ключа выпо лнить несколько шифрований и хэширование, вскрытие грубой силой самого ключа окажется быстрее.

Она не получит для проверки никаких известных кусочков открытого текста, и даже если она сумеет подбр осить нам наше же случайное значение, она никогда не получит от нас выбранный открытый текст, так как он будет преобразован хэш-функцией прежде, чем она его увидит.

8.5 Использование ключей

Программное шифрование рискованно. Ушли те дни, когда простые микрокомпьютеры работали под упра влением единственной программы. Сегодня время Macintosh System 7, Windows NT и UNIX. Невозможно сказать, когда операционная система остановит работающую программу шифрования, запишет все на диск и разрешит выполняться какой-то другой задаче. Когда операционная система, наконец, вернется к шифрованию, чтобы там не шифровалось, картинка может оказаться весьма забавной. Операционная система записала пр ограмму шифрования на диск, и ключ записан вместе с ней. Ключ, незашифрованный, будет лежать на диске, пока компьютер не напишет что-нибудь в эту же область памяти поверх. Это может случиться через несколько минут, а может через несколько месяцев. Этого может и никогда не случиться, но ключ все же может оказаться на диске в тот момент, когда жесткий диск густо прочесывается вашим противником. В приоритетной, многозадачной среде, для шифрования можно установить достаточно высокий приоритет, чтобы эта операция не пр ерывалась. Это снизило бы риск. Даже при этом система в целом в лучшем случае ненадежна.

Аппаратные реализации безопаснее. Многие из устройств шифрования разработаны так, чтобы любое вм ешательство приводило бы к уничтожению ключа. Например, в плате шифрования для IBM PS/2 залитый эпо ксидной смолой модуль содержит микросхему DES, батарею и память. Конечно, Вы должны верить, что прои зводитель аппаратуры правильно реализовал все необходимые свойства.

Ряд коммуникационных приложений, например, телефонные шифраторы, могут использовать **сеансовые ключи**. Сеансовым называется ключ, который используется только для одного сеанса связи - единственного телефонного разговора - и затем уничтожается. Нет смысла хранить ключ после того, как он был использован. И если вы используете для передачи ключа от одного абонента другому некоторый протокол обмена ключами, то этот ключ не нужно хранить и перед его использованием. Это значительно снижает вероятность компромет ации ключа.

Контроль использования ключей

В некоторых приложениях может потребоваться контролировать процесс использования сеансового ключа. Некоторым пользователям сеансовые ключи нужны только для шифрования или только для дешифрирования. Сеансовые ключи могут быть разрешены к использованию только на определенной машине или только в опр еделенное время. По одной из схем управления подобными ограничениями к ключу добавляется **вектор контроля** (Control Vector, CV), вектор контроля определяет для этого ключа ограничения его использования (см. раздел 24.1) [1025, 1026]. Этот CV хэшируется, а затем для него и главного ключа выполняется операция XOR. Результат используется как ключ шифрования для шифрования сеансового ключа. Полученный сеансовый ключ затем хранится вместе с CV. Для восстановления сеансового ключа нужно хэшировать CV и выполнить для него и главного ключа операцию XOR. Полученный результат используется для дешифрирования шифрованн ого сеансового ключа.

Преимущества этой схемы в том, что длина CV может быть произвольной, и что CV всегда хранится в открытом виде вместе с шифрованным ключом. Такая схема не выдвигает требований относительно устойчивости аппаратуры к взлому и предполагает отсутствие непосредственного доступа пользователей к ключам. Эта система рассматривается ниже в разделах 24.1 и 24.8.

8.6 Обновление ключей

Представьте себе шифрованный канал передачи данных, для которого вы хотите менять ключи каждый день. Иногда ежедневное распределение новых ключей является нелегкой заботой. Более простое решение - генерировать новый ключ из старого, такая схема иногда называется **обновлением ключа**.

Все, что нужно - это однонаправленная функция. Если Алиса и Боб используют общий ключ и применяют к нему одну и ту же однонаправленную функцию, они получат одинаковый результат. Они могут выбрать из результата нужные им биты и создать новый ключ.

Обновление ключей работает, но помните, что безопасность нового ключа определяется безопасностью старого ключа. Если Еве удастся заполучить старый ключ, она сможет выполнить обновление ключей самостоятельно. Однако, если старого ключа у Евы нет, и она пытается выпо отношению к шифрованному трафику по лнить вскрытие с использованием только шифротекста, обновление ключей является хорошим способом защиты для Алисы и Боба.

8.7 Хранение ключей

Наименее сложными при хранении ключей являются проблемы одного пользователя, Алисы, шифрующей файлы для последующего использования. Так как она является единственным действующим пользователем системы, только она и отвечает за ключ. В некоторых системах используется простой подход: ключ хранится в голове Алисы и больше нигде. Это проблемы Алисы - помнить ключ и вводить его всякий раз, когда ей нужно зашифровать или расшифровать файл.

Примером такой системы является IPS [881]. Пользователи могут либо вводить 64-битовый ключ непосредственно, либо ввести ключ как более длинную символьную строку. В последнем случае система генерирует 64-битовый ключ по строке символов, используя технику перемалывания ключа.

Другим решением является хранить ключ в виде карточки с магнитной полоской, пластикового ключа с встроенной микросхемой ROM (называемого **ROM-ключ**) или интеллектуальной карточки [556, 557, 455]. Пользователь может ввести свой ключ в систему, вставив физический носитель в считывающее устройство, встроенное в его шифрователь или подключенное к компьютерному терминалу. Хотя пользователь может использовать ключ, он не знает его и не может его скомпрометировать. Он может использовать его только тем способом и только для тех целей, которые определены вектором контроля.

ROM-ключ - это очень умная идея. Практически любой способен осознать, что такое физический ключ, каково его значение, и как его защитить. Придание криптографическому ключу некоторой физической формы делает хранение и защиту такого ключа интуитивно более понятным.

Эта техника становится более безопасной при разбиении ключа на две половины, одна из которых хранится в терминале, а вторая - в ROM-ключе. Так работает безопасный телефон STU-III правительства США. Потеря ROM-ключа не компрометирует криптографический ключ - замените этот ключ и все снова станет нормально. То же происходит и при потере терминала. Следовательно, компрометация ROM-ключа или системы не компрометирует криптографический ключ key - врагу нужно заполучить обе части.

Ключи, которые трудно запомнить можно хранить зашифрованными, используя что-то похожее на ключ шифрования ключей. Например, закрытый ключ RSA может быть зашифрован ключом DES и записан на диск. Для восстановления ключа RSA пользователь будет должен ввести ключ DES в программу дешифрирования.

Если ключи генерируются детерминировано (с помощью криптографически безопасного генератора псевдослучайных последовательностей), может быть при помощи легко запоминающегося пароля легче генерировать ключи повторно всякий раз, когда они понадобятся.

В идеале, ключ никогда не должен оказываться вне шифровального устройства в незашифрованном виде. Эта цель не всегда достижима, но к этому нужно стремиться.

8.8 Резервные ключи

Алиса работает главным финансистом в Secrets, Ltd. - "Наш девиз - Мы тебе не скажем." Как примерный служащий корпорации она в соответствии с инструкциями по безопасности шифрует все свои данные. К несчастью, она, проигнорировав инструкции по переходу улицы, попала под грузовик. Что делать президенту компании Бобу?

Если Алиса не оставила копии своего ключа, ему придется несладко. Весь смысл шифрования файлов - в невозможности восстановить их без ключа. Если Алиса не была дурой и не использовала плохих шифровальных программ, то ее файлы пропали навсегда.

У Боба есть несколько способов избежать этого. Простейший иногда называют **условным вручением клю-**

чей (см. раздел 4.14). Он требует, чтобы все сотрудники записали свои ключи на бумажках отдали их начальнику службы безопасности компании, который запрет их где-нибудь в сейф (или зашифрует их главным ключом). Теперь, чтобы не случилось с Алисой, Боб узнает ее ключ у начальника службы безопасности. Еще одну копию Боб также должен хранить в своем сейфе, в противном случае, если начальник службы безопасности попадет под другой грузовик, Бобу снова не повезет.

Проблема такой системы управления ключами в том, что Боб должен верить, что его начальник службы безопасности не воспользуется чужими ключами. Что еще серьезнее, все сотрудники должны верить, что начальник службы безопасности не воспользуется их ключами. Существенно лучшим решением является использование протокола совместного использования секрета (см. раздел 3.7).

Когда Алиса генерирует ключ, она одновременно делит ключ не несколько частей и затем посылает все части - зашифрованные, конечно - различным должностным лицам компании. Ни одна из этих частей сама по себе не является ключом, но все эти части можно собрать вместе и восстановить ключ. Теперь Алиса защищена от злоумышленников, а Боб - от потери всех данных Алисы после ее попадания под грузовик. Или, она может просто хранить разные части, зашифрованные открытыми ключами соответствующих должностных лиц компании, на своем жестком диске. Таким образом, никто не участвует в управлении ключами, пока это не станет необходимым.

Другая схема резервирования [188] использует для временного условного вручения ключей интеллектуальные карточки (см. раздел 24.13). Алиса может поместить ключ, которым закрыт ее жесткий диск, на интеллектуальную карточку и выдать ее Бобу, пока она в отъезде. Боб может использовать карточку для доступа к жесткому диску Алисы, но, так как ключ хранится на карточке, Боб не сможет его узнать. Кроме того, такая система контролируема с обеих сторон: Боб может проверить, что ключ открывает диск Алисы, а когда Алиса вернется, она сможет проверить, использовал ли Боб раз этот ключ, и если да, то сколько раз.

В подобной схеме не нужна передача данных. Для безопасного телефона ключ должен существовать только в течение разговора и не дольше. Для хранилищ данных, как было показано, условное вручение ключей может быть неплохой идеей. Я теряю ключи примерно раз в пять лет, а моя память получше, чем у многих. Если бы 200 миллионов человек пользовались криптографией, подобная частота привела бы к потере 40 миллионов ключей ежегодно. Я храню копии ключей от моего дома у соседа, потому что я могу потерять свои ключи. Если бы ключи от дома были подобны криптографическим ключам, то, потеряв их, я никогда не смог бы попасть внутрь и вступить в свои права владения. Также, как я храню где-то в другом месте копии своих данных, мне имеет смысл хранить и резервные копии моих ключей шифрования.

8.9 Скомпрометированные ключи

Все протоколы, методы и алгоритмы этой книги безопасны только, если ключ (закрытый ключ в системе с открытыми ключами) остается в тайне. Если ключ Алисы украден, потерян, напечатан в газете или скомпрометирован иным способом, то все ее безопасность исчезнет.

Если скомпрометированный ключ использовался для симметричной криптосистемы, Алисе придется изменить свой ключ и надеяться, что случившийся ущерб минимален. Если это закрытый ключ, ее проблемы намного больше, так как ее открытый ключ может храниться на многих серверах в сети. И если Ева получит доступ к закрытому ключу Алисы, она сможет выдать себя за нее в этой сети: читать шифрованную почту, подписывать корреспонденцию и контракты, и так далее. Ева действительно сможет стать Алисой.

Жизненно необходимо, чтобы известие о компрометации закрытого ключа быстро распространилось бы по сети. Нужно немедленно известить все базы данных открытых ключей о случившейся компрометации, чтобы ничего не подозревающий человек не зашифровал сообщение скомпрометированным ключом.

Хорошо, если Алиса знает, когда был скомпрометирован ее ключ. Если ключ распределяет KDC, то Алиса должна сообщить ему о компрометации своего ключа. Если KDC не используется, то ей следует известить всех корреспондентов, которые могут получать от нее сообщения. Кто-то должен опубликовать тот факт, что любое сообщение, полученное после потери ключа Алисой, является подозрительным, и что никто не должен посылать сообщения Алисе, используя соответствующий открытый ключ. Рекомендуется, чтобы программное обеспечение использовало какие-нибудь метки времени, тогда пользователи смогут определить, какие сообщения законны, а какие подозрительны.

Если Алиса не знает точно, когда ее ключ был скомпрометирован, то дело хуже. Алиса может захотеть отказаться от контракта, так как он подписан вместо нее человеком, укравшим у нее ключ. Если система дает такую возможность, то кто угодно сможет отказаться от контракта, утверждая, что его ключ был скомпрометирован перед подписанием. Вопрос должен быть решен арбитром.

Это серьезная проблема показывает, как опасно для Алисы связывать свою личность с единственным ключом. Лучше, чтобы у Алисы были различные ключи для различных приложений - точно также, как она держит в

своем кармане физические ключи для различных замков. Другие решения этой проблемы включают биометрические измерения, ограничения возможностей использования ключа, задержки времени и вторая подпись.

Эти процедуры и рекомендации наверняка не оптимальны, но это лучшее, что мы можем посоветовать. Мораль - защищайте ключи, и сильнее всего защищайте закрытые ключи.

8.10 Время жизни ключей

Ни один ключ шифрования нельзя использовать бесконечно. Время его действия должно истекать автоматически, подробно паспортам и лицензиям. Вот несколько причин этого:

— Чем дольше используется ключ, тем больше вероятность его компрометации. Люди записывают ключи и теряют их. Происходят несчастные случаи. Если вы используете ключ в течение года, то вероятность его компрометации гораздо выше, чем если бы вы использовали его только один день.

— Чем дольше используется ключ, тем больше потери при компрометации ключа. Если ключ используется только для шифрования одного финансового документа на файл-сервере, то потеря ключа означает компрометацию только этого документа. Если тот же самый ключ используется для шифрования всей финансовой информации на файл-сервере, то его потеря гораздо более разрушительна.

— Чем дольше используется ключ, тем больше соблазн приложить необходимые усилия для его вскрытия - даже грубой силой. Вскрытие ключа, используемого в течение дня для связи между двумя воинскими подразделениями, позволит читать сообщения, которыми обмениваются подразделения, и создавать поддельные. Вскрытие ключа, используемого в течение года всей военной командной структурой, позволило бы взломщику в течение года читать все сообщения, циркулирующие в этой системе по всему миру, и подделывать их. В нашем мире закончившейся холодный войны какой ключ выбрали бы для вскрытия вы?

— Обычно намного легче проводить криптоанализ, имея много шифротекстов, шифрованных одним и тем же ключом.

Для любого криптографического приложения необходима стратегия, определяющая допустимое время жизни ключа. У различных ключей могут быть различные времена жизни. Для систем с установлением соединения, таких как телефон, имеет смысл использовать ключ только в течение телефонного разговора, а для нового разговора - использовать новый ключ.

Для систем, использующих специализированные каналы связи, все не так очевидно. У ключей должно быть относительно короткое время жизни, в зависимости от значимости данных и количества данных, зашифрованных в течение заданного периода. Ключ для канала связи со скоростью передачи 1 Гигабит в секунду возможно придется менять гораздо чаще, чем для модемного канала 9600 бит/с. Если существует эффективный метод передачи новых ключей, сеансовые ключи должны меняться хотя бы ежедневно.

Ключи шифрования ключей так часто менять не нужно. Они используются редко (приблизительно раз в день) для обмена ключами. При этом шифротекста для криптоаналитика образуется немного, а у соответствующего открытого текста нет определенной формы. Однако, если ключ шифрования ключей скомпрометирован, потенциальные потери чрезвычайны: вся информация, зашифрованная ключами, зашифрованными ключом шифрования ключей. В некоторых приложениях ключи шифрования ключей заменяются только раз в месяц или даже раз в год. Вам придется как-то уравновесить опасность, связанную с использованием одного и того же ключа, и опасность, связанную с передачей нового ключа.

Ключи шифрования, используемые при шифровании файлов данных для длительного хранения, нельзя менять часто. Файлы могут храниться на диске зашифрованными месяцами или годами, прежде чем они кому-нибудь снова понадобятся. Ежедневное дешифрирование и повторное шифрование новым ключом никак не повысит безопасность, просто криптоаналитик получит больше материала для работы. Решением может послужить шифрование каждого файла уникальным ключом и последующее шифрование ключей файлов ключом шифрования ключей. Ключ шифрования ключей должен быть либо запомнен, либо сохранен в безопасном месте, может быть где-нибудь в сейфе. Конечно же, потеря этого ключа означает потерю всех индивидуальных файловых ключей.

Время жизни закрытых ключей для приложений криптографии с открытыми ключами зависит от приложения. Закрытые ключи для цифровых подписей и идентификации могут использоваться годами (даже в течение человеческой жизни). Закрытые ключи для протоколов бросания монеты могут быть уничтожены сразу же после завершения протокола. Даже если считается, что время безопасности ключа примерно равно человеческой жизни, благоразумнее менять ключ каждую пару лет. Во многих четях закрытые ключи используются только два года, затем пользователь должен получить новый закрытый ключ. Старый ключ, тем не менее, должен храниться в секрете на случай, когда пользователю будет нужно подтвердить подпись, сделанную во время действия старого ключа. Но для подписания новых документов должен использоваться новый ключ. Такая схема по-

зволит уменьшить количество документов, которое криптоаналитик сможет использовать для вскрытия .

8.11 Разрушение ключей

Принимая во внимание, что ключи должны регулярно меняться, старые ключи необходимо уничтожать. Старые ключи имеют определенное значение, даже если они никогда больше не используются. С их помощью враг сможет прочитать старые сообщения, зашифрованные этими ключами [65].

Ключи должны уничтожаться надежно (см. раздел 10.9). Если ключ записан на бумажке, бумажку нужно разрезать и сжечь. Пользуйтесь качественными уничтожителями бумаги, рынок заполнен дефектными устро йствами. Алгоритмы, описанные в этой книге, надежно противостоят вскрытию грубой силой, стоящему милли оны долларов и требующему миллионов лет. Если враг сможет раскрыть ваш ключ, добыв плохо измельченные документы из вашего мусорника и наняв сотню безработных в какой-нибудь отсталой стране за 10 центов в час склеивать вместе кусочки разрезанных страниц, он выгодно вложит пару десятков тысяч долларов .

Если ключ - это микросхема EEPROM, то ключ необходимо переписать несколько раз. Если ключ - это микросхема EPROM или PROM, то она должна быть стерта в порошок и развеяна во все стороны. Если ключ хранится на диске компьютера, действительные биты соответствующего участка памяти должны быть переписаны несколько раз (см. раздел 10.9) или диск должен быть уничтожен.

Возможная проблема состоит в том, что в компьютере ключи могут быть легко скопированы и сохранены во множестве мест. Любой компьютер, реализующий какую-либо схему управления памятью, постоянно выгруж ает программы из памяти и загружает их обратно, усугубляя проблему. Способа гарантировать надежное уни чтожение ключа в компьютере не существует, особенно когда процесс уничтожения контролируется операцио нной системой компьютера. Самым озабоченным необходимо использовать специальную программу, которая на физическом уровне искала бы на диске копию ключа даже в неиспользуемых блоках и затем стирала бы соо тветствующие блоки. Не забывайте также стирать все временных файлов.

8.12 Управление открытыми ключами

Криптография с открытыми ключами упрощает управление ключами, но у нее есть свои собственные пр облемы. У каждого абонента, независимо от числа людей в сети, есть только один открытый ключ . Если Алиса захочет отправить Бобу сообщение, ей придется где-то найти открытый ключ Боба. Она может действовать не сколькими способами:

— Получить ключ от Боба.

— Получить его из централизованной базы данных.

— Получить его из своей личной базы данных.

В разделе 2.5 обсуждаются возможные способы вскрытия криптографии с открытыми ключами , основанных на подмене ключа Боба ключом Мэллори. Используется следующий сценарий: пусть Алиса хочет послать с ообщение Бобу. Она обращается к базе данных открытых ключей и получает открытый ключ Боба . Но подлый Мэллори подменяет ключ Боба своим собственным. (Если Алиса запрашивает ключ непосредственно у Боба, Мэллори для успешной подмены придется перехватить ключ Боба при передаче .) Алиса шифрует сообщение ключом Мэллори и отправляет его Бобу. Мэллори перехватывает сообщение, расшифровывает и читает его . Затем шифрует открытым ключом Боба и отправляет по назначению. Ни Боб, ни Алиса ни о чем не догадыв аются.

Заверенные открытые ключи

Заверенным открытым ключом, или сертификатом, является чей-то открытый ключ, подписанный засл уживающим доверия лицом. Заверенные ключи используются, чтобы помешать попыткам подмены ключа [879]. Заверенный ключ Боба в базе данных открытых ключей состоит не только из открытого ключа Боба . Он содержит информацию о Бобе - его имя, адрес, и т.д. - и подписан кем-то, кому Алиса доверяет - Трентом (обычно известным как **орган сертификации**, certification authority, или CA). Подписав и ключ, и сведения о Бобе, Трент заверяет, что информация о Бобе правильна, и открытый ключ принадлежит ему . Алиса проверяет подпись Трента и затем использует открытый ключ, убедившись в том, что он принадлежит Бобу и никому другому. Заверенные ключи играют важную роль во многих протоколах с открытыми ключами, например, PEM [825] (см. раздел 24.10) и X.509 [304] (см. раздел 24.9).

В таких системах возникает сложная проблема, не имеющая прямого отношения к криптографии . Каков смысл процедуры заверения? Или, иначе говоря, кто для кого имеет полномочия выдавать сертификаты? Кто угодно может заверит своей подписью чей угодно открытый ключ, но должен же быть какой-то способ отфиль тровать ненадежные сертификаты: например, открытые ключи сотрудников компании, заверенные CA другой компании. Обычно создается цепочка передачи доверия: один надежный орган заверяет открытые ключи дов е-

ренных агентов, те сертифицируют CA компании, а CA компании заверяют открытые ключи своих работников. Вот еще вопросы, над которыми стоит подумать:

— Какой уровень доверия к чьей-то личности обеспечивает сертификат?

— Каковы взаимоотношения между человеком и CA, заверяющим его открытый ключ, и как эти отношения отражаются в сертификате?

— Кому можно доверить быть "одним надежным органом", возглавляющим сертификационную цепочку?

— Насколько длинной может быть сертификационная цепочка?

В идеале прежде, чем CA подпишет сертификат Боба, Бобу нужно пройти определенную процедуру авторизации. Кроме того, для защиты от скомпрометированных ключей важно использовать какие-нибудь метки времени или признаки срока действия сертификата [461].

Использование меток времени недостаточно. Ключи могут стать неправильными задолго до истечения их срока либо из-за компрометации, либо по каким-то административным причинам. Следовательно, важно, чтобы CA хранил список неправильных заверенных ключей, а пользователи регулярно сверялись бы с этим списком. Эта проблема отмены ключей все еще трудна для решения.

К тому же, одной пары открытый ключ/закрытый ключ недостаточно. Конечно же, в любая хорошая реализация криптографии с открытыми ключами должна использовать разные ключи для шифрования и для цифровых подписей. Такое разделение разрешает различные Это разделение учитывает различные уровни защиты, сроки действия, процедуры резервирования, и так далее. Кто-то может подписывать сообщения 2048-битовым ключом, который хранится на интеллектуальной карточке и действует двадцать лет, а кто-то может использовать для шифрования 768-битовый ключ, который хранится в компьютере и действует шесть месяцев.

Однако, одной пары для шифрования и одной для подписи также недостаточно. Закрытый ключ может идентифицировать роль человека также, как и личность, а у людей может быть несколько ролей. Алиса может хотеть подписать один документ как лично Алиса, другой - как Алиса, вице-президент Monolith, Inc., а третий - как Алиса, глава своей общины. Некоторые из этих ключей имеют большее значение, чем другие, поэтому они должны быть лучше защищены. Алисе может потребоваться хранить резервную копию своего рабочего ключа у сотрудника отдела безопасности, а она не хочет, чтобы у компании была копия ключа, которым она подписала закладную. Алиса собирается пользоваться несколькими криптографическими ключами точно также, как она использует связку ключей из своего кармана.

Распределенное управление ключами

В некоторых случаях такой способ централизованного управления ключами работать не будет. Возможно, не существует такого CA, которому доверяли бы Алиса и Боб. Возможно, Алиса и Боб доверяют только своим друзьям. Возможно, Алиса и Боб никому не доверяют.

Распределенное управление ключами, используемое в PGP (см. раздел 24.12), решает эту проблему с помощью поручителей. Поручители - это пользователи системы, которые подписывают открытые ключи своих друзей. Например, когда Боб создает свой открытый ключ, он передает копии ключа своим друзьям - Кэрол и Дэйву. Они знают Боба, поэтому каждый из них подписывает ключ Боба и выдает Бобу копию своей подписи. Теперь, когда Боб предъявляет свой ключ чужому человеку, Алисе, он предъявляет его вместе с подписями этих двух поручителей. Если Алиса также знает Кэрол и доверяет ей, у нее появляется причина поверить в правильность ключа Боба. Если Алиса знает Кэрол и Дэйва и хоть немного доверяет им, у нее также появляется причина поверить в правильность ключа Боба. Если она не знает ни Кэрол, ни Дэйва у нее нет причин доверять ключу Боба.

Спустя какое-то время Боб соберет подписи большего числа поручителей. Если Алиса и Боб вращаются в одних кругах, то с большой вероятностью Алиса будет знать одного из поручителей Боба. Для предотвращения подмены Мэллори одного ключа другим поручитель должен быть уверен, прежде чем подписывать ключ, что этот ключ принадлежит именно Бобу. Может быть, поручитель потребует передачи ключа при личной встрече или по телефону.

Выгода этого механизма - в отсутствии CA, которому каждый должен доверять. А отрицательной стороной является отсутствие гарантий того, что Алиса, получившая открытый ключ Боба, знает кого-то из поручителей, и, следовательно, нет гарантий, что она поверит в правильность ключа.

Глава 9
Типы алгоритмов и криптографические режимы

Существует два основных типа симметричных алгоритмов: блочные шифры и потоковые шифры. **Блочные шифры** работают с блоками открытого текста и шифротекста - обычно длиной 64 бита, но иногда длиннее. **Потоковые шифры** работают с битовыми или байтовыми потоками открытого текста и шифротекста (иногда даже с потоками 32-битных слов). Блочный шифр, использующий один и тот же ключ, при шифровании всегда превращает один и тот же блок открытого текста в один и тот же блок шифротекста. Потоковый шифр при каждом шифровании превращает один и тот же бит или байт открытого текста в различные биты или байты шифротекста.

Криптографический **режим** обычно объединяет базовый шифр, какую-то обратную связь и ряд простых операций. Операции просты, потому что безопасность является функцией используемого шифра, а не режима. Более того, режим шифра не должен компрометировать безопасность используемого алгоритма.

Существуют и другие соображения безопасности: должна быть скрыта структура открытого текста, должен быть рандомизирован ввод шифра, должно быть затруднено манипулирование открытым текстом посредством ввода ошибок в шифротекст, должно быть возможно шифрование нескольких сообщений одним ключом. Все это будет подробно рассматриваться в следующих разделах.

Другим важным соображением является эффективность. По эффективности режим не может быть сильно хуже используемого алгоритма. В некоторых обстоятельствах важно, чтобы размер шифротекста совпадал с размером открытого текста.

Третьим соображением является устойчивость к сбоям. Для ряда приложений требуется распараллеливать шифрование или дешифрирование, а другим нужна возможность выполнить как можно большую предобработку. В третьих важно, чтобы процесс дешифрирования умел исправлять сбои битов в потоке шифротекста, а также был устойчив к потере и добавлению битов. Как будет показано, различные режимы обладают различными подмножествами этих характеристик.

9.1 Режим электронной шифровальной книги

Режим **электронной шифровальной книги** (electronic codebook, ECB) - это наиболее очевидный способ использовать блочный шифр: блок открытого текста заменяется блоком шифротекста. Так как один и тот же блок открытого текста заменяется одним и тем же блоком шифротекста, то теоретически возможно создать шифровальную книгу блоков открытого текста и соответствующих шифротекстов. Однако, если размер блока - 64 бита, то кодовая книга будет состоять из 2^{64} записей - слишком много для предварительного вычисления и хранения. И не забывайте, для каждого ключа понадобится отдельная шифровальная книга.

Это самый легкий режим работы. Все блоки открытого текста шифруются независимо. Нет необходимости в последовательном шифровании файла, можно зашифровать сначала 10 блоков из середины текста, затем последние блоки, и наконец, первые. Это важно для шифрованных файлов с произвольным доступом, например, для баз данных. Если база данных зашифрована в режиме ECB, то любая запись может быть добавлена, удалена, зашифрована или расшифрована независимо от любой другой записи (при условии, что каждая запись состоит из целого числа блоков шифрования). Кроме того, обработка может быть распараллелена, если используются несколько шифровальных процессоров, они могут независимо друг от друга шифровать или дешифровать различные блоки.

Проблемой режима ECB является то, что если у криптоаналитика есть открытый текст и шифротекст для нескольких сообщений, он может начать составлять шифровальную книгу, не зная ключа. В большинстве реальных ситуаций фрагменты сообщений имеют тенденцию повторяться. В различных сообщениях могут быть одинаковые битовые последовательности. У сообщений, которые подобно электронной почте создаются компьютером, может быть регулярная структура. Сообщения могут иметь высокую степень избыточности или содержать длинные строки нулей или пробелов.

Если криптоаналитик знает, что блок открытого текста "5e081bc5" при шифровании превращается в блок шифротекста "7ea593a4," то он может мгновенно расшифровать этот блок шифротекста, в каком-бы другом сообщении он не появился. Если в шифрованном сообщении много повторов, которые имеют тенденцию занимать одинаковое место в различных сообщениях, криптоаналитик может получить много информации. Он может попытаться статистически вскрыть используемый открытый текст, независимо от силы блочного шифра.

Особенно уязвимы начало и окончание сообщений, где находится информация об отправителе, получателе дате и т.д. Эта проблема иногда называется **стандартными заголовками** и **стандартными окончаниями**.

Положительной стороной является возможность шифровать несколько сообщений одним ключом без сниже-

ния безопасности. По сути, каждый блок можно рассматривать как отдельное сообщение, шифрованное тем же самым ключом. При дешифрировании битовые ошибки в шифротексте приводят к неправильному дешифрированию соответствующего блока открытого текста, но не влияет на остальной открытый текст . Однако, если бит шифротекста случайно потерян или добавлен, то весь последующий шифротекст будет расшифрован неправильно, если для выравнивания границ блоков не используется какая-нибудь кадровая структура .

Набивка

Большинство сообщений точно не делятся на 64-битные (или любого другого размера) блоки шифрования, в конце обычно оказывается укороченный блок. ECB требует использовать 64-битные блоки. Способом решения этой проблемы является **набивка**.

Последний блок дополняется (набивается) некоторым регулярным шаблоном - нулями, единицами, чередующимися нулями и единицами - для получения полного блока. При необходимости удалить набивку после дешифрирования запишите количество байтов набивки в последний байт последнего блока . Например, пусть размер блока - 64 бита, и последний блок состоит из 3 байтов (24 бит) . Для дополнения блока до 64 бит требуется пять байтов, добавьте четыре байта нулей и последний байт с числом 5. После дешифрирования удалите последние 5 байтов последнего расшифрованного блока. Чтобы этот метод работал правильно, каждое сообщение должно быть дополнено. Даже если открытый текст содержит целое число блоков, вам придется добавить один полный блок. С другой стороны, можно использовать символ конца файла для обозначения последнего байта открытого текста и дополнить этот символ er.

На 8-й показан другой вариант, называемый **похищением шифротекста** [402]. P_{n-1} - последний полный блок открытого текста, а P_n - последний, короткий блок открытого текста. C_{n-1} - последний полный блок шифротекста, и C_n - последний, короткий блок шифротекста. C' - это промежуточный результат, не являющийся частью переданного шифротекста.

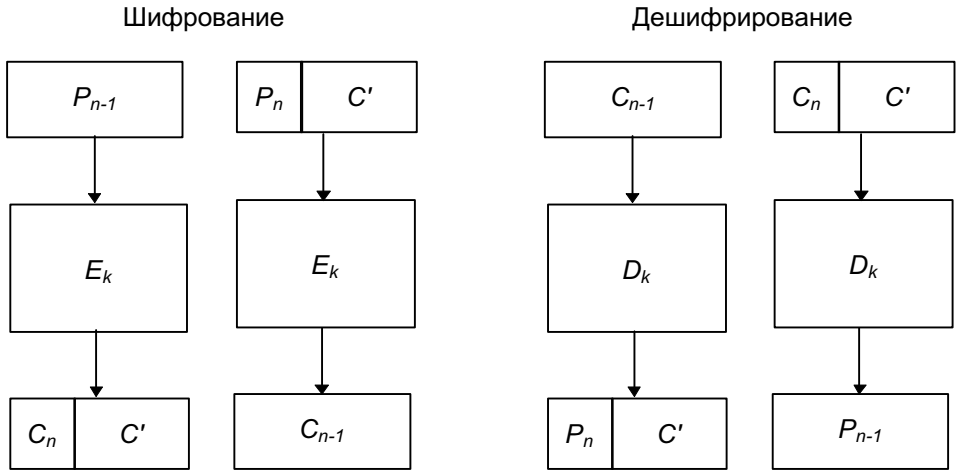


Рис. 9-1. Похищение шифротекста.

9.2 Повтор блока

Более серьезной проблемой режима ECB является то, что враг может изменить шифрованные сообщения, не зная ключа или даже алгоритма, чтобы обмануть предполагаемого получателя. Впервые эта проблемы была рассмотрена в [291].

Для иллюстрации этой проблемы рассмотрим систему передачи денег, которая переводит деньги из банка в банк. Чтобы облегчить жизнь банковских компьютеров, банки согласовали примерно следующий стандартный формат сообщения для передачи денег:

```
Банк 1: Передача            1.5 блока
Банк 2: Прием               1.5 блока
Имя вкладчика               6 блоков
Счет вкладчика              2 блока
Сумма вклада                1 блок
```

Блок соответствует 8-байтному блоку шифрования. Сообщения шифруются с помощью некоторого блочного алгоритма в режиме ECB.

Мэллори, который подслушивает линию связи между банками, банком Алисы и банком Боба, может использовать эту информацию для обогащения. Сначала, он программирует свой компьютер для записи всех шифрованных сообщений из банка Алисы в банк Боба. Затем, он переводит $100 из банка Алисы на свой счет в банк

Боба. Позже, он повторяет эту операцию еще раз. С помощью своего компьютера он проверяет записанные сообщения, разыскивая пару идентичных сообщений. Этими сообщениями являются те сообщения, которыми он переводит $100 на свой счет. Если он находит несколько пар одинаковых сообщений (что больше похоже на реальную жизнь), он делает еще один денежный перевод и записывает результат. В конце концов он сможет выделить сообщение, которым был проведен именно его перевод.

Теперь он может отправить это сообщение по каналу связи, когда захочет. Каждое сообщение приведет к зачислению на его счет в банке Боба еще $100. Когда оба банка сверят свои переводы (возможно в конце дня), они обнаружат переводы-призраки, но если Мэллори достаточно умен, он уже сбежит в какую-нибудь банановую республику без договора об экстрадиции, прихватив с собой деньги. И скорее всего он использует суммы несколько больше $100 и провернет операцию сразу для нескольких банков.

На первый взгляд банки могут легко пресечь это, добавляя метки времени к своим сообщениям.

```
Метка даты/времени          1 блок
Банк 1: Передача            1.5 блока
Банк 2: Прием               1.5 блока
Имя вкладчика               6 блоков
Счет вкладчика              2 блока
Сумма вклада                1 блок
```

В такой системе два идентичных сообщения будут легко обнаружены. Тем не менее, с помощью метода, называемого **повтором блока**, Мэллори все же сможет обогатиться. На 7-й показано, что Мэллори может собрать восемь блоков шифротекста, соответствующих его имени и номеру счета: блоки с 5 по 12. В этот момент уместно дьявольски рассмеяться, ведь Мэллори уже в полной готовности.

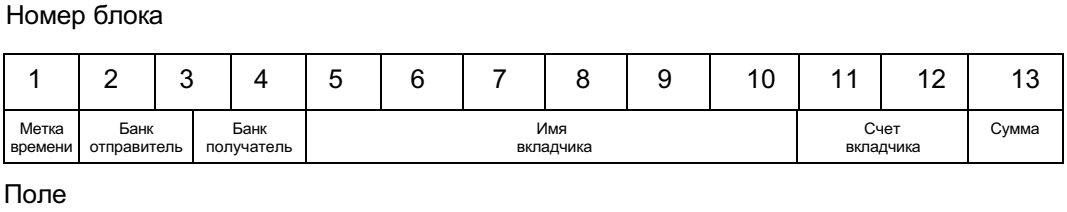


Рис. 9-2. Блоки шифрования в записи приведенного примера.

Он перехватывает сообщения из банка Алисы в банк Боба и заменяет блоки с 5 по 12 сообщения байтами, соответствующими его имени и номеру счета. Затем он посылает измененные сообщения в банк Боба. Ему не нужно знать, кто был отправителем денег, ему даже не нужно знать переводимую сумму (хотя он может связать подправленное сообщение с соответствующим увеличением своего счета и определить блоки, соответствующие определенным денежным суммам). Он просто изменяет имя и номер счета на свои собственные и следит за ростом своих доходов. (Я помню, что Мэллори надо быть осторожным, чтобы не модифицировать сообщение о снятии денег, но предположим на минутку, что у этих сообщений другая длина или иной отличительный признак.)

Для обнаружения такого способа банкам одного дня не хватит. Когда они сверят свои переводы в конце дня, все суммы совпадут. Возможно, пока настоящий вкладчик не заметит, что его вклады не зачисляются на счет, или пока кто-нибудь не обратит внимание на неожиданную активизацию работы со счетом Мэллори, банки не смогут заметить никаких следов. Мэллори не глуп и к этому времени закроет свой счет, изменит имя и купит виллу в Аргентине.

Банки могут минимизировать эту проблему, часто меняя свои ключи, но это означает только, что Мэллори придется действовать побыстрее. Однако, добавление MAC также решит проблему. Несмотря на это рассматриваемая проблема фундаментальна для режима ECB. Мэллори удалять, повторять или заменять блоки по своему усмотрению. Решением является способ, называемый **сцеплением**.

9.3 Режим сцепления блоков шифра

Сцепление добавляет к блочному шифру механизм обратной связи: результаты шифрования предыдущих блоков влияют на шифрование текущего блока. Другими словами, каждый блок используется для изменения шифрования следующего блока. Каждый блок шифротекста зависит не только от шифруемого блока открытого текста, но и от всех предыдущих блоков открытого текста.

В режиме **сцепления блоков шифра** (cipher block chaining, CBC) перед шифрованием над открытым текстом и предыдущим блоком шифротекста выполняется операция XOR. На 6-й (а) показано шифрование CBC в действии. ,Когда блок открытого текста зашифрован, полученный шифротекст сохраняется в регистре обратной связи. Прежде чем будет зашифрован следующий блок открытого текста, он подвергается операции XOR вместе

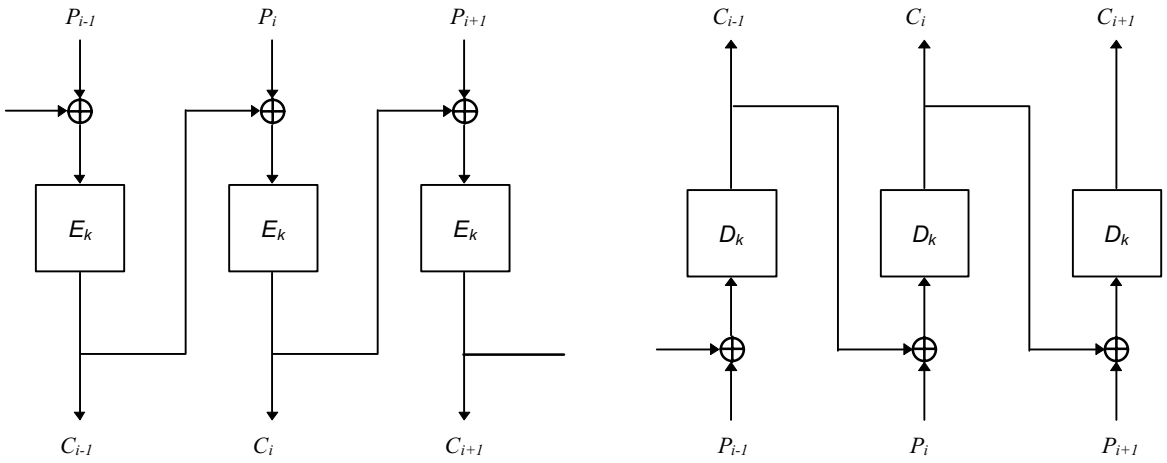
с содержимым регистра обратной связи. Таким образом создаются входные данные для следующего этапа процедуры шифрования. Полученный шифротекст снова сохраняется в регистре обратной связи, чтобы подвергнуться операции XOR вместе со следующим блоком открытого текста, и так до конца сообщения. Шифрование каждого блока зависит от всех предыдущих блоков.

Дешифрирование является обратной операцией (см. Figure 9.3 (б)). Блок шифротекста расшифровывается как обычно, но сохраняется в регистре обратной связи. Затем следующий блок дешифрируется и подвергается операции XOR вместе с содержимым регистра обратной связи. Теперь следующий блок шифротекста сохраняется в регистре обратной связи, и так далее, до конца сообщения.

Математически это выглядит следующим образом:

$$C_i = E_K(P_i \oplus C_{i-1})$$

$$P_i = C_{i-1} \oplus D_K(C_i)$$



(а) Шифрование CBC

(б) Дешифрирование CBC

Рис. 9-3. Режим сцепления блоков шифра.

Вектор инициализации

В режиме CBC одинаковые блоки открытого текста при шифровании переходят в различные блоки шифротекста только, если отличались какие-то из предшествующих блоков открытого текста. Два идентичных сообщения, однако, будут шифроваться как один и тот же шифротекст. Что еще хуже, два одинаково начинающихся сообщения будут шифроваться одинаково, пока не появится первое различие.

У ряда сообщений может быть одинаковый заголовок - тема письма, строка "From" или еще что-нибудь. Хотя повтор блока будет невозможен, такое одинаковое начало может предоставить криптоаналитику какую-нибудь полезную информацию.

Избежать этого можно, шифруя в качестве первого блока какие-то случайные данные. Этот блок случайных данных называется вектором инициализации (initialization vector, IV), инициализирующей переменной или начальным значением сцепления. IV не имеет никакого смыслового значения, он используется только для того, чтобы сделать каждое сообщение уникальным. Когда получатель расшифровывает этот блок, он испольтзует его только для заполнения регистра обратной связи. Хорошим IV служит метка времени. Или используйте какие-нибудь случайные биты.

С использованием IV сообщения с идентичным открытым текстом при шифровании переходят в сообщения с различным шифротекстом. Следовательно, злоумышленник не сможет предпринять повтор блока, и затруднится создание шифровальной книги. Хотя рекомендуется для каждого сообщения, шифруемого одним и тем же ключом, выбирать уникальный IV, это требование не является обязательным.

IV не должен храниться в секрете, он может передаваться открыто вместе с шифротекстом. Если вы не понимаете почему, рассмотрите следующий довод. Пусть наше сообщение состоит из нескольких блоков: B_1, B_2, ..., B_i. B_1 шифруется вместе с IV. B_2 шифруется с использованием шифротекста B_1 в роли IV. B_3 шифруется с использованием шифротекста B_2 в роли IV, и так далее. Итак, если количество блоков - n, то $n-1$ "векторов инициализации" открыты, даже если первоначальный IV хранится в секрете. Поэтому причин хранить в секрете IV нет, IV - это просто блок-заглушка, можно считать его нулевым блоком сцепления B_0.

Набивка

Набивка используется также, как и в режиме ECB, но в некоторых приложениях размер шифротекст должен в точности совпадать с размером открытого текста. Может быть, зашифрованный файл должен занять в точности тот же объем памяти, что и файл открытого текста. В этом случае последний короткий блок придется шифровать иначе. Пусть последний блок состоит из *l* битов. Зашифровав последний полный блок, снова зашифруйте шифротекст, выберите старшие *l* битов и выполните для них и короткого блока операцию XOR, создавая шифротекст. Эта процедура показана на 5-й.

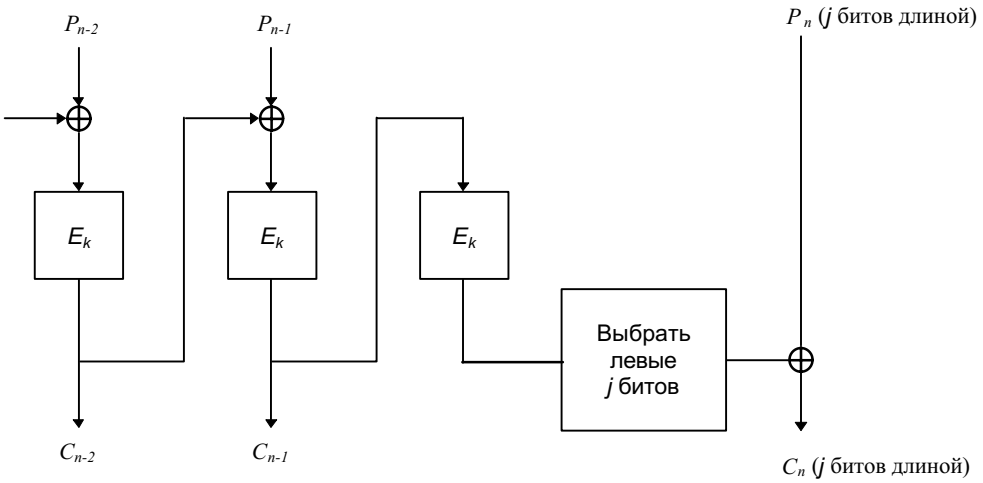


Рис. 9-4. Шифрование короткого последнего блока в режиме CBC.

Слабость этого способа в том, что хотя Мэллори не сможет раскрыть последний блок шифротекста, он может систематически изменять его, меняя отдельные биты шифротекста. Если последние несколько битов шифротекста содержат важную информацию, это опасно. Если последние биты просто содержат совет по домоводству, то ничего страшного.

Лучшим способом является похищение щифротекста (см. 4th) [402]. P_{n-1} - последний полный блок открытого текста, P_n - заключительный, короткий блок открытого текста. C_{n-1} - последний полный блок шифротекста, C_n - заключительный, короткий блок шифротекста. C' - это просто промежуточный результат, не являющийся частью переданного шифротекста. Преимуществом этого метода является то, что все биты открытого текста сообщения проходят через алгоритм шифрования.

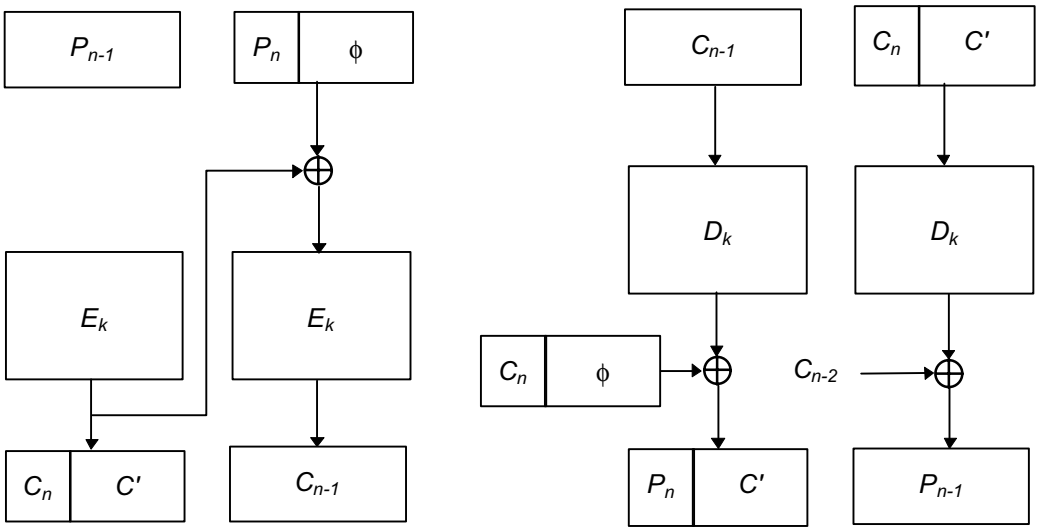


Рис. 9-5. Похищение шифротекста в режиме CBC.

Распространение ошибки

Режим CBC характеризуется **прямой обратной связью** шифротекста при шифровании и **инверсной обратной связью** шифротекста при дешифрировании. При этом приложения должны уметь бороться с ошибками. Единственная битовая ошибка в блоке открытого текста повлияет на данный блок шифротекста и все последующие блоки шифротекста. Это не важно, потому что дешифрирование инвертирует этот эффект, и восстано-

ленный открытый текст будет содержать ту же единственную ошибку.

Чаще встречаются ошибки шифротекста. Они легко появляются из-за шума линий передачи или сбоев устройств хранения. В режиме CBC ошибка одного бита шифротекста влияет на один блок и один бит восстановленного открытого текста. Блок, соответствующий содержащему ошибку блоку шифротекста, искажается полностью. В следующем блоке искажается единственный бит, находящийся в той же позиции, что и ощибочный бит.

Это свойство превращения малой ошибки шифротекста в большую ошибку открытого текста называется **распространением ошибки**. Это является главным недостатком. Эта ошибка не влияет на блоки, расположенные через один от испорченного и далее, поэтому режим CBC является **самовосстанавливающимся**. Ошибка влияет на два блока, но система продолжает работать правильно для всех последующих блоков. CBC представляет собой пример блочного шифра, используемого в самосинхронизирующейся манере, но только на блоковом уровне.

Хотя режим CBC быстро восстанавливается от битового сбоя, он абсолютно не устойчив к ошибкам синхронизации. Если в потоке шифротекста теряется или добавляется бит, то положение всех последующих блоков сдвигаются на один бит, и на выходе дешифрирования будет сплошной мусор. Любая криптосистема, использующая режим CBC должна обеспечивать целостность блочной структуры либо при помощи кадров, либо сохраняя данные в структуры из нескольких блоков.

Вопросы безопасности

Ряд возможных проблем обуславливаются структурой CBC. Во первых, так как блок шифротекста достаточно просто влияет на следующий блок, Мэллори может тайно добавлять блоки к концу зашифрованного сообщения. Конечно, при дешифрировании они превратятся в чепуху, но в некоторых ситуациях это нежелательно.

При использовании CBC вы должны структурировать ваш открытый текст так, чтобы вы знали, где находятся концы сообщений, и могли обнаружить добавление лишних блоков.

Во вторых, Мэллори может изменить блок шифротекста, изменения определенным образом блоки расшифрованного открытого текста. Например, если Мэллори изменит один бит шифротекста, весь блок будет расшифрован неправильно, а в следующем блоке в соответствующей позиции будет неправильный бит. Возможны ситуации, когда это нежелательно. Открытое сообщения должно обладать некоторой избыточностью или средствами идентификации.

Наконец, хотя структура открытого текста маскируется сцеплением, структура очень длинных сообщений все равно будет заметна. Парадокс дня рождения предсказывает, что после $2^{m/2}$ блоков, где m - размер блока, появляются одинаковые блоки. Для 64-битового блока длина такого сообщения примерно равны 32 Гбайтам. Подобная проблема возникает только для сообщений немаленького размера.

9.4 Потоковые шифры

Потоковые шифры преобразуют открытый текст в шифротекст по одному биту за операцию. Простейшая реализация потокового шифра показана на 3-й. **Генератор потока ключей** (иногда называемый генератором с бегущим ключом) выдает поток битов: k_1, k_2, k_3, ..., k_i. Этот поток ключей (иногда называемый бегущим ключом) и поток битов открытого текста, p_1, p_2, p_3, ..., p_i, подвергаются операции "исключающее или", и в результате получаетсяы поток битов шифротекста.

$c_i = p_i \oplus k_i$

При дешифрировании операция XOR выполняется над битами шифротекста и тем же самым потоком ключей для восстановления битов открытого текста.

$p_i = c_i \oplus k_i$

Так как

$p_i \oplus k_i \oplus k_i = p_i$

это работает правильно.

Безопасность системы полностью зависит от свойств генератора потока ключей. Если генератор потока ключей выдает бесконечную строку нулей, шифротекст будет совпадать с открытым текстом, и все операция будет бессмысленна. Если генератор потока ключей выплевывает повторяющийся 16-битовый шаблон, алгоритм будет являться простым XOR с пренебрежимо малой безопасностью (см. раздел 1.4). Если генератор потока ключей выплевывает бесконечный поток случайных (по настоящему, а не псевдослучайных - см. раздел 2.8) битов, вы получаете одноразовый блокнот и идеальную безопасность.

На деле безопасность потокового шифра находится где-то между простым XOR и одноразовым блокнотом.

Генератор потока ключей создает битовый поток, который похож на случайный, но в действительности детерминирован и может быть безошибочно воспроизведен при дешифрировании. Чем ближе выход генератора потока ключей к случайному, тем больше времени потребуется криптоаналитику, чтобы взломать шифр.

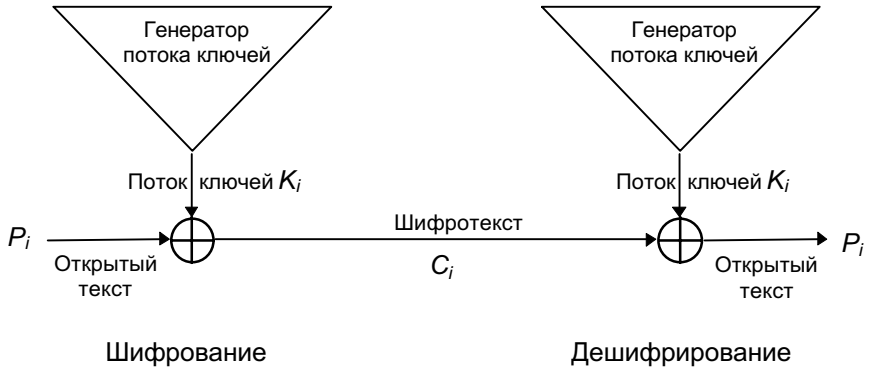


Рис. 9-6. Потоковый шифр

Однако, если генератор потока ключей при каждом включении создает один и тот же битовый поток, то использующую его криптосистему взломать нетрудно. Покажем на примере, почему это так.

Если к Еве попал шифротекст и соответствующий открытый текст, то она, выполняя операцию XOR над открытым текстом и шифротекстом, раскрывает поток ключей. Или, если у нее есть два различных шифротекста, зашифрованных одинаковым ключом, она может выполнить над ними операцию XOR, получая два открытых текста сообщений, над которыми выполнена операция XOR. Это нетрудно взломать, и затем она может получить поток ключей, выполняя операцию XOR над одним из открытых текстов и шифротекстом.

Теперь, перехватив любое другое шифрованное сообщение, она сможет расшифровать его, используя полученный поток ключей. Кроме того, она может расшифровать и прочитать любое из ранее перехваченных сообщений. Когда Ева получит пару открытый текст/шифротекст, она сможет читать все.

Поэтому для всех потоковых шифров используются ключи. Выход генератора потока ключей является функцией ключа. Теперь, если Ева получит пару открытый текст/шифротекст, она сможет читать только те сообщения, которые зашифрованы тем же ключом. Измените ключ, и противнику придется начать все сначала. Потоковые шифры особенно полезны для шифрования бесконечных потоков коммуникационного трафика, например, канала T1, связывающего два компьютера.

Генератор потока ключей состоит из трех основных частей (см. 2nd). Внутреннее состояние описывает текущее состояние генератора потока ключей. Два генератора потока ключей, с одинаковым ключом и одинаковым внутренним состоянием, выдают одинаковые потоки ключей. Функция выхода по внутреннему состоянию генерирует бит потока ключей. Функция следующего состояния по внутреннему состоянию генерирует новое внутреннее состояние.

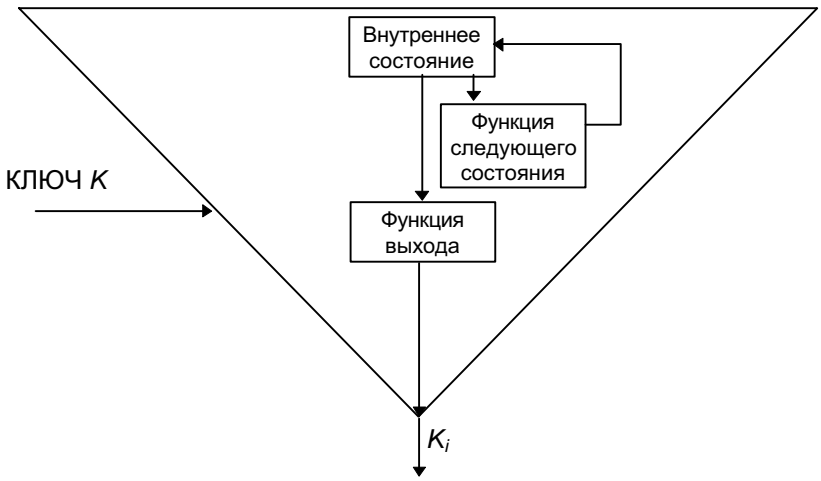


Рис. 9-7. Устройство генератора потока ключей.

9.5 Самосинхронизирующиеся потоковые шифры

В **самосинхронизирующихся потоковых шифрах** каждый бит потока ключей является функцией фиксированного числа предыдущих битов шифротекста [1378]. Военные называют этот шифр **автоключом шифротекста** (ciphertext auto key, CTAK). Основная идея была запатентована в 1946 [667].

Самосинхронизирующийся потоковый шифр показан на 1-й. Внутреннее состояние является функцией предыдущих n битов шифротекста. Криптографически сложной является выходная функция, которая использует внутреннее состояние для генерации бита потока ключей.

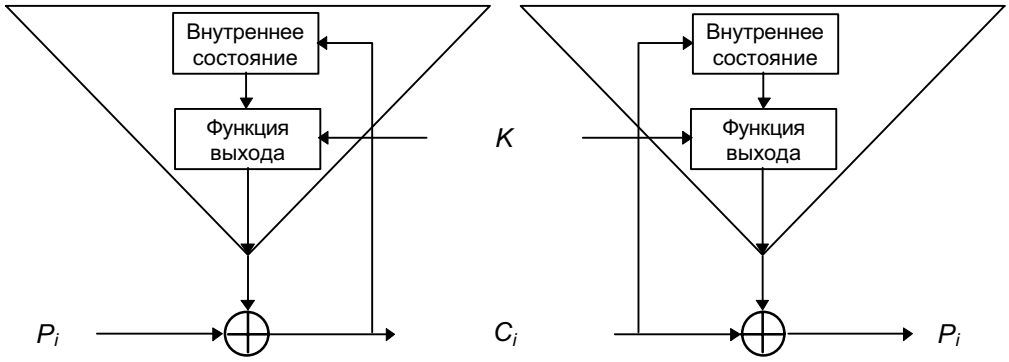


Рис. 9-8. Самосинхронизирующийся генератор потока ключей.

Так как внутреннее состояние полностью зависит от предыдущих n шифротекста, дешифрирующий генератор потока ключей автоматически синхронизируется с шифрующим генератором потока ключей, приняв n битов шифротекста.

В интеллектуальных реализациях этого режима каждое сообщение начинается случайным заголовком длиной n битов. Этот заголовок шифруется, передается и затем расшифровывается. Расшифровка будет неправильной, но после этих n битов оба генератора потока ключей будут синхронизированы.

Слабой стороной самосинхронизирующегося потокового шифра является распространение ошибки. Для каждого бита шифротекста, испорченного при передаче, дешифрирующий генератор потока ключей выдает n неправильных битов потока ключей. Следовательно, каждому неправильному биту шифротекста соответствуют n ошибок в открытом тексте, пока испорченный бит не перестанет влиять на внутреннее состояние.

Вопросы безопасности

Самосинхронизирующиеся потоковые шифры также чувствительны к вскрытию повторной передачей. Сначала Мэллори записывает несколько битов шифротекста. Затем, позднее, он вставляет эту запись в текущий трафик. После выдачи некоторой чепухи, пока принимающая сторона синхронизируется с вставленной записью, старый шифротекст будет расшифрован как нормальный. У принимающей стороны нет способа узнать, что полученные данные являются повторно передаваемой записью. Если не используются метки времени, Мэллори может убедить банк снова и снова зачислять деньги на его счет, повторно передавая одно и то же сообщение (конечно, при условии, что ключ не менялся). Другие слабые места этой схемы могут стать заметны при очень частой пересинхронизации [408].

9.6 Режим обратной связи по шифру

Блочный шифр также может быть реализованы как самосинхронизирующийся потоковый шифр, такой режим называется режимом обратной связи по шифру (cipher-feedback, CFB). В режиме CBC шифрование не могло начаться, пока не получен целый блок данных. Это создает проблемы для некоторых сетевых приложений. Например, в безопасной сетевой среде терминал должен иметь возможность передавать главному компьютеру каждый символ сразу, как только он введен. Если данные нужно обрабатывать байтами, режим CBC также не работает.

В режиме CFB единица зашифрованных данных может быть меньше размера блока. В следующем примере каждый раз шифруется только один символ ASCII (это называется 8-битовым шифрованием), но в числе 8 нет ничего волшебного. Вы можете шифровать данные по одному биту с помощью 1-битового CFB, хотя использование для единственного бита полного шифрования блочным шифром потребует много ресурсов, потоковый шифр в этом случае был бы идеей получше. (Уменьшение количества циклов блочного фильтра для повышения скорости не рекомендуется [1269].) Можно также использовать 64-битовый CFB, или любой n-битовый CFB, где n больше или равно размеру блока.

На 0-й показан 8-битовый режим CFB, работающий с 64-битовым алгоритмом. Блочный алгоритм в режиме CFB работает с очередью, размер которой равен размеру используемого блока. Сначала очередь заполнена IV, как и в режиме CBC. Очередь шифруется и для крайних левых восьми битов результата выполняется XOR с первыми 8-битовым символом открытого текста для получения первого 8-битового символа шифротекста. Теперь этот символ передается. Те же восемь битов также передвигаются на место крайних правых восьми битов очереди, а крайними левыми битами становятся следующие восемь битов. Крайние восемь левых битов отбрасывается. Следующий символ открытого текста шифруется тем же способом. Дешифрирование является обратным процессом. И шифрующей, и дешифрирующей стороной блочный алгоритм используется в режиме шифрования.

Если размер блока алгоритма - n, то -битовый CFB выглядит следующим образом (см. -1-й):

$C_i = P_i \oplus E_k(C_{i-1})$

$P_i = C_i \oplus E_k(C_{i-1})$

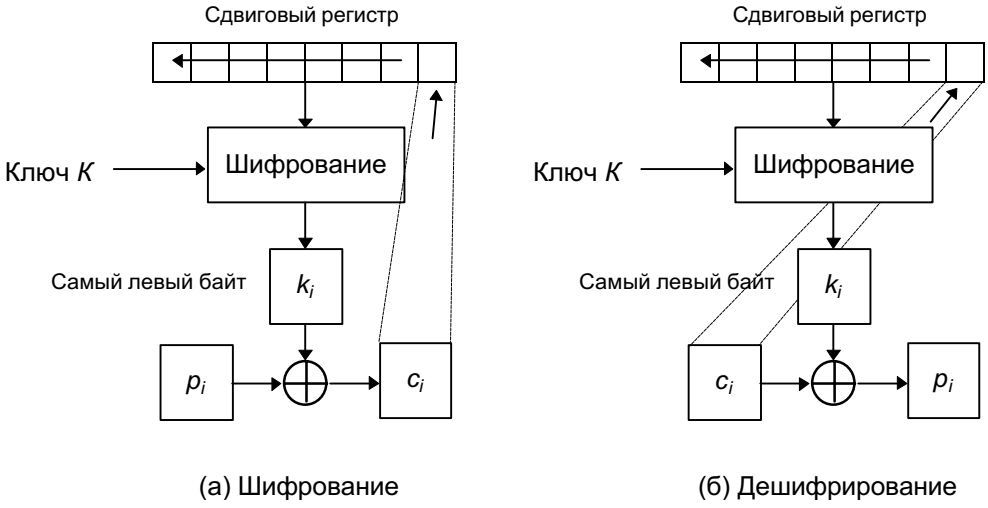


Рис. 9-9. Режим 8-битовой обратной связи по шифру.

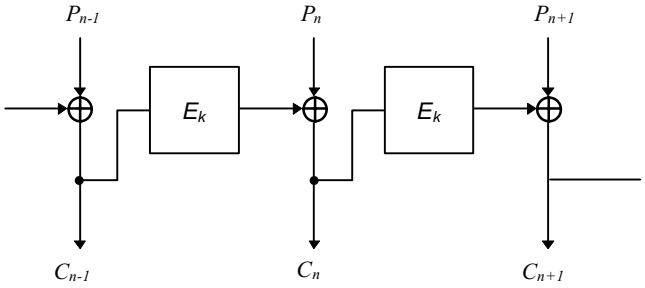


Рис. 9-10. n-битовый CBF с n-битовым алгоритмом.

Как и режим CBC, режим CFB связывает вместе символы открытого текста так, что шифротекст зависит от всего предшествующего открытого текста.

Вектор инициализации

Для инициализации процесса CFB в качестве входного блока алгоритма может использоваться вектор инициализации IV. Как и в режиме CBC IV не нужно хранить в секрете.

Однако IV должен быть уникальным. (В отличие от режима CBC, где IV не обязан быть уникальным, хотя это и желательно.) Если IV в режиме CFB не уникален, криптоаналитик может раскрыть соответствующий открытый текст. IV должен меняться для каждого сообщения. Это может быть последовательный номер, увеличивающийся для каждого нового сообщения и не повторяющийся в течение времени жизни ключа. Если данные шифруются с целью последующего хранения, IV может быть функцией индекса, используемого для поиска данных.

Распространение ошибки

В режиме CFB ошибка в открытом тексте влияет на весь последующий шифротекст, но самоустраняется при дешифрировании. Гораздо интереснее ошибка в шифротексте. Первым эффектом сбоя бита шифротекста является сбой одного бита открытого текста. Затем ошибка попадает в сдвиговый регистр, и пока сбойный бит не выйдет из регистра, будет формироваться неправильный шифротекст. В 8-битовом режиме CFB из-за сбоя единственного бита портятся 9 байтов расшифрованного открытого текста. Потом система восстанавливается, и весь последующий шифротекст расшифровывается правильно. В общем случае в *n*-битовом режиме CFB одна ошибка шифротекста влияет на дешифрирование текущего и следующих *m/n*-l блоков, где *m* - размер блока.

Более тонкой проблемой, связанной с такого рода распространением ошибки, является то, что если Мэллори знает открытый текст сообщения, он может поиграть битами данного блока, заставляя их расшифровываться в нужные ему данные. *Следующий* блок при дешифрировании превратится в чепуху, но вред уже будет причинен. К тому же, он может, оставаясь необнаруженным, менять последние биты сообщения.

CFB самовосстанавливается и после ошибок синхронизации. Ошибка попадает в сдвиговый регистр и, пока она находится там, портит 8 байтов данных. CFB представляет собой пример блочного шифра, который можно использовать как самосинхронизирующийся потоковый шифр (на уровне блоков).

9.7 Синхронные потоковые шифры

В **синхронном потоковом шифре** поток ключей генерируется независимо от потока сообщения. Военные называют этот шифр **ключевым автоключом** (Key Auto-Key, KAK). При шифровании генератор потока ключей один за другим выдает биты потока ключей. При дешифрировании другой генератор потока ключей один за другим выдает идентичные биты потока ключей. Это работает, если оба генератора синхронизированы. Если один из них пропускает один из циклов, или если бит шифротекста теряется при передаче, то после ошибки каждый символ шифротекста будет расшифрован неправильно.

Если такое случается, отправитель и получатель должны повторно синхронизировать свои генераторы потока ключей прежде, чем можно будет продолжить работу. Что еще хуже, они должны выполнить синхронизацию так, чтобы ни одна часть потока ключей не была повторена, поэтому очевидное решение перевести генератор в более раннее состояние не работает.

Положительная сторона синхронных фильтров - это отсутствие распространения ошибок. Если при передаче бит изменит свое значение, что намного вероятнее его потери, то только испорченный бит будет дешифрован неправильно. Все предшествующие и последующие биты не изменятся.

Генератор должен выдавать один и тот же поток ключей и для шифрования, и для дешифрирования, следовательно, выход генератора должен быть предопределен. Если он реализуется на конечном автомате (т.е., компьютере), последовательность со временем повторится. Такие генераторы потока ключей называются **периодическими**. За исключением одноразовых блокнотов все генераторы потока ключей являются периодическими.

Генератор потока ключей должен обладать длинным периодом, намного более длинным, чем количество битов, выдаваемых между сменой ключей. Если период меньше, чем размер открытого текста, то различные части открытого текста будут зашифрованы одинаковым образом, что сильно ослабляет безопасность системы. Если криптоаналитику известна часть открытого текста, он может раскрыть часть потока ключей и использовать ее для дальнейшего раскрытия открытого текста. Даже если у аналитика есть только шифротекст, он может выполнить XOR над разделами, шифрованными одинаковым потоком ключей, и получить XOR соответствующих участков открытого текста. При этом используемый алгоритм превращается в простой алгоритм XOR с очень длинным ключом.

Конкретная длина периода зависит от приложения. Генератор потока ключей, шифрующий непрерывный канал T1, будет шифровать $2^?$ бит в день. Период генератора должен быть на несколько порядков больше этого значения, даже если ключ меняется ежедневно. Если период имеет достаточную длину, ключ можно будет менять раз в неделю или даже раз в месяц.

Синхронные потоковые шифры также предохраняют от любых вставок и удалений шифротекста, так как они приводят к потере синхронизации и будут немедленно обнаружены. Однако, они не защищают полностью от битовых сбоев. Как и при блоковых шифрах в режиме CFB, Мэллори может изменить отдельные биты потока. Если ему известен открытый текст, он может изменить эти биты так, чтобы эти биты дешифрировались так, как ему надо. Дальнейшие биты при дешифрировании превратятся в чепуху (пока система не восстановится), но в определенных приложениях Мэллори может принести заметный ущерб.

Вскрытие вставкой

Синхронные потоковые шифры чувствительны к **вскрытию вставкой** [93]. Пусть Мэллори записал поток шифротекста, но не знает ни открытого текста, ни потока ключей, использованного для шифрования открытого

текста.

Оригинальный открытый текст: pl *p!* p3 *Pi* Оригинальный поток ключей: *kl k! kj ki* Оригинальный шифротекст: *cl c!* c3 *ci*

Мэллори вставляет один известный ему бит, w', в открытый текст после pl и затем пытается получить модифицированный открытый текст, шифрованный тем же потоком ключей. Он записывает получившийся новый шифротекст:

Новый открытый текст: pl *p' pl pi pi* Оригинальный поток: *k. k! k-i ks k!,* Обновленный шифротекст: cl c'z c'3 *c'i* c'i

Так как он знает значение *p'*, он может определить весь открытый текст после этого бита по оригинальному и новому шифротекстам:

k! = *c'z s p',* затем *p!* = *c! s k! kj* = c'3 *S pt,* затем p3 = c3 *S* fc3 *kt* = c', *S* p3, затем p,, = *cs S ks*

Мэллори даже не нужно знать точное положение вставленного бита, он может просто сравнить оригинальный и обновленный шифротексты, чтобы обнаружить, где они начинают отличаться. Для предотвращения такого вскрытия никогда не используйте один поток ключей для шифрования двух различных сообщений.

9.8 Режим выходной обратной связи

Режим выходной обратной связи (Output-feedback, OFB) представляет собой метод использования блочного шифра в качестве синхронного потокового шифра. Этот режим похож на CFB за исключением того, что *n* битов предыдущего выходного блока сдвигаются в крайние правые позиции очереди (см. -2nd). Дешифрирование является обратным процессом. Такой режим называется *n*-битовым OFB. И при шифровании, и при дешифровании блочный алгоритм работает в режиме шифрования. Это иногда называют **внутренней обратной связью**, потому что механизм обратной связи не зависит ни от потоков открытого текста, ни от потоков шифротекста [291]. Если размер блока алгоритма *n*, то *n*-битовый алгоритм OFB выглядит, как показано на :

C, = P, © *S,! S,* = *I, - I,) *P,* = C, © *Sh Si* = *Ek*Si,*

I,)

s - состояние, независящее ни от открытого текста, ни от шифротекста. К числу положительных свойств OFB относится то, что большая часть работы может быть выполнена автономно, даже до того, как появится открытый текст сообщения. Когда наконец сообщение наконец появится, для получения шифротекста над сообщением и выходом алгоритма нужно будет выполнить операцию XOR.

Рис. 9-11. 8-битовый режим

Вектор инициализации

В сдвиговый регистр OFB также сначала должен быть загружен IV. Он должен быть уникальным, но сохранять его в секрете не обязательно.

Распространение ошибки

В режиме OFB распространения ошибки не происходит. Неправильный бит шифротекста приводит к неправильному биту открытого текста. Это может быть полезно при цифровой передаче аналоговых величин, например оцифрованного звука или видеоизображения, когда случайный сбой бита допустим, но распространение ошибки нежелательно.

С другой стороны, потеря синхронизации смертельна. Если сдвиговые регистры при шифровании и при дешифрировании отличаются, то восстановленный открытый текст представляет собой бессмыслицу. Любая система, использующая режим OFB, должна включать механизм обнаружения потери синхронизации и механизм заполнения обоих сдвиговых регистров новым (или одинаковым) IV для восстановления синхронизации.

Рис. 9-12. n-битовый OFB с n-битовым алгоритмом.

OFB и проблемы безопасности

Анализ режима OFB [588, 430, 431, 789] показывает, что OFB стоит использовать только, когда размер обратной связи совпадает с размером блока. Например, 64-битовый алгоритм нужно использовать только в 64-битовом режиме OFB. Несмотря на то, что правительство США разрешает для DES и другие размеры обратных

связей DES [1143], избегайте их.

Режим OFB выполняет XOR над потоком ключей и текстом. Этот поток ключей со временем повторяется. Важно, чтобы он не повторялся для того же ключа, в противном случае нарушается безопасность . Когда размер обратной связи равен размеру блока, блочный шифр переставляет *m*-битовые значения (где *m* - это размер блока), и средняя длина цикла составляет 2™ -1. При длине блока 64 бита это очень большое число. Когда размер обратной связи *n* меньше длины блока, средняя длина цикла падает до приблизительно 2'"*. Для 64-битного шифра это только * - что явно недостаточно.

Потоковые шифры в режиме OFB

Потоковые шифры также могут работать в режиме OFB. В этом случае ключ влияет на функцию следующего состояния (см. -4-й). Функция выхода не зависит от ключа, очень часто она является чем-то простым, напр имер, одним битом внутреннего состояния или результатом XOR нескольких битов внутреннего состояния. Криптографически сложной является функция следующего состояния, которая зависит от ключа . Этот метод также называется внутренней обратной связью [291], потому что механизм обратной связи является вложенным по отношению к алгоритму генерации ключей.

Рис. 9-13. Генератор потока ключей в режиме с выходной обратной связью.

В одном из вариантов этого режима ключ определяет только начальное состояние генератора потока ключей . После того, как ключ определит внутреннее состояние генератора, генератор работает, не подвергаясь воздейс твиям извне.

9.9 Режим счетчика

Блочные шифры в **режиме счетчика** используют в качестве входов алгоритма последовательные номера [824, 498, 715]. Для заполнения регистра используется счетчик, а не выход алгоритма шифрования . После шифрования каждого блока счетчик инкрементируется на определенную константу, обычно единицу . Для этого режима свойства синхронизации и распространения ошибки такие же, как и для OFB. Режим счетчика решает проблему *n*-битового выхода режима OFB, где *n* меньше длины блока.

К счетчику не предъявляется никаких особых требований, он не должен проходить по порядку все возмо жные значения. В качестве входа блочного алгоритма можно использовать генераторы случайных чисел, описа нные в главах 16 и 17, независимо от того, являются ли они криптографически безопасными или нет .

Потоковые шифры в режиме счетчика

У потоковых шифров в режиме счетчика простые функции следующего состояния и сложные функции вых ода, зависящие от ключа. Этот метод, показанный на -5-й, был предложен в [498, 715]. Функция следующего состояния может быть чем-то простым, например, счетчиком, добавляющим единицу к предыдущему состо янию.

Рис. 9-14. Генератор потока ключей в режиме счетчика.

Потоковый шифр в режиме счетчика может генерировать *i*-ый бит, k_i, без выдачи всех предшествующих ключевых битов. Просто установите счетчик вручную в *i*-ое внутреннее состояние и генерируйте бит. Это полезно для закрытия файлов данных с произвольным доступом, можно расшифровать конкретный блок данных не расшифровывая целый файл.

9.10 Другие режимы блочных шифров

Режим сцепления блоков

Для использования блочного алгоритма в режиме **сцепления блоков** (block chaining, BC), просто выполните XOR входа блочного шифра и результата XOR всех предыдущих блоков шифротекста. Как и для CBC используется IV. Математически это выглядит как:

C, = *Ek(P, Q F*; F,* I = *F, © C, P, = F, ©* *(C,); *Fi** I = *F,*

© Ci

Как и CBC, обратная связь процесса BC приводит к распространению ошибки в открытом тексте. Главная

проблема ВС заключается в том, что из-за того, что дешифрирование блока шифротекста зависит от всех пр едыдущих блоков шифротекста, единственная ошибка шифротекста приведет к неправильной расшифровке всех последующих блоков шифротекста.

Режим распространяющегося сцепления блоков шифра

Режим **распространяющегося сцепления блоков шифра** (propagating cipher block chaining, PCBC) [1080] похож на режим CBC за исключением того, что и предыдущий блок открытого текста, и предыдущий блок шифротекста подвергаются операции XOR с текущим блоком открытого текста перед шифрованием (или после шифрования) (см. -6-й).

$$Ci = E*P, \odot Ci \text{ I} \odot P, \text{ I)} \quad P* = Cj \text{ I} \odot Pi \text{ I} \odot \text{a*,)}$$

PCBC используется в Kerberos версии 4 (см. раздел 24.5) для выполнения за один проход и шифрования, и проверки целостности. В режиме PCBC ошибка шифротекста приводит к неправильному дешифрированию всех последующих блоков. Это означает, что проверка стандартного блока в конце сообщения обеспечивает целос тность всего сообщения.

Рис. 9-15. Режим распространяющегося сцепления блоков шифра.

К несчастью в этом режиме существует одна проблема [875]. Перестановка двух блоков шифротекста прив одит к неправильной расшифровке двух соответствующих блоков открытого текста, но из-за природы операции XOR над открытым текстом и шифротекстом, дальнейшие ошибки компенсируются. Поэтому, если при провер ке целостности проверяются только несколько последних блоков расшифрованного открытого текста, можно получить частично испорченное сообщение. Хотя никто до сих пор не додумался, как воспользоваться этой сл абостью, Kerberos версии 5 после обнаружения ошибки переключается в режим CBC.

Сцепление блоков шифра с контрольной суммой

Сцепление блоков шифра с контрольной суммой (cipher block chaining with checksum, CBCC) представ ляет собой вариант CBC [1618]. Сохраняйте значение XOR всех уже зашифрованных блоков открытого текста, выполняя для каждого текущего блока открытого текста перед его шифрованием XOR с сохраняемым значен ием. CBCC обеспечивает, что любое изменение любого блока шифротекста изменит результат дешифровки п оследнего блока. Если последний блок содержит какую-нибудь константу или служит для проверки целостности, то целостность расшифрованного открытого текста может быть проверена с минимальными дополнительными накладными расходами.

Выходная обратная связь с нелинейной функцией

Выходная обратная связь с нелинейной функцией (output feedback with a nonlinear function, OFBNLF) [777] представляет собой вариант и OFB, и ECB, где ключ изменяется с каждым блоком:

$$\text{C, } = Ek*P*, \; K* = Edit, \; \text{,1 } P, = \text{a*,}); \; Ki = E*K, \text{ I)}$$

Ошибка одного бита шифротекста распространяется только на один блок открытого текста. Однако, если бит теряется или добавляется, то ошибка распространяется до бесконечности. С блочным алгоритмом, использу ющим сложный алгоритм планирования ключей, этот режим работает медленно. Я не знаю, как выполнять кри птоанализ этого режима.

Прочие режимы

Возможны и другие режимы, хотя они используются нечасто. Сцепление блоков открытого текста (plaintext block chaining, PBC) похоже на CBC за исключением того, что операция XOR выполняется для с блока откр ытого текста и для предыдущего блока открытого текста, а не блока шифротекста. Обратная связь по открытому тексту (plaintext feedback, PFB) похожа на CFB за исключением того, что для обратной связи используется не шифротекст, а открытый текст. Существует также сцепление блоков шифротекста по различиям открытого те кста (cipher block chaining of plaintext difference, CBCPD). Я уверен, что можно найти еще таинственнее.

Если у криптоаналитика есть машина для поиска ключей грубой силой, то он сможет раскрыть ключ, если угадает один из блоков открытого текста. Некоторые из упомянутых странных режимов, по сути, являются д ополнительным шифрованием перед использованием алгоритма шифрования: например, XOR текста и фиксиро ванной секретной строки или перестановка текста. Почти все отклонения от стандартов помешают подобному криптоанализу.

9.11 Выбор режима шифра

Если вашей основной заботой являются скорость и простота, то ECB является самым простым и самым быстрым способом использовать блочный шифр. Помимо уязвимости к вскрытию повтором, алгоритм в режиме ECB проще всего криптоанализировать. Я не советую использовать ECB для шифрования сообщений.

ECB хорошо использовать для шифрования случайных данных, например, других ключей. Так как данные невелики по размеру и случайны, недостатки ECB не существенны для такого применения.

Для обычного открытого текста используйте CBC, CFB или OFB. Конкретный режим зависит от ваших требований. В приведены безопасность и эффективность различных режимов.

Для шифрования файлов лучше всего подходит CBC. Значительно увеличивается безопасность, и при появлении ошибок в хранимых данных почти никогда не бывает сбоев синхронизации. Если ваше приложение - программное, то CBC почти всегда будет лучшим выбором.

Табл. 9-1.
Краткий обзор режимов работы блочных шифров

ECB:

Security:

-Plaintext patterns are not concealed.

- Input to the block cipher Is not randomlzed; It Is the same as the plaintext. + More than one message can be encrypted with the same

- plaintext Is easy to manipulate; blocks can be removed, repeated, or Interchanged.

Efficiency: + Speed is the same as the block cipher.

- Clphertext Is up to one block longer than the plaintext, due to padding.

- No preprocessing is possible. *Processing is paraUelizable.

Fault-tolerance:

-A ciphertext error affects one full block of plaintext.

- Synchronization error is unrecoverable.

CFB:

Security:

+ Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key, provided that a different IV is used. +/- Plaintext is somewhat difficult to manipulate; blocks call be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes.

Efficiency: + Speed is the same as the block cipher.

- Ciphertext is the same size as the plaintext, not counting the IV.

+/- Encryption is not paraUelizable; decryption is paral- Idizable and has a random-access property.

- Some preprocessing is possible before a block is seen; the Previous ciphertext block can be encrypted. +/- Encryption is not parallelizable; decry p-tion is paral- felizable and has a random-access property.

F'auh-toterance:

-A ciphertext error affects the corresponding bit of plaintext and the next full block.

+ Synchronization errors of full block sizes are recoverable. I. -bit CFB can recover from the addition or loss of single bits.

cbc:

Security:

+ Plaintext patterns are concealed by XORing with previous ciphertext block.

+ Input to the block cipher is randomized by XORing with the previous ciphertext block.

+ More than one message can be encrypted with the same key.

+/- Plaintext is somewhat difficult to manipulate; blocks can be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes.

Efficiency: + Speed is the same as the block cipher.

- Ciphertext is up to one block longer than the plaintext, not counting the IV.

- No preprocessing is possible.

+/- Encryption is not paraUelizable; decryption is paral- lelizable and has a random-access property.

Wau*-toterance:

- A ciphertext error affects one full block of plaintext and the corresponding bit in the next block.

- Synchronization error is unrecoverable.

OFB/Counter:

Security;

+ Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key, provided that a different IV is used. - Plaintext is very easy to manipulate; any change in ciphertext directly affects the plaintext.

C*lclency: + Speed is the same as the block cipher.

- Ciphertext is the same size as the plaintext, not counting the IV. + Processing is possible before the message is seen.

-/+ OFB processing is not paraUelizable; counter processing is paraUelizable.

Fau*t-tolerance:

+ A ciphertext error affects only the corresponding bit of plaintext. - Synchronization error is unrecoverable.

CFB-specifically 8-bit CFB-is generally the mode *ol* choice for encrypting streams of characters when each cha r-acter has to be treated individually, as in a link between a terminal and a host. OFB is most often used in high-speed synchronous systems where error propagation is intolerable. OFB is also the mode of choice if preprocessing is r e-quired.

OFB is the mode of choice in a error-prone environment, because it has no error extension.

Stay away from the weird modes. One of the four basic modes-ECB, CBC, OFB, and CFB-is suitable for almost any application. These modes are not overly complex and probably do not reduce the security of the system. While it is possible that a complicated mode might increase the security of a system, most likely it just increases the complexity. None of the weird modes has any better error propagation or error recovery characteristics.

9.12 INTERLEAVING

With most modes, encryption of a bit (or block) depends on the encryption of the previous bits (or blocks). This can often make it impossible to parallelize encryption. For example, consider a hardware box that does encryption in CBC mode. Even if the box contains four encryption chips, only one can work at any time. The next chip needs the results of the previous chip before it starts working.

The solution is to **interleave** multiple encryption streams. (This is not multiple encryption; that's covered in Se c-tions 15.1 and 15.2). Instead of a single CBC chain, use four. The first, fifth, and every fourth block thereafter are e n-crypted in CBC mode with one IV. The second, sixth, and every fourth block thereafter are encrypted in CBC mode with another IV, and so on. The total IV is much longer than it would have been without interleaving.

Think of it as encrypting four different messages with the same key and four different IVs. These messages are all i nterleaved.

This trick can also be used to increase the overall speed of hardware encryption. If you have three encryption chips, each c a-pable of encrypting data at 33 megabits/second, you can interleave them to encrypt a single 100 megabit/second data channel.

Figure 9.16 shows three parallel streams interleaved in CFB mode. The idea can also work in CBC and OFB modes, and with any number of parallel streams. Just remember that each stream needs its own IV. Don't share.

9.13 BLOCK CIPHERS VERSUS STREAM CIPHERS

Although block and stream ciphers are very different, block ciphers can be implemented as stream ciphers and stream ciphers can be implemented as block ciphers. The best definition of the difference I've found is from Ranier Rueppel [1362.]:

> Block ciphers operate on data with a fixed transformation on large blocks of plaintext data; stream ciphers ope r-ate with a time-varying transformation on individual plaintext digits.

Figure 9.16 Interleavingthtee CFB encryptions.

In the real world, block ciphers seem to be more general (i.e., they can be used in any of the four modes) and stream ciphers seem to be easier to analyze mathematically. There is a large body of theoretical work on the analysis and design of stream c i-phers-most of it done in Europe, for some reason. They have been used by the world's militaries since the invention of electronics. This seems to be changing; recently a whole slew of theoretical papers have been written on block cipher design. Maybe soon there will be a theory of block cipher design as rich as our current theory of stream cipher d esign.

Otherwise, the differences between stream ciphers and block ciphers are in the implementation. Stream ciphers that only e n-crypt and decrypt data one bit at a time are not really suitable for software implementation. Block ciphers can be easier to impl e-ment in software, because they often avoid time-consuming bit manipulations and they operate on data in computer-sized blocks. On the other hand, stream ciphers can be more suitable for hardware implementation because they can be implemented very eff i-ciently in silicon.

These are important considerations. It makes sense for a hardware encryption device on a digital communications channel to encrypt the individual bits as they go by. This is what the device sees. On the other hand, it makes no sense for a software encry p-tion device to encrypt each individual bit separately. There are some specific instances where bit- and byte-wise encryption might be necessary in a computer system-encrypting the link between the keyboard and the CPU, for example-but generally the encry p-tion block should be at least the width of the data bus.

Глава 10 Using Algorithms

Think of security - data security, communications security, information security, whatever - as a chain. The security of the entire system is only as strong as the weakest link. Everything has to be secure: cryptographic algorithms, protocols, key manag e-ment, and more. If your algorithms are great but your random-number generator stinks, any smart cryptanalyst is going to attack your system through the random-number generation. If you patch that hole but forget to securely erase a memory location that contains the key, a cryptanalyst will break your system via that route. If you do everything right and accidentally e-mail a copy of your secure files to The Wall Street Journal, you might as well not have bothered.

It's not fair. As the designer of a secure system, you have to think of every possible means of attack and protect against them all, but a cryptanalyst only has to find one hole in your security and exploit it.

Cryptography is only a part of security, and often a very small part. It is the mathematics of making a system secure, which is different from actually making a system secure. Cryptography has its "size queens": people who spend so much time arguing about how long a key should be that they forget about everything else. If the secret police want to know what is on your computer, it is far easier for them to break into your house and install a camera that can record what is on your computer screen than it is for them to cryptanalyze your hard drive.

Additionally, the traditional view of computer cryptography as "spy versus spy" technology is becoming increasingly ina p-propriate. Over 99 percent of the cryptography used in the world is not protecting military secrets; it's in applications such as bank cards, pay-TV, road tolls, office building and computer access tokens, lottery terminals, and prepayment electricity meters [43,44]. In these applications, the role of cryptography is to make petty crime slightly more difficult; the paradigm of the well-funded a d-versary with a rabbit warren of cryptanalysts and roomsful of computers just doesn't apply.

Most of those applications have used lousy cryptography, but successful attacks against them had nothing to do with cry p-tanalysts. They involved crooked employees, clever sting operations, stupid implementations, integration blunders, and random idiocies. (I strongly recommend Ross Anderson's paper, "Why Cryptosytems Fail" [44]; it should be required reading for anyone involved in this field.) Even the NSA has admitted that most security failures in its area of interest are due to failures in impl e-mentation, and not failures in algorithms or protocols [1119]. In these instances it didn't matter how good the cryptography was; the successful attacks bypassed it completely.

10.1 CHOOSING AN ALGORITHM

When it comes to evaluating and choosing algorithms, people have several alternatives:

- They can choose a published algorithm, based on the belief that a published algorithm has been scrutinized by many cry p-tographers; if no one has broken the algorithm yet, then it must be pretty good.

- They can trust a manufacturer, based on the belief that a well-known manufacturer has a reputation to uphold and is u n-likely to risk that reputation by selling equipment or programs with inferior algorithms.

- They can trust a private consultant, based on the belief that an impartial consultant is best equipped to make a reliable evaluation of different algorithms.

- They can trust the government, based on the belief that the government is trustworthy and wouldn't steer its citizens wrong.

- They can write their own algorithms, based on the belief that their cryptographic ability is second-to-none and that they should trust nobody but themselves.

Any of these alternatives is problematic, but the first seems to be the most sensible. Putting your trust in a single manufa c-turer, consultant, or government is asking for trouble. Most people who call themselves security consultants (even those from big-name firms usually don't know anything about encryption. Most security product manufacturers are no better. The NSA has some of the world's best cryptographers working for it, but they're not telling all they know. They have their own interests to further which are not congruent with those of their citizens. And even if you're a genius, writing your own algorithm and then using it without any peer review is just plain foolish.

The algorithms in this book are public. Most have appeared in the open literature and many have been cryptanalyzed by e x-perts in the field. I list all published results, both positive and negative. I don't have access to the cryptanalysts done by any of the myriad military security organizations in the world Which are probably better than the academic institutions,(they've been doing it longer and are better funded), so it is possible that these algorithms are easier to break than it appears. Even so, it is far more likely that they are more secure than an algorithm designed and implemented in secret in some corporate basement.

The hole in all this reasoning is that we don't know the abilities of the various military cryptanalysts organizations.

What algorithms can the NSA break? For the majority of us, there's really no way of knowing. If you are arrested with a DES-encrypted computer hard drive, the FBI is unlikely to introduce the decrypted plaintext at your trial; the fact that they can break an algorithm is often a bigger secret than any information that is recovered. During WWII, the Allies were forbidden from using decrypted German Ultra traffic unless they could have plausibly gotten the information elsewhere. The only way to get the NSA to admit to the ability to break a given algorithm is to encrypt something so valuable that its public dissemination is worth the admission. Or, better yet, create a really funny joke and send it via encrypted e-mail to shady characters in shadowy countries. NSA employees are people, too; I doubt even they can keep a good joke secret.

A good working assumption is that the NSA can read any message that it chooses, but that it cannot read all messages that it chooses. The NSA is limited by resources, and has to pick and choose among its various targets. Another good assumption is that they prefer breaking knuckles to breaking codes; this preference is so strong that they will only resort to breaking codes when they wish to preserve the secret that they have read the message. In any case, the best most of us can do is to choose among public a l-gorithms that have withstood a reasonable amount of public scrutiny and cryptanalysts. Algorithms for Export

Algorithms for export out of the United States must be approved by the U.S. government (actually, by the NSA (see Section 25.1). It is widely believed that these export-approved algorithms can be broken by the NSA. Although no one has admitted this on the record, these are some of the things the NSA is rumored to privately suggest to companies wishing to export their crypt o-graphic products:

- Leak a key bit once in a while, embedded in the ciphertext.

- "Dumb down" the effective key to something in the 30-bit range. For example, while the algorithm might accept a 100-bit key, most of those keys might be equivalent.

- Use a fixed IV, or encrypt a fixed header at the beginning of each encrypted message. This facilitates a known-plaintext attack.

- Generate a few random bytes, encrypt them with the key, and then put both the plaintext and the ciphertext of those ra n-dom bytes at the beginning of the encrypted message. This also facilitates a known- plaintext attack.

NSA gets a copy of the source code, but the algorithm's details remain secret from everyone else. Certainly no one adve r-tises any of these deliberate weaknesses, but beware if you buy a U.S. encryption product that has been approved for export.

10.2 PUBLIC-KEY CRYPTOGRAPHY VERSUS SYMMETRIC CRYPTOGRAPHY

Which is better, public-key cryptography or symmetric cryptography? This question doesn't make any sense, but has been d e-bated since public-key cryptography was invented. The debate assumes that the two types of cryptography can be compared on an equal footing. They can't.

Needham and Schroeder [1159] pointed out that the number and length of messages are far greater with public-key alg o-rithms than with symmetric algorithms. Their conclusion was that the symmetric algorithm was more efficient than the public-key algorithm. While true, this analysis overlooks the significant security benefits of public-key cryptography. Whitfield Diffie writes 492,494]:

In viewing public-key cryptography as a new form of cryptosystem rather than a new form of key management, I set the stage for criticism on grounds of both security and performance. Opponents were quick to point out that the RSA system ran about one-thousandth as fast as DES and required keys about ten times as large. Although it had been obvious from the beginning that the use of public key systems could be limited to exchanging keys for conventional [symmetric] cryptography, it was not immediately clear that this was necessary. In this context, the proposal to build hybrid systems [879] was hailed as a discovery in its own right.

Public-key cryptography and symmetric cryptography are different sorts of animals; they solve different sorts of problems. Symmetric cryptography is best for encrypting data. It is orders of magnitude faster and is not susceptible to chosen-ciphertext a t-tacks. Public-key cryptography can do things that symmetric cryptography can't; it is best for key management and a myriad of protocols discussed in Part I.

Other primitives were discussed in Part I: one-way hash functions, message authentication codes, and so on. Table 10.1 lists different types of algorithms and their properties [804].

10.3 ENCRYPTING COMMUN1CAT10NS CHANNELS

This is the classic Alice and Bob problem: Alice wants to send Bob a secure message. What does she do? She encrypts the message.

In theory, this encryption can take place at any layer in the OSI (Open Systems Interconnect) communications model. (See the OSI security architecture standard for more information [305].) In practice, it takes place either at the lowest layers (one and two) or at higher layers. If it takes place at the lowest layers, it is called link-by-link encryption; everything going through a particular data link is encrypted. If it takes place at higher layers, it is called end-to-end encryption; the data are encrypted selectively and stay encrypted until they are decrypted by the intended final recipient. Each approach has its own benefits and drawbacks.

Link-by Link Encryption

The easiest place to add encryption is at the physical layer (see Figure 10. 1). This is called link-by-link encryption. The interfaces to the physical layer are generally standardized and it is easy to connect hardware encryption devices at this point. These devices encrypt all data passing through them, including data, routing information, and protocol information. They can be used on any type of digital communication link. On the other hand, any intelligent switching or storing nodes between the sender and the receiver need to decrypt the data stream before processing it.

This type of encryption is very effective. Because everything is encrypted, a crypt- analyst can get no information about the structure of the information. He has no idea who is talking to whom, how long the messages they are sending are, what times of day they communicate, and so on. This is called traffic-flow security: the enemy is not only denied access to the information, but also access to the knowledge of where and how much information is flowing.

Security does not depend on any traffic management techniques. Key management is also simple; only the two endpoints of the line need a common key, and they can change their key independently from the rest of the network.

Imagine a synchronous communications line, encrypted using 1-bit CFB. After initialization, the line can run indefinitely, re-

covering automatically from bit or synchronization errors. The line encrypts whenever messages are sent from one end to the other; otherwise it just encrypts and decrypts random data. Eve has no idea when messages are being sent and when they are not; she has no idea when messages begin and end. All she sees is an endless stream of random-looking bits.

If the communications line is asynchronous, the same 1-bit CFB mode can be used. The difference is that the adversary can get information about the rate of transmission. If this information must be concealed, make some provision for passing dummy messages during idle times.

The biggest problem with encryption at the physical layer is that each physical link in the network needs to be encrypted: Leaving any link unencrypted jeopardizes the security of the entire network. If the network is large, the cost may quickly become prohibitive for this kind of encryption.

Additionally, every node in the network must be protected, since it processes unencrypted data. If all the network's users trust one another, and all nodes are in secure locations, this may be tolerable. But this is unlikely. Even in a single corporation, information might have to be kept secret within a department. If the network accidentally misroutes information, anyone can read it. Table 10.2 summarizes the pros and cons of link-by-link encryption.

End-to-End Encryption

Another approach is to put encryption equipment between the network layer and the transport layer. The encryption device must understand the data according to the protocols up to layer three and encrypt only the transport data units, which are then r e-combined with the unencrypted routing information and sent to lower layers for transmission.

This approach avoids the encryption/decryption problem at the physical layer. By providing end-to-end encryption, the data remains encrypted until it reaches its final destination (see Figure 10.2). The primary problem with end-to-end encryption is that the routing information for the data is not encrypted; a good cryptanalyst can learn much from who is talking to whom, at what times and for how long, without ever knowing the contents of those conversations. Key management is also more difficult, since individual users must make sure they have common keys.

Building end-to-end encryption equipment is difficult. Each particular communications system has its own protocols. Som e-times the interfaces between the levels are not well-defined, making the task even more difficult.

If encryption takes place at a high layer of the communications architecture, like the applications layer or the presentation layer, then it can be independent of the type of communication network used. It is still end-to-end encryption, but the encryption implementation does not have to bother about line codes, synchronization between modems, physical interfaces, and so forth. In the early days of electro- mechanical cryptography, encryption and decryption took place entirely offline; this is only one step r e-moved from that.

Encryption at these high layers interacts with the user software. This software is different for different computer archite c-tures, and so the encryption must be optimized for different computer systems. Encryption can occur in the software itself or in specialized hardware. In the latter case, the computer will send the data to the specialized hardware for encryption before sending it to lower layers of the communication architecture for transmission. This process requires some intelligence and is not suitable for dumb terminals. Additionally, there may be compatibility problems with different types of computers. The major disadvantage of end-to-end encryption is that it allows traffic analysis. Traffic analysis is the analysis of encrypted messages: where they come from, where they go to, how long they are, when they are sent, how frequent or infrequent they are, whether they coincide with outside events like meetings, and more. A lot of good information is buried in that data, and a cryptanalyst will want to get his hands on it. Table 10.3 presents the positive and negative aspects of end-to-end encryption.

Combining the Two

Table 10.4, primarily from [1244], compares link-by-link and end-to-end encryption. Combining the two, while most expe n-sive, is the most effective way of securing a network. Encryption of each physical link makes any analysis of the routing inform a-tion impossible, while end-to-end encryption reduces the threat of unencrypted data at the various nodes in the network. Key ma n-agement for the two schemes can be completely separate: The network managers can take care of encryption at the physical level, while the individual users have responsibility for end-to-end encryption.

10.4 ENCRYPTING DATA FOR STORAGE

Encrypting data for storage and later retrieval can also be thought of in the Alice and Bob model. Alice is still sending a me s-sage to Bob, but in this case "Bob" is Alice at some future time. However, the problem is fundamentally different. In communic a-tions channels, messages in transit have no intrinsic value. If Bob doesn't receive a particular message, Alice can always resend it. This is not true for data encrypted for storage. If Alice can't decrypt her message, she can't go back in time and re-encrypt it. She has lost it forever. This means that encryption applications for data storage should have some mechanisms to prevent unrecove r-able errors from creeping into the ciphertext. The encryption key has the same value as the message, only it is smaller. In effect, cryptography converts large secrets into smaller ones. Being smaller, they can be easily lost. Key management procedures should assume that the same keys will be used again and again, and that data may sit on a disk for years before being decrypted. Fu r-thermore, the keys will be around for a long time. A key used on a communications link should, ideally, exist only for the length of the communication. A key used for data storage might be needed for years, and hence must be stored securely for years.

Other problems particular to encrypting computer data for storage were listed in [357]:

- The data may also exist in plaintext form, either on another disk, in another computer, or on paper. There is much more opportunity for a cryptanalyst to perform a known-plaintext attack.

- In database applications, pieces of data may be smaller than the block size of most algorithms. This will cause the ciphe r-text to be considerably larger than the plaintext.

- The speed of I/O devices demands fast encryption and decryption, and will probably require encryption hardware. In some applications, special high-speed algorithms may be required.

- Safe, long-term storage for keys is required.

- Key management is much more complicated, since different people need access to different files, different portions of the same file, and so forth. If the encrypted files are not structured as records and fields, such as text files, retrieval is easier: The entire file is decrypted before use. If the encrypted files are database files, this solution is problematic. Decrypting the entire dat a-base to access a single record is inefficient, but encrypting records independently might be susceptible to a block-replay kind of attack. In addition, you must make sure the unencrypted file is erased after encryption (see Section 10.9). For further details and insights, consult [425,569].

Dereferencing Keys

When encrypting a large hard drive, you have two options. You can encrypt all the data using a single key. This gives a cryptanalyst a large amount of ciphertext to analyze and makes it impossible to allow multiple users to see only parts of the drive. Or, you can encrypt each file with a different key, forcing users to memorize a different key for each file.

The solution is to encrypt each file with a separate key, and to encrypt the keys with another key known by the users. Each user only has to remember that one key. Different users can have different subsets of the file-encryption keys encrypted with their key. And there can even be a master key under which every file-encryption key is encrypted. This is even more secure because the file-encryption keys are random and less susceptible to a dictionary attack.

Driver-Level vs. File-Level Encryption

There are two ways to encrypt a hard drive: at the file level and at the driver level. Encryption at the file level means that every file is encrypted separately. To use a file that's been encrypted, you must first decrypt the file, then use it, and then re- e n-crypt it.

Driver-level encryption maintains a logical drive on the user's machine that has all data on it encrypted. If done well, this can provide security that, beyond choosing good passwords, requires little worry on the part of the user. The driver must be consider a-bly more complex than a simple file-encryption program, however, because it must deal with the issues of being an installed d e-vice driver, allocation of new sectors to files, recycling of old sectors from files, random-access read and update requests for any data on the logical disk, and so on.

Typically, the driver prompts the user for a password before starting up. This is used to generate the master decryption key, which may then be used to decrypt actual decryption keys used on different data.

Providing Random Access to an Encrypted Drive

Most systems expect to be able to access individual disk sectors randomly. This adds some complication for using many stream ciphers and block ciphers in any chaining mode. Several solutions are possible.

Use the sector address to generate a unique IV for each sector being encrypted or decrypted. The drawback is that each se c-tor will always be encrypted with the same IV. Make sure this is not a security problem.

For the master key, generate a pseudo-random block as large as one sector. You can do this by running an algorithm in OFB mode, for example.) To encrypt any sec- tor, first XOR in this pseudo-random block, then encrypt normally with a block cipher in ECB mode. This is called ECB+OFB (see Section 15.4).

Since CBC and CFB are error-recovering modes, you can use all but the first block or two in the sector to generate the IV for that sector. For example, the IV for sector 3001 may be the hash of the all but the first 128 bits of the sector's data. After genera t-ing the IV, encrypt normally in CBC mode. To decrypt the sector, you use the second 64-bit block of the sector as an IV, and d e-crypt the remainder of the sector. Then, using the decrypted data, you regenerate the IV and decrypt the first 128 bits.

You can use a block cipher with a large enough block size that it can encrypt the whole sector at once. Crab See Section 14.6) is an example.

10.5 HARDWARE ENCRYPTION VERSUS SOFTWARE ENCRYPTION

Hardware

Until very recently, all encryption products were in the form of specialized hardware. These encryption/decryption boxes plugged into a communications line and encrypted all the data going across that line. Although software encryption is becoming more prevalent today, hardware is still the embodiment of choice for military and serious commercial applications. The NSA, for example, only authorizes encryption in hardware. There are several reasons why this is so.

The first is speed. As we will see in Part III, encryption algorithms consist of many complicated operations on plaintext bits. These are not the sorts of operations that are built into your run-of-the-mill computer. The two most common encryption algorithms, DES and RSA, run inefficiently on general-purpose processors. While some cryptographers have tried to make their algorithms more suitable for software implementation, specialized hardware will always win a speed race.

Additionally, encryption is often a computation-intensive task. Tying up the computer's primary processor for this is inefficient. Moving encryption to another chip, even if that chip is just another processor, makes the whole system faster. The second reason is security. An encryption algorithm running on a generalized computer has no physical protection. Mallory can go in with various debugging tools and surreptitiously modify the algorithm without anyone ever realizing it. Hardware encryption devices can be securely encapsulated to prevent this. Tamper- proof boxes can prevent someone from modifying a hardware encryption device. Special-purpose VLSI chips can be coated with a chemical such that any attempt to access their interior will result in the destruction of the chip's logic. The U.S. government's Clipper and Capstone chips See Sections 24.16 and 24.171 are designed to be tamperproof. The chips can be designed so that it is impossible for Mallory to read the unencrypted key.

IBM developed a cryptographic system for encrypting data and communications on mainframe computers [515,1027]. It includes tamper-resistant modules to hold keys. This system is discussed in Section 24.1.

Electromagnetic radiation can sometimes reveal what is going on inside a piece of electronic equipment. Dedicated encryption boxes can be shielded, so that they leak no compromising information. General-purpose computers can be shielded as well, but it is a far more complex problem. The U.S. military calls this TEMPEST; it's a subject well beyond the scope of this book.

The final reason for the prevalence of hardware is the ease of installation. Most encryption applications don't involve general-purpose computers. People may wish to encrypt their telephone conversations, facsimile transmissions, or data links. It is cheaper to put special-purpose encryption hardware in the telephones, facsimile machines, and modems than it is to put in a microprocessor and software.

Even when the encrypted data comes from a computer, it is easier to install a dedicated hardware encryption device than it is to modify the computer's system software. Encryption should be invisible; it should not hamper the user. The only way to do this in software is to write encryption deep into the operating system. This isn't easy. On the other hand, even a computer neophyte can plug an encryption box between his computer and his external modem.

The three basic kinds of encryption hardware on the market today are: self-contained encryption modules (that perform functions such as password verification and key management for banks), dedicated encryption boxes for communications links, and boards that plug into personal computers.

Some encryption boxes are designed for certain types of communications links, such as T-1 encryption boxes that are designed not to encrypt synchronization bits. There are different boxes for synchronous and asynchronous communications lines. Newer boxes tend to accept higher bit rates and are more versatile.

Even so, many of these devices have some incompatibilities. Buyers should be aware of this and be well-versed in their par-

ticular needs, lest they find themselves the owners of encryption equipment unable to perform the task at hand. Pay attention to restrictions in hardware type, operating system, applications software, net- work, and so forth. PC-board encryptors usually e n-crypt everything written to the hard disk and can be configured to encrypt everything sent to the floppy disk and serial port as well. These boards are not shielded against electromagnetic radiation or physical interference, since there would be no benefit in pr o-tecting the boards if the computer remained unaffected. More companies are starting to put encryption hardware into their co m-munications equipment. Secure telephones, facsimile machines, and modems are all available. Internal key management for these devices is generally secure, although there are as many different schemes as there are equipment vendors. Some schemes are more suited for one situation than another, and buyers should know what kind of key management is incorporated into the encryption box and what they are expected to provide themselves.

Software

Any encryption algorithm can be implemented in software. The disadvantages are in speed, cost, and ease of modification (or manipulation). The advantages are in flexibility and portability, ease of use, and ease of upgrade. The algorithms written in C at the end of this book can be implemented, with little modification, on any computer. They can be inexpensively copied and i n-stalled on many machines. They can be incorporated into larger applications, such as communications programs or word proce s-sors.

Software encryption programs are popular and are available for all major operating systems. These are meant to protect i n-dividual files; the user generally has to manually encrypt and decrypt specific files. It is important that the key management scheme be secure: The keys should not be stored on disk anywhere (or even written to a place in memory from where the processor swaps out to disk). Keys and unencrypted files should be erased after encryption. Many programs are sloppy in this regard, and a user has to choose carefully.

Of course, Mallory can always replace the software encryption algorithm with something lousy. But for most users, that isn't a problem. If Mallory can break into our office and modify our encryption program, he can also put a hidden camera on the wall, a wiretap on the telephone, and a TEMPEST detector down the street. If Mallory is that much more powerful than the user, the user has lost the game before it starts.

10.6 COMPRESSION, ENCODING, AND ENCRYPTION

Using a data compression algorithm together with an encryption algorithm makes sense for two reasons:

Cryptanalysis relies on exploiting redundancies in the plaintext; com- pressing a file before encryption reduces these redu n-dancies.

Encryption is time-consuming; compressing a file before encryption speeds up the entire process.

The important thing to remember is to compress before encryption. If the encryption algorithm is any good, the ciphertext will not be compressible; it will look like random data. (This makes a reasonable test of an encryption algorithm; if the cipher-text can be compressed, then the algorithm probably isn't very good.)

If you are going to add any type of transmission encoding or error detection and recovery, remember to add that after encry p-tion. If there is noise in the communications path, decryption's error-extension properties will only make that noise worse. Figure 10.3 summarizes these steps.

10.7 DETECTING ENCRYPTION

How does Eve detect an encrypted file? Eve is in the spy business, so this is an important question. Imagine that she's eave s-dropping on a network where messages are flying in all directions at high speeds; she has to pick out the interesting ones. E n-crypted files are certainly interesting, but how does she know they are encrypted?

Generally, she relies on the fact that most popular encryption programs have well-defined headers. Electronic-mail messages encrypted with either PEM or POP (see Sections 24.10 and 24.12) are easy to identify for that reason.

Other file encryptors just produce a ciphertext file of seemingly random bits. How can she distinguish it from any other file of seemingly random bits? There is no sure way, but Eve can try a number of things:

- Examine the file. ASCII text is easy to spot. Other file formats, such as TIFF, TeX, C, Postscript, G3 facsimile, or Microsoft Excel, have standard identifying characteristics. Executable code is detectable, as well. UNIX files often have "magic numbers" that can be detected.

- Try to uncompress the file, using the major compression algorithms. If the file is compressed (and not encrypted), this should yield the original file.

- Try to compress the file. If the file is ciphertext (and the algorithm is good), then the probability that the file can be appreciably compressed by a general-purpose compression routine is small. (By appreciably, I mean more than 1 or 2 percent.) If it is something else (a binary image or a binary data file, for examples it probably can be compressed.

Any file that cannot be compressed and is not already compressed is probably ciphertext. (Of course, it is possible to specifically make ciphertext that is compressible.) Identifying the algorithm is a whole lot harder. If the algorithm is good, you can't. If the algorithm has some slight biases, it might be possible to recognize those biases in the file. However, the biases have to be pretty significant or the file has to be pretty big in order for this to work.

10.8 HIDING CIPHERTEXT IN CIPHERTEXT

Alice and Bob have been sending encrypted messages to each other for the past year. Eve has been collecting them all, but she cannot decrypt any of them. Finally, the secret police tire of all this unreadable ciphertext and arrest the pair. "Give us your encryption keys," they demand. Alice and Bob refuse, but then they notice the thumbscrews. What can they do?

Wouldn't it be nice to be able to encrypt a file such that there are two possible decryptions, each with a different key. Alice could encrypt a real message to Bob in one of the keys and some innocuous message in the other key. If Alice were caught, she could surrender the key to the innocuous message and keep the real key secret.

The easiest way to do this is with one-time pads. Let P be the plaintext, D the dummy plaintext, C the ciphertext, K the real key, and K' the dummy key. Alice encrypts P:

$$P \oplus K = C$$

Alice and Bob share K, so Bob can decrypt C:

$$C \oplus K = P$$

If the secret police ever force them to surrender their key, they don't surrender K, but instead surrender:

$$K' = C \oplus D$$

The police then recover the dummy plaintext:

$$C \oplus K' = D$$

Since these are one-time pads and K is completely random, there is no way to prove that K' was not the real key. To make

matters more convincing, Alice and Bob should concoct some mildly incriminating dummy messages to take the place of the really incriminating real messages. A pair of Israeli spies once did this.

Alice could take P and encrypt it with her favorite algorithm and key K to get C. Then she takes C and XORs it with some piece of mundane plaintext - Pride and Prejudice for example, to get K'. She stores both C and the XOR on her hard disk. Now, when the secret police interrogate her, she can explain that she is an amateur cryptographer and that K' is a merely one-time pad for C. The secret police might suspect something, but unless they know K they cannot prove that Alice's explanation isn't valid.

Another method is to encrypt P with a symmetric algorithm and K, and D with K'. Intertwine bits (or bytes) of the ciphertext to make the final ciphertexts. If the secret police demand the key, Alice gives them K' and says that the alternating bits (or bytes) are random noise designed to frustrate cryptanalysts. The trouble is the explanation is so implausible that the secret police will probably not believe her (especially considering it is suggested in this book). A better way is for Alice to create a dummy me s-sage, D, such that the concatenation of P and D, compressed, is about the same size as D. Call this concatenation P'. Alice then encrypts P' with whatever algorithm she and Bob share to get C. Then she sends C to Bob. Bob decrypts C to get P', and then P and D. Then they both compute C 0 D = K'. This K' becomes the dummy one-time pad they use in case the secret police break their doors down. Alice has to transmit D so that hers and Bob's alibis match.

Another method is for Alice to take an innocuous message and run it through some error-correcting code. Then she can i n-troduce errors that correspond to the secret encrypted message. On the receiving end, Bob can extract the errors to reconstruct the secret message and decrypt it. He can also use the error-correcting code to recover the innocuous message. Alice and Bob might be hard pressed to explain to the secret police why they consistently get a 30 percent bit-error rate on an otherwise noise-free co m-puter network, but in some circumstances this scheme can work.

Finally, Alice and Bob can use the subliminal channels in their digital signature algorithms (see Sections 4.2 and 23.3). This is undetectable, works great, but has the drawback of only allowing 20 or so characters of subliminal text to be sent per signed innocuous message. It really isn't good for much more than sending keys.

10.9 DESTROYING INFORMATION

When you delete a file on most computers, the file isn't really deleted. The only thing deleted is an entry in the disk's index file, telling the machine that the file is there. Many software vendors have made a fortune selling file-recovery software that recovers files after they have been deleted.

And there's yet another worry: Virtual memory means your computer can read and write memory to disk any time. Even if you don't save it, you never know when a sensitive document you are working on is shipped off to disk. This means that even if you never save your plaintext data, your computer might do it for you. And driver-level compression programs like Stacker and DoubleSpace can make it even harder to predict how and where information is stored on a disk.

To erase a file so that file-recovery software cannot read it, you have to physically write over all of the file's bits on the disk. According to the National Computer Security Center [1148]:

> Overwriting is a process by which unclassified data are written to storage locations that previously held sensitive data.... To purge the ... storage media, the DoD requires overwriting with a pattern, then its complement, and finally with another pattern; e.g., overwrite first with 0011 0101, followed by 1100 1010, then 1001 0111. The number of times an overwrite must be acco m-plished depends on the storage media, sometimes on its sensitivity, and sometimes on different DoD component requirements. In any case, a purge is not complete until a final over- write is made using unclassified data.

You may have to erase files or you may have to erase entire drives. You should also erase all unused space on your hard disk.

Most commercial programs that claim to implement the DoD standard over- write three times: first with all ones, then with all zeros, and finally with a repeating one-zero pattern. Given my general level of paranoia, I recommend overwriting a deleted file seven times: the first time with all ones, the second time with all zeros, and five times with a cryptographically secure pseudo-random sequence. Recent developments at the National Institute of Standards and Technology with electron-tunneling microscopes suggest even that might not be enough. Honestly, if your data is sufficiently valuable, assume that it is impossible to erase data completely off magnetic media. Burn or shred the media; it's cheaper to buy media new than to lose your secrets.

Часть III

Криптографические алгоритмы

Глава 11
Математические основы

11.1 Теория информации

Современная теория информации впервые была опубликована в 1948 году Клодом Э. Шенноном (Claude Elmwood Shannon) [1431, 1432]. (Его работы были переизданы в IEEE Press [1433].) С математической точки зрения эта тема хорошо рассмотрена в [593]. В этой главе я только схем атично излагаю основные идеи.

Энтропия и неопределенность

Теория информации определяет **количество информации** в сообщении как минимальное количество бит, необходимое для кодирования всех возможных значений сообщения, считая все сообщения равновероятными. Например, для поля дня недели в базе данных достаточно использовать три бита информации, так как вся и нформация может быть закодирована 3 битами:

000 - Воскресенье
001 - Понедельник
010 - Вторник
011 - Среда
100 - Четверг
101 - Пятница
110 - Суббота
111 - Не используется

Если эта информация была бы представлена соответствующими строками ASCII символов, она заняла бы больше места в памяти, но не содержала бы больше информации. Аналогично, поле базы данных "пол" соде ржит только один бит информации, хотя эта информация может храниться как одно из двух 7-байтовых ASCII строк: "МУЖЧИНА" или "ЖЕНЩИНА".

Формально, количество информации в сообщении M измеряется **энтропией** сообщения, обозначаемое как $H(M)$. Энтропия сообщения, определяющего пол, составляет1 бит, а энтропия сообщения, определяющего день недели, немного меньше, чем 3 бита. В общем случае энтропия сообщения, измеряемая в битах, равна $\log_2 n$, где n - это количество возможных значений. При этом предполагается, что все значения равновероятны.

Энтропия сообщения также является мерой его **неопределенности**. Это количество битов открытого текста, которое нужно раскрыть в шифротексте сообщения, чтобы узнать весь открытый текст. Например, если блок шифротекста "QHP*5M " означает либо "МУЖЧИНА", либо "ЖЕНЩИНА", то неопределенность сообщения равна 1. Криптоаналитику нужно узнать только один правильно выбранный бит, чтобы раскрыть с ообщение.

Норма языка

Для данного языка **норма языка** равна

$r = H(M)/N$

где N - это длина сообщения. При больших N норма обычного английского языка принимает различные зн ачения от 1.0 бит/буква до 1.5 бит/буква. Шеннон в [1434] говорит, что энтропия зависит от длины текста. Ко нкретно он показал, что норма для 8-буквенных блоков равна 2.3 бит/буква, но ее значение падает и находится между 1.3 и 1.5 для 16-буквенных блоков. Томас Кавер (Thomas Cover) использовал игровую методику оценки и обнаружил, что энтропия равна 1.3 бит/символ [386]. (В этой книге я буду использовать значение 1.3.) **Абсолютная норма** языка равна максимальному количеству битов, которое может быть передано каждым символом при условии, что все последовательности символов равновероятны. Если в языке L символов, то абсолютная норма равна:

$R = \log_2 L$

Это максимум энтропии отдельных символов.

Для английского языка с 26 буквами абсолютная норма равна $\log_2 26$, или около 4.7 бит/буква. Вас не дол жно удивлять, что действительная норма английского языка намного меньше, чем абсолютная - естественные языки обладают высокой избыточностью. **Избыточность** языка, обозначаемая D, определяется как:

$D=R - r$

Считая, что норма английского языка равна 1.3, избыточность составит 3.4 бит/буква. Это означает, что к аждая английская буква содержит 3.4 бита избыточной информации.

У сообщения ASCII, состоящего только из английских букв, количество информации на каждый байт с о-

ставляет 1.3 бита. Значит, в каждом байте содержится 6.7 бита избыточной информации, что дает общую изб ыточность 0.84 бита информации на бит ASCII-текста и энтропию 0.16 бита информации на бит ASCII-текста. То же сообщение, набранное кодом BAUDOT, с 5 битами на символ, имеет избыточность 0.74 бита на бит и энтр опию 0.26 бита на бит. Пробелы, пунктуация, числа и форматирование изменяют эти результаты.

Безопасность криптосистемы

Шеннон определил точную математическую модель понятия безопасности криптосистемы. Смысл работы криптоаналитика состоит в определении ключа *K*, открытого текста *P* или и того, и другого. Однако, его может устроить и некоторая вероятностная информация о *P*: является ли этот открытый текст оцифрованным звуком, немецким текстом, данными электронных таблиц или еще чем-нибудь.

В реальном криптоанализе у криптоаналитика есть некоторая вероятностная информация о *P* еще до начала работы. Он, скорее всего, знает язык открытого текста. Этот язык обладает определенной, связанной с ним и з быточностью. Если это сообщения для Боба, оно, возможно, начинается словами "Дорогой Боб". Определенно, "Дорогой Боб" намного вероятнее, чем "e8T&.g [,m". Целью криптоаналитика является изменение вероятностей, связанных с каждым возможным открытым текстом. В конце концов, из груды возможных открытых текстов будет выбран один конкретный (или, по крайней мере, весьма вероятный).

Существуют криптосистемы, достигающие **совершенной безопасности**. Такой является криптосистема, в которой шифротекст не дает никакой информации об открытом тексте (кроме, возможно, его длины). Шеннон теоретически показал, что такое возможно только, если число возможных ключей также велико, как и число возможных сообщений. Другими словами, ключ должен быть не короче самого сообщения и не может испол ьзоваться повторно. Это означает, что единственной системой, которая достигает идеальной безопасности, может быть только криптосистема с одноразовым блокнотом (см. раздел 1.5).

За исключением идеально безопасных систем, шифротекст неизбежно дает определенную информацию о с ответствующем шифротексте. Хороший криптографический алгоритм сохраняет минимум этой информации, хороший криптоаналитик пользуется этой информацией для определения открытого текста.

Криптоаналитики используют естественную избыточность языка для уменьшения числа возможных откр ытых текстов. Чем избыточнее язык, тем легче его криптоанализировать. По этой причине многие криптограф ические реализации перед шифрованием используют программы сжатия для уменьшения размера текста. Сжатие уменьшает избыточность сообщения вместе с объемом работы, необходимым для его шифрования и дешифр ирования.

Энтропия криптосистемы является мерой размера пространства ключей, *K*. Она приблизительно равна лога рифму числа ключей по основанию 2:

$H(K) = \log_2 K$

Энтропия криптосистемы с 64-битовым ключом равна 64 битам, энтропия криптосистемы с 56-битовым ключом равна 56 битам. В общем случае чем больше энтропия, тем тяжелее взломать криптосистему.

Расстояние уникальности

Для сообщения длиной *n* число различных ключей, которые расшифруют шифротекст сообщения в какой-то осмысленный открытый текст на языке оригинального открытого текста (например, английском), определяется следующей формулой [712, 95]:

$2^{H(K)-nD}-1$

Шеннон [1432] определил **расстояние уникальности**, *U*, называемое также точкой уникальности, как такое приближенное количество шифротекста, для которого сумма реальной информации (энтропия) в соответству ющем открытом тексте плюс энтропия ключа шифрования равняется числу используемых битов шифротекста. Затем он показал, что имеет смысл считать, что шифротексты, которые длиннее расстояния уникальности, мо жно расшифровать только одним осмысленным способом. Шифротексты, которые заметно короче расстояния уникальности, скорее всего, можно расшифровать несколькими способами, каждый из которых может быть правилен, и таким образом обеспечить безопасность, поставив противника перед выбором правильного откр ытого текста.

Для большинства симметричных криптосистем расстояние уникальности определяется как энтропия крипт осистемы деленная на избыточность языка.

$U = H(K)/D$

Расстояние уникальности является не точным, а вероятностным значением. Оно позволяет оценить мин имальное количество шифротекста, при вскрытии которого грубой силой имеется, вероятно, только один разу мный способ дешифрирования. Обычно чем больше расстояние уникальности, тем лучше криптосистема. Для DES с 56-битовым ключом и англоязычного сообщения, записанного символами ASCII, расстояние уникальн о-

сти приблизительно равно 8.2 символа ASCII или 66 бит. В 1405-й приведены расстояния уникальности для различных длин ключа. Расстояния уникальности для некоторых классических криптосистем можно найти в [445].

Расстояние уникальности измеряет не количество криптотекста, нужного для криптоанализа, а количество криптотекста, необходимое для единственности результата криптоанализа. Криптосистема может быть вычислительно неуязвима, даже если теоретически ее возможно взломать, используя малое количество шифротекста. (Уместно вспомнить о весьма эзотерической теории релятивистской криптографии [230, 231, 232, 233, 234, 235].) Расстояние уникальности пропорционально избыточности. Если избыточность стремится к нулю, даже тривиальный шифр может не поддаться вскрытию с использованием только шифротекста.

Табл. 11-1.
Расстояния уникальности текста ASCII, зашифрованного алгоритмами с различной длиной ключа

Длина ключа (в битах)	Расстояние уникальности (в символах)
40	5.9
56	8.2
64	9.4
80	11.8
128	18.8
256	37.6

Шеннон определил криптосистему с бесконечным расстоянием уникальности, как обладающую **идеальной тайной**. Обратите внимание, что идеальная криптосистема не обязательно является совершенной, хотя совершенная криптосистема обязательно будет и идеальной. Если криптосистема обладает идеальной тайной, то даже при успешном криптоанализе останется некоторая неопределенность, является ли восстановленный открытый текст реальным открытым текстом.

Практическое использование теории информации

Хотя эти понятия имеют большое теоретическое значение, реальный криптоанализ использует их достаточно редко. Расстояние уникальности гарантирует ненадежность системы, если оно слишком мало, но его высокое значение не гарантирует безопасности. Несколько практических алгоритмов абсолютно не поддаются анализу, поведение параметров теории информации могло бы способствовать взлому некоторых шифрованных сообщений. Однако, подобные соображения теории информации иногда полезны, например, для определения в конкретном алгоритме рекомендуемого интервала изменения ключей. Криптоаналитики также используют ряд тестов не базе статистики и теории информации, чтобы выбирать наиболее перспективные направления анализа. К сожалению, большинство литературы по применению теории информации в криптоанализе остается секретной, включая основополагающую работу Алана Тьюринга (Alan Turing), написанную в 1940.

Путаница и диффузия

Двумя основными методами маскировки избыточности открытого текста сообщения, согласно Шеннону, служат путаница и диффузия [1432].

Путаница маскирует связь между открытым текстом и шифротекстом. Она затрудняет попытки найти в шифротексте избыточность и статистические закономерности. Простейшим путем создать путаницу является подстановка. В простом подстановочном шифре, например, шифре Цезаря, все одинаковые буквы открытого текста заменяются другими одинаковыми буквами шифротекста. Современные подстановочные шифры являются более сложными: длинный блок открытого текста заменяется блоком шифротекста, и способ замены меняется с каждым битом открытого текста или ключа. Такого типа подстановки обычно недостаточно - сложный алгоритм немецкой Энигмы был взломан в ходе второй мировой войны.

Диффузия рассеивает избыточность открытого текста, распространяя ее по всему шифротексту. Криптоаналитику потребуется немало времени для поиска избыточности. Простейшим способом создать диффузию является транспозиция (также называемая **перестановкой**). Простой перестановочный шифр только переставляет буквы открытого текста. Современные шифры также выполняют такую перестановку, но они также используют другие формы диффузии, которые позволяют разбросать части сообщения по всему сообщению.

Потоковые шифры используют только путаницу, хотя ряд схем с обратной связью добавляют диффузию. Блочные алгоритмы применяют и путаницу, и диффузию. Как правило, диффузию саму по себе несложно взло-

мать (хотя шифры с двойной перестановкой оказываются поустойчивее, чем другие некомпьютерные системы).

11.2 Теория сложности

Теория сложности обеспечивает методологию анализа **вычислительной сложности** различных криптографических методов и алгоритмов. Она сравнивает криптографические методы и алгоритмы и определяет их безопасность. Теория информации сообщает нам о том, что все криптографические алгоритмы (кроме однор азовых блокнотов) могут быть взломаны. Теория сложности сообщает, могут ли они быть взломаны до тепловой смерти вселенной.

Сложность алгоритмов

Сложность алгоритма определяется вычислительными мощностями, необходимыми для его выполнения. Вычислительная сложность алгоритма часто измеряется двумя параметрами: T (**временная сложность**) и S (**пространственная сложность**, или требования к памяти). И T, и S обычно представляются в виде функций от n, где n - это размер входных данных. (Существую и другие способы измерения сложности: количество случа йных бит, ширина канала связи, объем данных и т.п.)

Обычно вычислительная сложность алгоритма выражается с помощью нотации "О большого", т.е описыв ается порядком величины вычислительной сложности. Это просто член разложения функции сложности, быстрее всего растущий с ростом n, все члены низшего порядка игнорируются. Например, если временная сложность данного алгоритма равна $4n^2+7n+12$, то вычислительная сложность порядка n^2, записываемая как $O(n^2)$.

Временная сложность измеренная таким образом не зависит от реализации. Не нужно знать ни точное время выполнения различных инструкций, ни число битов, используемых для представления различных переменных, ни даже скорость процессора. Один компьютер может быть на 50 процентов быстрее другого, а у третьего шина данных может быть в два раза шире, но сложность алгоритма, оцененная по прядку величины, не изменится. Это не жульничество, при работе с алгоритмами настолько сложными, как описанные в этой книге, всем пр очим можно пренебречь (с точностью до постоянного множителя) в сравнении со сложностью по порядку вел ичины.

Эта нотация позволяет увидеть, как объем входных данных влияет на требования к времени и объему пам яти. Например, если $T= O(n)$, то удвоение входных данных удвоит и время выполнения алгоритма. Если $T=O(2^n)$, то добавление одного бита к входным данным удвоит время выполнения алгоритма.

Обычно алгоритмы классифицируются в соответствии с их временной или пространственной сложностью. Алгоритм называют **постоянным**, если его сложность не зависит от n: O(1). Алгоритм является **линейным**, если его временная сложность $O(n)$. Алгоритмы могут быть **квадратичными**, **кубическими** и т.д. Все эти алгоритмы - **полиномиальны**, их сложность - $O(n^m)$, где m - константа. Алгоритмы с полиномиальной временной сложностью называются алгоритмами **с полиномиальным временем**.

Алгоритмы, сложность которых равна $O(t^{f(n)})$, где t - константа, большая, чем 1, а $f(n)$ - некоторая полиномиальная функция от n, называются **экспоненциальными**. Подмножество экспоненциальных алгоритмов, сло жность которых равна $O(c^{f(n)})$, где где с - константа, а $f(n)$ возрастает быстрее, чем постоянная, но медленнее, чем линейная функция, называется **суперполиномиальным**.

В идеале, криптограф хотел бы утверждать, что алгоритм, лучший для взлома спроектированного алгоритма шифрования, обладает экспоненциальной временной сложностью. На практике, самые сильные утверждения, которые могут быть сделаны при текущем состоянии теории вычислительной сложности, имеют форму "все и звестные алгоритмы вскрытия данной криптосистемы обладают суперполиномиальной временной сложностью". То есть, известные нам алгоритмы вскрытия обладают суперполиномиальной временной сложностью, но пока невозможно доказать, что не может быть открыт алгоритм вскрытия с полиномиальной временной сложностью. Развитие теории вычислительной сложности возможно когда-нибудь позволит создать алгоритмы, для которых существование алгоритмов с полиномиальным временем вскрытия может быть исключено с математической точностью.

С ростом n временная сложность алгоритмов может стать настолько огромной, что это повлияет на практ ическую реализуемость алгоритма. В 9-й показано время выполнения для различных классов алгоритмов при n равном одному миллиону. В таблице игнорируются постоянные величины, но показано, почему это можно д елать.

Табл. 11-2
Время выполнения для различных классов алюритмов

Класс	Сложность	Количество операций для $n=10^6$	Время при 10^6 операций в секунду

Постоянные	O(1)	1	1 мкс
Линейные	O(n)	10^6	1 с
Квадратичные	O(n^2)	10^{12}	11.6 дня
Кубические	O(n^3)	10^{18}	32000 лет
Экспоненциальные	O(2^n)	10^{301030}	В 10^{301006} раз больше, чем время существования вселенной

При условии, что единицей времени для нашего компьютера является микросекунда, компьютер может выполнить постоянный алгоритм за микросекунду, линейный - за секунду, а квадратичный - за 11.6 дня. Выполнение кубического алгоритма потребует 32 тысяч лет, что в принципе реализуемо, компьютер, конструкция которого позволила бы ему противостоять следующему ледниковому периоду, в конце концов получил бы решение. Выполнение экспоненциального алгоритма тщетно, независимо от экстраполяции роста мощи компьютеров, параллельной обработки или контактов с инопланетным суперразумом.

Взглянем на проблему вскрытия алгоритма шифрования грубой силой. Временная сложность такого вскрытия пропорциональна количеству возможных ключей, которое экспоненциально зависит от длины ключа. Если n - длина ключа, то сложность вскрытия грубой силой равна O(2^n). В разделе 12.3 рассматривается дискуссия об использовании для DES 56-битового ключа вместо 112-битового. Сложность вскрытия грубой силой при 56-битовом ключе составляет 2^{56}, а при 112-битовом ключе - 2^{112}. В первом случае вскрытие возможно, а во втором - нет.

Сложность проблем

Теория сложности также классифицирует и сложность самих проблем, а не только сложность конкретных алгоритмов решения проблемы. (Отличным введением в эту тему являются [600, 211, 1226], см. также [1096, 27, 739].) Теория рассматривает минимальное время и объем памяти, необходимые для решения самого трудного варианта проблемы на теоретическом компьютере, известном как **машина Тьюринга**. Машина Тьюринга представляет собой конечный автомат с бесконечной лентой памяти для чтения-записи и является реалистичной моделью вычислений.

Проблемы, которые можно решить с помощью алгоритмов с полиномиальным временем, называются решаемыми, потому что для разумных входных данных обычно могут быть решены за разумное время. (Точное определение "разумности" зависит от конкретных обстоятельств.) Проблемы, которые невозможно решить за полиномиальное время, называются нерешаемыми, потому что вычисление их решений быстро становится невозможным. Нерешаемые проблемы иногда называют **трудными**. Проблемы, которые могут быть решены только с помощью суперполиномиальных алгоритмов, вычислительно нерешаемы, даже при относительно малых значениях n.

Что еще хуже, Алан Тьюринг доказал, что некоторые проблемы **принципиально неразрешимы**. Даже отвлекаясь от временной сложности алгоритма, невозможно создать алгоритм решения этих проблем.

Проблемы можно разбить на классы в соответствии со сложностью их решения. Самые важные классы и их предполагаемые соотношения показаны на 10-й. (К несчастью, лишь малая часть этих утверждений может быть доказана математически.)

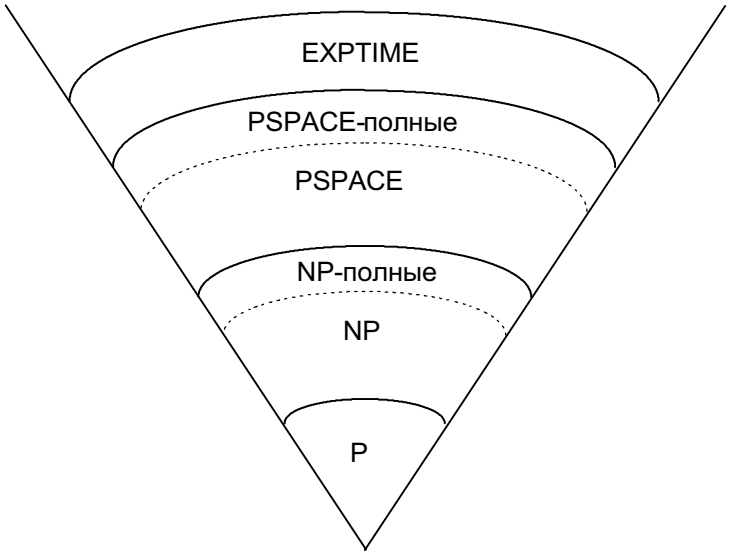


Рис. 11-1. Классы сложности

Находящийся в самом низу класс **P** состоит из всех проблем, которые можно решить за полиномиальное время. Класс **NP** - из всех проблем, которые можно решить за полиномиальное время только на недетермин ированной машине Тьюринга: вариант обычной машины Тьюринга, которая может делать предположения. М ашина предполагает решение проблемы - либо "удачно угадывая", либо перебирая все предположения пара ллельно - и проверяет свое предположение за полиномиальное время.

Важность **NP** в криптографии состоит в следующем: многие симметричные алгоритмы и алгоритмы с о ткрытыми ключами могут быть взломаны за недетерминированное полиномиальное время. Для данного шифр отекста *C*, криптоаналитик просто угадывает открытый текст, *X*, и ключ, *k*, и за полиномиальное время выполн яет алгоритм шифрования со входами *X* и *k* и проверяет, равен ли результат *C*. Это имеет важное теоретическое значение, потому что устанавливает верхнюю границу сложности криптоанализа этих алгоритмов. На практике, конечно же, это выполняемый за полиномиальное время детерминированный алгоритм, который и ищет кри птоаналитик. Более того, этот аргумент неприменим ко всем классам шифров, конкретно, он не применим для одноразовых блокнотов - для любого *C* существует множество пар *X*, *k*, дающих *C* при выполнении алгоритма шифрования, но большинство этих *X* представляют собой бессмысленные, недопустимые открытые тексты.

Класс **NP** включает класс **P**, так как любая проблема, решаемая за полиномиальное время на детерминир ованной машине Тьюринга, будет также решена за полиномиальное время на недетерминированной машине Тьюринга, просто пропускается этап предпол ожения.

Если все **NP** проблемы решаются за полиномиальное время на детерминированной машине, то **P** = **NP**. Хотя кажется очевидным, что некоторые **NP** проблемы намного сложнее других (вскрытие алгоритма шифрования грубой силой против шифрования произвольного блока шифротекста), никогда не было доказано, что **P** ≠ **NP** (или что **P** = **NP**). Однако, большинство людей, работающих над теорией сложности, убеждены, что эти классы неравны.

Что удивительно, можно доказать, что конкретные **NP**-проблемы настолько же трудны, как и любая пробл ема этого класса. Стивен Кук (Steven Cook) доказал [365], что проблема Выполнимости (Satisfiability problem, дано правильное логическое выражение, существует ли способ присвоить правильные значения входящим в него переменным так, чтобы все выражение стало истиной?) является **NP-полной**. Это означает, что, если про блема Выполнимости решается за полиномиальное время, то **P** = **NP**. Наоборот, если может быть доказано, что для любой проблемы класса **NP** не существует детерминированного алгоритма с полиномиальным временем решения, доказательство покажет, что и для проблемы Выполнимости не существует детерминированного алг оритма с полиномиальным временем решения. В **NP** нет проблемы труднее, чем проблема Выполнимости.

С тех пор, как основополагающая работа Кука была опубликована, было показано, что существует множес тво проблем, эквивалентных проблеме Выполнимости, сотни их перечислены в [600], ряд примеров приведен ниже. Из-за эквивалентности я полагаю, что эти проблемы также являются **NP-полными**, они входят в класс NP и так же сложны, как и любая проблема класса **NP**. Если бы была доказана их решаемость за детерминир ованное полиномиальное время, вопрос **P** против **NP** был бы решен. Вопрос, верно ли **P** = **NP**, является цен тральным нерешенным вопросом теории вычислительной сложности, и не ожидается, что он будет решен в ближайшее время. Если кто-то покажет, что **P** = **NP**, то большая часть этой книги станет ненужной: как объя снялось ранее многие классы шифров тривиально взламываются за недетерминированное полиномиальное вр е-

мя. Если **P** = **NP**, то они вскрываются слабыми, детерминированными алгоритмами.

Следующим в иерархии сложности идет класс **PSPACE**. Проблемы класса **PSPACE** могут быть решены в полиномиальном пространстве, но не обязательно за полиномиальное время. **PSPACE** включает **NP**, но ряд проблем **PSPACE** кажутся сложнее, чем **NP**. Конечно, и это пока недоказуемо. Существует класс проблем, так называемых **PSPACE-полных**, обладающих следующим свойством: если любая из них является **NP**-проблемой, то **PSPACE** = **NP**, и если любая из них является **P**-проблемой, то **PSPACE** = **P**.

И наконец, существует класс проблем **EXPTIME**. Эти проблемы решаются за экспоненциальное время. Может быть действительно доказано, что **EXPTIME-полные** проблемы не могут быть решены за детерминированное полиномиальное время. Также показано, что **P** не равно **EXPTIME**.

NP-полные проблемы

Майкл Кэри (Michael Carey) и Дэвид Джонсон (David Johnson) составили список более чем 300 NP-полных проблем [600]. Вот некоторые:

— Проблема путешествующего коммивояжера. Путешествующему коммивояжеру нужно посетить различные города, используя только один бак с горючим (существует максимальное расстояние, которое он может проехать). Существует ли маршрут, позволяющий ему посетить каждый голод только один раз, и с-пользуя этот единственный бак с горючим? (Это обобщение проблемы гамильтонова пути - см. раздел 5.1.)

— Проблема тройного брака. В комнате n мужчин, n женщин и n чиновников (священников, раввинов, кого угодно). Есть список разрешенных браков, записи которого состоят из одного мужчины, одной женщины и одного регистрирующего чиновника. Дан этот список троек, возможно ли построить n браков так, чтобы любой либо сочетался браком только с одним человеком или регистрировал только один брак?

— Тройная выполнимость. Есть список n логических выражений, каждое с тремя переменными. Например: если (x и y) то z, (x и w) или (не z), если ((не u и не x) или (z и (u или не x))) то (не z и u) или x), и т.д. Существует ли правильные значения всех переменных, чтобы все утверждения были истинными? (Это частный случай упомянутой выше проблемы Выполнимости.)

11.3 Теория чисел

Это не книга по теории чисел, поэтому я только набросаю ряд идей, используемых в криптографии. Если вам нужно подробное математическое изложение теории чисел, обратитесь к одной из этих книг: [1430, 72, 1171, 12, 959, 681, 742, 420]. Моими любимыми книгами по математике конечных полей являются [971, 1042]. См. также [88, 1157, 1158, 1060].

Арифметика вычетов

Вы все учили математику вычетов в школе. Иногда ее называли "арифметикой часов". Если Милдред сказала, что она будет дома к 10:00, и опоздала на 13 часов, то когда она придет домой, и на сколько лет отец лишит ее водительских прав? Это арифметика по модулю 12. Двадцать три по модулю 12 равно 11.

(10 + 13) mod 12 = 23 mod 12 = 11 mod 12

Другим способом записать это является утверждение об эквивалентности 23 и 11 по модулю 12:

10 + 13 ≡ 11 (mod 12)

В основном, $a \equiv b \pmod{n}$, если $a = b + kn$ для некоторого целого k. Если a неотрицательно и b находится между 0 и n, можно рассматривать b как остаток при делении a на n. Иногда, b называется **вычетом** a по модулю n. Иногда a называется **конгруэнтным** b по модулю n (знак тройного равенства, ≡, обозначает конгруэнтность). Одно и то же можно сказать разными способами.

Множество чисел от 0 до n-1 образует то, что называется **полным множеством вычетов** по модулю n. Это означает, что для любого целого a, его остаток по модулю n является некоторым числом от 0 до n-1.

Операция a mod n обозначает остаток от a, являющийся некоторым целым числом от 0 до n-1. Эта операция называется **приведением по модулю**. Например, 5 mod 3 = 2.

Это определение mod может отличаться от принятого в некоторых языках программирования. Например, оператор получения остатка в языке PASCAL иногда возвращает отрицательное число. Он возвращает число между -(n-1) и n-1. В языке C оператор % возвращает остаток от деления первого выражения на второе, оно может быть отрицательным числом, если любой из операндов отрицателен. Для всех алгоритмов в этой книге проверяйте, что вы добавляете n к результату операции получения остатка, если она возвращает отрицательное число.

Арифметика остатков очень похожа на обычную арифметику: она коммутативна, ассоциативна и дистриб утивна. Кроме того, приведение каждого промежуточного результата по модулю n дает тот же результат, как и выполнение всего вычисления с последующим приведением конечного результата по модулю n.

$(a + b) \bmod n == ((a \bmod n) + (b \bmod n)) \bmod n$

$(a - b) \bmod n == ((a \bmod n) - (b \bmod n)) \bmod n$

$(a * b) \bmod n == ((a \bmod n) * (b \bmod n)) \bmod n$

$(a * (b+c)) \bmod n == (((a*b) \bmod n) + ((a*c) \bmod n)) \bmod n$

Вычисление mod n часто используется в криптографии, так как вычисление дискретных логарифмов и ква дратных корней mod n может быть нелегкой проблемой. Арифметика вычетов, к тому же, легче реализуется на компьютерах, поскольку она ограничивает диапазон промежуточных значений и результата. Для k-битовых вычетов n, промежуточные результаты любого сложения, вычитание или умножения будут не длиннее, чем 2 k бит. Поэтому в арифметике вычетов мы можем выполнить возведение в степень без огромных промежуточных р езультатов. Вычисление степени некоторого числа по м одулю другого числа,

$a^x \bmod n$,

представляет собой просто последовательность умножений и делений, но существуют приемы, ускоряющие это действие. Один из таких приемов стремится минимизировать количество умножений по модулю, другой - оптимизировать отдельные умножения по модулю. Так как операции дистрибутивны, быстрее выполнить возв едение в степень как поток последовательных умножений, каждый раз получая вычеты. Сейчас вы не чувствуете разницы, но она будет заметна при умножении 200-битовых чисел.

Например, если вы хотите вычислить $a^8 \bmod n$, не выполняйте наивно семь умножений и одно приведение по модулю:

$(a * a * a * a * a * a * a * a) \bmod n$

Вместо этого выполните три меньших умножения и три меньших приведения по м одулю:

$((a^2 \bmod n)^2 \bmod n)^2 \bmod n$

Точно также,

$a^{16} \bmod n = (((a^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n$

Вычисление a^x, где x не является степенью 2, ненамного труднее. Двоичная запись представляет x в виде суммы степеней 2: 25 - это бинарное 11001, поэтому 25 = 24 + 23 + 20. Поэтому

$a^{25} \bmod n = (a*a^{24}) \bmod n = (a* a^8*a^{16}) \bmod n =$

$= (a*((a^2)^2)^2*(((a^2)^2)^2)^2) \bmod n = (a*(((a*a^2)^2)^2)^2) \bmod n$

С продуманным сохранением промежуточных результатов вам понадобится только шесть умножений:

$(((((((a^2 \bmod n)* a)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 *a) \bmod n$

Такой прием называется **цепочкой сложений** [863], или методом двоичных квадратов и умножения. Он и спользует простую и очевидную цепочку сложений, в основе которой лежит двоичное представление числа. На языке С это выглядит следующим образом:

```
unsigned long qe2(unsigned long x, unsigned long y, unsigned long n) {
     unsigned long s, t, u;
     int i;
     s=1; t=x; u=y;
     while (u) {
             if(u&1) s=(s*t)%n;
             u>>1;
             t=(t*t)%n;
     }
     return(s)
}
```

А вот другой, рекурсивный, алгоритм:

```
unsigned long fast_exp(unsigned long x, unsigned long y, unsigned long N) {
     unsigned long tmp;
```

```
    if(y==1) return(x % N);
    if (1^(x&1)) {
            tmp= fast_exp(x,y/2,N);
            return ((tmp*tmp)%N);
    else {
            tmp = fast_exp(x,(y-1)/2,N);
            tmp = (tmp*tmp)%N;
            tmp = (tmp*x)%N;
            return (tmp);
    }
}
```

Этот метод уменьшает количество операций, в среднем, до 1.5* k операций, где k - длина числа x в битах. Найти способ вычисления с наименьшим количеством операций - трудная проблема (было доказано, что посл е-довательность должна содержать не меньше k-1 операций), но нетрудно снизить число операций до 1.1* k или даже лучше при больших k.

Эффективным способом много раз выполнять приведение по модулю для одного n является **метод Монтгомери** [1111]. Другой метод называется **алгоритмом Баррета** [87]. Эффективность описанного алгоритма и этих двух методов рассматривается в [210]: алгоритм, рассмотренный мною, является наилучшим для едини ч-ного приведения по модулю, алгоритм Баррета - наилучшим для малых аргументов, а метод Монтгомери - на и-лучшим для обычного возведения в степень по модулю. (Метод Монтгомери также использует преимущество малых показателей степени, используя прием, называющийся смешанной арифметикой.)

Операция, обратная возведению в степень по модулю n, вычисляет **дискретный логарифм**. Я дальше вкратце рассмотрю эту операцию.

Простые числа

Простым называется целое число, большее единицы, единственными множителями которого является 1 и оно само: оно не делится ни на одно другое число. Два - это простое число. Простыми являются и 73, 2521, 2365347734339 и 2^{756839}-1. Существует бесконечно много простых чисел. Криптография, особенно криптография с открытыми ключами, часто использует большие простые числа (512 бит и даже бол ьше).

Евангелос Кранакис (Evangelos Kranakis) написал отличную книгу по теории чисел, простым числам и их применению в криптографии [896]. Паула Рибенбойм (Paula Ribenboim) написала две отличных справочных работы по простым числам вообще [1307, 1308].

Наибольший общий делитель

Два числа называются **взаимно простыми**, если у них нет общих множителей кроме 1. Иными словами, е с-ли **наибольший общий делитель** a и n равен 1. Это записывается как:

НОД(a,n)=1

Взаимно просты числа 15 и 28. 15 и 27 не являются взаимно простыми, а 13 и 500 - являются. Простое число взаимно просто со всеми другими числами, кроме ч исел, кратных данному простому числу.

Одним из способов вычислить наибольший общий делитель двух чисел является **алгоритм Эвклида**. Эвк-лид описал этот алгоритм в своей книге, *Элементы*, написанной в 300 году до нашей эры. Он не изобрел его. Историки считают, что этот алгоритм лет на 200 старше. Это самый древний нетривиальный алгоритм, который дошел до наших дней, и он все еще хорош. Кнут описал алгоритм и его современные модификации в [863]. На языке C:

```
/* возвращает НОД (gcd) x и y */
int gcd (int x, int y) {
    int g;
    if (x < 0)
            x = -x;
    if (y < 0)
            y = -y;
    if (x + y == 0 )
            ERROR ;
```

```
    g = y;
    while (x > 0) {
            g = x;
            x = y % x;
            y = g;
    }
    return g;
}
```

Этот алгоритм можно обобщить для получения НОД массива *m* чисел:

```
/* возвращает НОД (gcd) xl, x2...xm */
int multiple_gcd (int m, int *x) {
    slze_t i;
    int g;
    if (m < 1)
            return 0;
    g = x [0];
    for (i=l; i<m; ++i) {
            g = gcd(g, x[i]);
/* оптимизация, так как для случайных x[i], g==l в 60% случаев: */
            if (g == 1)
            return 1;
    }
    return g;
}
```

Обратные значения по модулю

Помните, что такое обратные значения? Обратное значение для 4 - 1/4, потому что 4*1/4 =1. В мире вычетов проблема усложняется:

$4*x = 1 \pmod 7$

Это уравнение эквивалентно обнаружению x и k, таких что

$4x = 7k + 1$

где x и k - целые числа. Общая задача состоит в нахождении x, такого что

$1 = (a*x) \bmod n$

Это также можно записать как

$a^{-1} \equiv x \pmod n$

Проблему обратных значений по модулю решить нелегко. Иногда у нее есть решение, иногда нет. Например, обратное значение 5 по модулю 14 равно 3. С другой стороны у числа 2 нет обратного значения по модулю 14.

В общем случае у уравнения $a^{-1} \equiv x \pmod n$ существует единственное решение, если a и n взаимно просты. Если a и n не являются взаимно простыми, то $a^{-1} \equiv x \pmod n$ не имеет решений. Если n является простым числом, то любое число от 1 до n -1 взаимно просто с n и имеет в точности одно обратное значение по модулю n.

Так, хорошо. А теперь как вы собираетесь искать обратное значение a по модулю n? Существует два пути. Обратное значение a по модулю n можно вычислить с помощью алгоритма Эвклида. Иногда это называется расширенным алгоритмом Эвклида.

Вот этот алгоритм на языке C++:

```
#define isEven(x) ((x & 0x01) == 0)
#define isOdd(x)  (x& 0x01)
#define swap(x,y) (x^= y, y^= x, x^= y)
void ExtBinEuclid(int *u, int *v, int *u1, int *u2, int *u3) {
    // предупреждение: u и v будут переставлены, если u<v
int k, tl, t2, t3;
```

```
if (*u < *v) swap(*u<,*v);
for (k = 0; isEven(*u) && isEven(*v); ++k) {
*u>>=1; *v >>1;
}
*u1 = 1; *u2 = 0; *u3 = *u; t1 = *v; t2 = *u - 1; t3 = *v;
do {
do {
if (isEven(*u3)) {
if (isOdd(*ul) || isOdd(*u2)) {
*u1 += *v; *u2 += *u;
                            }
*ul >>* 1; *u2 >>= 1; *u3 >>= 1;
                    }
if (isEven(t3) || *u3 < t3) {
swap(*ul,tl); smap(*u2,t2); smap(*u3,t3);
                    }
} while (isEven(*u3));
while (*ul < tl || *u2 < t2) {
*ul += *v; *u2 += *u;
            }
            ul -= tl; *u2 -= t2; *u3 -= t3;
} while (t3 > 0);
while (*ul >= *v && *u2 >= *u) {
*ul>l -= *v; *u2 -= *u;
     }
*u <<= k; *v <<= k; *u3 << k;
}
main(int argc, char **argv) {
int a, b, gcd;
if (argc < 3) {
cerr << "как использовать: xeuclid u v" << end1;
return -1;
     }
     int u = atoi(argv[1]);
int v = atoi(argv[2]);
if (u <= 0 || v <= 0 ) {
     cerr << "Аргумент должен быть положителен!" << end1;
return -2;
     }
// предупреждение: u и v будут переставлены если u < v
ExtBinEuclid(&u, &v, &a, &b, &gcd);
cout << a <<" * " << u << " + (-"
<< b << ") * " << v << " = " << gcd << end1;
if (gcd == 1)
cout << "Обратное значение " << v << " mod " << u << " is: "
<< u - b << end1;
return 0;
}
```

Я не собираюсь доказывать, что это работает, или приводить теоретическое обоснование. Подробности можно найти в [863] или в любой из приведенных ранее работ по теории чисел.

Алгоритм итеративен и для больших чисел может работать медленно. Кнут показал, что среднее число в ы-

полняемых алгоритмом делений равно:

$0.843*\log_2(n) + 1.47$

Решение для коэффициентов

Алгоритм Эвклида можно использовать и для решения следующих проблем: дан массив из m переменных $x_1, x_2, ..., x_m$, найти массив m коэффициентов, $u_1, u_2, ..., u_m$, таких что

$u_1 * x_1 + ... + u_m * x_m, = 1$

Малая теорема Ферма

Если m - простое число, и a не кратно m, то **малая теорема Ферма** утверждает

$a^{m-1} \equiv 1 \pmod m$

(Пьер де Ферма (Pierre de Fermat), французский математик, жил с 1601 по 1665 год. Эта теорема не имеет ничего общего с его знаменитой теоремой.)

Функция Эйлера

Существует другой способ вычислить обратное значение по модулю n, но его не всегда возможно использовать. **Приведенным множеством остатков** mod n называется подмножество полного множества остатков, члены которого взаимно просты с n. Например, приведенное множество остатков mod 12 - это {1, 5, 7, 11}. Если n - простое число, то приведенное множество остатков mod n - это множество всех чисел от 1 до n-1. Для любого n, не равного 1,число 0 никогда не входит в приведенное множество остатков.

Функция Эйлера, которую также называют функцией фи Эйлера и записывают как $\phi(n)$, - это количество элементов в приведенном множестве остатков по модулю n. Иными словами, $\phi(n)$ - это количество положительных целых чисел, меньших n и взаимно простых с n (для любого n, большего 1). (Леонард Эйлер (Leonhard Euler), швейцарский математик, жил с 1707 по 1783 год.)

Если n - простое число, то $\phi(n) = n$-1. Если $n = pq$, где p и q -простые числа, то $\phi(n) = (p - 1)(q - 1)$. Эти числа появляются в некоторых алгоритмах с открытыми ключами, и вот почему. В соответствии с обобщением Эйлера малой теоремы Ферма, если НОД(a,n) = 1, то

$a^{\phi(n)} \bmod n = 1$

Теперь легко вычислить $a^{-1} \bmod n$:

$x = a^{\phi(n)-1} \bmod n$

Например, какое число является обратным для 5 по модулю 7? Так как 7 - простое число, $\phi(7) = 7 - 1 = 6$. Итак, число, обратное к 5 по модулю 7, равно

$5^{6-1} \bmod 7 = 5^5 \bmod 7 = 3$

Эти методы вычисления обратных значений можно расширить для более общей проблемы нахождения x (если НОД(a,n) = 1):

$(a*x) \bmod n = b$

Используя обобщение Эйлера, решаем

$x = (b* a^{\phi(n)-1}) \bmod n$

Используя алгоритм Эвклида, находим

$x = (b* (a^{-1} \bmod n)) \bmod n$

В общем случае для вычисления обратных значений алгоритм Эвклида быстрее, чем обобщение Эйлера, особенно для чисел длиной порядка 500 бит. Если НОД(a,n) $\neq$ 1, не все потеряно. В этом общем случае ($a*x$) mod n=b, может иметь или несколько решений, или ни одного.

Китайская теорема об остатках

Если известно разложение числа n на простые сомножители, то для решения полной системы уравнений можно воспользоваться Китайской теоремой об остатках. Основной вариант этой теоремы был открыт в первом веке китайским математиком Сун Цзе.

В общем случае, если разложение числа n на простые сомножители представляет собой $p_1*p_2*...*p_t$, то система уравнений

($x \bmod p_i$) = a_i, где i = 1, 2, . . . , t

имеет единственное решение, x, меньшее n. (Обратите внимание, что некоторые простые числа могут появляться несколько раз. Например, p_1 может быть равно p_2.) Другими словами, число (меньшее, чем произведение нескольких простых чисел) однозначно определяется своими остатками от деления на эти простые числа.

Например, возьмем простые числа 3 и 5, и 14 в качестве заданного числа. 14 mod 3 = 2, и 14 mod 5 = 4. С уществует единственное число, меньшее 3*5 = 15, с такими остатками: 14. Два остатка однозначно определяют число.

Поэтому для произвольного $a < p$ и $b < q$ (где p и q - простые числа), существует единственное число x, меньшее pq, такое что

$x \equiv a \pmod p$, и $x \equiv b \pmod q$

Для получения x сначала воспользуемся алгоритмом Эвклида, чтобы найти u, такое что

$u*q \equiv 1 \pmod p$

Затем вычислим:

$x = (((a - b) * u) \bmod p) * q + b$

Вот как выглядит Китайская теорема об остатках на языке С:

```
/* r - это количество элементов в массивах m and u;
m - это массив (попарно взаимно простых) модулей
u - это массив коэффициентов
возвращает значение n, такое что n == u[k]%m[k] (k=0..r-1) и
n < [m[0]*m[l]*...*m[r-1]
*/
/* Получение функции Эйлера (totient) остается упражнением для читателя. */
int Chinese_remainder (size_t r, int *m, int *u) {
size_t i;
int modulus;
int n;
modulus=1;
for (i=0; i<r; ++i)
modulus*=m[i];
n=0;
for (i=0; i<r; ++i) {
n+=u[i] * modexp(modulus/m[i]*totient(m[i]),m[i]);
n %= modulus;
     }
return n;
}
```

Обращение Китайской теоремы об остатках может быть использовано для решения следующей проблемы: если p и q - простые числа, и p меньше q, то существует единственное x, меньшее, чем pq, такое что

$a \equiv x \pmod p$, и b ≡ x (mod q)

Если $a \geq b \bmod p$, то

$x = (((a - (b \bmod p)) * u) \bmod p) * q + b$

Если $a < b \bmod p$, то

$x = (((a + p - (b \bmod p))*u) \bmod p)*q + b$

Квадратичные вычеты

Если p - простое число, и a больше 0, но меньше p, то a представляет собой квадратичный вычет по модулю p, если

$x^2 \equiv a \pmod p$, для некоторых x

Не все значения *a* соответствуют этому требованию. Чтобы *a* было квадратичным вычетом по *n*, оно должно быть квадратичным вычетом по модулю всех простых сомножителей *n*. Например, если *p* = 7, квадратичными вычетами являются числа 1, 2, и 4:

$1^2 = 1 \equiv 1 \pmod 7$

$2^2 = 4 \equiv 4 \pmod 7$

$3^2 = 9 \equiv 2 \pmod 7$

$4^2 = 16 \equiv 2 \pmod 7$

$5^2 = 25 \equiv 4 \pmod 7$

$6^2 = 36 \equiv 1 \pmod 7$

Заметьте, что каждый квадратичный вычет дважды появляется в этом списке. Значений *x*, удовлетворяющих любому из следующих уравнений, не существует:

$x^2 \equiv 3 \pmod 7$

$x^2 \equiv 5 \pmod 7$

$x^2 \equiv 6 \pmod 7$

Эти числа - 3, 5 и 6 - не являются квадратичными вычетами по модулю 7.

Хотя я этого и не делаю, несложно доказать, что когда *p* нечетно, существует в точности (*p* - 1)/2 квадратичных вычетов по модулю *p*, и столько же чисел, не являющихся квадратичными вычетами по модулю *p*. Кроме того, если *a* - это квадратичный вычет по модулю *p*, то у *a* в точности два квадратных корня, один между 0 и (*p*-1)/2, а второй - между (*p* - 1)/2 и (*p* - 1). Один из этих квадратных корней одновременно является квадрати чным остатком по модулю *p*, он называется **главным квадратным корнем**.

Если *n* является произведением двух простых чисел, *p* и *q*, то существует ровно (*p* - l)(*q* - 1)/4 квадратичных вычетов по модулю *n*. Квадратичный вычет по модулю *n* является совершенным квадратом по модулю *n*, потому что для того, чтобы быть квадратом по модулю *n*, вычет должен быть квадратом по модулю *p* и квадратом по модулю *q*. Например, существует одиннадцать квадратичных остатков mod 35: 1, 4, 9, 11, 15, 16, 21, 25, 29 и 30. У каждого квадратичного вычета ровно четыре квадратных корня.

Символ Лежандра

Символ Лежандра, L(*a*,*p*), определен, если *a* - это любое целое число, а *p* - простое число, большее, чем 2. Он равен 0, 1 или -1.

L(*a*,*p*) = 0, если а делится на p.

L(*a*,*p*) = 1, если а - квадратичный вычет по модулю *p*.

L(*a*,*p*) = -1, если а не является квадратичным вычетом по модулю *p*.

L(*a*,*p*) можно рассчитать следующим образом:

$L(a,p) = a^{(p-1)/2} \bmod p$

Или можно воспользоваться следующим алгоритмом:

1. Если а = 1, то L(*a*,*p*) = 1
2. Если а четно, то $L(a,p) = L(a/2,p) * (-1)^{(з^2-1)/8}$
3. Если а нечетно (и ≠ 1), то $L(a,p) = L(p \bmod a, p) * (-1)^{(a-1)(p-1)/4}$

Обратите внимание, что этот метод также является эффективным способом определить, является ли *a* квадратичным вычетом по модулю *p* (для простого числа *p*).

Символ Якоби

Символ Якоби, J(*a*,*n*), представляет собой обобщение символа Лежандра на составные модули, он определ яется для любого целого *a* и любого нечетного целого *n*. Функция удобна при проверке на простоту. Символ Як оби является функцией на множестве полученных вычетов делителей *n* и может быть вычислен по различным формулам [1412]. Вот один из сп особов:

Определение 1: J(*a*,*n*) определен, только если *n* нечетно.

Определение 2: J(0,*n*) = 0.

Определение 3: Если n - простое число, то символ Якоби J(a,n) = 0, если a делится на n.

Определение 4: Если n - простое число, то символ Якоби J(a,n) = 1, если a - квадратичный вычет по модулю n.

Определение 5: Если n - простое число, то символ Якоби J(a,n) = -1, если a не является квадратичным вычетом по модулю n.

Определение 6: Если n - составное число, то символ Якоби J(a,n) = J(a,p_1)* ... * J(a,p_m), где p_1, ... , p_m - это разложение n на простые сомножители.

Следующий алгоритм рекурсивно рассчитывает символ Якоби:

Правило 1: J(1,n) = 1

Правило 2: J(a*b,n) = J(a,n)* J(b,n)

Правило 3: J(2,n) =, если (n^2-1) /8 нечетно, и -1 в противном случае

Правило 4: J(a,n)= J((a mod n),n)

Правило 5: J(a, b_1*b_2) = J(a, b_1)* J(a, b_2)

Правило 6: Если наибольший общий делитель a и b = 1, а также a и b нечетны:

Правило 6a: J(a,b)= J(b, a), если (a - l)(b - 1)/4 четно

Правило 6b: J(a,b)= -J(b, a), если (a - l)(b - 1)/4 нечетно

Вот алгоритм на языке C:

```
/* Этот алгоритм рекурсивно вычисляет символ Якоби */
int jacobi(int a, int b) {
int g;
     assert(odd(b));
     if (a >= b) a %= b; /* по правилу 4 */
     if (a == 0) return 0; /* по определению 1 */
     if (a == 1) return 1; /* по правилу 1 */
     if (a < 0)
            if ((b-l)/2 % 2 == 0)
                   return jacobi(-a,b);
            else
                   return -jacobi(-a,b);
     if (a % 2 == 0) /* a четно */
            if (((b*b -1)/8) % 2 == 0)
                   return +jacobi(a/2,b);
            else
                   return -jacobi(a/2,b); /* по правилам 3 и 2 */
     g = gcd(a,b);
     assert(odd(a)); /* это обеспечивается проверкой (a % 2 == 0) */
     if (g == a) /* b делится на a */
            return 0; /* по правилу 5 */
     else if (g != 1)
            return jacobi(g,b)*jacobi(a/g,b); /* по правилу 2 */
     else if (((a-l)*(b-l)/4) % 2 == 0)
            return +jacobi(b,a); /* по правилу 6a */
     else
            return -jacobi(b,a); /* по правилу 6b */
}
```

Если заранее известно, что n - простое число, вместо использования предыдущего алгоритма просто вычислите a((n-1)/2) mod n, в этом случае J(a,n) эквивалентен символу Лежандра.

Символ Якоби нельзя использовать для определения того, является ли a квадратичным вычетом по модулю

n (если, конечно, n не является простым числом). Обратите внимание, что если J(a,n) = 1 и n - составное число, то утверждение, что a является квадратичным вычетом по модулю n, не обязательно будет истиной. Например:

J(7,143) = J(7,11)* J(7,13) = (-1)(-1) = 1

Однако не существует таких целых чисел x, что $x^2 \equiv 7 \pmod{143}$.

Целые числа Блюма

Если p и q - два простых числа, конгруэнтных 3 по модулю 4, то $n = pq$ иногда называют **целым числом Блюма**. Если n - это целое число Блюма, у каждого квадратичного вычета ровно четыре квадратных корня, один из которых также является квадратом - это главный квадратный корень. Например, главный квадратный корень 139 mod 437 - это 24. Остальные три корня - это 185, 252 и 413.

Генераторы

Если p - простое число, и g меньше, чем p, то g называется **генератором** по модулю p, если для каждого числа b от 1 до p - 1 существует некоторое число a, что $g^a \equiv b \pmod{p}$.

Иными словами, g является **примитивом** по отношению к p. Например, если p = 11, то 2 - это генератор по модулю 11:

$2^{10} = 1024 \equiv 1 \pmod{11}$

$2^1 = 2 \equiv 2 \pmod{11}$

$2^8 = 256 \equiv 3 \pmod{11}$

$2^2 = 4 \equiv 4 \pmod{11}$

$2^4 = 16 \equiv 5 \pmod{11}$

$2^9 = 512 \equiv 6 \pmod{11}$

$2^7 = 128 \equiv 7 \pmod{11}$

$2^3 = 8 \equiv 8 \pmod{11}$

$2^6 = 64 \equiv 9 \pmod{11}$

$2^5 = 32 \equiv 10 \pmod{11}$

Каждое число от 1 до 10 может быть представлено как $2^a \pmod{p}$. Для p = 11 генераторами являются 2, 6, 7 и 8. Другие числа не являются генераторами. Например, генератором не является число 3, потому что не сущ е-ствует решения для

$3^a \equiv 2 \pmod{11}$

В общем случае проверить, является ли данное число генератором, нелегко. Однако задщача упрощается, е с-ли известно разложение на множители для p - 1. Пусть q_1, q_2, ... , q_n - это различные простые множители p - 1. Чтобы проверить, является ли число g генератором по модулю p, вычислите

$g^{(p-1)/q} \bmod p$

для всех значений $q = q_1, q_2, \ldots, q_n$.

Если это число равно 1 для некоторого q, то g не является генератором. Если для всех значений q рассчитан-ное значение не равно 1, то g - это генератор.

Например, пусть p = 11. Простые множители p - 1 = 10 - это 2 и 5. Для проверки того, является ли число 2 генератором, вычислим:

$2^{(11-1)/5} \pmod{11} = 4$

$2^{(11-1)/2} \pmod{11} = 10$

Ни один из ответов не равен 1, поэтому 2 - это генератор.

Проверим, является генератором ли число 3:

$3^{(11-1)/5} \pmod{11} = 9$

$3^{(11-1)/2} \pmod{11} = 1$

Следовательно, 3 - это не генератор.

При необходимости обнаружить генератор по модулю p просто случайно выбирайте число от 1 до p - 1 и проверяйте, не является ли оно генератором. Генераторов достаточно, поэтому один из них вы, скорее всего, найдете быстро.

Вычисление в поле Галуа

Не тревожьтесь, все это мы уже делали. Если n - простое число или степень большого простого числа, то мы получаем то, что математики называют **конечным полем**. В честь этого мы используем p вместо n. В действительности этот тип конечного поля настолько замечателен, что математики дали ему собственное имя - **поле Галуа**, обозначаемое как GF(p). (В честь Эвариста Галуа, французского математика, жившего в девятнадцатом веке и успевшего значительно продвинуть теорию чисел, прежде чем в 20 лет он был убит на дуэли.)

В поле Галуа определены сложение, вычитание, умножение и деление на ненулевые элементы. Существует нейтральный элемент для сложения - 0 - и для умножения - 1. Для каждого ненулевого числа существует единственное обратное число (это не было бы так, если бы p не было бы простым числом). Выполняются коммутативный, ассоциативный и дистрибутивный законы.

Арифметика поля Галуа широко используется в криптографии. В нем работает вся теория чисел, поле содержит числа только конечного размера, при делении отсутствуют ошибки округления. Многие криптосистемы основаны на GF(p), где p - это большое простое число.

Чтобы еще более усложнить вопрос, криптографы также используют арифметику по модулю **неприводимых** многочленов степени n, коэффициентами которых являются целые числа по модулю q, где q - это простое число. Эти поля называются GF(qn). Используется арифметика по модулю $p(x)$, где $p(x)$ - это неприводимый многочлен степени n.

Математическая теория, стоящая за этим, выходит далеко за рамки этой книги, хотя я и опишу ряд криптосистем, использующих ее. Если вы хотите попробовать с неприводимыми многочленами, то GF(2^3) включает следующие элементы: 0, 1, x, $x + 1$, x^2, $x^2 + 1$, $x^2 + x$, $x^2 + x + 1$. Удобный для параллельной реализации алгоритм вычисления обратных значений в GF(2^n) приведен в [421].

При обсуждении полиномов термин "простое число" заменяется термином " неприводимый многочлен". Полином называется неприводимым, если его нельзя представить в виде двух других полиномов (конечно же, кроме 1 и самого полинома). Полином $x^2 + 1$ неприводим над целыми числами, а полином $x^3 + 2x^2 + x$ не является неприводимым, он может быть представлен как $x(x + 1)(x + 1)$.

Полином, который в данном поле является генератором, называется примитивным или базовым, все его коэффициенты взаимно просты. Мы снова вернемся к примитивным полиномам, когда будем говорить о сдвиговых регистрах с линейной обратной связью (см. раздел 16.2).

Вычисления в GF(2^n) могут быть быстро реализованы аппаратно с помощью сдвиговых регистров с линейной обратной связью. По этой причине вычисления над GF(2^n) часто быстрее, чем вычисления над GF(p). Так как возведение в степень в GF(2^n) гораздо эффективнее, то эффективнее и вычисление дискретных логарифмов [180, 181, 368, 379]. Дополнительную информацию об этом можно найти в [140].

Для поля Галуа GF(2^n) криптографы любят использовать в качестве модулей трехчлены $p(x) = x^n + x + 1$, так как длинная строка нулей между коэффициентами при x^n и x позволяет просто реализовать быстрое умножение по модулю [183]. Полином должен быть примитивным, в противном случае математика не будет работать. $x^n + x + 1$ примитивен для следующих значений n, меньших чем 1000 [1649, 1648]:

1, 3, 4, 6, 9, 15, 22, 28, 30, 46, 60, 63, 127, 153, 172, 303, 471, 532, 865, 900

Существуют аппаратные реализации GF(2^{127}), где $p(x) = x^{127} + x + 1$ [1631, 1632, 1129]. Эффективная архитектура аппаратуры возведения в степень для GF(2^n) рассматривается в [147].

11.4 Разложение на множители

Разложить число на множители - значит найти его простые сомножители.

10 = 2*5

60 = 2*2*3*5

252601 = 41*61*101

2113- 1 =3391*23279*65993*1868569*1066818132868207

Разложение на множители является одной из древнейших проблем теории чисел. Этот процесс несложен, но требует времени. Это пока остается так, но ряд сдвигов в этом искусстве все же произошел. Сегодня самым лучшим алгоритмом является:

Решето числового поля чисел (Number field sieve, **NFS**) [953] (см. также [952, 16, 279]). **Решето общего числового поля** - это самый быстрый из известных алгоритм для чисел размером 110 и более разрядов [472, 635]. В своем первоначальном виде он был непрактичен, но за последние несколько лет он был последовательно улучшен [953]. NFS все еще слишком нов, чтобы бить рекорды разложения на множители, но скоро все перем енится. Ранняя версия использовалась для разложения на множители девятого числа Ферма: 2512 + 1 [955,954].

Другие алгоритмы, вытесненные NFS:

Квадратичное решето (Quadratic sieve, **QS**) [1257, 1617, 1259]. Это самый быстрый из известных и чаще всего использовавшийся алгоритм для чисел, длина которых меньше 110 десятичных разрядов [440]. Более б ыстрая версия этого алгоритма называется множественным полиномиальным квадратичным решетом [1453, 302]. Самая быстрая версия называется двойной вариацией множественного полиномиального квадратичного решета с большим простым числом.

Метод эллиптической кривой (Elliptic curve method, **ECM**) [957, 1112, 1113]. Этот метод использовался для поиска не более, чем 43-разрядных множителей.

Алгоритм Монте-Карло Полларда (Pollard's Monte Carlo algorithm) [1254, 248]. (Этот алгоритм также приведен у Кнута в томе 2 [863].)

Алгоритм непрерывных дробей (Continued fraction algorithm). См. [1123, 1252, 863]. Этот алгоритм не подходит по времени выполнения.

Проверка делением (Trial division). Этот самый старый алгоритм разложения на множители состоит из проверки каждого простого числа, меньшего или равного ква дратному корню из раскладываемого числа.

В качестве хорошего введения в различные алгоритмы разложения на множители, кроме NFS, можно и с пользовать [251]. NFS лучше всего рассмотрен в [953]. Более старыми рпаботами являются [505, 1602, 1258]. Сведения о параллельном разложении на множители можно найти в [250].

Если число n на множители раскладывается, то эвристическое время выполнения самых быстрых вариантов QS асимптотически равно:

$$e^{(1+O(1))(\ln(n))^{1/2}(\ln((\ln(n)))^{1/2}}$$

NFS намного быстрее, оценка его эвристического времени выполнения:

$$e^{(1.923+O(1))(\ln(n))^{1/3}(\ln((\ln(n)))^{2/3}}$$

В 1970 году большой новостью стало разложение на множители 41-разрядного трудного числа [1123]. ("Трудным" является такое число, у которого нет маленьких множителей, и которое не обладает специальной формой, позволяющей упростить процесс.) Десять лет спустя разложение в два раз более длинного числа заняло лишь несколько часов на компьютере Cray [440].

В 1988 году Карл Померанс (Carl Pomerance), используя обычные СБИС, спроектировал устройство для ра зложения на множители [1259]. Размер числа, которое можно было разложить, зависел только от размеров ус тройства, которое так и не было построено.

В 1993 году с помощью квадратичного решета было разложено на множители 120-разрядное трудное число. Расчет, потребовавший 825 mips-лет, был выполнен за три месяца реального времени [463]. Другие результаты приведены в [504].

Сегодня для разложения на множители используются компьютерные сети [302, 955]. Для разложения 116_разрядного числа Аржат Ленстра (Arjen Lenstra) и Марк Манасс (Mark Manasse) в течение нескольких м есяцев использовали свободное время массива компьютеров, разбросанных по всему миру, - 400 mips-лет.

В марте 1994 года с помощью двойной вариации множественного полиномиального QS [66] командой м атематиков под руководством Ленстры было разложено на множители 129-разрядное (428-битовое) число. В ычисления выполнялись добровольцами в Internet - в течение восьми месяцев трудились 600 человек и 1600 ко мпьютеров, возможно, самый большой в истории многопроцессорный конгломерат. Трудоемкость вычислений была в диапазоне от 4000 до 6000 mips-лет. Компьютеры соединялись по электронной почте, передавая свои результаты в центральное хранилище, где выполнялся окончательный анализ. В этих вычислениях использов ались QS и теория пятилетней давности, NFS мог бы ускорить выполнение расчетов раз в десять [949]. В соо тветствии с [66]: "Мы делаем вывод, что широко используемые 512-битовые модули RSA могут быть вскрыты организацией, готовой потратить несколько миллионов долларов и подождать несколько месяцев." По оценкам авторов разложение 512-битового числа в 100 раз более трудоемко при использовании той же техники и только в 10 сложнее при использовании NFS и современной техники [949].

С целью развития искусства разложения на множители RSA Data Security, Inc. в марте 1991 года объявило о

программе RSA Factoring Challenge (состязание RSA по разложению на множители) [532]. Состязание состоит в разложении на множители ряда трудных чисел, каждое из которых является произведением двух простых чисел примерно одинакового размера. Каждое простое число было выбрано конгруэнтным 2 по модулю 3. Всего было предложено 42 числа, по одному числу в диапазоне от 100 до 500 разрядов с шагом 10 разрядов (плюс одно д о-полнительное, 129-разрядное число). К моменту написания этой книги RSA-100, RSA-110, RSA-120, и RSA-129 были разложены на множители, все с помощью QS. Следующим (с помощью NFS) может быть RSA-130, или чемпионы по разложению на множители сразу возьмутся за RSA -140.

Данная область развивается быстро. Технику разложения на множители трудно экстраполировать, так как невозможно предсказать развитие математической теории. До открытия NFS многие считали, что любой метод разложения на множители не может асимптотически быть быстрее QS. Они были неправы.

Предстоящее развитие NFS, по видимому, будет происходить в форме уменьшения константы: 1.923. Для ряда чисел специальной формы, таких как числа Ферма, константа приближается к 1.5 [955, 954]. Если бы для трудных чисел, используемых в сегодняшней криптографии, константу тоже можно было снизить до этого уровня, то 1024-битовые числа раскладывались бы на множители уже сегодня. Одним из способов уменьшить константу является обнаружение лучших способов представления чисел как полиномов с маленькими коэфф и-циентами. Пока еще проблема не изучалась достаточно эффективно, но возможно решающий успех уже близок [949].

Последние результаты программы RSA Factoring Challenge можно узнать, отправив запрос по электронной почте по адресу challenge-info@rsa.com.

Квадратные корни по модулю n

Если n - произведение двух простых чисел, то возможность вычислить квадратные корни по модулю n вычислительно эквивалентна возможности разложить число n на множители [1283, 35, 36, 193]. Другими словами, тот, кто знает простые множители числа n, может легко вычислить квадратные корни любого числа по модулю n, но для любого другого вычисление окажется таким же трудным, как и разложение на простые множители числа n.

11.5 Генерация простого числа

Для алгоритмов с открытыми ключами нужны простые числа. Их нужно множество для любой достаточно большой сети. Прежде, чем обсуждать математику генерации простого числа, я отвечу на несколько очевидных вопросов.

Если каждому понадобится свое простое число, не иссякнет ли у нас запас? Нет. В действительности сущес т-вует приблизительно 10151 простых чисел длино1 до 512 бит включительно. Для чисел, близких n, вероятность того, что случайно выбранное число окажется простым, равна $1/\ln n$. Поэтому полное число простых чисел, меньших n, равно $n/(\ln n)$. Во вселенной всего 10^{77} атомов. Если бы для каждого атома во вселенной с начала времен каждую микросекунду требовался бы миллиард простых чисел, понадобилось бы только 10^{109} простых чисел, осталось бы еще примерно 10^{151} простых чисел.

Что если два человека случайно выберут одно и то же простое число? Этого не случится. При выборе из 10151 простых чисел вероятность совпадения выбора значительно меньше, чем вероятность, что ваш компь ю-тер случайно вспыхнет в тот самый момент, когда вы выиграете в лотерею.

Если кто-то создаст базу данных всех простых чисел, не сможет ли он использовать эту базу данных для вскрытия алгоритмов с открытыми ключами? Нет. Если бы вы хранили один гигабайт информации на устро й-стве, весящем один грамм, то перечень простых чисел размером до 512 бит включительно весил бы столько, что масса хранилища превысила бы предел Чандрасекара, и оно сколлапсировало бы в черную дыру ... в любом случае вы не сможете извлечь данные.

Но если так трудоемко разложение на множители, как может быть простой генерация простых чисел? Фокус в том, что ответить "да" или "нет" на вопрос "Является ли число n простым?" гораздо проще, чем ответить на более сложный вопрос "Каковы множители n?"

Генерация случайных чисел с последующей попыткой разложения их на множители - это неправильный сп о-соб поиска простых чисел. Существуют различные вероятностные проверки на простоту чисел, определяющие, является ли число простым, с заданной степенью достоверности. При условии, что эта "степень достоверности" достаточна велика, такие способы проверки достаточно хороши. Я слышал, что простые числа, генерированные таким образом называются "промышленно простыми числами": эти числа вероятно являются простыми с ко нтролируемой возможностью ошибки.

Предположим, что одна проверка из 2^{50} - ошибочна. Это означает, что с вероятностью $1/10^{15}$ проверка объя-вит простым составное число. (Простое число никогда не будет объявлено составным при проверке.) Если по

какой-то причине понадобится большая достоверность простоты числа, уровень ошибки можно понизить. С другой стороны, если вы установите вероятность того, что число является составным, в 300 миллионов раз меньшей, чем вероятность выиграть главный приз в государственной лотерее, вы можете больше об этом не волноваться.

Обзоры недавних исследований в этой области можно найти в [1256, 206]. Другими важными работами я вляются [1490, 384, 11, 19, 626, 651, 911].

Solovay-Strassen

Роберт Соловэй (Robert Solovay) и Фолькер Штрассен (Volker Strassen) разработали алгоритм вероятностной проверки простоты числа [1490]. Для проверки простоты числа p этот алгоритм использует символ Якоби:

(1) Выберите случайно число a, меньшее p.

(2) Если НОД(a,p) (1, то p не проходит проверку и является составным.

(3) Вычислите $j = a(p\text{-}1)/2 \bmod p$.

(4) Вычислите символ Якоби J(a,p).

(5) Если $j \neq$ J(a,p), то число p наверняка не является простым.

(6) Если $j =$ J(a,p), то вероятность того, что число p не является простым, не больше 50 процентов.

Число a, которое не показывает, что p наверняка не является простым числом, называется свидетелем. Если p - составное число, вероятность случайного числа a быть свидетелем не ниже 50 процентов. Повторите эту проверку t раз с t различными значениями a. Вероятность того, что составное число преодолеет все t проверок, не превышает $1/2^t$.

Lehmann

Другой, более простой тест был независимо разработан Леманном (Lehmann) [903]. Вот последовательность действий при проверке простоты числа p:

(1) Выберите случайно число a, меньшее p.

(2) Вычислите $a^{(p\text{-}1)/2} \bmod p$.

(3) Если $a^{(p\text{-}1)/2} \neq 1$ или -1 (mod p), то p не является простым.

(4) Если $a^{(p\text{-}1)/2} \equiv 1$ или -1 (mod p), то вероятность того, что число p не является простым, не больше 50 процентов.

И снова, вероятность того, что случайное число a будет свидетелем составной природы числа p, не меньше 50 процентов. Повторите эту проверку t раз. Если результат вычислений равен 1 или -1, но не всегда равен 1, то p является простым числом с вероятностью ошибки $1/^{2t}$.

Rabin-Miller

Повсеместно используемым является простой алгоритм, разработанный Майклом Рабином (Michael Rabin), частично основанным на идеях Гэри Миллера [1093, 1284]. По сути, это упрощенная версия алгоритма, рек омендованного в предложении DSS proposal [1149, 1154].

Выберите для проверки случайное число p. Вычислите b - число делений p - 1 на 2 (т.е., 2^b - это наибольшая степень числа 2, на которое делится p - 1). Затем вычислите m, такое что $p = 1 + 2^b * m$.

(1) Выберите случайное число a, меньшее p.

(2) Установите $j = 0$ и $z = am \bmod p$.

(3) Если $z = 1$ или если $z = p$ - 1, то p проходит проверку и может быть простым числом.

(4) Если $j > 0$ и $z = 1$, то p не является простым числом.

(5) Установите $j = j + 1$. Если $j < b$ и $z($ p - 1, установите $z = z^2 \bmod p$ и вернитесь на этап (4). Если $z = p$ - 1, то p проходит проверку и может быть простым числом.

(6) Если $j = b$ и $z \neq p$ - 1, то p не является простым числом.

В этом тесте вероятность прохождения проверки составным числом убывает быстрее, чем в предыдущих. Гарантируется, что три четверти возможных значений a окажутся свидетелями. Это означает, что составное число проскользнет через t проверок с вероятностью не большей $(1/4)^t$, где t - это число итераций. На самом деле и эти оценки слишком пессимистичны. Для большинства случайных чисел около 99.9 процентов возмо ж-

ных значений a являются свидетелями [96].

Существуют более точные оценки [417]. Для n-битового кандидата в простые числа (где n больше 100), вероятность ошибки в одном тесте меньше, чем $4n2^{(k/2)^{1/2}}$. И для 256-битового n вероятность ошибки в шести тестах меньше, чем $1/2^{51}$. Дополнительную теорию можно найти в [418].

Практические соображения

В реальных приложениях генерация простых чисел происходит быстро.

(1) Сгенерируйте случайное n-битовое число p.

(2) Установите старший и младший биты равными 1. (Старший бит гарантирует требуемую длину простого числа, а младший бит обеспечивает его нечетность.)

(3) Убедитесь, что p не делится на небольшие простые числа: 3, 5, 7, 11, и т.д. Во многих реализациях проверяется делимость p на все простые числа, меньшие 256. Наиболее эффективной является проверка на делимость для всех простых чисел, меньших 2000 [949]. Это может быть эффективно выполнено с помощью колеса [863].

(4) Выполните тест Rabin-Miller для некоторого случайного a. Если p проходит тест, сгенерируйте другое случайное a и повторите проверку. Выбирайте небольшие значения a для ускорения вычислений. Выполните пять тестов [651]. (Одного может показаться достаточным, но выполните пять.) Если p не проходит одной из проверок, сгенерируйте другое p и попробуйте снова.

Иначе, можно не генерировать p случайным образом каждый раз, но последовательно перебирать числа, начиная со случайно выбранного до тех пор, пока не будет найдено простое число.

Этап (3) не является обязательным, но это хорошая идея. Проверка, что случайное нечетное p не делится на 3, 5 и 7 отсекает 54 процента нечетных чисел еще до этапа (4). Проверка делимости на все простые числа, меньшие 100, убирает 76 процентов нечетных чисел, проверка делимости на все простые числа, меньшие 256, убирает 80 процентов нечетных чисел. В общем случае, доля нечетных кандидатов, которые не делятся ни на одно простое число, меньшее n, равна $1.12/\ln n$. Чем больше проверяемое n, тем больше предварительных вычислений нужно выполнить до теста Rabin-Miller.

Одна из реализаций этого метода на Sparc II способна находить 256-битовые простые числа в среднем за 2.8 секунды, 512-битовые простые числа - в среднем за 24.0 секунды, 768-битовые простые числа - в среднем за 2.0 минуты, а 1024-битовые простые числа - в среднем за 5.1 минуты [918].

Сильные простые числа

Если n - произведение двух простых чисел, p и q, то может понадобиться использовать в качестве p и q **сильные простые числа**. Такие простые числа обладают рядом свойств, которые усложняют разложение произведения n определенными методами разложения на множители. Среди таких свойств были предложены [1328, 651]:

Наибольший общий делитель $p - 1$ и $q - 1$ должен быть небольшим.

И $p - 1$, и $q - 1$ должны иметь среди своих множителей большие простые числа, соответственно p' и q'.

И $p' - 1$, и $q' - 1$ должны иметь среди своих множителей большие простые числа.

И $p + 1$, и $q + 1$ должны иметь среди своих множителей большие простые числа.

И $(p - 1)/2$, и $(q - 1)/2$ должны быть простыми [182]. (Обратите внимание, при выполнении этого условия выполняются и два первых.)

Насколько существенно применение именно сильных простых чисел, остается предметом продолжающихся споров. Эти свойства были разработаны, чтобы затруднить выполнение ряда старых алгоритмов разложения на множители. Однако самые быстрые алгоритмы одинаково быстры при разложении на множители любых чисел, как удовлетворяющих приведенным условиям, так и нет [831].

Я против специальной генерации сильных простых чисел. Длина простых чисел гораздо важнее их структуры. Более того, сама структура уменьшает случайность числа и может снизить устойчивость системы.

Но все может измениться. Могут быть созданы новые методы разложения на множители, которые лучше работают с числами, обладающими определенными свойствами. В этом случае снова могут потребоваться сильные простые числа. Заглядывайте в журналы по теоретической математике.

11.6 Дискретные логарифмы в конечном поле

В качестве другой однонаправленной функции в криптографии часто используется возведение в степень по модулю. Легко вычислить:

$a^x \bmod n$

Задачей, обратной возведению в степень по модулю, является поиск дискретного логарифма. А это уже н елегкая задача:

Найти x, для которого $a^x \equiv b \pmod n$.

Например:

Если $3^x \equiv 15 \bmod 17$, то $x = 6$

Решения существуют не для всех дискретных логарифмов (помните, речь идет только о целочисленных р ешениях). Легко заметить, что следующее уравнение не имеет решений

$3^x \equiv 7 \pmod{13}$

Еще сложнее решать эту задачу для 1024-битовых чисел.

Вычисление дискретных логарифмов в конечной группе

Криптографы интересуются дискретными логарифмами следующих трех групп:

— Мультипликативная группа полей простых чисел: GF(p)

— Мультипликативная группа конечных полей степеней 2: GF(2^n)

— Группы эллиптической кривой над конечными полями F: EC(F)

Безопасность многих алгоритмов с открытыми ключами основана на задаче поиска дискретных логарифмов, поэтому эта задача была глубоко изучена. Хороший подробный обзор этой проблемы и ее наилучшие решения на соответствующий момент времени можно найти в [1189, 1039]. Лучшей современной статьей на эту тему является [934].

Если p является простым числом и используется в качестве модуля, то сложность поиска дискретных лог арифмов в GF(p) по существу соответствует разложению на множители числа n того же размера, где n - это произведение двух простых чисел приблизительно равной длины [1378,934]. То есть:

$$e^{(1+O(1))(\ln(n))^{1/2}(\ln((\ln(n)))^{1/2}}$$

Решето числового поля быстрее, оценка его эвристического времени выполнения:

$$e^{(1.923+O(1))(\ln(n))^{1/3}(\ln((\ln(n)))^{2/3}}$$

Стивен Полиг (Stephen Pohlig) и Мартин Хеллман нашли способ быстрого вычисления дискретных лог арифмов в GF(p) при условии, что p - 1 раскладывается на малые простые множители [1253]. По этой причине в криптографии используются только такие поля, для которых p - 1 обладает хотя бы одним большим простым множителем. Другой алгоритм [14] вычисляет дискретных логарифм со скоростью, сравнимой с разложением на множители, он был расширен на поля вида GF(p^n) [716]. Этот алгоритм был подвергнут критике в [727] по ряду теоретических моментов. В других статьях [1588] можно увидеть, насколько на самом деле трудна пр облема в целом.

Вычисление дискретных логарифмов тесно связано с разложением на множители. Если вы можете решить проблему дискретного логарифма, то вы можете и разложить на множители. (Истинность обратного никогда не была доказана.) В настоящее время существует три метода вычисления дискретных логарифмов в поле простого числа [370, 934, 648]: линейное решето, схема целых чисел Гаусса и решето числового поля.

Предварительное, объемное вычисление для поля должно быть выполнено только один раз. Затем, быстро можно вычислять отдельные логарифмы. Это может серьезно уменьшить безопасность систем, основанных на таких полях. Важно, чтобы различные приложения использовали различные поля простых чисел. Хотя нескол ько пользователей одного приложения могут применять общее поле.

В мире расширенных полей исследователями не игнорируются и GF(2^n). Алгоритм был предложен в [727]. Алгоритм Копперсмита (Coppersmith) позволяет за приемлемое время находить дискретные логарифмы в таких полях как GF(2^{127}) и делает принципиально возможным их поиск в полях порядка GF(2^{400}) [368]. В его основе лежит [180]. У этого алгоритма очень велика стадия предварительных вычислений, но во всем остальном он хорош и эффективен. Реализация менее эффективной версии этого же алгоритма после семи часов предвар ительных вычислений тратила на нахождение каждого дискретного логарифма в поле GF(2^{127}) лишь несколько

секунд [1130, 180]. (Это конкретное поле, когда-то использовавшееся в некоторых криптосистемах [142, 1631, 1632], не является безопасным.) Обзор некоторых из этих результатов можно найти в [1189, 1039].

Позднее были выполнены предварительные вычисления для полей GF(2^{227}), GF(2^{313}) и GF(2^{401}), удалось значительно продвинуться и для поля GF(2^{503}). Эти вычисления проводились на nCube-2, массивном параллельном компьютере с 1024 процессорами [649, 650]. Вычисление дискретных логарифмов в поле GF(2^{593}) все еще находится за пределами возможного.

Как и для нахождения дискретных логарифмов в поле простого числа, для вычисления дискретных логарифмов в полиномиальном поле также требуется один раз выполнить предварительные вычисления. Тахер Эль-Джамаль (Taher ElGamal) [520] приводит алгоритм вычисления дискретных логарифмов в поле GF(p^2).

Глава 12 Стандарт шифрования данных DES (Data Encryption Standard)

12.1 Введение

Стандарт шифрования данных DES (Data Encryption Standard), который ANSI называет Алгоритмом ши фрования данных DEA (Data Encryption Algorithm), а ISO - DEA-1, за 20 лет стал мировым стандартом. Хотя на нем и появился налет старости, он весьма прилично выдержал годы криптоанализа и все еще остается безопа сным по отношению ко всем врагам, кроме, возможно, самых могущественных.

Разработка стандарта

В начале 70-х годов невоенные криптографические исследования были крайне редки. В этой области почти не публиковалось исследовательских работ. Большинство людей знали, что для своих коммуникаций военные используют специальную аппаратуру кодирования, но мало кто разбирался в криптографии как в науке. Заме тными знаниями обладало Агентство национальной безопасности (National Security Agency, NSA), но оно даже не признавало публично своего собственного существования.

Покупатели не знали, что они покупают. Многие небольшие компании изготавливали и продавали крипт ографическое оборудование, преимущественно заокеанским правительствам. Все это оборудование отличалось друг от друга и не могло взаимодействовать. Никто не знал, действительно ли какое-либо из этих устройств безопасно, не существовало независимой организации, которая засвидетельствовала бы безопасность. Как гов орилось в одном из правительственных докладов [441]:

> Влияние соответствующего изменения ключей и принципов работы на реальную мощь аппаратуры шифров а ния/дешифрирования было (и фактически осталось) неизвестным почти всем покупателям, и было очень трудно принимать обоснованные решения о генерации ключей, правильном диалоговом или автономном режиме, и т.д., которые отвечали бы потребностям покупателей в безопасности.

В 1972 году Национальное бюро стандартов (National Bureau of Standards, NBS), теперь называющееся Н ациональным институтом стандартов и техники (National Institute of Standards and Technology, NIST), выступило инициатором программы защиты линий связи и компьютерных данных. Одной из целей этой программы была разработка единого, стандартного криптографического алгоритма. Этот алгоритм мог бы быть проверен и се ртифицирован, а использующие его различные криптографические устройства могли бы взаимодействовать. Он мог бы, к тому же, быть относ ительно недорогим и легко доступным.

15 мая 1973 года в *Federal Register* NBS опубликовало требования к криптографическому алгоритму, кот орый мог бы быть принят в качестве стандарта. Было пр иведено несколько критериев оценки проекта:

— Алгоритм должен обеспечивать высокий уровень безопасности.

— Алгоритм должен быть полностью определен и легко понятен.

— Безопасность алгоритма должна основываться на ключе и не должна зависеть от сохранения в тайне с амого алгоритма.

— Алгоритм должен быть доступен всем пользователям.

— Алгоритм должен позволять адаптацию к различным применениям.

— Алгоритм должен позволять экономичную реализацию в виде электронных приборов.

— Алгоритм должен быть эффективным в использовании.

— Алгоритм должен предоставлять возможности проверки.

— Алгоритм должен быть разрешен для экспорта.

Реакция общественности показала, что к криптографическому стандарту существует заметный интерес, но опыт в этой области чрезвычайно мал. Ни одно из предложений не удовлетворяло предъявленным требованиям.

27 августа 1972 года в *Federal Register* NBS опубликовало повторное предложение. Наконец, у Бюро появи лся подходящий кандидат: алгоритм под именем Люцифер, в основе которого лежала разработка компании IBM, выполненная в начале 70-х (см. раздел 13.1). В IBM существовала целая команда криптографов, работавшая в Кингстоне (Kingston) и Йорктаун Хайтс (Yorktown Heights), в которую входили Рой Адлер (Roy Adler), Дон Копперсмит (Don Coppersmith), Хорст Фейстель (Horst Feistel), Эдна Кроссман (Edna Crossman), Алан Конхейм (Alan Konheim), Карл Майер (Carl Meyer), Билл Ноц (Bill Notz), Линн Смит (Lynn Smith), Уолт Тачмен (Walt Tuchman) и Брайант Такерман (Bryant Tuckerman).

Несмотря на определенную сложность алгоритм был прямолинеен. Он использовал только простые логич е-

ские операции над небольшими группами битов и мог быть довольно эффективно реализован в аппаратуре.

NBS попросило NSA помочь оценить безопасность алгоритма и определить, подходит ли он для использов ания в качестве федерального стандарта. IBM уже получила патент [514], но желала сделать свою интеллект уальную собственность доступной для производства, реализации и использования другими компаниями. В конце концов, NBS и IBM выработали соглашение, по которому NBS получало неисключительную, бесплатную л ицензию изготавливать, использовать и продавать устройства, реализующие этот алгоритм.

Наконец, 17 марта 1975 года в *Federal Register* NBS опубликовало и подробности алгоритма, и заявление IBM о предоставлении неисключительной, бесплатной лицензии на алгоритм, а также предложило присылать комментарии по поводу данного алгоритма [536]. В другой заметке в *Federal Register*, 1 августа 1975 года, различным организациям и широкой публике снова предлагалось прокомментировать предложенный алгоритм.

И комментарии появились [721, 497, 1120). Многие настороженно относились к участию "невидимой руки" NSA в разработке алгоритма. Боялись, что NSA изменит алгоритм, вставив в него потайную дверцу. Жалов ались, что NSA уменьшило длину ключей с первоначальных 128 битов до 56 (см. раздел 13.1). Жаловались на внутренние режимы работы алгоритма. Многие соображения NSA стали ясны и понятны в начале 90-х, но в 70-х они казались таинственными и тревожными.

В 1976 году NBS провело два симпозиума по оценке предложенного стандарта. На первом обсуждались м атематика алгоритма и возможность потайной дверцы [1139]. На втором - возможности увеличения длины ключа алгоритма [229]. Были приглашены создатели алгоритма, люди, оценивавшие алгоритм, разработчики аппар атуры, поставщики, пользователи и критики. По всем отчетам симпозиумы были весьма ожи вленными [1118].

Несмотря на критику Стандарт шифрования данных DES 23 ноября 1976 года был принят в качестве фед ерального стандарта [229] и разрешен к использованию на всех несекретных правительственных коммуникац иях. Официальное описание стандарта, FIPS PUB 46, "Data Encryption Standard", было опубликовано 15 января 1977 года и вступило в действие шестью месяцами позже [1140]. FIPS PUB 81, " Modes of DES Operation" (Режимы работы DES), было опубликовано в 1980 году [1143]. FIPS PUB 74, "Guidelines for Implementing and Using the NBS Data Encryption Standard" (Руководство по реализации и использованию Стандарта шифрования данных NBS), появилось в 1981 году [1142]. NBS также опубликовало FIPS PUB 112, специфицируя DES для шифрования паролей [1144], и FIPS PUB 113, специфицируя DES для проверки подлинности компьютерных данных [1145]. (FIPS обозначает Federal Information Processing Standard.)

Эти стандарты были беспрецедентными. Никогда до этого оцененный NSA алгоритм не был опубликован. Возможно эта публикация была следствием непонимания, возникшего между NSA и NBS. NSA считало, что DES будет реализовываться только аппаратно. В стандарте требовалась именно аппаратная реализация, но NBS опубликовало достаточно информации, чтобы можно было создать и программную реализацию DES. Не для печати NSA охарактеризовало DES как одну из своих самых больших ошибок. Если бы Агентство предполаг ало, что раскрытые детали позволят писать программное обеспечение, оно никогда бы не согласилось на это. Для оживления криптоанализа DES сделал больше, чем что-либо другое. Теперь для исследования был доступен алгоритм, который NSA объявило безопасным. Не случайно следующий правительственный стандарт алгори тма, Skipjack (см. раздел 13.12.), был засекречен.

Принятие стандарта

Американский национальный институт стандартов (American National Standards Institute, ANSI) одобрил DES в качестве стандарта для частного сектора в 1981 году (ANSI X3.92.) [50], назвав его Алгоритмом шифр ования данных (Data Encryption Algorithm, DEA). ANSI опубликовал стандарт режимов работы DEA (ANSI X3.106) [52], похожий на документ NBS, и стандарт для шифрования в сети, использующий DES (ANSI X3.105) [51].

Две другие группы внутри ANSI, представляющие банковские операции при розничной и оптовой торговле, разработали свои стандарты на основе DES. Банковские операции при розничной торговле включают транза кции между финансовыми организациями и отдельными личностями, а банковские операции при оптовой то рговле включают транзакции между финансовыми организациями.

Рабочая группа ANSI по безопасности финансовых организаций при розничной торговле разработала ста ндарт для управления PIN-кодами и их безопасностью (ANSI X9.8) [53] и другой использующий DES стандарт для проверки подлинности финансовых сообщений о розничных продажах (ANSI X9.19) [56]. Эта группа ра зработала и проект стандарта для безопасного распределения ключей (ANSI X9.2.4) [58].

Рабочая группа ANSI по безопасности финансовых организаций при оптовой торговле разработала свой со бственный набор стандартов для проверки подлинности сообщений (ANSI X9.9) [54], управления ключами (ANSIX9.17) [55, 1151], шифрования (ANSIX9.2.3) [57] и безопасной проверки подлинности личностей и узлов (ANSI X9.26) [59].

Американская ассоциация банкиров разрабатывает необязательные стандарты для финансовой индустрии.

Они опубликовали стандарт, рекомендующий DES для шифрования [1], и другой стандарт для управления криптографическими ключами [2].

До появления в 1987 году Акта о компьютерной безопасности (Computer Security Act) the за разработку ф едеральных стандартов в области телекоммуникаций отвечала Администрация общих служб (General Services Administration, CSA), а с этого момента ответственность перешла к NIST. CSA опубликовала три стандарта, использующих DES: два для требований к общей безопасности и возможности взаимодействия (Федеральный стандарт 1026 [662] и Федеральный стандарт 1027 [663]) и один для факс-аппаратов Group 3 (Федеральный стандарт 1028 [664]).

Казначейство издало стратегические директивы, требующие, чтобы подлинность всех сообщений о переводе электронных финансов удостоверялась с помощью DES [468, 470]. Оно также разработало основанный на DES критерий, которому должны удовлетворять все устройства проверки подлинности [469].

ISO сначала проголосовала за введение DES, называемого в ее интерпретации DEA-1, в качестве междун ародного стандарта, а затем приняла решение не заниматься стандартизацией криптографии. Однако в 1987 году группа ISO, занимающаяся международными стандартами в области оптовой торговли, применила DES в ме ждународном стандарте проверки подлинности [758] и для управления ключами [761]. DES также используется в качестве австралийского банковского стандарта [1497].

Проверка и сертификация оборудования DES

Частью стандарта DES является проверка NIST реализаций DES. Эта проверка подтверждает, что реализ ация соответствует стандарту. До 1994 года NIST проверял только аппаратные и программно-аппаратные реал изации - пока стандарт запрещал программные реализации. На март 1995 года 73 различных реализации были признаны соответствующими стандарту.

NIST также разработал программу сертификации устройств проверки подлинности на соответствие ANSI X9.9 и FIPS 113. На март 1995 года было сертифицировано 33 различных продукта. Казначейство использует свою собственную дополнительную процедуру сертификации. У NIST также есть программа проверки аппар атуры на соответствие ANSI X9.17 для управления ключами при оптовой торговле [1151], На март 1995 года было сертифицировано четыре продукта.

1987

В стандарте DES было оговорено, что он будет пересматриваться каждые пять лет. В 1983 DES был повто рно сертифицирован без всяких проблем. 6 марта 1987 года в Federal Register NBS попросило прокомментир овать предложение на следующие пять лет. NBS предложило на обсуждение следующие три альтернативы [1480, 1481]: вновь подтвердить стандарт на следующие пять лет, отказаться от стандарта или пересмотреть примен имость стандарта.

NBS и NSA пересмотрели стандарт. В этот раз NSA было задействовано в большей степени. Благодаря по дписанной Рейганом директиве NSDD-145 NSA получило право вето по отношению к деятельности NBS в обла сти криптографии. Первоначально NSA объявило, что оно не сертифицирует стандарт повторно. Проблема была не в том, что DES действительно был взломан, и даже не в том, что он, может быть, был взломан. По видим ому, предполагалось, что он вот-вот будет взломан.

Само по себе NSA предложило Программу коммерческой подписи COMSEC (Commercial COMSEC Endorsement Program, CCEP), которая по сути представляла собой набор алгоритмов для замены DES [85]. Эти разработанные NSA алгоритмы не были опубликованы и были доступны только в виде защищенных от взлома СБИС (см. раздел 25.1).

Это предложение не было принято. Было отмечено, что DES широко используется в бизнесе (особенно в ф инансах), и что приемлемой альтернативы не существует. Отказ от стандарта оставил бы многие организации без защиты данных. После длительных споров DES был вновь утвержден в качестве правительственного стандарта США до 1992 года [1141]. NBS решило, что DES никогда больше не будет сертифицирован снова [1480].

1993

Никогда не говори "никогда". В 1992 году альтернативы алгоритму DES все еще не было. NBS, называемый теперь NIST, снова в Federal Register предложило пр окомментировать DES [540]:

> Цель этого предложения состоит в том, чтобы объявить о предстоящем оценивании адекватности стандарта задаче защ иты компьютерных данных на современном уровне. Промышленности и широкой публике предлагаются три следующих вар ианта решения для FIPS 46-1. Комментарии должны содержать стоимость (последствия) и преимущества этих вариа нтов:
>
> —Повторно принять стандарт на следующие пять (5) лет. Национальный институт стандартов и технологии продолжит сертификацию аппаратуры, реализующей стандарт. FIPS 46-1 будет и дальше оставаться единственным признанным м етодом защиты несекретных компьютерных данных.
>
> —Отказаться от стандарта. Национальный институт стандартов и технологии больше не будет поддерживать стандарт.

> Организации могут продолжать использовать существующую аппаратуру, реализующую стандарт. Заменяя DES, NIST издаст другие стандарты.
> —Пересмотреть положения стандарта о применимости и/или провести ревизию реализации. Такая ревизия должна включать изменения стандарта, позволяющие использовать как аппаратные, так программные и реализации DES, и с-пользовать DES итеративно в определенных приложениях, использовать альтернативные алгоритмы, признанные и зар е-гистрированные NIST.

Срок принятия предложений истек 10 декабря 1992 года. Согласно Рэймонду Каммеру (Raymond Kammer), в то время директору NIST [812]:

> В прошлом году NIST формально предложило присылать комментарии по поводу повторной сертификации DES. Ра с-смотрев присланные предложения и другие технические источники, я собираюсь рекомендовать министру торговли, чтобы он повторно сертифицировал DES еще на пять лет. Я также собираюсь предложить министру, чтобы, объявляя о повторной сертификации, мы сформулировали наши намерения рассмотреть в течение этих пяти лет возможные альтернативы. Делая подобное заявление, мы надеемся дать людям возможность высказаться по поводу предстоящих технологических изменений. В то же время, нам нужно учитывать большое количество систем, использующих этот одобренный ста ндарт.

Несмотря на то, что Управление оценки технологий ссылалось на слова работавшего в NIST Денниса Бра н-стида (Dennis Branstead) от том, что полезное время жизни DES закончится в конце 90-х [1191], алгоритм был сертифицирован повторно на следующие пять лет [1150]. Наконец было разрешено сертифицировать и пр о-граммные реализации DES. Хотелось бы знать, что случится в 1998 году?

12.2 Описание DES

DES представляет собой блочный шифр, он шифрует данные 64-битовыми блоками. С одного конца алг о-ритма вводится 64-битовый блок открытого текста, а с другого конца выходит 64-битовый блок шифротекста. DES является симметричным алгоритмом: для шифрования и дешифрирования используются одинаковые алг о-ритм и ключ (за исключением небольших различий в использовании ключа).

Длина ключа равна 56 битам. (Ключ обычно представляется 64-битовым числом, но каждый восьмой бит используется для проверки четности и игнорируется. Биты четности являются наименьшими значащими битами байтов ключа.) Ключ, который может быть любым 56-битовым числом, можно изменить в любой момент вр е-мени. Ряд чисел считаются слабыми ключами, но их можно легко избежать. Безопасность полностью опред е-ляется ключом.

На простейшем уровне алгоритм не представляет ничего большего, чем комбинация двух основных методов шифрования: смещения и диффузии. Фундаментальным строительным блоком DES является применение к те к-сту единичной комбинации этих методов (подстановка, а за ней - перестановка), зависящей от ключа. Такой блок называется этапом. DES состоит из 16 этапов, одинаковая комбинация методов применяется к открытому тексту 16 раз (см. 11-й).

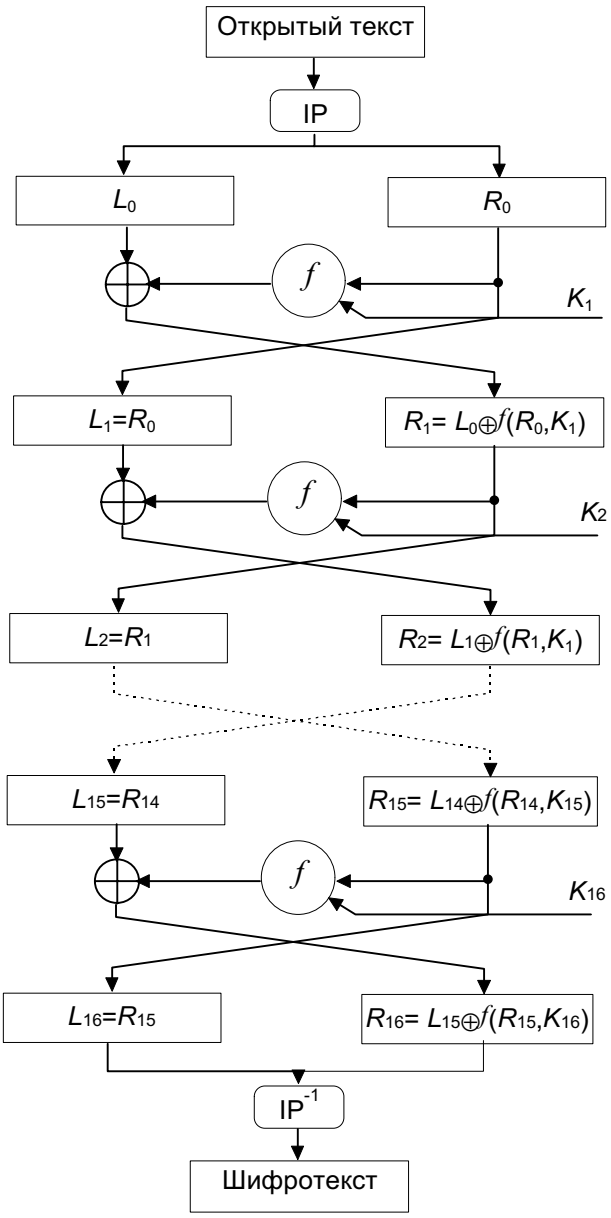


Рис. 12-1. DES.

Алгоритм использует только стандартную арифметику 64-битовых чисел и логические операции, поэтому он легко реализовывался в аппаратуре второй половины 70-х. Изобилие повторений в алгоритме делает его ид еальным для реализации в специализированной микросхеме. Первоначальные программные реализации были довольно неуклюжи, но сегодняшние программы намного лучше.

Схема алгоритма

DES работает с 64-битовым блоком открытого текста. После первоначальной перестановки блок разбивается на правую и левую половины длиной по 32 бита. Затем выполняется 16 этапов одинаковых действий, называ емых функцией f, в которых данные объединяются с ключом. После шестнадцатого этапа правая и левая полов ины объединяются и алгоритм завершается заключительной перестановкой (обратной по отношению к первон ачальной).

На каждом этапе (см. 10-й) биты ключа сдвигаются, и затем из 56 битов ключа выбираются 48 битов. Пр авая половина данных увеличивается до 48 битов с помощью перестановки с расширением, объединяется п осредством XOR с 48 битами смещенного и переставленного ключа, проходит через 8 S-блоков, образуя 32 н овых бита, и переставляется снова. Эти четыре операции и выполняются функцией f. Затем результат функции f объединяется с левой половиной с помощью другого XOR. В итоге этих действий появляется новая правая п оловина, а старая правая половина становится новой левой. Эти действия повторяются 16 раз, образуя 16 этапов DES.

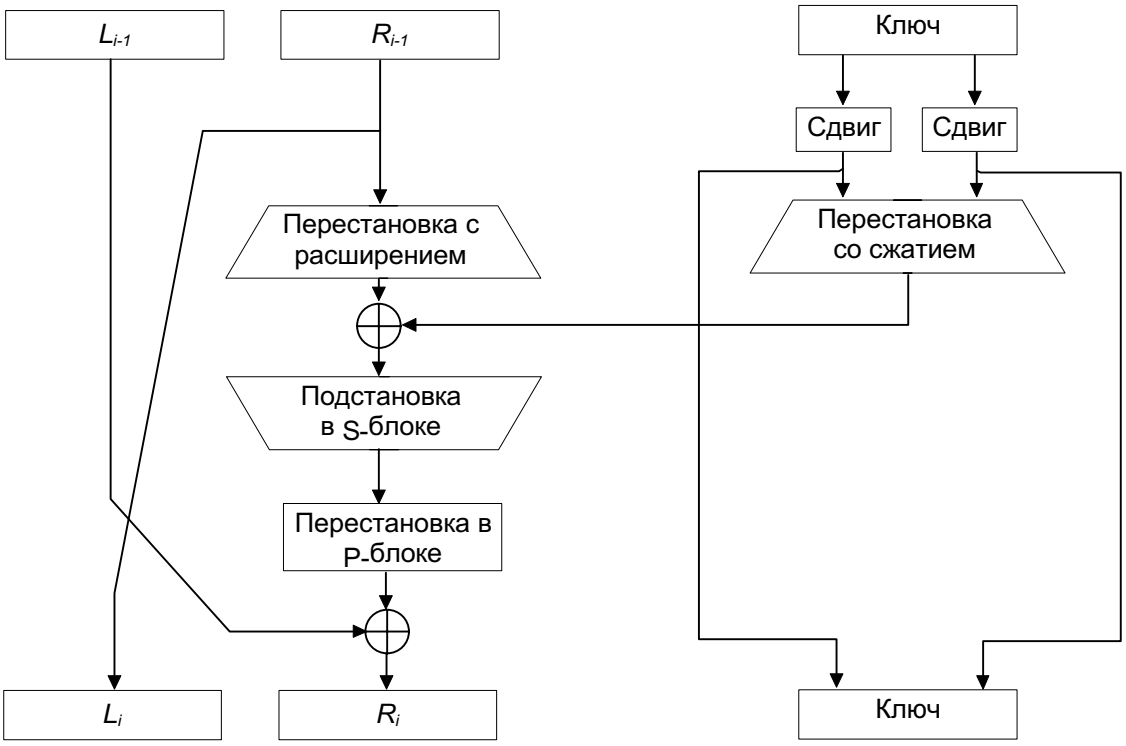


Рис. 12-2. Один этап DES.

Если B_i - это результат i-ой итерации, L_i и R_i - левая и правая половины B_i, K_i - 48-битовый ключ для этапа i, а f - это функция, выполняющие все подстановки, перестановки и XOR с ключом, то этап можно представить как:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i)$

Начальная перестановка

Начальная перестановка выполняется еще до этапа 1, при этом входной блок переставляется, как показано в 11-й. Эту и все другие таблицы этой главы надо читать слева направо и сверху вниз. Например, начальная пер е-становка перемещает бит 58 в битовую позицию 1, бит 50 - в битовую позицию 2, бит 42 - в битовую позицию 3, и так далее.

Табл. 12-1.
Начальная перестановка

58,	50,	42,	34,	26,	18,	10,	2,	60,	52,	44,	36,	28,	20,	12,	4,
62,	54,	46,	38,	30,	22,	14,	6,	64,	56,	48,	40,	32,	24,	16,	8,
57,	49,	41,	33,	25,	17,	9,	1,	59,	51,	43,	35,	27,	19,	11,	3,
61,	53,	45,	37,	29,	21,	13,	5,	63,	55,	47,	39,	31,	23,	15,	7

Начальная перестановка и соответствующая заключительная перестановка не влияют на безопасность DES. (Как можно легко заметить, эта перестановка в первую очередь служит для облегчения побайтной загрузки да нных открытого текста и шифротекста в микросхему DES. Не забывайте, что DES появился раньше 16- и 32-битовых микропроцессорных шин.) Так как программная реализация этой многобитовой перестановки нелегка (в отличие от тривиальной аппаратной), во многих программных реализациях DES начальная и заключител ьные перестановки не используются. Хотя такой новый алгоритм не менее безопасен, чем DES, он не соответс т-вует стандарту DES и, поэтому, не может называться DES.

Преобразования ключа

Сначала 64-битовый ключ DES уменьшается до 56-битового ключа отбрасыванием каждого восьмого бита, как показано в 10-й. Эти биты используются только для контроля четности, позволяя проверять правильность ключа. После извлечения 56-битового ключа для каждого из 16 этапов DES генерируется новый 48-битовый

подключ. Эти подключи, K_i, определяются следующим образом.

Табл. 12-2.
Перестановка ключа

57,	49,	41,	33,	25,	17,	9,	1,	58,	50,	42,	34,	26,	18,
10,	2,	59,	51,	43,	35,	27,	19,	11,	3,	60,	52,	44,	36,
63,	55,	47,	39,	31,	23,	15,	7,	62,	54,	46,	38,	30,	22,
14,	6,	61,	53,	45,	37,	29,	21,	13,	5,	28,	20,	12,	4

Во первых, 56-битовый ключ делится на две 28-битовых половинки. Затем, половинки циклически сдвигаются налево на один или два бита в зависимости от этапа. Этот сдвиг показан в 9-й.

Табл. 12-3.
Число битов сдвига ключа в зависимости от этапа

Этап	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Число	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

После сдвига выбирается 48 из 56 битов. Так как при этом не только выбирается подмножество битов, но и изменяется их порядок, эта операция называется **перестановка со сжатием**. Ее результатом является набор из 48 битов. Перестановка со сжатием (также называемая переставленным выбором) определена в 8-й. Например, бит сдвинутого ключа в позиции 33 перемещается в позицию 35 результата, а 18-й бит сдвинутого ключа отбрасывается.

Табл. 12-4.
Перестановка со сжатием

14,	17,	11,	2,4,	1,	5,	3,	28,	15,	6,	21,	10,
23,	19,	11,	4,	26,	8,	16,	7,	27,	20,	13,	2,
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32

Из-за сдвига для каждого подключа используется отличное подмножество битов ключа. Каждый бит используется приблизительно в 14 из 16 подключей, хотя не все биты используются в точности одинаковое число раз.

Перестановка с расширением

Эта операция расширяет правую половину данных, R_i, от 32 до 48 битов. Так как при этом не просто повторяются определенные биты, но и изменяется их порядок, эта операция называется **перестановкой с расширением**. У нее две задачи: привести размер правой половины в соответствие с ключом для операции XOR и получить более длинный результат, который можно будет сжать в ходе операции подстановки. Однако главный криптографический смысл совсем в другом. За счет влияния одного бита на две подстановки быстрее возрастает зависимость битов результата от битов исходных данных. Это называется **лавинным эффектом**. DES спроектирован так, чтобы как можно быстрее добиться зависимости каждого бита шифротекста от каждого бита открытого текста и каждого бита ключа.

Перестановка с расширением показана на 9-й. Иногда она называется **E-блоком** (от expansion). Для каждого 4-битового входного блока первый и четвертый бит представляют собой два бита выходного блока, а второй и третий биты - один бит выходного блока. В 7-й показано, какие позиции результата соответствуют каким позициям исходных данных. Например, бит входного блока в позиции 3 переместится в позицию 4 выходного блока, а бит входного блока в позиции 21 - в позиции 30 и 32 выходного блока.

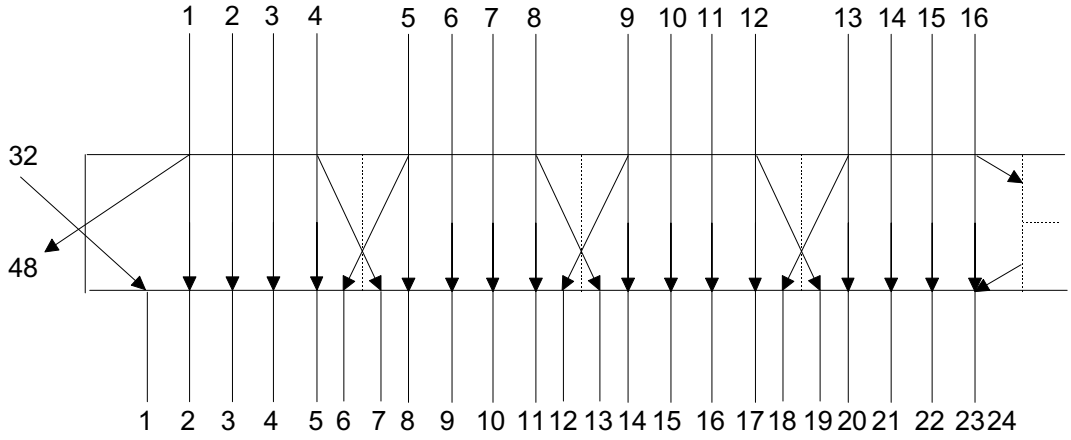


Рис. 12-3. Перестановка с расширением.

Хотя выходной блок больше входного, каждый входной блок генерирует уникальный выходной блок.

Табл. 12-5.
Перестановка с расширением

32,	1,	2,	3,	4,	5,	4,	5,	6,	7,	8,	9,
8,	9,	10,	11,	12.,	13,	12,	13,	14,	15,	16,	17,
16,	17,	18,	19,	20,	21,	20,	21,	22,	23,	24,	25,
24,	25,	26,	27,	28,	29,	28,	29,	30,	31,	32,	1

Подстановка с помощью S-блоков

После объединения сжатого блока с расширенным блоком с помощью XOR над 48-битовым результатом выполняется операция подстановки. Подстановки производятся в восьми **блоках подстановки**, или **S-блоках** (от substitution). У каждого S-блока 6-битовый вход и 4-битовый выход, всего используется восемь различных S-блоков. (Для восьми S-блоков DES потребуется 256 байтов памяти.) 48 битов делятся на восемь 6-битовых подблока. Каждый отдельный подблок обрабатывается отдельным S-блоком: первый подблок - S-блоком 1, вт о-рой - S-блоком 2, и так далее. См. 8-й.

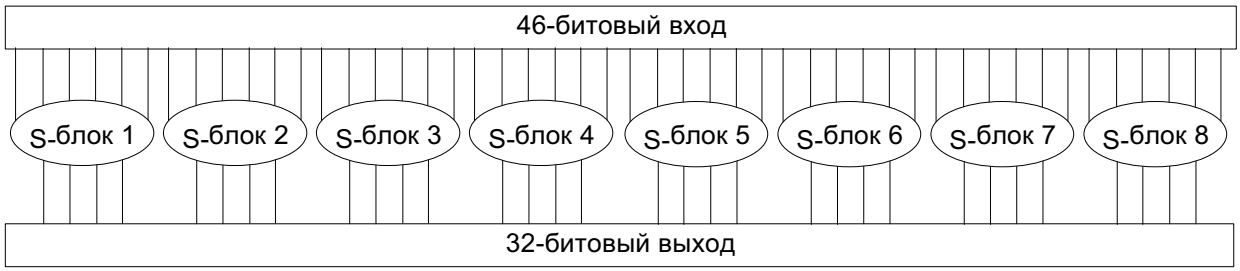


Рис. 12-4. Подстановка - S-блоки.

Каждый S-блок представляет собой таблицу из 2 строк и 16 столбцов. Каждый элемент в блоке является 4-битовым числом. По 6 входным битам S-блока определяется, под какими номерами столбцов и строк искать выходное значение. Все восемь S-блоков показаны в 6-й.

Табл. 12-6.
S-блоки

S-блок 1:															
14,	4,	13,	1,	2,	15,	11,	8,	3,	10,	6,	12.,	5,	9,	0,	7,
0,	15,	7,	4,	14,	2,	13,	1,	10,	6,	12.,	11,	9,	5,	3,	8,

4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13,

S-блок 2:

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,

S-блок 3:

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,

S-блок 4:

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
13, 8, 11, 5, 6. 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,

S-блок 5:

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,

S-блок 6:

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,

S-блок 7:

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,

S-блок 8:

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

Входные биты особым образом определяют элемент S-блока. Рассмотрим 6-битовый вход S-блока: b_1, b_2, b_3, b_4, b_5 и b_6. Биты b_1 и b_6 объединяются, образуя 2-битовое число от 0 до 3, соответствующее строке таблицы. Средние 4 бита, с b_2 по b_5, объединяются, образуя 4-битовое число от 0 до 15, соответствующее столбцу табл ицы.

Например, пусть на вход шестого S-блока (т.е., биты функции XOR с 31 по 36) попадает 110011. Первый и последний бит, объединяясь, образуют 11, что соответствует строке 3 шестого S-блока. Средние 4 бита образуют 1001, что соответствует столбцу 9 того же S-блока. Элемент S-блока 6, находящийся на пересечении строки 3 и столбца 9, - это 14. (Не забывайте, что строки и столбцы нумеруются с 0, а не с 1.) Вместо 110011 подставляется 1110.

Конечно же, намного легче реализовать S-блоки программно в виде массивов с 64 элементами. Для этого потребуется переупорядочить элементы, что не является трудной задачей. (Изменить индексы, не изменяя порядок элементов, недостаточно. S-блоки спроектированы очень тщательно.) Однако такой способ описания S-блоков помогает понять, как они работают. Каждый S-блок можно рассматривать как функцию подстановки 4-битового элемента: b_2 по b_5являются входом, а некоторое 4-битовое число - результатом. Биты b_1 и b_6 определяются соседними блоками, они определяют одну из четырех функций подстановки, возможных в данном S-блоке.

Подстановка с помощью S-блоков является ключевым этапом DES. Другие действия алгоритма линейны и легко поддаются анализу. S-блоки нелинейны, и именно они в большей степени, чем все остальное, обеспечивают безопасность DES.

В результате этого этапа подстановки получаются восемь 4-битовых блоков, которые вновь объединяются в единый 32-битовый блок. Этот блок поступает на вход следующего этапа - перестановки с помощью P-блоков.

Перестановка с помощью P-блоков

32-битовый выход подстановки с помощью S-блоков, перетасовываются в соответствии с P-блоком. Эта перестановка перемещает каждый входной бит в другую позицию, ни один бит не используется дважды, и ни один бит не игнорируется. Этот процесс называется прямой перестановкой или просто перестановкой. Позиции, в которые перемещаются биты, показаны в 5-й. Например, бит 21 перемещается в позицию 4, а бит 4 - в позицию 31.

Табл. 12-7.
Перестановка с помощью P-блоков

16,	7,	20,	21,	29,	12,	28,	17,	1,	15,	23,	26,	5,	18,	31,	10,
2,	8,	24,	14,	32,	27,	3,	9,	19,	13,	30,	6,	22,	11,	4,	25

Наконец, результат перестановки с помощью P-блока объединяется посредством XOR с левой половиной первоначального 64-битового блока. Затем левая и правая половины меняются местами, и начинается следующий этап.

Заключительная перестановка

Заключительная перестановка является обратной по отношению к начальной перестановки и описана в 4-й. Обратите внимание, что левая и правая половины не меняются местами после последнего этапа DES, вместо этого объединенный блок $R_{16}L_{16}$ используется как вход заключительной перестановки. В этом нет ничего особенного, перестановка половинок с последующим циклическим сдвигом привела бы к точно такому же результату. Это сделано для того, чтобы алгоритм можно было использовать как для шифрования, так и для дешифрования.

Табл. 12-8.
Заключительная перестановка

40,	8,	48,	16,	56,	24,	64,	32,	39,	7,	47,	15,	55,	23,	63,	31,
38,	6,	46,	14,	54,	22,	62,	30,	37,	5,	45,	13,	53,	21,	61,	29,
36,	4,	44,	12,	52,	20,	60,	28,	35,	3,	43,	11,	51,	19,	59,	27,
34,	2,	42,	10,	50,	18,	58,	26,	33,	1,	41,	9,	49,	17,	57,	25

Дешифрирование DES

После всех подстановок, перестановок, операций XOR и циклических сдвигов можно подумать, что алгоритм дешифрирования, резко отличаясь от алгоритма шифрования, точно также запутан. Напротив, различные компоненты DES были подобраны так, чтобы выполнялось очень полезное свойство: для шифрования и дешифрования

рирования используется один и тот же алгоритм.

DES позволяет использовать для шифрования или дешифрирования блока одну и ту же функцию. Единс т-венное отличие состоит в том, что ключи должны использоваться в обратном порядке. То есть, если на этапах шифрования использовались ключи K_1, K_2, K_3, ..., K_{16}, то ключами дешифрирования будут K_{16}, K_{15}, K_{14}, ..., K_1. Алгоритм, который создает ключ для каждого этапа, также цикличен. Ключ сдвигается направо, а число поз и-ций сдвига равно 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

Режимы DES

FIPS PUB 81 определяет четыре режима работы: ECB, CBC, OFB и CFB (см. главу 9) [1143]. Банковские стандарты ANSI определяют для шифрования ECB и CBC, а для проверки подлинности - CBC и n-битовый CFB [52].

В мире программного обеспечения сертификация обычно не важна. Из-за своей простоты в большинстве существующих коммерческих программ используется ECB, хотя этот режим наиболее чувствителен к вскр ытию. CBC используется редко несмотря на то, что он лишь незначительно сложнее, чем ECB, и обеспечивает большую безопасность.

Аппаратные и программные реализации DES

Об эффективных аппаратных и программных реализациях алгоритма много писалось [997, 81, 533, 534, 437, 738, 1573, 176, 271, 1572]. Утверждается, что самой быстрой является микросхема DES, разработанная в Digital Equipment Corporation [512]. Она поддерживает режимы ECB и CBC и основана на вентильной матрице GaAs, состоящей из 50000 транзисторов. Данные могут зашифровываться и дешифрироваться со скоростью 1 гигабит в секунду, обрабатывая 16.8 миллионов блоков в секунду. Это впечатляет. Параметры ряда коммерческих ми к-росхем DES приведены в 3-й. Кажущиеся противоречия между тактовой частотой и скоростью обработки да н-ных обусловлены конвейеризацией внутри микросхемы, в которой может быть реализовано несколько раб о-тающих параллельно DES-механизмов.

Наиболее выдающейся микросхемой DES является 6868 VLSI (ранее называвшаяся "Gatekeeper" - Вратарь). Она не только может выполнять шифрование DES за 8 тактов (лабораторные прототипы могут делать это за 4 такта), но также выполнять троекратный DES в режиме ECB за 25 тактов, а троекратный DES в режимах OFB или CBC - за 35 актов. Мне это кажется невозможным, но уверяю вас, она именно так и работает.

Программная реализация DES на мэйнфрейме IBM 3090 может выполнить 32000 шифрований DES в секу нду. На других платформах скорость ниже, но все равно достаточно велика. В 2-й [603, 793] приведены действ и-тельные результаты и оценки для различных ми кропроцессоров Intel и Motorola.

Табл. 12-9.
Коммерческие микросхемы DES

Производитель	Микросхема	Год	Тактовая частота	Скорость данных	Доступность
AMD	Am9518	1981	3 МГц	l.3 Мбайт/с	Н
AMD	Am9568	?	4 МГц	l.5 Мбайт/с	Н
AMD	AmZ8068	1982	4 МГц	l.7 Мбайт/с	Н
AT&T	T7000A	1985	?	l.9 Мбайт/с	Н
CE-Infosys	SuperCrypt CE99C003	1992	20 МГц	12.5 Мбайт/с	Д
CE-Infosys	SuperCrypt CE99C003A	1994	30 МГц	20.0 Мбайт/с	Д
Cryptech	Cry12C102	1989	20 МГц	2.8 Мбайт/с	Д
Newbridge	CA20C03A	1991	25 МГц	3.85 Мбайт/с	Д
Newbridge	CA20C03W	1992	8 МГц	0.64 Мбайт/с	Д
Newbridge	CA95C68/18/09	1993	33 МГц	14.67 Мбайт/с	Д
Pijnenburg	PCC100	?	?	2.5 Мбайт/с	Д
Semaphore Communications	Roadrunner284	?	40 МГц	35.5 Мбайт/с	Д

VLSI Technology	VM007	1993	32 МГц	200.0 Мбайт/с	Д
VLSI Technology	VM009	1993	33 МГц	14.0	Д
VLSI Technology	6868	1995	32 МГц	64.0 Мбайт/с	Д
Western Digital	WD2001/2002	1984	3 МГц	0.23 Мбайт/с	Н

Табл. 12-10.
Скорости DES на различных микропроцессорах и компьютерах

Процессор	Скорость (в МГц)	Блоки DES (в с)
8088	4.7	370
68000	7.6	900
80286	6	1100
68020	16	3500
68030	16	3900
80386	25	5000
68030	50	10000
68040	25	16000
68040	40	23000
80486	66	43000
Sun ELC		26000
HyperSparc		32000
RS6000-350		53000
Sparc 10/52		84000
DEC Alpha 4000/610		154000
HP9000/887	125	196,000

12.3 Безопасность DES

Люди давно интересуются безопасностью DES [458]. Было много рассуждений о длине ключа, количестве итераций и схеме S-блоков. S-блоки были наиболее таинственными - какие-то константы, без видимого объя снения для чего и зачем они нужны. Хотя IBM утверждала, что работа алгоритма была результатом 17 человеколет интенсивного криптоанализа, некоторые люди опасались, что NSA вставило в алгоритм лазейку, которая позволит агентству легко дешифрировать перехваченные соо бщения.

Комитет по разведке Сената США чрезвычайно тщательно расследовал этот вопрос в 1978 году. Результаты работы комитета были засекречены, но в открытых итогах этого расследования с NSA были сняты все обвин ения в неуместном вмешательстве в проектирование алгоритма [1552]. "Было сказано, что NSA убедило IBM в достаточности более короткого ключа, косвенно помогло разработать структуры S-блоков и подтвердило, что в окончательном варианте DES, с учетом всех знаний NSA, отсутствовали статистические или математические бреши " [435]. Однако, так как правительство не опубликовало подробности расследования, многих людей уб едить не удалось.

Тачмен (Tuchman) и Майер (Meyer), разработавшие DES криптографы IBM, заявили, что NSA не изменяло проект [841]:

> Их основным подходом был поиск сильных подстановок, перестановок и функций планирования ключей. . . . IBM по просьбе NSA засекретило информацию, касающуюся критериев выбора. ... "NSA сообщило нам, что мы самостоятельно зан о-во открыли ряд секретов, используемых для создания их собственных алгоритмов", - об ъясняет Тачмен.

Позже в одной из статей Тачмен писал: "Алгоритм DES был полностью разработан внутри IBM ее сотру дниками. NSA не продиктовало ни единой связи!" Тачмен подтвердил это утверждение в своем докладе по ист о-рии DES на Национальной конференции по компьютерной безопасности (National Computer Security Conference)

в 1992 году.

С другой стороны, Копперсмит писал [373, 374]: "Агентство национальной безопасности (NSA) также пом о-гало IBM техническими советами." А Конхейм (Konheim) утверждал: "Мы послали S-блоки в Вашингтон. Они вернулись полностью переработанными. Мы проверили их, и они прошли нашу проверку." На этот факт и сс ы-лаются как на доказательство, что NSA вставило лазейку в DES. По вопросу о каком-либо преднамеренном о с-лаблении DES NSA заявило [363]:

> Относительно Стандарта шифрования данных (DES) мы считаем, что ответ на ваш вопрос о роли NSA в разработке DES содержится в опубликованных итогах расследования Комитета Сената по разведке, проведенного в 1978 году. В сообщении Комитета указывается, что NSA никоим образом не искажало алгоритм, и что безопасность, предоставляемая DES для несе к-ретных данных, с целью защиты которых он и был разработан, была более чем адекватна в течение по крайней мере 5-10 лет. Короче говоря, NSA не вносило и не пыталось вносить никаких ослаблений в алгоритм DES.

Тогда почему они изменили S-блоки? Может быть, чтобы гарантировать, что лазейка не будет встроена в DES самой IBM. У NSA не было причин доверять исследователям IBM, и оно могло решить, что не до конца исполнит свой долг, если не обеспечит отсутствие лазеек в DES. Задание S-блоков и могло быть одним из сп о-собов гарантировать это.

Совсем недавно новые результаты криптоанализа прояснили этот вопрос, который в течение многих лет был предметом спекуляций.

Слабые ключи

Из-за того, что первоначальный ключ изменяется при получении подключа для каждого этапа алгоритма, определенные первоначальные ключи являются **слабыми** [721, 427]. Вспомните, первоначальное значение расщепляется на две половины, каждая из которых сдвигается независимо. Если все биты каждой половины равны 0 или 1, то для всех этапов алгоритма используется один и тот же ключ. Это может произойти, если ключ состоит из одних 1, из одних 0, или если одна половина ключа состоит из одних 1, а другая - из одних 0. Кроме того, у два слабых ключа обладают др угими свойствами, снижающими их безопасность [427].

Четыре слабых ключа показаны в шестнадцатиричном виде в 1-й. (Не забывайте, что каждый восьмой бит - это бит четности.)

Табл. 12-11.
Слабые ключи DES

Значение слабого ключа (с битами четн ости)				Действительный ключ
0101	0101	0101	0101	0000000 0000000
1F1F	1F1F	0E0E	0E0E	0000000 FFFFFFF
E0E0	E0E0	F1F1	F1F1	FFFFFFF 0000000
FEFE	FEFE	FEFE	FEFE	FFFFFFF FFFFFFF

Кроме того, некоторые пары ключей при шифровании переводят открытый текст в идентичный шифротекст. Иными словами, один из ключей пары может расшифровать сообщения, зашифрованные другим ключом пары. Это происходит из-за метода, используемого DES для генерации подключей - вместо 16 различных подключей эти ключи генерируют только два различных подключа. В алгоритме каждый из этих подключей используется восемь раз. Эти ключи, называемые **полуслабыми ключами**, в шестнадцатиричном виде приведены в 0-й.

Табл. 12-12.
Полуслабые пары ключей DES

01FE	01FE	01FE	01FE	и	FE01	FE01	FE01	FE01
1FE0	1FE0	0EF1	0EF1	и	E01F	E01F	F10E	F10E
01E0	01E0	01F1	01F1	и	E001	E001	F101	F101
1FFE	1EEE	0EFE	0EFE	и	FE1F	FE1F	FE0E	FE0E
011F	011F	010E	010E	и	1F01	1F01	0E01	0E01
E0FE	E0FE	F1FE	F1FE	и	FEE0	FEE0	FEE1	FEE1

Ряд ключей генерирует только четыре подключа, каждый из которых четыре раза используется в алгоритме.

Эти **возможно слабые ключи** перечислены в -1-й.

Табл. 12-13.
Возможно слабые ключи DES

1F	1F	01	01	0E	0E	01	01
01	1F	1F	01	01	0E	0E	01
1F	01	01	1F	0E	01	01	0E
01	01	1F	1F	01	01	0E	0E
E0	E0	01	01	F1	F1	01	01
FE	FE	01	01	FE	FE	01	01
FE	E0	1F	01	FE	F1	0E	01
E0	FE	1F	01	F1	FE	0E	01
FE	E0	01	1F	FE	F1	01	0E
E0	FE	01	1F	F1	FE	01	0E
E0	E0	1F	1F	F1	F1	0E	0E
FE	FE	1F	1F	FE	FE	0E	0E
FE	1F	E0	01	FE	0E	F1	01
E0	1F	FE	01	F1	0E	FE	01
FE	01	E0	1F	FE	01	F1	0E
E0	01	FE	1F	F1	01	FE	0E
01	E0	E0	01	01	F1	F1	01
1F	FE	E0	01	0E	FE	F0	01
1F	E0	FE	01	0E	F1	FE	01
01	FE	FE	01	01	FE	FE	01
1F	E0	E0	1F	0E	F1	F1	0E
01	FE	E0	1F	01	FE	F1	0E
01	E0	FE	1F	01	F1	FE	0E
1F	FE	FE	1F	0E	FE	FE	0E

E0	01	01	E0	F1	01	01	F1
FE	1F	01	E0	FE	0E	01	F1
FE	01	1F	E0	FE	01	0E	F1
E0	1F	1F	E0	F1	0E	0E	F1
FE	01	01	FE	FE	01	01	FE
E0	1F	01	FE	F1	0E	01	FE
E0	01	1F	FE	F1	01	0E	FE
FE	1F	1F	FE	FE	0E	0E	FE
1F	FE	01	E0	0E	FE	01	F1
01	FE	1F	E0	01	FE	0E	F1
1F	E0	01	FE	0E	F1	01	FE
01	E0	1F	FE	01	F1	0E	FE
01	01	E0	E0	01	01	F1	F1
1F	1F	E0	E0	0E	0E	F1	F1
1F	01	FE	E0	0E	01	FE	F1
01	1F	FE	E0	01	0E	FE	F1
1F	01	E0	FE	0E	01	F1	FE
01	1F	E0	FE	01	0E	F1	FE
01	01	FE	FE	01	01	FE	FE
1F	1F	FE	FE	0E	0E	FE	FE
FE	FE	E0	E0	FE	FE	F1	F1
E0	FE	FE	E0	F1	FE	FE	F1
FE	E0	E0	FE	FE	F1	F1	FE
E0	E0	FE	FE	F1	F1	FE	FE

Прежде, чем порицать DES слабые ключи, обратите внимание на то, что эти 64 ключа - это крошечная часть полного набора из 72057594037927936 возможных ключей. Если вы выбираете ключ случайно, вероятность выбрать один из слабых ключей пренебрежимо мала. Если вы настоящий параноик, можете всегда проверять "на слабость" сгенерированный ключ. Некоторые думают, что нечего и беспокоиться на этот счет. Другие у т-верждают, что проверка очень легка, почему бы ее и не в ыполнить.

Дальнейший анализ слабых и полуслабых ключей приведен в [1116]. Других слабых ключей в процессе и с-следований найдено не было.

Ключи-дополнения

Выполним побитное дополнение ключа, заменяя все 0 на 1 и все 1 - на 0. Теперь, если блок открытого текста зашифрован оригинальным ключом, то дополнение ключа при шифровании превратит дополнение блока о т-крытого текста в дополнение блока шифротекста. Если x' обозначает дополнение x, то следующее верно:

$E_K(P) = C$

$E_K(P') = C'$

В этом нет ничего таинственного. На каждом этапе после перестановки с расширением подключи подверг а-ются операции XOR с правой половиной. Прямым следствием этого факта и является приведенное свойство комплиментарности.

Это означает, что при выполнении вскрытия DES с выбранным открытым текстом нужно проверять только половину возможных ключей: 2^{55} вместо 2^{56} [1080]. Эли Бихам (Eli Biham) и Ади Шамир показали [172], что существует вскрытие с известным открытым текстом, имеющее ту же сложность, для которого нужно не меньше 2^{33} известных открытых текстов.

Остается вопросом, является ли такое свойство слабостью, так как в большинстве сообщений нет компл и-ментарных блоков открытого текста (для случайного открытого текста шансы "против" чрезвычайно велики), а пользователей можно предупредить не пользоваться дополняющими.

Алгебраическая структура

Все возможные 64-битовые блоки открытого текста можно отобразить на 64-битовые блоки шифротекста

2^{64}! Различными способами. Алгоритм DES, используя 56-битовый ключ, предоставляет нам 2^{56} (приблизительно 10^{17}) таких отображений. Использование многократного шифрования на первый взгляд позв оляет значительно увеличить долю возможных отображений. Но это правильно только, если действие DES не обладает определенной алгебраической структурой.

Если бы DES был **замкнутым**, то для любых K_1 и K_2 всегда существовало бы такое K_3, что

$$E_{K_2}(E_{K_1}(P)) = E_{K_3}(P)$$

Другими словами, операция шифрования DES образовала бы группу, и шифрование набора блоков открыт ого текста последовательно с помощью K1 и K2 было бы идентично шифрованию блоков ключом K3. Что еще хуже, DES был бы чувствителен к вскрытию "встреча посередине" с известным открытым текстом, для которого потребовалось бы только 2^{28} этапов [807].

Если бы DES был **чистым**, то для любых K_1, K_2 и K_3 всегда существовало бы такое K_4, что

$$E_{K_3}(E_{K_2}(E_{K_1}(P))) = E_{K_4}(P)$$

Тройное шифрование было бы бесполезным. (Заметьте, что замкнутый шифр обязательно является и чи стым, но чистый шифр не обязательно является замкнутым.)

Ряд подсказок можно найти в ранней теоретической работе Дона Копперсмита, но этого недостаточно [377]. Различные криптографы пытались решить эту проблему [588, 427, 431, 527, 723, 789]. В повторяющихся эксп ериментах собирались "неопровержимые доказательства" того, что DES не является группой [807, 371, 808, 1116, 809], но только в 1992 году криптографам удалось это доказать окончательно [293]. Копперсмит утве рждает, что команда IBM знала об этом с самого н ачала.

Длина ключа

В оригинальной заявке фирмы IBM в NBS предполагалось использовать 112-битовый ключ. К тому времени, когда DES стандартом, длина ключа уменьшилась до 56 бит. Многие криптографы настаивали на более дли нном ключе. Основным их аргументом б ыло вскрытие грубой силой (см. раздел 7.1).

В 1976 и 1977 гг. Диффи и Хеллман утверждали, что специализированный параллельный компьютер для вскрытия DES, стоящий 20 миллионов долларов, сможет раскрыть ключ за день. В 1981 году Диффи увеличил время поиска до двух дней, а стоимость - до 50 миллионов долларов [491]. Диффи и Хеллман утверждали, что вскрытие в тот момент времени находилось за пределами возможностей любой организации, кроме подобных NSA, но что к 1990 году DES должен полностью утратить свою безопасность [714].

Хеллман [716] продемонстрировал еще один аргумент против малого размера ключа: разменивая объем п амяти на время, можно ускорить процесс поиска. Он предложил вычислять и хранить 2^{56} возможных результатов шифрования каждым возможным ключом единственного блока открытого текста. Тогда для взлома неизвестн ого ключа криптоаналитику потребуется только вставить блок открытого текста в шифруемый поток, вскрыть получившийся результат и найти ключ. Хеллман оценил стоимость такого устройства вскрытия в 5 миллионов долларов.

Аргументы за и против существования в каком-нибудь тайном бункере правительственного устройства вскрытия DES продолжают появляться. Многие указывают на то, что среднее время наработки на отказ для микросхем DES никогда не было большим настолько, чтобы обеспечивать работу устройства. В [1278] было показано, что этого возражения более чем достаточно. Другие исследователи предлагают способы еще больше ускорить процесс и уменьшить эффект отказа микросхем.

Между тем, аппаратные реализации DES постепенно приблизились к реализации требования о миллионе шифрований в секунду, предъявляемого специализированной машиной Диффи и Хеллмана. В 1984 году были выпущены микросхемы DES, способные выполнять 256000 шифрования в секунду [533, 534]. К 1987 году были разработаны микросхемы DES, выполняющие 512000 шифрований в секунду, и стало возможным появление варианта, способного проверять свыше миллиона ключей в секунду [738, 1573]. А в 1993 Майкл Винер (Michael Wiener) спроектировал машину стоимостью 1 миллион долларов, которая может выполнить вскрытие DES гр убой силой в среднем за 3.5 часа (см. раздел 7.1).

Никто открыто не заявил о создании этой машины, хоте разумно предположить, что кому-то это удалось. Миллион долларов - это не слишком большие деньги для большой и даже не очень большой страны.

В 1990 году два израильских математика, Бихам (Biham) и Шамир, открыли **дифференциальный крип тоанализ**, метод, который позволил оставить в покое вопрос длины ключа. Прежде, чем мы рассмотрим этот метод, вернемся к некоторым другим крит ическим замечаниям в адрес DES.

Количество этапов

Почему 16 этапов? Почему не 32? После пяти этапов каждый бит шифротекста является функцией всех б итов открытого текста и всех битов ключа [1078, 1080], а после восьми этапов шифротекст по сути представляет собой случайную функцию всех битов открытого текста и всех битов ключа [880]. (Это называется лавинным эффектом.) Так почему не остановиться после восьми этапов?

В течение многих лет версии DES с уменьшенным числом этапов успешно вскрывались. DES с тремя и ч етырьмя этапами был легко взломан в 1982 году [49]. DES с шестью этапами пал несколькими годами позже [336]. Дифференциальный криптоанализ Бихама и Шамира объяснил и это: DES с любым количеством этапов, меньшим 16, может быть взломан с помощью вскрытия с известным открытым текстом быстрее, чем с пом ощью вскрытия грубой силой. Конечно грубый взлом является более вероятным способом вскрытия, но интер есен тот факт, что алгоритм содержит ровно 16 этапов.

Проектирование S-блоков

Помимо уменьшения длины ключа NSA также обвиняют в изменении содержания S-блоков. Настаивая на подтверждении схемы S-блоков, NSA заявило, что детали алгоритма являются "чувствительными" и не могут быть опубликованы. Многие криптографы подозревали, что разработанные в NSA S-блоки содержат лазейку, позволяющую NSA легко выполнять криптоанализ алгоритма.

С момента появления алгоритма для анализа схемы и работы S-блоков были предприняты значительные усилия. В середине 70-х Lexar Corporation [961, 721] и Bell Laboratories [1120] исследовали работу S-блоков. Ни одно из исследований не обнаружило никаких слабостей, хотя оба исследования обнаружили непонятный сво йства. S-блоки имеют больше свойств, общих с линейным преобразованием, чем можно было ожидать при их формировании случайным образом. Команда Bell Laboratories констатировала, что S-блоки могут содержать скрытые лазейки, а доклад Lexar з авершался следующей фразой:

> В DES были найдены структуры, несомненно вставленные для повышения устойчивости системы к определенным типам вскрытия. Также были найдены структуры, кот орые, по видимому, ослабили систему.

С другой стороны этот доклад также содержал следующее предупреждение:

> ... проблема [поиска структур в S-блоках] усложняется из-за способности человеческого сознания находить в случайных данных структуры, которые в действител ьности вовсе не являются структурами.

На втором симпозиуме по DES Агентство национальной безопасности раскрыло ряд критериев проектиров ания S-блоков [229]. Но это не смогло снять всех подозрений, и спор продолжился [228, 422, 714, 1506, 1551].

В литературе про S-блоки писались удивительные вещи. Последние три бита результата четвертого S-блока могут быть получены тем же способом, что и первые, при помощи дополнения некоторых из входных битов [436, 438]. Различные, но тщательно подобранные входные данные для S-блоков могут давать одинаковый р езультат [436]. Можно получить результат одного этапа DES, меняя биты только в трех соседних S-блоках [487]. Шамир заметил, что элементы S-блоков, казалось, были несколько неустойчивы, но не собирался использовать эту неустойчивость для вскрытия [1423]. (Он упомянул об особенности пятого S-блока, но только спустя восемь лет линейный криптоанализ воспользовался этой особенностью.) Другие исследователи показали, что для пол учения S-блоков с наблюдаемыми характеристиками могли использоваться общеизвестные принципы проект ирования [266).

Дополнительные результаты

Были предприняты и другие попытки криптоанализировать DES. Один из криптографов искал закономерн ости, используя спектральные тесты [559]. Другие анализировали последовательность линейных множителей, но их вскрытие потерпело неудачу после восьми этапов [1297, 336, 531]. Неопубликованное вскрытие, выполне нное в 1987 году Дональдом Дэвисом (Donald Davies), использовало способ, с помощью которого перестановка с расширением повторяет биты в соседних S-блоках, это вскрытие также оказалось бесполезным после восьми этапов [172, 429].

12.4 Дифференциальный и линейный криптоанализ

Дифференциальный криптоанализ

В 1990 году Эли Бихам и Ади Шамир ввели понятие **дифференциального криптоанализа** [167, 168, 171, 172]. Это был новый, ранее неизвестный метод криптоанализа. Используя этот метод, Бихам и Шамир нашли способ вскрытия DES с использованием выбранного открытого текста, который был эффективнее вскрытия гр убой силой.

Дифференциальный криптоанализ работает с **парами шифротекстов**, открытые тексты которых содержат определенные отличия. Метод анализирует эволюцию этих отличий в процессе прохождения открытых текстов

через этапы DES при шифровании одним и тем же ключом.

Просто выберем пару открытых текстов с фиксированным различием. Можно выбрать два открытых текста случайным образом, лишь бы они отличались друг от друга определенным образом, криптоаналитику даже не нужно знать их значений. (Для DES термин "различие" определяется с помощью XOR. Для других алгоритмов этот термин может определяться по другому.) Затем, используя различия в получившихся шифротекстах, пр исвоим различные вероятности различным ключам. В процессе дальнейшего анализа следующих пар шифроте кстов один из ключей станет наиболее вероятным. Это и есть правильный ключ.

Подробности гораздо сложнее. На 7-й представлена функция одного этапа DES. Представьте себе пару вх одов, X и X', с различием ΔX. Выходы, Y и Y' известны, следовательно, известно и различие между ними ΔY. Известны и перестановка с расширением, и P-блок, поэтому известны ΔA и ΔC. B и B' неизвестны, но их разность ΔB известна и равна ΔA. (При рассмотрении различия XOR K_i с A и A' нейтрализуются.) Пока все просто. Фокус вот в чем: для любого заданного ΔA не все значения ΔC равновероятны. Комбинация ΔA и ΔC позволяет предположить значения битов для A XOR K_i и A' XOR K_i. Так как A и A' известны, это дает нам информацию о K_i.

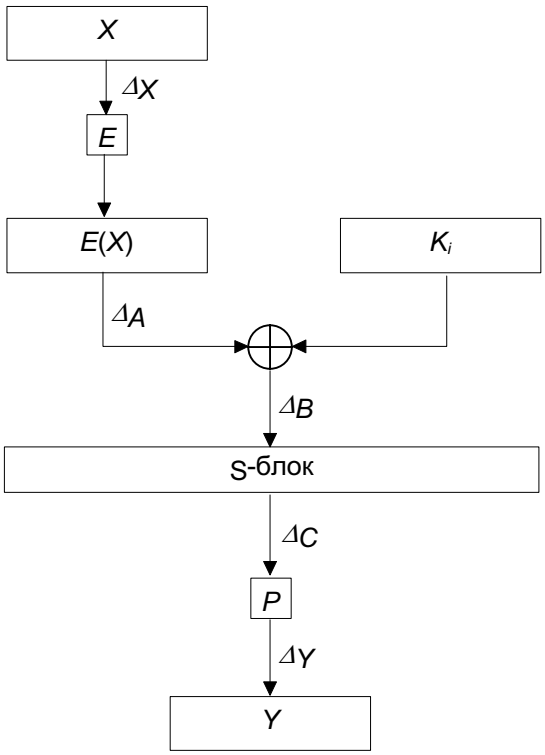


Рис. 12-5. Функция этапа DES.

Взглянем на последний этап DES. (При дифференциальном криптоанализе начальная и заключительная п ерестановки игнорируются. Они не влияют на вскрытие, только затрудняя объяснение.) Если мы сможем опред елить K_{16}, то мы получим 48 битов ключа. (Не забывайте, на каждом этапе подключ состоит из 48 битов 56-битового ключа.) Оставшиеся 8 битов мы можем получить грубым взломом. K_{16} даст нам дифференциальный криптоанализ.

Определенные различия пар открытых текстов обладают высокой вероятностью вызвать определенные ра зличия получаемых шифротекстов. Эти различия называются **характеристиками**. Характеристики распростр аняются на определенное количество этапов и по существу определяют прохождение этих этапов. Существуют входное различие, различие на каждом этапе и выходное различие - с определенной вероятностью.

Эти характеристики можно найти, создав таблицу, строки которой представляют возможные входы XOR (XOR двух различных наборов входных битов), столбцы - возможные результаты XOR, а элементы - сколько раз конкретный результат XOR встречается для заданного входа XOR. Такую таблицу можно сгенерировать для каждого из восьми S-блоков DES.

Например, на 6-йа показана характеристика одного этапа. Входное различие слева равно L, оно может быть произвольным. Входное различие справа равно 0. (У двух входов одинаковая правая половина, поэтому их ра зличие - 0.) Так как на входе функции этапа нет никаких различий, то нет различий и на выходе функции этапа. Следовательно, выходное различие левой части - $L \oplus 0 = L$, а выходное различие правой части - 0. Это трив иальная характеристика, она истинна с вероятностью 1.

На 6-йб показана менее очевидная характеристика. Снова, различие L левых частей произвольно. Входное различие правых частей равно 0x60000000, два входа отличаются только первым и третьим битами. С вероя т-ностью 14/64 различие на выходе функции этапа равно $L \oplus$ 0x00808200. Это означает, что выходное различие левых половин равно $L \oplus$ 0x00808200, а выходное различие правых половин - 0x60000000 (с вероятностью 14/64)

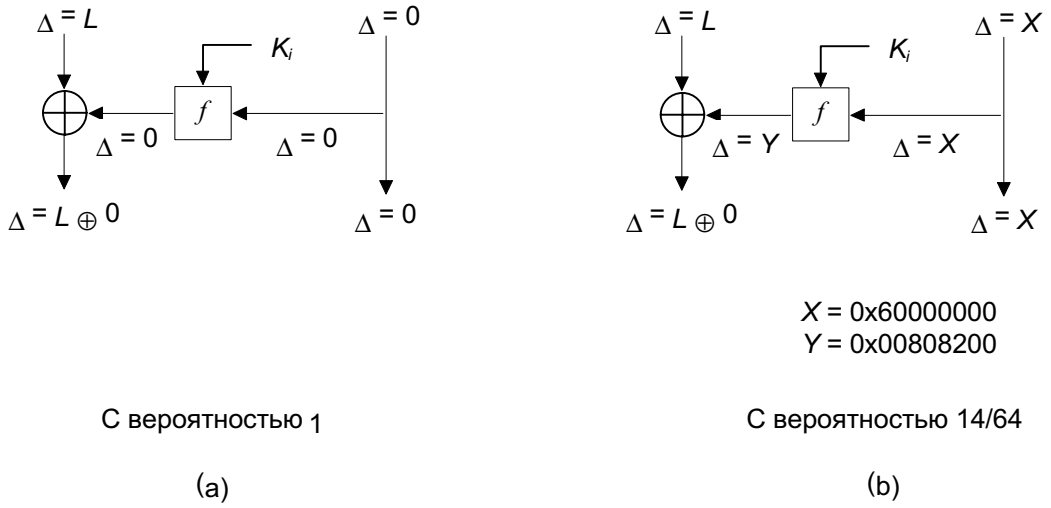


Рис. 12-6. Характеристики DES.

Различные характеристики можно объединять. Также, при условии, что этапы независимы, вероятности м о-гут перемножаться. На 5-й объединяются две ранее описанных характеристики. Входное различие слева равно 0x00808200, а справа - 0x60000000. В конце первого этапа входное различие и результат функции этапа нейтр а-лизуют друг друга, и выходное различие равно 0. Это различие поступает на вход второго этапа, окончательное выходное различие слева равно 0x60000000, а справа - 0. Вероятность этой двухэтапной характерист ики - 14/64.

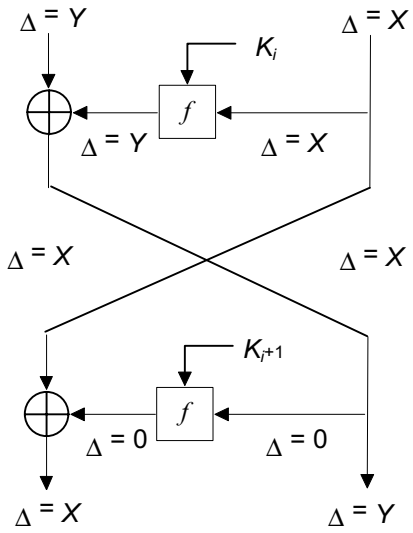


Рис. 12-7. Двухэтапная характеристика DES.

Пара открытых текстов, соответствующих характеристике, называется правильной парой, а пара открытых текстов, несоответствующих характеристике - неправильной парой. Правильная пара подсказывает правильный ключ этапа (для последнего этапа характеристики), неправильная пара - случайный ключ этапа.

Чтобы найти правильный ключ этапа, нужно просто собрать достаточное количество предположений. Один

из подключей будет встречаться чаще, чем все остальные. Фактически, правильный подключ возникнет из всех случайный возможных подключей.

Итак, дифференциальное основное вскрытие n-этапного DES дает 48-битовый подключ, используемый на этапе n, а оставшиеся 8 битов ключа получаются с помощью грубого взлома.

Но ряд заметных проблем все же остается. Во первых, пока вы не перейдете через некоторое пороговое зн ачение, вероятность успеха пренебрежимо мала. То есть, пока не будет накоплено достаточное количество да нных, выделить правильный подключ из шума невозможно. Кроме того, такое вскрытие не практично. Для хр анения вероятностей 2^{48} возможных ключей необходимо использовать счетчики, и к тому же для вскрытия п отребуется слишком много данных.

Бихам и Шамир предложили свой способ вскрытия. Вместо использования 15-этапной характеристики 16-этапного DES, они использовали 13-этапную характеристику и ряд приемов для получения последних нескол ьких этапов. Более короткая характеристика с большей вероятностью будет работать лучше. Они также испол ьзовали некоторые сложные математические приемы для получения вероятных 56-битовых ключей, которые и проверялись немедленно, таким образом устранялась потребность в счетчиках. Такое вскрытие достигает усп еха, как только находится правильная пара. Это позволяет избежать порогового эффекта и получить линейную зависимость для вероятности успеха. Если у вас в 1000 раз меньше пар, то вероятность успеха в 1000 раз мен ьше. Это звучит ужасно, но это намного лучше, чем порог. Всегда есть некоторая вероятность немедленной уд ачи.

Результаты являются весьма интересными. В -2-й проведен обзор лучших дифференциальных вскрытий DES с различным количеством этапов [172]. Первый столбец содержит количество этапов. Элементы следующих двух столбца представляют собой количество выбранных или известных открытых текстов, которые должны быть проверены для вскрытия, а четвертый столбец содержит количество действительно проанализированных открытых текстов. В последнем столбце приведена сложность анализа, после обнаружения требуемой пары.

Табл. 12-14. Вскрытие с помощью дифференциального криптоанализа

Количество этапов	Выбранные открытые тексты	Известные открытые тексты	Проанализированные открытые тексты	Сложность анализа
8	2^{14}	2^{38}	4	29
9	2^{24}	2^{44}	2	2^{32}†
10	2^{24}	2^{43}	2^{14}	2^{15}
11	2^{31}	2^{47}	2	2^{32}†
12	2^{31}	2^{47}	2^{21}	2^{21}
13	2^{39}	2^{52}	2	2^{32}†
14	2^{39}	2^{51}	2^{29}	2^{29}
15	2^{47}	2^{56}	27	2^{37}
16	2^{47}	2^{55}	2^{36}	2^{37}

† Сложность анализа для этих вариантов может быть значительно уменьшена за счет использования примерно в четыре раза большего количество открытых текстов и метода группировок.

Наилучшее вскрытие полного 16-этапного DES требует 2^{47} выбранных открытых текстов. Можно преобраз овать его к вскрытию с известным открытым текстом, но для него потребуется уже 2^{55} известных открытых тек стов. При анализе потребуется 2^{37} операций DES.

Дифференциальный криптоанализ эффективен против DES и аналогичных алгоритмов с постоянными S-блоками. Эффективность вскрытие сильно зависит от структуры S-блоков, блоки DES по счастливой случайн ости были оптимизированы против дифференциального криптоанализа. Для всех режимов работы DES - ECB, CBC, CFB и OFB - вскрытие с дифференциальным криптоанализом имеет одинаковую сложность [172].

Устойчивость DES может быть повышена путем увеличения количества этапов. Дифференциальный кри птоанализ с выбранным открытым текстом для DES с 17 или 18 этапами потребует столько же времени, сколько нужно для вскрытия грубой силой [160]. При 19 и более этапах дифференциальный криптоанализ становится невозможным, так как для него потребуется более, чем 2^{64} выбранных открытых текстов - не забудьте, DES и с пользует блоки размером 64 битов, поэтому для него существует только 2^{64} возможных открытых текстов. (В общем случае, вы можете доказать устойчивость алгоритма к дифференциальному криптоанализу, показав, что количество открытых текстов, необходимых для выполнения вскрытия, превышает количество возможных о ткрытых текстов.)

Нужно отметить ряд важных моментов. Во первых, это вскрытие в значительной степени теоретическое. О громные требования к времени и объему данных, необходимых для выполнения вскрытия с помощью дифф еренциального криптоанализа, находятся почти для всех вне пределов досягаемости. Чтобы получить нужные данные для выполнения такого вскрытия полного DES, вам придется почти три года шифровать поток выбра нных шифротекстов 1.5 Мегабит/с. Во вторых, это в первую очередь вскрытие с выбранным открытым текстом. Оно может быть преобразовано к вскрытию с известным открытым текстом, но вам придется просмотреть все пары "открытый текст/шифротекст" в поисках полезных. В случае полного 16-этапного DES это делает вскр ытие чуть менее эффективным по сравнению с грубой силой (вскрытие дифференциальным криптоанализом тр ебует $2^{55.1}$ операций, а вскрытие грубой силой - 2^{55}). Таким образом, правильно реализованный DES сохраняет устойчивость к дифференциальному криптоанализу.

Почему DES так устойчив к дифференциальному криптоанализу? Почему S-блоки оптимизированы так, что усложняют такое вскрытие насколько возможно? Почему используется ровно столько, а не больше этапов? П отому что создатели DES знали о дифференциальном анализе. Дон Копперсмит из IBM недавно писал [373, 374]:

> При проектировании использовались преимущества определенных криптоаналитических методов, особенно метода "дифференциального криптоанализа", который не был опубликован в открытой литературе. После дискуссий с NSA было р ешено, что раскрытие процесса проектирования раскроет и метод дифференциального криптоанализа, мощь которого может быть использована против многих шифров. Это, в свою очередь, сократило бы преимущество Соединенных Штатов перед другими странами в области криптографии.

Ади Шамир откликнулся, предложив Копперсмиту признаться, что с тех пор ему не удалось найти эффе ктивного способа вскрытия DES. Копперсмит предпочел отмолчат ься [1426].

Криптоанализ со связанными ключами

В 9-й показано количество битов, на которые циклически смещается ключ DES на каждом этапе: на 2 бита на каждом этапе, кроме этапов 1, 2, 9 и 16, когда ключ сдвигается на 1 бит. Почему?

Криптоанализ со связанными ключами похож на дифференциальный криптоанализ, но он изучает разл ичие между ключами. Вскрытие отличается от любого из ранее рассмотренных: криптоаналитик выбирает связь между парой ключей, но сами ключи остаются ему неизвестны. Данные шифруются обоими ключами. В варианте с известным открытым текстом криптоаналитику известны открытый текст и шифротекст данных, шифр ованных двумя ключами. В варианте с выбранным открытым текстом криптоаналитик пытается выбрать откр ытый текст, зашифрованный двумя ключами.

Модифицированный DES, в котором ключ сдвигается на два бита после каждого этапа, менее безопасен. Криптоанализ со связанными ключами может взломать такой вариант алгоритма, использовав только 2^{17} выбранных открытых текстов для выбранных ключей или 2^{33} известных открытых текстов для выбранных ключей [158, 163].

Такое вскрытие также не реализуемо на практике, но оно интересно по трем причинам. Во первых, это пе рвая попытка криптоаналитического вскрытия алгоритма генерации подключей в DES. Во вторых, это вскрытие не зависит от количества этапов криптографического алгоритма, он одинаково эффективен против DES с 16, 32 или 1000 этапами. И в третьих, DES невосприимчив к такому вскрытию. Изменение количества битов циклич еского сдвига мешает криптоанализу со связанными ключами.

Линейный криптоанализ

Линейный криптоанализ представляет собой другой тип криптоаналитического вскрытия, изобретенный Мицуру Мацуи (Mitsuru Matsui) [1016, 1015, 1017]. Это вскрытие использует линейные приближения для оп исания работы блочного шифра (в данном случае DES.)

Это означает, что если вы выполните операцию XOR над некоторыми битами открытого текста, затем над некоторыми битами шифротекста, а затем над результатами, вы получите бит, который представляет собой XOR некоторых битов ключа. Это называется линейным приближением, которое может быть верным с некот орой вероятностью p. Если $p \neq 1/2$, то это смещение можно использовать. Используйте собранные открытые те ксты и связанные шифротексты для предположения о значениях битов ключа. Чем больше у вас данных, тем вернее предположение. Чем больше смещение, тем б ыстрее вскрытие увенчается успехом.

Как определить хорошее линейное приближение для DES? Найдите хорошие одноэтапные линейные пр иближения и объедините их. (Начальная и заключительная перестановки снова игнорируются, так как они не влияют на вскрытие.) Взгляните на S-блоки. У них 6 входных битов и 4 выходных. Входные биты можно объ единить с помощью операции XOR 63 способами ($2^6 - 1$), а выходные биты - 15 способами. Теперь для каждого S-блока можно оценить вероятность того, что для случайно выбранного входа входная комбинация XOR равна некоторой выходной комбинации XOR. Если существует комбинация с достаточно большим смещением, то л инейный криптоанализ может сработать.

Если линейные приближения не смещены, то они будут выполняться для 32 из 64 возможных входов. Я и з-

бавлю вас от длительного изучения таблиц, наиболее смещенным S-блоком является пятый S-блок. Действ ительно, для 12 входов второй входной бит равен XOR всех четырех выходных битов. Это соответствует вероя тности 3/16 или смещению 5/16, что является самым большим смещением для всех S-блоков. (Шамир писал об этом в [1423], но не смог найти способа использовать.)

На 4-й показано, как воспользоваться этим для вскрытия функции этапа DES. b26 - это входной бит S-блока 5. (Я нумерую биты слева направо от 1 до 64. Мацуи игнорирует это принятое для DES соглашение и нумерует свои биты справа налево и от 0 до 63. Этого хватит, чтобы свести вас с ума.) c_{17}, c_{18}, c_{19}, c_{20} - это 4 выходных бита S-блока 5. Мы можем проследить b_{26} в обратном направлении от входа в S-блок. Для получения b_{26} бит объединяется с помощью XOR с битом подключа $K_{i,26}$. А бит X_{17} проходит через подстановку с расширением, чтобы превратиться в a_{26}. После S-блока 4 выходных бита проходят через P-блок, превращаясь в четыре выхо дных бита функции этапа: Y_3, Y_8, Y_{14} и Y_{25}. Это означает, что с вероятностью 1/2 - 5/6:

$X_{17} \oplus Y_3 \oplus Y_8 \oplus Y_{14} \oplus Y_{25} = K_{i,26}$

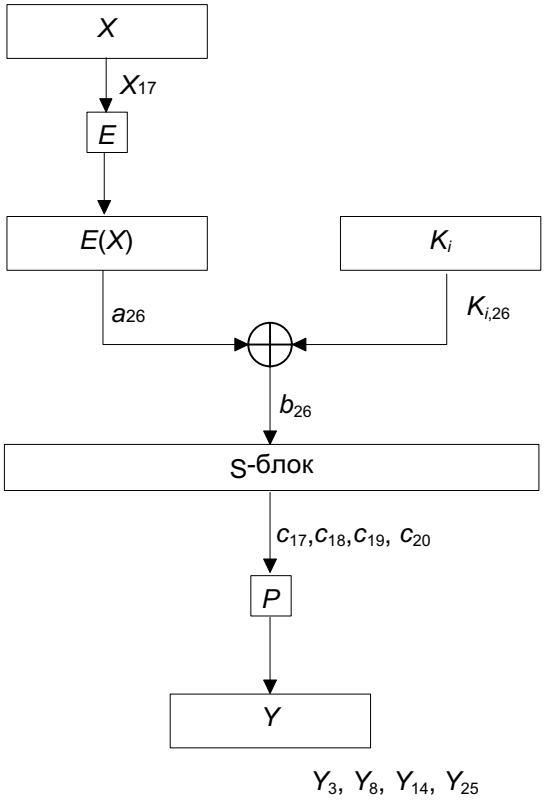
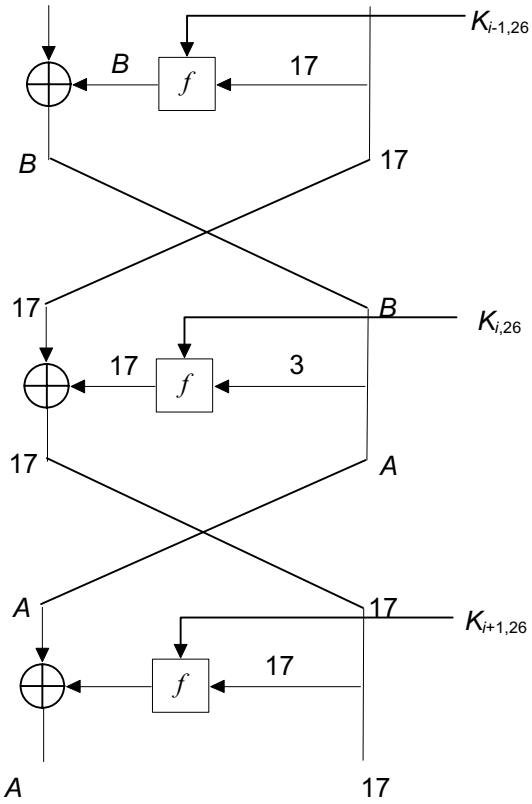


Рис. 12-8. 1-этапное линейное приближение для DES.

Способ, которым можно объединить линейные приближения для различных этапов, похож на тот, который обсуждался для дифференциального криптоанализа. На 3-й показано 3-этапное линейное приближение с вер оятностью 1/2+0.0061. Качество отдельных приближений различно: последнее очень хорошо, первое достаточно хорошо, а среднее - плохо. Но вместе эти три 1-этапных приближения дают очень хорошее трехэтапное пр иближение.



A=[3, 8, 14, 25] B=[8, 14, 25]

С вероятностью $1/2+6.1*10^{-3}$

Рис. 12-9. 3-этапное линейное приближение DES.

Базовое вскрытие должно использовать наилучшее линейное приближение для 16-этапного DES. Для него требуется 2^{47} известных открытых блоков, а результатом вскрытия является 1 бит ключа. Это не очень полезно. Если вы поменяете местами открытый текст и шифротекст и используете дешифрирование вместе с шифров анием, вы сможете получить 2 бита. Это все еще не очень полезно.

Существует ряд тонкостей. Используйте 14-этапное линейное приближение для этапов с 2 по 15. Попробуем угадать 6 битов подключа для S-блока 5 первого и последнего этапов (всего, таким образом, 12 битов ключа). Для эффективности выполняем линейный криптоанализ параллельно 2^{12} раз и выбираем правильный вариант, основываясь на вероятностях. Это раскрывает 12 битов и b_{26}, а поменяв местами открытый текст и шифротекст мы получим еще 13 битов. Для получения оставшихся 30 битов используйте исчерпывающий поиск. Существ уют и друге приемы, но описанный является основным.

При вскрытии таким образом полного 16 этапного DES ключ будет раскрыт в среднем с помощью 2^{43} известных открытых текстов. Программная реализации этого вскрытия, работая на 12 рабочих станциях HP9735, раскрыла ключ DES за 50 дней [1019]. В момент написания этой книги это наиболее эффективный способ вскрытия DES.

Линейный криптоанализ сильно зависит от структуры S-блоков, оказалось, что S-блоки DES не оптимизир ованы против такого способа вскрытия. Действительно, смещение в S-блоках, выбранных для DES, находится между 9 и 16 процентами, что не обеспечивает надежной защиты против линейного криптоанализа [1018]. С огласно Дону Копперсмиту [373, 374] устойчивость к линейному криптоанализу "не входило в число критериев проектирования DES". Либо разработчикам не было известно о линейном криптоанализе, либо при проектир овании они отдали преимущество устойчив ости против известного им еще более мощного средства вскрытия.

Линейный криптоанализ новее, чем дифференциальный, и в ближайшее время возможно дальнейшее пр одвижение в этом направлении. Некоторые идеи выдвинуты в [1270, 811], но не ясно, можно ли их эффективно применить против полного DES. Однако они очень хорошо работают против вариантов с уменьшенным числом этапов.

Дальнейшие направления

Был предпринят ряд попыток расширить концепцию дифференциального криптоанализа на дифференциалы более высоких порядков [702, 161, 927, 858, 860]. Ларс Кнудсен (Lars Knudsen) использует нечто, называемое частичными дифференциалами для вскрытия 6-этапного DES. Этот метод требует 32 выбранных открытых текста и 20000 шифрований [860]. Но этот метод слишком нов, чтобы можно было утверждать, что он облегчит вскрытие полного 16-этапного DES.

Другим способом вскрытия является дифференциально-линейный криптоанализ - объединение дифференциального и линейного криптоанализа. Сьюзен Лангфорд (Susan Langford) и Хеллман предлагают вскрытие 8-этапного DES, которое раскрывает 10 битов ключа с вероятностью успеха 80 процентов, используя 512 выбранных открытых текстов, и с вероятностью успеха 95 процентов, используя 768 выбранных открытых текстов [938]. После вскрытия необходим поиск грубой силой в оставшемся пространстве ключей (2^{46} возможных ключей). Хотя по времени это вскрытие сравнимо с предыдущими способами, для него требуется намного меньше открытых текстов. Однако расширение этого метода на большее количество этапов легким не кажется.

Но этот метод нов, и работа продолжается. В ближайшие годы возможны заметные успехи. Может быть у успеха добьется сочетание этого вскрытия с дифференциальным криптоанализом более высоких порядков. Кто знает?

12.5 Реальные критерии проектирования

После появления публикаций о дифференциальном криптоанализе IBM раскрыла критерии проектирования S-блоков и P-блока [373, 374]. Критериями проектирования S-блоков являлись:

— У каждого S-блока 6 входных битов и 4 выходных бита. (Это самый большой размер, который мог быть реализован в одной микросхеме по технологии 1974 года.)

— Ни один выходной бит S-блока не должен быть слишком близок к линейной функции входных битов.

— Если зафиксировать крайние левый и правый биты S-блока, изменяя 4 средних бита, то каждый возможный 4-битовый результат получается только один раз.

— Если два входа S-блока отличаются только одним битом, результаты должны отличаться по крайней мере на 2 бита.

— Если два входа S-блока отличаются только двумя центральными битами, результаты должны отличаться по крайней мере на 2 бита.

— Если два входа S-блока отличаются двумя первыми битами, а последние их последние 2 бита совпадают, результаты не должны быть одинаковыми.

— Для любого ненулевого 6-битового отличия между входами, не более, чем 8 из 32 пар входов могут приводить на выходе к одинаковому различию.

— Аналогичный предыдущему критерий, но для случая трех активных S-блоков.

Критериями проектирования P-блока являлись:

— 4 выходных бита каждого S-блока на этапе i распределены так, чтобы 2 из них влияют на средние биты S-блоков на этапе i + 1, а другие 2 бита влияют на последние биты.

— 4 выходных бита каждого S-блока влияют на шесть различных S-блоков, никакие 2 не влияют на один и тот же S-блок.

— Если выходной бит одного S-блока влияет на средние биты другого S-блока, то выходной бит этого другого S-блока не может влиять на средние биты первого S-блока.

Эта работа продолжала обсуждение критериев. Сегодня совсем нетрудно генерировать S-блоки, но в начале 70-х это было нелегкой задачей. Тачмен говорил, что программы, готовившие S-блоки, работали месяцами.

12.6 Варианты DES

Многократный DES

В ряде реализаций DES используется трехкратный DES (см. 2-й) [55]. Так как DES е является группой, полученный шифротекст гораздо сложнее вскрыть, используя исчерпывающий поиск: 2^{112} попыток вместо 2^{56}. Подробности можно найти в разделе 15.2.

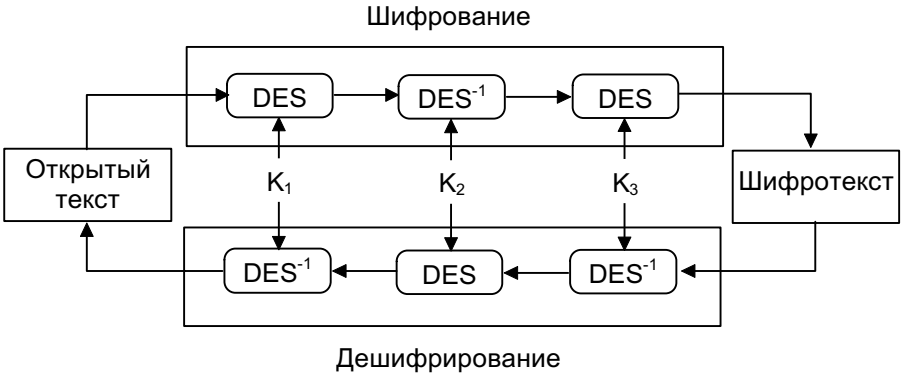


Рис. 12-10. Трехкратный DES.

DES с независимыми подключами

Другой возможностью является использование различных подключей на каждом этапе, не создавая их из одного 56-битового ключа [851]. Так как на каждом из 16 этапов используется 48 битов ключа, то длина ключа для такого варианта составит 768 битов. Такой вариант резко увеличивает сложность вскрытия алгоритма гр убой силой, сложность такого вскрытия составит 2^{768}.

Однако возможно использование вскрытия "встреча посередине" (см. раздел 15.1). Сложность такого вскр ытия уменьшается до 2^{384}, что, тем не менее, вполне достаточно для обеспечения любой мыслимой безопасн ости.

Хотя независимые подключи мешают линейному криптоанализу, этот вариант чувствителен к дифференц иальному криптоанализу и может быть вскрыт с помощью 2^{61} выбранных открытых текстов (см. -3-й) [167, 172]. По видимому, никакая модификация распределения ключей не сможет н амного усилить DES.

DESX

DESX - это вариант DES, разработанный RSA Data Security, Inc., и включенный в 1986 году в программу обеспечения безопасности электронной почты MailSafe, а в 1987 году в набор BSAFE. DESX использует метод, называемый отбеливанием (см. раздел 15.6), для маскировки входов и выходов DES. Кроме 56-битового ключа DES в DESX используется дополнительный 64-битовый ключ отбеливания. Эти 64 бита используются для в ыполнения операции XOR с блоком открытого текста перед первым этапом DES. Дополнительные 64 бита, я вляющиеся результатом применения однонаправленной функции к полному 120-битовому ключу DESX, испол ьзуются для выполнения XOR с шифротекстом, полученным в результате последнего этапа [155]. По сравнению с DES отбеливание значительно повышает устойчивость DESX к вскрытию грубой силой, вскрытие требует $(2^{120})/n$ операций при n известных открытых текстах. Также повышается устойчивость к дифференциальному и линейному криптоанализу, для вскрытия потребуется 2^{61} выбранных и 2^{60} известных открытых текстов, соответственно [1338].

CRYPT(3)

CRYPT(3) представляет собой вариант DES, используемый в системах UNIX. Он в основном используется в качестве однонаправленной функции для паролей, но иногда может быть использован и для шифрования. Ра зличие между CRYPT(3) и DES состоит в том, что в CRYPT(3) включена независимая от ключа перестановка с расширением с 2^{12} вариантами. Это сделано для того, чтобы для создания аппаратного устройства вскрытия паролей нельзя было использовать промышленные микросхемы DES.

Обобщенный DES

Обобщенный DES (Generalized DES, GDES) был спроектирован для ускорения DES и повышения устойч ивости алгоритма [1381, 1382]. Общий размер блока увеличился, а количество вычислений осталось неизме нным.

На 1-й показана поблочная диаграмма GDES. GDES работает с блоками открытого текста переменной дл ины. Блоки шифрования делятся на q 32-битовых подблоков, точное число которых зависит от полного размера блока (который по идее может меняться, но фиксирован для конкретной реализации). В общем случае q равно размеру блока, деленному на 32.

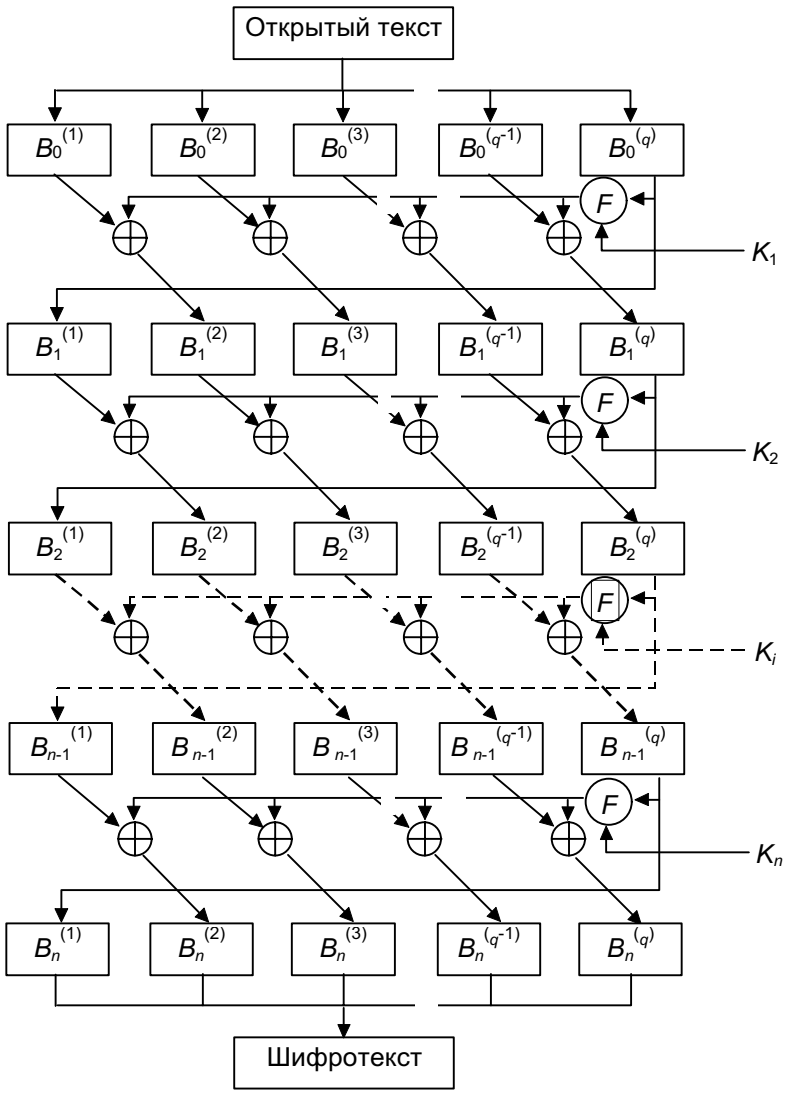


Рис. 12-11. GDES.

Функция f для каждого этапа рассчитывается один раз для крайнего правого блока. Результат при помощи операции XOR объединяется со всеми остальными частям, которые затем циклически смещаются направо. GDES использует переменное число этапов n. В последний этап внесено незначительное изменение, чтобы пр о-цессы шифрования и дешифрирования отличались только порядком подключей (точно также, как в DES). Де й-ствительно, если $q = 2$ и $n = 16$, то описанный алгоритм превращается в DES.

Бихам и Шамир [167, 168] показали, что дифференциальный криптоанализ вскрывает GDES с $q = 8$ и $n = 16$ с помощью всего шести выбранных открытых текстов. При использовании независимых подключей требуется 16 выбранных открытых текстов. GDES с $q = 8$ и $n = 22$ вскрывается с помощью всего 48 выбранных открытых текстов, а для вскрытия GDES с $q = 8$ и $n = 31$ требуется всего 500000 выбранных открытых текстов. Даже GDES с $q = 8$ и $n = 64$ слабее, чем DES - для его вскрытия нужно только 249 выбранных открытых текстов. Действительно, любая более быстрая, чем DES, схема GDES является также и менее безопасной (см. -3-й).

Недавно появился еще один вариант этой схемы [1591]. Возможно он не более безопасен, чем оригинальный GDES. Общем случае любой вариант DES с большими блоками, который быстрее DES, скорее всего менее безопасен по сравнению с DES.

DES с измененными S-блоками

Другие модификации DES связаны с S-блоками. В некоторых проектах используется переменный порядок S-блоков. Другие разработчики меняют содержание самих S -блоков. Бихам и Шамир показали [170,172], что построение S-блоков и даже их порядок оптимальны с точки зрения устойчивости к дифференциальному кри п-тоанализу:

> Изменение порядка восьми S-блоков DES (без изменения их значений) также значительно ослабляет DES: DES с 16 эт а-пами и конкретным измененным порядком вскрывается примерно за 2^{38} шагов. ... Доказано, что DES со случайными S-блоками вскрыть очень легко. Даже минимальное изменение одного из элементов S_блоков DES может снизить устойч и-

вость DES к вскрытию.

S-блоки DES не были оптимизированы против линейного криптоанализа. Существуют и лучшие S-блоки, чем предлагаемые в DES, но бездумная замена S-блоков новыми - не самая лучшая идея.

В -3-й [167, 169] перечислены некоторые модификации DES и количество выбранных открытых текстов, нужное для выполнения дифференциального криптоанализа. В таблицу не включена одна из модификаций, об ъединяющая левую и правую половины с помощью сложения по модулю 24 вместо XOR, ее в 2^{17} раз труднее вскрыть, чем DES [689].

RDES

RDES - это модификация, в которой в конце каждого этапа обмениваются местами правая и левая половины с использованием зависимой от ключа перестановки [893]. Обмены местами фиксированы и зависят только от ключа. Это означает, что может быть 15 обменов, зависимых от ключа, и 2^{15} возможных вариантов, а также что эта модификация не устойчива по отношению к дифференциальному криптоанализу [816, 894, 112]. У RDES большое количество слабых ключей. Действительно, почти каждый ключ слабее, чем типичный ключ DES. И с-пользовать эту модификацию нельзя.

Лучшей является идея выполнять обмен местами только в пределах правой половины и в начале каждого этапа. Другой хорошей идеей является выполнение обмена в зависимости от входных данных, а не как статич е-ской функции ключа. Существует множество возможных вариантов [813, 815]. В RDES-1 используется завис я-щая от данных перестановка 16-битовых слов в начале каждого этапа. В RDES-2 применяется зависящая от данных перестановка байтов в начале каждого этапа после 16-битовых перестановок, аналогичных RDES-1. Развитием этой идеи является RDES-4, и т.д. RDES-1 устойчив и к дифференциальному [815], и к линейному криптоанализу [1136]. По видимому, RDES-2 и последующие варианты достаточно хороши.

Табл. 12-15.
Вскрытия вариантов DES с помощью дифференциального криптоанализа

Изменение работы	Количество выбранных открытых текстов
Полный DES (без изменений)	2^{47}
P-перестановка	Не может усилить
Тождественная перестановка	2^{19}
Порядок S-блоков	2^{38}
Замена XOR сложениями	2^{39}, 2^{31}
S-блоки	
Случайные	2^{18} - 2^{20}
Случайные перестановки	2^{33} - 2^{41}
Одноэлементные	2^{33}
Однородные таблицы	2^{26}
Удаление E-расширения	2^{26}
Порядок E-расширения и XOR подключа	2^{44}
GDES (ширина q=8)	
16 этапов	6, 16
64 этапа	2^{49} (независимый ключ)

s"DES

Группа корейских исследователей под руководством Кванджо Кима (Kwangjo Kim) попыталась найти набор S-блоков, оптимально устойчивых и против дифференциального, и против линейного криптоанализа. Их первая попытка, известная как s^2DES, представленная в [834], оказалась, как было показано в [855, 858], менее усто й-чивой, чем DES, против дифференциального криптоанализа. Следующий вариант, s^3DES, был представлен в [839] и оказался менее устойчив, чем DES, к линейному криптоанализу [856, 1491, 1527, 858, 838]. Бихам пре д-

ложил незначительно изменить алгоритм, чтобы сделать s^3DES безопасным по отношению и к дифференциальному, и к линейному криптоанализу [165]. Исследователи вернулись к своим компьютерам и разработали улучшенную технику проектирования S-блоков [835, 837]. Они предложили s^4DES [836], а затем s^5DES [838, 944].

В -4-й приведены для s3DES (с обращенными S-блоками 1 и 2), которые безопасны по отношению к обоим видам криптоанализа. Использование этого варианта вместе с трехкратным DES наверняка помешает крипто анализу.

DES с S-блоками, зависящими от ключа

Линейный и дифференциальный криптоанализ работают только, если аналитику известно строение S-блоков. Если S-блоки зависят от ключа и выбираются криптографически сильным методом, то линейный и дифференциальный криптоанализ значительно усложнятся. Хотя надо помнить, что даже у хранящихся в секрете случайно созданных S-блоков очень плохие дифференциальные и линейные характеристики.

Табл. 12-16.
S-блоки s3DES (с обращенными S-блоками 1 и 2)

S-блок 1:															
13	14	0	3	10	4	7	9	11	8	12	6	1	15	2	5
8	2	11	13	4	1	14	7	5	15	0	3	10	6	9	12
14	9	3	10	0	7	13	4	8	5	6	15	11	12	1	2
1	4	14	7	11	13	8	2	6	3	5	10	12	0	15	9
S-блок 2:															
15	8	3	14	4	2	9	5	0	11	10	1	13	7	6	12
6	15	9	5	3	12	10	0	13	8	4	11	14	2	1	7
9	14	5	8	2,	4	15	3	10	7	6	13	1	11	12	0
10	5	3	15	12	9	0	6	1	2	8	4	11	14	7	13
S-блок 3:															
13	3	11	5	14	8	0	6	4	15	1	12	7	2	10	9
4	13	1	8	7	2	14	11	15	10	12	3	9	5	0	6
6	5	8	11	13	14	3	0	9	2	4	1	10	7	15	12
1	11	7	2	8	13	4	14	6	12	10	15	3	0	9	5
S-блок 4:															
9	0	7	11	12,	5	10	6	15	3	1	14	2	8	4	13
5	10	12	6	0	15	3	9	8	13	11	1	7	2	14	4
10	7	9	12	5	0	6	11	3	14	4	2	8	13	15	1
3	9	15	0	6	10	5	12	14	2	1	7	13	4	8	11
S-блок 5:															
5	15	9	10	0	3	14	4	2	12	7	1	13	6	8	11
6	9	3	15	5	12	0	10	8	7	13	4	2	11	14	1
15	0	10	9	3	5	4	14	8	11	1	7	6	12	13	2
12	5	0	6	15	10	9	3	7	2	14	11	8	1	4	13
S-блок 6:															
4	3	7	10	9	0	14	13	15	5	12	6	2	11	1	8
14	13	11	4	2	7	1	8	9	10	5	3	15	0	12	6
13	0	10	9	4	3	7	14	1	15	6	12	8	5	11	2
1	7	4	14	11	8	13	2	10	12	3	5	6	15	0	9

S-блок 7:

4	10	15	12	2	9	1	6	11	5	0	3	7	14	13	8
10	15	6	0	5	3	12	9	1	8	11	13	14	4	7	2
2	12	9	6	15	10	4	1	5	11	3	0	8	7	14	13
12	6	3	9	0	5	10	15	2	13	4	14	7	11	1	8

S-блок 8:

13	10	0	7	3	9	14	4	2	15	12	1	5	6	11	8
2	7	13	1	4	14	11	8	15	12	6	10	9	5	0	3
4	13	14	0	9	3	7	10	1	8	2	11	15	5	12	6
8	11	7	14	2	4	13	1	6	5	9	0	12	15	3	10

Вот как можно использовать 48 дополнительных битов ключа для создания S-блоков, устойчивых как к л инейному, так и к дифференциальному криптоанализу [165].

(1) Изменить порядок S-блоков DES: 24673158.

(2) Выбрать 16 из оставшихся битов ключа. Если первый бит 1, обменять местами первые и последние два ряда S-блока 1. Если второй бит 1, обменять местами первые и последние восемь столбцов S-блока 1. П овторить то же самое для третьего и четвертого битов и S-блока 2. Повторить то же самое для S-блоков с 3 по 8.

(3) Взять оставшиеся 32 бита ключа. Выполнить XOR первых четырех битов с каждым элементом S-блока 1, XOR следующих четырех битов с каждым элементом S-блока 2, и так далее.

Сложность вскрытия такой системы с помощью дифференциального криптоанализа составит 251, с пом ощью линейного криптоанализа - 2^{53}. Сложность исчерпывающего перебора составит 2102.

Что хорошо в этом варианте DES так это то, что он может быть реализован в существующей аппаратуре. Различные поставщики микросхем DES продают микросхемы DES с возможностью загрузки S-блоков. Можно реализовать любой способ генерации S-блоков вне микросхемы и затем загрузить их в нее. Для дифференц иального и линейного криптоанализа нужно так много известных или выбранных открытых текстов, что эти сп особы вскрытия становятся неосуществимыми. Вскрытие грубой силой также трудно себе представить, не пом ожет никакое увеличение скорости.

12.7 Насколько безопасен сегодня DES?

Ответ одновременно и прост, и труден. При простом ответе учитывается только длина ключа (см. раздел 7.1). Машина для вскрытия DES грубой силой, способная найти ключ в среднем за 3.5 часа, в 1993 году стоила 1 миллион долларов [1597, 1598]. DES используется очень широко, и наивно было бы предполагать, что NSA и аналогичные организации в других странах не построили по такому устройству. И не забывайте, что стоимость уменьшается в 5 раз каждые 10 лет. С течением времени DES будет становиться все менее и менее безопасным.

Для трудного ответа нужно попытаться оценить криптоаналитические методы. Дифференциальный крипто анализ был известен в NSA задолго до середины 70-х, когда DES впервые стал стандартом. Наивно считать, что с тех пор теоретики NSA ничего не делали, почти наверняка они разработали новые криптоаналитические мет оды, которые можно использовать против DES. Но фактов у нас нет, одни слухи.

Винн Шварцтау (Winn Schwartau) пишет, что NSA построило огромную параллельную машину для вскр ытия DES уже в середине 80-х [1404]. По крайней мере одна такая машина была построена в Harris Corp. С и спользованием Cray Y-MP. Предположительно существует ряд алгоритмов, которые на несколько порядков уменьшают сложность вскрытия DES грубой силой. Контекстные алгоритмы, основанные на внутренней работе DES, позволяют отбросить ряд ключей, используя частичные решения. Статистические алгоритмы уменьшают эффективную длину ключа еще сильнее. Другие алгоритмы также проверяют вероятные ключи - слова, печ атаемые последовательности ASCII, и т.д. (см. раздел 8.1). По слухам NSA может вскрыть DES за время от 3 до 15 минут, в зависимости от того коков будет выполненный объем предварительной обработки. И каждая такая машина стоит порядка 50000 долларов.

Согласно другим слухам, если у NSA есть большое количество открытых текстов и шифротекстов, его эк сперты могут выполнить некоторые статистические расчеты и затем считать ключ из архива на оптических ди сках.

И то, что это только слухи, не дает мне чувство уверенности в DES. Этот алгоритм очень долго был очень большой мишенью. Почти любое изменение DES послужит дополнительной защитой, может быть получивши йся шифр и будет менее устойчив к вскрытию, но у NSA может не оказаться средств решения этой конкретной задачи.

Я рекомендую использовать схему Бихама для зависящих от ключа S-блоков. Она может быть легко реал изована программно или аппаратно (с помощью микросхем с загружаемыми S-блоками), и не приводит к потере эффективности по сравнению с DES. Эта схема повышает устойчивость алгоритма к вскрытию грубой силой, усложняет дифференциальный и линейный криптоанализ и заставляет NSA столкнуться с алгоритмом, по кра йней мере таким же сильным как DES, но другим.

Глава 13 Другие блочные шифры

13.1 LUCIFER

В конце 60-х IBM начала выполнение исследовательской программы по компьютерной криптографии, наз ыванной Люцифером (Lucifer) и руководимой сначала Хорстом Фейстелем (Horst Feistel), а затем Уолтом Тачм еном (Walt Tuchman). Это же название - Lucifer - получил блочный алгоритм, появившийся в результате этой программыв начале 70-х [1482, 1484]. В действительности существует по меньшей мере два различных алг оритма с таким именем [552, 1492]. [552] содержит ряд пробелов в спецификации алгоритма. Все это привело к заметной путанице.

Lucifer - это набор перестановок и подстановок, его блоки похожи на блоки DES. В DES результат функции f объединяется с помощью XOR со входом предыдущего этапа, образуя вход следующего этапа. У S-блоков алг оритма Lucifer 4-битовые входы и 4-битовые выходы, вход S-блоков представляет собой перетасованный выход S_блоков предыдущего этапа, входом S-блоков первого этапа является открытый текст. Для выбора использу емого S-блока из двух возможных применяется бит ключа. (Lucifer реализует это, как один T-блок с 9 битами на входе и 8 битами на выходе.) В отличие от DES половины блока между этапами не переставляются и вообще понятие половины блока не используется в алгоритме Lucifer. У этого алгоритма 16 этапов, 128-битовые блоки и более простое, чем в DES, распределение ключей.

Применив дифференциальный криптоанализ к первой реализации Lucifer'a, Бихам и Шамир [170, 172] пок азали, что Lucifer с 32-битовыми блоками и 8 этапами может быть взломан с помощью 40 выбранных открытых текстов за 2^{39} шагов, тот же способ позволит вскрыть Lucifer с 128-битовыми блоками и 8 этапами с помощью 60 выбранных открытых текстов за 2^{53} шагов. 18-этапный, 128-битовый Lucifer вскрывается дифференциал ьным криптоанализом с помощью 24 выбранных открытых текстов за 2^{21} шагов. Все эти вскрытия использовали сильные S-блоки DES. Применив дифференциальный криптоанализ против второй реализации Lucifer, Бихам и Шамир обнаружили, что S-блоки намного слабее, чем в DES. Дальнейший анализ показал, что более половины возможных ключей не являются безопасными [112]. Криптоанализ со связанными ключами может взломать 128-битовый Lucifer с любым числом этапов с помощью 2^{33} выбранных открытых текстов для выбранных кл ючей или 2^{65} известных открытых текстов для выбранных ключей [158]. Вторая реализация Lucifer еще слабее [170, 172, 112].

Некоторые думают, что Lucifer безопаснее, чем DES, из-за большей длины ключа и малого количества опу бликованных сведений. Но очевидно, что это не так.

Lucifer является объектом нескольких патентов США: [553, 554, 555, 1483]. Сроки действия всех этих п атентов истекли.

13.2 MADRYGA

В.Е. Мадрига (W. E. Madryga) предложил этот блочный алгоритм в 1984 году [999]. Он может быть эффе ктивно реализован как программа: в нем нет надоедливых перестановок, и все операции выполняются над ба йтами. Стоит перечислить задачи, которые решал автор при проектировании алгоритма:

1. Открытый текст нельзя получить из шифротекста без помощи ключа. (Это означает только то, что а лгоритм безопасен.)
2. Количество операций, нужное для определения ключа по имеющимся шифротексту и открытому те ксту, должно быть статистически равно произведению количества операций при шифровании на число возможных ключей. (Это означает, что никакое вскрытие с открытым текстом не может быть лучше, чем вскрытие грубой силой.)
3. Известность алгоритма не влияет на силу шифра. (Безопасность полностью опр еделяется ключом.)
4. Изменение одного бита ключа должно вызывать для того же открытого текста радикальное изменение шифротекста, и Изменение одного бита открытого текста должно вызывать для того же ключа рад икальное изменение шифротекста. (Это лавинный э ффект.)
5. Алгоритм должен содержать некоммутативную комбинацию подстановок и перестан овок.
6. Подстановки и перестановки, используемые в алгоритме, должны определяться и входными данными, и ключом.
7. Избыточные группы битов открытого текста должны быть полностью замаскированы в шифротексте.
8. Длина шифротекста должна равняться длине открытого текста.

9. Не должно быть простых взаимосвязей между любыми возможными ключами и особенностями ши фротекста.
10. Все возможные ключи должны давать сильный шифр. (Не должно быть слабых кл ючей.)
11. Длина ключа и текста могут регулироваться для реализации различных требований к безопасности.
12. Алгоритм должен позволять эффективную программную реализацию на больших мэйнфреймах, м иникомпьютерах, микрокомпьютерах и с помощью дискретной логики. (По сути используемые в алг оритме функции ограничены XOR и битовым сдвигом.)

DES удовлетворял первым девяти требованиям, но последние три были новыми. В предположении, что лу чшим способом вскрытия алгоритма является грубая сила, переменная длина ключа, конечно же, заставит з амолчать тех, кто считает, что 56 битов - это слишком мало. Такие люди могут реализовать этот алгоритм с л юбой нужной им длиной ключа. А любой, кто когда-нибудь пытался реализовать DES программно, обрадуется алгоритму, который учитывает возможности программных реал изаций.

Описание Madryga

Madryga состоит из двух вложенных циклов. Внешний цикл повторяется восемь раз (но это количество м ожет быть увеличено для повышения) и содержит применение внутреннего цикла к открытому тексту. Внутре нний цикл превращает открытый текст в шифротекст, повторяясь для каждого 8-битового блока (байта) открыт ого текста. Следовательно, весь открытый текст восемь раз последовательно обрабатывается а лгоритмом.

Итерация внутреннего цикла оперирует с 3-байтовым окном данных, называемым рабочим кадром (см. 12-й). Это окно смещается на 1 байт за итерацию. (При работе с последними 2 байтами данные считаются цикл ически замкнутыми.) Первые два байта рабочего кадра циклически сдвигаются на переменное число позиций, а для последнего байта выполняется XOR с некоторыми битами ключа. По мере продвижения рабочего кадра все байты последовательно "вращаются" и подвергаются операции XOR с частями ключа. Последовательные вр ащения перемешивают результаты предыдущих операций XOR и вращения, а результат XOR влияет на вращ ение. Это делает весь процесс обратимым.

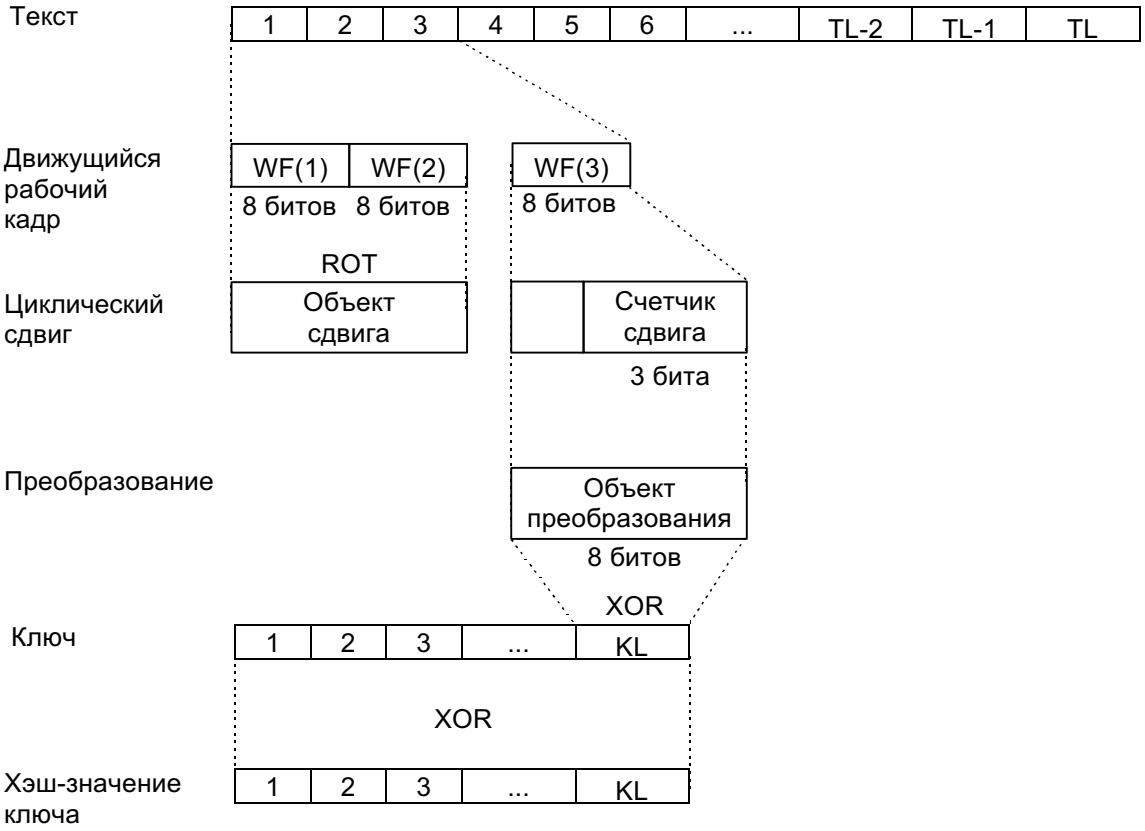


Рис. 13-1. Одна итерация Madryga.

Так как каждый байт данных влияет на два байта слева от себя и на один байт справа, после восьми прох одов каждый байт шифротекста зависит от 16 байтов слева и от восьми байтов справа.

При шифровании каждая итерация внутреннего цикла устанавливает рабочий кадр на предпоследний байт открытого текста и циклически перемещает его к байту открытого текста, третьему слева от последнего. Снач а-

ла весь ключ подвергается операции XOR со случайной константой и затем циклически смещается влево на 3 бита. Младшие три бита младшего байта рабочего кадра сохраняются, они определяют вращение остальных двух байтов. Затем для младшего байта рабочего кадра выполняется операция XOR с младшим байтом ключа. Далее объединение двух старших байтов циклически смещается влево на переменное число битов (от 0 до 7). Наконец рабочий кадр смещается вправо на один байт и весь процесс повторяется.

Смысл случайной константы в том, чтобы превратить ключ в псевдослучайную последовательность. Длина константы должна быть равна длине ключа. При обмене данными абоненты должны пользоваться константой одинаковой длины. Для 64-битового ключа Мадрига рекомендует константу 0x0f1e2d3c4b5a6978.

При дешифрировании процесс инвертируется. При каждой итерации внутреннего цикла рабочий кадр уст анавливается на байт, третий слева от последнего байта шифротекста, и циклически перемещается в обратном направлении до байта, который находится на 2 байта левее последнего байта шифротекста. И ключ, и 2 байта шифротекста в процессе циклически смещаются направо, а XOR выполняется перед циклическими сдвигами.

Криптоанализ и Madryga

Исследователи из Технического университета в Квинсланде (Queensland University of Technology) [675] и с следовали Madryga вместе с некоторыми другими блочными шифрами. Они обнаружили, что в этом алгоритме не проявляется лавинный эффект для преобразования открытого текста в шифротекст. Кроме того, во многих шифротекстах процент единиц был выше, чем процент нулей.

Хотя у меня нет сведений о проведении формального анализа этого алгоритма, он не производит впечатл ение супернадежного. При поверхностном знакомстве с ним Эли Бихам пришел к следующим выводам [160]:

> Алгоритм состоит только из линейных операций (циклическое смещение и XOR), незначительно изменяемых в завис имости от данных.
>
> В этом нет ничего похожего на мощь S-блоков DES.
>
> Четность всех битов шифротекста и открытого текста неизменна и зависит только от ключа. Поэтому, обладая открытым текстом и соответствующим шифротекстом, можно предсказать четность шифротекста для любого открытого текста.

По отдельности ни одно из этих замечаний не являются критическими, но этот алгоритм не вызывает у меня положительных эмоций. Я не рекомендую использовать Madryga.

13.3 NewDES

NewDES (новый DES) был спроектирован в 1985 году Робертом Скоттом (Robert Scott) как возможная зам ена DES [1405, 364]. Алгоритм не является модификацией DES, как может показаться из его названия. Он оп ерирует 64-битовыми блоками шифротекста, но использует 120-битовый ключ. NewDES проще, чем DES, в нем нет начальной и заключительной перестановок. Все операции выполняются над целыми байтами. (На самом деле NewDES ни коим образом не является новой версией DES, название было выбрано неудачно.)

Блок открытого текста делится на восемь 1-байтовых подблоков: B_0, B_1, . . ., B_6, B_7. Затем подблоки проходят через 17 этапов. В каждом этапе восемь действий. В каждом действии один из подблоков подвергается опер ации XOR с частью ключа (есть одно исключение), заменяется другим байтом с помощью функции f и затем подвергается операции XOR с другим подблоком, который и заменяется результатом. 120-битовый ключ дели тся на 15 подблоков ключа: K_0, K_1, . . ., K_{13}, K_{14}. Процесс легче понять, увидев его схему, чем прочитав его оп исание. Алгоритм шифрования NewDES показан на 11-й.

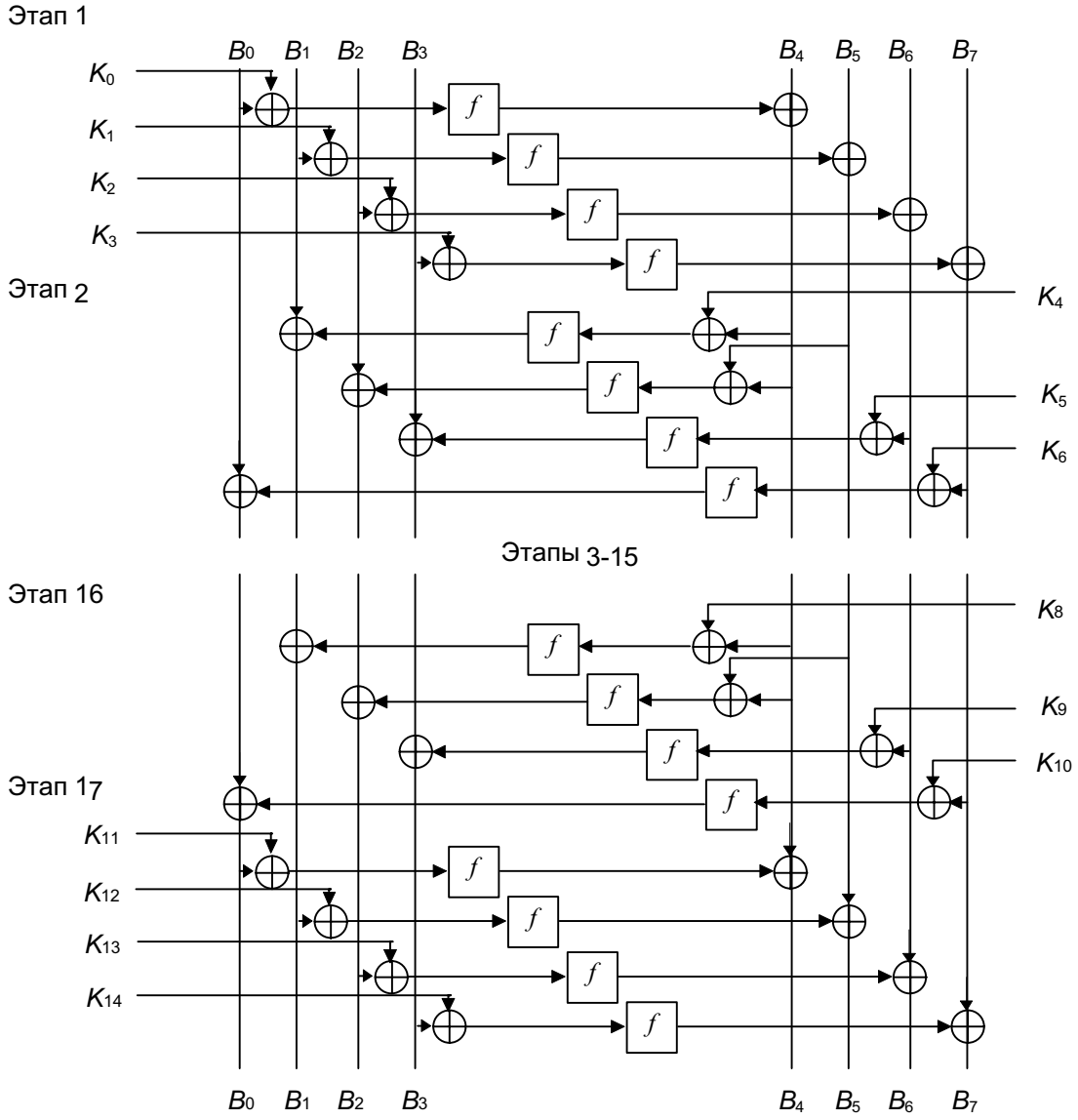


Рис. 13-2. NewDES.

Функция f выводится из Декларации независимости. Подробности можно найти в [1405].

Скотт показал, что каждый бит блока открытого текста влияет на каждый бит шифротекста уже после 7 этапов. Он также проанализировал функцию f и не нашел каких-либо очевидных проблем. NewDES обладает той же комплиментарностью, что и DES [364]: если $E_K(P\} = C$, то $E_K(P'\} = C'$. Это уменьшает объем работы, необходимой для вскрытия грубой силой, с 2^{110} действий до 2^{119}. Бихам заметил, что любое изменение полного байта, примененное ко всем байтам ключа и данных, также приводит к комплиментарности [160]. Это уменьшает объем грубого вскрытия до 2^{112} действий.

Это не является критичным, но предложенное Бихамом криптоаналитическое вскрытие со связанными ключами может вскрыть NewDES с помощью 2^{33} выбранных открытых текстов для выбранных ключей за 2^{48} действий [160]. Хотя такое вскрытие требует много времени и в большой степени является теоретическим, оно показывает, что NewDES слабее, чем DES.

13.4 FEAL

FEAL был предложен Акихиро Шимузу (Akihiro Shimizu) Шоджи Миягучи (Shoji Miyaguchi) из NTT Japan [1435]. В нем используются 64-битовый блок и 64-битовый ключ. Его идея состоит в том, чтобы создать алгоритм, подобный DES, но с более сильной функцией этапа. Используя меньше этапов, этот алгоритм мог бы работать быстрее. К несчастью действительность оказалась далека от целей проекта.

Описание FEAL

На 10-й представлена блок-схема одного этапа FEAL. В качестве входа процесса шифрования используется 64-битовый блок открытого текста. Сначала блок данных подвергается операции XOR с 64 битами ключа. З а-

тем блок данных расщепляется не левую и правую половины. Объединение левой и правой половин с помощью XOR образует новую правую половину. Левая половина и новая правая половина проходят через n этапов (первоначально четыре). На каждом этапе правая половина объединяется с помощью функции f с шестнадцатью битами ключа и с помощью XOR - с левой половиной, создавая новую правую половину. Исходная правая п о-ловина (на начало этапа) становится новой левой половиной. После n этапов (не забывайте, что левая и правая половины не переставляются после n-го этапа) левая половина снова объединяется с помощью XOR с правой половиной, образуя новую правую половину, затем левая и правая соединяются вместе в 64-битовое целое. Блок данных объединяется с помощью XOR с другими 64 битами ключа, и алгоритм завершается.

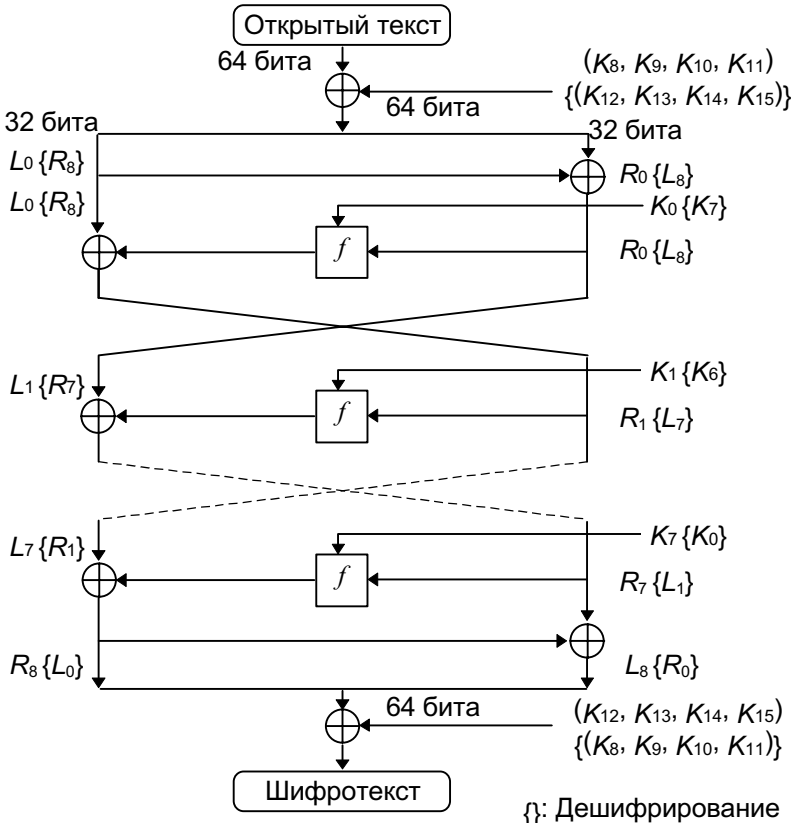


Рис. 13-3. Один этап FEAL.

Функция f берет 32 бита данных и 16 битов ключа и смешивает их вместе. Сначала блок данных разбивается на 8-битовые кусочки, которые затем объединяются с помощью XOR и заменяют друг друга. Блок-схема фун к-ции f представлена на 9-й. Две функции S_0и S_1 определяются следующим образом:

$S_0(a,b)$ = циклический сдвиг влево на два бита ((a + b) mod 256)

$S_1(a,b)$ = циклический сдвиг влево на два бита((a + b + 1) mod 256)

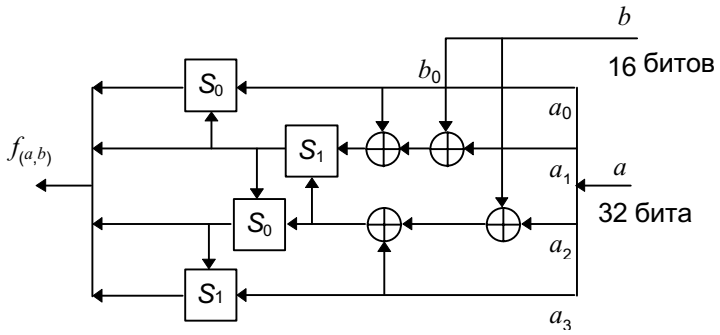


Рис. 13-4. Функция f.

Тот же алгоритм может быть использован для дешифрирования. Единственным отличием является то, что при дешифрировании порядок использования частей ключа меняется на обратный.

На 8-й представлена блок-схема функции генерации ключа. Сначала 64-битовый ключ делится на две пол о-

вины, к которым применяются операции XOR и функции f_k, как показано на схеме. На 7-й показана блок-схема функции f_k. Два 32-битовых входа разбиваются на 8-битовые блоки, объединяемые и заменяемые в соответс твии со схемой. S_0 и S_1 определяются, как показано на рисунке. Затем в алгоритме шифрования/дешифрирования используются 16-битовые блоки ключа.

На микропроцессоре 80286/10 МГц ассемблерная реализация FEAL-32 может шифровать данные со скор остью 220 Кбит/с. FEAL-64 может шифровать данные со скоростью 120 Кбит/с [1104].

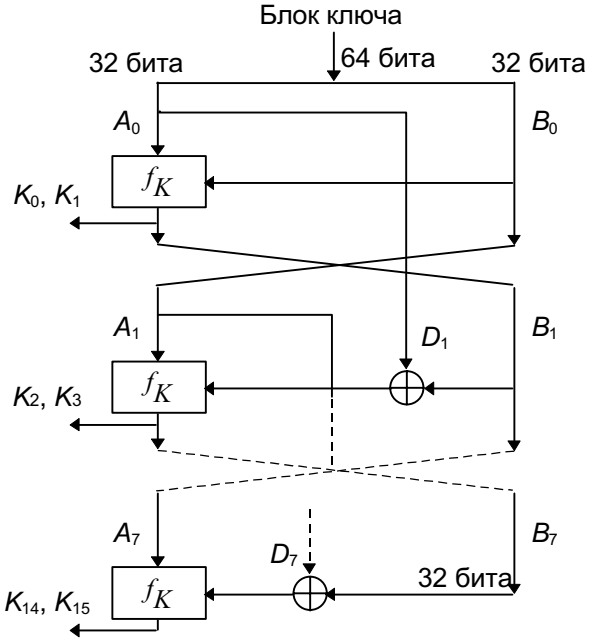
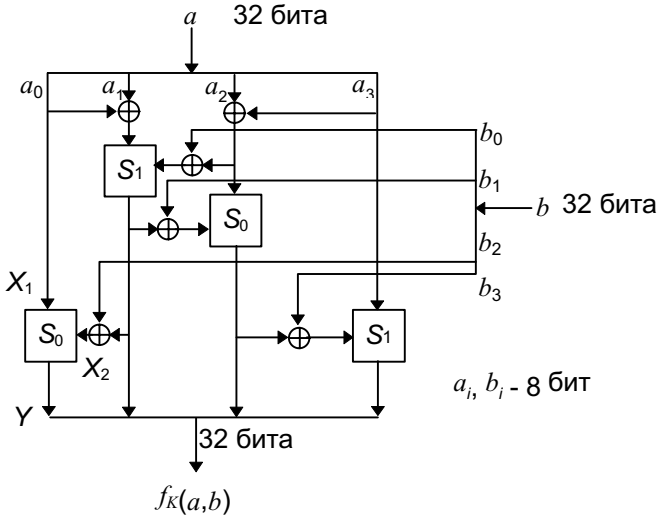


Рис. 13-5. Обработка ключа в FEAL.



$Y=S_0(X_1,X_2)=\mathrm{Rot2}((X_1+X_2) \bmod 256)$
$Y=S_1(X_1,X_2)=\mathrm{Rot2}((X_1+X_2+1) \bmod 256)$
Y: выходные 8 битов, X_1,X_2 (8 битов): входы
Rot2(Y): циклический сдвиг влево на 2 бита 8-битовых данных Y

Рис. 13-6. Функция f_K.

Криптоанализ FEAL

Успешный криптоанализ FEAL-4, FEAL с четырьмя этапами, был выполнен с помощью вскрытия с выбра нными открытыми текстами [201], а позже слабость этого алгоритма была показана в [1132]. Последнее вскр ытие, выполненное Сином Мерфи (Sean Murphy), было первым опубликованным вскрытием, использовавшим дифференциальный криптоанализ, и для него потребовалось только 20 выбранных открытых текстов. Ответом разработчиков стал 8-этапный FEAL [1436, 1437, 1108], криптоанализ которого был представлен Бихамом и

Шамиром на конференции SECURICOM '89 [1424]. Для вскрытия FEAL-8 с выбранными открытыми текстами потребовалось только 10000 блоков [610], что заставило разработчиков алгоритма засучить рукава и опред елить FEAL-N [1102, 1104], алгоритм с переменным числом этапов (конечно же, большим 8).

Бихам и Шамир применили против FEAL-N дифференциальный криптоанализ, хотя они могли бы еще б ыстрее вскрыть его грубой силой (с помощью менее, чем 2^{64} шифрований выбранного открытого текста) для N , меньшего 32. [169]. Для вскрытия FEAL-16 нужно 2^{28} выбранных или $2^{46.5}$ известных открытых текстов. Для вскрытия FEAL-8 требуется 2000 выбранных или $2^{37.5}$ известных открытых текстов. FEAL-4 может быть вскрыт с помощью всего 8 правильно выбранных открытых текстов.

Разработчики FEAL определили также модификацию FEAL - FEAL-NX, в которой используется 128-битовый ключ (см. 6-й) [1103, 1104]. Бихам и Шамир показали, что для любого значения N FEAL-NX со 128-битовым ключом взламывать не сложнее, чем FEAL-N с 64-битовым ключом [169]. Недавно был предложен FEAL-N(X)S, усиливающий FEAL за счет динамической фун кции обмена местами [1525].

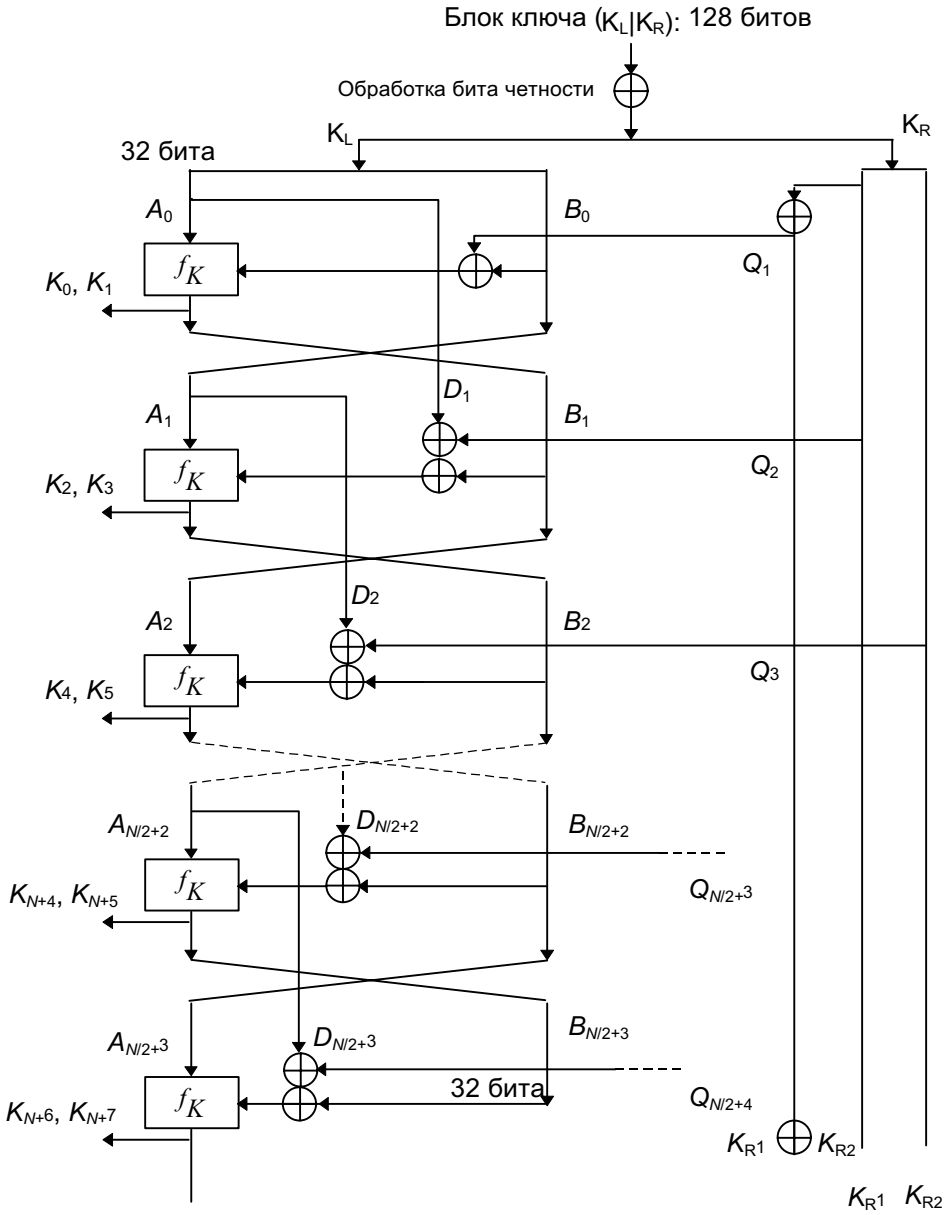


Рис. 13-7. Обработка ключа в FEAL-NX.

Более того. В [1520] было представлено другое вскрытие FEAL-4, требующее только 1000 известных откр ытых текстов, и FEAL-8, для которого нужно только 20000 известных открытых текстов. Другие вскрытия прив едены в [1549, 1550]. Наилучшим является выполненное Мицуру Мацуи (Mitsuru Matsui) и Атшуиро Ямагиши

(Atshuiro Yamagishi) [1020]. Это было первое применение линейного криптоанализа, и оно позволило вскрыть FEAL-4 с помощью 5 известных открытых текстов, FEAL-6 - с помощью 100 известных открытых текстов, а FEAL-8 - с помощью 2^{15} известных открытых текстов. Дальнейшие уточнения можно найти в [64]. Дифференциальный криптоанализ позволяет вскрывать FEAL-8, используя только 12 выбранных открытых текстов [62]. Кто бы не изобрел новый метод криптоаналитического вскрытия, кажется, что он всегда сначала пробует его на FEAL.

Патенты

FEAL запатентован в Соединенных Штатах [1438], соответствующие патенты приняты к рассмотрению в Англии, Франции и Германии. Желающий лицензировать использование алгоритма должен связаться с Департаментом интеллектуальной собственности (Intellectual Property Department), NTT, 1-6 Uchisaiwai-cho, 1-chome, Chiyada-ku, 100 Japan.

13.5 REDOC

REDOC II представляет собой другой блочный алгоритм, разработанный Майклом Вудом (Michael Wood) для Cryptech, Inc. [1613, 400]. В нем используются 20-байтовый (160-битовый) ключ и 80-битовый блок.

REDOC II выполняет все манипуляции - перестановки, подстановки и XOR с ключом - с байтами, этот алгоритм эффективен при программной реализации. REDOC II использует меняющиеся табличные функции. В отличие от DES, имеющего фиксированный (хотя и оптимизированных для безопасности) набор таблиц подстановок и перестановок REDOC II использует зависимые от ключа и открытого текста наборы таблиц (по сути S-блоков). У REDOC II 10 этапов, каждый этап представляет собой сложную последовательность манипуляций с блоком.

Другой уникальной особенностью является использование **масок**, которые являются числами, полученными из таблицы ключей, и используются для выбора таблиц данной функции для данного этапа. Для выбора таблиц функции используются как значение данных, так и маски.

При условии, что самым эффективным средством вскрытия этого алгоритма является грубая сила, REDOC II очень надежен: для вскрытия ключа требуется 2^{160} операций. Томас Кузик (Thomas Cusick) выполнил криптоанализ одного этапа REDOC II, но ему не удалось расширить вскрытие на несколько этапов [400]. Используя дифференциальный криптоанализ, Бихам и Шамир достигли успеха в криптоанализе одного этапа REDOC II с помощью 2300 выбранных открытых текстов [170]. Они не смогли расширить это вскрытие на несколько этапов, но им удалось получить три значения маски после 4 этапов. О других попытках криптоанализа мне не известно.

REDOC III

REDOC представляет собой упрощенную версию REDOC II, также разработанную Майклом Вудом [1615]. Он работает с 80-битовым блоком. Длина ключа может меняться и достигать 2560 байтов (20480 битов). Алгоритм состоит только из операций XOR для байтов ключа и открытого текста, перестановки или подстановки не используются.

(1) Создать таблицу ключей из 256 10-байтовых ключей, используя секретный ключ.

(2) Создать 2 10-байтовых блока маски M_1 и M_2. M_1 представляет собой XOR первых 128 10-байтовых ключей, а M_2 - XOR вторых 128 10-байтовых ключей.

(3) Для шифрования 10-байтового блока:

   (a) Выполнить XOR для первого байта блока данных и первого байта M_1. Выбрать ключ из таблицы ключей, рассчитанной на этапе (1). Использовать вычисленное значение XOR в качестве индекса таблицы. Выполнить XOR каждого, кроме первого, байта блока данных с соответствующим байтом выбранного ключа.

   (b) Выполнить XOR для второго байта блока данных и второго байта M_1. Выбрать ключ из таблицы ключей, рассчитанной на этапе (1). Использовать вычисленное значение XOR в качестве индекса таблицы. Выполнить XOR каждого, кроме второго, байта блока данных с соответствующим байтом выбранного ключа.

   (c) Продолжать для всего блока данных (для байтов с 3 по 10), пока каждый байт не будет использован для выбора ключа из таблицы после выполнения для него XOR с соответствующим значением M_1. Затем выполнить XOR с ключом для каждого, кроме использованного для выбора ключа, байта.

   (d) Повторить для M_2 этапы (a)-(c).

Этот алгоритм несложен и быстр. На 33 мегагерцовом процессоре 80386 он шифрует данные со скоростью

2.75 Мбит/с. Вуд оценил, что конвейеризированная реализация на СБИС с 64 битовой шиной данных могла бы шифровать данные со скоростью свыше 1.28 Гбит/с при тактовой частоте 20 МГц.

REDOC III не безопасен [1440]. Он чувствителен к дифференциальному криптоанализу. Для восстановления обеих масок нужно всего примерно 223 выбранных открытых текстов.

Патенты и лицензии

Обе версии REDOC запатентованы в Соединенных штатах [1614]. Рассматриваются и иностранные патенты. При заинтересованности в REDOC II или REDOC III обращайтесь к Майклу Вуду (Michael C. Wood, Delta Computec, Inc., 6647 Old Thompson Rd., Syracuse, NY 13211).

13.6 LOKI

LOKI разработан в Австралии и впервые был представлен в 1990 году в качестве возможной альтернативы DES [273]. В нем используются 64-битовый блок и 64-битовый ключ. Общая структура алгоритма и использ о-вания ключа описана в [274, 275], а схема S-блоков - в [1247].

Используя дифференциальный криптоанализ, Бихам и Шамир смогли взломать LOKI с 11 и менее этапами быстрее, чем грубой силой [170]. Более того, алгоритм обладает 9-битовой комплиментарностью, что уменьш а-ет сложность вскрытия грубой силой в 256 раз [170, 916, 917].

Ларс Кнудсен (Lars Knudsen) показал, что LOKI с 14 и менее этапами чувствителен к дифференциальному криптоанализу [852, 853]. Кроме того, если в LOKI используются альтернативные S-блоки, получающийся шифр вероятно также будет чувствителен к дифференциальному криптоанализу.

LOKI91

В ответ на эти вскрытия разработчики LOKI вернулись за чертежную доску и пересмотрели свой алгоритм. Результатом было появление LOKI91 [272]. (Предыдущая версия LOKI была переименована в LOKI89.)

Чтобы повысить устойчивость алгоритма к дифференциальному криптоанализу и избавиться от комплиме н-тарности, в оригинальный проект были внесены следующие изменения:

1. Алгоритм генерации подключей был изменен так, чтобы половины переставлялись не после каждого, а после каждого второго этапа.
2. Алгоритм генерации подключей был изменен так, чтобы количество позиций циклического сдвига л е-вого подключа было равно то 12, то 13 битам.
3. Были устранены начальная и заключительная операции XOR блока и ключа.
4. Была изменена функция S-блока с целью сгладить XOR профили S-блоков (чтобы повысить их усто й-чивость к дифференциальному криптоанализу), и не допустить, чтобы для какого-то значения выпо л-нялось f(x) = 0, где f - это комбинация E-, S- и P-блоков.

Описание LOKI91

Механизм LOKI91 похож на DES (см. Рис. 13-8). Блок данных делится на левую и правую половины и пр о-ходит через 16 этапов, что очень походе на DES. На каждом этапе правая половина сначала подвергается оп е-рации XOR с частью ключа, а затем над ней выполняется перестановка с расширением (см. Табл. 13-1).

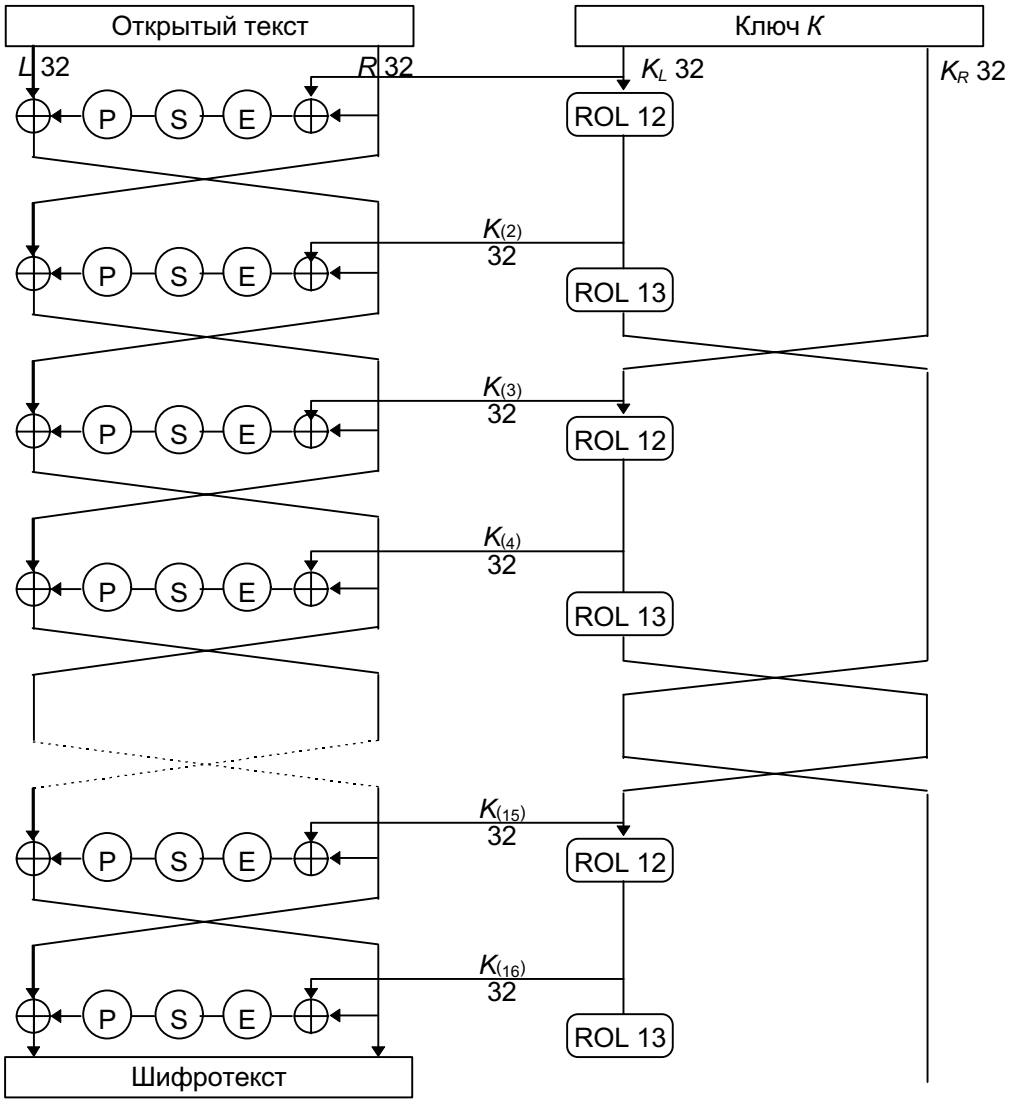


Рис. 13-8. LOKI91.

Табл. 13-1.
Перестановка с расширением

4,	3,	2,	1,	32,	31,	20,	29,	28,	27,	26,	25,
28,	27,	26,	25,	24,	23,	22,	21,	20,	19,	18,	17,
20,	19,	18,	17,	16,	15,	14,	13,	12,	11,	10,	9,
12,	11,	10,	9,	8,	7,	6,	5,	4,	3,	2,	1

48-битовый результат делится на четыре 12-битовых блока, для каждого из которых выполняется следующая подстановка с использованием S-блока: берется каждый 12-битовый вход, по 2 крайних левых и крайних пр а-вых бита используются для получения номера r, в 8 центральных бит образуют номер c. Результатом S-блока - *O* - является следующее значение:

$O(r,c) = (c + ((r * 17) \oplus 0xff) \,\&\, 0xff)^{31} \bmod P_r$

P_r приведено в Табл. 13-2.

Табл. 13-2.
P_r

r:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13,	14,	15,	16
P_r:	375,	279,	391,	395,	397,	415,	419,	425,	433,	445,	451,	463,	471,	477,	487,	488

Затем четыре 8-битовых результата снова объединяются, образуя 32-битовое число, которое подвергается операции перестановки, описанной в Табл. 13-3. Наконец для получения новой левой половины выполняется XOR правой половины с прежней левой половиной, а левая половина становится новой правой половиной. П осле 16 этапов для получения окончательного шифротекста снова выполняется XOR блока и ключа.

Табл. 13-3.
Перестановка с помощью P-блока

32,	24,	16,	8,	31,	23,	15,	7,	30,	22,	14,	6,	29,	21,	13,	5,
28,	20,	12,	4,	27,	19,	11,	3,	26,	18,	10,	2,	25,	17,	9,	1

Подключи из ключа выделяются достаточно прямолинейно. 64-битовый ключ разбивается на левую и пр авую половины. На каждом этапе подключом является левая половина. Далее она циклически сдвигается влево на 12 или 13 битов, затем после каждых двух этапов левая и правая половины меняются местами. Как и в DES для шифрования и дешифрирования используется один и тот же алгоритм с некоторыми изменениями в испол ьзовании подключей.

Криптоанализ LOKI91

Кнудсен предпринял попытку криптоанализа LOKI91 [854, 858], но нашел, что этот алгоритм устойчив к дифференциальному криптоанализу. Однако ему удалось обнаружить, что вскрытие со связанными ключами для выбранных открытых текстов уменьшает сложность вскрытия грубой силой почти вчетверо. Это вскрытие использует слабость использования ключа и может быть также применено, если алгоритм используется в кач естве однонаправленной хэш-функции (см. раздел 18.11).

Другое вскрытие со связанными ключами может вскрыть LOKI91 с помощью 2^{32} выбранных открытых текстов для выбранных ключей или с помощью 2^{48} известных открытых текстов для выбранных ключей [158]. Это вскрытие не зависит от числа этапов алгоритма. (В той же работе Бихам вскрывает LOKI89, используя кри птоанализ со связанными ключами, с помощью 2^{17} выбранных открытых текстов для выбранных ключей или с помощью 2^{33} известных открытых текстов для выбранных ключей.) Несложно повысить устойчивость LOKI91 к вскрытию такого типа, усложнив схему использования ключа.

Патенты и лицензии

LOKI не запатентован. Кто угодно может реализовать алгоритм и использовать его. Исходный код, прив еденный в этой книге, написан в Университете Нового Южного Уэльса. При желании использовать эту реализ ацию (или другие реализации, которые на несколько порядков быстрее) в коммерческом продукте обращайтесь к Директору CITRAD, Факультет компьютерных наук, Университетский колледж, Университет Нового Южного Уэльса, Академия австралийских вооруженных сил, Канберра, Австралия (Director CITRAD, Department of Computer Science, University College, UNSW, Australian Defense Force Academy, Canberra ACT 2600, Australia; FAX: +61 6 268 8581.

13.7 KHUFU и KHAFRE

В 1990 году Ральф Меркл (Ralph Merkle) предложил два алгоритма. В основе их проектирования лежали следующие принципы [1071]:

1. 56-битовый размер ключа DES слишком мал. Так как стоимость увеличения размера ключа прене брежимо мала (компьютерная память недорога и доступна), он должен быть увеличен.
2. Интенсивное использование перестановок в DES хотя и удобно для аппаратных реализаций, чрезв ычайно затрудняет программные реализации. Наиболее быстрые реализации DES выполняют перест ановки табличным образом. Просмотр таблицы может обеспечить те же характеристики "рассеяния", что и собственно перестано вки, и может сделать реализацию намного более гибкой.
3. S-блоки DES, всего с 64 4-битовыми элементами, слишком малы. Теперь с увеличением памяти должны увеличиться и S-блоки. Более того, все восемь S-блоков используются одновременно. Хотя это и удобно для аппаратуры, для программной реализации это кажется ненужным ограничением. Должны быть реализованы больший размер S-блоков и последовательное (а не параллельное) их и спользование.
4. Широко признано, что начальная и заключительная перестановки криптографически бессмысленны, поэтому они должны быть устранены.

5. Все быстрые реализации DES заранее рассчитывают ключи для каждого этапа. При данном условии нет смысла усложнять эти вычисления.

6. В отличие от DES критерии проектирования S-блоков должны быть общедоступны.

К этому перечню Меркл, возможно, теперь добавил бы "устойчивость к дифференциальному и линейному криптоанализу", ведь в то время эти способы вскрытия не были известны.

Khufu

Khufu - это 64-битовый блочный шифр. 64-битовый открытый тест сначала разбивается на две 32-битовые половины, *L* и *R*. Над обеими половинами и определенными частями ключа выполняется операция XOR. Затем, аналогично DES, результаты проходят через некоторую последовательность этапов. На каждом этапе младший значащий байт *L* используется в качестве входных данных S-блока. У каждого S-блока 8 входных битов и 32 выходных бита. Далее выбранный в S-блоке 32-битовый элемент подвергается операции XOR с *R*. Затем *L* циклически сдвигается не несколько из восьми битов, *L* и *R* меняются местами, и этап заканчивается. Сам S-блок не является статическим, но меняется каждые восемь этапов. Наконец после последнего этапа над *L* и *R* выполняется операция XOR с другими частями ключа, и половины объединяются, образуя блок шифротекста.

Хотя части ключа используются для XOR с блоком шифрования в начале и в конце алгоритма, главная цель ключа -генерация S-блоков. Эти S-блоки - секретны, по сути являются они являются частью ключа. Полный размер ключа Khufu равен 512 битам (64 байтам), алгоритм предоставляет способ генерации S-блоков по кл ючу. Количество этапов алгоритма остается открытым. Меркл упомянул, что 8-этапный Khufu чувствителен к вскрытию с выбранным открытым текстом и рекомендует 16, 24 или 32 этапа [1071]. (Он ограничивает выбор количества этапов числами, кратными восьми.)

Так как в Khufu используются зависимые от ключа и секретные S-блоки, он устойчив к дифференциальному криптоанализу. Существует дифференциальное вскрытие 16-этапного Khufu, которое раскрывает ключ после 2^{31} выбранных открытых текстов [611], но его не удалось расширить на большее количество этапов. Если лучшим способом вскрыть Khufu является грубая сила, то его надежность производит сильное впечатление. 512-битовый ключ обеспечивает сложность 2^{512} - огромное число при любых условиях.

Khafre

Khafre - это вторая из криптосистем, предложенных Мерклом [1071]. (Khufu (Хуфу) и Khafre (Хафр) - это имена египетских фараонов.) По конструкции этот алгоритм похож на Khufu, но он спроектирован для прил ожений, не использующих предварительных вычислений. S-блоки не зависят от ключа. Вместо этого Khafre и с пользует фиксированные S-блоки. Блок шифрования подвергается операции XOR с ключом не только перед первым этапом и после последнего, но и после каждых 8 этапов шифрования.

Меркл предположил, что с Khafre должны использоваться 64- или 128-битовые ключи, и что для Khafre п отребуется больше этапов, чем для Khufu. Это наряду с тем, что каждый этап Khafre сложнее этапа Khufu, делает Khafre более медленным. Зато для Khafre не нужны никакие предварительны расчеты, что позволяет быстрее шифровать небольшие порции данных.

В 1990 году Бихам и Шамир применили свой метод дифференциального анализа против Khafre [170]. Им удалось взломать 16-этапный Khafre с помощью вскрытия с выбранным открытым текстом после 1500 разли чных шифрований. На их персональном компьютере это заняло около часа. Преобразование этого вскрытия во вскрытие с известным открытым текстом потребует около 238 шифрований. Khafre с 24 этапами может быть вскрыт с помощью вскрытия с выбранным открытым текстом за 253 шифрования, а с помощью вскрытия с и звестным открытым текстом - за 259 шифрования.

Патенты

И Khufu, и Khafre запатентованы [1072]. Исходный код этих алгоритмов содержится в патенте. При желании получить лицензию на любой или оба алгоритма следует обратиться к директору по лицензированию корпор ации Xerox (Director of Licensing, Xerox Corporation, P.0. Box 1600, Stamford, CT, 06904-1600).

13.8 RC2

RC2 представляет собой алгоритм с переменной длиной ключа, спроектированный Роном Ривестом (Ron Rivest) для RSA Data Security, Inc. (RSADSI). Очевидно "RC" - это сокращенное "Ron's Code" ("Код Рона"), хотя официально это "Rivest Cipher" ("Шифр Ривеста"). (RC3 был взломан в RSADSI в процессе разработки, RC1 не вышел за пределы записной книжки Ривеста.) Он представляет собой частную собственность, и его детали не были опубликованы. Не думайте ни минуты, что это увеличивает его безопасность. RC2 уже появился в ко ммерческих продуктах. Насколько мне известно, RC2 не был запатентован и защищен только как торговый се крет.

RC2 - это шифр с 64-битовым блоком и переменной длиной ключа, предназначенный заменить DES. В соо тветствии с утверждениями компании программные реализации RC2 в три раза быстрее DES. Алгоритм может использовать ключ переменной длины, от 0 байтов до максимальной длины строки, поддерживаемой компь ютерной системой, скорость шифрования не зависит от размера ключа. Этот ключ предварительно используется для заполнения 128-байтовой таблицы, зависящей от ключа. Поэтому множество действительно различных ключей составляет 21024. RC2 не использует S-блоков [805], используются две операции - "смешивание" и "перемешивание" ("mix" и "mash"), для каждого этапа выбирается одна из них. В соответствии с литературой [1334]:

> . . . RC2 не является итеративным блочным шифром. Это предполагает, что RC2 более устойчив к дифференциальному и линейному криптоанализу, чем другие блочные шифры, безопасность которых опирается на копирование схемы DES.

Отказ RSADSI опубликовать RC2 заставляет сомневаться в намерениях этой компании. Она обещает пр едоставить детали алгоритма всем, кто подпишет соглашение о нераспространении информации, и утверждает, что позволит криптоаналитикам опубликовать любые обнаруженные негативные результаты. Мне неизвестно ни об одном криптоаналитике, не работающем в этой компании, кто бы исследовал алгоритм, так как это по сути означало бы выполнить работу по анализу для компании.

Тем не менее, Рон Ривест - не шарлатан. Он уважаемый и компетентный криптограф. Я лично в значител ьной степени верю в этот алгоритм, хотя я лично и не видел кода. RC4, также являющийся интеллектуальной собственностью RSADSI, был опубликован в Internet (см. раздел 17.1), и, вероятно, опубликование RC2 являе тся только вопросом времени.

По соглашению между Ассоциацией издателей программного обеспечения (Software Publishers Association, SPA) и правительством США RC2 и RC4 (см. раздел 17.1) получили специальный экспортный статус (см. ра здел 25.14). Процесс получения разрешения на экспорт продуктов, реализующих один из этих двух алгоритмов, значительно упрощен при условии, что длина ключа не превышает 40 битов.

Достаточен ли 40-битовый ключ? Существует всего один триллион возможных ключей. При условии, что наиболее эффективным методом криптоанализа является вскрытие грубой силой (большое допущение, ведь а лгоритм никогда не был опубликован), и что микросхема грубого вскрытия может проверить миллион ключей в секунду, поиск правильного ключа займет 12.7 дней. Тысяча машин, работающих параллельно, смогут ра скрыть ключ за двадцать минут.

RSA Data Security, Inc., утверждает, что, хотя шифрование и дешифрирования выполняются для быстро, и счерпывающего поиска потребуется намного больше времени. Заметное количество времени тратится на форм ирование плана использования ключа. Хотя это время пренебрежимо мало при шифровании и дешифрировании сообщений, это не так при проверке каждого возможного ключа.

Правительство США никогда не позволило бы экспортировать любой алгоритм, который оно, по крайней мере в теории, не смогло бы вскрыть. Оно может создать магнитную ленту или CD с конкретным блоком о ткрытого текста, зашифрованным каждым возможным ключом. Для вскрытия сообщения остается только вст авить ленту и сравнить блоки шифротекста в сообщении с блоками шифротекста на ленте. При совпадении мо жно проверить возможный ключ и посмотреть, имеет ли сообщение какой-нибудь смысл. Если они выберут часто встречающийся блок (все нули, ASCII-символы пробела, и т.д.), этот метод будет работать. Объем данных, нужный для хранения результатов шифрования 64-битового блока открытого текста всеми 10^{12} возможными ключами, составляет 8 терабайтов - вполне реально. По поводу лицензирования RC2 обращайтесь в RSADSI (см. раздел 25.4).

13.9 IDEA

Первый вариант шифра IDEA, предложенный Ксуеджа Лай (Xuejia Lai) и Джеймсом Масси (James Massey), появился в 1990 году [929]. Он назывался PES (Proposed Encryption Standard, предложенный стандарт шифр ования). В следующем году, после демонстрации Бихамом и Шамиром возможностей дифференциального кри птоанализа, авторы усилили свой шифр против такого вскрытия и назвали новый алгоритм IPES (Improved Proposed Encryption Standard, улучшенный предложенный стандарт шифрования) [931, 924]. В 1992 году назв ание IPES было изменено на IDEA (International Data Encryption Algorithm, международный алгоритм шифров ания данных) [925].

IDEA основывается на некоторых впечатляющих теоретических положениях и, хотя криптоанализ добился некоторых успехов в отношении вариантов с уменьшенным количеством этапов, алгоритм все еще кажется сильным. По моему мнению это самый лучший и самый безопасный блочный алгоритм, опубликованный сег одня.

Будущее IDEA пока неясно. Попыток заменить им DES предпринято не было, частично потому, что он зап атентован и должен быть лицензирован для коммерческих приложений, и частично потому, что люди пока все еще ждут, наблюдая насколько хорошо поведет себя алгоритм в предстоящие годы криптоанализа. Его сег о-

дняшняя известность объясняется тем, что он является частью PGP (см. раздел 24.12).

Обзор IDEA

IDEA является блочным шифром, он работает с 64-битовыми блоками открытого текста. Длина ключа - 128 битов. Для шифрования и дешифрирования используется один и тот же алгоритм.

Как и другие, уже рассмотренные блочные шифры IDEA использует и запутывание, и рассеяние. Флософия, лежащая в основе проекта, представляет собой "объединение операций из различных алгебраических групп". Смешиваются три алгебраические группы, и все они могут быть легко реализованы как аппаратно, так и пр ограммно:

— XOR

— Сложение по модулю 2^{16}

— Умножение по модулю $2^{16} + 1$. (Это операцию можно рассматривать как S-блок IDEA.)

Все эти операции (а в алгоритме используются только они, перестановки на битовом уровне не применяю тся) работают с 16-битовыми подблоками. Этот алгоритм даже эффективнее на 16-битовых процессорах.

Описание IDEA

Схема IDEA представлена на Рис. 13-9. 64-битовый блок данных делится на четыре 16-битовых подблока: X_1, X_2, X_3 и X_4. Эти четыре подблока становятся входными данными для первого этапа алгоритма. Всего в алг оритме восемь этапов. На каждом этапе четыре подблока подвергаются операциям XOR, сложениям и умнож ениям друг с другом и с шестью 16-битовыми подключами. Между этапами обмениваются местами второй и третий подблоки. Наконец четыре подблока объединяются с четырьмя подключами в окончательном преобраз овании. На каждом этапе события происходят в следующей последовательности:

(1) Перемножаются X_1 и первый подключ.

(2) Складываются X_2 и второй подключ.

(3) Складываются X_3 и третий подключ.

(4) Перемножаются X_4 и четвертый подключ.

(5) Выполняется XOR над результатами этапов (1) и (3).

(6) Выполняется XOR над результатами этапов (2) и (4).

(7) Перемножаются результаты этапа (5) и пятый подключ.

(8) Складываются результаты этапов (6) и (7).

(9) Перемножаются результаты этапа (8) и шестой подключ.

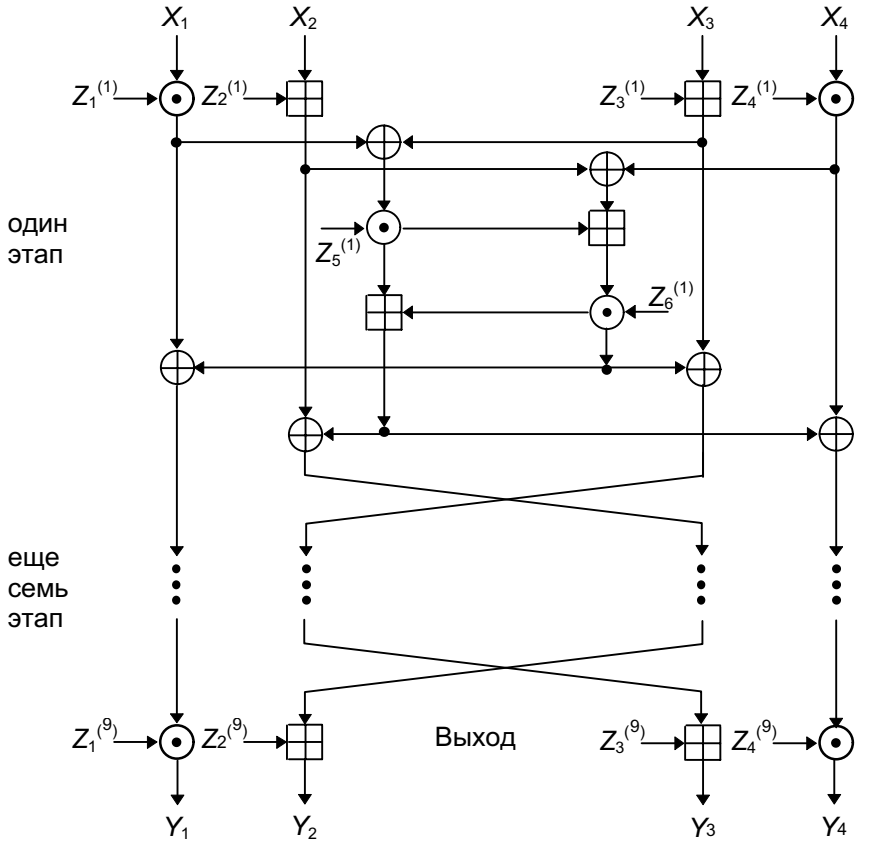
(10) Складываются результаты этапов (7) и (9).

(11) Выполняется XOR над результатами этапов (1) и (9).

(12) Выполняется XOR над результатами этапов (3) и (9).

(13) Выполняется XOR над результатами этапов (1) и (10).

(14) Выполняется XOR над результатами этапов (4) и (10).



X_i : 16-битовый подблок открытого текста
Y_i : 16-битовый подблок шифротекста
$Z_i^{(r)}$: 16-битовый подблок ключа
$\oplus$: побитовое "исключающее или" (XOR) 16-битовых подблоков
$\boxplus$: сложение по модулю 2^{16} 16-битовых целых
$\odot$: умножение по модулю $2^{16}+1$ 16-битовых целых при условии, что нулевой подблок соответствует 2^{16}

Рис. 13-9. IDEA.

Выходом этапа являются четыре подблока - результаты действий (11), (12), (13) и (14). Поменяйте местами два внутренних подблока (но не в последнем этапе), и вы получите исходные данные для следующего этапа.

После восьмого этапа выполняется заключительное преобразование:

(1) Перемножаются X_1 и первый подключ.

(2) Складываются X_2 и второй подключ.

(3) Складываются X_3 и третий подключ.

(4) Перемножаются X_4 и четвертый подключ.

Наконец четыре подблока снова соединяются, образуя шифротекст.

Также несложно создавать подключи. Алгоритм использует 52 из них (шесть для каждого из восьми этапов и еще четыре для заключительного преобразования). Сначала 128-битовый ключ делится на восемь 16-битовых подключей. Это первые восемь подключей алгоритма (шесть для первого этапа и два - для второго). Затем ключ циклически сдвигается налево на 25 битов и снова делится на восемь подключей. Первые четыре используются на этапе 2, а оставшиеся четыре - на этапе 3. Ключ циклически сдвигается налево на 25 битов для получения следующих восьми подключей, и так до конца алгоритма.

Дешифрирование выполняется точно также за исключением того, что подключи инвертируются и слегка и з-меняются. Подключи при дешифрировании представляют собой обратные значения ключей шифрования по отношению к операциям либо сложения, либо умножения. (Для IDEA подблоки, состоящие из одних нулей, считаются равными $2^{16} = -1$ для умножения по модулю $2^{16} + 1$, следовательно, обратным значением 0 относ и-тельно умножения является 0.) Эти вычисления могут занять некоторое время, но их нужно выполнить один раз для каждого ключа дешифрирования. В Табл. 13-4 представлены подключи шифрования и соответствующие

подключи дешифрирования.

Табл. 13-4.
Подключи шифрования и дешифрирования IDEA

Этап	Подключи шифрования						Подключи дешифрирования					
1	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_4^{(1)}$	$Z_5^{(1)}$	$Z_6^{(1)}$	$Z_1^{(9)-1}$	$-Z_2^{(9)}$	$-Z_3^{(9)}$	$Z_4^{(9)-1}$	$Z_5^{(8)}$	$Z_6^{(8)}$
2	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$	$Z_4^{(2)}$	$Z_5^{(2)}$	$Z_6^{(2)}$	$Z_1^{(8)-1}$	$-Z_2^{(8)}$	$-Z_3^{(8)}$	$Z_4^{(8)-1}$	$Z_5^{(7)}$	$Z_6^{(7)}$
3	$Z_1^{(3)}$	$Z_2^{(3)}$	$Z_3^{(3)}$	$Z_4^{(3)}$	$Z_5^{(3)}$	$Z_6^{(3)}$	$Z_1^{(7)-1}$	$-Z_2^{(7)}$	$-Z_3^{(7)}$	$Z_4^{(7)-1}$	$Z_5^{(6)}$	$Z_6^{(6)}$
4	$Z_1^{(4)}$	$Z_2^{(4)}$	$Z_3^{(4)}$	$Z_4^{(4)}$	$Z_5^{(4)}$	$Z_6^{(4)}$	$Z_1^{(6)-1}$	$-Z_2^{(6)}$	$-Z_3^{(6)}$	$Z_4^{(6)-1}$	$Z_5^{(5)}$	$Z_6^{(5)}$
5	$Z_1^{(5)}$	$Z_2^{(5)}$	$Z_3^{(5)}$	$Z_4^{(5)}$	$Z_5^{(5)}$	$Z_6^{(5)}$	$Z_1^{(5)-1}$	$-Z_2^{(5)}$	$-Z_3^{(5)}$	$Z_4^{(5)-1}$	$Z_5^{(4)}$	$Z_6^{(4)}$
6	$Z_1^{(6)}$	$Z_2^{(6)}$	$Z_3^{(6)}$	$Z_4^{(6)}$	$Z_5^{(6)}$	$Z_6^{(6)}$	$Z_1^{(4)-1}$	$-Z_2^{(4)}$	$-Z_3^{(4)}$	$Z_4^{(4)-1}$	$Z_5^{(3)}$	$Z_6^{(3)}$
7	$Z_1^{(7)}$	$Z_2^{(7)}$	$Z_3^{(7)}$	$Z_4^{(7)}$	$Z_5^{(7)}$	$Z_6^{(7)}$	$Z_1^{(3)-1}$	$-Z_2^{(3)}$	$-Z_3^{(3)}$	$Z_4^{(3)-1}$	$Z_5^{(2)}$	$Z_6^{(2)}$
8	$Z_1^{(8)}$	$Z_2^{(8)}$	$Z_3^{(8)}$	$Z_4^{(8)}$	$Z_5^{(8)}$	$Z_6^{(8)}$	$Z_1^{(2)-1}$	$-Z_2^{(2)}$	$-Z_3^{(2)}$	$Z_4^{(2)-1}$	$Z_5^{(1)}$	$Z_6^{(1)}$
заключительное преобразование	$Z_1^{(9)}$	$Z_2^{(9)}$	$Z_3^{(9)}$	$Z_4^{(9)}$			$Z_1^{(1)-1}$	$-Z_2^{(1)}$	$-Z_3^{(1)}$	$Z_4^{(1)-1}$		

Скорость IDEA

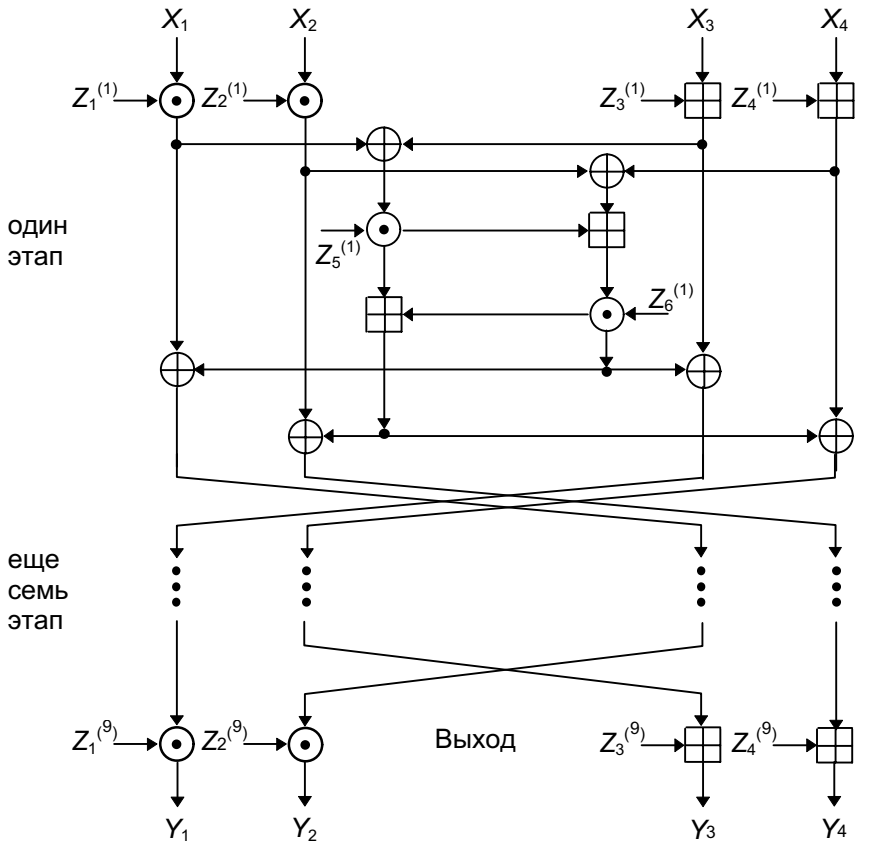
Современные программные реализации IDEA примерно в два раза быстрее, чем DES. На компьютере с i386/33 МГц IDEA шифрует данные со скоростью 880 Кбит/с, а на компьютере с i486/33 МГц - со скоростью 2400 Кбит/с. Вы могли подумать, что IDEA должен был быть побыстрее, но умножения - недешевое удовольс т-вие. Умножение двух 32-битовых чисел на процессоре i486 занимает 40 тактов (10 на процессоре Pentium).

Реализация PES на базе СБИС шифрует данные со скоростью 55 Мбит/с при тактовой частоте 25 МГц [208,398]. Другая СБИС, разработанная ETH Zurich и состоящая из of 251000 транзисторов на кристалле пл о-щадью 107.8 мм^2, шифрует данные с помощью алгоритма IDEA со скоростью 177 Мбит/с при тактовой частоте 25 МГц [926, 207, 397].

Криптоанализ IDEA

Длина ключа IDEA равна 128 битам - более чем в два раза длиннее ключа DES. При условии, что наиболее эффективным является вскрытие грубой силой, для вскрытия ключа потребуется 2^{128} (10^{38}) шифрований. Созд айте микросхему, которая может проверять миллиард ключей в секунду, объедините миллиард таких микр о-схем, и вам потребуется 10^{13} лет для решения проблемы - это больше, чем возраст вселенной. 10^{24} таких микро-схем могут найти ключ за день, но во вселенной не найдется столько атомов кремния, чтобы построить такую машину. Наконец мы чего-то достигли, хотя в некоторых темных вопросах я лучше останусь сторонним набл ю-дателем.

Может быть вскрытие грубой силой - не лучший способ вскрытия IDEA. Алгоритм все еще слишком нов, чтобы можно было говорить о каких-то конкретных криптографических результатах. Разработчики сделали все возможное, чтобы сделать алгоритм устойчивым к дифференциальному криптоанализу. Они определили пон я-тие марковского шифра и продемонстрировали, что устойчивость к дифференциальному криптоанализу может быть промоделирована и оценена количественно [931, 925]. (Для сравнения с алгоритмом IDEA, устойчивость которого к дифференциальному криптоанализу была усилена, и который показан на Рис. 13-9, на Рис. 13-10 приведен первоначальный алгоритм PES. Удивительно, как такие незначительные изменения могут привести к столь большим различиям.) В [925] Лай (Lai) утверждал (он привел подтверждение, но не доказательство), что IDEA устойчив к дифференциальному криптоанализ уже после 4 из 8 этапов. Согласно Бихаму, его попытка вскрыть IDEA с помощью криптоанализа со связанными кл ючами также не увенчалась успехом [160].



X_i : 16-битовый подблок открытого текста
Y_i : 16-битовый подблок шифротекста
$Z_i^{(r)}$: 16-битовый подблок ключа
$\oplus$: побитовое "исключающее или" (XOR) 16-битовых подблоков
$\boxplus$: сложение по модулю 2^{16} 16-битовых целых
$\odot$: умножение по модулю $2^{16}+1$ 16-битовых целых при условии, что нулевой подблок соответствует 2^{16}

Рис. 13-10. PES.

Вилли Майер (Willi Meier) исследовал три алгебраических операции IDEA и показал, что, хотя они нес о-вместимы, есть случаи, когда эти операции можно упростить так, чтобы в некоторой степени облегчить [1050]. Его вскрытие 2-этапного IDEA оказалось эффективнее вскрытия грубой силой (2^{42} операций), но для IDEA с 3 и более этапами эффективность этого вскрытия была ниже вскрытия грубой силой. Безопасность полного 8-этапного IDEA осталась непоколебимой.

Джоан Дэймен (Joan Daemen) открыла класс слабых ключей IDEA [405, 409]. Эти ключи не являются сл а-быми в том смысле, в котором слабы некоторые ключи DES, для которых функция шифрования обратна самой себе. Слабость этих ключей состоит в том, что взломщик может легко определить их с помощью вскрытия с выбранным открытым текстом. Например, слабым является следующий ключ (в шестнадцатиричной записи):

0000,0000,0*x*00,0000,0000,000*x*,*xxxx*,*x*000

В позиции "*x*" может стоять любая цифра. При использовании такого ключа побитовое XOR определенных пар открытых текстов равно побитовому XOR получившихся пар шифротекстов.

В любом случае вероятность случайной генерации одного из таких слабых ключей очень мала: $1/2^{96}$. Опас-ность случайно выбрать такой ключ практически не существует. К тому же, несложно модифицировать IDEA так, чтобы исключить наличие слабых ключей - достаточно выполнить XOR каждого подключа с числом 0x0dae [409].

Хотя попыток выполнить криптоанализ IDEA было много, мне неизвестно ни об одной успешной.

Режимы работы и варианты IDEA

IDEA может работать в любом из режимов работы блочного шифра, описанных в главе 9. Против двойных реализаций IDEA может быть предпринято то же вскрытие "встреча посередине", что и против DES (см. раздел

15.1). Однако, так как ключ IDEA более чем в два раза длиннее ключа DES, это вскрытие непрактично. Объем нужной для такого вскрытия памяти составит $64*2^{128}$ битов, или 10^{39} байтов. Может быть во вселенной и достаточно материи, чтобы построить такое хранилище, но я в этом сомневаюсь.

Если вы учитываете возможность использования параллельной вселенной, используйте утроенную реализацию IDEA (см. раздел 15.2):

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P)))$$

Такая реализация устойчива против вскрытия "встреча посередине".

Кроме того, почему бы вам не реализовать IDEA независимыми подключами, особенно если ваши средства распределения ключей позволяют работать с длинными ключами. Для IDEA нужно всего 52 16-битовых ключа, общей длиной 832 битов. Этот вариант определенно безопасней, но никто не сможет сказать насколько.

В наивной модификации может быть увеличен вдвое размер блока. Алгоритм также прекрасно работал бы с 32-битовыми подблоками вместо 16-битовых и с 256-битовым ключом. Шифрование выполнялось бы быстрее, и безопасность возросла бы в 2^{32} раза. Или нет? Теория, на которой основан алгоритм, опирается на то, что $2^{16}+1$ является простым числом. А $2^{32} + 1$ простым числом не является. Может быть алгоритм и можно изменить так, чтобы он работал, но его безопасность будет совсем иной. Лай говорит, что заставить работать такой алгоритм будет нелегко [926].

Хотя IDEA кажется намного безопаснее DES, не всегда можно легко заменить один алгоритм другим в существующем приложении. Если ваша база данных и шаблоны сообщений могут работать с 64-битовым ключом, реализация 128-битового ключа IDEA может быть возможной.

Для таких приложений создайте 128-битовый ключ, объединив 64-битовый ключ сам с собой. Не забывайте, что эта модификация заметно ослабляет IDEA.

Если вас больше волнует скорость работы, а не безопасность, попробуйте вариант IDEA с меньшим числом этапов. Сегодня лучшее вскрытие IDEA быстрее вскрытия грубой силой только для 2.5 и менее этапов [1050], 4-этапный IDEA будет в два раза быстрее и, насколько мне известно, его безопасность не уменьшится.

Caveat Emptor[1]

IDEA - это относительно новый алгоритм, многие вопросы пока остаются открытыми. Образует ли IDEA группу? (Лай думает, что нет [926].) Не существует ли пока не открытых способов вскрытия этого шифра? У IDEA твердая теоретическая основа, но снова и снова казавшиеся безопасными алгоритмы капитулируют перед новыми формами криптоанализа. Ряд групп академических и военных исследователей не опубликовали свои результаты криптоанализа IDEA. Возможно, кто-нибудь уже добился или когда-нибудь добьется успеха.

Патенты и лицензии

IDEA запатентован в Европе и Соединенных Штатах [1012, 1013]. Патент принадлежит Ascom-Tech AG. Для некоммерческого использования лицензирование не нужно. При заинтересованности в лицензии для коммерческого применения алгоритма следует обратиться по адресу Ascom Systec AG, Dept CMVV, Cewerbepark, CH-5506, Mgenwil, Switzerland; +41 64 56 59 83; Fax: +41 64 56 59 90; idea@ascom.ch.

13.10 MMB

Недовольство использованием в IDEA 64-битового блока шифрования привело к созданию Джоном Дэймоном алгоритма под названием MMB (Modular Multiplication-based Block cipher, модульный блочный шифр, использующий умножения) [385, 405, 406]. В основе MMB лежит теория, используемая и в IDEA: перемешивающие операции из различных групп. MMB - это итеративный алгоритм, главным образом состоящий из линейных действий (XOR и использование ключа) и параллельное использование четырех больших нелинейных и зменяющих обычный порядок подстановок. Эти подстановки определяются с помощью умножения по модулю 2^{32}-1 с постоянными множителями. Результатом применения этих действий является алгоритм, использующий и 128-битовый ключ и 128-битовый блок.

MMB оперирует 32-битовыми подблоками текста (x_0, x_1, x_2, x_3) и 32-битовыми подблоками ключа (k_0, k_1, k_2, k_3). Это делает удобным реализацию алгоритма на современных 32-битовых процессорах. Чередуясь с XOR, шесть раз используется нелинейная функция f. Вот этот алгоритм (все операции с индексами выполняются по модулю 3):

$x_i = x_i \oplus k_i$, для $i = 0$ до 3

---

[1] Предупреждение покупателю

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+1}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+2}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_i$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+1}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+2}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

У функции f три этапа:

(1) $x_1 = c_i * x_i$, для $i = 0$ до 3 (Если на входе умножения одни единицы, то на выходе - тоже одни единицы.)

(2) Если младший значащий бит $x_0 = 1$, то $x_0 = x_0 \oplus C$. Если младший значащий бит $x_3 = 0$, то $x_3 = x_3 \oplus C$.

(3) $x_i = x_{i-1} \oplus x_i \oplus x_{i+1}$, для $i = 0$ до 3

Все операции с индексами выполняются по модулю 3. Операция умножения на этапе (1) выполняется по м одулю 2^{32}-1. В данном алгоритме если второй операнд - это 2^{32}-1, то результат также равен 2^{32}-1. В алгоритме используются следующие константы:

C = 2aaaaaaa

c_0 = 025f1cdb

$c_1 = 2 * c_0$

$c_2 = 2^3 * c_0$

$c_3 = 2^7 * c_0$

Константа C - это "простейшая" константа с высоким троичным весом, нулевым младшим значащим битом и без круговой симметрии. У константы c0 несколько иные характеристики. Константы c_1, c_2 и c_3 являются смещенными версиями c_0, и используются для предотвращения вскрытий основанных на симметрии. Подробности можно найти в [405].

Дешифрирование является обратным процессом. Этапы (2) и (3) заменяются на свою инверсию. На этапе (1) вместо c_i^{-1} используется c_i. c_i^{-1} = 0dad4694.

Безопасность MMB

Схема MMB обеспечивает на каждом этапе значительное и независимое от ключа рассеяние. В IDEA ра ссеяние до определенной степени зависит от конкретных подключей. В отличие от IDEA у MMB нет слабых ключей.

К сожалению MMB - это умерший алгоритм [402]. Это утверждение справедливо по многим причинам, хотя криптоанализ MMB и не был опубликован. Во первых, он проектировался без учета требований устойчивости к линейному криптоанализу. Выбор мультипликативных множителей обеспечил устойчивость к дифференциал ьному криптоанализу, но о линейном криптоанализе авторам алгоритма было еще неизвестно.

Во вторых, Эли Бихам реализовал эффективное вскрытие с выбранным ключом [160], использующеее тот факт, что все этапы идентичны, а ключ при использовании просто циклически сдвигается на 32 бита. В третьих, несмотря на то, что программные реализации MMB были бы очень эффективны, в аппаратном исполнении а лгоритм менее эффективен, чем DES.

Дэймон предлагает, что тот, кто захочет улучшить MMB, должен сначала проанализировать умножение по модулю с помощью линейного криптоанализа и подобрать новый множитель, а затем сделать константу C ра зличной для каждого этапа [402]. Затем, улучшив использование ключа, добавляя к ключам этапов константы с целью устранения смещения. Но сам не стал заниматься этим и разработал 3-Way (см. раздел 14.5).

13.11 CA-1.1

CA - это блочный шифр, основанный на клеточных автоматах и разработанный Говардом Гутовицом (Howard Gutowitz) [677, 678, 679]. Он шифрует 384-битовые блоки открытого текста 1088-битовым ключом (на самом деле используется два ключа - 1024-битовый и 64- битовый). Из-за природы клеточных автоматов алг оритм наиболее эффективен при реализации в больших параллельных интегрированных схемах.

CA-1.1 использует как обратимые, так и необратимые правила клеточного автомата. При обратимом прав иле каждое состояние структуры получается из единственного предшествующего состояния, а при необратимом правиле у каждого состояния может быть несколько предшественников. При шифровании необратимые правила пошагово обращаются во времени. Для продвижения обратно от текущего состояния случайным образом дол жно выбираться одно из состояний-предшественников. Этот процесс многократно повторяется. Таким образом, обратная итерация служит для смешивания случайной информации с инфорамацией сообщения. CA-1.1 испол ьзует особый сорт частично линейного необратимого правила, такого, что для любого данного состояния может быть быстро построено случайное состояние-предшественник. На некоторых стадиях шифрования используются и обратимые правила.

Обратимые правила (простые параллельные перестановки подблоков состояния) нелинейны. Необратимые правила полностью определяются ключом, а обратимые зависят как от ключа, так и от случайной информации, вставленной в ходе шифрования необратимыми правилами.

CA-1.1 основан на структуре блочных связей. То есть, обработка блока сообщения частично отделена от о бработки потока случайной информации, вставленной при шифровании. Эта случайная информация служит для связи друг с другом стадий шифрования. Она также может быть использована для связи с потоком шифроте кста. Информация связи генерируется как часть шифрования.

Так как CA-1.1 представляет собой новый алгоритм, слишком рано делать какие-либо заявления о его без опасности. Гутовиц упоминает некоторые возможные вскрытия, включая дифференциальный криптоанализ, но ему не удалось вскрыть алгоритм. В качестве стимула Гутовиц предложил награду в 1000 долларов для "первого человека, который разработает доступную процедуру вскрытия CA-1.1."

CA-l.1 запатентован [678], но доступен для некоммерческого использования. При необходимости получить лицензию на алгоритм или объявленную награду за криптоанализ обращайтесь к Говарду Гутовицу по адресу Howard Cutowitz, ESPCI, Laboratorie d'Electronique, 10 rue Vauquelin, 75005 Paris, France.

13.12 SKIPJACK

Skipjack разработан NSA в качестве алгоритма шифрования для микросхем Clipper и Capstone (см. разделы 24.16 и 24.17). Так как этот алгоритм объявлен секретным, его подробности никогда не публиковались. Он б удет реализован только как защищенная от взлома аппаратура.

Этот алгоритм объявлен секретным не потому, что это повышает его надежность, а потому что NSA не х очет, чтобы Skipjack использовался без механизма условного вручения ключей Clipper. Агентство не хочет, чт обы программные реализации алгоритма распространились по всему миру.

Безопасен ли Skipjack? Если NSA захочет создать безопасный алгоритм, оно, скорее всего, это сделает. С другой стороны, если NSA захочет создать алгоритм с лазейкой, то оно сможет сделать и это. Вот что было опубликовано [1154, 462].

— Это итеративный блочный шифр.

— Размер блока - 64 бита.

— Алгоритм использует 80-битовый ключ.

— Он может быть использован в режимах ECB, CBC, 64-битовый OFB, либо 1-, 8-, 16-, 32- или 64-битовый CFB.

— Операция шифрования или дешифрирования состоит из 32 этапов.

— NSA начало работу над ним в 1985 и завершило проверку в 1990.

В документации на микросхему Mykotronx Clipper утверждается, что задержка в выдаче результата, прис ущая алгоритму Skipjack, составляет 64 такта. Это означает, что на каждый этап приходится два такта: один предположительно для подстановки с помощью S-блока, а другой - для заключительного XOR в конце каждого этапа. (Не забывайте, перестановки при аппаратных реализациях не занимают времени.) В документации Mykotronx эта двухтактная операция называется "G-блоком", а все вместе - "сдвигом". (Часть G-блока носит название "F-таблицы" и является таблицей констант, а может быть таблицей функций.)

По одним слухам Skipjack использует 16 S-блоков, а по другим для хранения S-блоков нужно всего 128 байт

памяти. Непохоже, чтобы оба этих слуха были правдой.

Еще один слух утверждает, что этапы Skipjack, в отличие от DES, работают не с половиной блока. Это вместе с замечанием о "сдвигах" и случайном заявлении на Crypto '94 о том, что в Skipjack применяется "48-битовая внутренняя структура", позволяет сделать вывод, что алгоритм по своей схеме похож на SHA (см. раздел 18.7), но использует четыре 16-битовых подблока. Три подблока, обработанные зависящей от ключа однонаправленной функцией, дают 16 битов, которые подвергаются операции XOR с оставшимся подблоком. Затем весь блок циклически сдвигается на 16 битов и поступает на вход следующего этапа, или сдвига. При этом также используются 128 байтов данных S-блока. Я подозреваю, что S-блоки зависят от ключа.

По своей структуре Skipjack вероятно похож на DES. NSA понимает, что его защищенная от взлома аппаратура в конце концов будет вскрыта и исследована, они не будут рисковать никакими передовыми криптографическими методами.

То, что NSA планирует использовать алгоритм Skipjack для шифрования своей Системы защиты сообщений (Defense Messaging System, DMS), свидетельствует о безопасности алгоритма. Чтобы убедить скептиков, NIST разрешил комиссии "уважаемых неправительственных экспертов . . . получить доступ к конфиденциальным подробностям алгоритма, чтобы они исследовали его возможности и опубликовали результаты своих исследований " [812].

В предварительном отчете этой комиссии экспертов [262] (окончательного отчета не было, и возможно никогда не будет) сообщалось:

> Принимая во внимание, что стоимость вычислительных мощностей уменьшается в два раза каждые 18 месяцев, сложность вскрытия Skipjack сравняется с сегодняшней сложностью вскрытия DES только через 36 лет. Следовательно, риск, что Skipjack будет взломан в ближайшие 30-40 лет, незначителен.
>
> Незначителен и риск взлома Skipjack с помощью более быстрых способов вскрытия, включая дифференциальный криптоанализ. У алгоритма не слабых ключей, отсутствует и свойство комплиментарности. Эксперты в отсутствие времени для самостоятельного большого исследования алгоритма изучили представленное NSA описание разработки и проверки алгоритма
>
> Устойчивость Skipjack к криптоанализу не зависит от хранения в тайне самого алгоритма.

Итак, участники дискуссии не смогли поработать с алгоритмом достаточно долго, чтобы прийти к каким-нибудь выводам самостоятельно. Все, что они смогли сделать - это взглянуть на результаты, показанные им NSA.

Остался без ответа вопрос, является ли плоским пространство ключей Skipjack (см. раздел 8.2). Даже если у Skipjack нет ключей, слабых в смысле DES, ряд особенностей процесса использования ключа может сделать одни ключи сильнее других. У Skipjack может быть 2^{70} сильных ключей, гораздо больше чем у DES, вероятность случайно выбрать один из этих сильных ключей будет приблизительно 1 к 1000. Лично я думаю, что пространство ключей Skipjack - плоское, но то, что об этом никто не заявил публично, вызывает тревогу.

Skipjack запатентован, но в соответствии с соглашением о секретности патента [1122] этот патент хранится в тайне. Патент будет опубликован тогда и только тогда, когда алгоритм Skipjack будет успешно восстановлен кем-то посторонним. Это дает возможность правительству воспользоваться и преимуществом защиты патентом, и преимуществом конфеденциальности торгового секрета.

Глава 14
И еще о блочных шифрах

14.1 ГОСТ

ГОСТ - это блочный алгоритм, разработанный в бывшем Советском Союзе [655, 1393]. Название "ГОСТ" является сокращением от "Государственный стандарт", нечто похожее на FIPS за исключением того, что это название могут носить стандарты практически любого типа. (Полным названием является "Государственный стандарт Союза ССР", или "Государственный стандарт Союза Советских Социалистических Республик".) Н омер данного стандарта - 28147-89. Все эти стандарты утверждаются Государственным комитетом по стандартам Союза ССР.

Я не знаю, использовался ли ГОСТ 28147-89 для засекреченного трафика или только для гражданского шифрования. Замечание в начале стандарта гласит, что алгоритм "удовлетворяет всем криптографическим тр е-бованиям, а степень защищаемой информации не ограничивается". Я слышал утверждения, что этот алгоритм первоначально использовался только для очень важных линий связи, включая секретные военные коммуник а-ции, но у меня нет подтверждений.

Описание ГОСТ

ГОСТ является 64-битовым алгоритмом с 256-битовым ключом. ГОСТ также использует дополнительный ключ, который рассматривается ниже. В процессе работы алгоритма на 32 этапах последовательно выполняется простой алгоритм шифрования.

Для шифрования текст сначала разбивается на левую половину L и правую половину R. На этапе i использу-ется подключ K_i. На этапе i алгоритма ГОСТ выполняется следующее:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i)$

Этап ГОСТ показан на Рис. 14-1. Функция f проста. Сначала правая половина и i-ый подключ складываются по модулю 2^{32}. Результат разбивается на восемь 4-битовых кусочков, каждый из которых поступает на вход св о-его S-блока. ГОСТ использует восемь различных S-блоков, первые 4 бита попадают в первый S-блок, вторые 4 4 бита - во второй S-блок, и т.д. Каждый S-блок представляет собой перестановку чисел от 0 до 15. Например, S-блок может выглядеть как:

7, 10, 2, 4, 15, 9, 0, 3, 6, 12, 5, 13, 1, 8, 11

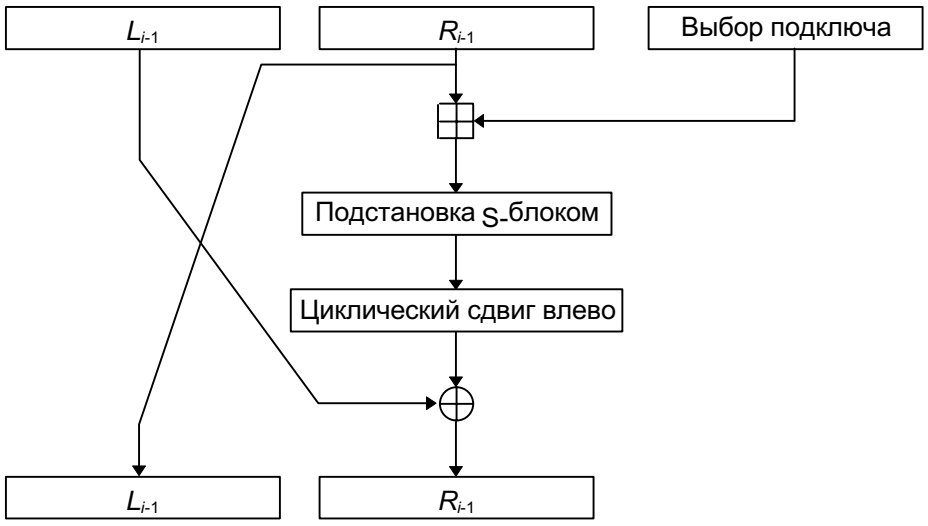


Рис. 14-1. Этап ГОСТ.

В этом случае, если на входе S-блока 0, то на выходе 7. Если на входе 1, на выходе 10, и т.д. Все восемь S-блоков различны, они фактически являются дополнительным ключевым материалом. S-блоки должны хр а-ниться в секрете.

Выходы всех восьми S-блоков объединяются в 32-битовое слово, затем все слово циклически сдвигается влево на 11 битов. Наконец результат объединяется с помощью XOR с левой половиной, и получается новая правая половина, а правая половина становится новой левой половиной. Выполните это 32 раза, и все в поря д-

ке.

Генерация подключей проста. 256-битовый ключ разбивается на восемь 32-битовых блоков: k_1, k_2, . . .k_8. На каждом этапе используется свой подключ, как показано в Табл. 14-1. Дешифрирование выполняется также, как и шифрование, но инвертируется порядок подключей k_i.

Стандарт ГОСТ не определяет способ генерации S-блоков, говорится только, что блоки должны быть пр е-доставлены каким-то образом [655]. Это породило домыслы о том, что советский производитель может поста в-лять хорошие S-блоки "хорошим" организациям и плохие S-блоки тем организациям, которых производитель собирается надуть. Это вполне может быть так, но неофициальные переговоры с российским производителем микросхем ГОСТ выявили другую альтернативу. Производитель создает перестановки S-блока самостоятельно с помощью генератора случайных чисел.

Табл. 14-1.
Использование подключей на различных этапах ГОСТ

Этап:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Подключ:	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Этап:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Подключ:	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1

Совсем недавно стал известным набор S-блоков, используемых в приложениях Центрального Банка РФ. Эти S-блоки также используются в однонаправленной хэш-функции ГОСТ (см. раздел 18.11) [657]. Они перечисл е-ны в Табл. 14-2.

Криптоанализ ГОСТ

Вот главные различия между DES и COST.

— DES использует сложную процедуру для генерации подключей из ключей. В ГОСТ эта процедура очень проста.

— В DES 56-битовый ключ, а в ГОСТ - 256-битовый. Если добавить секретные перестановки S-блоков, то полный объем секретной информации ГОСТ составит примерно 610 битов.

— У S-блоков DES 6-битовые входы и 4-битовые выходы, а у S-блоков ГОСТ 4-битовые входы и выходы. В обоих алгоритмах используется по восемь S-блоков, но размер S-блока ГОСТ равен одной четвертой размера S-блока DES.

— В DES используются нерегулярные перестановки, названные P-блоком, а в ГОСТ используется 11-битовый циклический сдвиг влево.

— В DES 16 этапов, а в ГОСТ - 32.

Табл. 14-2.
S-блоки ГОСТ

S-блок 1:															
4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
S-блок 2:															
14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
S-блок 3:															
5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
S-блок 4:															
7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
S-блок 5:															
6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
S-блок 6:															

4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
S-блок 7:															
13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
S-блок 8:															
1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Если лучшим способом вскрытия ГОСТ является грубая сила, то это очень безопасный алгоритм. ГОСТ и спользует 256-битовый ключ, а если учитывать секретные S-блоки, то длина ключа возрастает. ГОСТ, по вид имому, более устойчив к дифференциальному и линейному криптоанализу, чем DES. Хотя случайные S-блоки ГОСТ возможно слабее фиксированных S-блоков DES, их секретность увеличивает устойчивость ГОСТ к ди ференциальному и линейному криптоанализу. К тому же, эти способы вскрытия чувствительны к количеству этапов - чем больше этапов, тем труднее вскрытие. ГОСТ использует в два раза больше этапов, чем DES, одно это возможно делает несостоятельными и дифференциальный, и линейный криптоанализ.

Другие части ГОСТ такие же, как в DES, или слабее. ГОСТ не использует существующую в DES перестановку с расширением. Удаление этой перестановки из DES ослабляет его из-за уменьшения лавинного эффекта, разумно считать, что отсутствие такой операции в ГОСТ ослабляет этот алгоритм. Сложение, используемое в ГОСТ, не менее безопасно, чем используемая в DES операция XOR.

Самым большим различием представляется использование в ГОСТ циклического сдвига вместо перестано вки. Перестановка DES увеличивает лавинный эффект. В ГОСТ изменение одного входного бита влияет на один S-блок одного этапа, который затем влияет на два S-блока следующего этапа, три блока следующего этапа, и т.д.. В ГОСТ потребуется 8 этапов прежде, чем изменение одного входного бита повлияет на каждый бит р езультата, алгоритму DES для этого нужно только 5 этапов. Это, конечно же, слабое место. Но не забывайте: ГОСТ состоит из 32 этапов, а DES только из 16.

Разработчики ГОСТ пытались достигнуть равновесия между безопасностью и эффективностью. Они изм енили идеологию DES так, чтобы создать алгоритм, который больше подходит для программной реализации. Они, по видимому, менее уверены в безопасности своего алгоритма и попытались скомпенсировать это очень большой длиной ключа, сохранением в секрете S-блоков и удвоением количества итераций. Вопрос, увенчались ли их усилия созданием более безопасного, чем DES, алгоритма, остается открытым.

14.2 CAST

CAST был разработан в Канаде Карлайслом Адамсом (Carlisle Adams) и Стаффордом Таваресом (Stafford Tavares) [10, 7]. Они утверждают, что название обусловлено ходом разработки и должно напоминать о вероя тностном характере процесса, а не об инициалах авторов. Описываемый алгоритм CAST использует 64-битовый блок и 64-битовый ключ.

CAST имеет знакомую структуру. Алгоритм использует шесть S-блоков с 8-битовым входом и 32-битовым выходом. Работа этих S-блоков сложна и зависит от реализации, подробности можно найти в литературе.

Для шифрования сначала блок открытого текста разбивается на левую и правую половины. Алгоритм сост оит из 8 этапов. На каждом этапе правая половина объединяется с частью ключа с помощью функции f, а затем XOR результата и левой половины выполняется для получения новой правой половины. Первоначальная (до этапа) правая половина становится новой левой половиной. После 8 этапов (не переставьте левую и правую п оловины после восьмого этапа) две половины объединяются, образуя шифротекст. Функция f проста:

(1) Разбейте 32-битовый вход на четыре 8-битовых части: a, b, c, d.

(2) Разбейте 16-битовый подключ на две 8-битовых половины: e, f.

(3) Подайте a на вход S-блока 1, b - на вход S-блока 2, c - на вход S-блока 3, d - на вход S-блока 4, e - на вход S-блока 5 и f - на вход S-блока 6.

(4) Выполните XOR шести выходов S-блоков, получая 32-битовый результат.

Иначе, 32-битовый вход может быть объединен с помощью XOR с 32 битами ключа, разбит на четыре 8-битовых части, которые обрабатываются S-блоками и затем объединяются с помощью XOR [7]. Безопасность N этапов, организованных таким образом, по видимому, соответствует $N + 2$ этапам другого варианта.

16-битовые подключи этапов легко получаются из 64-битового ключа. Если k_1, k_2, . . .k_8 - это 8 байтов ключа, то на этапах алгоритма используются следующие подключи:

Этап 1: k_1, k_2

Этап 2: k_3, k_4

Этап 3: k_5, k_6

Этап 4: k_7, k_8

Этап 5: k_4, k_3

Этап 6: k_2, k_1

Этап 7: k_8, k_7

Этап 8: k_6, k_5

Сила этого алгоритма заключена в его S-блоках. У CAST нет фиксированных S-блоков, для каждого прил ожения они конструируются заново. Критерии проектирования описаны в [10], изогнутыми функциями являются столбцы S-блоков, обеспечивающие необходимые свойства S-блоков (см. раздел 14.10). Созданный для данной реализации CAST S-блоков уже больше никогда не меняется. S-блоки зависят от реализации, а не от ключа.

В [10] было показано, что CAST устойчив к дифференциальному криптоанализу, а в [728] - что CAST усто йчив и к линейному криптоанализу. Неизвестно иного, чем грубая сила, способа вскрыть CAST.

Northern Telecom использует CAST в своем пакете программ Entrust для компьютеров Macintosh, PC и р абочих станций UNIX. Выбранные ими S-блоки не опубликованы. Канадское правительство считает CAST н овым стандартом шифрования. Патентная заявка на CAST находится в процессе рассмотрения.

14.3 BLOWFISH

Blowfish - это алгоритм, разработанный лично мной для реализации на больших микропроцессорах [1388, 1389]. Алгоритм незапатентован, и его код на языке C приведен в конце этой книги для широкого пользования. При проектировании Blowfish я использовал следующие критерии:

1. Скорость. Blowfish шифрует данные на 32-битовых микропроцессорах со скоростью 26 тактов на байт.
2. Компактность. Blowfish может работать менее, чем в 5 Кбайт памяти.
3. Простота. Blowfish использует только простые операции: сложение, XOR и выборка из таблицы по 32-битовому операнду. Анализ его схемы несложен, что делает при реализации алгоритма уменьшает к оличество ошибок [1391].
4. Настраиваемая безопасность. Длина ключа Blowfish переменна и может достигать 448 битов.

Blowfish оптимизирован для тех приложений, в которых нет частой смены ключей, таких как линии связи или программа автоматического шифрования файлов. При реализации на 32-битовых микропроцессорах с большим кэшем данных, таких как Pentium и PowerPC, Blowfish заметно быстрее DES. Blowfish не подходит для использования в приложениях с частой сменой ключей, например, при коммутации пакетов, или для и спользования в качестве однонаправленной хэш-функции. Большие требования к памяти делают невозможным использование этого алгоритма в интеллектуальных платах.

Описание Blowfish

Blowfish представляет собой 64-битовый блочный шифр с ключом переменной длины. Алгоритм состоит из двух частей: развертывание ключа и шифрование данных. Развертывание ключа преобразует ключ длиной до 448 битов в несколько массивов подключей, общим объемом 4168 байтов.

Шифрование данных состоит из простой функции, последовательно выполняемой 16 раз. Каждый этап с остоит из зависимой от ключа перестановки и зависимой от ключа и данных подстановки. Используются только сложения и XOR 32-битовых слов. Единственными дополнительными операциями на каждом этапе являются четыре извлечения данных из индексированного массива.

В Blowfish используется много подключей. Эти подключи должны быть рассчитаны до начала шифрования или дешифрирования данных.

P-массив состоит из 18 32-битовых подключей:

P1, P2, . . ., P18

Каждый из четырех 32-битовых S-блоков содержит 256 элементов:

$S_{1,0}$, $S_{1,1}$, . . ., $S_{1,255}$

$S_{2,0}$, $S_{2,2}$, . . ., $S_{2,255}$

$S_{3,0}$, $S_{3,3}$, . . ., $S_{3,255}$

$S_{4,0}$, $S_{4,4}$, . . ., $S_{4,255}$

Точный метод, используемый при вычислении этих подключей описан в этом разделе ниже.

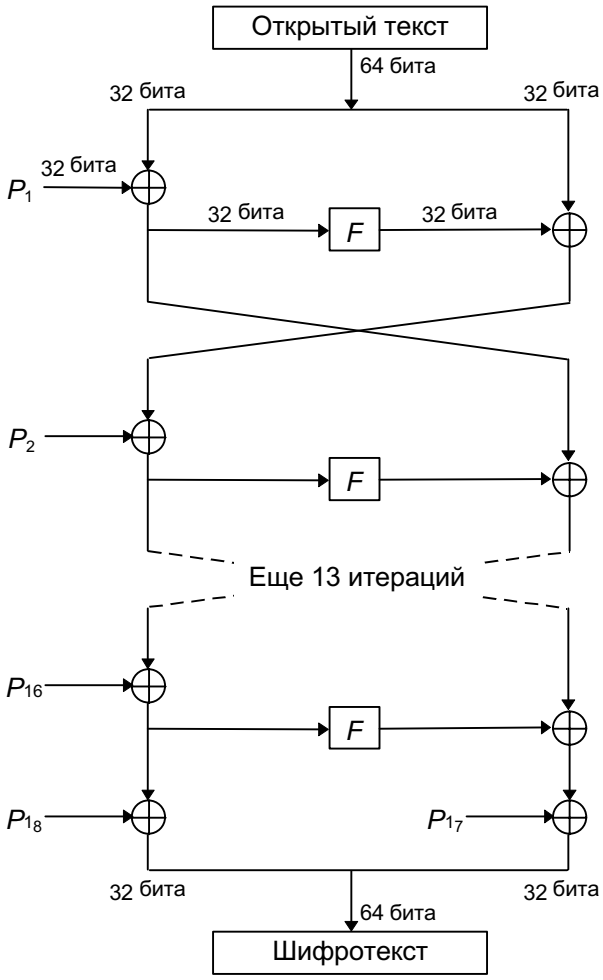


Рис. 14-2. Blowfish.

Blowfish является сетью Фейстела (Feistel) (см. раздел 14.10), состоящей из 16 этапов. На вход подается 64-битовый элемент данных x. Для шифрования:

Разбейте x на две 32-битовых половины: x_L, x_R

Для i = 1 по 16:

$x_L = x_L \oplus P_{18}$

$x_R = \mathrm{F}(x_L) \oplus x_R$

Переставить x_L и x_R (кроме последнего этапа.)

$x_R = x_R \oplus \mathrm{P}_{17}$

$x_L = x_L \oplus P_{18}$

Объединить x_L и x_R

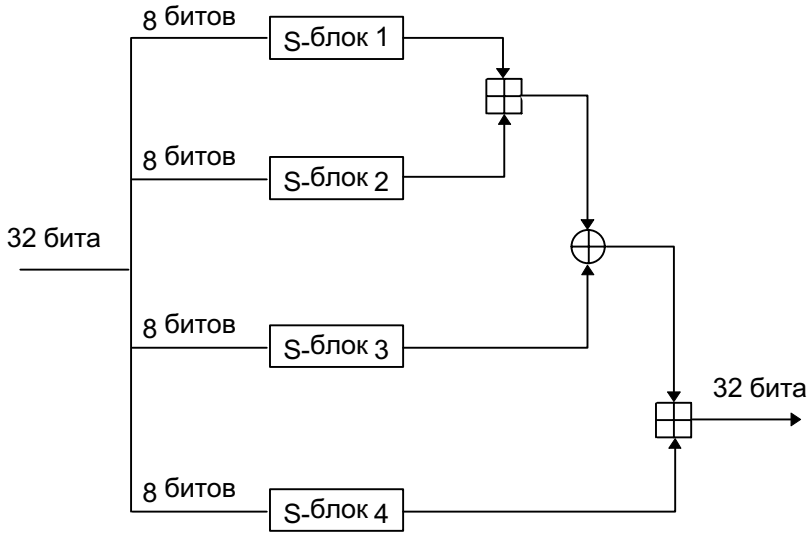


Рис. 14-3. Функция F.

Функция F представляет собой следующее (см. Рис. 14-3):

Разделить x_L на четыре 8-битовых части: a, b, c и d
$F(x_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$

Дешифрирование выполняется точно также, как и шифрование, но P_1, P_2, . . ., P_{18} используются в обратном порядке.

В реализациях Blowfish, для которых требуется очень большая скорость, цикл должен быть развернут, а все ключи должны храниться в кэше. Подробности приведены в [568].

Подключи рассчитываются с помощью специального алгоритма. Вот какова точная последовательность де йствий.

(1) Сначала P-массив, а затем четыре S-блока по порядку инициализируются фиксированной строкой. Эта строка состоит из шестнадцатиричных цифр π.

(2) Выполняется XOR P_1 с первыми 32 битами ключа, XOR P_2 со вторыми 32 битами ключа, и так далее для всех битов ключа (до P_{18}). Используется циклически, пока для всего P-массива не будет выполнена опер ация XOR с битами ключа.

(3) Используя подключи, полученные на этапах (1) и (2), алгоритмом Blowfish шифруется строка из одних нулей.

(4) P_1 и P_2 заменяются результатом этапа (3).

(5) Результат этапа (3) шифруется с помощью алгоритма Blowfish и измененных подключей.

(6) P_3 и P_4 заменяются результатом этапа (5).

(7) Далее в ходе процесса все элементы P-массива и затем по порядку все четыре S-блока заменяются вых одом постоянно меняющегося алгоритма Blowfish.

Всего для генерации всех необходимых подключей требуется 521 итерация. Приложения могут сохранять подключи - нет необходимости выполнять процесс их получения многократно.

Безопасность Blowfish

Серж Воденэ (Serge Vaudenay) исследовал Blowfish с известными S-блоками и r этапами, дифференциал ьный криптоанализ может раскрыть P-массив с помощью 2^{8r+1} выбранных открытых текстов [1568]. Для некот орых слабых ключей, которые генерируют плохие S-блоки (вероятность выбора такого ключа составляет 1 к 2^{14}), это же вскрытие раскрывает P-массив с помощью всего 2^{4r+1}. При неизвестных S-блоках это вскрытие может обнаружить использование слабого ключа, но не может определит сам ключ (ни S-блоки, ни P-массив). Это вскрытие эффективно только против вариантов с уменьшенным числом этапов и совершенно бесполезно против 16-этапного Blowfish.

Конечно, важно и раскрытие слабых ключей, даже хотя они скорее всего не будут использоваться. Слабым является ключ, для которого два элемента данного S-блока идентичны. До выполнения развертывания ключа невозможно определить, является ли он слабым. Если вы беспокоитесь об этом, вам придется выполнить ра з-

вертывание ключа и проверить, нет ли в S-одинаковых элементов. Хотя я не думаю, что это так уж необходимо.

Мне неизвестно об успешном криптоанализе Blowfish. Для безопасности не реализуйте Blowfish с умен ьшенным числом этапов.

Kent Marsh Ltd. встроила Blowfish в свой продукт обеспечения безопасности FolderBolt, предназначенный для Microsoft Windows и Macintosh. Алгоритм также входит в Nautilus и PGPfone.

14.4 SAFER

SAFER K-64 означает Secure And Fast Encryption Routine with a Key of 64 bits - Безопасная и быстрая проц едура шифрования с 64-битовым ключом [1009]. Этот не являющийся частной собственностью алгоритм, разр аботанный Джеймсом Массеем (James Massey) для Cylink Corp., используется в некоторых из продуктов этой компании. Правительство Сингапура собирается использовать этот алгоритм - с 128-битовым ключом [1010] - для широкого спектра приложений. Его использование не ограничено патентом, авторскими правами или др угими ограничениями.

Алгоритм работает с 64-битовым блоком и 64-битовым ключом. В отличие от DES он является не сетью Фейстела (см. раздел 14.10), а итеративным блочным шифром: для некоторого количества этапов применяется одна и та же функция. На каждом этапе используются два 64-битовых подключа. Алгоритм оперирует только байтами.

Описание SAFER K-64

Блок открытого текста делится на восемь байтовых подблоков: $B_1, B_2, \ldots, B_7, B_8$. Затем подблоки обрабатываются в ходе r этапов. Наконец подблоки подвергаются заключительному преобразованию. На каждом этапе используется два подключа: K_{2r-1} и K_{2r}.

На Рис. 14-4 показан один этап SAFER K-64. Сначала над подблоками выполняется либо операция XOR, л ибо сложени с байтами подключа K_{2r-1}. Затем восемь подблоков подвергаются одному из двух нелинейных пр еобразований:

$y = 45^x \bmod 257$. (Если $x = 0$, то $y = 0$.)

$y = \log_{45} x$. (Если $x = 0$, то $y = 0$.)

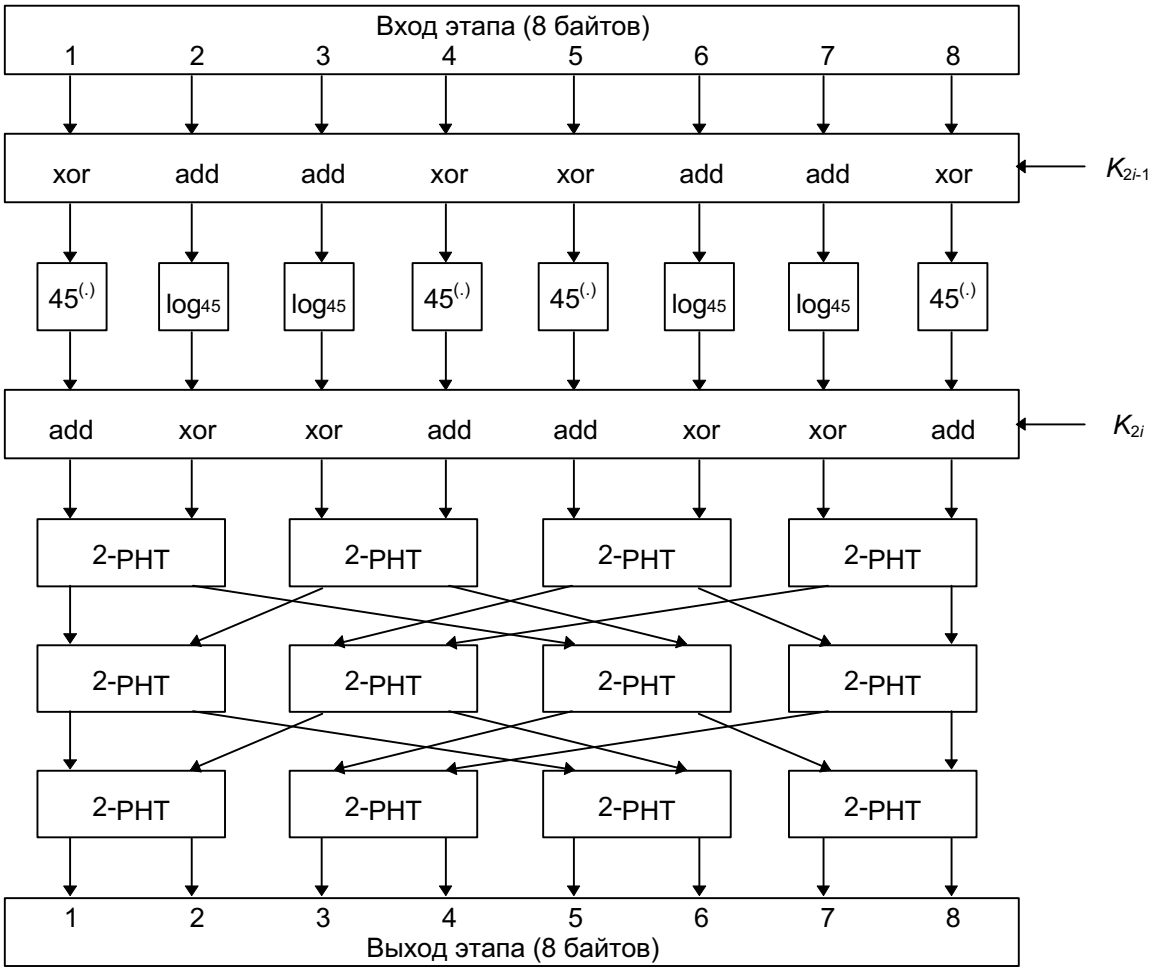


Рис. 14-4. Один этап SAFER.

Это операции в конечном поле GF(257), а 45 - элемент поля, являющийся примитивом. В реализациях SAFER K-64 быстрее выполнять поиск в таблице, чем все время рассчитывать новые результаты.

Затем подблоки либо подвергаются XOR, либо складываются с байтами подключа K_{2r}. Результат этого действия проходит через три уровня линейных операций, целью которых является увеличение лавинного эффекта. Каждая операция называется псевдоадамаровым преобразованием (Pseudo-Hadamard Transform, PHT). Если на входе PHT a_1 и a_2, то на выходе:

$b_1 = (2a_1 + a_2) \bmod 256$

$b_2 = (a_1 + a_2) \bmod 256$

После r этапов выполняется заключительное преобразование. Оно совпадает с преобразованием, являющимся первым действием каждого этапа. Над B_1, B_4, B_5 и B_8 выполняется XOR с соответствующими байтами последнего подключа, а B_2, B_3, B_6 и B_7 складываются с соответствующими байтами последнего подключа. В результате и получается шифротекст.

Дешифрирование представляет собой обратный процесс: сначала заключительное преобразование (с вычитанием вместо сложения), затем r инвертированных этапов. Обратное PHT (Inverse PHT, IPHT) - это:

$a_1 = (b_1 - b_2) \bmod 256$

$a_2 = (-b_1 + 2b_2) \bmod 256$

Массей рекомендует использовать 6 этапов, но для большей безопасности количество этапов можно увеличить.

Генерировать подключи совсем не трудно. Первый подключ, K_1, - это просто ключ пользователя. Последующие ключи генерируются в соответствии со следующей процедурой:

$K_{i+1} = (K_i <<< 3i) + c_i$

Символ "<<<" обозначает циклический сдвиг налево. Сдвиг выполняется побайтно, а c_i является константой этапа. Если c_{ij} - это j-ый байт константы i-го этапа, то можно рассчитать все константы этапов по следующей

формуле

$c_{ij} = 45^{45^{\wedge}((9i+j) \bmod 256) \bmod 257} \bmod 257$

Обычно эти значения хранятся в таблице.

SAFER K-128

Этот альтернативный способ использования ключа был разработан Министерством внутренних дел Синг апура, а затем был встроен Массеем в SAFER [1010]. В этом способе используются два ключа, K_a и K_b, по 64 бита каждый. Прием состоит в том, чтобы генерировать параллельно две последовательности подключей, а з атем чередовать подключи из каждой последовательности. Это означает, что при выборе $K_a = K_b$ 128-битовый ключ совместим с 64-битовым ключом K_a.

Безопасность SAFER K-64

Массей показал, что SAFER K-64 после 6 этапов абсолютно защищен от дифференциального криптоанализа после 8 этапов и достаточно безопасен. Уже после 3 этапов против этого алгоритма становится неэффективным и линейный криптоанализ [1010].

Кнудсен (Knudsen) обнаружил слабое место в распределении ключей: практически для каждого ключа сущ ествует не меньше одного (а иногда даже девять) другого ключа, который при шифровании какого-то другого открытого текста превращает его в тот же шифротекст [862]. Число различных открытых текстов, которые шифруются одинаковыми шифротекстами, находится в промежутке от 2^{22} до 2^{28}. Хотя такое вскрытие не может повлиять на безопасность SAFER как алгоритма шифрования, оно значительно уменьшает его надежность при использовании в качестве однонаправленной хэш-функции. В любом случае Кнудсен рекомендует использовать не меньше 8 этапов.

SAFER был спроектирован для Cylink, а Cylink были предъявлены различные обвинения со стороны NSA [80]. Я рекомендовал бы потратить несколько лет на интенсивный криптоанализ прежде, чем как-либо испол ьзовать SAFER.

14.5 3-WAY

3-Way - это блочный шифр, разработанный Джоном Дэйменом (Joan Daemen) [402, 410]. Он использует блок и ключ длиной 96 бит, и его схема предполагает очень эффективную аппаратную реализацию.

3-Way является не сетью Фейстела, а итеративным блочным шифром. У 3-Way может быть n этапов, Дэ ймен рекомендует 11.

Описание S-Way

Этот алгоритм описать несложно. Для шифрования блока открытого текста x:

For $i = 0$ to $n - 1$

$x = x$ XOR K_i

x = theta (x)

x = pi - 1 (x)

x = gamma (x)

x = pi - 2 (x)

$x = x \oplus K^{n+1}$

x = theta (x)

При этом используются следующие функции:

— theta(x) - функция линейной подстановки, в основном набор циклических сдвигов и XOR.

— pi - 1 (x) и pi - 2 (x) - простые перестановки.

— gamma (x) - функция нелинейной подстановки. Именно это действие и дало имя всему алгоритму, оно представляет собой параллельное выполнение подстановки 3-битовых данных.

Дешифрирование аналогично шифрованию за исключением того, что нужно изменить на обратный порядок битов исходных данных и результата. Исходный код, реализующий 3-Way, можно найти в конце этой книги.

Пока об успешном криптоанализе 3-Way неизвестно. Алгоритм незапатентован.

14.6 CRAB

Этот алгоритм был разработан Бертом Калиски [Burt Kaliski] и Мэттом Робшоу [Matt Robshaw] из RSA Laboratories [810]. В основе Crab лежит идея использовать методы однонаправленных хэш-функций для созд ания быстрого алгоритма шифрования. Следовательно, Crab очень похож на MD5, и в этом разделе предполаг ается, что вы знакомы с материалом раздела 18.5.

У Crab очень большой блок: 1024 байта. Так как Crab был представлен скорее как материал для исследов ания, а не реальный алгоритм, конкретной процедуры генерации ключей не было предложено. Авторы рассмо трели метод, который позволяет превратить 80-битовый ключ в три вспомогательных подключа, хотя алгоритм позволяет легко использовать ключи переменной длины. В Crab используется два набора больших подключей:

Перестановка чисел с 0 до 255: P_0, P_1, P_2, ..., P_{255}.

Массив из 2048 32-битовых чисел: S_0, S_1, S_2,..., S_{2047}.

Все эти подключи должны быть вычислены до шифрования или дешифрирования. Для шифрования 1024-байтового блока X:

(1) Разделите X на 256 32-битовых подблоков: X_0, X_1, X_2, ..., X_{255}.

(2) Переставьте подблоки X в соответствии с P.

(3) For r=0 to 3

For g = 0 to 63

$A = X_{(4g) <<< 2r}$

$B = X_{(4g+1) <<< 2r}$

$C = X_{(4g+2) <<< 2r}$

$D = X_{(4g+3) <<< 2r}$

For step s = 0 to 7

$A = A \oplus (B + f_r(B, C, D) + S_{512r+8g+s})$

$TEMP = D$

$D = C$

$C = B$

$B = A <<< 5$

$A = TEMP$

$X_{(4g) <<< 2r} = A$

$X_{(4g+1) <<< 2r} = B$

$X_{(4g+2) <<< 2r} = C$

$X_{(4g+3) <<< 2r} = D$

(4) Снова объединить X_0, X_1, X_2, ..., X_{255}, получая шифротекст.

Функции $f_r(B, C, D)$ аналогичны используемым в MD5:

$f_0(B, C, D) = (B \wedge C) \vee ((\neg B) \wedge D)$

$f_1(B, C, D) = (B \wedge D) \vee (C \wedge (\neg D))$

$f_2(B, C, D) = B \oplus C \oplus D$

$f_3(B, C, D) = C \oplus (B \vee (\neg D))$

Дешифрирование представляет собой обратный процесс. Генерирование подключей является непростой з адачей. Вот как по 80-битовому ключу K может быть сгенерирован массив перестановок P.

(1) Проинициализируйте K_0, K_1, K_2, ..., K_9 10 байтами K.

(2) For i=10 to 255

$K_i = K_{i-2} \oplus K_{i-6} \oplus K_{i-7} \oplus K_{i-10}$

(3) For i=10 to 255, $P_i = i$

(4) m=0

(5) For j=0 to 1

For i = 256 to 1 step -1

$m = (K_{256-i} + K_{257-i}) \bmod i$

$K_{257-i} = K_{257-i} <<< 3$

Переставить P_i и P_{i-1}

S-массив из 2048 32-битовых слов может быть сгенерирован аналогичным образом либо по тому же 80-битовому ключу, либо по другому ключу. Авторы предупреждают, что эти детали должны "рассматриваться только в качестве мотивации, могут быть и другие эффективные схемы, обеспечивающие лучшую безопасность" [810].

Crab был предложен как испытательный стенд для новых идей, а не как рабочий алгоритм. Во многом он использует те же приемы, что и MD5. Бихам заметил, что очень большой блок упрощает криптоанализ алгоритма [160]. С другой стороны Crab может позволять эффективно использовать очень большой ключ. В этом случае "упрощение криптоанализа" может ничего не значить.

14.7 SXAL8/MBAL

Это 64-битовый блочный алгоритм из Японии [769]. SXAL8 - это основной алгоритм, а MBAL представляет собой расширенную версию с переменной длиной блока. Так как внутри MBAL выполняется ряд умных действий, авторы утверждают, что они могут обеспечить достаточную безопасность за малое число этапов. При длине блока 1024 байта MBAL примерно в 70 раз быстрее, чем DES. К несчастью в [1174] показано, что MBAL чувствителен к дифференциальному криптоанализу, а в [865] - что он чувствителен и к линейному криптоанализу.

14.8 RC5

RC5 представляет собой блочный фильтр с большим числом параметров: размером блока, размером ключа и количеством этапов. Он был изобретен Роном Ривестом и проанализирован в RSA Laboratories [1324, 1325].

Используется три действия: XOR, сложение и циклические сдвиги. На большинстве процессоров операции циклического сдвига выполняются за постоянное время, переменные циклические сдвиги являются нелинейной функцией. Эти циклические сдвиги, зависящие и от ключа, и от данных, представляют собой интересную операцию.

RC5 использует блок переменной длины, но в приводимом примере мы остановимся на 64-битовом блоке данных. Шифрование использует $2r+2$ зависящих от ключа 32-битовых слов - S_0, S_1, S_2, ... S_{2r+1} - где r - число этапов. Эти слова мы сгенерируем позднее. Для шифрования сначала разделим блок открытого текста на два 32-битовых слова: A и B. (RC5 предполагает следующее соглашение по упаковке байтов в слова: первый байт занимает младшие биты регистра A, и т.д.) Затем:

$A = A + S_0$

$B = B + S_1$

For i = 1 to r:

$A = ((A \oplus B) <<< B) + S_{2i}$

$B = ((B \oplus A) <<< A) + S_{2i+1}$

Результат находится в регистрах A и B.

Дешифрирование также просто. Разбейте блок открытого текста на два слова, A и B, а затем:

For $i = r$ down to 1:

$B = ((B - S_{2i+1}) >>> A) \oplus A$

$A = ((A - S_{2i}) >>> B) \oplus B$

$B = B - S_1$

$A = A - S_0$

Символ ">>>" обозначает циклический сдвиг направо. Конечно же, все сложения и вычитания выполняются по модулю 2^{32}.

Создание массива ключей более сложно, но также прямолинейно. Сначала, байты ключа копируются в ма с-сив L из с 32-битовых слов, дополняя при необходимости заключительное слово нулями. Затем массив S инициализируется при помощи линейного конгруэнтного генератора по модулю 2^{32}:

$S_0 = P$

for $i = 1$ to $2(r + 1) - 1$:

$S_i = (S_{i-1} + Q) \bmod 2^{32}$

P = 0xb7e15163 и Q = 0x9e3779b9, эти константы основываются на двоичном представлении e и phi.

Наконец, подставляем L в S:

$i = j = 0$

$A = B = 0$

выполнить n раз (где n - максимум $2(r + 1)$ и c):

$A = S_i = (S_i + A + B) <<< 3$

$B = L_i = (L_i + A + B) <<< (A + B)$

$i = (i + 1) \bmod 2(r + 1)$

$j = (j + 1) \bmod c$

По сути, RC5 представляет собой семейство алгоритмов. Только что мы определили RC5 с 32-битовым сл о-вом и 64-битовым блоком, не существуе причин, запрещающих использовать тот же алгоритм с 64-битовым словом и 128-битовым. Для $w = 64$, P и Q равны 0xb7e151628aed2a6b и 0x9e3779b97f4a7c15, соответственно. Ривест обозначил различные реализации RC5 как RC5-w/r/b, где w - это размер слова, r - число этапов, а b - длина ключа в байтах.

RC5 является новым алгоритмом, но RSA Laboratories потрптила достаточно много времени, анализируя его работу с 64-битовым блоком. После 5 этапов статистика выглядит очень хорошо. После 8 этапов каждый бит открытого текста влияет по крайней мере на один циклический сдвиг. Дифференциальное вскрытие требует 2^{24} выбранных открытых текстов для 5 этапов, 2^{45} для 10 этапов, 2^{53} для 12 этапов и 2^{68} для 15 этапов. Конечно же, существует только 2^{64} возможных открытых текстов, поэтому такое вскрытие неприменимо против алгоритма с 15 и более этапами. Оценка для линейного криптоанализа показывает, что алгоритм безопасен после 6 этапов. Ривест рекомендует использовать не меньше 12 этапов, а лучше 16 [1325]. Это число может меняться.

RSADSI в настоящее время патентует RC5, а это названия заявлено, как торговая марка. Компания утве р-ждает, что плата за лицензирование будет очень мала, но это лучше проверить.

14.9 Другие блочные алгоритмы

Существует алгоритм, называемый в литературе CRYPTO-MECCANO [301], но он не является безопасным. Четыре японских криптографа на Eurocrypt '91 представили алгоритм, основанный на хаотичных отображениях [687, 688], Бихам выполнил криптоанализ этого алгоритма на той же конференции [157]. Другой алгоритм оп ирается на подмножества некоторого множества случайных кодов [693]. Существует множество алгоритмов, о сн-ованных на теории кодов, исправляющих ошибки: вариант алгоритма МакЭлайса (McEliece) (см. раздел 19.7) [786, 1290], алгоритм Rao-Nam [1292, 733, 1504, 1291, 1056, 1057, 1058, 1293], варианты алгоритма Rao-Nam [464, 749, 1503] и алгоритм Li-Wang [964, 1561] - все они небезопасны. CALC также небезопасен [1109]. Алг о-ритм TEA (Tiny Encryption Algorithm, Крошечный алгоритм шифрования) слишком нов, чтобы его коммент и-ровать [1592]. Другим алгоритмом является Vino [503]. MacGuffin, блочный алгоритм, предложенный Мэттом Блэйзом и мной, также небезопасен [189], он был взломан на той же конференции, на которой он был предл о-жен. BaseKing, похожий по философии на 3-way, но использующий 192-битовый блок [385], слишком нов, чт о-бы его комментировать.

Кроме того, существует множество блочных алгоритмов, разработанных вне криптографического сообщес т-ва. Некоторые из них используются различными военными и правительственными организациями. У меня нет данных о таких алгоритмах. Существуют также десятки частных коммерческих алгоритмов. Некоторые из них могут быть хороши, некоторые нет. Если компания предполагает, что опубликование ее алгоритмов не будет служить интересам компании, то лучше согласиться с ней и не использовать эти алгоритмы.

14.10 Теория проектирования блочного шифра

В раздел 11.1 я описывал принципы Шеннона для смешения и рассеяния. Спустя пятьдесят лет после того, как эти принципы были впервые сформулированы, они остаются краеугольным камнем проектирования хор о-

шего блочного шифра.

Смешение служит для маскировки взаимосвязей между открытым текстом, шифротекстом и ключом. По мните, как даже незначительная зависимость между этими тремя вещами может быть использована при дифф еренциальном и линейном криптоанализе? Хорошее смешение настолько усложняет статистику взаимосвязей, что не работают даже мощные криптографические средства.

Диффузия распространяет влияние отдельных битов открытого текста на как можно большее количество шифротекста. Это также маскирует статистические взаимосвязи и усложняет криптоанализ.

Для безопасности достаточно одного смешения. Алгоритм, состоящий из единственной зависящей от ключа таблицы соответствия 64 битов открытого текста 64 битам шифротекста был бы достаточно сильным. Проблема в том, что для такой таблицы потребовалось бы слишком много памяти: 1020 байтов. Смысл создания блочного шифра и состоит в создании чего-то похожего на такую таблицу, но предъявляющего к памяти более умеренные требования.

Прием состоит в том, чтобы в одном шифре в различных комбинациях периодически перемежать смешив ание (с гораздо меньшими таблицами) и диффузию. Это называется **результирующим шифром**. Иногда блочный шифр, который включает последовательные перестановки и подстановки, называют **сетью перестановок-подстановок** (substitution-permutation network), или **SP-сетью**.

Взгляните снова на функцию f в DES. Перестановка с расширением и P-блок реализуют диффузию, а S-блоки - смешение. Перестановка с расширением и P-блок линейны, S-блоки - нелинейны. Каждая операция сама по себе очень проста, но вместе они работают очень хорошо.

На примере DES также можно показать еще несколько принципов проектирования блочного шифра. Первым является идея **итеративного блочного шифра**. При этом предполагается, что простая функция этапа будет п оследовательно использована несколько раз. Двухэтапный DES не очень силен, чтобы все биты результата зав исели от всех битов ключа и всех битов исходных данных, нужно 5 этапов [1078, 1080]. 16-этапный DES - это сильный алгоритм, 32-этапный DES еще сильнее.

Сети Фейстела

Большинство блочных алгоритмов являются **сетями Фейстела** (Felstel networks). Эта идея датируется нач алом 70-х годов [552, 553]. Возьмите блок длиной n и разделите его на две половины длиной $n/2$: L и R. Конечно, n должно быть четным. Можно определить итеративный блочный шифр, в котором результат j-го этапа определяется результатом предыдущего этапа:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i)$

K_i - это подключ, используемый на j-ом этапе, а f - это произвольная функция этапа.

Эту концепцию можно увидеть в DES, Lucifer, FEAL, Khufu, Khafre, LOKI, COST, CAST, Blowfish и других алгоритмах. Почему это так важно? Гарантируется, что эта функция является обращаемой. Так как для объед инения левой половины с результатом функции этапа используется XOR, следующее выражение обязательно я вляется истинным:

$L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i) \oplus \mathrm{f}(R_{i-1}, K_i) = L_{i-1}$

Гарантируется, что шифр, использующий такую конструкцию, обратим, если можно восстановить исходные данные f на каждом этапе. Сама функция f неважна, он не обязана быть обратимой. Мы можем спроектировать f настолько сложной, насколько захотим, и нам не потребуется реализовывать два различных алгоритма - один для шифрования, а другой для дешифрирования. Структура сети Фейстела автоматически позаботится об этом.

Простые соотношения

DES обладает следующим свойством: если $E_K(P) = C$, то $E_{K'}(P') = C'$, где P', C' и K' - побитовые дополнения P, C и K. Это свойство вдвое уменьшает сложность вскрытия грубой силой. Свойства комплиментарности алг оритма LOKI уменьшают сложность вскрытия грубой силой в 256 раз.

Простое соотношение можно определить как [857]:

Если $E_K(P) = C$, то $E_{\mathrm{f}(K)}(g(P,K)) = h(C,K)$

где f, g и h - простые функции. Под "простыми функциями" я подразумеваю функции, которые вычисляются легко, намного легче, чем выполнение итерации блочного шифра. В DES f представляет собой побитовое л ополнение K, g - побитовое дополнение P, а h - побитовое дополнение C. Это является результатом вкрапления ключа в часть текста с помощью XOR.

Для хорошего блочного шифра не существует простых соотношений. Методы поиска некоторых из подобных слабых мест можно найти в [917].

Групповая структура

При изучении алгоритма возникает вопрос, не образует ли он группу. Элементами группы являются блоки шифротекста для каждого возможного ключа, а групповой операцией является композиция. Изучение групповой структуры алгоритма представляет собой попытку понять, насколько увеличивается пространство шифрования при множественном шифровании.

Полезным, однако, является не вопрос о том, действительно ли алгоритм является группой, а о том, наскол ько он близок к группе. Если не хватает только одного элемента, то алгоритм не образует группу, но двойное шифрование было бы - статистически говоря - просто потерей времени. Работа над DES показала, что DES очень далек от группы. Существует также ряд интересных вопросов о полугруппе, получаемой при шифровании DES. Содержит ли она тождество, то есть, не образует ли она группу? Иными словами, не генерирует ли когда-нибудь некоторая комбинация операций шифрования (не дешифрирования) тождественную функцию? Если так, насколько длинна самая короткая такая комбинация?

Целью исследования является оценка пространства ключей для теоретического вскрытия грубой силой, а р езультат представляет собой наибольшую нижнюю границу энтропии пространства ключей.

Слабые ключи

В хорошем блочном шифре все ключи одинаково сильны. Обычно нет проблем и при алгоритме с малым количеством слабых ключей, таком как DES. Вероятность случайно выбрать один из них очень мала, такой ключ легко проверить и при необходимости отбросить. Однако, иногда эти слабые ключи могут быть задейс т-ованы, если блочный фильтр используется как однонаправленная хэш-функция (см. раздел 18.11).

Устойчивость к дифференциальному и линейному криптоанализу

Исследование дифференциального и линейного криптоанализа значительно прояснило теорию проектиров а-ния хорошего блочного шифра. Авторы IDEA ввели понятие **дифференциалов**, обобщение основной идеи х а-рактеристик [931]. Они утверждали, что можно создавать блочные шифры, устойчивые к вскрытиям такого т и-па. Результатом подобного проектирования и является IDEA [931]. Позднее это понятие было формализовано в [1181, 1182], когда Кайса Ниберг (Kaisa Nyberg) и Ларс Кнудсен (Lars Knudsen) показали, как создавать бло ч-ные шифры доказуемо безопасные по отношению к дифференциальному криптоанализу. Эта теория была ра с-ширена на дифференциалы высших порядков [702, 161, 927, 858, 860] и частичные дифференциалы [860]. К а-жется, что дифференциалы высших порядков применимы только к шифрам с малым числом этапов, но части ч-ные дифференциалы прекрасно объединяются с дифференциалами.

Линейный криптоанализ новее, и он все еще совершенствуется. Были определены понятия классификации ключей [1019] и нескольких приближений [811, 812]. Еще одно расширение криптоанализа можно найти в [1270]. В [938] была предпринята попытка объединить дифференциальный и линейный криптоанализ в одном вскрытии. Пока неясно, какая методика проектирования сможет противостоять подобным вскрытиям.

Кнудсен добился некоторого успеха, рассматривая некоторые необходимые (но, возможно, не достаточные) критерии того, что он назвал **практически безопасными сетями Фейстела** - шифров, устойчивых как к ди ф-ференциальному, так и к линейному криптоанализу [857]. Ниберг ввел для линейного криптоанализа аналог понятия дифференциалов в дифференциальном криптоанализе [1180].

Достаточно интересной кажется двойственность дифференциального и линейного криптоанализа. Эта дво й-ственность становится очевидной как при разработке методики создания хороших дифференциальных характ е-ристик и линейных приближений [164, 1018], так и при разработке критерия проектирования, обеспечивающего устойчивость алгоритмов к обоим типам вскрытия [307]. Пока точно неизвестно, куда заведет это направление исследований. Для начала Дэймен разработал стратегию проектирования алгоритма, основанную на диффере н-циальном и линейном криптоанализе [402].

Проектирование S-блоков

Сила большинства сетей Фейстела - и особенно их устойчивость к дифференциальному и линейному кри п-тоанализу - непосредственно связана с их S-блоками. Это явилось причиной потока исследований, что же обр а-зует хороший S-блок.

S-блок - это просто подстановка: отображение m-битовых входов на n-битовые выходы. Ранее я упоминал об одной большой таблице отображения 64-битовых входов на 64-битовые выходы, такая таблица представляла бы собой S-блок размером 64*64 бита. S-блок с *m*-битовым входом и n-битовым выходом называется **m*n-битовым S-блоком**. S-блоки обычно являются единственным нелинейным действием в алгоритме, именно они

обеспечивают безопасность блочного шифра. В общем случае чем S-блоки больше, тем лучше.

В DES восемь различных 6*4-битовых S-блоков. В Khufu и Khafre единственный 8*32-битовый S-блок, в LOKI 12*8-битовый S-блок, а в Blowfish и CAST 8*32-битовые S-блоки. В IDEA S-блоком по сути является умножение по модулю, это 16*16-битовый S-блок. Чем больше S-блок, тем труднее обнаружить статистические отклонения, нужные для вскрытия с использованием либо дифференциального, либо линейного криптоанализа [653, 729, 1626]. Кроме того, хотя случайные S-блоки обычно не оптимальны с точки зрения устойчивости к дифференциальному и линейному криптоанализу, сильные S-блоки легче найти среди S-блоков большего ра змера. Большинство случайных S-блоков нелинейны, невырождены и обладают сильной устойчивостью к лине йному криптоанализу - и с уменьшением числа входных битов эта доля снижается медленно [1185, 1186, 1187].

Размер m важнее размера n. Увеличение размера n снижает эффективность дифференциального криптоан ализа, но значительно повышает эффективность дифференциального криптоанализа. Действительно, если $n \geq 2^m - m$, то наверняка существует линейная зависимость для входных и выходных битов S-блока. И если $n \geq 2^m$, то линейная зависимость существует только для выходных битов [164].

Заметной частью работы по проектированию S-блоков является изучение **логических функций** [94, 1098, 1262, 1408]. Для обеспечения безопасности булевы функции, используемые в S-блоках, должны отвечать опр еделенным условиям. Они не должны быть ни линейными, ни аффинными, ни даже быть близкими к линейным или аффинным [9, 1177, 1178, 1188]. Количество нулей и единиц должно быть сбалансированным, и не должно быть никаких корреляций между различными комбинациями битов. При изменении на противоположный л юбого входного бита выходные биты должны вести себя независимо. Эти критерии проектирования также связ аны с изучением **функций изгиба**: функций, которые, как может быть показано, являются оптимально нелине йными. Хотя они определены просто и естественно, их изучение очень нелегко [1344, 1216, 947, 905, 1176, 1271, 295, 296, 297, 149, 349, 471, 298].

Очень важным свойством представляется лавинный эффект: сколько выходных битов S-блока изменяется при изменении некоторого подмножества выходных битов. Нетрудно задать для булевых функций условия, в ыполнение которых обеспечивает определенный лавинный эффект, но проектирование таких функций является более сложной задачей. **Строгий лавинный критерий** (strict avalanche criteria, SAC) обеспечивает, что с изм енением одного входного бита изменяется ровно половина выходных битов [1586]. См. также [982, 571, 1262, 399]. В одной из работ эти критерии рассматриваются в терминах утечки информации [1640].

Несколько лет назад криптографы предложили выбирать S-блоки так, чтобы таблица распределения разл ичий для каждого S-блока была однородной. Это обеспечило бы устойчивость к дифференциальному криптоан ализу за счет сглаживания дифференциалов на любом отдельном этапе [6, 443, 444, 1177]. Примером такого проектирования является LOKI. Однако такой подход иногда способствует дифференциальному криптоанализу [172]. Действительно, лучшим подходом является минимизирование максимального дифференциала. Кванджо Ким (Kwangjo Kim) выдвинул пять критериев проектирования S-блоков [834], похожих на критерии проектир ования S-блоков DES.

Выбор хороших S-блоков - не простая задача, существует множество различных идей, как лучше сделать это. Можно выделить четыре главных подхода.

1. Случайно выбрать. Ясно, что небольшие случайные S-блоки небезопасны, но большие случайные S-блоки могут оказаться достаточно хороши. Случайные S-блоки с восемью и более входами дост аточно сильны [1186, 1187]. Еще лучше 12-битовые S-блоки. Устойчивость S-блоков возрастает, если они одновременно являются и случайными, и зависящими от ключа. В IDEA используются большие зависящие от ключа S-блоки.

2. Выбрать и проверить. В некоторых шифрах свойства S-блоков, генерированных случайным образом, проверяются. Примеры такого подхода содержатся в [9, 729].

3. Разработать вручную. При этом математический аппарат используется крайне незначительно: S-блоки создаются с использованием интуитивных приемов. Барт Пренел (Bart Preneel) заявил, что "... теор етически интересные критерии недостаточны [для выбора булевых функций S-блоков] ...", и что "... н еобходимы специальные критерии проектирования" [1262].

4. Разработать математически. S-блоки создаются в соответствии с математическими законами, поэтому они обладают гарантированной надежностью по отношению к дифференциальному и линейному криптоанализу, а также хорошими диффузными свойствами. Прекрасный пример такого подхода можно найти в [1179].

Существует ряд призывов объединить "математический" и "ручной" подходы [1334], но реально, по вид имому, конкурируют случайно выбранные S-блоки и S-блоки с определенными свойствами. Конечно преимущ еством последнего подхода является оптимизация против известных методов вскрытия - дифференциального и линейного криптоанализа - но обеспечиваемая этим подходом степень защиты от неизвестных методов вскр ы-

тия также неизвестна. Разработчикам DES было известно о дифференциальном криптоанализе, и его S-блоки были оптимизированы соответствующим образом. Скорее всего, о линейном криптоанализе они не знали, и S-блоки DES очень слабы по отношению к такому способу вскрытия [1018]. Случайно выбранные S-блоки в DES были бы слабее против дифференциального криптоанализа, но сильнее против линейного криптоанализа.

С другой стороны случайные S-блоки могут не быть оптимальными по отношению к данным способам вскрытия, но они могут быть достаточно большими и, следовательно, достаточно надежными. Кроме того, они, скорее всего, будут достаточно устойчивы и против неизвестных способов вскрытия. Спор все еще кипит, но лично мне кажется, что S-блоки должны быть такими большими, насколько это возможно, случайными и зав исеть от ключа.

Проектирование блочного шифра

Проектировать блочный шифр нетрудно. Если вы рассматривает 64-битовый блочный шифр как перестано вку 64-битовых чисел, ясно, что почти все эти перестановки безопасны. Трудность состоит в проектировании блочного шифра, который не только безопасен, но также может быть легко описан и просто реализован.

Легко можно спроектировать блочный шифр, если вы используете память, достаточную для размещения S-блоков 48*32. Трудно спроектировать небезопасный вариант DES, если вы собираетесь использовать в нем 128 этапов. При длине ключа 512 битов не стоит беспокоиться о том, нет ли какой-либо зависящей от ключа ко мплиментарности.

14.11 Использование однонаправленных хэш-функций

Сымым простым способом использовать для шифрования однонаправленную хэш-функцию является хэш ирование предыдущего блока шифротекста, объединенного с ключом, а затем выполнение XOR результата с т екущим блоком открытого текста:

$C_i = P_i \oplus H(K, C_{i-1})$

$P_i = C_i \oplus H(K, P_{i-1})$

Установите длину блока равной длине результата однонаправленной хэш-функции. По сути это приводит к использованию однонаправленной хэш-функции как блочного шифра в режиме CFB. При помощи аналогичной конструкции можно использовать однонаправленную хэш-функцию и в режиме OFB:

$C_i = P_i \oplus S_i$; $S_i = H(K, C_{i-1})$

$P_i = C_i \oplus S_i = H(K, C_{i-1})$

Надежность такой схемы определяется безопасностью однонаправленной хэш-функции.

Karn

Этот метод, изобретенный Филом Карном (Phil Karn) и открытый им для свободного использования, создает обратимый алгоритм шифрования из определенных однонаправленных хэш-функций.

Алгоритм работает с 32-байтовыми блоками открытого текста и шифротекста. Длина ключа может быть произвольной, хотя определенные дины ключей более эффективны для конкретных однонаправленных хэш-функций. Для однонаправленных хэш-функций MD4 и MD5 лучше всего подходят 96-байтовые ключи.

Для шифрования сначала разбейте открытый текст на две 16-байтовых половины: P_l и P_r. Затем разбейте на две 48-байтовых половины ключ: K_l и K_r.

$P = P_l , P_r$,

$K = K_l , K_r$

Добавьте K_l к P_l и выполните хэширование однонаправленной хэш-функцией, затем выполните XOR резул ьтата с P_r, получая C_r, правую половину шифротекста. Затем, добавьте K_r к C_r выполните хэширование однон аправленной хэш-функцией. Выполните XOR результата с P_l , получая C_l. Наконец, объедините C_r и C_l , получая шифротекст.

$C_r = P_r \oplus H(P_l, K_l)$

$C_l = P_l \oplus H(C_r, K_r)$

$C = C_l, C_r$

Для дешифрирования просто инвертируйте процесс. Добавьте K_r к Cr, выполните хэширование и XOR р езультата с C_l , получая P_l. Добавьте K_l к P_l, выполните хэширование и XOR результата с C_r , получая P_r.

$P_l = C_l \oplus H(C_r, K_r)$

$P_r = C_r \oplus H(P_l, K_l)$

$P = P_l, P_r$

Общая структура Karn совпадает с структурой множества других блочных алгоритмов, рассмотренных в этом разделе. У алгоритма только два этапа, так как его сложность определяется однонаправленной хэш-функцией. А, так как ключ используется только как вход хэш-функции, он не может быть раскрыт даже при помощи вскрытия с выбранным открытым текстом, если, конечно, безопасна используемая однонаправленная хэш-функция.

Luby-Rackoff

Майкл Любы (Michael Luby) и Чарльз Ракофф (Charles Rackoff) показали, что Karn не является безопасным [992]. Рассмотрим два одноблочных сообщения: AB и AC. Если криптоаналитику известны открытый текст и шифротекст первого сообщения, а также первая половина открытого текста второго сообщения, то он может легко вычислить все второе сообщение. Хотя такое вскрытие с известным открытым текстом работает только при определенных условиях, оно представляет собой главную проблему в безопасности алгоритма.

Ее удается избежать при помощи трехэтапного алгоритма шифрования [992,1643,1644]. Он использует три различных хэш-функции: H_1, H_2 и H_3. Дальнейшие исследования показали, что H_1 может совпадать с H_2, или H_2 может совпадать с H_3, но не одновременно [1193]. Кроме того, H_1, H_2 и H_3 не могут быть основаны на итерациях одной и той же базовой функции [1643]. В любом случае при условии, что $H(k,x)$ ведет себя как псевдослучайная функция, трехэтапная версия выглядит следующим образом:

(1) Разделите ключ на две половины: K_l и K_r.

(2) Разделите блок открытого текста на две половины: L_0 и R_0.

(3) Объедините K_l и L_0 и выполните хэширование. Выполните XOR результата хэширования с R_0, получая R_1:

$R_1 = R_0 \oplus H(K_l, L_0)$

(4) Объедините K_r и R_1 и выполните хэширование. Выполните XOR результата хэширования с L_0, получая L_1

$L_1 = L_0 \oplus H(K_r, R_1)$

(5) Объедините K_l и L_1 и выполните хэширование. Выполните XOR результата хэширования с R_1, получая R_2:

$R_2 = R_1 \oplus H(K_l, L_1)$

(6) Объедините L_1 и R_2, получая сообщение.

Шифр краткого содержания сообщения

Шифр краткого содержания сообщения(Message Digest Cipher, M DC), изобретенный Питером Гутманном (Peter Cutmann) [676], представляет собой способ превратить однонаправленные хэш-функции в блочный шифр, работающий в режиме CFB. Шифр работает почти также быстро, как и хэш-функция, и по крайней мере н астолько же безопасен. Оставшаяся часть этого раздела предполаг ает знакомство с главой 18.

Хэш функции, например MD5 и SHA, используют 512-битовый текстовый блок для преобразования входн ого значения (128 битов в MD5, и 160 битов в SHA) в результат того же размера. Это преобразование необрат имо, но прекрасно подходит для режима CFB: и для шифрования, и для дешифрирования используется одна и та же операция.

Рассмотрим MDC с SHA. MDC использует 160-битовый блок и 512-битовый ключ. Используется побочный эффект хэш-функции, когда в качестве прежнего хэш-значения берется входной блок открытого текста (160 б итов), а 512-битовый вход хэш-функции играет роль ключа (см. Рис 14.5). Обычно при использовании хэш-функции для хэширования некоторого входа 512-битовый вход меняется при хэшировании каждого нового 512-битового блока. Но в данном случае 512-битовый вход становится неизменяемым ключом.

MDC можно использовать с любой однонаправленной хэш-функцией: MD4, MD5, Snefru, и т.д. Он незап атентован и может быть совершенно бесплатно использован кем угодно когда угодно и для чего угодно [676].

Однако лично я не верю в эту схему. Можно подобрать такой способ взлома, на противостояние которому хэш-функция не была рассчитана. Хэш-функции не обязаны противостоять вскрытию с выбранным открытым текстом, когда криптоаналитик выбирает некоторые начальные 160-битовые значения, получает их "зашифрованными" одним и тем же 512-битовым "ключом" и пользуется этим для получения некоторой и нформации об используемом 512-битовом ключе. Так как разработчики хэш-функций не должны беспокоиться о такой возможности, считать ваш шифр безопасным по отношению к приведенному способу вскрытия - не лу ч-

шая идея.

Безопасность шифров, основанных на однонаправленных хэш-функциях

Хотя эти конструкции и могут быть безопасными, они зависят от используемой однонаправленной хэш-функции. Хорошая однонаправленная хэш-функция не обязательно дает безопасный алгоритм шифрования. Существуют различные криптографические требования. Например, линейный криптоанализ бесполезен против однонаправленных хэш-функциях, но действенен против алгоритмов шифрования. Однонаправленная хэш-функция, такая как SHA, может обладать определенными линейными характеристиками, которые, не влияя на ее безопасность как однонаправленной хэш-функции, могут сделать небезопасным ее использование в таком алгоритме шифрования, как MDC. Мне неизвестно ни о каких результатах криптоанализа использования ко нкретной однонаправленной хэш-функции в качестве блочного шифра. Прежде чем использовать их дождитесь проведения подобного анализа.

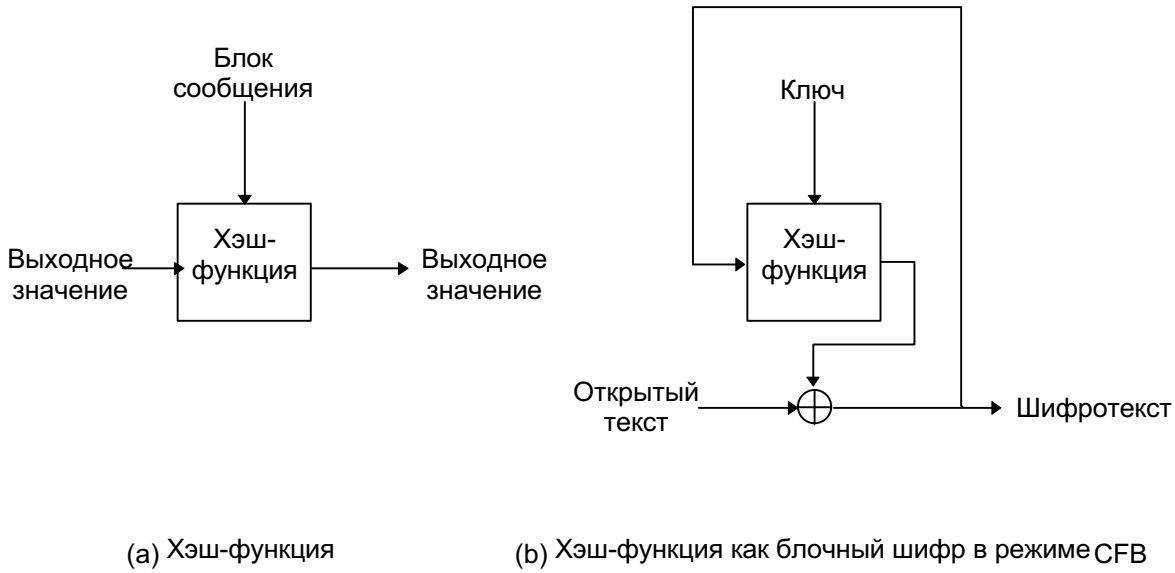


Рис. 14-5. Шифр краткого содержания сообщения (MDC).

14.12 Выбор блочного алгоритма

Это очень трудное решение. DES почти наверняка небезопасен при использовании против правительств в еликих держав, если только вы не шифруете одним ключом очень малые порции данных. Возможно этот алг оритм пока неплох против кого-нибудь другого, но вскоре и это изменится. Машины для вскрытия ключа DES грубой силой скоро станут по карману всем организациям.

Предложенные Бихамом зависимые от ключа S-блоки DES будут безопасны в течение по крайней мере н ескольких лет, может быть за исключением использования против самых хорошо обеспеченных противников. Если необходимая безопасность должна быть обеспечена на десятилетия, или вы опасаетесь криптоаналитич еских усилий правительств великих держав, воспользуйтесь тройным DES с тремя независимыми ключами.

Небеполезны и другие алгоритмы. Мне нравится Blowfish, потому что он быстр, и потому что я его прид умал. Неплохо выглядит 3-WAY, возможно все в порядке и с ГОСТом. Проблема посоветовать что-нибудь с остоит в том, что NSA почти наверняка обладает набором эффективных криптоаналитических приемов, которые до сих пор засекречены, и я не знаю, какие алгоритмы могут быть вскрыты. В Табл. 14.3 для сравнения прив едены временные соотношения для некоторых алгоритмов.

Мой любимый алгоритм - IDEA. Его 128-битовый ключ в сочетании с устойчивостью к общеизвестным средствам криптоанализа - вот источники моего теплого и нежного чувства к этому алгоритму. Этот алгоритм анализировался различными группами, и никаких серьезных замечаний не было опубликовано. В отсутствие необычайных криптоаналитических прорывов я сегодня ставлю на IDEA.

Табл. 14-3.
Скорости шифрования для некоторых блочных шифров наi486SX/33 МГц

Алгоритм	Скорость шифрования (Кбайт/с)	Алгоритм	Скорость шифрования (Кбайт/с)
Blowfish (12 этапов)	182	MDC (с MD4)	186

Blowfish (16 этапов)	135	MDC (с MD5)	135
Blowfish (20 этапов)	110	MDC (с SHA)	23
DES	35	NewDES	233
FEAL-8	300	REDOC II	1
FEAL-16	161	REDOC III	78
FEAL-32	91	RC5-32/8	127
ГОСТ	53	RC5-32/12	86
IDEA	70	RC5-32/16	65
Khufu (16 этапов)	221	RC5-32/20	52
Khufu (24 этапов)	153	SAFER (6 этапов)	81
Khufu (32 этапов)	115	SAFER (8 этапов)	61
Luby-Rackoff (с MD4)	47	SAFER (10 этапов)	49
Luby-Rackoff (с MD5)	34	SAFER (12 этапов)	41
Luby-Rackoff (с SHA)	11	3-Way	25
Lucifer	52	Тройной DES	12

Глава 15
Объединение блочных шифров

Существует множество способов объединять блочные алгоритмы для получения новых алгоритмов. Стимулом создавать подобные схемы является желание повысить безопасность, не пробираясь через тернии создания нового алгоритма. DES является безопасным алгоритмом, он подвергался криптоанализу добрых 20 лет и, тем не менее, наилучшим способом вскрытия остается грубая сила. Однако ключ слишком короток. Разве не плохо было бы использовать DES в качестве компонента другого алгоритма с более длинным ключом? Это позволило бы получить преимущества длинного ключа с гарантией двух десятилетий криптоанализа.

Одним из способов объединения является **многократное шифрование** - для шифрования одного и того же блока открытого текста алгоритм шифрования используется несколько раз с несколькими ключами. Шифрование каскадом похоже на многократное шифрование, но использует различные алгоритмы. Существуют и другие методы.

Повторное шифрование блока открытого текста одним и тем же ключом с помощью того же или другого алгоритма неразумно. Повторное использование того же алгоритма не увеличивает сложность вскрытия грубой силой. (Не забывайте, мы предполагаем, что алгоритм, включая количество шифрований, известен криптоаналитику.) При различных алгоритмах сложность вскрытия грубой силой может возрастать, а может и остаться неизменной. Если вы собираетесь использовать методы, описанные в этой главе, убедитесь, что ключи для последовательных шифрований различны и независимы.

15.1 Двойное шифрование

Наивным способом повысить безопасность алгоритма является шифрование блока дважды с двумя различными ключами. Сначала блок шифруется первым ключом, а затем получившийся шифротекст шифруется вторым ключом. Дешифрирование является обратным процессом.

$$C = E_{K_2}(E_{K_1}(P))$$

$$P = D_{K_1}(D_{K_2}(C))$$

Если блочный алгоритм образует группу (см. раздел 11.3), то всегда существует K_3, для которого

$$C = E_{K_2}(E_{K_1}(P)) = E_{K_3}(P)$$

Если алгоритм не образует группу, то при помощи исчерпывающего поиска взломать получающийся дважды зашифрованный блок шифротекста намного сложнее. Вместо 2^n (где n - длина ключа в битах), потребуется 2^{2n} попыток. Если алгоритм использует 64-битовый ключ, для обнаружения ключей, которыми дважды зашифрован шифротекст, потребуется 2^{128} попыток.

Но при вскрытии с известным открытым текстом это не так. Меркл и Хеллман [1075] придумали способ обменять память на время, который позволяет вскрыть такую схему двойного шифрования за 2^{n+1} шифрований, а не за 2^{2n}. (Они использовали эту схему против DES, но результаты можно обобщить на все блочные алгоритмы.) Это вскрытие называется "**встреча посередине**", с одной стороны выполняется шифрование а с другой - дешифрирование, получившиеся посередине результаты сравниваются.

В этом вскрытии криптоаналитику известны P_1, C_1, P_2 и C_2, такие что

$$C_1 = E_{K_2}(E_{K_1}(P_1))$$

$$C_2 = E_{K_2}(E_{K_1}(P_2))$$

Для каждого возможного K (или K_1, или K_2), криптоаналитик рассчитывает $E_K(P_1)$ и сохраняет результат в памяти. Собрав все результаты, он для каждого K вычисляет $D_K(C_1)$ и ищет в памяти такой же результат. Если такой результат обнаружен, то возможно, что текущий ключ - K_2, а ключ для результата в памяти - K_1. Затем криптоаналитик шифрует P_1 с помощью K_1 и K_2. Если он получает C_2, то он может гарантировать (с вероятностью успеха 1 к 2^{2n-2m}, где m - размер блока), что он узнал и K_1, и K_2. Если это не так, он продолжает поиск. Максимальное количество попыток шифрования, которое ему, возможно, придется предпринять, равно $2*2^n$, или 2^{n+1}. Если вероятность ошибки слишком велика, он может использовать третий блок шифротекста, обеспечивая вероятность успеха 1 к 2^{2n-3m}. Существуют и другие способы оптимизации [912].

Для такого вскрытия нужен большой объем памяти: 2^n блоков. Для 56-битового ключа нужно хранить 2^{56} 64-битовых блоков, или 10^{17} байтов. Такой объем памяти пока еще трудно себе представить, но этого хватает, чтобы убедить самых параноидальных криптографов в том, что двойным шифрованием пользоваться не стоит.

При 128-битовом ключе для хранения промежуточных результатов потребуется 10^{39} байтов. Если предположить, что есть способ хранить бит информации, используя единственный атом алюминия, устройство памяти, нужное для выполнения такого вскрытия, будет представлять собой алюминиевый куб с ребром, длиной 1 км. Кроме того, вам понадобится куда-то его поставить! Вскрытие "встреча посередине" кажется невозможным для ключей такого размера.

Другим способом двойного шифрования, который иногда называют **Davies-Price**, является вариантом CBC [435].

$$C_i = E_{K_1}(P_1 \oplus E_{K_2}(C_{i-1}))$$
$$P_i = D_{K_1}(C_i) \oplus E_{K_2}(C_{i-1}))$$

Утверждается, что "у этого режима нет никаких особых достоинств", к тому же он, по видимому, так же чувствителен ко вскрытию "встреча посередине" как и другие режимы двойного шифрования.

15.2

Тройное шифрование с двумя ключами

В более интересном методе, предложенном Тачменом в [1551], блок обрабатывается три раза с помощью двух ключей: первым ключом, вторым ключом и снова первым ключом. Он предлагает, чтобы отправитель сначала шифровал первым ключом, затем дешифрировал вторым, и окончательно шифровал первым ключом. Получатель расшифровывает первым ключом, затем шифрует вторым и, наконец, дешифрирует первым.

$$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_1}(C)))$$

Иногда такой режим называют **шифрование-дешифрирование-шифрование** (encrypt-decrypt-encrypt, EDE) [55]. Если блочный алгоритм использует n-битовый ключ, то длина ключа описанной схемы составляет $2n$ бит. Любопытный вариант схемы шифрование-дешифрирование-шифрование был разработан в IBM для совместимости с существующими реализациями алгоритма: задание двух одинаковых ключей эквивалентно одинарному шифрованию. этим ключом. Схема шифрование-дешифрирование-шифрование сама по себе не обладает никакой безопасностью, но этот режим был использован для улучшения алгоритма DES в стандартах X9.17 и ISO 8732 [55, 761].

K_1 и K_2 чередуются для предотвращения описанного выше вскрытия "встреча посередине". Если $C = E_{K_1}(E_{K_1}(E_{K_1}(P)))$, то криптоаналитик для любого возможного K_1 может заранее вычислить $E_{K_1}(E_{K_1}(P))$ и затем выполнить вскрытие. Для этого потребуется только 2^{n+2} шифрований.

Тройное шифрование с двумя ключами устойчиво к такому вскрытию. Но Меркл и Хеллман разработали другой способ размена памяти на время, который позволяет взломать этот метод шифрования за 2^{n-1} действий, используя 2^n блоков памяти [1075].

Для каждого возможного K_2 расшифруйте 0 и сохраните результат. Затем расшифруйте 0 для каждого возможного K_1, чтобы получить P. Выполните тройное шифрование P, чтобы получить C, и затем расшифруйте C ключом K_1. Если полученное значение совпадает с значением (хранящемся в памяти), полученным при дешифровании 0 ключом K_2, то пара K_1 K_2 является возможным результатом поиска. Проверьте, так ли это. Если нет, продолжайте поиск.

Выполнение этого вскрытия с выбранным открытым текстом требует огромного объема памяти. Понадобится 2^n времени и памяти, а также 2^m выбранных открытых текстов. Вскрытие не очень практично, но все же чувствительность к нему является слабостью алгоритма.

Пауль ван Оорсчот (Paul van Oorschot) и Майкл Винер (Michael Wiener) преобразовали это вскрытие ко вскрытию с известным открытым текстом, для которого нужно p известных открытых текстов. В примере предполагается, что используется режим EDE.

(1) Предположить первое промежуточное значения a.

(2) Используя известный открытый текст, свести в таблицу для каждого возможного K_1 второе промежуточное значение b, при первом промежуточном значении, равном a:

$b = D_{K_1}(C)$

где C - это шифротекст, полученный по известному открытому тексту.

(3) Для каждого возможного K_2 найти в таблице элементы с совпадающим вторым промежуточным значение

b:

$b = E_{K_2}(a)$

(4) Вероятность успеха равно p/m, где p - число известных открытых текстов, а m - размер блока. Если совпадения не обнаружены, выберите другое a и начните сначала.

Вскрытие требует $2^{n+m}/p$ времени и p - памяти. Для DES это равно $2^{120}/p$ [1558]. Для p, больших 256, это вскрытие быстрее, чем исчерпывающий поиск.

Тройное шифрование с тремя ключами

Если вы собираетесь использовать тройное шифрование, я рекомендую три различных ключа. Общая длина ключа больше, но хранение ключа обычно не является проблемой. Биты дешевы.

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_3}(C)))$$

Для наилучшего вскрытия с разменом памяти на время, которым является "встреча посередине", потребуется 2^{2n} действий и 2^n блоков памяти [1075]. Тройное шифрование с тремя независимыми ключами безопасно настолько, насколько на первый взгляд кажется безопасным двойное шифрование.

Тройное шифрование с минимальным ключом (TEMK)

Существует безопасный способ использовать тройное шифрование с двумя ключами, противостоящий описанному вскрытию и называемый Тройным шифрованием с минимальным ключом (Triple Encryption with Minimum Key, TEMK) [858]. Фокус в тои, чтобы получить три ключа из: X_1 и X_2.

$$K_1 = E_{X_1}(D_{X_2}(E_{X_1}(T_1)))$$
$$K_2 = E_{X_1}(D_{X_2}(E_{X_1}(T_2)))$$
$$K_3 = E_{X_1}(D_{X_2}(E_{X_1}(T_3)))$$

T_1, T_2 и T_3 представляют собой константы, которые необязательно хранить в секрете. Эта схема гарантирует, что для любой конкретной пары ключей наилучшим будет вскрытие с известным открытым текстом.

Режимы тройного шифрования

Недостаточно просто определить тройное шифрование, нужно выбрать один из способов его использования. Решение зависит от требуемых безопасности и эффективности. Вот два возможных режима тройного шифрования:

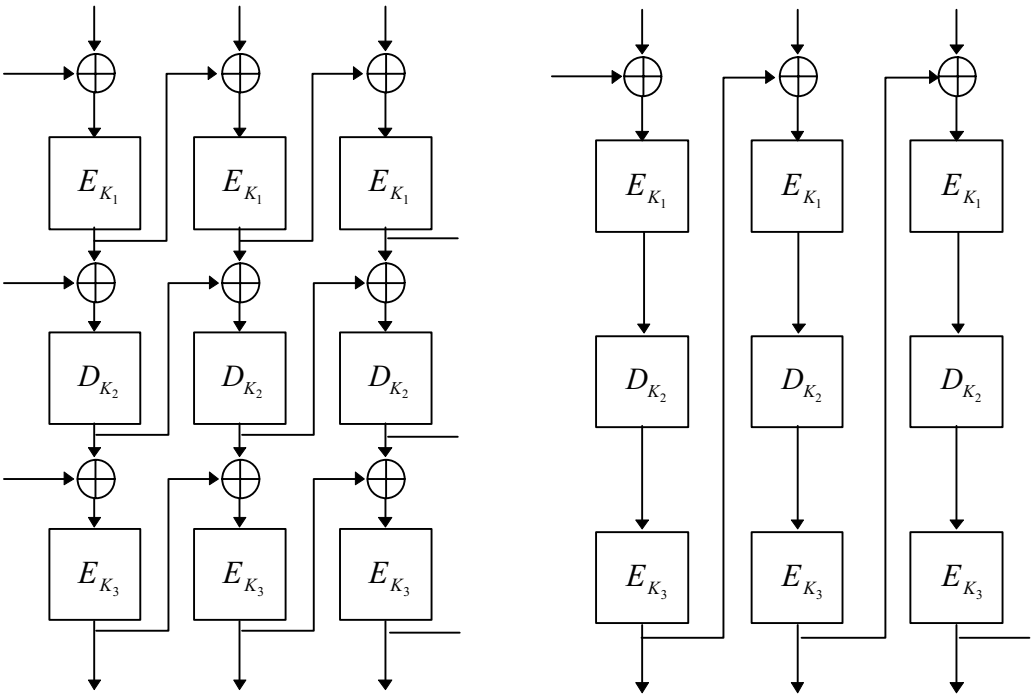
Внутренний CBC: Файл три раза шифруется в режиме CBC (см. 14tha). Для этого нужно три различных IV.

$$C_i = E_{K_3}(S_i \oplus C_{i-1}); S_i = D_{K_2}(T_i \oplus S_{i-1}); T_i = E_{K_1}(P_i \oplus T_{i-1})$$
$$P_i = T_{i-1} \oplus D_{K_1}(T_i); T_i = S_{i-1} \oplus E_{K_2}(S_i); S_i = C_{i-1} \oplus D_{K_3}(C_i)$$

C_0, S_0 и T_0 являются IV.

Внешний CBC: Файл троекратно шифруется в режиме CBC (см. 14thb). Для этого нужен один IV.

$$C_i = E_{K_3}(D_{K_2}(E_{K_1}(P_i \oplus C_{i-1})))$$
$$P_i = C_{i-1} \oplus D_{K_1}(E_{K_2}(D_{K_3}(C_i)))$$



(a) Внутренний CBC

(b) Внешний CBC

Рис. 15-1. Тройное шифрование в режиме CBC.

Для обоих режимов нужно больше ресурсов, чем для однократного шифрования: больше аппаратуры или больше времени. Однако при трех шифрующих микросхемах производительность внутреннего CBC не меньше, чем при однократном шифровании. Так как три шифрования CBC независимы, три микросхемы могут быть загружены постоянно, подавая свой выход себе на вход.

Напротив во внешнем CBC обратная связь находится снаружи по отношению к трем шифрованиям. Это означает, что даже с тремя микросхемами производительность будет равна только одной трети производительности при однократном шифровании. Чтобы получить ту же производительность для внешнего CBC, потребуется чередование IV (см. раздел 9.12):

$$C_i = E_{K_3}(D_{K_2}(E_{K_1}(P_i \oplus C_{i-3})))$$

В этом случае C_0, C_{-1} и C_{-2} являются IV. Это не поможет при программной реализации, разве только при использовании параллельного компьютера.

К сожалению менее сложный режим является также и менее безопасным. Бихам проанализировал различные режимы по отношению к дифференциальному криптоанализу и обнаружил, что безопасность внутреннего CBC по сравнению с однократным шифрованием увеличивается незначительно. Если рассматривать тройное шифрование как единый большой алгоритм, то внутренние обратные связи позволяют вводить внешнюю и известную информацию внутрь алгоритма, что облегчает криптоанализ. Для дифференциальных вскрытий нужно огромное количество выбранных шифротекстов, что делает эти вскрытия не слишком практичными, но этих результатов должно хватить, чтобы насторожить параноидальных пользователей. Анализ устойчивости алгоритмов к вскрытиям грубой силой и "встречей посередине" показал, что оба варианта одинаково безопасны [806].

Кроме этих существуют и другие режимы. Можно зашифровать файл один раз в режиме ECB, затем дважды в CBC, или один раз в CBC, один в ECB и еще раз в CBC, или дважды в CBC и один раз в ECB. Бихам показал, что эти варианты не безопаснее, чем однократный DES, против вскрытия дифференциальным криптоанализом с выбранным открытым текстом [162]. Он не оставил больших надежд и для других вариантов. Если вы собираетесь применять тройное шифрование, используйте внешнюю обратную связь.

Варианты тройного шифрования

Прежде, чем появились доказательства того, что DES не образует группу, для многократного шифрования предлагались различные схемы. Одним из способов обеспечить то, что тройное шифрование не выродится в однократное, было изменение эффективной дины блока. Простым методом является добавление бита-заполнителя. Между первым и вторым, а также между вторым и третьим шифрованиями текст дополняется

строкой случайных битов (см. Рис. 15.2). Если *PP* - это функция дополнения, то:

$$C = E_{K_3}(PP(E_{K_2}(PP(E_{K_1}(P)))))$$

Это дополнение не только разрушает шаблоны, но также обеспечивает перекрытие блоков шифрования, как кирпичей в стене. К длине сообщения добавляется только один блок.

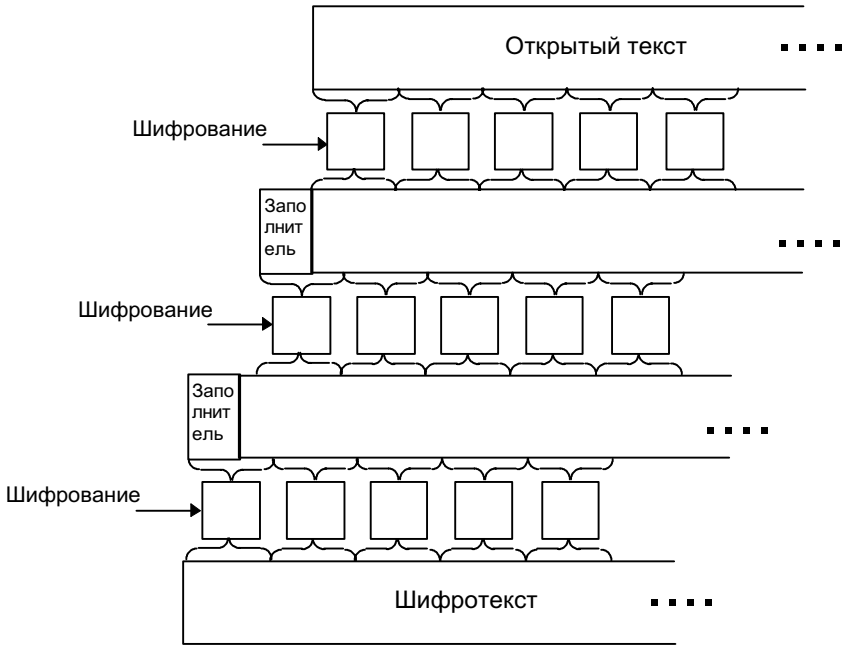


Рис. 15-2. Тройное шифрование с заполнением.

Другой метод, предложенный Карлом Эллисоном (Carl Ellison), использует некоторую функцию независимой от ключа перестановки между тремя шифрованиями. Перестановка должна работать с большими блоками - 8 Кбайт или около этого, что делает эффективный размер бока для этого варианта равным 8 Кбайтам. При условии, что перестановка выполняется быстро, этот вариант ненамного медленнее, чем базовое тройное шифрование.

$$C = E_{K_3}(T(E_{K_2}(T(E_{K_1}(P)))))$$

T собирает входные блоки (до 8 Кбайт в длину) и использует генератор псевдослучайных чисел для их перемешивания. Изменение одного бита входа приводит к изменению 8 байтов результата первого шифрования, к изменению до 64 байтов результата второго шифрования и к изменению до 512 байтов результата третьего шифрования. Если каждый блочный алгоритм работает в режиме CBC, как было первоначально предложено, то изменение единичного бита входа скорее всего приведет к изменению всего 8-килобайтового блока, даже если этот блок не является первым.

Самый последний вариант этой схемы отвечает на вскрытие внутреннего CBC, выполненное Бихамом, добавлением процедуры отбеливания, чтобы замаскировать структуру открытых текстов. Эта процедура представляет собой потоковую операцию XOR с криптографически безопасным генератором псевдослучайных чисел и ниже обозначена как *R*. *T* мешает криптоаналитику определить *a priori*, какой ключ используется для шифрования любого заданного байта входа последнего шифрования. Второе шифрование обозначено *nE* (шифрование с циклическим использованием *n* различных ключей):

$$C = E_{K_3}(R(T(nE_{K_2}(T(E_{K_1}(P))))))$$

Все шифрования выполняются в режиме ECB, используется не меньше *n*+2 ключей шифрования и криптографически безопасный генератор псевдослучайных чисел.

Эта схема была предложена для использования вместе с DES, но она работает с любым блочным алгоритмом. Результаты криптоанализа такой схемы мне неизвестны.

15.3 Удвоение длины блока

В академическом сообществе давно спорят на тему, достаточна ли 64-битовая длина блока. С одной стороны 64-битовый блок обеспечивает диффузию открытого текста только в 8 байтах шифротекста. С другой стороны более длинный блок затрудняет безопасную маскировку структуры, кроме того, больше возможностей ошибить-

ся.

Существуют предложения удваивать длину блока алгоритма с помощью многократного шифрования [299]. Прежде, чем реализовывать одно из них, оцените возможность вскрытия "встреча посередине" . Схема Ричарда Аутбриджа (Richard Outerbridge) [300], показанная на 12-й, не более безопасна, чем тройное шифрование с одинарным блоком и двумя ключами [859].

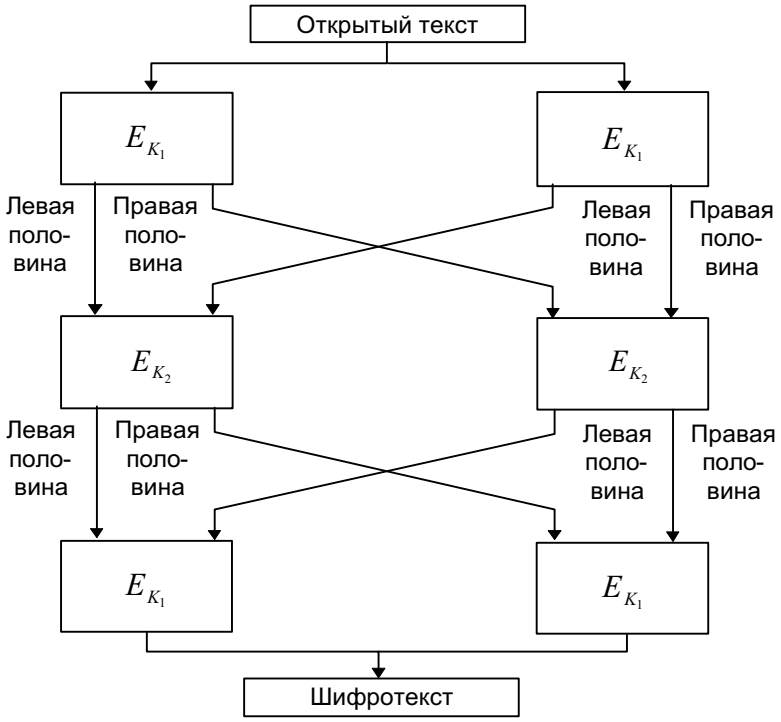


Рис. 15-3. Удвоение длины блока.

Однако я не рекомендую использовать подобный прием . Он не быстрее обычного тройного шифрования : для шифрования двух блоков данных все также нужно шесть шифрований . Характеристики обычного тройного шифрования известны, а за новыми конструкциями часто прячутся новые проблемы .

15.4 Другие схемы многократного шифрования

Проблемой тройного шифрования с двумя ключами является то, что для увеличения вдвое пространства ключей нужно выполнять три шифрования каждого блока открытого текста . Разве не здорово было бы найти какой-нибудь хитрый способ объединить два шифрования, которые удвоили бы пространство ключей ?

Двойной OFB/счетчик

Этот метод использует блочный алгоритм для генерации двух потоков ключей, которые используются для шифрования открытого текста .

$$S_i = E_{K_1}(S_{i-1} \oplus I_1); I_1 = I_1 + 1$$
$$T_i = E_{K_2}(T_{i-1} \oplus I_2); I_2 = I_2 + 1$$
$$C_i = P_i \oplus S_i \oplus T_i$$

S_i и T_i - внутренние переменные, а I_1 и I_2 - счетчики. Две копии блочного алгоритма работают в некотором гибридном режиме OFB/счетчик, а открытый текст, S_i и T_i объединяются с помощью XOR. Ключи K_1 и K_2 независимы. Результаты криптоанализа этого варианта мне неизвестны .

ECB + OFB

Этот метод был разработан для шифрования нескольких сообщений фиксированной длины, например, бл оков диска [186, 188]. Используются два ключа: K_1 и K_2. Сначала для генерации маски для блока нужной длины используется выбранный алгоритм и ключ . Эта маска будет использована повторно для шифрования сообщений теми же ключами. Затем выполняется XOR открытого текста сообщения и маски . Наконец результат XOR шифруется с помощью выбранного алгоритма и ключа K_2 в режиме ECB.

Анализ этого метода проводился только в той работе, в которой он и был опубликован . Понятно, что он не слабее одинарного шифрования ECB и возможно также силен, как и двойное применение алгоритма . Вероятно, криптоаналитик может выполнять поиск ключей независимо, если он получит несколько открытых текстов фа йлов, зашифрованных одним ключом.

Чтобы затруднить анализ идентичных блоков в одних и тех же местах различных сообщений, можно испол ьзовать IV. В отличии от использования IV в других режимах в данном случае перед шифрованием ECB выполняется XOR каждого блока сообщения с IV.

Мэтт Блэйз (Matt Blaze) разработал этот режим для своей UNIX Cryptographic File System (CFS, криптографическая файловая система). Это хороший режим, поскольку скрытым состоянием является только одно ши фрование в режиме ECB, маска может быть сгенерирована только один раз и сохранена . В CFS в качестве блочного алгоритма используется DES.

xDES^i

В [1644, 1645] DES используется как компонент ряда блочных алгоритмов с увеличенными размерами кл ючей и блоков. Эти схемы никак не зависят от DES, и в них может использоваться любой блочный алгоритм .

Первый, xDES^1, представляет собой просто схему Luby-Rackoff с блочным шифром в качестве базовой функции (см. раздел 14.11). Размер блока в два раза больше размера блока используемого блочного фильтра, а размер ключа в три раза больше, чем у используемого блочного фильтра . В каждом из 3 этапов правая полов ина шифруется блочным алгоритмом и одним из ключей, затем выполняется XOR результата и левой половины, и половины переставляются.

Это быстрее, чем обычное тройное шифрование, так как тремя шифрованиями шифруется блок, длина кот орого в два раза больше длины блока используемого блочного алгоритма . Но при этом существует простое вскрытие "встреча посередине", которое позволяет найти ключ с помощью таблицы размером 2^k, где k - это размер ключа блочного алгоритма. Правая половина блока открытого текста шифруется с помощью всех во зможных значений K_1, выполняется XOR с левой половиной открытого текста и полученные значения сохран яются в таблице. Затем правая половина шифротекста шифруется с помощью всех возможных значений K_3, и выполняется поиск совпадений в таблице. При совпадении пара ключей K_1 и K_3 - возможный вариант правого ключа. После нескольких повторений вскрытия останется только один кандидат. Таким образом, xDES^1 не является идеальным решением. Даже хуже, существует вскрытие с выбранным открытым текстом, доказывающее, что xDES^1 не намного сильнее используемого в нем блочного алгоритма [858].

В xDES^2 эта идея расширяется до 5-этапного алгоритма, размер блока которого в 4 раза, а размер ключа в 10 раз превышают размеры блока и ключа используемого блочного шифра . На 11th показан один этап xDES^2, каждый из четырех подблоков по размеру равен блоку используемого блочного шифра, а все 10 ключей независимы.

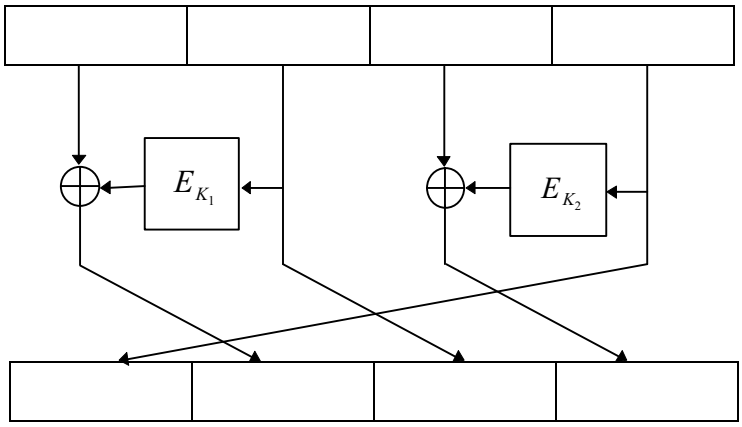


Рис. 15-4. Один этап xDES^2.

К тому же, эта схема быстрее, чем тройное шифрование: для шифрования блока, который в четыре раза больше блока используемого блочного шифра, нужно 10 шифрований . Однако этот метод чувствителен к ди ф-ференциальному криптоанализу [858] и использовать его не стоит. Такая схема остается чувствительной к ди ф-ференциальному криптоанализу, даже если используется DES с независимыми ключами этапов.

Для $i \geq 3$ xDES^i вероятно слишком велик, чтобы использовать его в качестве блочного алгоритма . Например, размер блока для xDES^3 в 6 раз больше, чем у лежащего в основе блочного шифра, ключ в 21 раз длиннее, а для шифрования блока, который в 6 раз длиннее блока лежащего в основе блочного шифра, нужно 21 шифрование .

Это медленнее, чем тройное шифрование.

Пятикратное шифрование

Если тройное шифрование недостаточно безопасно - может быть, вам нужно шифровать ключи тройного шифрования, используя еще более сильный алгоритм - то кратность шифрования можно увеличить. Очень устойчиво к вскрытию "встреча посередине" пятикратное шифрование. (Аргументы, аналогичные рассмотренным для двойного шифрования, показывают, что четырехкратное шифрование по сравнению с тройным лишь незначительно повышает надежность.)

$$C = E_{K_1}(D_{K_2}(E_{K_3}(D_{K_2}(E_{K_1}(P)))))$$
$$P = D_{K_1}(E_{K_2}(D_{K_3}(E_{K_2}(D_{K_1}(C)))))$$

Эта схема обратно совместима с тройным шифрованием, если $K_1 = K_2$, и с однократным шифрованием, если $K_1 = K_2 = K_3$. Конечно, она будет еще надежней, если использовать пять независимых ключей.

15.5 Уменьшение длины ключа в CDMF

Этот метод был разработан IBM для продукта CDMF (Commercial Data Masking Facility, Коммерческое средство маскирования данных) (см. раздел 24.8), чтобы превратить 56-битовый ключ DES в 40-битовый, разрешенный для экспорта [785]. Предполагается, что первоначальный ключ DES содержит биты четности.

(1) Обнуляются биты четности: биты 8, 16, 24, 32, 40, 48, 56, 64.

(2) Результат этапа (1) шифруется с помощью DES ключом 0xc408b0540ba1e0ae, результат шифрования об ъединяется посредством XOR с результатом этапа (1).

(3) В результате этапа (2) обнуляются следующие биты: 1, 2, 3, 4, 8, 16, 17, 18, 19, 2.0, 2.4, 32, 33, 34, 35, 36, 40, 48, 49, 50, 51, 52, 56, 64.

(4) Результат этапа (3) шифруется с помощью DES ключом 0xef2c041ce6382fe6. Полученный ключ использ уется для шифрования сообщения.

Не забывайте, что этот метод укорачивает ключ и, следовательно, ослабляет алгоритм.

15.6 Отбеливание

Отбеливанием (whitening) называется способ, при котором выполняется XOR части ключа с входом блочного алгоритма и XOR другой части ключа с выходом блочного алгоритма. Впервые этот метод был применен для варианта DESX, разработанного RSA Data Security, Inc., а затем (по-видимому, независимо) в Khufu и Khafre. (Ривест и дал имя этому методу, это необычное использование слова.)

Смысл этих действий в том, чтобы помешать криптоаналитику получить пару "открытый текст/шифротекст" для лежащего в основе блочного алгоритма. Метод заставляет криптоаналитика угадывать не только ключ алгоритма, но и одно из значений отбеливания. Так как XOR выполняется и перед, и после блочного алгоритма, считается, что этот метод устойчив против вскрытия "встреча посередине".

$$C = K_3 \oplus E_{K_2}(P \oplus K_1)$$
$$P = K_1 \oplus D_{K_2}(C \oplus K_3)$$

Если $K_1 = K_2$, то для вскрытия грубой силой потребуется $2^{n+m/p}$ действий, где n - размер ключа, m - размер блока, и p - количество известных открытых текстов. Если K_1 и K_2 различны, то для вскрытия грубой силой с тремя известными открытыми текстами потребуется 2^{n+m+1} действий. Против дифференциального и линейного криптоанализа, такие меры обеспечивают защиту только для нескольких битов ключа. Но с вычислительной точки зрения это очень дешевый способ повысить безопасность блочного алгоритма.

15.7 Многократное последовательное использование блочных алгоритмов

А как насчет шифрования сначала алгоритмом A и ключом K_A, а затем еще раз алгоритмом B и ключом K_B? Может быть у Алисы и Боба различные представления о том, какой алгоритм безопаснее : Алиса хочет пользоваться алгоритмом A, а Боб - алгоритмом B. Этот прием, иногда называемый **последовательным использованием** (cascading), можно распространить и на большее количество алгоритмов и ключей.

Пессимисты утверждали, что совместное использование двух алгоритмов не гарантирует повышения без опасности. Алгоритмы могут взаимодействовать каким-то хитрым способом, что на самом деле даже *уменьшит*. Даже тройное шифрование тремя различными алгоритмами может не быть настолько безопасным, насколько

вам это кажется. Криптография - достаточно темное искусство, если вы не совсем понимаете, что делаете, то можете легко попасть в беду.

Действительность намного светлее. Упомянутые предостережения верны, только если различные ключи зависят друг от друга. Если все используемые ключи независимы, то сложность взлома последовательности алгоритмов по крайней мере не меньше, чем сложность взлома первого из применяемых алгоритмов [1033]. Если второй алгоритм чувствителен к вскрытию с выбранным открытым текстом, то первый алгоритм может облегчить это вскрытие и при последовательном использовании сделать второй алгоритм чувствительным к вскрытию с известным открытым текстом. Такое возможное облегчение вскрытия не ограничивается только алгоритмами шифрования: если вы позволите кому-то другому определить любой из алгоритмов, делающих что-то с вашим сообщением до шифрования, стоит удостовериться, что ваше шифрование устойчиво по отношению к вскрытию с выбранным открытым текстом. (Обратите внимание, что наиболее часто используемым алгоритмом для сжатия и оцифровки речи до модемных скоростей, применяемым перед любым алгоритмом шифрования, является CELP, разработанный NSA.)

Это можно сформулировать и иначе: При использовании вскрытия с выбранным открытым текстом последовательность шифров взломать не легче, чем любой из шифров последовательности [858]. Ряд результатов показал, что последовательное шифрование взломать по крайней мере не легче, чем самый сильный из шифров последовательности, но в основе этих результатов лежат некоторые несформулированные предположения [528]. Только если алгоритмы коммутативны, как в случае каскадных потоковых шифров (или блочных шифров в режиме OFB), надежность их последовательности не меньше, чем у сильнейшего из используемых алгоритмов.

Если Алиса и Боб не доверяют алгоритмам друг друга, они могут использовать их последовательно. Для потоковых алгоритмов их порядок не имеет значения. При использовании блочных алгоритмов Алиса может сначала использовать алгоритм A, а затем алгоритм B. Боб, который больше доверяет алгоритму B, может использовать алгоритм B перед алгоритмом A. Между алгоритмами они могут вставить хороший потоковый шифр. Это не причинит вреда и может значительно повысить безопасность.

Не забудьте, что ключи для каждого алгоритма последовательности должны быть независимыми. Если алгоритм A использует 64-битовый ключ, а алгоритм B - 128-битовый ключ, то получившаяся последовательность должна использовать 192-битовый ключ. При использовании зависимых ключей у пессимистов гораздо больше шансов оказаться правыми.

15.8 Объединение нескольких блочных алгоритмов

Вот другой способ объединить несколько блочных алгоритмов, безопасность которого гарантировано будет по крайней мере не меньше, чем безопасность обоих алгоритмов. Для двух алгоритмов (и двух независимых ключей):

(1) Генерируется строка случайных битов R того же размера, что и сообщение M.

(2) R шифруется первым алгоритмом.

(3) $M \oplus R$ шифруется вторым алгоритмом.

(4) Шифротекст сообщения является объединением результатов этапов (2) и (3).

При условии, что строка случайных битов действительно случайна, этот метод шифрует M с помощью одноразового блокнота, а затем содержимое блокнота и получившееся сообщение шифруются каждым из двух алгоритмов. Так как и то, и другое необходимо для восстановления M, криптоаналитику придется взламывать оба алгоритма. Недостатком является удвоение размера шифротекста по сравнению с открытым текстом.

Этот метод можно расширить для нескольких алгоритмов, но добавление каждого алгоритма увеличивает шифротекст. Сама по себе идей хороша, но, как мне кажется, не очень практична.

Глава 16
Генераторы псевдослучайных последовательностей и потоковые шифры

16.1 Линейные конгруэнтные генераторы

Линейными конгруэнтными генераторами являются генераторы следующей формы

$X_n = (aX_{n-1} + b) \bmod m$

в которых X_n - это n-ый член последовательности, а X_{n-1} - предыдущий член последовательности. Переменные a, b и m - постоянные: a - **множитель**, b - **инкремент**, и m - модуль. Ключом, или затравкой, служит значение X_0.

Период такого генератора не больше, чем m. Если a, b и m выбраны правильно, то генератор будет **генератором с максимальным периодом** (иногда называемым максимальной длиной), и его период будет равен m. (Например, b должно быть взаимно простым с m.) Подробное описание выбора констант для получения максимального периода можно найти в [863, 942]. Еще одной хорошей статьей по линейным конгруэнтным генераторам и их теории является [1446].

В 15-й, взятой из [1272,], перечисляются хорошие константы линейных конгруэнтных генераторов. Все они обеспечивают генераторы с максимальным периодом и, что даже более важно, удовлетворяют спектральному тесту на случайность для размерностей 2, 3, 4, 5 и 6 [385, 863]. Таблица организована по максимальному произведению, которое не вызывает переполнения в слове указанной длины.

Преимуществом линейных конгруэнтных генераторов является их быстрота за счет малого количества операций на бит.

К несчастью линейные конгруэнтные генераторы нельзя использовать в криптографии, так как они предсказуемы. Впервые линейные конгруэнтные генераторы были взломаны Джимом Ридсом (Jim Reeds) [1294, 1295, 1296], а затем Джоан Бояр (Joan Boyar) [1251]. Ей удалось также вскрыть квадратичные генераторы:

$Xn = (aX_{n-1}^2 + bX_{n-1} + c) \bmod m$

и кубические генераторы:

$Xn = (aX_{n-1}^3 + bX_{n-1}^2 + c\,X_{n-1} + d) \bmod m$

Другие исследователи расширили идеи Бояр, разработав способы вскрытия любого полиномиального генератора [923, 899, 900]. Были взломаны и усеченные линейные конгруэнтные генераторы [581, 705, 580], и усеченные линейные конгруэнтные генераторы с неизвестными параметрами [1500, 212]. Таким образом была доказана бесполезность конгруэнтных генераторов для криптографии.

Табл. 16-1.
Константы для линейных конгруэнтных генераторов

Переполняется при	a	b	m
2^{20}	106	1283	6075
2^{21}	211	1663	7875
2^{22}	421	1663	7875
2^{23}	430	2531	11979
	936	1399	6655
	1366	1283	6075
2^{24}	171	11213	53125
	859	2531	11979
	419	6173	29282
	967	3041	14406
2^{25}	141	28411	134456
	625	6571	31104
	1541	2957	14000
	1741	2731	12960
	1291	4621	21870
	205	29573	139968
2^{26}	421	17117	81000
	1255	6173	29282

	281	28411	134456
2^{27}	1093	18257	86436
	421	54773	259200
	1021	24631	116640
	1021	25673	121500
2^{28}	1277	24749	117128
	741	66037	312500
	2041	25673	121500
2^{29}	2311	25367	120050
	1807	45289	214326
	1597	51749	244944
	1861	49297	233280
	2661	36979	175000
	4081	25673	121500
	3661	30809	145800
2^{30}	3877	29573	139968
	3613	45289	214326
	1366	150889	714025
2^{31}	8121	28411	134456
	4561	51349	243000
	7141	54773	259200
2^{32}	9301	49297	233280
	4096	150889	714025
2^{33}	2416	374441	1771875
2^{34}	17221	107839	510300
	36261	66037	312500
2^{35}	84589	45989	217728

Однако, линейные конгруэнтные генераторы сохраняют свою полезность для некриптографических прил о-жений, например, для моделирования. Они эффективны и в большинстве используемых эмпирических тестах демонстрируют хорошие статистические характеристики. Важную информацию о линейных конгруэнтных г е-нераторах и их теории можно найти в [942].

Объединение линейных конгруэнтных генераторов

Был предпринят ряд попыток объединения линейных конгруэнтных генераторов [1595, 941]. Криптографи-ческая безопасность полученных результатов не повышается, но они обладают более длинными периодами и лучшими характеристиками в некоторых статистических тестах. Для 32-битовых компьютеров можно испол ь-зовать следующий генератор [941]:

```
static long sl = 1 ; /* "long" должно быть 32-битовым целым. */ static long s2 = 1 ;

#define MODMULT(a,b,c,m,s) q = s/a; s = b*(s-a*q) - c*q; if (s<0) s+=m ;
/* MODMIJLT(a,b,c,nl,s) рассчитывает s*b mod m при условии, что m=a*b+c и 0 <= c < m. */

/* combinedLCG возвращает действительное псевдослучайное значение в диапазоне
* (0,1). Она объединяет линейные конгруэнтные генераторы с периодами
* 2^31-85 и 2^31-249, и ее период равен произведению этих двух простых чисел. */

double combinedLCG ( void )
{
     long q ;
     long z ;
     MODMULT ( 53668, 40014, 12211, 2147483563L, s1 )
     MODMULT ( 52774, 40692, 3791, 2147483399L, s2 )
     z = s1 - s2 ;
     if ( z < 1 )
            z += 2147483562 ;
     return z * 4.656613e-10 ;
}

/* В общем случае перед использованием combinedLCG вызывается initLCG. */
void initLCG( long InitS1, long InitS2 )
{
     sl = InitS1;
     s2 = InitS2;
}
```

Этот генератор работает при условии, что компьютер может представить все целые числа между $-2^{31}+85$ и $2^{31}-249$. Переменные s_1 и s_2 глобальны и содержат текущее состояние генератора. Перед первым вызовом их необходимо проинициализировать. Для переменной s_1 начальное значение должно лежать в диапазоне между 1

и 2147483562, для переменной s_2 - между 1 и 2147483398. Период генератора близок к 10^{18}.

На 16-битовом компьютере используйте другой генератор:

```
static int s1 = 1 ; /* "int" должно быть 16-битовым целым. */
static int s2 = 1 ;
static int s3 = 1 ;

#define MODMULT(a,b,c,m,s) q = s/a; s = b*(s-a*q) - c*q; if (s<0) s+=m ;
/* combinedLCG возвращает действительное псевдослучайное значение в диапазоне
* (0,1). Она объединяет линейные конгруэнтные генераторы с периодами 2^15-405,
* 2^15-1041 и 2^15-1111, и ее период равен произведению этих трех простых чисел. */

double combinedLCG ( void )
{
     long q ;
     long z ;

     MODMULT ( 206, 157, 21, 32363, s1 )
     MODMULT ( 217, 146, 45, 31727, s2 )
     MODMULT ( 222, 142, 133, 31657, s3 )
     z = s1 - s2 ;
     if ( z < 1 )
             z -= 32362 ;
     z += s3 ;
     if ( z < 1 )
             z += 32362 ;
     return z * 3.0899e-5 ;

}

/* В общем случае перед использованием combinedLCG вызывается initLCG. */
void initLCG( long InitS1, long InitS2, long InitS3)
{
     s1 = InitS1;
     s2 = InitS2;
     s3 = InitS3;
}
```

Этот генератор работает при условии, что компьютер может представить все целые числа между -32363 и 32363. Переменные s_1, s_2 и s_3 глобальны и содержат текущее состояние генератора. Перед первым вызовом их необходимо проинициализировать. Для переменной s_1 начальное значение должно лежать в диапазоне между 1 и 32362, для переменной s_2 - между 1 и 31726, для переменной s_3 - между 1 и 31656. Период генератора равен $1.6*10^{13}$. Для обоих генераторов константа b равна 0.

16.2 Сдвиговые регистры с линейной обратной связью

Последовательности сдвиговых регистров используются как в криптографии, так и в теории кодирования. Их теория прекрасно проработана, потоковые шифры на базе сдвиговых регистров являлись рабочей лошадкой военной криптографии задолго до появления электроники.

Сдвиговый регистр с обратной связью состоит из двух частей: сдвигового регистра и **функции обратной связи** (см. 15th). Сдвиговый регистр представляет собой последовательность битов. (Количество битов определяется **длиной** сдвигового регистра. Если длина равна n битам, то регистр называется n-битовым сдвиговым регистром.) Всякий раз, когда нужно извлечь бит, все биты сдвигового регистра сдвигаются вправо на 1 позицию. Новый крайний левый бит является функцией всех остальных битов регистра. На выходе сдвигового регистра оказывается один, обычно младший значащий, бит. **Периодом** сдвигового регистра называется длина получаемой последовательности до начала ее повторения.

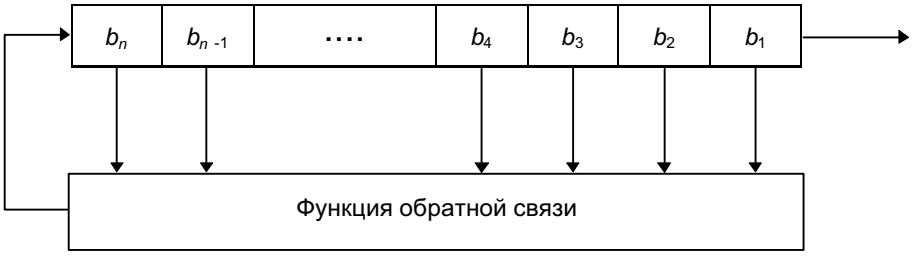


Рис. 16-1. Сдвиговый регистр с обратной связью

Криптографам нравились потоковые шифры на базе сдвиговых регистров: они легко реализовывались с помощью цифровой аппаратуры. Я лишь слегка затрону математическую теорию. В 1965 году Эрнст Селмер (Ernst Selmer), главный криптограф норвежского правительства, разработал теорию последовательности сдвиговых регистров [1411]. Соломон Голомб (Solomon Golomb), математик NSA, написал книгу, излагающие ее некоторые свои резальтаты и результаты Селмера [643]. См. также [970, 971, 1647].

Простейшим видом сдвигового регистра с обратной связью является **линейный сдвиговый регистр с обратной связью** (linear feedback shift register, или LFSR) (см. 14th). Обратная связь представляет собой просто XOR некоторых битов регистра, перечень этих битов называется **отводной последовательностью** (tap sequence). Иногда такой регистр называется **конфигурацией Фиббоначи**. Из-за простоты последовательности обратной связи для анализа LFSR можно использовать довольно развитую математическую теорию. Криптографы любят анализировать последовательности, убеждая себя, что эти последовательности достаточно случа йны, чтобы быть безопасными. LFSR чаще других сдвиговых регистров используются в криптографии.

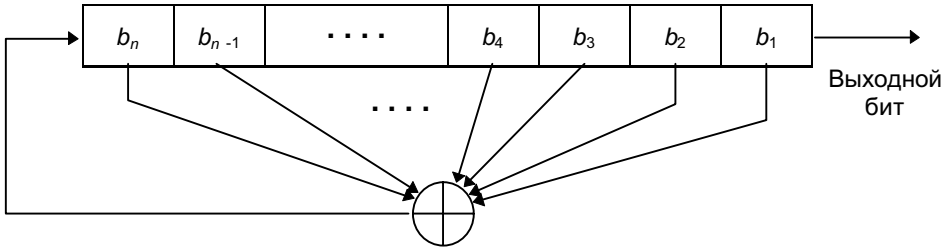


Рис. 16-2. Сдвиговый регистр с линейной обратной связью.

На 13-й показан 4-битовый LFSR с отводом от первого и четвертого битов. Если его проинициализировать значением 1111, то до повторения регистр будет принимать следующие внутренние состояния :

1 1 1 1
0 1 1 1
1 0 1 1
0 1 0 1
1 0 1 0
1 1 0 1
0 1 1 0
0 0 1 1
1 0 0 1
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
1 1 0 0
1 1 1 0

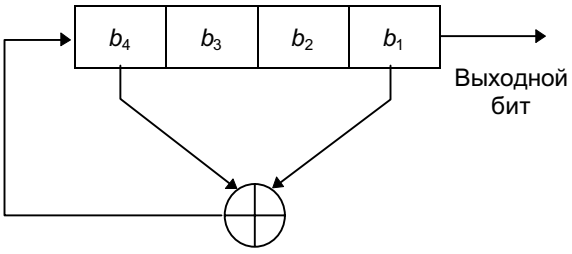


Рис. 16-3. 4-битовый LFSR.

Выходной последовательностью будет строка младших значащих битов :

1 1 1 1 0 1 0 1 1 0 0 1 0 0 0. . . .

n-битовый LFSR может находиться в одном из 2^n-1 внутренних состояний. Это означает, что теоретически такой регистр может генерировать псевдослучайную последовательность с периодом 2^n-1 битов. (Число внутренних состояний и период равны 2^n-1, потому что заполнение LFSR нулями, приведет к тому, что сдвиговый регистр будет выдавать бесконечную последовательность нулей, что абсолютно бесполезно .) Только при определенных отводных последовательностях LFSR циклически пройдет через все 2^n-1 внутренних состояний, такие

LFSR являются LFSR с максимальным периодом. Получившийся результат называется **M-последовательностью**.

Для того, чтобы конкретный LFSR имел максимальный период, многочлен, образованный из отводной последовательности и константы 1, должен быть примитивным по модулю 2. **Степень** многочлена является длиной сдвигового регистра. Примитивный многочлен степени n - это неприводимый многочлен, который является делителем $x^{2^n-1}+1$, но не является делителем x^d+1 для всех d, являющихся делителями 2^n-1 (см. раздел 11.3). Соответствующую математическую теорию можно найти в [643, 1649, 1648].

В общем случае не существует простого способа генерировать примитивные многочлены данной степени по модулю 2. Проще всего выбирать многочлен случайным образом и проверять, не является ли он примитивным. Это нелегко - и чем-то похоже на проверку, не является ли простым случайно выбранное число - но многие математические пакеты программ умеют решать такую задачу. Ряд методов приведен в [970, 971].

Некоторые, но, конечно же, не все, многочлены различных степеней, примитивные по модулю 2, приведены в 14-й [1583, 643, 1649, 1648, 1272, 691]. Например, запись (32, 7, 5, 3, 2, 1, 0) означает, что следующий многочлен примитивен по модулю 2:

$$x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$$

Это можно легко обобщить для LFSR с максимальным периодом. Первым числом является длина LFSR. Последнее число всегда равно 0, и его можно опустить. Все числа, за исключением 0, задают отводную последовательность, отсчитываемую от левого края сдвигового регистра. То есть, члены многочлена с меньшей степенью соответствуют позициям ближе к правому краю регистра.

Продолжая пример, запись (32, 7, 5, 3, 2, 1, 0) означает, что для взятого 32-битового сдвигового регистра новый бит новый бит генерируется с помощью XOR тридцать второго, седьмого, пятого, третьего, второго и первого битов (см. 12th), получающийся LFSR будет иметь максимальную длину, циклически проходя до повторения через 2^{32}-1 значений.

Код для этого LFSR на языке C выглядит следующим образом:

```
int LFSR ( ) {
     static unsigned long ShiftRegister = 1;
     /* Все, кроме 0. */
     ShiftRegister = ((((ShiftRegister >> 31)
            ^(ShiftRegister >> 6)
            ^(ShiftRegister >> 4)
            ^(ShiftRegister >> 2)
            ^(ShiftRegister >> 1)
            ^ShiftRegister))
            & 0x00000001)
            <<31)
            | (ShiftRegister >> 1) ;
     return ShiftRegister & 0x00000001;
}
```

Если сдвиговый регистр длиннее компьютерного слова, код усложняется, но не намного.

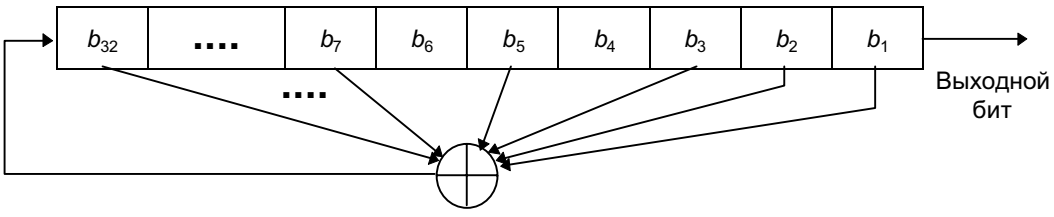


Рис. 16-4. 32-битовый LFSR с максимальной длиной.

Табл. 16-2.
Некоторые примитивные многочлены по модулю 2

(1, 0)	(7, 3, 0)	(14, 5, 3, 1, 0)	(18, 5, 2, 1, 0)
(2, 1, 0)	(8, 4, 3, 2, 0)	(15, 1, 0)	(19, 5, 2, 1, 0)
(3, 1, 0)	(9, 4, 0)	(16, 5, 3.2, 0)	(20, 3, 0)
(4, 1, 0)	(10, 3, 0)	(17, 3, 0)	(21, 2, 0)
(5, 2, 0)	(11, 2, 0)	(17, 5, 0)	(22, 1, 0)
(6, 1, 0)	(12, 6, 4, 1, 0)	(17, 6, 0)	(23, 5, 0)
(7, 1, 0)	(13, 4, 3, 1, 0)	(18, 7, 0)	(24, 4, 3, 1, 0)

(25, 3, 0)
(26, 6, 2, 1, 0)
(27, 5, 2, 1, 0)
(28, 3, 0)
(29, 2, 0)
(30, 6, 4, 1.0)
(31, 3, 0)
(31, 6, 0)
(31, 7, 0)
(31, 13, 0)
(32, 7, 6, 2, 0)
(32, 7, 5, 3, 2, 1, 0)
(33, 13, 0)
(33, 16, 4, 1, 0)
(34, 8, 4, 3, 0)
(34, 7, 6, 5, 2, 1, 0)
(35, 2, 0)
(135, 11, 0)
(135, 16, 0)
(135, 22, 0)
(136, 8, 3, 2, 0)
(137, 21, 0)
(138, 8, 7, 1, 0)
(139, 8, 5, 3, 0)
(140, 29, 0)
(141, 13, 6, 1, 0)
(142, 21, 0)
(143, 5, 3, 2, 0)
(144, 7, 4, 2, 0)
(145, 52, 0)
(145, 69, 0)
(146, 5, 3, 2, 0)
(147, 11, 4, 2, 0)
(148, 27, 0)
(149, 10, 9, 7, 0)
(150, 53, 0)
(151, 3, 0)
(151, 9, 0)
(151, 15, 0)
(151, 31, 0)
(151, 39, 0)
(151, 43, 0)
(151, 46, 0)
(151, 51, 0)
(151, 63, 0)
(151, 66, 0)
(151, 67, 0)
(151, 70, 0)
(36, 11, 0)
(36, 6, 5, 4, 2, 1, 0)
(37, 6, 4, 1, 0)
(37, 5, 4, 3, 2, 1, 0)
(38, 6, 5, 1, 0)
(39, 4, 0)
(40, 5, 4, 3, 0)
(41, 3, 0)
(42, 7, 4, 3, 0)
(42, 5, 4, 3, 2, 1, 0)
(43, 6, 4, 3, 0)
(44, 6, 5, 2, 0)
(45, 4, 3, 1, 0)
(46, 8, 7, 6, 0)
(46, 8, 5, 3, 2, 1, 0)
(47, 5, 0)
(48, 9, 7, 4, 0)
(48, 7, 5, 4, 2, 1, 0)
(49, 9, 0)
(49, 6, 5, 4, 0)
(50, 4, 3, 2, 0)
(51, 6, 3, 1, 0)
(52, 3, 0)
(53, 6, 2, 1, 0)
(54, 8, 6, 3, 0)
(54, 6, 5, 4, 3, 2, 0)
(55, 24, 0)
(55, 6, 2, 1, 0)
(56, 7, 4, 2, 0)
(57, 7, 0)
(57, 5, 3, 2, 0)
(58, 19.0)
(58, 6, 5, 1, 0)
(59, 7, 4, 2, 0)
(59, 6, 5, 4, 3, 1, 0)
(60, 1, 0)
(61, 5, 2, 1, 0)
(62, 6, 5, 3, 0)
(63, 1, 0)
(64, 4, 3, 1, 0)
(65, 18, 0)
(65, 4, 3, 1, 0)
(66, 9, 8, 6, 0)
(66, 8, 6, 5, 3, 2, 0)
(67, 5, 2, 1, 0)
(152, 6, 3, 2, 0)
(153, 1, 0)
(153, 8, 0)
(154, 9, 5, 1, 0)
(155, 7, 5, 4, 0)
(156, 9, 5, 3, 0)
(157, 6, 5, 2, 0)
(158, 8, 6, 5, 0)
(159, 31, 0)
(159, 34, 0)
(159, 40, 0)
(160, 5, 3, 2, 0)
(161, 18, 0)
(161, 39, 0)
(161, 60, 0)
(162, 8, 7, 4, 0)
(163, 7, 6, 3, 0)
(164, 12, 6, 5, 0)
(165, 9, 8, 3, 0)
(166, 10, 3, 2, 0)
(167, 6, 0)
(170, 23, 0)
(172, 2, 0)
(174, 13, 0)
(175, 6, 0)
(175, 16, 0)
(175, 18, 0)
(175, 57, 0)
(177, 8, 0)
(177, 22, 0)
(1 77, 88, 0)
(68, 9, 0)
(68, 7, 5, 1, 0)
(69, 6, 5, 2, 0)
(70, 5, 3, 1, 0)
(71, 6, 0)
(71, 5, 3, 1, 0)
(72, 10, 9, 3, 0)
(72, 6, 4, 3, 2, 1, 0)
(73, 25, 0)
(73, 4, 3, 2, 0)
(74, 7, 4, 3, 0)
(75, 6, 3, 1, 0)
(76, 5, 4, 2, 0)
(77, 6, 5, 2, 0)
(78, 7, 2, 1, 0)
(79, 9, 0)
(79, 4, 3, 2, 0)
(80, 9, 4, 2, 0)
(80, 7, 5, 3, 2, 1, 0)
(81, 4, 0)
(82, 9, 6, 4, 0)
(82, 8, 7, 6, 1, 0)
(83, 7, 4, 2, 0)
(84, 13, 0)
(84, 8, 7, 5, 3, 1, 0)
(85, 8, 2, 1, 0)
(86, 6, 5, 2, 0)
(87, 13, 0)
(87, 7, 5, 1, 0)
(88, 11, 9, 8, 0)
(88, 8, 5, 4, 3, 1, 0)
(89, 38, 0)
(89, 51, 0)
(89, 6, 5, 3, 0)
(90, 5, 3, 2, 0)
(91, 8, 5, 1, 0)
(91, 7, 6, 5, 3, 2, 0)
(92, 6, 5, 2, 0)
(93, 2, 0)
(94, 21, 0)
(94, 6, 5, 1, 0)
(95, 11, 0)
(95, 6, 5, 4, 2, 1, 0)
(96, 10, 9, 6, 0)
(96, 7, 6, 4, 3, 2, 0)
(178, 87, 0)
(183, 56, 0)
(194, 87, 0)
(198, 65, 0)
(201, 14, 0)
(201, 17, 0)
(201, 59, 0)
(201, 79, 0)
(202, 55, 0)
(207, 43, 0)
(212, 105, 0)
(218, 11, 0)
(218, 15, 0)
(218, 71, 0)
(218.83, 0)
(225, 32, 0)
(225, 74, 0)
(225, 88, 0)
(225, 97, 0)
(225, 109, 0)
(231, 26, 0)
(231, 34, 0)
(234, 31, 0)
(234, 103, 0)
(236, 5, 0)
(250, 103, 0)
(255, 52, 0)
(255, 56, 0)
(255, 82, 0)
(258, 83, 0)
(266, 47, 0)
(97, 6, 0)
(98, 11, 0)
(98, 7, 4, 3, 1, 0)
(99, 7, 5, 4, 0)
(100, 37, 0)
(100, 8, 7, 2, 0)
(101, 7, 6, 1, 0)
(102, 6, 5, 3, 0)
(103, 9, 9)
(104, 11, 10, 1, 0)
(105, 16, 0)
(106, 15, 0)
(107, 9, 7, 4, 0)
(108, 31, 0)
(109, 5, 4, 2.0)
(110, 6, 4, 1, 0)
(111, 10, 0)
(111, 49, 0)
(113, 9, 0)
(113, 15, 0)
(113, 30, 0)
(114, 11, 2, 1, 0)
(115, 8, 7, 5, 0)
(116, 6, 5, 2, 0)
(117, 5, 2, 1, 0)
(118, 33, 0)
(119, 8, 0)
(119, 45, 0)
(120, 9, 6, 2, 0)
(121, 18, 0)
(122, 6, 2, 1, 0)
(123, 2, 0)
(124, 37, 0)
(125, 7, 6, 5, 0)
(126, 7, 4, 2, 0)
(127, 1, 0)
(127, 7, 0)
(127, 63, 0)
(128, 7, 2, 1, 0)
(129, 5, 0)
(130, 3, 0)
(131, 8, 3, 2, 0)
(132, 29, 0)
(133, 9, 8, 2, 0)
(134, 57, 0)
(270, 133, 0)
(282, 35, 0)
(282, 43, 0)

(286, 69, 0)	(378, 43, 0)	(521, 168, 0)	(2281, 915, 0)
(286, 73, 0)	(378, 107, 0)	(607, 105, 0)	(2281, 1029, 0)
(294, 61, 0)	(390, 89, 0)	(607, 147, 0)	(3217, 67, 0)
(322, 67, 0)	(462, 73, 0)	(607, 273, 0)	(3217, 576, 0)
(333, 2, 0)	(521, 32, 0)	(1279, 216, 0)	(4423, 271, 0)
(350, 53, 0)	(521, 48, 0)	(1279, 418, 0)	(9689, 84, 0)
(366, 29, 0)	(521, 158, 0)	(2281, 715, 0)	

Обратите внимание, что у всех элементов таблицы нечетное число коэффициентов . Я привел такую длинную таблицу, так как LFSR часто используются для криптографии с потоковыми шифрами, и я хотел, чтобы разные люди могли подобрать различные примитивные многочлены . Если $p(x)$ примитивен, то примитивен и $x^n p(1/x)$, поэтому каждый элемент таблицы на самом деле определяет два примитивных многочлена .

Например, если (a, b, 0) примитивен, то примитивен и (a, a - b, 0). Если примитивен (a, b, c, d, 0), то примитивен и (a, a - d, a - c, a - b, 0). Математически:

если примитивен $x^a + x^b + 1$, то примитивен и $x^a + x^{a-b} + 1$

если примитивен $x^a + x^b + x^c + x^d + 1$, то примитивен и $x^a + x^{a-d} + x^{a-c} + x^{a-b} + 1$

Быстрее всего программно реализуются примитивные трехчлены, так как для генерации нового бита тужно выполнять XOR только двух битов сдвигового регистра. Действительно, все многочлены обратной связи, приведенные в 14-й, являются **разреженными**, то есть, у них немного коэффициентов. Разреженность всегда представляет собой источник слабости, которой иногда достаточно для вскрытия алгоритма . Для криптографических алгоритмов гораздо лучше использовать **плотные** примитивные многочлены, те, у которых много коэффициентов. Применяя плотные многочлены, особенно в качестве части ключа, можно использовать значительно более короткие LFSR.

Генерировать плотные примитивные многочлены по модулю 2 нелегко . В общем случае для генерации примитивных многочленов степени k нужно знать разложение на множители числа 2^k-1. Примитивные многочлены можно найти в следующих трех хороших работах: [652, 1285, 1287].

Сами по себе LFSR являются хорошими генераторами псевдослучайных последовательностей, но они обладают некоторыми нежелательными неслучайными свойствами . Последовательные биты линейны, что делает их бесполезными для шифрования. Для LFSR длины n внутреннее состояние представляет собой предыдущие n выходных битов генератора. Даже если схема обратной связи хранится в секрете, она может быть определена по $2n$ выходным битам генератора с помощью высоко эффективного алгоритма Berlekamp-Massey [1082,1083]: см. раздел 16.3.

Кроме того, большие случайные числа, генерируемые с использованием идущих подряд битов этой последовательности, сильно коррелированны и для некоторых типов приложений вовсе не являются случайными . Несмотря на это LFSR часто используются для создания алгоритмов шифрования .

Программная реализация LFSR

Программные реализации LFSR медленны и быстрее работают, если они написаны на ассемблере, а не на C. Одним из решений является использование параллельно 16 LFSR (или 32, в зависимости от длины слова вашего компьютера). В этой схеме используется массив слов, размер которого равен длине LFSR, а каждый бит слова массива относится к своему LFSR. При условии, что используются одинаковые многочлены обратной связи , это может дать заметный выигрыш производительности . Вообще, лучшим способом обновлять сдвиговые регистры является умножение текущего состояния на подходящие двоичные матрицы [901].

Схему обратной связи LFSRможно модифицировать. Получающийся генератор не будет криптографически более надежным, но он все еще будет обладать максимальным периодом, и его легче реализовать программно [1272]. Вместо использования для генерации нового крайнего левого бита битов отводной последовательности выполняется XOR каждого бита отводной последовательности с выходом генератора и замена его результатом этого действия, затем результат генератора становится новым крайним левым битом (см. 11th). Иногда эту модификацию называют **конфигурацией Галуа**. На языке C это выглядит следующим образом:

```
#define mask 0x80000057

static unsigned long ShiftRegister=1;
void seed_LFSR (unsigned long seed)
{
     if (seed == 0) /* во избежание беды */
             seed = 1 ;
     ShiftRegister = seed;
}

int modified_LFSR (void)
```

```
{
    if (ShiftRegister & 0x00000001) {
           ShiftRegister = (ShiftRegister ^ mask >> 1) | 0x8000000 ;
    return 1;
    } else {
           ShiftRegister >>= 1;
           return 0;
    }
}
```

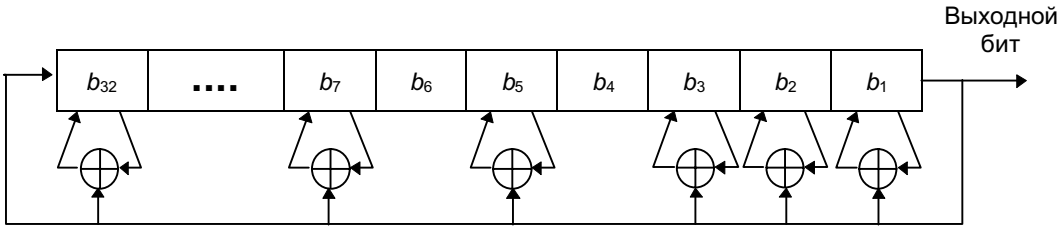


Рис. 16-5. LFSR Галуа.

Выигрыш состоит в том, что все XOR можно сделать за одну операцию. Эта схема также может быт распараллелена, а полиномы различных обратных связей могут быть различны. Такая конфигурация Галуа может дать выигрыш и при аппаратной реализации, особенно в виде СБИС. Вообще, при использовании аппаратуры, которая хорошо выполняет сдвиги применяйте конфигурацию Фиббоначи, если есть возможность использовать параллелизм, применяйте конфигурацию Галуа.

16.3 Проектирование и анализ потоковых шифров

Большинство реальных потоковых шифров основаны на LFSR. Даже в первые дни электроники построить их было несложно. Сдвиговый регистр не представляет из себя ничего большего, чем массив битов, а последовательность обратной связи - набор вентилей XOR. Даже при использовании СБИС потоковый шифр на базе LFSR обеспечивает немалую безопасность с помощью нескольких логических вентилей.

Проблема LFSR состоит в том, что их программная реализация очень неэффективна. Вам приходится избегать разреженных многочленов обратной связи - они облегчают корреляционные вскрытия [1051, 1090, 350] - а плотные многочлены обратной связи неэффективны. Выход любого потокового шифра является побитовым, для шифрования того, что можно выполнить за одну итерацию DES, необходимо выполнить 64 итерации потокового алгоритма. Действительно, программная реализация простого алгоритма LFSR, подобного описываемому ниже сжимающему генератору, не быстрее, чем DES.

Эта отрасль криптографии быстро развивается и very politically charged. Большинство разработок засекречены - множество используемых сегодня военных систем шифрования основаны на LFSR. Действительно, у большинства компьютеров Cray (Cray 1, Cray X-MP, Cray Y-MP) есть весьма любопытная инструкция, обычно называемая как "счетчик совокупности" (population count). Она подсчитывает количество единиц в регистре и может быть использована как для эффективного вычисления расстояния Хэмминга между двумя двоичными словами и для реализации векторизированной версии LFSR. Я слышал, что эта инструкция считается канонической инструкцией NSA, обязательно фигурирующей почти во всех контрактах, касающихся компьютеров.

С другой сторон было взломано удивительно большое число казавшихся сложными генераторов на базе сдвиговых регистров. И, конечно же, число таких генераторов, взломанных военными криптоаналитическими учреждениями, такими как NSA, еще больше. Иногда удивляешься тому, что самые простые из них предлагаются снова и снова.

Линейная сложность

Анализировать потоковые шифры часто проще, чем блочные. Например, важным параметром, используемым для анализа генераторов на базе LFSR, является **линейная сложность** (linear complexity), или линейный интервал. Она определяется как длина n самого короткого LFSR, который может имитировать выход генератора. Любая последовательность, генерированная конечным автоматом над конечным полем, имеет конечную линейную сложность [1006]. Линейная сложность важна, потому что с помощью простого алгоритма, называемого **алгоритмом Berlekamp-Massey**, можно определить этот LFSR, проверив только $2n$ битов потока ключей [1005]. Воссоздавая нужный LFSR, вы взламываете потоковый шифр.

Эта идея можно расширить с полей на кольца [1298] и на случаи, когда выходная последовательность рассматривается как числа в поле нечетной характеристики [842]. Дальнейшее расширение приводит к вводу понятия профиля линейной сложности, который определяет линейную сложность последовательности по мере ее удлинения [1357, 1168, 411, 1582]. Другой алгоритм вычисления линейной сложности прост только в очень сп е-

цифических условиях [597, 595, 596, 1333]. Обобщение понятия линейной сложности выполнено в [776]. Существую также понятия сферической и квадратичной сложности [844].

В любом случае помните, что высокая линейная сложность не обязательно гарантирует безопасность генератора, но низкая линейная сложность указывает на недостаточную безопасность генератора [1357, 12.49].

Корреляционная независимость

Криптографы пытаются получить высокую линейную сложность, нелинейно объединяя результаты некоторых выходных последовательностей. При этом опасность состоит в том, что одна или несколько внутренних выходных последовательностей - часто просто выходы отдельных LFSR - могут быть связаны общим ключевым потоком и вскрыты при помощи линейной алгебры. Часто такое вскрытие называют **корреляционным вскрытием** или вскрытием разделяй-и-властвуй. Томас Сигенталер (Thomas Siegenthaler) показал, что можно точно определить **корреляционную независимость**, и что существует компромисс между корреляционной независимостью и линейной сложностью [1450].

Основной идеей корреляционного вскрытия является обнаружение некоторой корреляции между выходом генератора и выходом одной из его составных частей. Тогда, наблюдая выходную последовательность, можно получить информацию об этом промежуточном выходе. Используя эту информацию и другие корреляции, можно собирать данные о других промежуточных выходах до тех пор, пока генератор не будет взломан.

Против многих генераторов потоков ключей на базе LFSR успешно использовались корреляционные вскрытия и их вариации, такие как быстрые корреляционные вскрытия, предлагающие компромисс между вычислительной сложностью и эффективностью [1451, 278, 1452, 572, 1636, 1051, 1090, 350, 633, 1054, 1089, 995]. Ряд интересных новых идей в этой области можно найти в [46, 1641].

Другие вскрытия

Существуют и другие способы вскрытия генераторов потоков ключей. Тест на **линейную корректность** (linear consistency) пытается найти некоторое подмножество ключа шифрования с помощью матричной техники [1638]. Существует и **вскрытие корректности "встречей посередине"** (meet-in-the-middle consistency attack) [39, 41]. **Алгоритм линейного синдрома** (linear syndrome algorithm) основан на возможности записать фрагмент выходной последовательности в виде линейного уравнения [1636, 1637]. Существует **вскрытие лучшим аффинным приближением** (best afflne approximation attack) [502] и **вскрытие выведенным предложением** (derived sequence attack) [42]. К потоковым шифрам можно применить также методы дифференциального [501] и линейного [631] криптоанализа.

16.4 Потоковые шифры на базе LFSR

Основной подход при проектировании генератора потока ключей на базе LFSR прост. Сначала берется один или несколько LFSR, обычно с различными длинами и различными многочленами обратной связи. (Если длины взаимно просты, а все многочлены обратной связи примитивны, то у образованного генератора будет максимальная длина.) Ключ является начальным состоянием регистров LFSR. Каждый раз, когда необходим новый бит, сдвиньте на бит регистры LFSR (это иногда называют **тактированием** (clocking)). Бит выхода представляет собой функцию, желательно нелинейную, некоторых битов регистров LFSR. Эта функция называется **комбинирующей функцией**, а генератор в целом - **комбинационным генератором**. (Если бит выхода является функцией единственного LFSR, то генератор называется **фильтрующим генератором**.) Большая часть теории подобного рода устройств разработана Селмером (Selmer) и Нилом Цирлером (Neal Zierler) [1647].

Можно ввести ряд усложнений. В некоторых генераторах для различных LFSR используется различная тактовая частота, иногда частота одного генератора зависит от выхода другого. Все это электронные версии идей шифровальных машин, появившихся до Второй мировой войны, которые называются генераторами с **управлением тактовой частотой** (clock-controlled genelators) [641]. Управление тактовой частотой может быть с прямой связью, когда выход одного LFSR управляет тактовой частотой другого LFSR, или с обратной связью, когда выход одного LFSR управляет его собственной тактовой частотой.

Хотя все эти генераторы чувствительны, по крайней мере теоретически, к вскрытиям вложением и вероятной корреляцией [634, 632], многие из них безопасны до сих пор. Дополнительную теорию сдвиговых регистров с управляемой тактовой частотой можно найти в [89].

Ян Касселлс (Ian Cassells), ранее возглавлявший кафедру чистой математики в Кембридже и работавший криптоаналитиком в Bletchly Park, сказал, что "криптография - это смесь математики и путаницы, и без путаницы математика может быть использована против вас." Он имел в виду, что в потоковых шифрах для обеспечения максимальной длины и других свойств необходимы определенные математические структуры, такие как LFSR, но, чтобы помешать кому-то получить содержание регистра и вскрыть алгоритм, необходимо внести некоторый сложный нелинейный беспорядок. Этот совет справедлив и для блочных алгоритмов.

Поэтому не стоит серьезно увлекаться генераторами потока ключей на базе LFSR, описания которых появились в литературе. Я не знаю, используется ли хоть один из них в реальных криптографических продуктах. Большей частью они представляют лишь теоретический интерес. Некоторые были взломаны, некоторые могли остаться безопасными.

Так как шифры на базе LFSR обычно реализуются аппаратно, на рисунках используются символы электронной логики. В тексте, $\oplus$ означает XOR, $\wedge$ - AND, $\vee$ - OR, и $\neg$ - NOT.

Генератор Геффа

В этом генераторе потока ключей используются три LFSR, объединенные нелинейным образом (см. 10th) [606]. Два LFSR являются входами мультиплексора, а третий LFSR управляет выходом мультиплексора. Если a_1, a_2 и a_3 - выходы трех LFSR, выход генератора Геффа (Geffe) можно описать как:

$b = (a_1 \wedge a_2) \oplus((\neg a_1) \wedge a_3)$

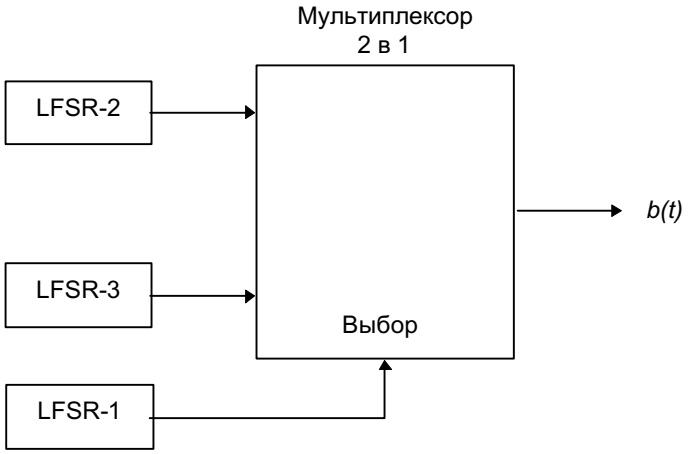


Рис. 16-6. Генератор Геффа.

Если длины LFSR равны n_1, n_2 и n_3, соответственно, то линейная сложность генератора равна

$(n_1 + 1)\, n_2 + n_1 n_3$,

Период генератора равен наименьшему общему делителю периодов трех генераторов. При условии, что степени трех примитивных многочленов обратной связи взаимно просты, период этого генератора будет равен произведению периодов трех LFSR.

Хотя этот генератор неплохо выглядит на бумаге, он криптографически слаб и не может устоять против корреляционного вскрытия [829, 1638]. В 75 процентах времени выход генератора равен выходу LFSR-2. Поэтому, если известны отводные последовательности обратной связи, можно догадаться о начальном значении LFSR-2 и сгенерировать выходную последовательность этого регистра. Тогда можно подсчитать, сколько раз выход LFSR совпадает с выходом генератора. Если начальное значение определено неверно, две последовательности будут согласовываться в 50 процентах времени, а если правильно, то в 75 процентах времени.

Аналогично, выход генератора равен выходу LFSR в 75 процентах времени. С такими корреляциями генератор потока ключей может быть легко взломан. Например, если примитивные многочлены состоят только из трех членов, и длина самого большого LFSR равна n, для восстановления внутренних состояний всех трех LFSR нужен фрагмент выходной последовательности длиной $37n$ битов [1639].

Обобщенный генератор Геффа

Вместо выбора между двумя LFSR в этой схеме выбирается один из k LFSR, где k является степенью 2. Всего используется $k + 1$ LFSR (см. 9th). Тактовая частота LFSR-l должна быть в $\log_2 k$ раз выше, чем у остальных k LFSR.

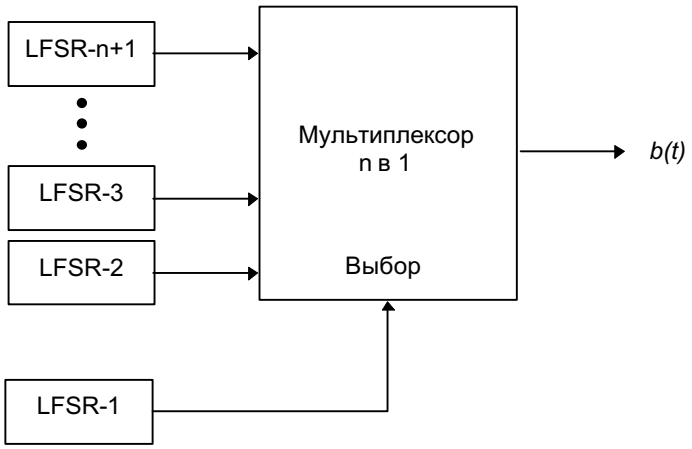


Рис. 16-7. Обобщенный генератор Геффа.

Несмотря на то, что эта схема сложнее генератора Геффа, для взлома можно использовать то же корреляционное вскрытие. Не рекомендую этот генератор.

Генератор Дженнингса

В этой схеме мультиплексор используется для объединения двух LFSR [778, 779, 780]. Мультиплексор, управляемый LFSR-1, выбирает 1 бит LFSR-2 в качестве очередного выходного бита. Кроме того, используется функция, которая отображает выход LFSR-2 на вход мультиплексора (см. 8th).

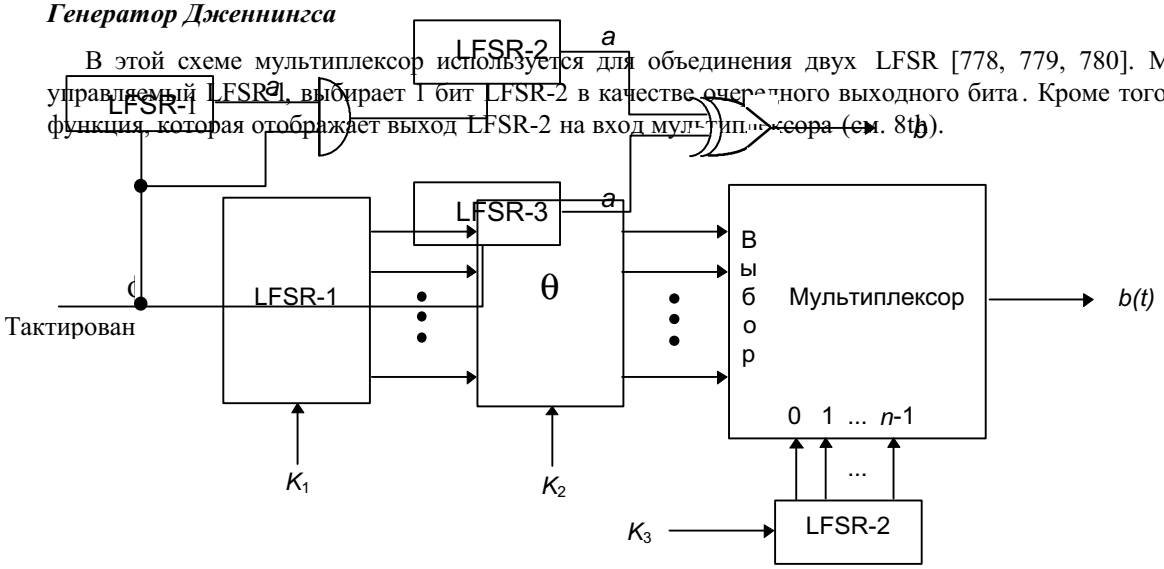


Рис. 16-8. Генератор Дженнингса.

Ключом является начальное состояние двух LFSR и функции отображения. Хотя у этого генератора замечательные статистические свойства, он пал перед выполненным Россом Андерсоном (Ross Anderson) вскрытием корректности встречей посередине [39] и вскрытием линейной корректности [1638,442]. Не используйте этот генератор.

Генератор "стоп-пошел" (Stop-and-Go) Both-Piper

Этот генератор, показанный на 7th, использует выход одного LFSR для управления тактовой частотой другого LFSR [151]. Тактовый вход LFSR-2 управляется выходом LFSR-l, так что LFSR-2 может изменять свое состояние в момент времени *t* только, если выход LFSR-l в момент времени *t* - 1 был равен 1.

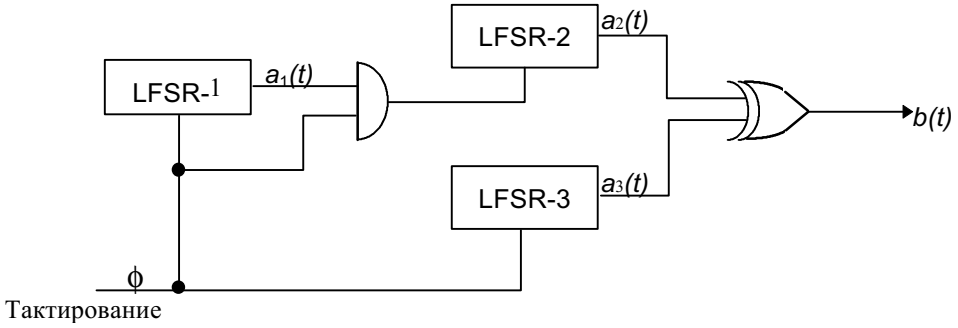


Рис. 16-9. Генератор "стоп-пошел" Beth-Piper.

Никому не удалось привести для общего случая достоверные данные о линейной сложности этого генератора. Однако он не устоял перед корреляционным вскрытием [1639].

Чередующийся генератор "стоп-пошел"

В этом генераторе используются три LFSR различной длины. LFSR-2 тактируется, когда выход LFSR-l равен 1, LFSR-3 тактируется, когда выход LFSR-l равен 0. Выходом генератора является XOR LFSR-2 и LFSR-3 (см. Рис. 16.10) [673].

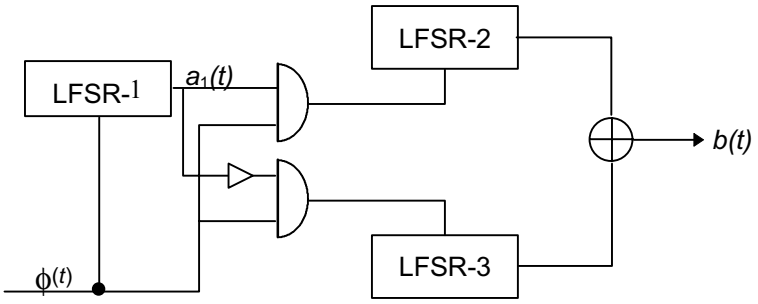


Рис. 16-10. Чередующийся генератор "стоп-пошел"

У этого генератора большой период и большая линейная сложность. Авторы показали способ корреляционного вскрытия LFSR-1, но это не сильно ослабляет генератор. Были предложены и другие генераторы такого типа [1534, 1574, 1477].

Двусторонний генератор "стоп-пошел"

В этом генераторе используется два LFSR с одинаковой длиной n (см. Рис. 16.11) [1638]. Выходом генератора является XOR выходов каждого LFSR. Если выход LFSR-l в момент времени t-1 равен 0, а в момент времени t-2 - 1, то LFSR-2 не тактируется в момент времени t. Наоборот, если выход LFSR-2 в момент времени t-1 равен 0, а в момент времени t-2 - 1, и если LFSR-2 тактируется в момент времени t, то LFSR-l не тактируется в момент времени t.

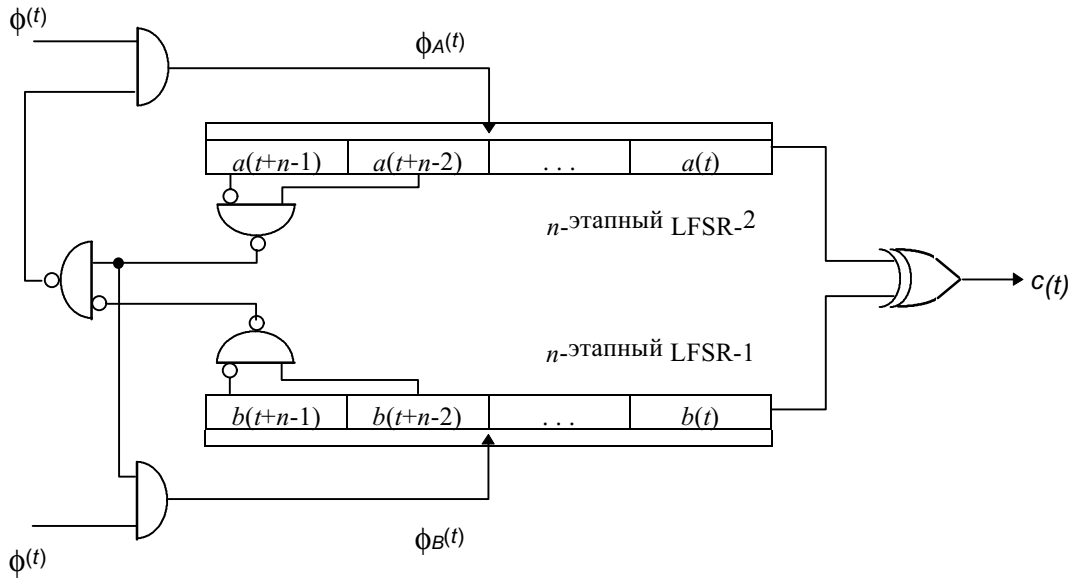


Рис. 16-11. Двусторонний генератор "стоп-пошел".

Линейная сложность такой системы примерно равна ее периоду. Согласно [1638], "в такой системе не очевидная избыточность ключа не наблюдается".

Пороговый генератор

Этот генератор пытается обойти проблемы безопасности, характерные для предыдущих генераторов, с помощью переменного числа LFSR [277]. По теории при использовании большего количества LFSR вскрыть шифр сложнее.

Этот генератор показан на 4-й. Возьмите выход большого числа LFSR (используя нечетное число регистров). Для получения максимального периода убедитесь, что длины всех LFSR взаимно просты, а многочлены обратной связи - примитивны.. Если более половины выходных битов LFSR - 1, то выходом генератора является 1. Если более половины выходных битов LFSR - 0, то выходом генератора является 0.

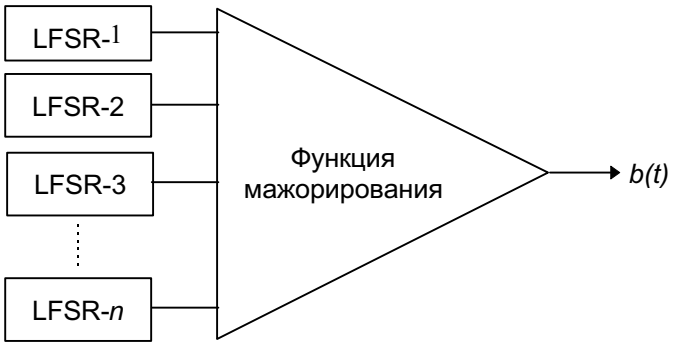


Рис. 16-12. Пороговый генератор.

Для трех LFSR выход генератора можно представить как:

$b = (a_1 \wedge a_2) \oplus (a_1 \wedge a_3) \oplus (a_2 \wedge a_3)$

Это очень похоже на генератор Геффа за исключением того, что пороговый генератор обладает большей линейной сложностью

$n_1n_2 + n_1n_3 + n_2n_3$

где n_1, n_2 и n_3 - длины первого, второго и третьего LFSR.

Этот генератор не слишком хорош. Каждый выходной бит дает некоторую информацию о состоянии LFSR - точнее 0.189 бита - и генератор в целом не может устоять перед корреляционным вскрытием. Я не советую использовать такой генератор.

Самопрореживающие (Self-Decimated) генераторы

Самопрореживающими называются генераторы, которые управляют собственной тактовой частотой . Было предложено два типа таких генераторов, один Рэйнером Рюппелом (Ranier Rueppel) (см. 3-й) [1359] другой Биллом Чамберсом (Bill Chambers) и Дитером Коллманом (Dieter Collmann) [308] (см. 2nd). В генераторе Рюппела если выход LFSR равен 0, LFSR тактируется *d* раз. Если выход LFSR равен 0, LFSR тактируется *k* раз. Генератор Чамберса и Коллмана сложнее, но идея остается той же . К сожалению оба генератора не безопасны [1639], хотя был предложен ряд модификаций, которые могут исправить встречающиеся проблемы [1362.].

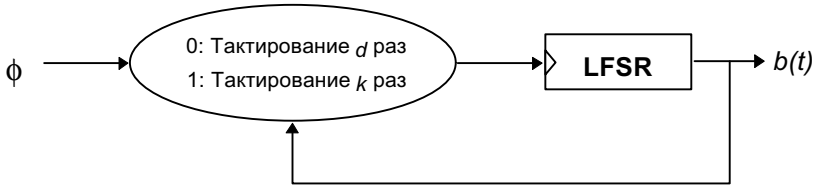


Рис. 16-13. Самопрореживающий генератор Рюппела.

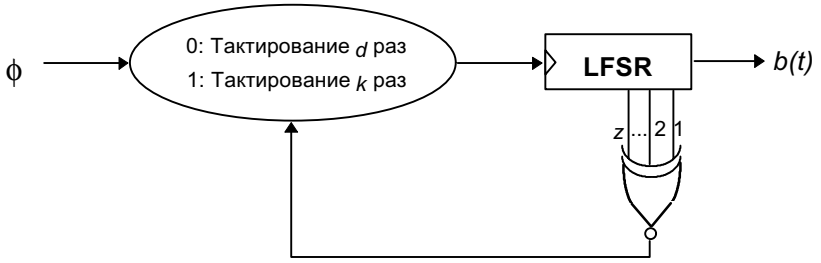


Рис. 16-14. Самопрореживающий генератор Чамберса и Голлмана.

Многоскоростной генератор с внутренним произведением (inner-product)

Этот генератор, предложенный Массеем (Massey) и Рюппелом [1014], использует два LFSR с разными тактовыми частотами (см. 1st). Тактовая частота LFSR-2 в *d* раз больше, чем у LFSR-l. Отдельные биты этих LFSR объединяются операцией AND, а затем для получения выходного бита генератора они объединяются посредс т-вом XOR.

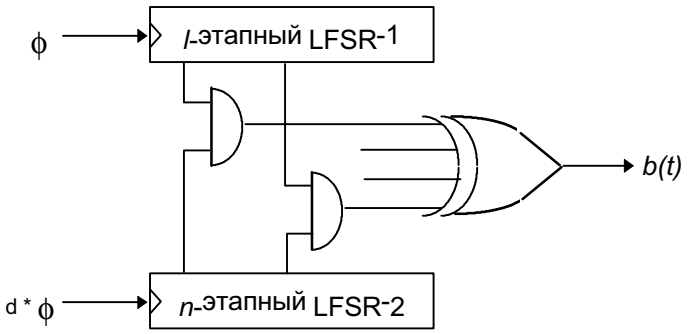


Рис. 16-15. Многоскоростной генератор с внутренним произведением.

Хотя этот генератор обладает высокой линейной сложностью и великолепными статистическими характер истиками, он все же не может устоять перед вскрытием линейной согласованности [1639]. Если n_1 - длина LFSR-l, n_2 - длина LFSR-2, а *d* - отношение тактовых частот, то внутреннее состояние генератора может быть получено по выходной последовательн ости длиной

$n_2 + n_2 + \log_2 d$

Суммирующий генератор

Еще одно предложение Рэйнер Рюппела , этот генератор суммирует выходы двух LFSR (с переносом) [1358, 1357]. Это в высокой степени нелинейная операция . В конце 80-х этот генератор был лидером в отношении безопасности, но он пал перед корреляционным вскрытием [1053, 1054, 1091]. Кроме того, было показано, что этот генератор является частным случаем обратной связи, использующей сдвиговый регистр с переносом (см.

раздел 17.4), и может быть взломан [844].

DNRSG

Это означает "динамический генератор случайной последовательности" ("dynamic random-sequence generator") [1117]. Идея состоит в том, чтобы взять два различных фильтруемых генератора - пороговых, суммирующих, и т.п. - использующих один набор LFSR, а управляемых другим LFSR.

Сначала тактируются все LFSR. Если выходом LFSR-0 является 1, то вычисляется выход первого фильтрующего генератора. Если выходом LFSR-0 является 0, то вычисляется выход второго фильтрующего генератора. Окончательным результатом является XOR выходов первого и второго генераторов.

Каскад Голлманна

Каскад Голлманна (см. 0-й), описанный в [636, 309], представляет собой усиленную версию генератора "стоп-пошел". Он состоит из последовательности LFSR, тактирование каждого из которых управляется предыдущим LFSR. Если выходом LFSR-l в момент времени *t* является 1, то тактируется LFSR-2. Если выходом LFSR-2 в момент времени *t* является 1, то тактируется LFSR-3, и так далее. Выход последнего LFSR и является выходом генератора. Если длина всех LFSR одинакова и равна *n*, линейная сложность системы из *k* LFSR равна

$n(2^n - 1)^{k-1}$

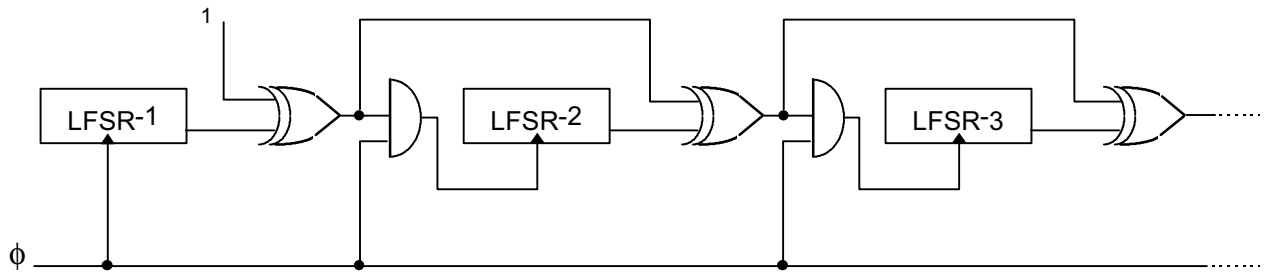


Рис. 16-16. Каскад Голлманна.

Это дерзкая идея: концептуально они очень просты и могут быть использованы для генерации последовательностей с огромными периодами, огромными линейными сложностями и хорошими статистическими свойствами. Они чувствительны к вскрытию, называемому **запиранием** (lock-in) [640] и представляющему метод, с помощью которого сначала криптоаналитик восстанавливает вход последнего сдвигового регистра в каскаде, а затем взламывает весь каскад, регистр за регистром. В некоторых случаях это представляет собой серьезную проблему и уменьшает эффективную длину ключа алгоритма, но для минимизации возможности такого вскрытия можно предпринять ряд определенных мер.

Дальнейший анализ показал, что с ростом *k* последовательность приближается к случайной [637, 638, 642, 639]. На основании недавних вскрытий коротких каскадов Голлманна [1063], я советую использовать *k* не меньше 15. Лучше использовать больше коротких LFSR, чем меньше длинных LFSR.

Прореживаемый генератор

Прореживаемый (shrinking) генератор [378] использует другую форму управления тактированием. Возьмем два LFSR: LFSR-l и LFSR -2. Подадим тактовый импульс на оба регистра. Если выходом LFSR-l является 1, то выходом генератора является выход LFSR-2. Если выход LFSR-l равен 0, оба бита сбрасываются, LFSR тактируются заново и все повторяется.

Идея проста, достаточно эффективна и кажется безопасной. Если многочлены обратной связи прорежены, генератор чувствителен к вскрытию, но других проблем обнаружено не было. Хотя этот тип генератора достаточно нов. Одна из проблем реализации состоит в том, что скорость выдачи результата не постоянна, если LFSR-l генерирует длинную последовательность нулей, то на выходе генератора ничего нет. Для решения этой проблемы авторы предлагают использовать буферизацию [378]. Практическая реализация прореживаемого генератора рассматривается в [901].

Самопрореживаемый генератор

Самопрореживаемый (self-shrinking) генератор [1050] является вариантом прореживаемого генератора. Вместо двух LFSR используется пара битов одного LFSR. Протактируйте LFSR дважды. Если первым битом пары будет 1, то второй бит будет выходом генератора. Если первый бит - 0, сбросьте оба бита и попробуйте снова. Хотя для самопрореживаемого генератора нужно примерно в два раза меньше памяти, чем для прореживаемо-

го, он работает в два раза медленнее.

Хотя самопрореживаемый генератор также кажется безопасным, он может вести себя непредсказуемым образом и обладать неизвестными свойствами. Это очень новый генератор, дайте ему немного времени.

16.5 A5

A5 - это потоковый шифр, используемый для шифрования GSM (Group Special Mobile). Это европейский стандарт для цифровых сотовых мобильных телефонов. Он используется для шифрования канала "телефон-базовая станция". Оставшаяся часть канала не шифруется, телефонная компания может легко сделать что-нибудь с вашими разговорами.

Вокруг этого протокола ведутся странные политические игры. Первоначально предполагалось, что криптография GSM позволит запретить экспорт телефонов в некоторые страны. Теперь ряд чиновников обсуждает, не повредит ли A5 экспортным продажам несмотря на то, что он так слаб, что вряд ли сможет служить препятствием. По слухам в середине 80-х различные секретные службы НАТО сцепились по вопросу, должно ли шифрование GSM быть сильным или слабым. Немцам была нужна сильная криптография, так как рядом с ними находился Советский Союз. Взяла верх другая точка зрения, и A5 представляет собой французскую разработку.

Большинство деталей нам известно. Британская телефонная компания передала всю документацию Брэдфордскому университету (Bradford University), не заставив подписать соглашение о неразглашении. Информация где-то просочилась и наконец была опубликована в Internet. A5 описывается в [1622], также в конце этой книги приведен код этого протокола.

A5 состоит из трех LFSR длиной 19, 22 и 23, все многочлены обратной связи - прорежены. Выходом является XOR трех LFSR. В A5 используется изменяемое управление тактированием. Каждый регистр тактируется в зависимости от своего среднего бита, затем выполняется XOR с обратной пороговой функцией средних битов всех трех регистров. Обычно на каждом этапе тактируется два LFSR.

Существует тривиальное вскрытие, требующее 2^{40} шифрований: предположите содержание первых двух LFSR и попытайтесь определить третий LFSR по потоку ключей. (Действительно ли такой способ вскрытия возможен, остается под вопросом, который скоро будет разрешен разрабатываемой машиной для аппаратного поиска ключей [45].)

Тем не менее, становится ясно, что идеи, лежащие в основе A5, неплохи. Алгоритм очень эффективен. Он удовлетворяет всем известным статистическим тестам, единственной его слабостью является то, что его регистры слишком коротки, чтобы предотвратить поиск ключа перебором. Варианты A5 с более длинными сдвиговыми регистрами и более плотными многочленами обратной связи должны быть безопасны.

16.6 Hughes XPD/KPD

Этот алгоритм был предложен Hughes Aircraft Corp. Эта фирма встроила его в армейские тактические рации и оборудование поиска направления для продажи за границу. Алгоритм был разработан в 1986 году и получил название XPD, сокращение от Exportable Protection Device - Экспортируемое устройство защиты. Позднее он был переименован в KPD - Устройство кинетической защиты - и рассекречен [1037, 1036].

Алгоритм использует 61-битовый LFSR. Существует 2^{10} различных примитивных многочлена обратной связи, одобренных NSA. Ключ выбирает один из этих многочленов (хранящихся где-то в ПЗУ), а также начальное состояние LFSR.

В алгоритме восемь различных нелинейных фильтров, каждый из которых использует шесть отводов LFSR, выдавая один бит. Объединяясь, эти биты образуют байт, который и применяется для шифрования или дешифрования потока данных.

Этот алгоритм выглядит очень привлекательно, но у меня есть определенные сомнения. NSA разрешило его экспорт, следовательно должен быть способ вскрытия порядка, не большего чем 2^{40}. Но какой?

16.7 Nanoteq

Nanoteq - это южноафриканская электронная компания. Именно этот алгоритм используется южноафриканской полицией при шифровании передачи факсов, а возможно и для прочих нужд.

Более или менее этот алгоритм описан в [902, 903]. Он использует 127-битовый LFSR с фиксированным многочленом обратной связи, ключ представляет собой начальное состояние регистра. При помощи 25 элементарных ячеек 127 битов регистра превращаются в один бит потока ключей. У каждой ячейки 5 входов и один выход:

$$f(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + (x_1 + x_3)(x_2 + x_4 + x_5) + (x_1 + x_4)(x_2 + x_3) + x_5$$

Каждый выход функции подвергается операции XOR с некоторым битом ключа. Кроме того, существует секретная перестановка, зависящая от конкретной реализации и не описанная в статьях подробно. Этот алгоритм доступен только в аппаратном виде.

Безопасен ли он? Я не уверен. Ряд интересных факсов, передаваемых между полицейскими участками, ин огда появлялся в либеральных газетах. Это вполне могло быть результатом американской, английской или сове тской разведывательной деятельности. Росс Андерсон (Ross Anderson) предпринял ряд первых шагов, крипто анализируя этот алгоритм в [46], я думаю, что скоро появятся новые результаты.

16.8 Rambutan

Rambutan - это английский алгоритм, разработанный Communications Electronics Security Croup (Группа по безопасности электронных коммуникаций, одно из объединений, использованное CCHQ). Он продается только в виде аппаратного модуля и одобрен для защиты документов вплоть до грифа "Конфиденциально". Сам алгоритм засекречен, и микросхема не предназначена для широкой коммерческой продажи.

Rambutan использует 112-битовый ключ (плюс биты четности) и может работать трех режимах: ECB, CBC, и 8-битовый CFB. Это сильный аргумент в пользу того, что этот алгоритм - блочный, но слухи утверждают иное. Предположительно это потоковый шифр с LFSR. У него пять приблизительно 80-битовых сдвиговых р егистров различной длины. Полиномы обратной связи значительно прорежены, в каждом из них всего лишь 10 отводов. Каждый сдвиговый регистр обеспечивает четыре входа для очень большой и сложной нелинейной функции, которая и выдает единственный бит.

Почему Rambutan? Возможно из-за фрукта, который колючий и неприступный снаружи, но мягкий и не жный внутри. Но с другой стороны никакой причины может и не быть.

16.9 Аддитивные генераторы

Аддитивные генераторы (иногда называемые запаздывающими генераторами Фиббоначи) очень эффективны, так как их результатом являются случайные слова, а не случайные биты [863]. Сами по себе они не безопасны, но их можно использовать в качестве составных блоков для безопасных генераторов.

Начальное состояние генератора представляет собой массив n-битовых слов: 8-битовых слов, 16-битовых слов, 32-битовых слов, и т.д.: X_1, X_2, X_3, ..., X_m. Это первоначальное состояние и является ключом. i-ое слово генератора получается как

$$X_i = (X_{i-a} + X_{i-b} + X_{i-c} + + X_{i-m}) \bmod 2^n$$

При правильном выборе коэффициентов a, b, c, . . . , m период этого генератора не меньше 2^n-1. Одним из требований к коэффициентам является то, что младший значащий бит образует LFSR максимальной длины.

Например, (55,24,0) - это примитивный многочлен mod 2 из 14-й. Это означает, что длина следующего адд итивного генератора максимальна.

$$X_i = (X_{i-55} + X_{i-24}) \bmod 2^n$$

Это работает, так как у примитивного многочлена три коэффициента. Если бы их было больше, для получ ения максимальной длины потребовались бы дополнительные условия. Подробности можно найти в [249].

Fish

Fish - это аддитивный генератор, основанный на методах, используемых в прореживаемом генераторе [190]. Он выдает поток 32-битовых слов, которые могут быть использованы (с помощью XOR) с потоком открытого текста для получения шифротекста или с потоком шифротекста для получения открытого текста. Название алгоритма представляет собой сокращение от Fibonacci shrinking generator - прореживаемый генератор Фиббоначи.

Во первых используйте два следующих аддитивных генератора. Ключом является начальные состояния этих генераторов.

$$A_i = (A_{i-55} + A_{i-24}) \bmod 2^{32}$$

$$B_i = (B_{i-52} + B_{i-19}) \bmod 2^{32}$$

Эти последовательности прореживаются попарно в зависимости от младшего значащего бита B_i: если его значение равно 1, то пара используется, если 0 - игнорируется. C_j - это последовательность используемых слов Ai, а D_j - это последовательность используемых слов B_i. Для генерации двух 32-битовых слов-результатов K_{2j} и K_{2j+1} эти слова используются пар ами - C_{2j}, C_{2j+1}, D_{2j}, D_{2j+1}.

$$E_{2j} = C_{2j} \oplus (D_{2j}, \wedge D_{2j+1})$$

$F_{2j} = D_{2j+1} \wedge (E_j, \wedge C_{2j+1})$

$K_{2j} = E_{2j} \oplus F_{2j}$

$K_{2j+1} = C_{2j+1} \oplus F_{2j}$

Этот алгоритм быстр. на процессоре i486/33 реализация Fish на языке C шифрует данные со скоростью 15-Мбит/с. К сожалению он также не безопасен, порядок вскрытия составляет около 2^{40} [45].

Pike

Pike - это обедненная и урезанная версия Fish, предложенная Россом Андерсоном, тем, кто взломал Fish [45]. Он использует три аддитивных генератора. Например:

$A_i = (A_{i-55} + A_{i-24}) \bmod 2^{32}$

$B_i = (B_{i-57} + B_{i-7}) \bmod 2^{32}$

$C_i = (C_{i-58} + C_{i-19}) \bmod 2^{32}$

Для генерации слова потока ключей взгляните на биты переноса при сложении . Если все три одинаковы (все нули или все единицы), то тактируются все три генератора. Если нет, то тактируются только два совпадающих генератора. Сохраните биты переноса для следующего раза. Окончательным выходом является XOR выходов трех генераторов.

Pike быстрее Fish, так как в среднем для получения результата нужно 2.75 действия, а не 3. Он также слишком нов, чтобы ему доверять, но выглядит очень неплохо.

Mush

Mush представляет собой взаимно прореживающий генератор. Его работу объяснить легко [1590]. Возьмем два аддитивных генератора: A и B. Если бит переноса A установлен, тактируется B. Если бит переноса B установлен, тактируется A. Тактируем A и при переполнении устанавливаем бит переноса. Тактируем B и при переполнении устанавливаем бит переноса. Окончательным выходом является XOR выходов A и B. Проще всего использовать те же генераторы, что и в Fish:

$A_i = (A_{i-55} + A_{i-24}) \bmod 2^{32}$

$B_i = (B_{i-52} + B_{i-19}) \bmod 2^{32}$

В среднем для генерации одного выходного слова нужно три итерации генератора . И если коэффициенты аддитивного генератора выбраны правильно и являются взаимно простыми, длина выходной последовательности будет максимальна. Мне неизвестно об успешных вскрытиях, но не забывайте, что этот алгоритм очень нов .

16.10 Gifford

Дэвид Джиффорд (David Gifford) изобрел потоковый шифр и использовал его для шифрования сводок новостей в районе Бостона с 1984 по 1988 год [608, 607, 609]. Алгоритм использует единственный 8-байтовый регистр: $b_0, b_1, \ldots, b_7$. Ключом является начальное состояние регистра. Алгоритм работает в режиме OFB, открытый текст абсолютно не влияет на работу алгоритма . (См. -1-й).

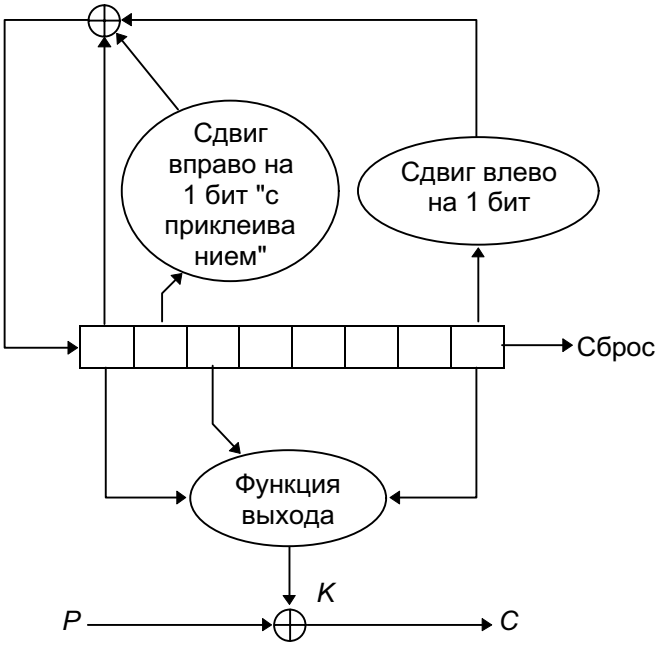


Рис. 16-17. Gifford.

Для генерации байта ключа k_i объединим b_0 и b_1, а также объединим b_4 и b_7. Перемножим полученные числа, получая 32-битовое число. Третьим слева байтом и будет k_i.

Для обновления регистра возьмем b_1 и сдвинем вправо "с приклеиванием" на 1 бит следующим образом: крайний левый бит одновременно и сдвигается, и остается на месте. Возьмем b_7 и сдвинем его на один бит влево, в крайней правой позиции должен появиться 0. Выполним XOR измененного b_1, измененного b_7 и b_0. Сдвинем первоначальный байт регистра на 1 бит вправо и поместим этот байт в крайнюю левую позицию.

В течение всего времени использования этот алгоритм оставался безопасным, но он был взломан в 1994 году [287]. Оказалось, что многочлен обратной связи не был примитивным и, таким образом, мог быть вскрыт.

16.11 Алгоритм M

Это название дано Кнутом [863]. Алгоритм представляет собой способ объединить несколько псевдослучайных потоков, увеличивая их безопасность. Выход одного генератора используется для выбора отстающего выхода другого генератора [996, 1003]. На языке C:

```
#define ARR_SIZE (8192) /* например - чем больше, тем лучше */
static unsigned char delay[ ARRSIZE ] ;
unsigned char prngA( void )
long prngB( void ) ;
void init_algM( void ) {
     long i ;
     for ( i = 0 ; i < ARR_SIZE ; i++ )
            delay[i] = prngA() ;
} /* lnlt_algM */
unsigned char alglM( void ) {
     long j,v ;
     j = prngB() % ARR_SIZE ; /* получить индекс delay[]*/
     v = delay[j] ; /* получить возвращаемое значение */
     delay[j] = prngA() ; /* заменить его */
     return ( v ) ;
     } /* algM */
```

Смысл состоит в том, что если prngA - действительно случайно, невозможно ничего узнать о prngB (и, следовательно, невозможно выполнить криптоанализ). Если prngA имеет такой вид, что его криптоанализ может быть выполнен только, если его выход доступен в свою очередь (т.е., только если сначала был выполнен кри п-

тоанализ prngB), а в противном случае оно по сути действительно случайно, то эта комбинация должна быть безопасной.

16.12 PKZIP

Алгоритм шифрования, встроенный в программу сжатия данных PKZIP, был разработан Роджером Щлафлы (Roger Schlafly). Это потоковый шифр, шифрующий данные побайтно. По крайней мере этот алгоритм используется в версии 2.04g. Я не могу ничего сказать о более поздних версиях, но если не было сделано никаких заявлений об обратном, можно считать с большой вероятностью, что алгоритм не изменился. Алгоритм использует три 32-битовых переменных, инициализированных следующим образом :

K_0 = 305419896

K_1 = 591751049

K_2 = 878082192

Используется 8-битовый ключ K_3, полученный из K_2. Вот этот алгоритм (в стандартной нотации C):

$C_i = P_i \wedge K_3$

K_0= crc32 (K_0, P_i)

$K_1 = K_1 + (K_0 \,\&\, 0x000000ff)$

$K_1 = K_1 * 134775813 + 1$

K_2 = crc32 (K_2, $K_1 >> 24$)

$K_3 = ((K_2 \mid 2) * ((K_2 \mid 2) \wedge 1)) >> 8$

Функция crc32 берет свое предыдущее значение и байт, выполняет их XOR и вычисляет следующее значение с помощью многочлена CRC, определенного 0xedb88320. На практике 256-элементная таблица может быть рассчитана заранее, и вычисление crc32 превращается в:

crc32 (a, b) = ($a >> 8$) ^ table [(a & 0xff) $\oplus$ b]

Таблица рассчитывается в соответствии с первоначальным определением crc32:

table [i] = crc32 (i, 0)

Для шифрования потока открытого текста сначала для обновления ключей зациклим байты ключа в алгоритме шифрования. Полученный шифротекст на этом этапе игнорируется. Затем побайтно зашифруем открытый текст. Открытому тексту предшествуют двенадцать случайных байтов, но это на самом деле неважно. Дешифрирование похоже на шифрование за исключением того, что во втором действии алгоритма вместо P_i используется C_i.

Безопасность PKZIP

К сожалению она не слишком велика. Для вскрытия нужно от 40 до 2000 байтов известного открытого текста, временная сложность вскрытия составит около 2^{27} [166]. На вашем персональном компьютере это можно сделать за несколько часов. Если в сжатом файле используются какие-нибудь стандартные заголовки, получение известного открытого текста не представляет собой проблемы. Не используйте встроенное в PKZIP шифрование.

Глава 17
Другие потоковые шифры и генераторы настоящих случайных последовательностей

17.1 RC4

RC4 - это потоковый шифр с переменным размером ключа, разработанный в 1987 году Роном Ривестом для RSA Data Security, Inc. В течение семи лет он находился в частной собственности, и подробное описание алгоритма предоставлялось только после подписания соглашения о неразглашении.

В сентябре 1994 кто-то анонимно опубликовал исходный код в списке рассылки "Киберпанки" (Cypherpunks). Он быстро распространился в телеконференнции Usenet sci.crypt и через Internet по различным ftp-серверам во всем мире. Обладатели легальных копий RC4 достоверность этого кода. RSA Data Security, Inc. попыталась загнать джинна обратно в бутылку, утверждая, что несмотря на опубликование алгоритм остается торговым секретом, было слишком поздно. С тех пор алгоритм обсуждался и изучался в Usenet, распространялся на конференциях и служил в качестве учебного пособия на курсах по криптографии.

Описывать RC4 просто. Алгоритм работает в режиме OFB: поток ключей не зависит от открытого текста. Используется S-блок размером 8*8: S_0, S_1, . . . , S_{255}. Элементы представляют собой перестановку чисел от 0 до 255, а перестановка является функцией ключа переменной длины. В алгоритме применяются два счетчика, i и j, с нулевыми начальными значениями.

Для генерации случайного байта выполняется следующее:

$i = (i + 1) \bmod 256$

$j = (j + S_i) \bmod 256$

поменять местами S_i и Sj

$t = (S_i + S_j) \bmod 256$

$K = S_t$

Байт K используется в операции XOR с открытым текстом для получения шифротекста или в операции XOR с шифротекстом для получения открытого текста. Шифрование выполняется примерно в 10 раз быстрее, чем DES.

Также несложна и инициализация S-блока. Сначала заполним его линейно: $S_0 = 0$, $S_1 = 1$, . . . , $S_{255} = 255$. Затем заполним ключом другой 256-байтовый массив, при необходимости для заполнения всего массива повторяя ключ: K_0, K_1, . . . , K_{255}. Установим значение индекса j равным 0. Затем:

for $i = 0$ to 255:

$j = (j + S_i + K_i) \bmod 256$

поменять местами S_i и S_j

И это все. RSADSI утверждает, что алгоритм устойчив к дифференциальному и линейному криптоанализу, что, по-видимому, в нем нет никаких коротких циклов, и что он в высокой степени нелинеен. (Опубликованных криптоаналических результатов нет. RC4 может находиться в примерно 2^{1700} ($256! * 256^2$) возможных состояний: невероятное число.) S-блок медленно изменяется при использовании: i обеспечивает изменение каждого элемента, а j - что элементы изменяются случайным образом. Алгоритм настолько несложен, что большинство программистов могут закодировать его просто по памяти.

Эту идею можно обобщить на S-блоки и слова больших размеров. Выше была описана 8-битовая версия RC4. Нет причин, по которым нельзя бы было определить 16-битовый RC4 с 16*16 S-блоком (100 K памяти) и 16-битовым словом. Начальная итерация займет намного больше времени - для сохранения приведенной схемы нужно заполнить 65536-элементный массив - но получившийся алгоритм должен быть быстрее.

RC4 с ключом длиной не более 40 битов обладает специальным экспортным статусом (см. раздел 13.8). Этот специальный статус никак не влияет на безопасность алгоритма, хотя в течение многих лет RSA Data Security, Inc. намекало на обратное. Название алгоритма является торговой маркой, поэтому каждый, кто напишет собственный код, должен назвать его как-то иначе. Различные внутренние документы RSA Data Security, Inc. до сих пор не были опубликованы [1320, 1337].

Итак, какова же ситуация вокруг алгоритма RC4? Он больше не является торговым секретом, поэтому кто угодно имеет возможность воспользоваться им. Однако RSA Data Security, Inc. почти наверняка возбудит дело против каждого, кто применит нелицензированный RC4 в коммерческом продукте. Возможно им и не удастся

выиграть процесс, но почти наверняка для другой компании дешевле купить лицензию, чем судиться .

RC4 входит в десятки коммерческих продуктов, включая Lotus Notes, AOCE компании Apple Computer и and Oracle Secure SQL. Этот алгоритм также является частью спецификации Сотовой цифровой пакетной передачи данных (Cellular Digital Packet Data) [37].

17.2 SEAL

SEAL - это программно эффективный потоковый шифр, разработанный в IBM Филом Рогэвэем (Phil Rogaway) и Доном Копперсмитом (Don Coppersmith) [1340]. Алгоритм оптимизирован для 32-битовых процессоров: Для нормальной работы ему нужно восемь 32-битовых регистров и кэш-память на несколько килобайт . Чтобы избежать влияния использования медленных операций SEAL выполняет ряд предварительных действий с ключом, сохраняя результаты в нескольких таблицах. Эти таблицы используются для ускорения шифрования и дешифрирования.

Семейство псевдо случайных функций

Особенностью SEAL является то, что он в действительности является не традиционным потоковым шифром, а представляет собой **семейство псевдослучайных функций**. При 160-битовом ключе k и 32-битовом n SEAL растягивает n в L-битовую строку $k(n)$. L может принимать любое значение, меньшее 64 Кбайт. SEAL, по видимому, использует следующее свойство: если k выбирается случайным образом, то $k(n)$ должно быть вычислительно неотличимо от случайной L-битовой функции n.

Практический эффект того, что SEAL является семейством псевдослучайных функций, состоит в том, что он удобен в ряде приложений, где неприменимы традиционные потоковые шифры . Используя большинство потоковых шифров, вы создаете однонаправленную последовательность битов: единственным способом определить i-ый бит, зная ключ и позицию i, является генерирование всех битов вплоть до i-ого. Отличие семейства псевдослучайных функций состоит в том, что вы можете легко получить доступ к любой позиции потока ключей . Это очень полезно.

Представим себе, что вам нужно "закрыть" жесткий диск. Вы хотите зашифровать каждый 512-байтовый сектор. Используя семейство псевдослучайных функций, подобное SEAL, содержимое сектора n можно зашифровать, выполнив его XOR с $k(n)$. Это то же самое, как если бы была выполнена операция XOR всего диска с длинной псевдослучайной функцией, и любая часть этой длинной строки может быть независимо вычислена без всяких проблем.

Семейство псевдослучайных функций также упрощает проблему синхронизации, встречающуюся в стандартных потоковых шифрах. Предположим, что вы посылаете шифрованные сообщения по каналу, в котором данные иногда теряются. С помощью семейства псевдослучайных функций можно зашифровать ключом k n-ое передаваемое сообщение, x_n, выполнив XOR x_n and $k(n)$. Получателю не нужно хранить состояние шифра для восстановления x_n, ему не приходится беспокоиться и о потерянных сообщениях, влияющих на процесс дешифрирования.

Описание SEAL

Внутренний цикл SEAL показан на 16th. Алгоритм управляется тремя полученными из ключа таблицами: R, S и T. Предварительная обработка отображает ключ k на эти таблицы с помощью процедуры, основанной на SHA (см. раздел 18.7). 2-килобайтная таблица T представляет собой S-блок 9*32 битов.

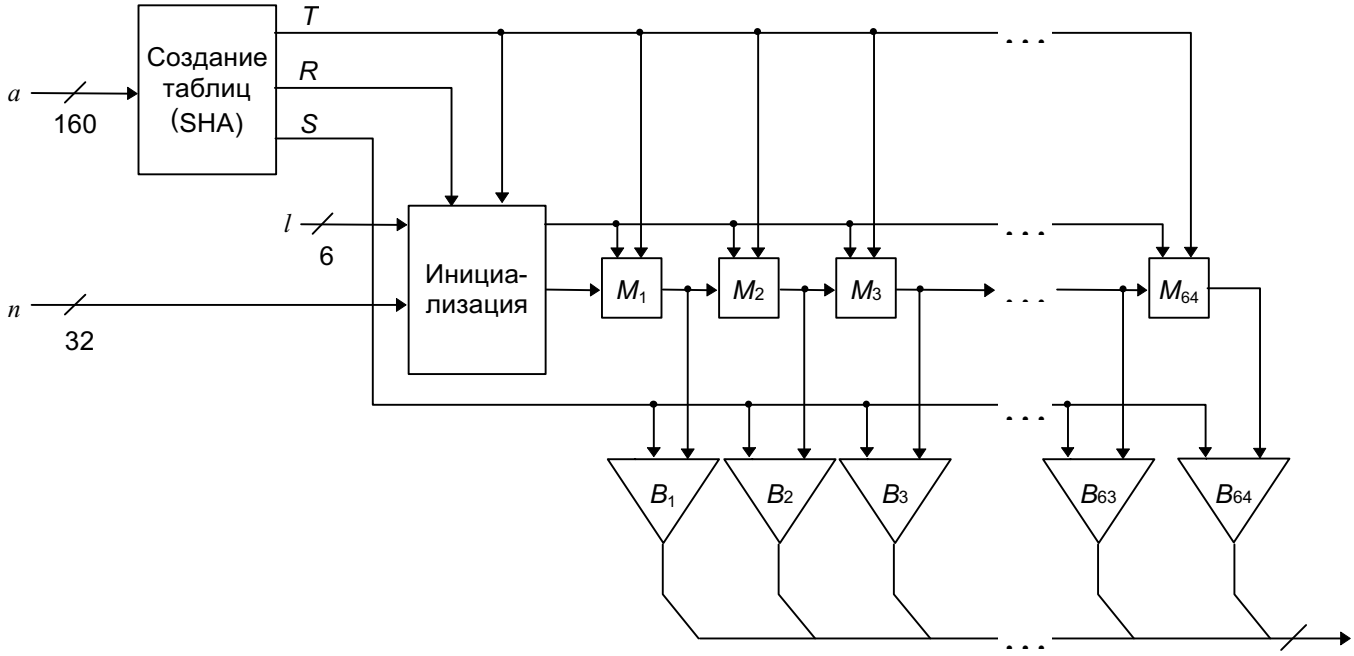


Рис. 17-1. Внутренний цикл SEAL.

SEAL также использует четыре 32-битовых регистра, *A*, *B*, *C* и *D*, начальные значения которых определяются *n* и полученными по *k* таблицами *R* и *T*. Эти регистры изменяются в ходе итераций, каждая из которых состоит из восьми этапов. На каждом этапе 9 битов первого регистра (все равно *A*, *B*, *C* или *D*) используются в качестве индекса таблицы *T*. Затем выбранное из *T* значение складывается со вторым регистром (снова одному из *A*, *B*, *C* или *D*) или объединяется с его содержимым с помощью XOR. Потом первый регистр циклически сдвигается на 9 позиций. На некоторых этапах второй регистр далее модифицируется с помощью сложения или XOR с содержимым первого регистра (уже сдвинутым). После 8 таких этапов *A*, *B*, *C* и *D* добавляются к потоку ключей, при этом каждый из них маскируется сложением или XOR с определенным словом из *S*. Итерация завершается прибавлением к *A* и *C* дополнительных значений, зависящих от *n*, n_1, n_2, n_3, n_4, выбор конкретного значения определяется четностью номера итерации. По видимому, при разработке этой схемы главными были следующие идеи:

1. Использование большого, секретного, получаемого из ключа S-блока (*T*).
2. Чередующиеся некоммутируемые арифметические операции (сложение и XOR).
3. Использование внутреннего состояния, поддерживаемого шифром, которое не проявляется явно в потоке данных (значения *ni*, которые модифицируют *A* и *C* в конце каждой итерации).
4. Изменение функции этапа в соответствии с номером этапа и изменение функции итерации в соответствии с номером итерации.

Для шифрования каждого байта текста SEAL требует около пяти элементарных операций. На 50-мегагерцовом процессоре i486 он работает со скоростью 58 Мбит/с. SEAL возможно является самым быстрым из описанных в этой книге.

С другой стороны SEAL должен выполнить предварительную обработку, заполняя внутренние таблицы. Размер этих таблиц составляет примерно 3 Кбайт, а для их расчета нужно примерно 200 вычислений SHA. Таким образом, SEAL не подходит для тех случаев, когда не хватает времени для обработки ключа или памяти для хранения таблиц.

Безопасность SEAL

SEAL достаточно новый алгоритм, ему еще предстоит пройти через горнило открытого криптоанализа. Это вызывает определенную настороженность. Однако SEAL кажется хорошо продуманным алгоритмом. Его особенности, в конечном счете, наполнены смыслом. К тому же Дон Копперсмит считается лучшим криптоаналитиком в мире.

Патенты и лицензии

SEAL запатентован [380]. По поводу лицензирования нужно обращаться к Управляющему по лицензиям IBM (Director of Licenses, IBM Corporation, 500 Columbus Ave., Thurnwood, NY, 10594).

17.3 WAKE

WAKE - сокращение от Word Auto Key Encryption (Автоматическое шифрование слов ключом)- это алгоритм, придуманный Дэвидом Уилером (David Wheeler) [1589]. Он выдает поток 32-битовых слов, которые с помощью XOR могут быть использованы для получения шифротекста из открытого текста или открытого текста из шифротекста. Это быстрый алгоритм.

WAKE работает в режиме CFB, для генерации следующего слова ключа используется предыдущее слово шифротекста. Алгоритм также использует S-блок из 256 32-битовых значений. Этот S-блок обладает одним особым свойством: Старший байт всех элементов представляет собой перестановку всех возможных байтов, а 3 младших байта случайны.

Сначала по ключу сгенерируем элементы S-блока, S_i. Затем проинициализируем четыре регистра с использованием того же или иного ключа: a_0, b_0, c_0 и d_0. Для генерации 32-битового слова потока ключей K_i.

$K_i = d_i$

Слово шифротекста C_i представляет собой XOR слова открытого текста P_i с K_i. Затем обновим четыре регистра:

$a_{i+1} = M(a_i,d_i)$

$b_{i+1} = M(b_i,a_{i+1})$

$c_{i+1} = M(c_i,b_{i+1})$

$d_{i+1} = M(d_i,c_{i+1})$

Функция M представляет собой

$M(x,y) = (x + y) >> 8 \oplus S_{(x+y)\text{^}255}$

Схема алгоритма показана на 15-й. Знак >> обозначает обычный, не циклический сдвиг вправо. Младшие 8 битов $x+y$ являются входом S-блока. Уилер приводит процедуру генерации S-блока, но на самом деле она неполна. Будет работать любой алгоритм генерации случайных байтов и случайной перестановки .

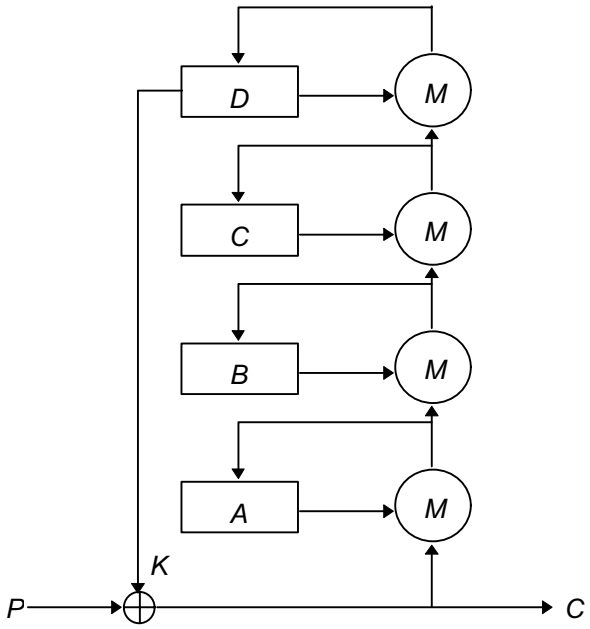


Рис. 17-2. WAKE.

Самым ценным качеством WAKE является его скорость. Однако он чувствителен к вскрытию с выбранным открытым текстом или выбранным шифротекстом. Этот алгоритм использовался в предыдущей версии антивирусной программы д-ра Соломона.

17.4 Сдвиговые регистры с обратной связью по переносу

Сдвиговый регистр с обратной связью по переносу, или FCSR (feedback with carry shift register), похож на LFSR. В обоих есть сдвиговый регистр и функция обратной связи, разница в том, что в FCSR есть также регистр переноса (см. 14-й). Вместо выполнения XOR над всеми битами отводной последовательности эти биты скла-

дываются друг с другом и с содержимым регистра переноса. Результат mod 2 и становится новым битом. Результат, деленный на 2, становится новым содержимым регистра переноса .

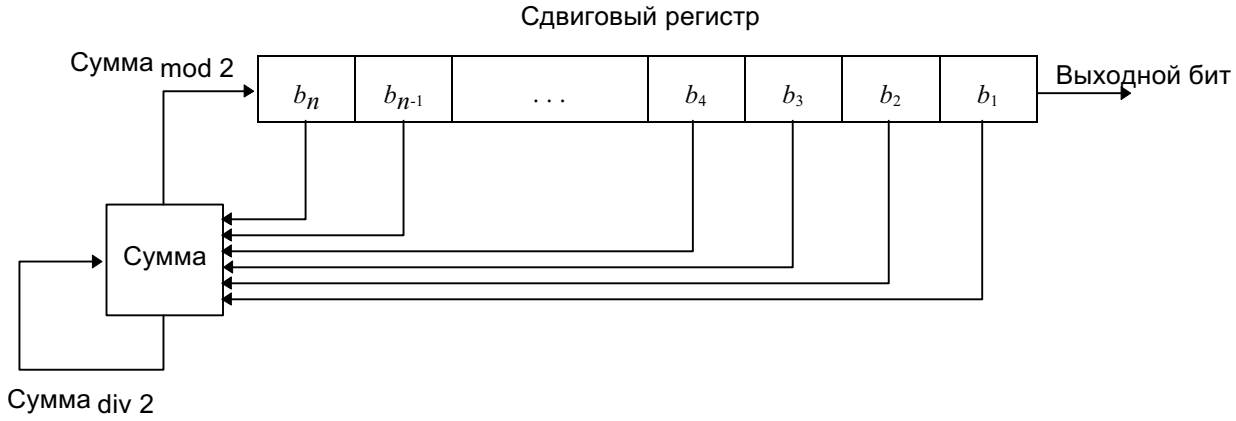


Рис. 17-3. Сдвиговый регистр с обратной связью по переносу.

На 13-й приведен пример 3-битового FCSR с ответвлениями в первой и второй позициях. Пусть его начальное значение 001, а начальное содержимое регистра переноса равно 0. Выходом будет является крайний правый бит сдвигового регистра.

Сдвиговый регистр	Регистр переноса
0 0 1	0
1 0 0	0
0 1 0	0
1 0 1	0
1 1 0	0
1 1 1	0
0 1 1	1
1 0 1	1
0 1 0	1
0 0 1	1
0 0 0	1
1 0 0	0

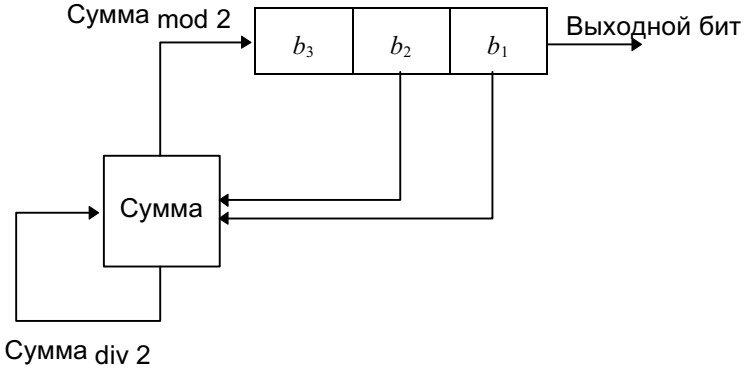


Рис. 17-4. 3-битовый FCSR.

Заметим, что конечное внутреннее состояние (включая содержимое регистра переноса) совпадает со вторым внутренним состоянием. С этого момента последовательность циклически повторяется с периодом, равным 10 .

Стоит упомянуть и еще о нескольких моментах. Во первых, регистр переноса является не битом, а числом. Размер регистра переноса должен быть не меньше $\log_2 t$, где t - это число ответвлений. В предыдущем примере

только три ответвления, поэтому регистр переноса однобитовый. Если бы было четыре ответвления, то регистр переноса состоял бы из двух битов и мог принимать значения 0, 1, 2 или 3.

Во вторых, существует начальная задержка прежде, чем FCSR перейдет в циклический режим. В предыдущем примере никогда не повторяется только одно состояние. Для больших и более сложных FCSR задержка может быть больше.

В третьих, максимальный период FCSR не 2^{n-1}, где n - длина сдвигового регистра. Максимальный период равен q-1, где q - это **целое число связи**. Это число задает ответвления и определяется как:

$q = 2q_1 + 2^2q_2 + 2^3q_3 + \ldots + 2^nq_n$-1

(Да, q_i отсчитываются слева направо.) И даже хуже, q должно быть простым числом, для которого 2 является примитивным корнем. В дальнейшем предполагается, что q удовлетворяет этому условию.

В приведенном примере q = 2*0 + 4*1 + 8*1 - 1 == 11. 11 - это простое число, примитивным корнем которого является 2. Роэтому максимальный период равен 10.

Не все начальные состояния дают максимальный период. Например, рассмотрим FCSR с начальным значением 101 и регистром переноса, установленным в 4.

Сдвиговый регистр	Регистр переноса
1 0 1	4
1 1 0	2
1 1 1	1
1 1 1	1

С этого момента регистр выплевывает бесконечную последовательность единиц.

Любое начальное состояние приводит к одной из четырех ситуаций. Во первых, оно может быть частью максимального периода. Во вторых, оно может перейти в последовательность максимального периода после начальной задержки. В третьих, после начальной задержки оно может породить бесконечную последовательность нулей. В четвертых, после начальной задержки оно может породить бесконечную последовательность единиц.

Для определения, чем закончится конкретное начальное состояние, существует математическая формула, но намного проще проверить это опытным путем. Запустите на некоторое время FCSR. (Если m - это начальный объем памяти, а t - количество ответвлений, то достаточно $\log_2(t) + \log_2(m)$ +1 тактов.) Если выходной поток вырождается в бесконечную последовательность нулей или единиц за n битов, где n - это длина FCSR, не используйте это начальное состояние. В противном случае его можно использовать. Так как начальное состояние FCSR соответствует ключу потокового шифра, это означает, что ряд ключей генератора на базе FCSR будут слабыми.

В 16-й перечислены все целые числа связи, меньшие 10000, для которых 2 является примитивным корнем. Для всех этих чисел максимальный период равен q-1. Чтобы получить по одному из этих чисел последовательность ответвлений, рассчитаем бинарный состав q+1. Например, 9949 дает последовательность ответвлений в позициях 1, 2, 3, 4, 6, 7, 9, 10 и 13, так как

$9950 = 2^{13} + 2^{10} + 2^9 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1$

В 15-й перечислены *все* отводные последовательности из четырех ответвлений, которые дают FCSR максимальной длины для сдвиговых регистров с длиной 32 бита, 64 бита и 128 битов. q, простое число, примитивным корнем которого является 2, получается объединением всех четырех значений, a, b, c и d.

$q = 2^a + 2^b + 2^c + 2^d$ - 1

Для создания FCSR с периодом q - 1 можно использовать любую из этих последовательностей.

Идея использовать в криптографии FCSR все еще является очень новой, впервые она была выдвинута Энди Клаппером (Andy Klapper) и Марком Горески (Mark Goresky) [844, 845, 654, 843, 846]. Также, как анализ LFSR основан на сложении примитивных многочленов mod 2, анализ FCSR основан на сложении неких чисел, называемых **2-adic**. Соответствующая теория выходит далеко за пределы этой книги, но в мире 2-adic чисел существуют аналоги для всего. Точно также, как определяется линейная сложность, можно определить и 2-adic сложность. Существует 2-adic аналог и для алгоритма Berlekamp-Massey. Это означает, что перечень возможных потоковых шифров по крайней мере удвоился. Все, что можно делать с LFSR, можно делать и с FCSR.

Существуют работы, развивающие эту идею и рассматривающие несколько регистров переноса. Анализ этих генераторов последовательностей основан на сложении разветвленных расширений 2-adic чисел [845, 846].

17.5 Потоковые шифры, использующие FCSR

Потоковые шифры на базе FCSR не описаны в литературе, теория все еще слишком нова. Чтобы как-то "погнать зайца дальше" я предложу здесь несколько вариантов. Я охватываю два направления: предлагаю потоковые шифры на базе FCSR, которые совпадают с ранее предложенными генераторами LFSR, а также предлагаю потоковые шифры, использующие FCSR и LFSR одновременно. Безопасность первого варианта возможно может быть проанализирована с помощью 2-adic чисел, генераторы второго варианта не могут быть проанализированы с использованием алгебраических методов - возможно их анализ может быть выполнен только косвенным образом. В любом случае, важно выбирать LFSR и FCSR с взаимно простыми периодами.

Все придет потом. Сейчас мне неизвестно ни о реализации, ни об анализе ни одной из этих идей. Подождите несколько лет и просматривайте литературу, прежде чем вы поверите в одну из этих идей.

Каскадные генераторы

Существует два способа использовать FCSR в каскадных генераторах:

— Каскад FCSR. Каскад Голлманна с FCSR вместо LFSR.

— Каскад LFSR/FCSR. Каскад Голлманна с генераторами, меняющими LFSR на FCSR и наоборот.

Комбинированные генераторы FCSR

Эти генераторы используют переменное количество LFSR и/или FCSR и множество функций, объединяющих регистры. Операция XOR разрушает алгебраические свойства FCSR, поэтому имеет смысл использовать эту операцию для их объединения. Генератор, показанный на 12th, использует переменное число FCSR. Его выходом является XOR выходов отдельных FCSR.

Другими генераторами, являющимися развитием аналогичных линий, являются:

— Генератор четности FCSR. Все регистры - FCSR, а объединяющая функция - XOR.

— Генератор четности LFSR/FCSR. Используется смесь LFSR и FCSR, объединяемых с помощью XOR.

— Пороговый генератор FCSR. Все регистры - FCSR, а объединяющей функцией является мажорирование.

— Пороговый генератор LFSR/FCSR. Используется смесь LFSR и FCSR, объединяемых с помощью мажорирования.

— Суммирующий генератор FCSR. Все регистры - FCSR, а объединяющая функция - сложение с переносом.

— Суммирующий генератор LFSR/FCSR. Используется смесь LFSR и FCSR, объединяемых с помощью сложения с переносом.

Табл. 17-1.
Целые значения связи для FCSR с максимальным периодом

2	211	587	947
5	227	613	1019
11	269	619	1061
13	293	653	1091
19	317	659	1109
29	347	661	1117
37	349	677	1123
53	373	701	1171
59	379	709	1187
61	389	757	1213
67	419	773	1229
83	421	787	1237
101	443	797	1259
107	461	821	1277
131	467	827	1283
139	491	829	1291
149	509	853	1301
163	523	859	1307
173	541	877	1373
179	547	883	1381
181	557	907	1427
197	563	941	1451

1453
1483
1493
1499
1523
1531
1549
1571
1619
1621
1637
1667
1669
1693
1733
1741
1747
1787
1861
1867
1877
1901
1907
1931
1949
1973
1979
1987
1997
2027
2029
2053
2069
2083
2099
2131
2141
2213
2221
2237
2243
2267
2269
2293
2309
2333
2339
2357
2371
2389
2437
2459
2467
2477
2531
2539
2549
2557
2579
2621
2659
2677
2683
2693
2699
2707
2741
2789
2797
2803
2819
2837
2843
2851
2861
2909
2939
2957
2963
3011
3019
3037
3067
3083
3187
3203
3253
3299
3307
3323
3347
3371
3413
3461
3467
3469
3491
3499
3517
3533
3539
3547
3557
3571
3581
3613
3637
3643
3659
3677
3691
3701
3709
3733
3779
3797
3803
3851
3853
3877
3907
3917
3923
3931
3947
3989
4003
4013
4019
4021
4091
4093
4099
4133
4139
4157
4219
4229
4243
4253
4259
4261
4283
4349
4357
4363
4373
4397
4451
4483
4493
4507
4517
4547
4603
4621
4637
4691
4723
4787
4789
4813
4877
4933
4957
4973
4987
5003
5011
5051
5059
5077
5099
5107
5147
5171
5179
5189
5227
5261
5309
5333
5387
5443
5477
5483
5501
5507
5557
5563
5573
5651
5659
5683
5693
5701
5717
5741
5749
5779
5813
5827
5843
5851
5869
5923
5939
5987
6011
6029
6053
6067
6101
6131
6173
6197
6203
6211
6229
6269
6277
6299
6317
6323
6373
6379
6389
6397
6469
6491
6547
6619
6637
6653
6659
6691
6701
6709
6733
6763
6779
6781
6803
6827
6829
6869
6883
6899

6907	7589	8429	9293
6917	7603	8443	9323
6947	7621	8467	9341
6949	7643	8539	9349
6971	7669	8563	9371
7013	7691	8573	9397
7019	7717	8597	9419
7027	7757	8627	9421
7043	7789	8669	9437
7069	7829	8677	9467
7109	7853	8693	9491
7187	7877	8699	9533
7211	7883	8731	9539
7219	7901	8741	9547
7229	7907	8747	9587
7237	7933	8803	9613
7243	7949	8819	9619
7253	8053	8821	9629
7283	8069	8837	9643
7307	8093	8861	9661
7331	8117	8867	9677
7349	8123	8923	9733
7411	8147	8933	9749
7451	8171	8963	9803
7459	8179	8971	9851
7477	8219	9011	9859
7499	8221	9029	9883
7507	8237	9059	9901
7517	8243	9173	9907
7523	8269	9181	9923
7541	8291	9203	9941
7547	8293	9221	9949
7549	8363	9227	
7573	8387	9283	

Табл. 17-2.
Отводные последовательности для FCSR максимальной длины

(32, 6, 3, 2)	(32, 29, 19, 2)	(64, 27, 22, 2)	(64, 49, 19, 2)
(32, 7, 5, 2)	(32, 29, 20, 2)	(64, 28, 19, 2)	(64, 49, 20, 2)
(32, 8, 3, 2)	(32, 30, 3, 2)	(64, 28, 25, 2)	(64,52,29,2)
(32, 13, 8, 2)	(32, 30, 7, 2)	(64, 29, 16, 2)	(64,53,8,2)
(32, 13, 12, 2)	(32, 31, 5, 2)	(64, 29, 28, 2)	(64, 53, 43, 2)
(32, 15, 6, 2)	(32, 31, 9, 2)	(64, 31, 12, 2)	(64, 56, 39, 2)
(32, 16, 2, 1)	(32, 31, 30, 2)	(64, 32, 21, 2)	(64, 56, 45, 2)
(32, 16, 3, 2)		(64, 35, 29, 2)	(64, 59, 5, 2)
(32, 16, 5, 2)	(64, 3, 2, 1)	(64, 36, 7, 2)	(64, 59, 8, 2)
(32, 17, 5, 2)	(64,14,3,2)	(64, 37, 2, 1)	(64, 59, 28, 2)
(32, 19, 2, 1)	(64,15,8,2)	(64, 37, 1 1, 2)	(64, 59, 38, 2)
(32, 19, 5, 2)	(64, 17, 2, 1)	(64,39,4,2)	(64,59,44,2)
(32, 19, 9, 2)	(64, 17, 9, 2)	(64, 39, 25, 2)	(64, 60, 49, 2)
(32, 19, 12, 2)	(64, 17, 16, 2)	(64, 41, 5, 2)	(64, 61, 51, 2)
(32, 19, 17, 2)	(64, 19, 2, 1)	(64, 41, 1 1, 2)	(64, 63, 8, 2)
(32, 20, 17, 2)	(64, 19, 18, 2)	(64,41,27,2)	(64, 63, 13, 2)
(32, 21, 9, 2)	(64, 24, 19, 2)	(64, 43, 21, 2)	(64, 63, 61, 2)
(32, 21, 15, 2)	(64, 25, 3, 2)	(64, 43, 28, 2)	
(32,23,8,2)	(64,25,4,2)	(64, 45, 28, 2)	(96, 15, 5. 2)
(32, 23, 21, 2)	(64, 25, 1 1, 2)	(64, 45, 41, 2)	(96, 21, 17, 2)
(32, 25, 5, 2)	(64, 25, 19, 2)	(64, 47, 5, 2)	(96, 25, 19, 2)
(32, 25, 12, 2)	(64, 27, 5, 2)	(64, 47, 21, 2)	(96, 25, 20, 2)
(32,27,25,2)	(64, 27, 16, 2)	(64, 47, 30, 2)	(96, 29, 15, 2)

(96, 29, 17, 2)
(96, 30, 3, 2)
(96, 32, 21, 2)
(96, 32, 27, 2)
(96,33,5,2)
(96, 35, 17, 2)
(96, 35, 33, 2)
(96, 39, 21, 2)
(96,40,25,2)
(96, 41, 12, 2)
(96, 41, 27, 2)
(96, 41, 35, 2)
(96, 42, 35, 2)
(96, 43, 14, 2)
(96, 44, 23, 2)
(96, 45, 41, 2)
(96, 47, 36, 2)
(96, 49, 31, 2)
(96,51,30,2)
(96,53,17,2)
(96, 53, 19, 2)
(96, 53, 32, 2)
(96, 53, 48, 2)
(96, 54, 15, 2)
(96, 55, 44, 2)
(96, 55, 53, 2)
(96, 56, 9, 2)
(96,56,51,2)
(96, 57, 3, 2)
(96, 57, 17, 2)
(96, 57, 47, 2)
(96, 58, 35, 2)
(96, 59, 46, 2)
(96, 60, 29, 2)
(96, 60, 41, 2)
(96, 60, 45, 2)
(96, 61, 17, 2)
(96, 63, 20, 2)
(96, 65, 12, 2)
(96, 65, 39, 2)
(96, 65, 51, 2)
(96, 67, 5, 2)
(96, 67, 25, 2)
(96,67,34,2)
(96, 68, 5, 2)
(96, 68, 19, 2)
(96, 69, 17, 2)
(96,69,36,2)
(96, 70, 23, 2)
(96, 71, 6, 2)
(96, 71, 40, 2)
(96, 72, 53, 2)
(96, 73, 32, 2)
(96, 77, 27, 2)
(96, 77, 31, 2)
(96, 77, 32, 2)
(96, 77, 33, 2)
(96,77,71,2)
(96,78,39,2)
(96, 79, 4, 2)
(96, 81, 80, 2)
(96, 83, 14, 2)
(96, 83, 26, 2)
(96, 83, 54, 2)
(96, 83, 60, 2)
(96, 83, 65, 2)
(96, 83, 78, 2)
(96, 84, 65, 2)
(96, 85, 17, 2)
(96, 85, 31, 2)
(96, 85, 76, 2)
(96,85,79,2)
(96,86,39,2)
(96,86,71,2)
(96, 87, 9, 2)
(96, 87, 44, 2)
(96, 87, 45, 2)
(96, 88, 19, 2)
(96, 88, 35, 2)
(96, 88, 43, 2)
(96,88,79,2)
(96, 89, 35, 2)
(96, 89, 51, 2)
(96, 89, 69, 2)
(96, 89, 87, 2)
(96, 92, 51, 2)
(96,92,71,2)
(96, 93, 32, 2)
(96, 93, 39, 2)
(96, 94, 35, 2)
(96, 95, 4, 2)
(96, 95, 16, 2)
(96, 95, 32, 2)
(96, 95, 44, 2)
(96, 95, 45, 2)

(128, 5, 4, 2)
(128, 15, 4, 2)
(128, 21, 19, 2)
(128, 25, 5, 2)
(128, 26, 11, 2)
(128,27,25,2)
(128, 31, 25, 2)
(128, 33, 21, 2)
(128, 35, 22, 2)
(128, 37, 8, 2)
(128, 41, 12, 2)
(128, 42, 35, 2)
(128, 43, 25, 2)
(128,43,42,2)
(128,45,17,2)
(128,45,27,2)
(128, 49, 9, 2)
(128, 51, 9, 2)
(128, 54, 51, 2)
(128, 55, 45, 2)
(128, 56, 15, 2)
(128, 56, 19, 2)
(128,56,55,2)
(128, 57, 21, 2)
(128, 57, 37, 2)
(128, 59, 29, 2)
(128, 59, 49, 2)
(128, 60, 57, 2)
(128,61,9,2)
(128, 61, 23, 2)
(128, 61, 52, 2)
(128, 63, 40, 2)
(128, 63, 62, 2)
(128, 67, 41, 2)
(128, 69, 33, 2)
(128, 71, 53, 2)
(128, 72, 15, 2)
(128,72,41,2)
(128, 73, 5, 2)
(128, 73, 65, 2)
(128, 73, 67, 2)
(128, 75, 13, 2)
(128, 80, 39, 2)
(128,80,53,2)
(128, 81, 55, 2)
(128, 82, 67, 2)
(128, 83, 60, 2)
(128, 83, 61, 2)
(128, 83, 77, 2)
(128, 84, 15, 2)
(128, 84, 43, 2)
(128,85,63,2)
(128,87,57,2)
(128,87,81,2)
(128, 89, 81, 2)
(128, 90, 43, 2)
(128, 91, 9, 2)
(128, 91, 13, 2)
(128, 91, 44, 2)
(128, 92, 35, 2)
(128,95,94,2)
(128, 96, 23, 2)
(128, 96, 61, 2)
(128, 97, 25, 2)
(128, 97, 68, 2)
(128, 97, 72, 2)
(128,97,75,2)
(128, 99, 13, 2)
(128, 99, 14, 2)
(128, 99, 26, 2)
(128, 99, 54, 2)
(128, 99, 56, 2)
(128, 99, 78, 2)
(128, 100, 13, 2)
(128, 100, 39, 2)
(128,101,44,2)
(128, 101, 97, 2)
(128, 103, 46, 2)
(128, 104, 13, 2)
(128, 104, 19, 2)
(128, 104, 35, 2)
(128,105,7,2)
(128, 105, 11, 2)
(128, 105, 31, 2)
(128, 105, 48, 2)
(128, 107, 40, 2)
(128, 107, 62, 2)
(128, 107, 102, 2)
(128, 108, 35, 2)
(128,108,73,2)
(128,108,75,2)
(128,108,89,2)
(128, 109, 1 1, 2)
(128, 109, 108, 2)
(128, 1 10, 23, 2)
(128, Ill, 61, 2)
(128, 113, 59, 2)
(128, 114, 83, 2)
(128,115,73,2)
(128, 117, 105, 2)
(128, 119, 30, 2)
(128, 119, 101, 2)
(128, 120, 9, 2)
(128, 120, 27, 2)
(128,120,37,2)
(128, 120, 41, 2)
(128, 120, 79, 2)
(128, 120, 81, 2)
(128, 121, 5, 2)
(128, 121, 67, 2)
(128, 121, 95, 2)
(128, 121, 96, 2)
(128, 123, 40, 2)
(128,123,78,2)
(128, 124, 41, 2)
(128, 124, 69, 2)
(128, 124, 81, 2)
(128, 125, 33, 2)
(128, 125, 43, 2)
(128,127,121,2)

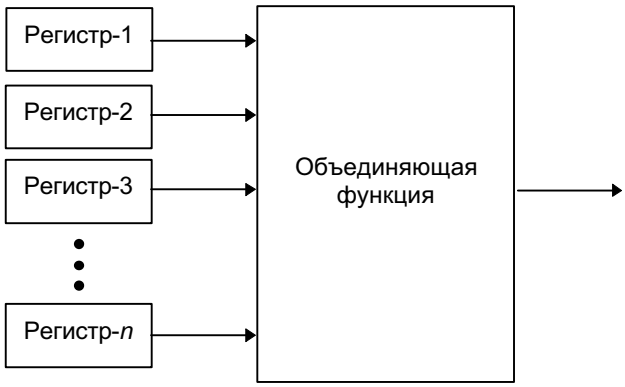


Рис. 17-5. Комбинированные генераторы.

Каскад LFSR/FCSR с суммированием/четностью

По теории сложение с переносом разрушает алгебраические свойства LFSR, а XOR разрушает алгебраические свойства FCSR. Данный генератор объединяет эти идеи, используемые в перечисленных суммирующем генераторе LFSR/FCSR и генераторе четности LFSR/FCSR, с каскадом Голлманна.

Генератор представляет собой последовательность массивов регистров , тактирование каждого массива определяется выходом предыдущего массива. На 11-й показан один этап такого генератора. Тактируется первый массив LFSR, и результаты объединяются сложением с переносом. Если выход функции объединения равен 1, то тактируется следующий массив (из FCSR), и выход этих FCSR объединяется с выходом предыдущей функции объединения с помощью XOR. Если выход первой функции объединения равен 0, то массив FCSR не тактируется, и выход просто складывается с переносом, полученным на предыдущем этапе Если выход этой второй функции объединения равен 1, то тактируется третий массив (из LFSR), и т.д.

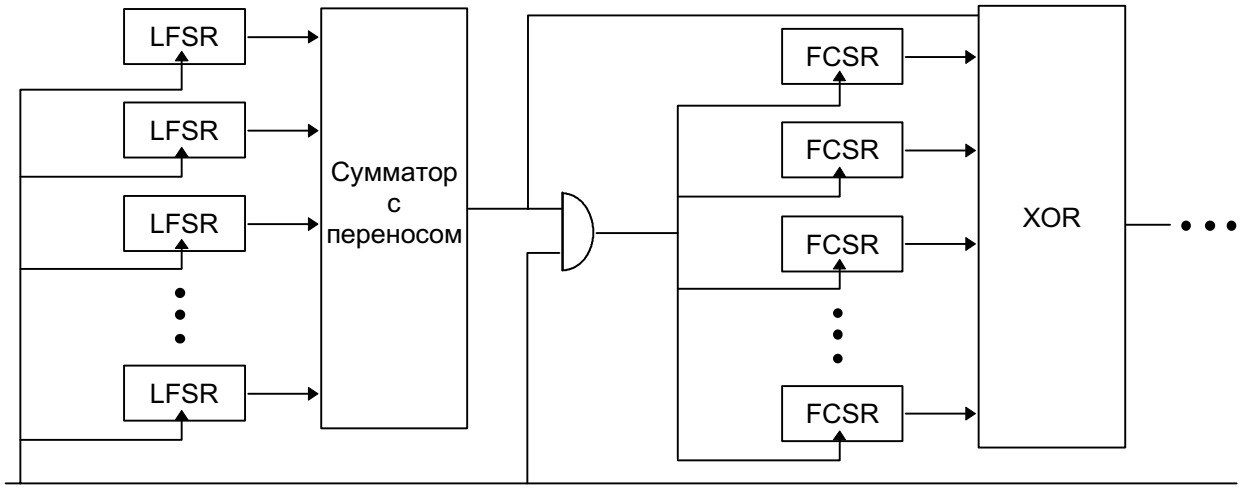


Рис. 17-6. Придуманный генератор.

Генератор использует много регистров: $n*m$, где n - количество этапов, а m - количество регистров на этапе. Я рекомендую $n = 10$ и $m = 5$.

Чередующиеся генераторы "стоп-пошел"

Эти генераторы использую FCSR вместо некоторых LFSR. Кроме того, операция XOR может быть заменена сложением с переносом (см. 10-й).

— Генератор "стоп-пошел" FCSR. Регистр-1, Регистр-2 и Регистр-3 - это FCSR. Объединяющая функция - XOR.

— Генератор "стоп-пошел" FCSR/LFSR. Регистр-1 - FCSR, а Регистр-2 и Регистр-3 - LFSR. Объединяющая функция - сложение с переносом.

— Генератор "стоп-пошел" LFSR/FCSR. Регистр-1 - LFSR, а Регистр-2 и Регистр-3 - FCSR. Объединяющая функция - XOR.

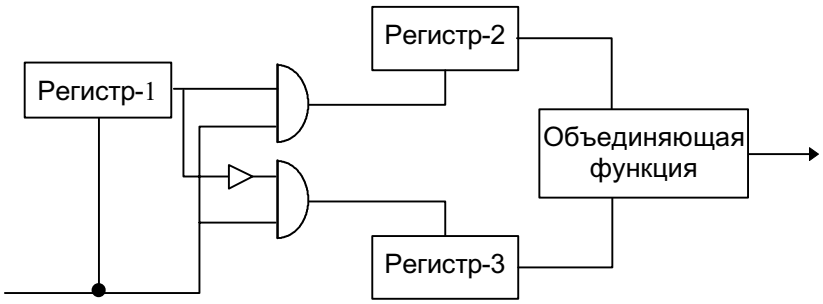


Рис. 17-7. Чередующийся генератор "стоп-пошел"

Прореживаемые генераторы

Существует четыре основных типа генераторов, использующих FCSR:

— Прореживаемый генератор FCSR. Прореживаемый генератор с FCSR вместо LFSR.

— Прореживаемый генератор FCSR/LFSR. Прореживаемый генератор с LFSR, прореживающим FCSR.

— Прореживаемый генератор LFSR/FCSR. Прореживаемый генератор с FCSR, прореживающим LFSR.

— Самопрореживаемый генератор FCSR. Самопрореживаемый генератор с FCSR вместо LFSR.

17.6 Сдвиговые регистры с нелинейной обратной связью

Нетрудно представить более сложную, чем используемая в LFSR или FCSR, последовательность обратной связи. Проблема в том, что не существует математического аппарата, позволяющего провести анализ таких п о-следовательностей. Что-то получится, но кто знает что? Вот некоторые из проблем, связанных со сдвиговыми регистрами с нелинейной обратной связью.

— В выходной последовательности могут быть смещения, например, единиц может быть больше, чем нулей.

— Максимальный период последовательности может быть меньше, чем ожидалось .

— Период последовательности для различных начальных значений может быть различным .

— Последовательность какое-то время может выглядеть как случайная, а потом "скатываться" к единстве н-ному значению. (Это можно легко устранить, выполняя XOR крайнего правого бита с нелинейной фун к-цией.)

Плюсом является то, что из-за отсутствия теории анализа сдвиговых регистров с нелинейной обратной св я-зью существует немного способов криптоанализировать потоковые шифры, основанные на таких регистрах . Использовать сдвиговые регистры с нелинейной обратной связью можно, но очень осторожно .

В сдвиговом регистре с нелинейной обратной связью функция обратной связи может быть произвольной (например, как на).

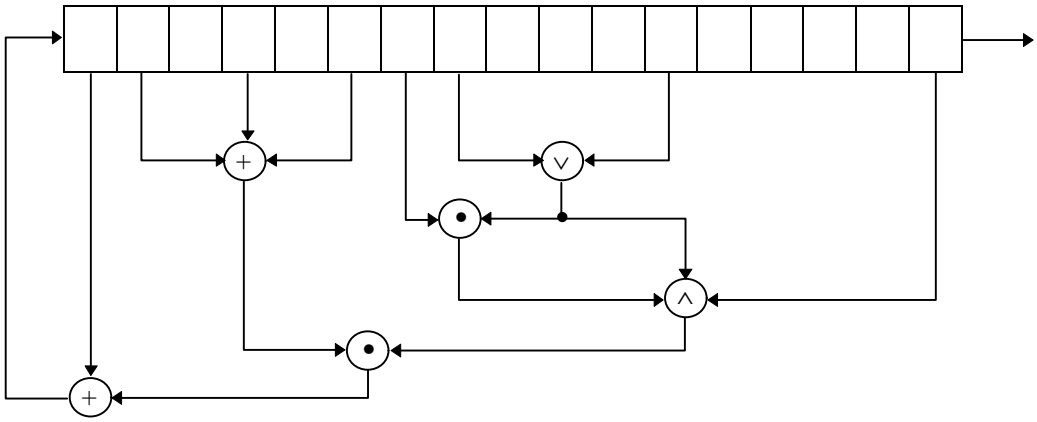


Рис. 17-8. Сдвиговый регистр с нелинейной обратной связью (возможно небезопасный).

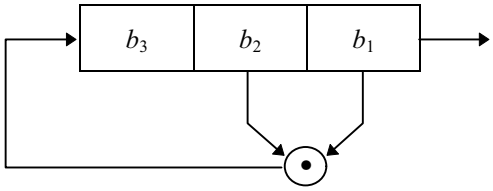


Рис. 17-9. 3-битовый сдвиговый регистр с нелинейной обратной связью.

На 8-й показан 3-битовый генератор со следующей обратной связью: новым битом является произведение первого и второго битов. Если его проинициализировать значением 110, то последовательность внутренних состояний будет следующей:

1 1 0

0 1 1

1 0 1

0 1 0

0 0 1

0 0 0

0 0 0

И так до бесконечности. Выходом является последовательность младших значащих битов:

0 1 1 0 1 0 0 0 0 0 0 0. . . .

Это не слишком полезно.

Может быть и хуже. Если начальное значение 100, то следующими состояниями являются 010, 001, а затем всегда 000. Если начальным значением является 111, то оно будет повторяться всегда и с самого начала.

Была проделана определенная работа по вычислению линейной сложности произведения двух LFSR [1650, 726, 1364, 630, 658, 659]. Конструкция, включающая вычисление LFSR над полем нечетных характеристик [310] не является безопасной [842.].

17.7 Другие потоковые шифры

В литературе описывались и другие потоковые шифры. Вот некоторые из них.

Генератор Плесса (Pless)

Этот генератор использует свойства J-K триггеров [1250]. Восемь LFSR управляют четырьмя J-K триггерами; каждый триггер нелинейно объединяет два LFSR. Чтобы избежать проблемы, что выход триггера определяет и источник, и значение следующего выходного бита, после тактирования четырех триггеров их выходы перемешиваются для получения окончательного потока ключей.

Этот алгоритм был криптоаналитически взломан с помощью вскрытия каждого триггера в отдельности

[1356]. К тому же, объединение J-K триггеров слабо криптографически; генераторы такого типа не устоят перед корреляционным вскрытием [1451].

Генератор на базе клеточного автомата

В [1608, 1609], Стив Вольфрам (Steve Wolfram) предложил использовать в качестве генератора псевдослучайных чисел одномерный клеточный автомат. Рассмотрение клеточного автомата не является предметом этой книги, но генератор Вольврама состоит из одномерного массива битов $a_1, a_2, a_3, \ldots, a_k, \ldots, an$ и функции обновления:

$a_k' = a_{k-1} \oplus (a_k \vee a_{k+1})$

Бит извлекается из одного из значений a_k, реально все равно какого.

Генератор ведет себя как вполне случайный. Однако для этих генераторов существует успешное вскрытие с известным открытым текстом [1052]. Это вскрытие выполнимо на PC со значениями n вплоть до 500 битов. Кроме того, Пол Барделл (Paul Bardell) доказал, что выход клеточного автомата может быть также сгенерирован с помощью сдвигового регистра с линейной обратной связью той же длины и, следовательно, не дает большей безопасности [83].

Генератор 1/p

Этот генератор был предложен и подвергнут криптоанализу в [193]. Если внутреннее состояние генератора в момент времени t равно x_t, то

$x_{t+1} = bx_t \bmod p$

Выходом генератора является младший значащий бит $x_t \operatorname{div} p$, где div - это целочисленное деление с усечением. Для максимального периода константы b и p должны быть выбраны так, что p - простое число, а b - примитивный корень mod p. К сожалению, этот генератор не безопасен. (Заметим, что для $b = 2$ FCSR целыми числами связи выдает последовательность, обратную данной.)

crypt(1)

Оригинальный алгоритм шифрования UNIX, crypt(1), представляет собой потоковый шифр, использующий те же идеи, что и Энигма. Это 256-элементный, одноротorный подстановочный шифр с отражателем. И ротор, и отражатель получаются из ключа. Этот алгоритм намного проще, чем немецкая Энигма времен второй мировой войны, и квалифицированному криптоаналитику несложно его взломать [1576, 1299]. Для вскрытия файлов, зашифрованных crypt(1), можно использовать свободно доступную программу UNIX, называемую Crypt Breakers Workbench (CBW, инструмент взломщика шифров).

Другие схемы

Еще один генератор основан на проблеме рюкзака (см. раздел 19.2) [1363]. CRYPTO-LEGGO небезопасен [301]. Джоан Дэймен (Joan Daemen) разработала SubStream, Jam и StepRightUp [402], но они слишком новы, чтобы их комментировать. Множество других алгоритмов описано в литературе, но еще больше хранится в секрете и встроено в аппаратуру.

17.8 Системно-теоретический подход к проектированию потоковых шифров

На практике, проектирование потокового шифра во многом похоже проектирование блочного шифра. В этом случае используется больше математической теории, но в конце концов криптограф предлагает какую-то схему и затем пытается выполнить ее анализ.

Согласно Райнеру Рюппелу существует четыре различных подхода к проектированию потоковых шифров [1360, 1362]:

— Системно-теоретический подход. Используя ряд фундаментальных критериев и законов проектирования, пытается удостовериться, что каждая схема создает сложную и неизвестную проблему для криптоанализа,.

— Информационно-теоретический подход. Пытается сохранить открытый текст в тайне от криптоаналитика. Независимо от того, как много действий выполнит криптоаналитик, он никогда не получит однозначного решения.

— Сложностно-теоретический подход. Пытается использовать в качестве основания для криптосистемы некоторую известную и сложную проблему, такую как разложение на множители или взятие дискретных логарифмов, или сделать криптосистему эквивалентной этой проблеме.

— Рандомизированный подход. Пытается создать чрезвычайно большую проблему, заставляя криптоанал итика проверить множество бессмысленных данных в ходе попыток криптоанализа .

Эти подходы отличаются предположениями о возможностях и способностях криптоаналитика, определением успеха криптоанализа и пониманием безопасности. Большинство исследований в этой области - теоретические, но среди бесполезных потоковых шифров есть и вполне приличные .

Системно-теоретический подход использовался во всех ранее приведенных потоковых шифрах, результатом его применения являются большинство используемых в реальном мире потоковых шифров . Криптограф разрабатывает генераторы потока ключей, обладающие проверяемыми характеристиками безопасности - периодом, распределением битов, линейной сложностью и т.д. - а не шифры, основанные на математической теории . Криптограф также изучает различные методы криптоанализа этих генераторов и проверяет, устойчивы ли ген ераторы по отношению к этим способам вскрытия.

Со временем этот подход привел к появлению набора критериев проектирования потоковых шифров [1432, 99, 1357, 1249]. Они рассматривались Рюппелом в [1362], где он подробно приводит теоретические основы этих критериев.

— Длинный период без повторений.

— Критерий линейной сложности - большая линейная сложность , линейный профиль сложности, локальная линейная сложность и т.д.

— Статистические критерии, например, идеальные k-мерные распределения.

— Путаница - каждый бит потока ключей должен быть сложным преобразованием всех или большинства битов ключа.

— Диффузия - избыточность в подструктурах должна рассеиваться, приводя к более "размазанной" стат истике.

— Критерии нелинейности для логических функций, такие как отсутствие корреляции m-го порядка, расстояние до линейных функций, лавинный критерий, и т.д.

Этот перечень критериев проектирования не уникален для потоковых шифров, разработанных с помощью системно-теоретического подхода, он справедлив для всех потоковых шифров. Это справедливо и для всех блочных шифров. Особенностью системно-теоретического подхода является то, что потоковые шифры неп осредственно разрабатываются, чтобы удовлетворить этим критериям .

Главной проблемой таких криптосистем является невозможность доказать их безопасность, никогда не было доказано, что эти критерии проектирования необходимы или достаточны для безопасности . Генератор потока ключей может удовлетворять всем правилам разработки, но тем не менее оказаться небезопасным . Другой может оказаться безопасным. Этом процессе все еще остается что-то магическое.

С другой стороны вскрытие любого из этих генераторов потока ключей представляет собой отличную пр облему для криптоаналитика. Если будет разработано достаточно различных генераторов , может оказаться, что криптоаналитик не станет тратить время, взламывая каждый из них . Может, его больше заинтересует возмо жность прославиться, достигнув успеха, разлагая на множители большие числа или вычисляя дискретные лог арифмы.

17.9 Сложностно-теоретический подход к проектированию потоковых шифров

Рюппел также очертил сложностно-теоретический подход к проектированию потоковых шифров . В соответствии с ним криптограф пытается использовать теорию сложности, чтобы доказать его генераторы безопасны . Следовательно, генераторы должны быть как можно больше сложнее, основываясь на тех же трудных пробл емах, что и криптография с открытыми ключами. И, также как алгоритмы с открытыми ключами, они оказыв аются медленными и громоздкими.

Генератор псевдослучайных чисел Шамира

Эди Шамир использовал в качестве генератора псевдослучайных чисел алгоритм RSA [1417]. Хотя Шамир показал, что предсказание выхода генератора псевдослучайных чисел равносильно взлому RSA, потенциальное смещение выхода была продемонстрирована в [1401, 200].

Генератор Blum-Micali

Безопасность этого генератора определяется трудностью вычисления дискретных логарифмов [200]. Пусть g - простое число, а p - еще одно простое число. Ключ x_0 начинает процесс:

$x_{i+1} = g^{x_i} \bmod p$

Выходом генератора является 1, если $x_i < (p - 1)/2$, и 0 в противном случае.

Если p достаточно велико, чтобы вычисление дискретных логарифмов mod p стало физически невозможным, то этот генератор безопасен. Дополнительные теоретические результаты можно найти в [1627, 986, 985, 1237, 896, 799].

RSA

Этот генератор RSA [35, 36] является модификацией [200]. Начальные параметры - модуль N, произведение двух больших простых чисел p и q, и целое число e, относительно простое с $(p-1)(q-1)$, а также стартовое случайное число x_0, меньшее N.

$x_{i+1} = x^e_i \bmod N$

Выход генератора представляет собой младший значащий бит x_i. Безопасность этого генератора опирается на сложность вскрытия RSA. Если N достаточно велико, то генератор безопасен. Дополнительная теория приведена в [1569, 1570, 1571, 30, 354].

Blum, Blum, and Shub

Простейший и наиболее эффективный генератор, использующий сложностно-теоретический подход, в честь своих авторов называется Blum, Blum, and Shub. Мы сократим его название до BBS, хотя иногда его называют генератором с квадратичным остатком [193].

Теория генератора BBS использует квадратичные остатки по модулю n (см. раздел 11.3). Вот как он работает.

Сначала найдем два простых числа, p и q, которые конгруэнтны 3 modulo 4. Произведение этих чисел, n, является целым числом Блюма (Blum). Выберем другое случайное целое число x, взаимно простое с n. Вычислим

$x_0 = x^2 \bmod n$

Это стартовое число генератора.

Теперь можно начать вычислять биты. i-ым псевдослучайным битом является младший значащий бит x_i, где

$x_i = x_{i-1}^2 \bmod n$

Самым интригующим свойством этого генератора является то, что для получения i-го бита не нужно вычислять предыдущие i-1 биты. Если вам известны p и q, вы можете вычислить i-ый бит непосредственно.

b_i - это младший значащий бит x_i, где $x_i = x_0^{(2^i) \bmod ((p-1)(q-1))}$

Это свойство означает, что вы можете использовать этот криптографически сильный генератор псевдослучайных чисел в качестве потоковой криптосистемы для файла с произвольным доступом.

Безопасность этой схемы основана на сложности разложения n на множители. Можно опубликовать n, так что кто угодно может генерировать биты с помощью генератора. Однако пока криптоаналитик не сможет разложить n на множители, он никогда не сможет предсказать выход генератора - ни даже утверждать что-нибудь вроде: "Следующий бит с вероятностью 51 процент будет единицей".

Более того, генератор BBS **непредсказуем в левом направлении** и непредсказуем в правом направлении. Это означает, что получив последовательность, выданную генератором, криптоаналитик не сможет предсказать ни следующий, ни предыдущий бит последовательности. Это вызвано не безопасностью, основанной на каком-то никому не понятном сложном генераторе битов, а математикой разложения n на множители.

Этот алгоритм медленен, но есть способы его ускорить. Оказывается, что в качестве псевдослучайных битов можно использовать несколько каждого x_i. В соответствии с [1569, 1570, 1571, 35, 36] если n - длина x_i, можно использовать $\log_2 n$ младших значащих битов x_i. Генератор BBS сравнительно медленный и не подходит для потоковых шифров. Однако для высоконадежных приложений, таких как генерация ключей, этот генератор лучше многих других.

17.10 Другие подходы к проектированию потоковых шифров

При информационно-теоретическом подходе к потоковым шифрам предполагается, что криптоаналитик обладает неограниченными временем и вычислительной мощностью. Единственным практически реализованным потоковым шифром, защищенным от такого противника, является одноразовый блокнот (см. раздел 1.5). Так как писать биты в блокноте не очень удобно, его иногда называют **одноразовой лентой**. На двух магнитных лентах, на одной для шифрования, а на другой для дешифрирования, должен быть записан идентичный поток ключей. Для шифрования просто выполняется XOR открытого текста с битами ленты. Для дешифрирования

выполняется XOR шифротекста с битами другой, идентичной ленты. Один и тот же поток ключей нельзя использовать дважды. Так как биты потока ключей действительно случайны, предсказать поток ключей невозможно. Если сжигать ленты после использования, то безопасность будет абсолютной (при условии, что у кого-то другого нет копии ленты).

Другой информационно-теоретический потоковый шифр, разработанных Клаусом Шнорром (Claus Schnorr) предполагает, что криптоаналитик имеет доступ только к ограниченному числу битов шифротекста [1395]. Результаты являются слишком теоретическими results и не имеют практического значения. Подробности можно найти [1361, 1643,1193].

С помощью рандомизированного потокового шифра криптограф пытается сделать решение проблемы, стоящей перед криптоаналитиком, физически невозможным. Для этого, сохраняя небольшой размер секретного ключа, криптограф значительно увеличивает количество битов, с которыми придется иметь дело криптоанализу. Это может быть сделано за счет использования при шифровании и дешифрировании большой опубликованной случайной строки. Ключ же указывает, какие части строки будут использованы при шифровании и дешифрировании. Криптоаналитику, не знающему ключа, придется перебирать случайные комбинации частей строки. Безопасность такого шифра можно выразить с помощью среднего числа битов, которые должен проверить криптоаналитик прежде, чем вероятностьопределить ключ значительно удучшится по сравнению с вероятностью простого угадывания.

Шифр "Рип ван Винкль"

Джеймс Массей (James Massey) и Ингемар Ингемарсон (Ingemar Ingemarsson) предложили шифр "Рип ван Винкль" [1011], названный так, потому что получатель, чтобы начать дешифрирование, должен получить 2^n битов шифротекста. Алгоритм, показанный на 7-й, прост в реализации, гарантировано безопасен и совершенно непрактичен. Просто выполните XOR открытого текста с потоком ключей и задержите поток ключей на время от 0 до 20 лет - точная задержка является частью ключа. По словам Массея: "Можно легко доказать, что вражескому криптоаналитику для вскрытия шифра понадобятся тысячи лет, если кто-то согласится подождать с чтением открытого текста миллионы лет." Развитие этой идеи можно найти в [1577, 755].

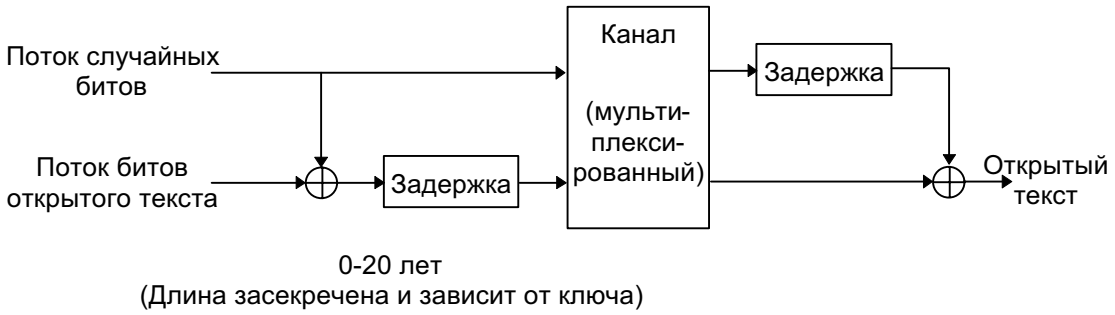


Рис. 17-10. Шифр "Рип ван Винкль".

Рандомизированный потоковый шифр Диффи

Эта схема впервые была предложена Уитфилдом Диффи [1362]. Используется 2^n случайных последовательностей. Ключ представляет собой случайную n-битовую строку. Для шифрования сообщения Алиса использует k-ую случайную строку как одноразовый блокнот. Затем она отправляет шифротекст и 2^n случайных строк по 2^n+1 различным каналам связи.

Боб знает k-, поэтому он может легко выбрать, какой из одноразовых блокнотов использовать для дешифрования сообщения. Еве остается только перебирать случайные последовательности, пока она не найдет правильный одноразовый блокнот. Для вскрытия потребуется проверить некоторое число битов, по порядку равное $O(2^n)$. Рюппел указал, что, если вы отправляете n случайных строк вместо2^n, и если ключ используется для задания линейной комбинации этих случайных строк, безопасность остается на прежнем уровне .

Рандомизированный потоковый шифр Маурера

Уели Маурер (Ueli Maurer) описал схему, основанную на выполнении XOR открытого текста с несколькими большими открытыми последовательностями случайных битов [1034, 1029, 1030]. Ключ является набором стартовых позиций в каждой последовательности. Можно доказать, что такой шифр почти безопасен, с вероятность взлома определяется объемом памяти, имеющейся в распоряжении взломщика, независимо от доступной ему вычислительной мощности. Маурер утверждает, что эта схема становится практичной при 100 различных последовательностях длиной 10^{20} случайных битов каждая. Одним из способов получить рак много битов явля-

ется оцифровка поверхности Луны.

17.11 Шифры с каскадом нескольких потоков

Если производительность не важна, то нет причин выбирать несколько потоковых шифров и объединять их в каскад. Для получения шифротекста просто выполните XOR выхода каждого генератора с открытым текстом. Результат Уели Маурера (см. раздел 15.7) показывает, что если генераторы используют независимые ключи, то безопасность каскада по крайней мере не меньше безопасности самого сильного алгоритма каскада, а скорее всего и намного больше.

Потоковые шифры объединяются теми же способами, что и блоковые (см. главу 15). Потоковые шифры можно объединить в каскад (см. раздел 15.7) с другими потоковыми шифрами или с блочными шифрами.

Ловким трюком является использование одгого алгоритма, потокового или блочного, для частого обновления ключа быстрого потокового алгоритма (которым может быть и блочный алгоритм в режиме OFB). Быстрый алгоритм может быть слабым, так как криптоаналитик никогда не получит достаточно открытого текста, зашифрованного одним ключом.

Существует способ разменять размер внутреннего состояния быстрого алгоритма (который может влиять на безопасность) на частоту смены ключа. Смена ключа должна быть относительно частой, не стоит использовать для этого алгоритмы с длинной процедурой установки ключа. Кроме того, смена ключа не должна зависеть от внутреннего состояния быстрого алгоритма.

17.12 Выбор потокового шифра

Если изучение потоковых шифров и дает какой-либо результат, так это появление с пугающей регулярностью все новых способов вскрытия. Традиционно потоковые шифры опирались на большую математическую теорию. Эту теорию можно было использовать для доказательства положительных качеств шифра, но ее же можно было использовать для поиска новых способов вскрытия шифра. По этой причины любой потоковый шифр, основанный только на LFSR, вызывает мое беспокойство.

Я предпочитаю потоковые шифры, спроектированные подобно блочным шифрам: нелинейные преобразования, большие S-блоки, и т.д. Больше всего мне нравится RC4, а затем SEAL. Мне бы очень хотелось увидеть результаты криптоанализа предложенных мной генераторов, объединяющих LFSR и FCSR. Эта область кажется весьма привлекательной для изучения возможности использования в реальных разработках. Или для получения потокового шифра можно использовать блочный шифр в режиме OFB или CFB.

В 14-й для сравнения приведены временные соотношения для некоторых алгоритмов.

Табл. 17-3.
Скорости шифрования нескольких потоковых шифров на i486SX/33 МГц

Алгоритм	Скорость шифрования (Мбайт/с)
A5	5
PIKE	62
RC4	164
SEAL	381

17.13 Генерация нескольких потоков из одного генератора псевдослучайной последовательности

Если нужно зашифровать несколько каналов связи при помощи одного блока - например, мультиплексора - простым решением является использование для каждого потока своего генератора псевдослучайной последовательности. При этом возникают две следующих проблемы: нужна дополнительная аппаратура, и все генераторы должны быть синхронизированы. Проще было бы использовать один генератор.

Одно из решений - тактировать генератор несколько раз. Если нужно три независимых потока, тактируйте генератор три раза и отправьте по одному биту в каждый поток. Этот метод работает, но могут быть сложности при получении большой частоты. Например, если вы можете тактировать генератор только в три раза быстрее тактирования потока данных, вы сможете создать только три потока. Другим способом является использование одной и той же последовательности для каждого канала, возможно с переменной временной задержкой. Это небезопасно.

Действительно удачная идея [1489], запатентованная NSA, показана на 6-й. Записывайте выход вашего любимого генератора в простой m-битовый сдвиговый регистр. По каждому тактовому импульсу сдвигайте регистр на один бит вправо. Затем для каждого выходного потока выполните AND регистра с другим m-битовым вектором, рассматриваемым как уникальный идентификатор для выбранного выходного потока, затем объедините с помощью XOR все биты, получая выходной бит для этого потока. Если требуется получить параллельно несколько выходных потоков, для каждого выходного потока нужно использовать отдельный вектор и логический массив XOR/AND.

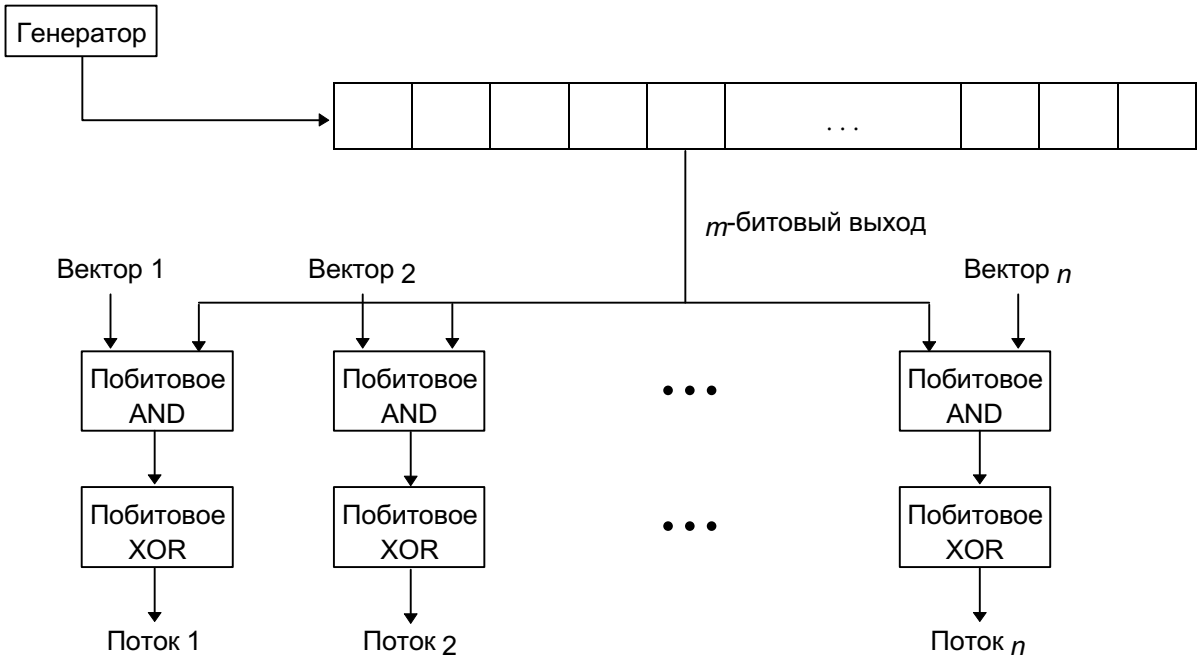


Рис. 17-11. Генератор нескольких битов.

Существует ряд вещей, которые нужно отслеживать. Если любой из этих потоков является линейной комбинацией других потоков, то система может быть взломана. Но если вы достаточно аккуратны, описанный способ является простым и безопасным способом решения проблемы.

17.14 Генераторы реальных случайных последовательностей

Иногда криптографически безопасные псевдослучайные последовательности недостаточно хороши. В криптографии вам могут понадобиться действительно случайные числа. Первое, что приходит в голову - это генерация ключей. Прекрасно можно генерировать случайные криптографические ключи, используя генератор псевдослучайных последовательностей, но если враг добудет копию этого генератора и главный ключ, он сможет создать те же ключи и взломать вашу криптосистему, независимо от надежности ваших алгоритмов. Последовательность, выдаваемую генератором случайных последовательностей, воспроизвести невозможно. Никто, даже вы сами, не сможет воспроизвести последовательность битов, выдаваемую этими генераторами.

Крупной философской проблемой является вопрос о том, дают ли эти методы действительно случайные биты. Я не собираюсь ввязываться в этот спор. Здесь я рассматриваю выдачу битов, которые невозможно воспроизвести, и у которых статистические свойства как у случайных битов.

Для любого генератора действительно случайных последовательностей важным вопросом является его проверка. На эту тему существует множество литературы. Тесты на случайность можно найти в [863, 99]. Маурер показал, что все эти тесты можно получить из попытки сжать последовательность [1031, 1032]. Если случайная последовательность сжимается, то она не является по настоящему случайной.

В любом случае, все, что мы имеем в этой области, во многом относится к черной магии. Главным моментом является генерация последовательности битов, которую не сможет угадать ваш противник. Это гораздо более трудная задача, чем кажется. Я не могу доказать, что любой из описанных методов генерирует случайные биты. Результатом их работы являются последовательности битов, которые невозможно легко воспроизвести. Подробности можно найти в [1375, 1376, 511].

Таблицы RAND

Давным давно, в 1955 году, когда компьютеры все еще были в новинку, Rand Corporation издала книгу, содержавшую миллион случайных цифр [1289]. Их метод описывался так:

> Случайные цифры этой книги были получены при помощи рандомизации основной таблицы, сгенерированной эле к-тронной рулеткой. Вкратце, источник импульсов, выдающий их со случайной частотой в среднем около 100000 импульсов в секунду, открывался раз в секунду импульсом постоянной частоты. Цепи нормализации импульса пропускали импульсы ч е-рез 5-разрядный бинарный счетчик. По сути машина являлась колесом рулетки с 32-позициями, которое в среднем делало около 3000 оборотов за выборку и выдавало одно число в секунду. Использовался двоично-десятичный преобразователь, к о-торый преобразовывал 20 из 32 чисел (оставшиеся двенадцать отбрасываются) и оставлял только последнюю цифру двузна ч-ных чисел. Эти последние цифры попадали в компостер IBM, образуя в конце концов таблицу пробитых карточек случайных цифр.

В книге рассматривались и результаты различных проверок данных на случайность. В ней также предлагался способ, как использовать эту книгу для выбора случайного числа:

> Строки таблицы цифр нумеруются от 00000 до 19999. При использовании таблицы нужно сначала выбрать случайную стартовую позицию. Обычной процедурой для этого является следующее: откройте эту книгу на произвольной странице та б-лицы цифр и, закрыв глаза, выберите пятиразрядное число. Это число после замены первой цифры остатком от деления ее на 2 определяет стартовую строку. Остаток от деления двух цифр справа от первоначально выбранного пятиразрядного числа на 50 задает стартовый столбец в стартовой строке. Чтобы защититься от открытия книги все время на одной странице и естес т-венного стремления выбрать число поближе к центру страницы, каждое использованное для определения стартовой позиции пятиразрядное число должно быть помечено и не должно больше использоваться для этой ц ели.

Главным содержанием этой книги была "Таблица случайных цифр". Цифры приводились пяти разрядными группами - "10097 32533 76520 13586 . . ." - по 50 в строке и по пятьдесят строк на странице. Таблица занимала 400 страниц и, за исключением особенно выдающейся группы на странице 283, выглядевшей как "69696", была достаточно скучным чтивом. В книгу также входила таблица 100000 нормальных отклонений.

Интересным в книге RAND являются не миллионы случайных цифр, а то, что они были созданы до компь ю-терной революции. Во многих криптографических алгоритмах используются произвольные константы - так н а-зываемые "магические числа". Выбор магических чисел из таблиц RAND гарантировал, что они не были в ы-браны специально по каким-то жульническим причинам. Так, например, было сделано в Khafre.

Использование случайного шума

Лучшим способом получить большое количество случайных битов является извлечение их из естественной случайности реального мира. Часто такой метод требует специальной аппаратуры, но этот трюк можно прим е-нить и в компьютерах.

Найдите событие, которое случается регулярно, но случайно: атмосферный шум, преодолевающий какой-то порог, ребенок, падающий, учась ходить. Измерьте интервал между одним подобным событием и событием, следующим за ним. Запишите. Измерьте временной интервал между вторым и третьим событиями. Снова запишите. Если первый временной интервал больше второго, выходным битом будет 1. Если второй интервал больше первого, то выходом события будет 0. Сделайте это снова для следующего соб ытия.

Бросьте стрелу дартс в перечень котировок Нью-Йоркской фондовой бирже в местной газете. Сравните котировку акции, в которую вы попали, с котировкой акции прямо над ней. Если больше та, в которую вы попали, выход равен 0, а если меньше - 1.

Подключите к компьютеру счетчик Гейгера, подсчитайте количество импульсов за фиксированный интервал времени и возьмите младший бит. Или измерьте время между последовательными тиками ticks. (Так как радио-активный источник распадается, среднее время между последовательными тиками непрерывно увеличивается. Чтобы этого избежать, надо выбирать источник с достаточно длинным периодом полураспада - такой как пл у-тоний. Если вы беспокоитесь о своем здоровье, можете внести соответствующие статистические поправки.)

Дж. Б. Эгнью (G. B. Agnew) предложил генератор реально случайных битов, который можно интегрировать в СБИС [21]. Это конденсатор металл-изолятор-полупроводник (metal insulator semiconduction capacitor, MISC). Два таких конденсатора помещаются рядом друг с другом, а случайный бит является функцией разности зар я-дов этих конденсаторов. Другой генератор случайных чисел генерирует поток случайных битов, используя н е-стабильность частоты свободно колеблющегося осциллятора [535]. Коммерческая микросхема от AT&T генери-рует случайные числа, опираясь именно на это явление [67]. М. Гьюд (M. Gude) построил генератор случайных чисел, собирающий случайные биты из физических явлений, например, радиоактивного распада [668, 669]. Манфилд Рихтер (Manfield Richter) разработал генератор случайных чисел на базе температурного шума пол у-проводникового диода [1309].

Предположительно случайны временные интервалы между последовательными 2e4 излучениями света рас-падающегося атома ртути. Используйте. А лучше найдите полупроводниковую фирму, которая изготавливает микросхемы генераторов случайных чисел, их достаточно много.

Существует также генератор случайных чисел, использующий диск компьютера [439]. Он измеряет время, нужное для чтения блока диска, и использует изменения этого времени в качестве источника случайных чисел. Данные фильтруются, чтобы удалить структуру, вызванную квантованием, затем к векторам чисел применяется быстрое преобразование Фурье. Это устраняет смещение и корреляцию. Наконец, в качестве случайных битов используются спектральные углы для частот в диапазоне $(0, \pi)$, нормализованные на единичный интервал.

Большая часть изменений скорости вращения диска вызвана турбулентностью воздуха, которая и является источником случайности в системе. Хотя надо учесть следующее. Если вы выдаете на выход слишком много битов, то вы используете в качестве генератора случайных чисел быстрое преобразование Фурье и рискуете получить определенную предсказуемость. И лучше снова и снова читать один и тот же дисковый блок, чтобы вам не пришлось фильтровать структуру, источником которой является планировщик диска. Реализация такой системы позволяла получать около 100 битов в минуту [439].

Использование таймера компьютера

Если вам нужен один случайный бит (или даже несколько), воспользуйтесь младшим значащим битом любого регистра таймера. В системе UNIX он может быть не слишком случайным из-за различной возможной синхронизации, но на некоторых персональных компьютерах это работает.

Не стоит извлекать таким образом слишком много битов. Выполнение много раз одной и той же процедуры последовательно может легко сместить биты, генерированные этим способом. Например, если выполнение каждой процедуры генерации бита занимает четное число тиков таймера, на выходе вашего генератора будет бесконечная последовательность одинаковых битов. Если выполнение каждой процедуры генерации бита занимает нечетное число тиков таймера, на выходе вашего генератора будет бесконечная последовательность череду ющихся битов. Даже если зависимость не так очевидна, получающийся битовый поток будет далек от случайного. Один генератор случайных чисел работает следующим образом [918]:

> Наш генератор действительно случайных чисел . . . работает, устанавливая будильник и затем быстро инкрементируя регистр счетчика процессора до тех пор, пока не произойдет прерывание. Далее выполняется XOR содержимого регистра и содержимого байта выходного буфера (данные регистра усекаются до 8 битов). После того, как будет заполнен каждый байт выходного буфера, буфер подвергается дальнейшей обработке циклическим сдвигом каждого символа вправо на два бита. Это приводит к эффекту перемещения наиболее активных (и случайных) младших значащих битов в старшие значащие позиции. Затем весь процесс повторяется три раза. Наконец после прерываний два самых случайных бита регистра счетчика повлияют на каждый символ буфера. То есть происходит $4n$ прерываний, где n - число нужных случайных битов.

Этот метод очень чувствителен к случайности системных прерываний и квантованности таймера. При тестировании на реальных UNIX-машинах результат был очень неплох.

Измерение скрытого состояния клавиатуры

Процесс печатания и случаен, и неслучаен. Он достаточно неслучаен, чтобы его можно было использовать для идентификации печатающего человека, но он достаточно случаен, чтобы его можно было использовать для генерации случайных битов. Измерьте время между последовательными нажатиями клавиш, затем воспользуйтесь младшими значащими битами этих измерений. Эти биты оказываются достаточно случайными. Этот метод не работает на UNIX-терминалах, так как нажатия клавиш прежде, чем они будут переданы вашей программе, проходят через фильтры и другие механизмы, но это будет работать на большинстве персональных компьютеров.

В идеале вы должны по каждому нажатию клавиши генерировать только один бит. Использование большего количества битов может сместить результаты в зависимости от навыков машинистки. Однако этот метод имеет ряд ограничений. Хотя нетрудно посадить за клавиатуру человека, печатающего со скоростью 100 слов в минуту или около того, если есть время для генерации ключа, глупо просить машинистку печатать текст из 100000 слов, чтобы использовать результат работы генератора в качестве одноразового блокнота.

Смещения и корреляции

Главной проблемой подобных систем являются возможные закономерности в генерируемой последовательности. Используемые физические процессы могут быть случайны, но между физическим процессом и компьютером находятся различные измерительные инструменты. Эти инструменты могут легко привести к появлению проблем.

Способом устранить **смещение**, или отклонение, является XOR нескольких битов друг с другом. Если случайный бит смещен к 0 на величину e, то вероятность 0 можно записать как:

$P(0) = 0.5 + e$

XOR двух из таких битов дает:

$P(0) = (0.5 + e)^2 + (0.5 - e)^2 = 0.5 + 2e^2$

Те же вычисления для XOR 4 битов дают:

$P(0) = 0.5 + 8e^4$

XOR m битов экспоненциально сходится к равной вероятности 0 и 1. Если известно максимальное смещение, которое допустимо в вашем приложении, вы можете вычислить, сколько битов вам нужно объединить с помощью XOR, чтобы уменьшить смещение до этого значения.

Еще лучше рассматривать биты попарно. Если 2 бита одинаковы отбросьте их и взгляните на следующую пару. Если 2 бита различны, используйте первый бит в качестве выхода генератора. Это полностью устраняет смещение. Другие методы уменьшения смещения используют распределение переходов сжатие и быстрое пр е-образование Фурье [511].

Потенциальной проблемой обоих методов является то, что при наличии **корреляции** между соседними битами эти методы увеличивают смещение. Одним из способов исправить это является использование нескольких случайных источников. Возьмите четыре случайных источника и выполните XOR битов друг с другом или возьмите два случайных источника и взгляните на их биты попарно.

Например, возьмите радиоактивный источник и присоедините счетчик Гейгера к вашему компьютеру. Возьмите пару шумящих диодов и записывайте в качестве события каждое превышение определенного значения. Измерьте атмосферный шум. Извлеките из каждого источника случайный бит и выполните их XOR друг с другом, получая случайный бит. Возможности бесконечны.

Одно то, что генератор случайных чисел смещен не обязательно означает его бесполезность. Это только означает, что он менее безопасен. Например, рассмотрим проблему Алисы, генерирующей 168-битовый ключ для тройного DES. А все, что у нее есть, - это генератор случайных битов со смещением к 0 : с вероятностью 55 процентов он выдает нули и с вероятностью 45 процентов - единицы. Это означает, что энтропия на бит ключа с оставит только 0.99277 (для идеального генератора она равна 1). Мэллори, пытаясь раскрыть ключ, может оптимизировать выполняемое вскрытие грубой силой, проверяя сначала наиболее вероятные ключи (000 . . . 0) и двигаясь к наименее вероятному ключу (111 . . . 1). Из-за смещения Мэллори может ожидать, что ему удастся обнаружить ключ за 2^{109} попыток. При отсутствии смещения Мэллори потребуется 2^{111} попыток. Полученный ключ менее безопасен, но это практически неощутимо.

Извлеченная случайность

В общем случае лучший способ генерировать случайные числа - найти большое количество кажущихся сл у-чайными событий и извлечь случайность из них. Эта случайность может храниться в накопителе и извлекаться при необходимости. .Однонаправленные хэш-функции прекрасно подходят для этого. Они быстры, поэтому вы можете пропускать биты через них, не слишком заботясь о производительности или действительной случайн о-сти каждого наблюдения. Попробуйте хэшировать почти все, что вам кажется хоть чуть-чуть случайным. Н а-пример:

— Копия каждого нажатия на клавиши

— Команды мыши

— Номер сектора, время дня и задержка поиска для каждой дисковой операции

— Действительное положение мыши

— Номер текущей строки развертки монитора

— Содержание действительно выводимого на экран изображения

— Содержание FAT-таблиц, таблиц ядра, и т.д.

— Времена доступа/изменения /dev/tty

— Загрузка процессора

— Времена поступления сетевых пакетов

— Выход микрофона

— /dev/audio без присоединенного микрофона

Если ваша система использует различные кристаллы-осцилляторы для своего процессора и часов, попытайтесь считывать время дня в плотном цикле. В некоторых (но не всех) системах это приведет к случайным кол е-баниям фазы между двумя осцилляторами.

Так как случайность в этих событиях определяется синхронизацией осцилляторов, используйте часы с как можно меньшим квантом времени. В стандартном PC используется микросхема таймера Intel 8254 (или эквивалентная), работающая на тактовой частоте 1.1931818 МГц, поэтому непосредственное считывание регистра счетчика даст разрешение в 838 наносекунд. Чтобы избежать смещения результатов, не используйте в качестве источника событий прерывание таймера. Вот как выглядит этот процесс на языке C с MD5 (см. раздел 18.5) в качестве хэш-функции:

```
char Randpool[16];

/* Часто вызывается для широкого множества случайных или полуслучайных системных событий для to
churn the randomness pool . Точный формат и длина randevent не имеет значения, пока его содержание
```

```
является в некоторой мере чем-то непредсказуемым. */
   void churnrand(char *randevent,unsigned lnt randlen) {
        MD5_CTX md5;
        MD5Init(&md5);
        MD5Update(&md5, Randpool , sizeof(Randpool));
        MD5Update(&md5 , randevent , randlen );
        MD5Final(Randpool,&md5);
   }
```

После достаточных вызовов churnrand() накопления достаточной случайности в Randpool, можно генерировать из этого случайные биты. MD5 снова становится полезной, в этот раз в качестве генератора псевдослуча йного байтового потока, работающего в режиме счетчика.

```
long Randcnt;
void genrand(char *buf,unsigned int buflen) {
     MD5_CTX md5;
     char tmp[16];
     unsigned int n;
     while(buflen != 0) {
             /* Пул хэшируется счетчиком */
             MD5Init(&md5);
             MD5Update(&md5, Randpool, sizeof(Randpool));
             MD5Update(&md5,(unsigned char *)&Randcnt,sizeof(Randcnt));
             MD5Final(tmp,&md5);
             Randcnt++; /* Инкрементируем счетчик */
             /*Копируем 16 или запрошенное число байтов, если оно меньше 16, в буфер
             пользователя*/
             n = (buflen < 16) ? buflen : 16;
             memcpy(buf, tmp, n);
             buf += n ;
             buflen -= n;
     }
}
```

По многим причинам хэш-функция имеет ключевое значение. Во первых она обеспечивает простой способ генерировать произвольное количество псевдослучайных данных, не вызывая всякий раз churnrand(). На деле, когда запас в накопителе подходит к концу, система постепенно переходит от совершенной случайности к пра к-тической. В этом случае становится *теоретически* возможным использовать результат вызова genrand() для определения предыдущего или последующего результата. Но для этого потребуется инвертировать MD5, что вычислительно невозможно.

Это важно, так как процедуре неизвестно, что делается потом со случайными данными, которые она возвр ащает. Один вызов процедуры может генерировать случайное число для протокола, которое посылается в явном виде, возможно в ответ на прямой запрос взломщика. А следующий вызов может генерировать секретный ключ для совсем другого сеанса связи, в суть которого и хочет проникнуть взломщик. Очевидна важность того, чтобы взломщик не смог получить секретный ключ, используя подобную схему действий.

Но остается одна проблема. Прежде, чем в первый раз будет вызвана genrand() в массиве Randpool[] должно быть накоплено достаточно случайных данных. Если система какое-то время работала с локальным пользовате лем, что-то печатающим на клавиатуре, то проблем нет. Но как насчет независимой системы, которая перегр ужается автоматически, не обращая внимания ни на какие данные клавиатуры или мыши ?

Но есть одна трудность. В качестве частичного решения можно потребовать, чтобы после самой первой з агрузки оператор какое-то время поработал на клавиатуре и создал на диске стартовый файл перед выгрузкой операционной системы, чтобы в ходе перезагрузок использовались случайные данные, переданные в Randseed[]. Но не сохраняйте непосредственно сам Randseed[]. Взломщик, которому удастся заполучить этот файл, сможет определить все результаты genrand() после последнего обращения к churnrand() прежде, чем этот файл будет создан.

Решением этой проблемы является хэширование массива Randseed[] перед его сохранением, может даже вызовом genrandO. При перезагрузке системы вы считываете данные из стартового файла, передаете их

churnrand(), а затем немедленно стираете их. К сожалению это не устраняет угрозы того, что злоумышленник добудет файл между перезагрузками и использует его для предсказания будущих значений функции genrand(). Я не вижу иного решения этой проблемы кроме, как подождать накопления достаточного количества случайных событий, случившихся после перезагрузки, прежде, чем позволить genrand() выдавать результаты.

Глава 18
Однонаправленные хэш-функции

18.1 Основы

Однонаправленная функция $H(M)$ применяется к сообщению произвольной длины M и возвращает значение фиксированной длины h.

$h = H(M)$, где h имеет длину m

Многие функции позволяют вычислять значение фиксированной длины по входным данным произвольной длины, но у однонаправленных хэш-функций есть дополнительные свойства, делающие их однонаправленными [1065]:

Зная M, легко вычислить h.

Зная H, трудно определить M, для которого $H(M)=h$.

Зная M, трудно определить другое сообщение, M', для которого $H(M)= H(M')$.

Если бы Мэллори умел делать трудные вещи, он смог бы разрушить безопасность любого протокола, и с-пользующего однонаправленную хэш-функцию. Смысл однонаправленных хэш-функций и состоит в обеспеч е-нии для M уникального идентификатора ("отпечатка пальца"). Если Алиса подписала M с помощью алгоритма цифровой подписи на базе $H(M)$, а Боб может создать M', другое сообщение, отличное от M, для которого $H(M)= H(M')$, то Боб сможет утверждать, что Алиса подписала M'.

В некоторых приложениях однонаправленности недостаточно, необходимо выполнение другого требования, называемого **устойчивостью к столкновениям**.

Должно быть трудно найти два случайных сообщения, M и M', для которых $H(M)= H(M')$.

Помните вскрытие методом дня рождения из раздела 7.4? Оно основано не на поиске другого сообщения M', для которого $H(M)= H(M')$, а на поиске двух случайных сообщений, M и M', для которых $H(M)= H(M')$.

Следующий протокол, впервые описанный Гидеоном Ювалом (Gideon Yuval) [1635], показывает, как, если предыдущее требование не выполняется, Алиса может использовать вскрытие методом дня рождения для обм а-на Боба.

(1) Алиса готовит две версии контракта: одну, выгодную для Боба, и другую, приводящую его к банкротству

(2) Алиса вносит несколько незначительных изменений в каждый документ и вычисляет хэш-функции. (Этими изменениями могут быть действия, подобные следующим: замена ПРОБЕЛА комбинацией ПРО-БЕЛ-ЗАБОЙ-ПРОБЕЛ, вставка одного-двух пробелов перед возвратом каретки, и т.д. Делая или не делая по одному изменению в каждой из 32 строк, Алиса может легко получить 2^{32} различных документов.)

(3) Алиса сравнивает хэш-значения для каждого изменения в каждом из двух документов, разыскивая пару, для которой эти значения совпадают. (Если выходом хэш-функции является всего лишь 64-разрядное зн а-чение, Алиса, как правило, сможет найти совпадающую пару сравнив 2^{32} версий каждого документа.) Она восстанавливает два документа, дающих одинаковое хэш-значение.

(4) Алиса получает подписанную Бобом выгодную для него версию контракта, используя протокол, в котором он подписывает только хэш-значение.

(5) Спустя некоторое время Алиса подменяет контракт, подписанный Бобом, другим, который он не подпис ы-вал. Теперь она может убедить арбитра в том, что Боб подписал другой контракт.

Это заметная проблема. (Одним из советов является внесение косметических исправлений в подписываемый документ.)

При возможности успешного вскрытия методом дня рождения, могут применяться и другие способы вскр ы-тия. Например, противник может посылать системе автоматического контроля (может быть спутниковой) сл у-чайные строки сообщений со случайными строками подписей. В конце концов подпись под одним из этих сл у-чайных сообщений окажется правильной. Враг не сможет узнать, к чему приведет эта команда, но, если его единственной целью является вмешательство в работу спутника, он своего добьется.

Длины однонаправленных хэш-функций

64-битовые хэш-функции слишком малы, чтобы противостоять вскрытию методом дня рождения. Более практичны однонаправленные хэш-функции, выдающие 128-битовые хэш-значения. При этом, чтобы найти два

документа с одинаковыми хэш-значениями, для вскрытия методом дня рождения придется хэшировать 2^{64} случайных документов, что, впрочем, недостаточно, если нужна длительная безопасность. NIST в своем Стандарте безопасного хэширования (Secure Hash Standard, SHS), использует 160-битовое хэш-значение. Это еще сильнее усложняет вскрытие методом дня рождения, для которого понадобится 2^{80} хэширований.

Для удлинения хэн-значений, выдаваемых конкретной хэш-функцией, был предложен следующий метод.

(1) Для сообщения с помощью одной из упомянутых в этой книге однонаправленных хэш-функций генерируется хэш-значение.

(2) Хэш значение добавляется к сообщению.

(3) Генерируется хэш-значение объединения сообщения и хэш-значения этапа (1).

(4) Создается большее хэш-значение, состоящее из объединения хэш-значения этапа (1) и хэш-значения этапа (3).

(5) Этапы (1)-(4) повторяются нужное количество раз для обеспечения требуемой длины хэш-значения.

Хотя никогда не была доказана безопасность или небезопасность этого метода, уряд людей этот метод вызывает определенные сомнения [1262,859].

Обзор однонаправленных хэш-функций

Не легко построить функцию, вход которой имеет произвольный размер, а тем более сделаьть ее однонаправленной. В реальном мире однонаправленные хэш-функции строятся на идее **функции сжатия**. Такая однонаправленная функция выдает хэш-значение длины n при заданных входных данных большей длины m [1069, 414]. Входами функции сжатия являются блок сообщения и выход предыдущего блока текста (см. 17-й). Выход представляет собой хэш-значение всех блоков до этого момента. То есть, хэш-значение блока M_i равно

$h_i = f(M_i, h_{i-1})$

Это хэш-значение вместе со следующим блоком сообщения становится следующим входом функции сжатия. Хэш-значением всего сообщения является хэш-значение последнего блока.

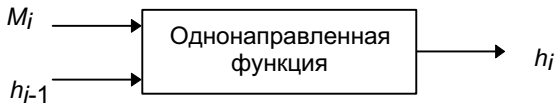


Рис. 18-1. Однонаправленная функция

Хэшируемый вход должен каким-то способом содержать бинарное представление длины всего сообщения. Таким образом преодолевается потенциальная проблема, вызванная тем, что сообщения различной длины могут давать одно и то же хэш-значение [1069, 414]. Иногда такой метод называется **MD-усилением** [930].

Различные исследователи выдвигали предположения, что если функция сжатия безопасна, то этот метод хэширования исходных данных произвольной длины также безопасен - но ничего не было доказано [1138, 1070, 414].

На тему проектирования однонаправленных хэш-функций написано много. Более подробную математическую информацию можно найти [1028, 793, 791, 1138, 1069, 414, 91, 858, 1264]. Возможно самым толковой интерпретацией однонаправленных хэш-функций являются тезисы Барта Пренела (Bart Preneel) [1262].

18.2 Snefru

Snefru - это однонаправленная хэш-функция, разработанная Ральфом Мерклом [1070]. (Snefru, также как Khufu и Khafre, был египетским фараоном.) Snefru хэширует сообщения произвольной длины, превращая их в 128-битовые 256-битовые значения.

Сначала собщение разбивается на кусочки длиной по 512-m. (Переменная m является длитной хэш-значения.) Если выход - это 128-битовое значение, то длина кусочков равна 384 битам, а если выход - 128-битовое значение, то длина кусочков - 256 битов.

Сердцем алгоритма служит функция H, хэширующая 512-битовое значение в m-битовое. Первые m битов выхода H являются хэш-значением блока, остальные отбрасываются. Следующий блок добавляется к хэш-значению предыдущего блока и снова хэшируется. (К первоначальному блоку добавляется строка нулей.) После последнего блока (если сообщение состоит не из целого числа блоков, последний блок дополняется нулями) первые m битов добавляются к бинарному представлению длины сообщения и хэшируются последний раз.

Функция H основывается на E, обратимой функции блочного шифрования, работающей с 512 битовыми

блоками. H - это последние *m* битов выхода E, объединенные посредством XOR с первыми *m* битами входа E.

Безопасность Snefru опирается на функцию E, которая рандомизирует данные за несколько проходов. Каждый проход состоит из 64 рандомизирующих этапов. В каждом этапе в качестве входа S-блока используется другой байт данных. Выходное слово подвергается операции XOR с двумя соседними словами сообщения. Построение S-блоков аналогично построению S-блоков в Khafre (см. раздел 13.7). Кроме того, выполняется ряд циклических сдвигов. Оригинальный Snefru состоял из двух проходов.

Криптоанализ Snefru

Используя дифференциальный криптоанализ, Бихам и Шамир показали небезопасность двухпроходного Snefru (с 128-битовым хэш-значением) [172]. Их способ вскрытия за несколько минут обнаруживает пару сообщений с одинаковым хэш-значением.

Для 128-битового Snefru их вскрытия работают лучше, чем вскрытие грубой силой для четырех и менее проходов. Вскрытие Snefru методом дня рождения требует 2^{64} операций; дифференциальный криптоанализ может найти пару сообщений с одинаковым хэш-значением за $2^{28.5}$ операций для трехпроходного Snefru и за $2^{44.5}$ операций для четырехпроходного Snefru. Нахождение сообщения, хэш-значение которого совпадает с заданным, при использовании грубой силы требует 2^{128} операций, при дифференциальном криптоанализе для этого нужно 2^{56} операций для трехпроходного и 2^{88} операций для четырехпроходного Snefru.

Хотя Бихам и Шамир не анализировали 256-битовые хэш-значения, они провели анализ вплоть до 224-битовых хэш-значений. В сравнении с вскрытием методом дня рождения, требующим 2^{112} операций они могут найти сообщения с одинаковым хэш-значением за $2^{12.5}$ операций для двухпроходного Snefru, за 2^{33} операций для трехпроходного Snefru и за для 2^{81} операций для четырехпроходного Snefru.

В настоящее время Меркл рекомендует использовать Snefru по крайней мере с восемью проходами [1073]. Однако с таким количеством проходов алгоритм становится намного медленнее, чем MD5 или SHA.

18.3 *N*-хэш

N-хэш - это алгоритм, придуманный в 1990 году исследователями Nippon Telephone and Telegraph, теми же людьми, которые изобрели FEAL [1105, 1106]. *N*-хэш использует 128-битовые блоки сообщения, сложную рандомизирующую функцию, похожую на FEAL, и выдает 128-битовое хэш-значение.

Хэш-значение каждого 128-битового блока является функцией блока и хэш-значения предыдущего блока.

$H_0 = I$, где I - случайное начальное значение

$H_i = g(M_i, H_{i-1}) \oplus M_i \oplus H_{i-1}$

Хэш-значение всего сообщения представляет собой хэш-значение последнего блока сообщения. Случайное начальное значение I может быть любым числом, определенным пользователем (даже одними нулями).

Функция g достаточно сложна. Схема алгоритма приведена на 16-й. Сначала переставляются левая и правая 64-битовые половины 128-битового хэш-значения предыдущего блока H_{i-1}, а затем выполняется XOR с повторяющимся шаблоном (128-битовым) и XOR с текущим блоком сообщения M_i. Далее это значение каскадно преобразуется в N (на рисунках N= 8) стадий обработки. Другим входом стадии обработки является предыдущее хэш-значение, подвергнутое XOR с одной из восьми двоичных констант.

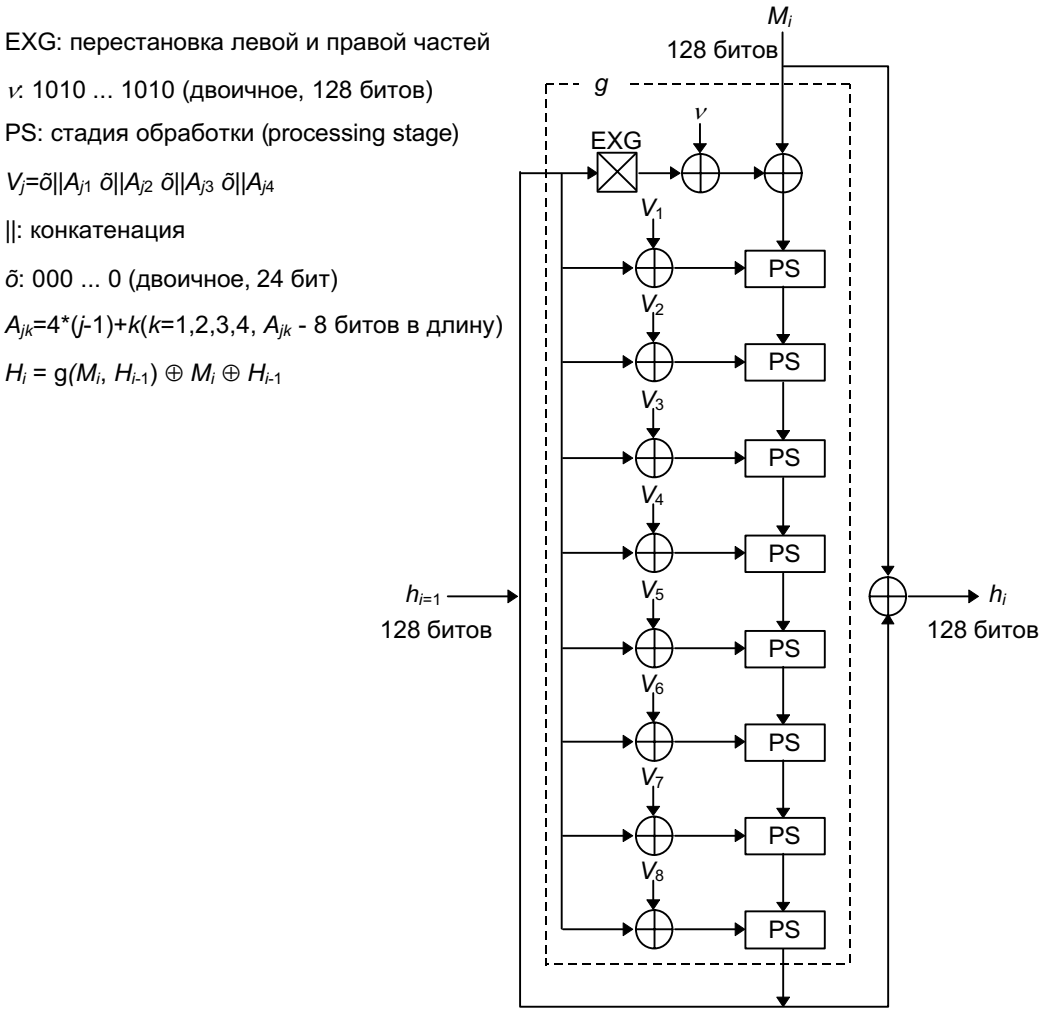


Рис. 18-2. Схема *N*-хэш.

Одна стадия обработки показана на 15-й. Блок сообщения разбивается на четыре 32-битовых значения . Предыдущее хэш-значение также разбивается на четыре 32-битовых значения . Функция f представлена на 14th. Функции S_0 и S_1 те же самые, что и в FEAL.

$S_0(a,b)$ = циклический сдвиг влево на два бита ((a + b) mod 256)

$S_1(a,b)$ = циклический сдвиг влево на два бита((a + b + 1) mod 256)

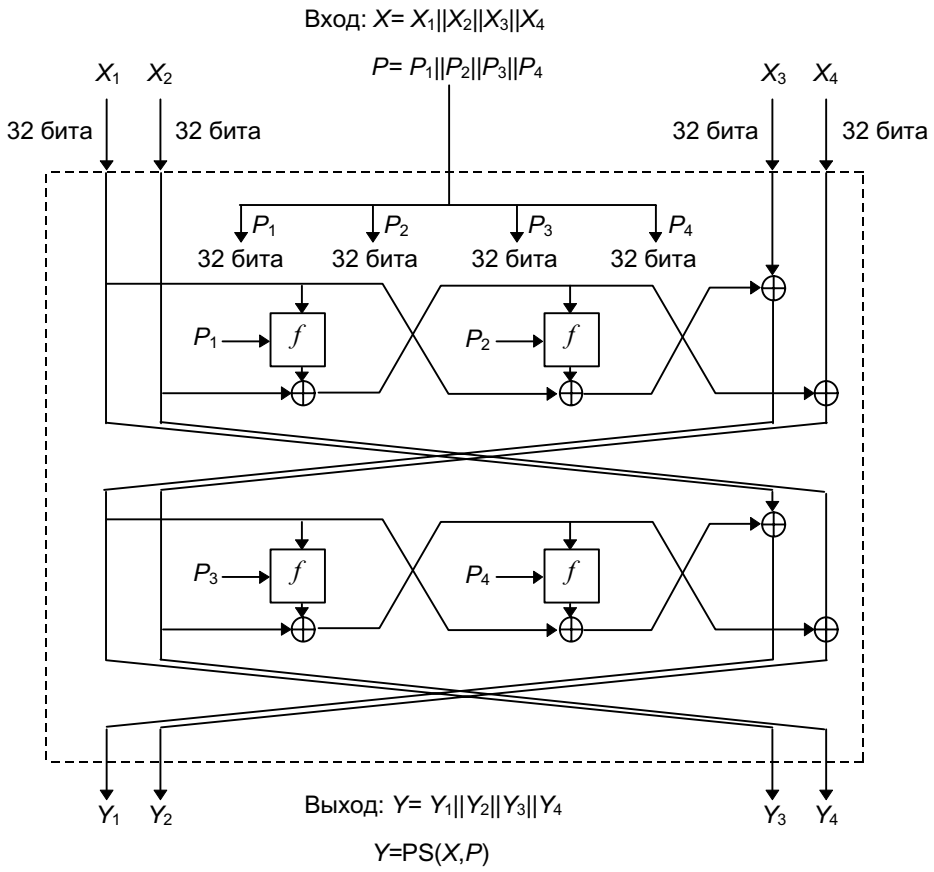
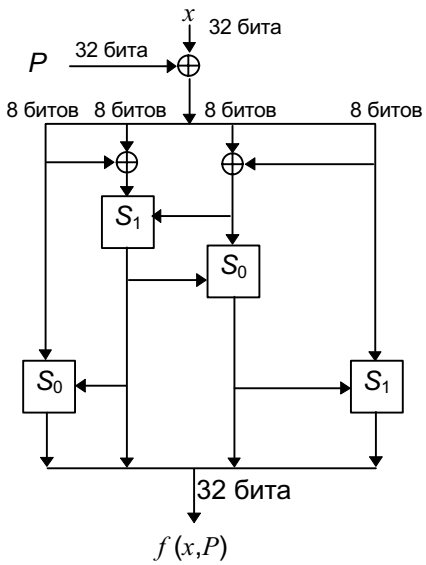


Рис. 18-3. Одна стадия обработки *N*-хэш.

Выход одной стадии обработки становится входом следующей стадии обработки . После последней стадии обработки выполняется XOR выхода с M_i и H_{i-1}, а затем к хэшированию готов следующий блок .



$Y=S_0(X_1,X_2)$=Rot2((X_1+X_2) mod 256)
$Y=S_1(X_1,X_2)$=Rot2((X_1+X_2+1) mod 256)
Y: выходные 8 битов, X_1,X_2 (8 битов): входы
Rot2(*Y*): циклический сдвиг влево на 2 бита 8-битовых данных *Y*

Рис. 18-4. Функция *f*.

Криптоанализ N-хэш

Берт ден Боер (Bert den Boer) открыл способ создавать столкновения в функции этапа *N*-хэш [1262]. Бихам и Шамир применили дифференциальный криптоанализ для вскрытия 6-этапной *N*-хэш [169, 172]. Конкретное выполненное ими вскрытие (конечно же, могли быть и другие) работает для любого *N*, делящегося на 3, и эффективнее вскрытия методом дня рождения для любого *N*, меньшего 15.

То же самое вскрытие может обнаруживать пары сообщений с одинаковым хэш-значением для 12-этапной *N*-хэш за 2^{56} операций (для вскрытия грубой силой нужно 2^{64} операций). *N*-хэш с 15 этапами безопасна по отношению к дифференциальному криптоанализу: для вскрытия потребуется 2^{72} операций.

Разработчики алгоритма рекомендуют использовать *N*-хэш не меньше, чем с 8 этапами [1106]. С учетом доказанной небезопасности *N*-хэш и FEAL (и ее скорости при 8 этапах) я рекомендую полностью отказаться от этого алгоритма.

18.4 MD4

MD4 - это однонаправленная хэш-функция, изобретенная Роном Ривестом [1318, 1319, 1321]. MD обозначает Message Digest (краткое изложение сообщения), алгоритм для входного сообщения выдает 128-битовое хэш-значение, или краткое изложение сообщения.

В [1319] Ривест описал цели, преследуемые им при разработке алгоритма:

Безопасность. Вычислительно невозможно найти два сообщения с одинаковым хэш-значением. Вскрытие грубой силой является самым эффективным.

Прямая безопасность. Безопасность MD4 не основывается на каких-либо допущениях, например, предположении о трудности разложения на множители.

Скорость. MD4 подходит для высокоскоростных программных реализаций. Она основана на простом наборе битовых манипуляций с 32-битовыми операндами.

Простота и компактность. MD4 проста, насколько это возможна, и не содержит больших структур данных или сложных программных модулей.

Удачна архитектура. MD4 оптимизирована для микропроцессорной архитектуры (особенно для микропроцессоров Intel), для более крупных и быстрых компьютеров можно выполнить любые необходимые изменения.

После первого появления алгоритма Берт ден Боер и Антон Босселаерс (Antoon Bosselaers) достигли успеха при криптоанализе последних двух из трех этапов алгоритма [202]. Ральфу Мерклу совершенно независимо удалось вскрыть первые два этапа [202]. Эли Бихам рассмотрел использование дифференциального криптоанализа против первых двух этапов MD4 [159]. Хотя все эти вскрытия не были распространены на полный алгоритм, Ривест усилил свою разработку. В результате появилась MD5.

18.5 MD5

MD5 - это улучшенная версия MD4 [1386, 1322]. Хотя она сложнее MD4, их схемы похожи, и результатом MD5 также является 128-битовое хэш-значение.

Описание MD5

После некоторой первоначальной обработки MD5 обрабатывает входной текст 512-битовыми блоками, разбитыми на 16 32-битовых подблоков. Выходом алгоритма является набор из четырех 32-битовых блоков, которые объединяются в единое 128-битовое хэш-значение.

Во первых, сообщение дополняется так, чтобы его длина была на 64 бита короче числа, кратного 512. Этим дополнением является 1, за которой вплоть до конца сообщения следует столько нулей, сколько нужно. Затем, к результату добавляется 64-битовое представление длины сообщения (истинной, до дополнения). Эти два действия служат для того, чтобы длина сообщения была кратна 512 битам (что требуется для оставшейся части алгоритма), и чтобы гарантировать, что разные сообщения не будут выглядеть одинаково после дополнения. Инициализируются четыре переменных:

A = 0x01234567

B = 0x89abcdef

C = 0xfedcba98

D = 0x76543210

Они называются **переменными сцепления**.

Теперь перейдем к основному циклу алгоритма. Этот цикл продолжается, пока не исчерпаются 512-битовые блоки сообщения.

Четыре переменных копируются в другие переменные: *A* в *a*, *B* в *b*, *C* в *c* и *D* в *d*.

Главный цикл состоит из четырех очень похожих этапов (у MD4 было только три этапа). На каждом этапе 16 раз используются различные операции. Каждая операция представляет собой нелинейную функцию над тремя из *a*, *b*, *c* и *d*. Затем она добавляет этот результат к четвертой переменной, подблоку текста и константе. Далее результат циклически сдвигается вправо на переменное число битов и добавляет результат к одной из переменных *a*, *b*, *c* и *d*. Наконец результат заменяет одну из переменных *a*, *b*, *c* и *d*. См. 13-й и 12-й. Существуют четыре нелинейных функции, используемые по одной в каждой операции (для каждого этапа - другая функция).

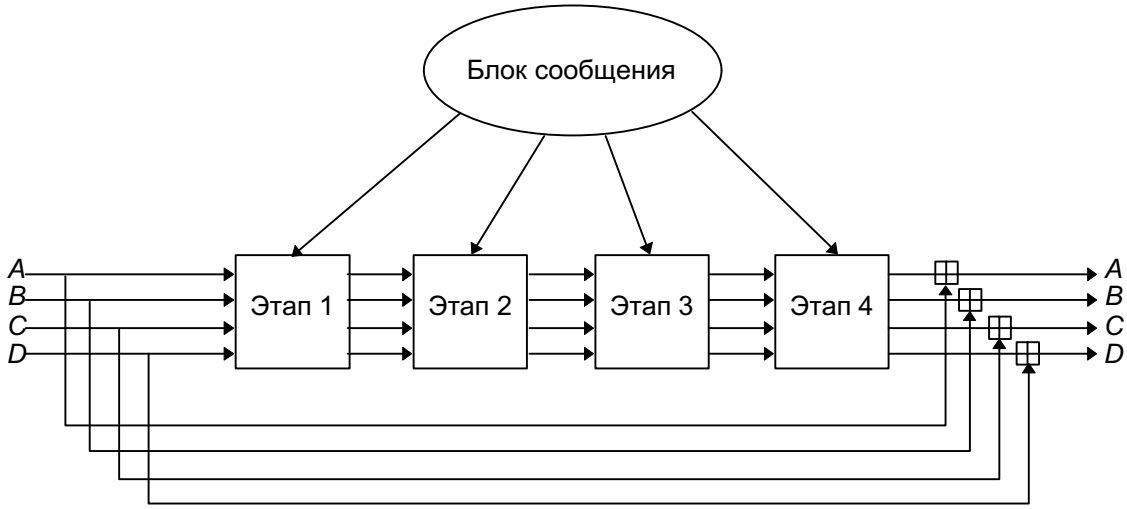


Рис. 18-5. Главный цикл MD5.

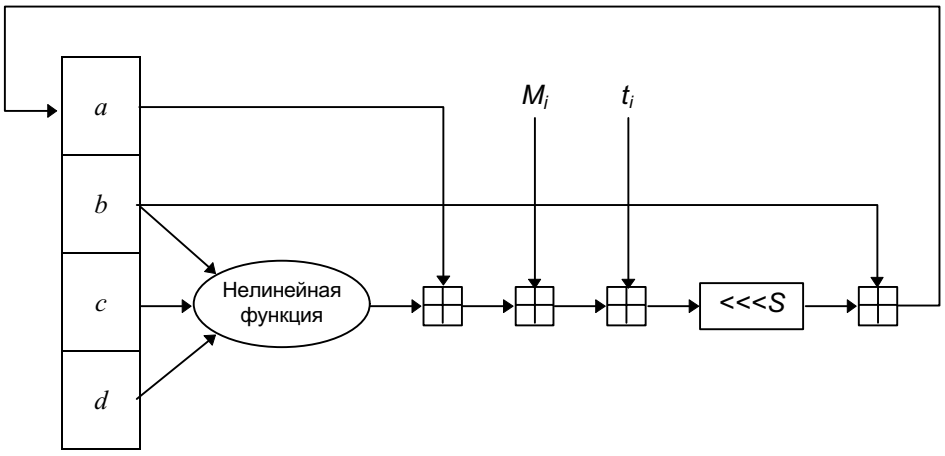


Рис. 18-6. Одна операция MD5.

$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$

$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$

$H(X,Y,Z) = X \oplus Y \oplus Z$

$I(X,Y,Z) = Y \oplus (X \vee (\neg Z))$

($\oplus$ - это XOR, $\wedge$ - AND, $\vee$ - OR, а $\neg$ - NOT.)

Эти функции спроектированы так, чтобы, если соответствующие биты *X*, *Y* и *Z* независимы и несмещены, каждый бит результата также был бы независимым и несмещенным. Функция F - это побитовое условие: если *X*, то *Y*, иначе *Z*. Функция H - побитовая операция четности.

Если M_j обозначает j-ый подблок сообщения (от 0 до 15), а $<<<s$ обозначает циклический сдвиг влево на *s*

битов, то используются следующие четыре операции:

FF(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + F(b,c,d) + M_j + t_i) <<< s)$

GG(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + G(b,c,d) + M_j + t_i) <<< s)$

HH(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + H(b,c,d) + M_j + t_i) <<< s)$

II(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + I(b,c,d) + M_j + t_i) <<< s)$

Четыре этапа (64 действия выглядят следующим образом):

Этап 1:

FF(a, b, c, d, M_0, 7, 0xd76aa478)

FF(d, a, b, c, M_1, 12, 0xe8c7b756)

FF(c, d, a, b, M_2, 17, 0x242070db)

FF(b, c, d, a, M_3, 22, 0xc1bdceee)

FF(a, b, c, d, M_4, 7, 0xf57c0faf)

FF(d, a, b, c, M_5, 12, 0x4787c62a)

FF(c, d, a, b, M_6, 17, 0xa8304613)

FF(b, c, d, a, M_7, 22, 0xfd469501)

FF(a, b, c, d, M_8, 7, 0x698098d8)

FF(d, a, b, c, M_9, 12, 0x8b44f7af)

FF(c, d, a, b, M_{10}, 17, 0xffff5bb1)

FF(b, c, d, a, M_{11}, 22, 0x895cd7be)

FF(a, b, c, d, M_{12}, 7, 0x6b901122)

FF(d, a, b, c, M_{13}, 12, 0xfd987193)

FF(c, d, a, b, M_{14}, 17, 0xa679438e)

FF(b, c, d, a, M_{15}, 22, 0x49b40821)

Этап 2:

GG(a, b, c, d, M_1, 5, 0xf61e2562)

GG(d, a, b, c, M_6, 9, 0xc040b340)

GG(c, d, a, b, M_{11}, 14, 0x265e5a51)

GG(b, c, d, a, M_0, 20, 0xe9b6c7aa)

GG(a, b, c, d, M_5, 5, 0xd62fl05d)

GG(d, a, b, c, M_{10}, 9, 0x02441453)

GG(c, d, a, b, M_{15}, 14, 0xd8ale681)

GG(b, c, d, a, M_4, 20, 0xe7d3fbc8)

GG(a, b, c, d, M_9, 5, 0x2,lelcde6)

GG(d, a, b, c, M_{14}, 9, 0xc33707d6)

GG(c, d, a, b, M_3, 14, 0xf4d50d87)

GG(b, c, d, a, M_8, 20, 0x455al4ed)

GG(a, b, c, d, M_{13}, 5, 0xa9e3e905)

GG(d, a, b, c, M_2, 9, 0xfcefa3f8)

GG(c, d, a, b, M_7, 14, 0x676f02d9)

GG(b, c, d, a, M_{12}, 20, 0x8d2a4c8a)

Этап 3:

HH(a, b, c, d, M_5, 4, 0xfffa3942)

HH(d, a, b, c, M_8, 11, 0x8771f681)

HH(c, d, a, b, M_{11}, 16, 0x6d9d6122)

HH(b, c, d, a, M_{14}, 23, 0xfde5380c)

HH(a, b, c, d, M_1, 4, 0xa4beea44)

HH(d, a, b, c, M_4, 11, 0x4bdecfa9)

HH(c, d, a, b, M_7, 16, 0xf6bb4b60)

HH(b, c, d, a, M_{10}, 23, 0xbebfbc70)

HH(a, b, c, d, M_{13}, 4, 0x289b7ec6)

HH(d, a, b, c, M_0, 11, 0xeaa127fa)

HH(c, d, a, b, M_3, 16, 0xd4ef3085)

HH(b, c, d, a, M_6, 23, 0x04881d05)

HH(a, b, c, d, M_9, 4, 0xd9d4d039)

HH(d, a, b, c, M_{12}, 11, 0xe6db99e5)

HH(c, d, a, b, M_{15}, 16, 0x1fa27cf8)

HH(b, c, d, a, M_2, 23, 0xc4ac5665)

Этап 4:

II(a, b, c, d, M_0, 6, 0xf4292244)

II(d, a, b, c, M_7, 10, 0x432aff97)

II(c, d, a, b, M_{14}, 15, 0xab9423a7)

II(b, c, d, a, M_5, 21, 0xfc93a039)

II(a, b, c, d, M_{12}, 6, 0x655b59c3)

II(d, a, b, c, M_3, 10, 0x8f0ccc92)

II(c, d, a, b, M_{10}, 15, 0xffeff47d)

II(b, c, d, a, M_1, 21, 0x85845ddl)

II(a, b, c, d, M_8, 6, 0x6fa87e4f)

II(d, a, b, c, M_{15}, 10, 0xfe2ce6e0)

II(c, d, a, b, M_6, 15, 0xa3014314)

II(b, c, d, a, M_{13}, 21, 0x4e081lal)

II(a, b, c, d, M_4, 6, 0xf7537e82)

II(d, a, b, c, M_{11}, 10, 0xbd3af235)

II(c, d, a, b, M_2, 15, 0x2ad7d2bb)

II(b, c, d, a, M_9, 21, 0xeb86d391)

Эти константы, t_i, выбирались следующим образом:

На i-ом этапе t_i является целой частью 2^{32}*abs(sin(i)), где i измеряется в радианах.

После всего этого a, b, c и d добавляются к A, B, C и D, соответственно, и алгоритм продолжается для следующего блока данных. Окончательным результатом служит объединение A, B, C и D.

Безопасность MD5

Рон Ривест привел следующие улучшения MD5 в сравнении с MD4 [1322]:

1. Добавился четвертый этап.
2. Теперь в каждом действии используется уникальная прибавляемая константа .

3. Функция G на этапе 2 с $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ была изменена на $(X \wedge Z) \vee (Y \wedge (\neg Z))$, чтобы сделать G менее симметричной.

4. Теперь каждое действие добавляется к результату предыдущего этапа. Это обеспечивает более быстрый лавинный эффект.

5. Изменился порядок, в котором использовались подблоки сообщения на этапах 2 и 3, чтобы сделать шаблоны менее похожими.

6. Значения циклического сдвига влево на каждом этапе были приближенно оптимизированы для ускорения лавинного эффекта. Четыре сдвига, используемые на каждом этапе, отличаются от значений, используемых на других этапах.

Том Берсон (Tom Berson) попытался применить дифференциальный криптоанализ к одному этапу MD5 [144], но его вскрытие не оказалось эффективным ни для одного из четырех этапов. Более успешное вскрытие ден Боера и Босселаерса, использующее функцию сжатия, привело к обнаружению столкновений в MD5 [203, 1331, 1336]. Само по себе это вскрытие невозможно для вскрытия MD5 в практических приложениях, оно не влияет и на использование MD5 в алгоритмах шифрования, подобных Luby-Rackoff (см. раздел 14.11). Успех этого вскрытия означает только, что одна из основных целей проектирования MD5- создать устойчивую к столкновениям функцию сжатия - не была достигнута. Хотя справедливо, что "кажется, что у функции сжатия есть слабое место, но это практически не влияет на безопасность хэш-функции " [1336], я отношусь к использованию MD5 очень осторожно.

18.6 MD2

MD2 - это другая 128-битовая однонаправленная хэш-функция, разработанная Роном Ривестом [801, 1335]. Она, вместе с MD5, используется в протоколах PEM (см. раздел 24.10). Безопасность MD2 опирается на случайную перестановку байтов. Эта перестановка фиксирована и зависит от разрядов π. S_0, S_1, S_2, . . . , S_{255} и являются перестановкой. Чтобы выполнить хэширование сообщения M:

(1) Дополните сообщение i байтами, значение i должно быть таким, чтобы длина полученного сообщения была кратна 16 байтам.

(2) Добавьте к сообщению 16 байтов контрольной суммы.

(3) Проинициализируйте 48-байтовый блок: X_0, X_1, X_2, . . ., X_{47}. Заполните первые 16 байтов X нулями, во вторые 16 байтов X скопируйте первые 16 байтов сообщения, а третьи 16 байтов X должны быть равны XOR первых и вторых 16 байтов X.

(4) Вот как выглядит функция сжатия:

$t = 0$

For $j = 0$ to 17

For $k = 0$ to 47

$t = X_t$ XOR S_t

$X_k = t$

$t = (t + j)$ mod 256

(5) Скопируйте во вторые 16 байтов X вторые 16 байтов сообщения, а третьи 16 байтов X должны быть равны XOR первых и вторых 16 байтов X. Выполните этап (4). Повторяйте этапы (5) и (4) по очереди для каждых 16 байтов сообщения.

(6) Выходом являются первые 16 байтов X.

Хотя в MD2 пока не было найдено слабых мест (см. [1262]), она работает медленнее большинства других предлагаемых хэш-функций.

18.7 Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA)

NIST, вместе с NSA, для Стандарта цифровой подписи (Digital Signature Standard, см. Раздел 20.2) разработал Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA) [1154 (Digital Signature Standard]. (Сам стандарт называется Стандарт безопасного хэширования (Secure Hash Standard, SHS), а SHA - это алгоритм, используемый в стандарте.) В соответствии с *Federal Register* [539]:

> Предлагается Федеральный стандарт обработки информации (Federal Information Processing Standard, FIPS) для Стандарта безопасного хэширования (Secure Hash Standard, SHS). Этот предложение определяет Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA) для использования вместе со Стандартом цифровой подписи (Digital Signature Standard) . . ..

> Кроме того, для приложений, в которых не требуется цифровая подпись, SHA должен использоваться во всех Федеральных приложениях, в которых понадобится алгоритм безопасного хэширования.

И

> Этот Стандарт определяет Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA), необходимый для обеспечения безопасности Алгоритма цифровой подписи (Digital Signature Algorithm, DSA). Для любого входного сообщения длиной меньше 2^{64} битов SHA выдает 160-битовый результат, называемый кратким содержанием сообщения. Далее, краткое содержание сообщения становится входом DSA, который вычисляет подпись для сообщения. Подписывание краткого содержания вместо всего сообщения часто повышает эффективность процесса, так как краткое содержание сообщения намного меньше, чем само сообщение. То же краткое содержание сообщения должно быть получено тем, кто проверяет подпись, если принятая им версия сообщения используется в качестве входа SHA. SHA называется безопасным, так как он разработан так, чтобы было вычислительно невозможно найти сообщение, соответствующее данному краткому содержанию сообщения или найти два различных сообщения с одинаковым кратким содержанием сообщения. Любые изменения, произошедшие при передаче сообщения, с очень высокой вероятностью приведут к изменению краткого содержания сообщения, и подпись не пройдет проверку. Принципы, лежащие в основе SHA, аналогичны использованным профессором Рональдом Л. Ривестом из MIT при проектировании алгоритма краткого содержания сообщения MD4 [1319]. SHA разработан по образцу упомянутого алгоритма.

SHA выдает 160-битовое хэш-значение, более длинное, чем у MD5.

Описание SHA

Во первых, сообщение дополняется, чтобы его длина была кратной 512 битам. Используется то же дополнение, что и в MD5: сначала добавляется 1, а затем нули так, чтобы длина полученного сообщения была на 64 бита меньше числа, кратного 512, а затем добавляется 64-битовое представление длины оригинального сообщения.

Инициализируются пять 32-битовых переменных (в MD5 используется четыре переменных, но рассматриваемый алгоритм должен выдавать 160-битовое хэш-значение):

A = 0x67452301

B = 0xefcdab89

C = 0x98badcfe

D = 0x10325476

E = 0xc3d2e1fO

Затем начинается главный цикл алгоритма. Он обрабатывает сообщение 512-битовыми блоками и продолжается, пока не исчерпаются все блоки сообщения.

Сначала пять переменных копируются в другие переменные: A в a, B в b, C в c, D в d и E в e.

Главный цикл состоит из четырех этапов по 20 операций в каждом (в MD5 четыре этапа по 16 операций в каждом). Каждая операция представляет собой нелинейную функцию над тремя из a, b, c, d и e, а затем выполняет сдвиг и сложение аналогично MD5. В SHA используется следующий набор нелинейных функций:

$f_t(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$, для t=0 до 19

$f_t(X,Y,Z) = X \oplus Y \oplus Z$, для t=20 до 39

$f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$, для t=40 до 59

$f_t(X,Y,Z) = X \oplus Y \oplus Z$, для t=60 до 79

в алгоритме используются следующие четыре константы:

K_t = 0x5a827999, для t=0 до 19

K_t = 0x6ed9eba1, для t=20 до 39

K_t = 0x8flbbcdc, для t=40 до 59

K_t = 0xca62c1d6, для t=60 до 79

(Если интересно, как получены эти числа, то: 0x5a827999 = $2^{1/2}/4$, 0x6ed9eba1 = $3^{1/2}/4$, 0x8flbbcdc = $5^{1/2}/4$, 0xca62c1d6 = $10^{1/2}/4$.)

Блок сообщения превращается из 16 32-битовых слов (M_0 по M_{15}) в 80 32-битовых слов (W_0 по W_{79}) с помощью следующего алгоритма:

$W_t = M_t$, для t = 0 по 15

$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$, для t = 16 по 79

(В качестве интересного замечания, в первоначальной спецификации SHA не было циклического сдвига вле-

во. Изменение "исправляет технический изъян, который делал стандарт менее безопасным, чем предполагалось " 1543]. NSA отказалось уточнить истинную причину изъяна.)

Если t - это номер операции (от 1 до 80), W_t представляет собой t-ый подблок расширенного сообщения, а $<<<s$ - это циклический сдвиг влево на s битов, то главный цикл выглядит следующим образом :

FOR t = 0 to 79

$TEMP = (a <<< 5) + f_t(b,c,d) + e + W_t + K_t$

$e = d$

$d = c$

$c = b <<< 30$

$b = a$

$a = TEMP$

На 11-й показана одна операция. Сдвиг переменных выполняет ту же функцию, которую в MD5 выполняет использование в различных местах различных переменных .

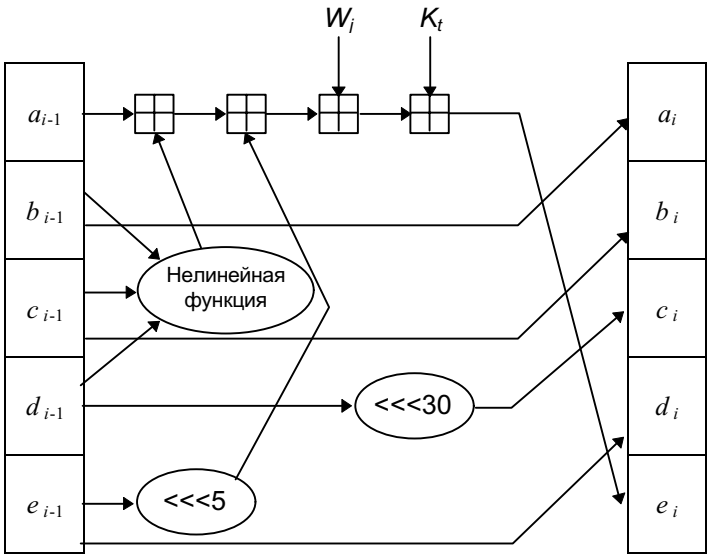


Рис. 18-7. Одна операция SHA.

После всего этого a, b, c, d и e добавляются к A, B, C D и E, соответственно, и алгоритм продолжается для следующего блока данных. Окончательным результатом служит объединение A, B, C D и E.

Безопасность SHA

SHA очень похожа на MD4, но выдает 160-битовое хэш-значение. Главным изменением является введение расширяющего преобразования и добавление выхода предыдущего шага в следующий с целью получения более быстрого лавинного эффекта. Рон Ривест опубликовал цели, преследуемые им при проектировании MD5, но разработчики SHA этого не сделали. Вот улучшения, внесенные Ривестом в MD5 относительно MD4, и их сравнение с SHA:

1. "Добавился четвертый этап." В SHA тоже. Однако в SHA на четвертом этапе используется та же функция f, что и на втором этапе.
2. "Теперь в каждом действии используется уникальная прибавляемая константа ." SHA придерживается схемы MD4, повторно используя константы для каждой группы их 20 этапов.
3. "Функция G на этапе 2 с $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ была изменена на $(X \wedge Z) \vee (Y \wedge (\neg Z))$, чтобы сделать G менее симметричной." В SHA используется версия функции из MD4: $(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$.
4. "Теперь каждое действие добавляется к результату предыдущего этапа . Это обеспечивает более быстрый лавинный эффект." Это изменение было внесено и в SHA. Отличие состоит в том, что в SHA добавлена пятая переменная к b, c и d, которые уже используются в f_t. Это незначительное изменение делает применения вскрытия MD5 ден Боером и Босселаерсом невозможным по отношению к SHA.
5. "Изменился порядок, в котором использовались подблоки сообщения на этапах 2 и 3 , чтобы сделать

шаблоны менее похожими." SHA в этом месте совершенно отличается, так как использует циклический код исправления ошибок.

6. "Значения циклического сдвига влево на каждом этапе были приближенно оптимизированы для ускорения лавинного эффекта. Четыре сдвига, используемые на каждом этапе, отличаются от значений, используемых на других этапах." SHA на каждом этапе использует постоянное значение сдвига. Это значение - взаимно простое число с размером слова, как и в MD4.

Это приводит к следующему заключению: SHA - это MD4 с добавлением расширяющего преобразования, дополнительного этапа и улучшенным лавинным эффектом. MD5 - это MD4 с улучшенным битовым хэшированием, дополнительным этапом и улучшенным лавинным эффектом.

Сведения об успешных криптографических вскрытиях SHA отсутствуют. Так как эта однонаправленная хэш-функция выдает 160-хэш-значение, она устойчивее к вскрытию грубой силой (включая вскрытие методом дня рождения), чем 128-битовые хэш-функции, рассматриваемые в этой главе.

18.8 RIPE-MD

RIPE-MD была разработана для проекта RIPE Европейского сообщества [1305] (см. раздел 25.7). Этот алгоритм представляет собой вариант MD4, разработанный так, чтобы противостоять известным методам криптографического вскрытия, и выдает 128-битовое хэш-значение. Внесены изменения в циклические сдвиги и порядок слов сообщения. Кроме того, параллельно работают две копии алгоритма, отличающиеся константами. После каждого блока результат обоих копий добавляется к переменным сцепления. По видимому, это повышает устойчивость алгоритма к криптоанализу.

18.9 HAVAL

HAVAL - это однонаправленная хэш-функция переменной длины [1646]. Она является модификацией MD5. HAVAL обрабатывает сообщение блоками по 1024 бита, в два раза большими, чем в MD5. Используется восемь 32-битовых переменных сцепления, в два раза больше, чем в MD5, и переменное число этапов, от трех до пяти (в каждом 16 действий). Функция может выдавать хэш-значения длиной 128, 160, 192, 224 или 256 битов.

HAVAL заменяет простые нелинейные функции MD5 на сильно нелинейные функции 7 переменных, каждая из которых удовлетворяет строгому лавинному критерию. На каждом этапе используется одна функция, но при каждом действии входные переменные переставляются различным образом. Используется новый порядок сообщения, и при каждом этапе (кроме первого этапа) используется своя прибавляемая константа. В алгоритме также используется два циклических сдвига.

Ядром алгоритма являются следующие действия:

$TEMP = (f(j,A,B,C,D,E,F,G) <<<7) + (H <<<11) + M[i][r(j)+K(j)]$

$H = G$; $G = F$; $F = E$; $E = D$; $D = C$; $C = B$; $B = A$; $A = TEMP$

Переменное количество этапов и переменная длина выдаваемого значения означают, что существует 15 версий алгоритма. Вскрытие MD5, выполненное ден Боером и Босселаерсом [203], неприменимо к HAVAL из-за циклического сдвига H.

18.10 Другие однонаправленные хэш-функции

MD3 является еще одной хэш-функцией, предложенной Роном Ривестом. Она имела ряд недостатков и никогда не выходила за пределы лаборатории, хотя ее описание недавно было опубликовано в [1335].

Группа исследователей из Университета Ватерлоо предложила однонаправленную хэш-функцию на базе итеративного возведения в степень в $GF(2^{593})$ [22]. По этой схеме сообщение разбивается на 593-битовые блоки. Начиная с первого блока блоки последовательно возводятся в степень. Показатель степени - это результат вычислений для предыдущего блока, первый показатель задается с помощью IV.

Айвэн Дамгард (Ivan Damgård) разработал однонаправленную хэш-функцию, основанную на проблеме рюкзака (см. раздел 19.2) [414], она может быть взломана примерно за 2^{32} операций [290, 1232, 787].

В качестве основы для однонаправленных хэш-функций предлагался и клеточный автомат Стива Вольфрама [1608]. Ранняя реализация [414] небезопасна [1052,404]. Другая однонаправленная хэш-функция, Cellhash [384, 404], и улучшенная версия, Subbash [384,402, 405], также основаны на клеточных автоматах и предназначены для аппаратной реализации. Boognish объединил принципы Cellhash и MD4 [402, 407]. StepRightUp также может быть реализована как хэш-функция [402].

Летом 1991 года Клаус Шнорр (Claus Schnorr) предложил однонаправленную хэш-функцию на базе дис-

кретного преобразования Фурье, названную FFT-Hash [1399]. Через несколько месяцев она была взломана двумя независимыми группами [403, 84]. Шнорр предложил новую версию, FFT-Hash II (предыдущая была переименована в FFT-Hash I) [1400], которая была взломана через несколько недель [1567]. Шнорр предложил дальнейшие модификации [1402, 1403] но, при данных обстоятельствах, они намного медленнее, чем другие алгоритмы этой главы. Еще одна хэш-функция, SL_2 [1526], небезопасна [315].

Дополнительную информацию по теории проектирования однонаправленных хэш-функций из однонаправленных функций и однонаправленных перестановок можно найти в [412, 1138, 1342].

18.11 Однонаправленные хэш-функции, использующие симметричные блочные алгоритмы

В качестве однонаправленных хэш-функций можно использовать симметричные блочные алгоритмы шифрования. Идея в том, что если безопасен блочный алгоритм, то и однонаправленная хэш-функция будет безопасной.

Самым очевидным способом является шифрование сообщения в режиме CBC или CFB с помощью фиксированного ключа и IV, хэш-значением будет последний блок шифротекста. Эти методы описаны в различных стандартах, использующих DES: оба режима в [1143], CBC в [1145], CFB в [55, 56, 54]. Этот способ не слишком подходит для однонаправленных хэш-функций, хотя он будет работать для MAC (см. раздел 18.14) [29].

Способ поумнее использует в качестве ключа блок сообщения, предыдущее хэш-значение в качестве входа, а текущее хэш-значение служит выходом.

Действительные хэш-функции даже еще сложнее. Размер блока обычно совпадает с длиной ключа, и размером хэш-значения будет длина блока. Так как большинство блочных алгоритмов 64-битовые, спроектирован ряд схем, дающих хэш-значение в два раза большее длины блока.

При условии, что хэш-функция правильна, безопасность этой схемы основана на безопасности используемой блочной функции. Однако есть и исключения. Дифференциальный криптоанализ лучше работает против блочных функций в хэш-функциях, чем против блочных функций, используемых для шифрования: ключ известен, поэтому можно использовать различные приемы. Для успеха нужна только одна правильная пара, и можно генерировать столько выбранного открытого текста, сколько нужно. Это направление освещается в [1263, 858, 1313].

Ниже приведен обзор различных хэш-функций, описанных в литературе [925, 1465, 1262]. Выводы о возможности вскрытия предполагают, что используемый блочный алгоритм безопасен, и лучшим вскрытием является вскрытие грубой силой.

Полезной мерой для хэш-функций, основанных на блочных шифрах, является **скорость хэширования**, или количество n-битовых блоков сообщения (n - это размер блока алгоритма), обрабатываемых при шифровании. Чем выше скорость хэширования, тем быстрее алгоритм. (Другое определение этого параметра дается в [1262], но определение, приведенное мной, более интуитивно и шире используется. Это может запутать.)

Схемы, в которых длина хэш-значения равна длине блока

Вот общая схема (см. 10-й):

$H_0 = I_H$, , где I_H - случайное начальное значение

$H_i = E_A(B) \oplus C$

где A, B и C могут быть либо M_i, H_{i-1}, ($M_i \oplus H_{i-1}$), либо константы (возможно равные 0). H_0 - это некоторое случайное начальное число I_H. Сообщение разбивается на части в соответствии с размером блока, M_i, обрабатываемые отдельно. Кроме того, используется вариант MD-усиления, возможно та же процедура дополнения, что и в MD5 и SHA.

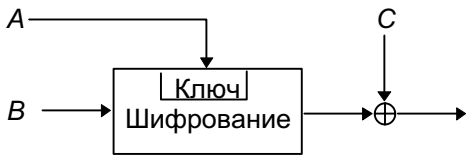


Рис. 18-8. Обобщенная хэш-функция, у которой длина хэш-значения равна длине блока

Табл. 18-1.
Безопасные хэш-функции, у которых

длина хэш-значения равна длине блока

$H_i = E_{H_{i-1}}(M_i) \oplus M_i$
$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{H_{i-1}}(M_i) \oplus H_{i-1} \oplus M_i$
$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i$
$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{M_i}(H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(M_i) \oplus M_i$
$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(M_i) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus M_i$

Три различные переменные могут принимать одно из четырех возможных значений, поэтому всего сущес т-вует 64 варианта схем этого типа. Они все были изучены Бартом Пренелом (Bart Preneel) [1262].

Пятнадцать из них тривиально слабы, так как результат не зависит от одного из входов . Тридцать семь не-безопасны по более тонким причинам. В 17-й перечислены оставшиеся 12 безопасных схем: первые четыре безопасны против всех вскрытий (см. 9th), а последние 8 безопасны против всех типов вскрытий, кроме вскр ы-тия с фиксированной точкой, о котором в реальных условиях не стоит беспокоиться .

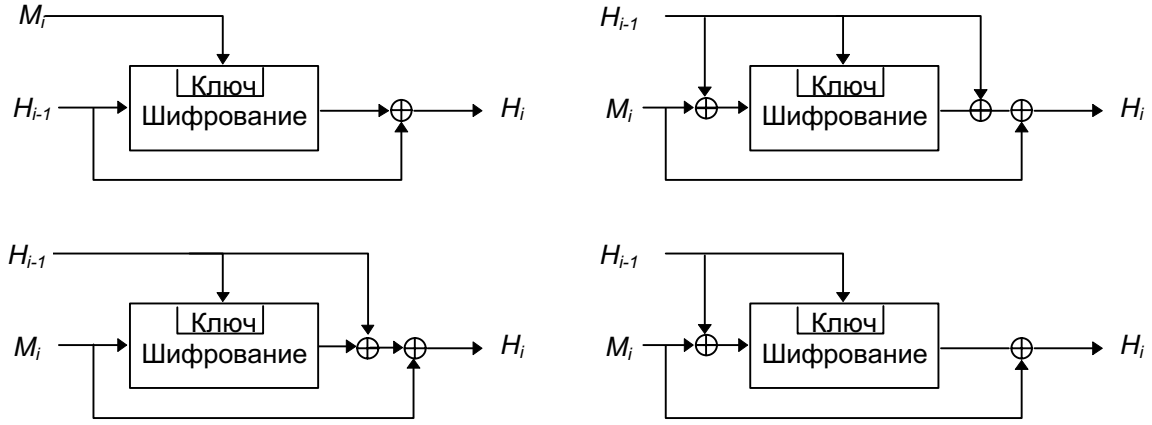


Рис. 18-9. Четыре безопасных хэш-функции, у которых длина хэш-значения равна длине блока

Первая схема была описана в [1028]. Третья схема была описана в [1555, 1105, 1106] и предлагалась в каче-стве стандарта ISO [766]. Пятая схема была предложена Карлом Майером (Carl Meyer), но в литературе обычно называется Davies-Meyer [1606, 1607, 434, 1028]. Десятая схема была предложена в качестве режима хэш-функции для LOKI [273].

Скорость хэширования первой, второй, третьей, четвертой, пятой и одиннадцатой схем равна 1 - длина кл ю-ча равна длине блока. Скорость хэширования других схем составляет k/n, где k -длина ключа. Это означает, что если длина ключа короче длины блока, то блок сообщения должен быть по длине равен ключу . Не рекомендует-ся, чтобы блок сообщения был длиннее ключа, даже если длина ключа алгоритма шифрования больше, чем длина блока.

Если блочный алгоритм подобно DES обладает свойством комплиментарности и слабыми ключами , для всех 12 схем существует возможность дополнительного вскрытия. Оно не слишком опасно и в действительности не стоит об это м беспокоиться. Однако вы можете обезопасить себя от такого вскрытия , зафиксировав значение второго и третьего битов ключа, равное "01" или "10" [1081,1107]. Конечно же это уменьшит длину k с 56 битов до 54 битов (для DES) и уменьшит скорость хэширования.

Было показано, что следующие схемы, описанные в литературе, небезопасны .

Эта схема [1282] была взломана в [369]:

$H_i = E_{M_i}(H_{i-1})$

Дэвис (Davies) и Прайс (Price) предложили вариант, в котором все сообщение циклически обрабатывается алгоритмом дважды [432, 433]. Вскрытие Копперсмита взламывает такую схему даже при небольшой вычисл и-тельной мощности [369]. В [1606] была показана небезопасность еще одной схемы [432, 458]:

$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1})$

В [1028] была показана небезопасность следующей схемы (c - константа):

$H_i = E_c(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$

Модификация схемы Davies-Meyer

Лай (Lai) и Массей (Massey) модифицировали метод Davies-Meyer, чтобы можно было использовать шифр IDEA [930, 925]. IDEA использует 64-битовый блок и 128-битовый ключ. Вот предложенная ими схема:

$H_0 = I_H$, , где I_H - случайное начальное значение

$H_i = E_{H_{i-1}, M_i}(H_{i-1})$

Эта функция хэширует сообщение 64-битовыми блоками и выдает 64-битовое значение (см. 8-й).

Более простое вскрытие этой схемы, чем метод грубой силы, неизвестно.

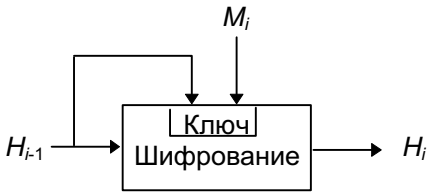


Рис. 18-10. Модификация схемы Davies-Meyer.

Preneel-Bosselaers-Govaerts-Vandewalle

Эта хэш-функция, впервые предложенная в [1266], выдает хэш-значение, в два раза большее длины блока алгоритма шифрования: при 64-битовом алгоритме получается 128-битовое хэш-значение.

При 64-битовом блочном алгоритме схема выдает два 64-битовых хэш-значения, G_i и H_i, объединение которых и дает 128-битовое хэш-значение. У большинства блочных алгоритмов длина блока равна 64 битам. Два соседних блока, L_i и R_i, размер каждого равен размеру блока, хэшируются вместе.

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$G_i = E_{L_i \oplus H_{i-1}}(R_i \oplus G_{i-1}) \oplus R_i \oplus G_{i-1} \oplus H_{i-1}$

$H_i = E_{L_i \oplus R_i}(H_{i-1} \oplus G_{i-1}) \oplus L_i \oplus G_{i-1} \oplus H_{i-1}$

Лай приводит вскрытие этой схемы, которое в некоторых случаях делает вскрытие методом дня рождения тривиальным [925, 926]. Пренел (Preneel) [1262] и Копперсмит (Coppersmith0 [372] также успешно взломали эту схему. Не используйте ее.

Quisquater-Girault

Эта схема, впервые предложенная в [1279], генерирует хэш-значение, в два раза большее длины блока. Ее скорость хэширования равна 1. Она использует два хэш-значения, G_i и H_i, и хэширует вместе два блока, L_i и R_i.

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$W_i = E_{L_i}(G_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

$G_i = E_{R_i}(W_i \oplus L_i) \oplus G_{i-1} \oplus H_{i-1} \oplus L_i$

$H_i = W_i \oplus G_{i-1}$

Эта схема появилась в 1989 году в проекте стандарта ISO [764], но была заменена более поздней версией [765]. Проблемы безопасности этой схемы были описаны в [1107, 925, 1262, 372]. (В действительности, версия, описанная в материалах конференции, была после того, как версия, представленная на конференции, была вскрыта.) В ряде случаев сложность вскрытия методом дня рождения имеет равна 2^{39}, а не 2^{64}, как у вскрытия грубой. Не используйте эту схему.

LOKI с удвоенным блоком

Этот алгоритм представляет собой модификацию Quisquater-Cirault, специально спроектированную для ра-

боты с LOKI [273]. Все параметры - те же, что и в Quisquater-Girault.

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$W_i = E_{L_i \oplus G_{i-1}}(G_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

$G_i = E_{R_i \oplus H_{i-1}}(W_i \oplus L_i) \oplus G_{i-1} \oplus H_{i-1} \oplus L_i$

$H_i = W_i \oplus G_{i-1}$

И снова в некоторых случаях вскрытие методом дня рождения оказывается тривиальным [925, 926, 1262, 372, 736]. Не используйте эту схему.

Параллельная схема Davies-Meyer

Это еще одна попытка создать алгоритм со скоростью хэширования 1, который выдает хэш-значение, в два раза большее длины блока. [736].

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$G_i = E_{L_i \oplus R_i}(G_{i-1} \oplus L_i) \oplus L_i \oplus H_{i-1}$

$H_i = E_{L_i}(H_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

К сожалению эта схема тоже небезопасна [928, 861]. Оказывается, что хэш-функция удвоенной длины со скоростью хэширования, равной 1, не может быть безопаснее, чем Davies-Meyer [861].

Тандемная (Tandem) и одновременная (Abreast) схемы Davies-Meyer

Другой способ обойти ограничения, присущие блочным шифрам с 64-битовым ключом, использует алг о-ритм, подобный IDEA (см. раздел 13.9), с 64-битовым блоком и 128-битовым ключом. Следующие две схемы выдают 128-битовхэш-значение, а их скорость хэширования равна $^1/_2$ [930, 925].

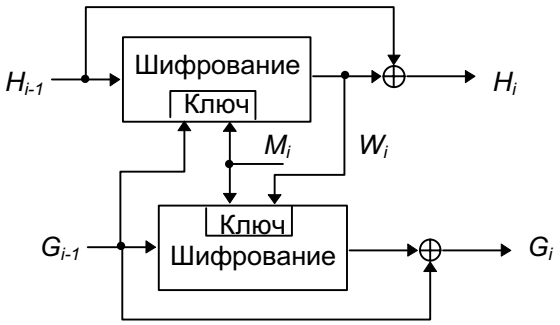


Рис. 18-11. Тандемная (Tandem) схема Davies-Meyer.

В первой схеме две модифицированные функции Davies-Meyer работают тандемом, конвейерно (см. 7-й).

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$W_i = E_{G_{i-1}, M_i}(H_{i-1})$

$G_i = G_{i-1} \oplus E_{M_i, W_i}(G_{i-1})$

$H_i = W_i \oplus H_{i-1}$

В следующей схеме используются две модифицированные функции, работающие одновременно (см. 6-й).

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$G_i = G_{i-1} \oplus E_{M_i, H_{i-1}}(\neg G_{i-1})$

$H_i = H_{i-1} \oplus E_{G_{i-1}, M_i}(H_{i-1})$

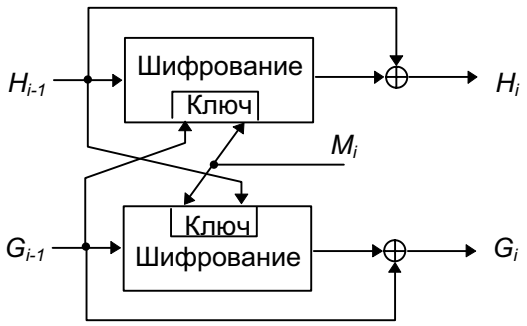


Рис. 18-12. Одновременная (Abreast) схема Davies-Meyer.

В обеих схемах два 64-битовых значения, G_i и H_i, объединяются, образуя единое 128-битовое хэш-значение.

Насколько известно, безопасность 128-битовой хэш-функции этих алгоритмов идеальна: для обнаружения сообщения с заданным хэш-значением требуется 2^{128} попыток, а для нахождения двух случайных сообщений с одинаковым хэш-значением - 2^{64} попыток, при условии, что лучшим способом вскрытия является применение грубой силы.

MDC-2 и MDC-4

MDC-2 и MDC-4 разработаны в IBM [1081, 1079]. В настоящее время изучается вопрос использования MDC-2, иногда называемой Meyer-Schilling, в качестве стандарта ANSI и ISO [61, 765], этот вариант был предложен в [762]. MDC-4 определена для проекта RIPE [1305] (см. раздел 25.7). Спецификация использует DES в качестве блочной функции, хотя теоретически может быть использован любой блочный алгоритм.

Скорость хэширования MDC-2 равна $^1/_2$, длина хэш-значения этой функции в два раза больше размера блока.Ее схема показана на 5-й. MDC-4 также выдает хэш-значение в два раза большее размера блока, а ее ск о-рость хэширования равна $^1/_4$ (см. 4-й).

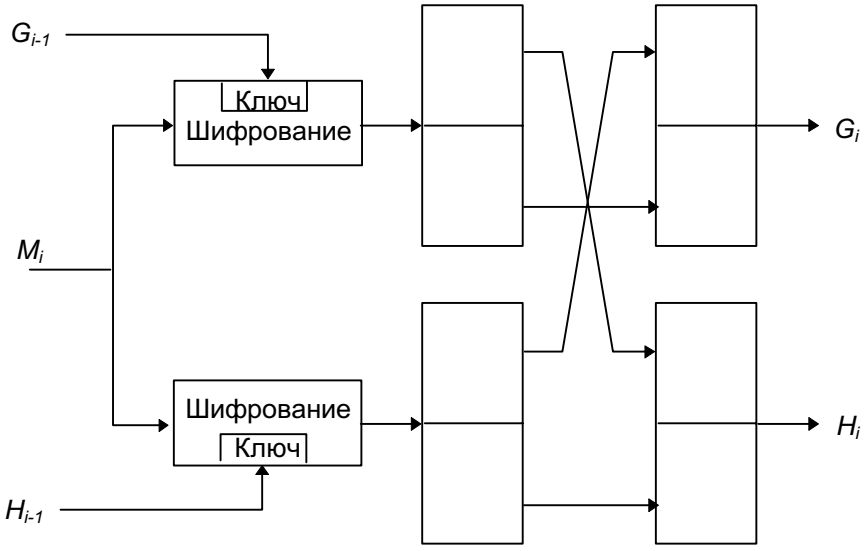


Рис. 18-13. MDC-2.

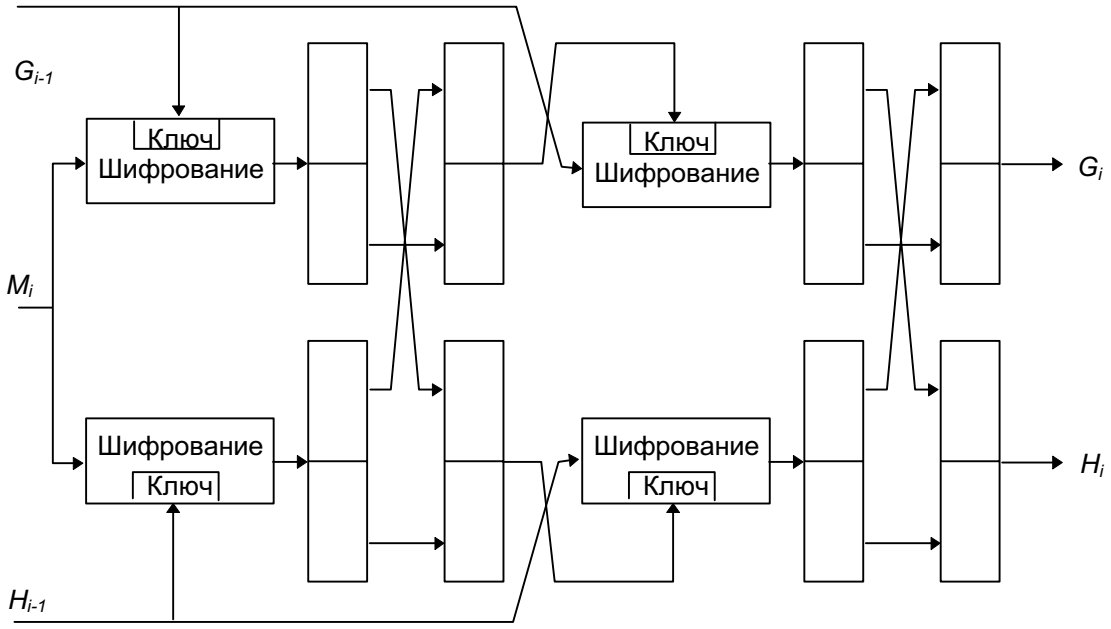


Рис. 18-14. MDC-4.

Эти схемы были проанализированы в [925, 1262]. Они безопасны с учетом сегодняшних возможностей вычислительной техники, но их надежность не так велика, как хотелось разработчикам. Их устойчивость к дифференциальному криптоанализу при DES в качестве блочного алгоритма была рассмотрена в [1262].

MDC-2 и MDC-4 запатентованы [223].

Хэш-функция AR

Хэш-функция AR была разработана Algorithmic Research, Ltd. и затем распространена ISO только для информации [767]. Ее базовая структура является вариантом используемого блочного шифра (DES в упомянутой статье) в режиме CBC. Выполняется XOR последних двух блоков шифротекста, константы и текущего блока сообщения, результат шифруется алгоритмом. Хэш-значением являются последние вычисленные два блока шифротекста. Сообщение обрабатывается дважды, двумя различными ключами, поэтому скорость хэширования равна $^1/_2$. Первым ключом служит 0x0000000000000000, вторым - 0x2a41522f4446502a, а значение константы c равно 0x0123456789abcdef. Результат сжимается до одного 128-битового хэш-значения. Подробности приведены в [750].

$H_i = E_K(M_i \oplus H_{i-1} \oplus H_{i-2} \oplus c) \oplus M_i$

Функция выглядит привлекательной, но не является безопасной. После некоторой значительной предобработки становится возможным легко находить сообщения с одинаковым хэш-значением [416].

Хэш-функция ГОСТ

Эта хэш-функция появилась в России и определена в стандарте ГОСТ Р 34.11.94 [657]. В ней используется блочный алгоритм ГОСТ (см. раздел 14.1), хотя теоретически может использоваться любой блочный алгоритм с 64-битовым блоком и 256-битовым ключом. Функция выдает 256-битовое хэш-значение.

Функция сжатия, $H_i = f(M_i,H_{i-1})$ (оба операнда - 256-битовые величины) определяется следующим образом:

(1) При помощи линейного смешивания M_i, H_{i-1} и некоторых констант генерируется четыре ключа шифрования ГОСТ.

(2) Каждый ключ используется для шифрования отличных 64 битов H_{i-1} в режиме ECB. Полученные 256 битов сохраняются во временной переменной S.

(3) H_i является сложной, хотя и линейной функцией S, M_i и H_{i-1}.

Хэш-значение последнего блока сообщения не является его окончательным хэш-значением. На деле используется три переменные сцепления: H_n - это хэш-значение последнего блока, Z - это XOR всех блоков сообщения, а L - длина сообщения. С использованием этих переменных и дополненного последнего блока M', окончательное хэш-значение равно:

$H = f(Z \oplus M', f(L, f(M', H_n)))$

Документация немного запутана (и на русском языке), но я думаю, что понял все правильно . Во всяком случае эта хэш-функция определена как часть российского Стандарта цифровой подписи (см. раздел 20.3).

Другие схемы

Ральф Меркл предложил схему, использующую DES, но она медленна - обрабатывает только семь битов сообщения за итерацию, и каждая итерация состоит из двух шифрований DES [1065, 1069]. Другая схема [1642, 1645] небезопасна [1267], когда-то она предлагалась в качестве стандарта ISO.

18.12 Использование алгоритмов с открытым ключом

В качестве однонаправленной хэш-функции можно использовать и алгоритм шифрования с открытым ключом в режиме сцепления блоков. Если затем выбросить личный ключ, то взломать хэш-функцию будет также трудно, как и прочитать сообщение без личного ключа.

Вот пример, использующий RSA. Если M - это хэшируемое сообщение, n - произведение двух простых чисел p и q, а e - другое большое число, взаимно простое с $(p - 1)(q - 1)$, то хэш-функция, $H(M)$, будет равна

$$H(M) = M^e \bmod n$$

Еще проще использовать одно сильное простое число в качестве модуля p. Тогда:

$$H(M) = M^e \bmod p$$

Вскрытие этой проблемы возможно не легче, чем поиск дискретного логарифма e. Проблема этого алгоритма состоит в том, что он намного медленнее, чем другие обсуждаемые алгоритмы . По этой причине я не советую его.

18.13 Выбор однонаправленной хэш-функции

Лучшими кажутся SHA, MD5 и схемы, основанные на блочных шифрах. Другие на самом деле не были исследованы в достаточной степени. Я голосую за SHA. У нее более длинное хэш-значение, чем у MD5, она быстрее, чем многие схемы с блочными шифрами, и разработана NSA. Я верю в криптоаналитические возможности NSA, даже если они не публикуют свои результаты.

В 16-й для сравнения приведены временные соотношения для некоторых хэш-функций . They are meant for comparison purposes only.

Табл. 18-2.
Скорости шифрования некоторых хэш-функций на i486SX/33 МГц

Алгоритм	Длина хэш-значения	Скорость шифрования (Кбайт/с)
Одновременная схема Davies-Meyer (с IDEA)	128	22
Davies-Meyer (с DES)	64	9
Хэш-функция ГОСТ	256	11
HAVAL (3 прохода)	переменная	168
HAVAL (4 прохода)	переменная	118
HAVAL (5 прохода)	переменная	95
MD2	128	23
MD4	128	236
MD5	128	174
N-хэш (12 этапов)	128	29
N-хэш (15 этапов)	128	24
RIPE-MD	128	182
SHA	160	75
Snerfu (4 прохода)	128	48
Snerfu (8 проходов)	128	23

18.14 Коды проверки подлинности сообщения

Код проверки подлинности сообщения (message authentication code, MAC) - это зависящая от ключа однонаправленная хэш-функция. Коды MAC обладают теми же свойствами, что и рассмотренные ранее хэш-функции, но они, кроме того, включают ключ. (Это не означает, что вы можете опубликовать ключ MAC и использовать MAC как однонаправленную хэш-функцию.) Только владелец идентичного ключа может проверить хэш-значение. Коды MAC очень полезны для обеспечения проверки подлинности без нарушения безопасности.

Коды MAC могут быть использованы для проверки подлинности файлов, которыми обмениваются пользователи. Также они могут быть использованы одним пользователем для проверки, не изменились ли его файлы, может быть из-за вируса. Пользователь может вычислить MAC его файлов и сохранить эти значения в таблице. Если пользователь воспользуется вместо MAC однонаправленной хэш-функцией, то вирус может вычислить новые хэш-значения после заражения файлов и заменить элементы таблицы. С MAC вирус не сможет этого добиться, так как ключ вирусу неизвестен.

Простым способом преобразовать однонаправленную хэш-функцию в MAC является шифрование хэш-значения симметричным алгоритмом. Любой MAC может быть преобразован в однонаправленную хэш-функцию с помощью раскрытия ключа.

CBC-MAC

Простейший способ создать зависящую от ключа однонаправленную хэш-функцию - шифрование сообщения блочным алгоритмом в режимах CBC или CFB. Хэш-значением является последний шифрованный блок, зашифрованный в режимах CBC или CFB. Метод CBC определен в ANSI X9.9 [54], ANSI X9.19 [56], ISO 8731-1 [759], ISO 9797 [763] и австралийском стандарте [1496]. Дифференциальный криптоанализ может вскрыть эту схему, если в качестве блочного алгоритма используется DES с уменьшенным числом этапов или FEAL [1197].

Потенциальная проблема, связанная с безопасностью этого метода, состоит в том, что получатель должен знать ключ, и этот ключ позволяет ему генерировать сообщения с тем же хэш-значением, что и у присланного сообщения, с помощью дешифрирования в обратном направлении.

Алгоритм проверки подлинности сообщения (Message Authenticator Algorithm, MAA)

Этот алгоритм является стандартом ISO [760]. Он выдает 32-битовое хэш-значение и был спроектирован для мэйнфреймов с быстрыми инструкциями умножения [428].

$v = v <<< 1$

$e = v \oplus w$

$x = ((((e + y) \bmod 2^{32}) \vee A \wedge C) * (x \oplus M_i)) \bmod 2^{32}\text{-}1$

$y = ((((e + x) \bmod 2^{32}) \vee B \wedge D) * (y \oplus M_i)) \bmod 2^{32}\text{-}1$

Эти действия повторяются для каждого блока сообщения, M_i, и результирующее хэш-значение получается с помощью XOR x и y. Переменные v и e зависят от ключа. A, B, C и D являются константами.

Возможно, этот алгоритм широко используется, но я не верю, что он достаточно безопасен. Он был разработан давным давно и не слишком сложен.

Двунаправленный MAC

Этот MAC выдает хэш-значение, которое в два раза длиннее блока алгоритма [978). Сначала для сообщения вычисляется CBC-MAC. Затем вычисляется CBC-MAC сообщения с обратным порядком блоков. Двунаправленный MAC просто является объединением этих двух значений. К сожалению эта схема небезопасна [1097].

Методы Джунемана

Этот MAC также называют квадратичным конгруэнтным кодом обнаружения манипуляции (quadratic congruential manipulation detection code, QCMDC) [792, 789]. Сначала разделим сообщение на m-битовые блоки. Затем:

$H_0 = I_H$, , где I_H - секретный ключ

$H_i = (H_{i-1} + M_i)^2 \bmod p$, где p - простое число, меньшее 2^m-1, а + обозначает целочисленное сложение.

Джунеман (Jueneman) предлагает $n = 16$ и $p = 2^{31}$-1. В [792] он также предлагает, чтобы H_1 использовался в качестве дополнительного ключа, а действительное сообщение начиналось бы с H_2.

Из-за множества вскрытий типа дня рождения, выполненных в сотрудничестве с Доном Копперсмитом, Джунеман предложил вычислять QCMDC четыре раза, используя результат одной итерации в качестве IV для

следующей итерации, а затем результаты объединяются в 128-битовое хэш-значение [793]. В дальнейшем эта идея была усилена за счет параллельного выполнения четырех итераций с поперечными связями между ними [790, 791]. Эта схема была взломана Копперсмитом [376].

В другом варианте [432, 434] операция сложения заменена XOR, и используются блоки сообщения, намного меньшие p. Кроме того, был задан H_0, что превратило алгоритм в однонаправленную хэш-функцию без ключа. После того, как эта схема была вскрыта [612], она была усилена для использования в качестве части проекта European Open Shop Information-TeleTrust [1221], процитирована в CCITT X.509 [304] и принята ISO в 10118 [764, 765]. К сожалению Копперсмит взломал и эту схему [376]. В ряде исследований изучалась возможность использовать отличные от 2 основания экспоненты [603], но ни одно не оказалось перспективным.

RIPE-MAC

RIPE-MAC был изобретен Бартом Пренелом [1262] и использован в проекте RIPE [1305] (см. раздел 18.8). Он основан на ISO 9797 [763] и использует DES в качестве функции блочного шифрования. Существует два варианта RIPE-MAC: один, который использует обычный DES, называется RIPE-MAC1, а другой, использующий для еще большей безопасности тройной DES, называется RIPE-MAC3. RIPE-MAGI использует одно шифрование DES на 64-битовый блок сообщения, а RIPE-MAC3 - три.

Алгоритм состоит из трех частей. Во первых, сообщение увеличивается так, чтобы его длина была кратна 64 битам. Затем, увеличенное сообщение разбивается на 64-битовые блоки. Для хэширования этих блоков в один блок используется функция сжатия, зависящая от секретного ключа. На этом этапе используется либо DES, либо тройной DES. Наконец, выход этой функции сжатия подвергается еще одному DES-шифрованию с другим ключом, полученным из ключа, используемого при сжатии. Подробности можно найти в [1305].

IBC-хэш

IBC-хэш - это еще один MAC, используемый в проекте RIPE [1305] (см. раздел 18.8). Он интересен потому, что его безопасность доказана, вероятность успешного вскрытия может быть оценена количественно. К сожалению каждое сообщение должно хэшироваться новым ключом. Выбранный уровень безопасности ограничивает максимальный размер хэшируемого сообщения, чего не делает ни одна другая из рассмотренных в этой главе функция. С учетом этих соображений в отчете RIPE рекомендуется, чтобы IBC-хэш использовалась бы только для длинных, редко посылаемых сообщений. Ядром функции является

$h_i = ((M_i \bmod p) + v) \bmod 2^n$

Секретный ключ представляет собой пару p и v, где p - n-битовое простое число, а v - случайное число, меньшее 2^n. Значения M_i получаются с помощью строго определенной процедуры дополнения. Вероятности вскрыть как однонаправленность, так и устойчивость к столкновениям, могут быть оценены количественно, и пользователи, меняя параметры, могут выбрать нужный уровень безопасности.

Однонаправленная хэш-функция MAC

В качестве MAC может быть использована и однонаправленная хэш-функция [1537]. Пусть Алиса и Боб используют общий ключ K, и Алиса хочет отправить Бобу MAC сообщения M. Алиса объединяет K и M, и вычисляет однонаправленную хэш-функцию объединения: $H(K,M)$. Это хэш-значение и является кодом MAC. Так как Боб знает K, он может воспроизвести результат Алисы, а Мэллори, которому ключ неизвестен, не сможет это сделать.

Со методами MD-усиления этот способ работает, но есть серьезные проблемы. Мэллори всегда может добавить новые блоки к концу сообщения и вычислить правильный MAC. Это вскрытие может быть предотвращено, если к началу сообщения добавить его длину, но Пренел сомневается в этой схеме [1265]. Лучше добавлять ключ к концу сообщения, $H(M,K)$, но при этом также возникают проблемы [1265]. Если H однонаправленная функция, которая не защищена от столкновений, Мэллори может подделывать сообщения. Еще лучше $H(K,M,K)$ или $H(K_1,M,K_2)$, где K_1 и K_2 различны [1537]. Пренел не уверен и в этом [1265].

Безопасными кажутся следующие конструкции:

$H(K_1, H(K_2, M))$

$H(K, H(K,M))$

$H(K, p,M,K))$, где p дополняет K до полного блока сообщения.

Лучшим подходом является объединение с каждым блоком сообщения по крайней мере 64 битов ключа. Это делает однонаправленную функцию менее эффективной, так как уменьшаются блоки сообщения, но так она становится намного безопаснее [1265].

Или используйте однонаправленную хэш-функцию и симметричный алгоритм. Сначала хэшируйте файл,

потом зашифруйте хэш-значение. Это безопаснее, чем сначала шифровать файл, а затем хэшировать зашифр о-ванный файл, но эта схема чувствительна к тому же вскрытию, что и конструкция $H(M,K)$ [1265].

MAC с использованием потокового шифра

Эта схема MAC использует потоковые шифры (см. 3-й) [932]. Криптографически безопасный генератор псевдослучайных битов демультиплексирует поток сообщения на два подпотока. Если на выходе генератора битов k_i единица, то текущий бит сообщения m_i отправляется в первый подпоток, если ноль, то m_i отправляется во второй подпоток. Каждый подпоток отправляется на свой LFSR (раздел 16.2). Выходом MAC просто является конечное состояние обоих регистров.

К несчастью этот метод небезопасен по отношению к небольшим изменениям в сообщении [1523]. Например, если изменить последний бит сообщения, то для создания поддельного MAC нужно будет изменить только 2 бита соответствующего MAC; это может быть выполнено с заметной вероятностью. Автор предлагает более безопасный, и более сложный, вариант.

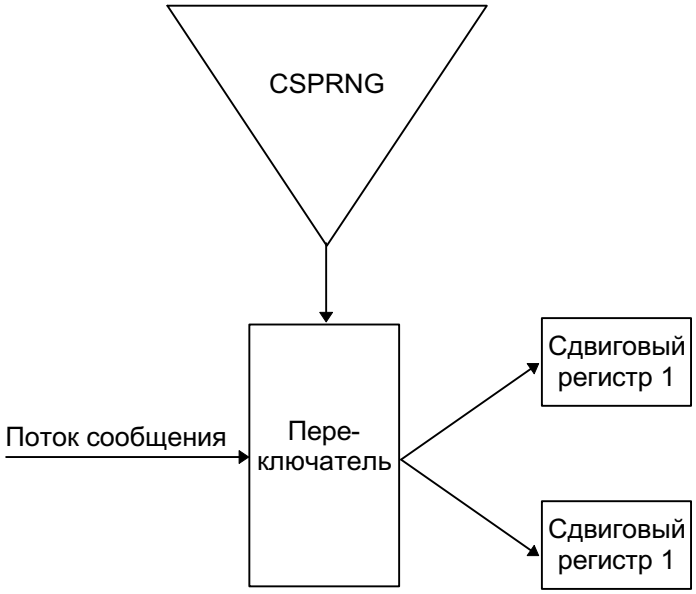


Рис. 18-15. MAC с использованием потокового шифра

Глава 19 Алгоритмы с открытыми ключами

19.1 Основы

Концепция криптографии с открытыми ключами была выдвинута Уитфилдом Диффи (Whitfield Diffie) и Мартином Хеллманом (Martin Hellman), и независимо Ральфом Мерклом (Ralph Merkle). Их вкладом в криптографию было убеждение, что ключи можно использовать парами - ключ шифрования и ключ дешифрирования - и что может быть невозможно получить один ключ из другого (см. Раздел 2.5). Диффи и Хеллман впервые представили эту идею на Национальной компьютерной конференции (National Computer Conference) 1976 года [495], через несколько месяцев была опубликована их основополагающая работа "New Directions in Cryptography" ("Новые направления в криптографии") [496]. (Из-за бесстрастного процесса публикации первый вклад Меркла в эту область вышел появился только в 1978 году [1064].)

С 1976 года было предложено множество криптографических алгоритмов с открытыми ключами . Многие из них небезопасны. Из тех, которые являются безопасными, многие непригодны для практической реализации . Либо они используют слишком большой ключ, либо размер полученного шифротекста намного превышает размер открытого текста.

Немногие алгоритмы являются и безопасными, и практичными . Обычно эти алгоритмы основаны на одной из трудных проблем, рассмотренных в разделе 11.2. Некоторые из этих безопасных и практичных алгоритмов подходят только для распределения ключей. Другие подходят для шифрования (и для распределения ключей). Третьи полезны только для цифровых подписей. Только три алгоритма хорошо работают как при шифровании, так и для цифровой подписи: RSA, ElGamal и Rabin. Все эти алгоритмы медленны. Они шифруют и дешифруют данные намного медленнее, чем симметричные алгоритмы. Обычно их скорость недостаточна для шифрования больших объемов данных.

Гибридные криптосистемы (см. раздел 2.5) позволяют ускорить события: для шифрования сообщения используется симметричный алгоритм со случайным ключом, а алгоритм с открытым ключом применяется для шифрования случайного сеансового ключа.

Безопасность алгоритмов с открытыми ключами

Так как у криптоаналитика есть доступ к открытому ключу, он всегда может выбрать для шифрования любое сообщение. Это означает, что криптоаналитик при заданном $C = E_K(P)$ может попробовать угадать значение P и легко проверить свою догадку. Это является серьезной проблемой, если количество возможных открытых текстов настолько мало, что делает возможным исчерпывающий поиск, но эту проблему легко можно решить, дополняя сообщения строкой случайных битов. Это приводит к тому, что идентичным открытым текстам соответствуют различные шифротексты. (Более подробно эта идея описана в разделе 23.15.)

Это особенно важно, если алгоритм с открытым ключом используется для шифрования сеансового ключа . Ева может создать базу данных всех возможных сеансовых ключей, зашифрованных открытым ключом Боба . Конечно, это потребует много времени и памяти, но взлом грубой силой разрешенного к экспорту 40-битового ключа или 56-битового ключа DES потребует намного больше времени и памяти. Как только Ева создаст такую базу данных, она получит ключ Боба и сможет читать его почту .

Алгоритмы с открытыми ключами спроектированы так, чтобы противостоять вскрытиям с выбранным открытым текстом. Их безопасность основана как на трудности получения секретного ключа по открытому, так и на трудности получить открытый текст по шифротексту. Однако большинство алгоритмов с открытым ключом особенно чувствительны к вскрытию с выбранным шифротекстом (см. раздел 1.1).

В системах, в которых операция, обратная шифрованию, используется для цифровой подписи, это вскрытие невозможно предотвратить, если для шифрования и подписей использовать одинаковые ключи .

Следовательно, важно увидеть всю систему целиком, а не только составные части . Хорошие протоколы с открытыми ключами спроектированы таким образом, чтобы различные стороны не могли расшифровать произвольные сообщения, генерированные другими сторонами, - хорошим примером являются протоколы доказательства идентичности (см. раздел 5.2).

19.2 Алгоритмы рюкзака

Первым алгоритмом для обобщенного шифрования с открытым ключом стал алгоритм рюкзака, разработанный Ральфом Мерклом и Мартином Хеллманом [713, 1074]. Он мог быть использован только для шифрования, хотя позднее Ади Шамир адаптировал систему для цифровой подписи [1413]. Безопасность алгоритмов рюкзака опирается на проблему рюкзака, **NP-полную** проблему. Хотя позже было обнаружено, что этот алгоритм небезопасен, его стоит изучить, так как он демонстрирует возможность применения **NP-полной** проблемы

в криптографии с открытыми ключами.

Проблема рюкзака несложна. Дана куча предметов различной массы, можно ли положить некоторые из этих предметов в рюкзак так, чтобы масса рюкзака стала равна определенному значению ? Более формально, дан набор значений M_1, M_2, . . . , M_n и сумма S, вычислить значения b_i, такие что

$S = b_1M_1 + b_2M_2 + . . . + b_nM_n$

b_i может быть либо нулем, либо единицей. Единица показывает, что предмет кладут в рюкзак, а ноль - что не кладут.

Например, массы предметов могут иметь значения 1, 5, 6, 11, 14 и 20. Вы можете упаковать рюкзак так, чтобы его масса стала равна 22, использовав массы 5, 6 и 11. Невозможно упаковать рюкзак так, чтобы его масса была равна 24. В общем случае время, необходимое для решения этой проблемы, с ростом количества предметов в куче растет экспоненциально.

В основе алгоритма рюкзака Меркла-Хеллмана лежит идея шифровать сообщение как решение набора проблем рюкзака. Предметы из кучи выбираются с помощью блока открытого текста, по длине равного количеству предметов в куче (биты открытого текста соответствуют значениям b), а шифротекст является полученной суммой. Пример шифротекста, зашифрованного с помощью проблемы рюкзака, показан на .

Открытый текст	1 1 1 0 0 1	0 1 0 1 1 0	0 0 0 0 0 0	0 1 1 0 0 0
Рюкзак	1 5 6 11 14 20	1 5 6 11 14 20	1 5 6 11 14 20	1 5 6 11 14 20
Шифротекст	1+5+6+20=32	5+11+14=30	0=0	5+6=11

Рис. 19-1. Шифрование с рюкзаками

Фокус в том, что на самом деле существуют две различные проблемы рюкзака , одна решается за линейное время, а другая, как считается, - нет. Легкую проблему можно превратить в трудную. Открытый ключ представляет собой трудную проблему, которую легко использовать для шифрования, но невозможно для дешифрирования сообщений. Закрытый ключ является легкой проблемой, давая простой способ дешифрировать сообщения . Тому, кто не знает закрытый ключ, придется попытаться решить трудную проблему рюкзака .

Сверхвозрастающие рюкзаки

Что такое легкая проблема рюкзака? Если перечень масс представляет собой **сверхвозрастающую последовательность**, то полученную проблему рюкзака легко решить. Сверхвозрастающая последовательность - это последовательность, в которой каждой член больше суммы всех предыдущих членов . Например, последовательность {1,3,6,13,27,52} является сверхвозрастающей, а {1,3,4,9, 15,25} - нет.

Решение **сверхвозрастающего рюкзака** найти легко. Возьмите полный вес и сравните его с самым большим числом последовательности. Если полный вес меньше, чем это число, то его не кладут в рюкзак . Если полный вес больше или равен этому числу, то оно кладется в рюкзак . Уменьшим массу рюкзака на это значение и перейдем к следующему по величине числу последовательности. Будем повторять, пока процесс не закончится. Если полный вес уменьшится до нуля, то решение найдено. В противном случае , there isn't.

Например, пусть полный вес рюкзака - 70, а последовательность весов {2,3,6, 13,27,52}. Самый большой вес, 52, меньше 70, поэтому кладем 52 в рюкзак. Вычитая 52 из 70, получаем 18. Следующий вес, 27, больше 18, поэтому 27 в рюкзак не кладется. вес, 13,меньше 18, поэтому кладем 13 в рюкзак. Вычитая 13 из 18, получаем 5. Следующий вес, 6, больше 5, поэтому 6 не кладется в рюкзак. Продолжение этого процесса покажет, что и 2, и 3 кладутся в рюкзак, и полный вес уменьшается до 0, что сообщает о найденном решении. Если бы это был блок шифрования методом рюкзака Меркла-Хеллмана , открытый текст, полученный из значения шифротекста 70, был бы равен 110101.

Не сверхвозрастающие, или нормальные, рюкзаки представляют собой трудную проблему - быстрого алгоритма для них не найдено. Единственным известным способом определить, какие предметы кладутся в рюкзак, является методическая проверка возможных решений, пока вы не наткнетесь на правильное . Самый быстрый алгоритм, принимая во внимание различную эвритсику , имеет экспоненциальную зависимость от числа возможных предметов. Добавьте к последовательности весов еще один член, и найти решение станет вдвое труднее. Это намного труднее сверхвозрастающего рюкзака, где, если вы добавите один предмет к последовательности, поиск решения увеличится на одну операцию .

Алгоритм Меркла-Хеллмана основан на этом свойстве. Закрытый ключ является последовательностью весов проблемы сверхвозрастающего рюкзака. Открытый ключ - это последовательность весов проблемы нормального рюкзака с тем же решением. Меркл и Хеллман, используя модульную арифметику, разработали способ преобразования проблемы сверхвозрастающего рюкзака в проблему нормального рюкзака.

Создание открытого ключа из закрытого

Рассмотрим работу алгоритма, не углубляясь в теорию чисел: чтобы получить нормальную последовательность рюкзака, возьмем сверхвозрастающую последовательность рюкзака, например, {2,3,6,13,27,52}, и умножим по модулю *m* все значения на число *n*. Значение модуля должно быть больше суммы всех чисел последовательности, например, 105. Множитель должен быть взаимно простым числом с модулем, например, 31. Нормальной последовательностью рюкзака будет

2*31 mod 105 = 62

3*31 mod 105 = 93

6*31 mod 105 = 81

13*31 mod 105 = 88

27*31 mod 105 = 102

52*31 mod 105 = 37

Итого - {62,93,81,88,102,37}.

Сверхвозрастающая последовательность рюкзака является закрытым ключом, а нормальная последовательность рюкзака - открытым.

Шифрование

Для шифрования сообщение сначала разбивается на блоки, равные по длине числу элементов последовательности рюкзака. Затем, считая, что единица указывает на присутствие члена последовательности, а ноль - на его отсутствие, вычисляем полные веса рюкзаков - по одному для каждого блока сообщения.

Например, если сообщение в бинарном виде выглядит как 011000110101101110, шифрование, использующее предыдущую последовательность рюкзака, будет происходить следующим образом:

сообщение = 011000 110101 101110

011000 соответствует 93 + 81 = 174

110101 соответствует 62 + 93 + 88 + 37 = 280

101110 соответствует 62 + 81 + 88 + 102 = 333

Шифротекстом будет последовательность 174,280,333

Дешифрирование

Законный получатель данного сообщения знает закрытый ключ: оригинальную сверхвозрастающую последовательность, а также значения *n* и *m*, использованные для превращения ее в нормальную последовательность рюкзака. Для дешифрирования сообщения получатель должен сначала определить n^{-1}, такое что $n(n^{-1}) \equiv 1 \pmod{m}$. Каждое значение шифротекста умножается на $n^{-1} \bmod m$, а затем разделяется с помощью закрытого ключа, чтобы получить значения открытого текста.

В нашем примере сверхвозрастающая последовательность - {2,3,6,13,27,52), *m* равно 105, а *n* - 31. Шифротекстом служит 174,280,333. В этом случае n^{-1} равно 61, поэтому значения шифротекста должны быть умножены на 61 mod 105.

174*61 mod 105 = 9 = 3 + 6, что соответствует 011000

280*61 mod 105 = 70 = 2 + 3 + 13 + 52, что соответствует 110101

333*61 mod 105 = 48 = 2 + 6 + 13 + 27, что соответствует 101110

Расшифрованным открытым текстом является 011000 110101 101110.

Практические реализации

Для последовательности из шести элементов нетрудно решить задачу рюкзака, даже если последовательность не является сверхвозрастающей. Реальные рюкзаки должны содержать не менее 250 элементов. Длина каждого члена сверхвозрастающей последовательности должна быть где-то между 200 и 400 битами, а длина модуля должна быть от 100 до 200 битов. Для получения этих значений практические реализации используют генераторы случайной последовательности.

Вскрывать подобные рюкзаки при помощи грубой силы бесполезно. Если компьютер может проверять миллион вариантов в секунду, проверка всех возможных вариантов рюкзака потребует свыше 10^{46} лет. Даже мил-

лион машин, работающих параллельно, не успеет решить эту задачу до превращения солнца в сверхновую звезду.

Безопасность метода рюкзака

Взломали криптосистему, основанную на проблеме рюкзака, не миллион машин, а пара криптографов. Сначала был раскрыт единственный бит открытого текста [725]. Затем Шамир показал, что в определенных обстоятельствах рюкзак может быть взломан [1415, 1416]. Были и другие достижения - [1428, 38, 754, 516, 488] - но никто не мог взломать систему Мартина-Хеллмана в общем случае. Наконец Шамир и Циппел (Zippel) [1418, 1419, 1421] обнаружили слабые места в преобразовании, что позволило им восстановить сверхвозрастающую последовательность рюкзака по нормальной. Точные доказательства выходят за рамки этой книги, но их хороший обзор можно найти в [1233, 1244]. На конференции, где докладывались эти результаты, вскрытие было продемонстрировано по стадиям на компьютере Apple II [492, 494].

Варианты рюкзака

После вскрытия оригинальной схемы Меркла-Хеллмана было предложено множество других систем на принципе рюкзака: несколько последовательных рюкзаков, рюкзаки Грэм-Шамира (Graham-Shamir), и другие. Все они были проанализированы и взломаны, как правило, с использованием одних и тех же криптографических методов, и их обломки были сметены со скоростного шоссе криптографии [260, 253, 269, 921, 15, 919, 920, 922, 366, 254, 263, 255]. Хороший обзор этих систем и их криптоанализ можно найти в [267, 479, 257, 268].

Были предложены и другие алгоритмы, использующие похожие идеи, но все они тоже были взломаны. Криптосистема Lu-Lee [990, 13] была взломана в [20, 614, 873], ее модификация [507] также оказалась небезопасной [1620]. Вскрытия криптосистемы Goodman-McAuley приведены в [646, 647, 267, 268]. Криптосистема Pieprzyk [1246] была взломана аналогичным образом. Криптосистема Niemi [1169], основанная на модульных рюкзаках, взломана в [345, 788]. Новый, многостадийный рюкзак [747] пока еще не был взломан, но я не оптимистичен. Другим вариантом является [294].

Хотя вариант алгоритма рюкзака в настоящее время безопасен - алгоритм рюкзака Char-Rivest [356], несмотря на "специализированное вскрытие" [743] - количество необходимых вычислений делает его намного менее полезным, чем другие рассмотренные здесь алгоритмы. Вариант, названный Powerline System (система электропитания) небезопасен [958]. Более того, учитывая легкость с которой пали все остальные варианты, доверять устоявшим пока вариантом, по видимому, неосторожно.

Патенты

Оригинальный алгоритм Меркла-Хеллмана запатентован в Соединенных Штатах [720] и в остальном мире (см. 18th). Public Key Partners (PKP) получила лицензию на патент вместе с другими патентами криптографии с открытыми ключами (см. раздел 25.5). Время действия патента США истечет 19 августа 1997 года.

Табл. 19-1.
Иностранные патенты на алгоритм рюкзака Меркла-Хеллмана

Страна	Номер	Дата получения
Бельгия	871039	5 апреля 1979 года
Нидерланды	7810063	10 апреля 1979 года
Великобритания	2006580	2 мая 1979 года
Германия	2843583	10 мая 1979 года
Швеция	7810478	14 мая 1979 года
Франция	2405532	8 июня 1979 года
Германия	2843583	3 января 1982 года
Германия	2857905	15 июля 1982 года
Канада	1128159	20 июля 1982 года
Великобритания	2.006580	18 августа 1982 года
Швейцария	63416114	14 января 1983 года
Италия	1099780	28 сентября 1985 года

19.3 RSA

Вскоре после алгоритма рюкзака Меркла появился первый полноценный алгоритм с открытым ключом , который можно использовать для шифрования и цифровых подписей : RSA [1328, 1329]. Из всех предложенных за эти годы алгоритмов с открытыми ключами RSA проще всего понять и реализовать. (Мартин Гарднер (Martin Gardner) опубликовал раннее описание алгоритма в своей колонке "Математические игры" в *Scientific American* [599].) Он также является самым популярным. Названный в честь трех изобретателей - Рона Ривеста (Ron Rivest), Ади Шамира (Adi Shamir) и Леонарда Эдлмана (Leonard Adleman) - этот алгоритм многие годы противостоит интенсивному криптоанализу. Хотя криптоанализ ни доказал, ни опроверг безопасность RSA, он, по сути, обосновывает уровень доверия к алгоритму.

Безопасность RSA основана на трудности разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших (100 - 200 разрядов или даже больше) простых чисел. Предполагается, что восстановление открытого текста по шифротексту и открытому ключу эквивалентно разложению на множители двух больших чисел.

Для генерации двух ключей используются два больших случайных простых числа , p и q. Для максимальной безопасности выбирайте p и q равной длины. Рассчитывается произведение:

$n = p\ q$

Затем случайным образом выбирается ключ шифрования e, такой что e и $(p\text{-}1)(q\text{-}1)$ являются взаимно простыми числами. Наконец расширенный алгоритм Эвклида используется для вычисления ключа дешифрирования d, такого что

$ed = 1 \pmod{(p\text{-}1)(q\text{-}1)}$

Другими словами

$d = e^{-1} \bmod ((p\text{-}1)(q\text{-}1))$

Заметим, что d и n также взаимно простые числа. Числа e и n - это открытый ключ, а число d - закрытый. Два простых числа p и q больше не нужны. Они должны быть отброшены, но не должны быть раскрыты.

Для шифрования сообщения m оно сначала разбивается на цифровые блоки, меньшие n (для двоичных данных выбирается самая большая степень числа 2, меньшая n). То есть, если p и q - 100-разрядные простые числа, то n будет содержать около 200 разрядов, и каждый блок сообщения m_i должен быть около 200 разрядов в длину. (Если нужно зашифровать фиксированное число блоков, их можно дополнить несколькими нулями слева, чтобы гарантировать, что блоки всегда будут меньше n. Зашифрованное сообщение c будет состоять из блоков c_i той же самой длины. Формула шифрования выглядит так

$c_i = m_i^e \bmod n$

Для расшифровки сообщения возьмите каждый зашифрованный блок c_i и вычислите

$m_i = c_i^d \bmod n$

Так как

$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p\text{-}1)(q\text{-}1)+1} = m_i m_i^{k(p\text{-}1)(q\text{-}1)} = m_i * 1 = m_i;$ все $(\bmod n)$

формула восстанавливает сообщение. Это сведено в 17-й.

Табл. 19-2.
Шифрование RSA

Открытый ключ:	
n	произведение двух простых чисел p и q (p и q должны храниться в секрете)
e	число, взаимно простое с $(p\text{-}1)(q\text{-}1)$
Закрытый ключ:	
d	$e^{-1} \bmod ((p\text{-}1)(q\text{-}1))$
Шифрование:	
$c = m^e \bmod n$	
Дешифрирование:	
$m = c^d \bmod n$	

Точно также сообщение может быть зашифровано с помощью d, а зашифровано с помощью e, возможен любой выбор. Я уберегу вас от теории чисел, доказывающей, почему этот алгоритм работает. В большинстве

книг по криптографии этот вопрос подробно рассмотрен .

Короткий пример возможно поможет пояснить работу алгоритма . Если $p = 47$ и $q = 71$, то

$n = pq = 3337$

Ключ e не должен иметь общих множителей

$(p-1)(q-1)= 46*70 =3220$

Выберем (случайно) e равным 79. В этом случае $d = 79^{-1} \bmod 3220 = 1019$

При вычислении этого числа использован расширенный алгоритм Эвклида (см. раздел 11.3). Опубликуем e и n, сохранив в секрете d. Отбросим p и q. Для шифрования сообщения

m = 6882326879666683

сначала разделим его на маленькие блоки . Для нашего случая подойдут трехбуквенные блоки . Сообщение разбивается на шесть блоков m_i:

$m_1 = 688$

$m_2 = 232$

$m_3 = 687$

$m_4 = 966$

$m_5 = 668$

$m_6 = 003$

Первый блок шифруется как $688^{79} \bmod 3337 = 1570 = c_1$

Выполняя те же операции для последующих блоков, создает шифротекст сообщения :

c = 1570 2756 2091 2276 2423 158

Для дешифрирование нужно выполнить такое же возведение в степень, используя ключ дешифрирования 1019:

$1570^{1019} \bmod 3337 = 688 = m_1$

Аналогично восстанавливается оставшаяся часть сообщения .

Аппаратные реализации RSA

Существует много публикаций, затрагивающих тему аппаратных реализаций RSA [1314, 1474, 1456, 1316, 1485, 874, 1222, 87, 1410, 1409, 1343, 998, 367, 1429, 523, 772]. Хорошими обзорными статьями служат [258, 872]. Шифрование RSA выполняется многими микросхемами [1310, 252, 1101, 1317, 874, 69, 737, 594, 1275, 1563, 509, 1223]. Частичный список доступных в настоящее время микросхем RSA, взятый из [150, 258], приведен в 16th. Не все из них доступны в свободной продаже .

Табл. 19-3.
Существующие микросхемы RSA

Компания	Тактовая частота	Скорость передачи в Бодах на 512 бит	Тактовые циклы для шифрования 512 бит	Технология	Битов на микросхему	Количество транзисторов
Alpha Techn.	25 МГц	13K	0.98 M	2 микрона	1024	180000
AT&T	15 МГц	19K	0.4 M	1.5 микрона	298	100000
British Telecom	10 МГц	5.IK	1 M	2.5 микрона	256	-----
Business Sim. Ltd.	5 МГц	3.8K	0.67 M	Вентильная матрица	32	-----
CalmosSyst-Inc.	20 МГц	2.8K	0.36 M	2 микрона	593	95000
CNET	25 МГц	5.3K	2.3 M	1 микрон	1024	100000
Cryptech	14 МГц	17K	0.4 M	Вентильная матрица	120	33000
Cylink	30 МГц	6.8K	1.2 M	1.5 микрона	1024	150000
GEC Marconi	25 МГц	10.2K	0.67 M	1.4 микрона	512	160000
Pijnenburg	25 МГц	50K	0.256 M	1 микрон	1024	400000
Sandia	8 МГц	IOK	0.4 M	2 микрона	272	86000
Siemens	5 МГц	8.5K	0.03 M	1 микрон	512	60000

Скорость RSA

Аппаратно RSA примерно в 1000 раз медленнее DES. Скорость работы самой быстрой СБИС-реализации RSA с 512-битовым модулем - 64 килобита в секунду [258]. Существуют также микросхемы, которые выполня-

ют 1024-битовое шифрование RSA. В настоящее время разрабатываются микросхемы, которые, используя 512-битовый модуль, приблизятся к рубежу 1 Мбит/с. Возможно, они появятся в 1995 году. Производители также применяют RSA в интеллектуальных карточках, но эти реализации медленнее.

Программно DES примерно в 100 раз быстрее RSA. Эти числа могут незначительно измениться при изменении технологии, но RSA никогда не достигнет скорости симметричных алгоритмов. В 15-й приведены примеры скоростей программного шифрования RSA [918].

Табл. 19-4.
Скорости RSA для различных длин модулей при 8-битовом открытом ключе (на SPARC II)

	512 битов	768 битов	1024 бита
Шифрование	0.03 с	0.05 с	0.08 с
Дешифрирование	0.16 с	0.48 с	0.93 с
Подпись	0.16 с	0.52 с	0.97 с
Проверка	0.02 с	0.07 с	0.08 с

Программные Speedups

Шифрование RSA выполняется намного быстрей, если вы правильно выберете значение e. Тремя наиболее частыми вариантами являются 3, 17 и 65537 ($2^{16} + 1$). (Двоичное представление 65537 содержит только две единицы, поэтому для возведения в степень нужно выполнить только 17 умножений.) X.509 советует 65537 [304], PEM рекомендует 3 [76], а PKCS #l (см. раздел 24.14) - 3 или 65537 [1345]. Не существует никаких проблем безопасности, связанных с использованием в качестве e любого из этих трех значений (при условии, что вы дополняете сообщения случайными числами - см. раздел ниже), даже если одно и то же значение e используется целой группой пользователей.

Операции с закрытым ключом можно ускорить при помощи китайской теоремы от остатках, если вы сохранили значения p и q, а также дополнительные значения: $d \bmod (p - 1)$, $d \bmod (q - 1)$ и $q^{-1} \bmod p$ [1283, 1276]. Эти дополнительные числа можно легко вычислить по закрытому и открытому ключам.

Безопасность RSA

Безопасность RSA полностью зависит от проблемы разложения на множители больших чисел. Технически, это утверждение о безопасности лживо. Предполагается, что безопасность RSA зависит от проблемы разложения на множители больших чисел. Никогда не было доказано математически, что нужно разложить n на множители, чтобы восстановить m по c и e. Понятно, что может быть открыт совсем иной способ криптоанализа RSA. Однако, если этот новый способ позволит криптоаналитику получить d, он также может быть использован для разложения на множители больших чисел. Я не слишком волнуюсь об этом.

Также можно вскрыть RSA, угадав значение $(p-1)(q-1)$. Это вскрытие не проще разложения n на множители [1616].

Для сверхскептиков: доказано, что некоторые варианты RSA также сложны, как и разложение на множители (см. раздел 19.5). Загляните также в [361, где показано, что раскрытие даже нескольких битов информации по зашифрованному RSA шифротексту не легче, чем дешифрирование всего сообщения.

Самым очевидным средством вскрытия является разложение n на множители. Любой противник сможет получить открытый ключ e и модуль n. Чтобы найти ключ дешифрирования d, противник должен разложить n на множители. Современное состояние технологии разложения на множители рассматривалось в разделе 11.4. В настоящее время передним краем этой технологии является число, содержащее 129 десятичных цифр. Значит, n должно быть больше этого значения. Рекомендации по выбору длины открытого ключа приведены в разделе 7.2.

Конечно, криптоаналитик может перебирать все возможные d, пока он не подберет правильное значение. Но такое вскрытие грубой силой даже менее эффективно, чем попытка разложить n на множители.

Время от времени появляются заявления о том, что найден простой способ вскрытия RSA, но пока ни одно из подобных заявлений не подтвердилось. Например, в 1993 году в черновике статьи Вильяма Пейна (William Payne) был предложен метод, основанный на малой теореме Ферма [1234]. К сожалению, этот метод оказался медленнее разложения на множители

Существует еще один повод для беспокойства. Большинство общепринятых алгоритмов вычисления простых чисел p и q вероятностны, что произойдет, если p или q окажется составным? Ну, во первых, можно свести вероятность такого события до нужного минимума. И даже если это произойдет, скорее всего такое событие будет

сразу же обнаружено - шифрование и дешифрирование не будут работать. Существует ряд чисел, называемых числами Кармайкла (Carmichael), которые не могут обнаружить определенные вероятностные алгоритмы поиска простых чисел. Они небезопасны, но чрезвычайно редки [746]. Честно говоря, меня бы это не обеспокоило.

Вскрытие с выбранным шифротекстом против RSA

Некоторые вскрытия работают против реализаций RSA. Они вскрывают не сам базовый алгоритм, а надстроенный над ним протокол. Важно понимать, что само по себе использование RSA не обеспечивает безопасности. Дело в реализации.

Сценарий 1: Еве, подслушавшей линии связи Алисы, удалось перехватить сообщение c, шифрованное с помощью RSA открытым ключом Алисы. Ева хочет прочитать сообщение. На языке математики, ей нужно m, для которого

$m = c^d$

Для раскрытия m она сначала выбирает первое случайное число r, меньшее n. Она достает открытый ключ Алисы e. Затем она вычисляет

$x = r^e \bmod n$

$y = xc \bmod n$

$t = r^{-1} \bmod n$

Если $x = r^e \bmod n$, то $r = x^d \bmod n$.

Теперь просит Алису подписать y ее закрытым ключом, таким образом расшифровав y. (Алиса должна подписать сообщение, а не его хэш сумму.) Не забывайте, Алиса никогда раньше не видела y. Алиса посылает Еве

$u = y^d \bmod n$

Теперь Ева вычисляет

$tu \bmod n = r^{-1} y^d \bmod = r^{-1} x^d c^d \bmod n = c^d \bmod n = m$

И Ева получает m.

Сценарий 2: Трент - это компьютер-нотариус. Если Алиса хочет заверить документ, она посылает его Тренту. Трент подписывает его цифровой подписью RSA и отправляет обратно. (Однонаправленные хэш-функции не используются, Трент шифрует все сообщение своим закрытым ключом.)

Мэллори хочет, чтобы Трент подписал такое сообщение, которое в обычном случае он он никогда не подпишет. Может быть это фальшивая временная метка, может быть автором этого сообщения является другое лицо. Какой бы ни была причина, Трент никогда не подпишет это сообщение, если у него будет возможность выбора. Назовем это сообщение m'.

Сначала Мэллори выбирает произвольное значение x и вычисляет $y = x^e \bmod n$. e он может получить без труда - это открытый ключ Трента, который должен быть опубликован, чтобы можно было проверять подписи Трента. Теперь Мэллори вычисляет $m = ym' \bmod n$ и посылает m Тренту на подпись. Трент возвращает $m^d \bmod n$. Now Мэллори вычисляет $(m^d \bmod n)x^{-1} \bmod n$, которое равно $n^d \bmod n$ и является подписью m'.

На самом деле Мэллори может использовать множество способов решить подобную задачу [423, 458, 486]. Слабым местом, которое используют такие вскрытия, является сохранение мультипликативной структуры входа при возведении в степень. То есть:

$(xm)^d \bmod n = x^d m^d \bmod n$

Сценарий 3: Ева хочет, чтобы Алиса подписала m_3. Она создает два сообщения, m_1 и m_2, такие что

$m_3 = m_1 m_2 \pmod n$

Если Ева сможет заставить Алису подписать m_1 и m_2, она может вычислить подпись для m_3:

$m_3^d = (m_1^d \bmod n)\ (m_2^d \bmod n)$

Мораль: Никогда не пользуйтесь алгоритмом RSA для подписи случайных документов, подсунутых вам посторонними. Всегда сначала воспользуйтесь однонаправленной хэш-фунцией. Формат блоков ISO 9796 предотвращает это вскрытие.

Вскрытие общего модуля RSA

При реализации RSA можно попробовать раздать всем пользователям одинаковый модуль n, но каждому свои значения показателей степени e и d. К сожалению, это не работает. Наиболее очевидная проблема в том,

что если одно и то же сообщение когда-нибудь шифровалось разными показателями степени (с одним и тем же модулем), и эти два показателя - взаимно простые числа (как обычно и бывает), то открытый текст может быть раскрыт, даже не зная ни одного ключа дешифрирования [1457].

Пусть m - открытый текст сообщения. Два ключа шифрования - e_1 и e_2. Общий модуль - n. Шифротекстами сообщения являются:

$$c_1 = m^{e_1} \bmod n$$

$$c_2 = m^{e_2} \bmod n$$

Криптоаналитик знает n, e_1, e_2, c_1 и c_2. Вот как он узнает m.

Так как e_1 и e_2 - взаимно простые числа, то с помощью расширенного алгоритма Эвклида r и s, для которых

$$re_1 + se_2 = 1$$

Считая r отрицательным (или r, или s должно быть отрицательным, пусть отрицательным будет r), то снова можно воспользоваться расширенным алгоритмом для вычисления c_1^{-1}. Затем

$$(c_1^{-1})^{-r} * c_2^{s} = m \bmod n$$

Существует два других, более тонких вскрытия систем такого типа. Одно использует вероятностный метод для разложения n на множители. Другой - детерминированный алгоритм вычисления какого-нибудь секретного ключа без разложения модуля на множители. Оба вскрытия подробно описаны в [449].

Мораль: Не делайте n общим для группы пользователей.

Вскрытие малого показателя шифрования RSA

Шифрование и проверка подписи RSA выполняется быстрее, если для e используется небольшое значение, но это также может быть небезопасным [704]. Если $e(e + 1)/2$ линейно зависящих сообщений с различными открытыми ключами шифруются одним и тем же значением e, существует способ вскрыть такую систему. Если сообщений не так много, или если сообщения не связаны, то проблем нет. Если сообщения одинаковы, то достаточно e сообщений. Проще всего дополнять сообщения независимыми случайными числами.

Это также гарантирует, что $m^e \bmod n \neq m^e$. Так делается в большинстве практических реализаций RSA, например, в PEM и PGP (см. разделы 24.10 и 24.12).

Мораль: Дополняйте сообщения перед шифрованием случайными значениями, убедитесь, что размер m примерно равен n.

Вскрытие малого показателя дешифрирования RSA

Другим вскрытием, предложенным Майкл Винер (Michael Wiener), раскрывает d, где d не превышает четверти размера n, а e меньше n [1596]. При случайном выборе e и d это встречается редко, и никогда не произойдет, если значение e мало.

Мораль: Выбирайте большое значение d.

Полученные уроки

Джудит Мур (Judith Moore) на основании перечисленных вскрытий приводит следующие ограничения RSA [1114, 1115]:

— Знание одной пары показателей шифрования/дешифрирования для данного модуля позволяет взломщику разложить модуль на множители.

— Знание одной пары показателей шифрования/дешифрирования для данного модуля позволяет взломщику вычислить другие пары показателей, не раскладывая модуль на множители.

— В протоколах сетей связи, применяющих RSA, не должен использоваться общий модуль. (Это является быть очевидным следствием предыдущих двух пунктов.)

— Для предотвращения вскрытия малого показателя шифрования сообщения должны быть дополнены случайными значениями.

— Показатель дешифрирования должен быть большим.

Не забывайте, недостаточно использовать безопасный криптографический алгоритм, должны быть безопасными вся криптосистема и криптографический протокол. Слабое место любого из трех этих компонентов сделает небезопасной всю систему.

Вскрытие шифрования и подписи с использованием RSA

Имеет смысл подписывать сообщение перед шифрованием (см. раздел 2.7), но на практике никто не выполняет этого. Для RSA можно вскрыть протоколы, шифрующие сообщение до его подписания [48].

Алиса хочет послать сообщение Бобу. Сначала она шифрует его открытым ключом Боба, а затем подписывает своим закрытым ключом. Ее зашифрованное и подписанное сообщение выглядит так:

$$m^{e_B} \bmod n_B)^{d_A} \bmod n_A$$

Вот как Боб может доказать, что Алиса послала ему m', а не m. Так как Бобу известно разложение на множители n_B (это его собственный модуль), он может вычислить дискретные логарифмы по основанию n_B. Следовательно, ему нужно только найти x, для которого

$$m^x = m \bmod n_B$$

Тогда, если он может опубликовать xe_B в качестве своего нового открытого показателя степени и сохранить свой прежний модуль n_B, он сможет утверждать, что Алиса послала ему сообщение m', зашифрованное этим новым показателем.

В некоторых случаях это особенно неприятное вскрытие. Заметим, что хэш-функции не решают проблему. Однако она решается при использовании для каждого пользователя фиксированного показ ателя шифрования.

Стандарты

RSA *de facto* является стандартом почти по всему миру. ISO почти, but not quite, created an RSA digital-signature standard; RSA служит информационным дополнением ISO 9796 [762.]. Французское банковское сообщество приняло RSA в качестве стандарта [525], так же поступили и австралийцы [1498]. В Соединенных Штатах из-за давления NSA и патентных вопросов в настоящее время нет стандарта для шифрования с открытым ключом. Многие американские компании используют PKCS (см. раздел 24.14), написанный RSA Data Security, Inc. RSA определен и в качестве чернового банковского стандарта ANSI [61].

Патенты

Алгоритм RSA запатентован в Соединенных Штатах [1330], но ни водной другой стране. PKP получила лицензию вместе с другими патентами в области криптографии с открытыми ключами (раздел 25.5). Срок действия патента США истекает 20 сентября 2000 года.

19.4 Pohlig-Hellman

Схема шифрования Pohlig-Hellman [1253] похожа на RSA. Это не симметричный алгоритм, так как для шифрования и дешифрирования используются различные ключи. Это не схема с открытым ключом, потому что ключи легко получаются один из другого, и ключ шифрования, и ключ дешифрирования должны храниться в секрете. Как и в RSA,

$$C = P^e \bmod n$$

$$P = C^d \bmod n$$

где

$ed \equiv 1$ (mod какое-нибудь составное число)

В отличие от RSA n не определяется с помощью двух простых чисел и остается частью закрытого ключа. Если у кого-нибудь есть e и n, он может вычислить d. Не зная e или d, противник будет вынужден вычислить

$$e = \log_p C \bmod n$$

Мы уже видели, что это является трудной проблемой.

Патенты

Алгоритм Pohlig-Hellman запатентован в США [722] и в Канаде. PKP получила лицензию вместе с другими патентами в области криптографии с открытыми ключами (см. раздел 25.5).

19.5 Rabin

Безопасность схемы Рабина (Rabin) [1283, 1601] опирается на сложность поиска квадратных корней по модулю составного числа. Эта проблема аналогична разложению на множители. Вот одна из реализаций этой схемы.

Сначала выбираются два простых числа p и q, конгруэнтных 3 mod 4. Эти простые числа являются закрытым ключом, а их произведение $n = pq$ - открытым ключом.

Для шифрования сообщения M (M должно быть меньше n), просто вычисляется

$C = M^2 \bmod n$

Дешифрирование сообщения также несложно, но немного скучнее. Так как получатель знает p и q, он может решить две конгруэнтности с помощью китайской теоремы об остатках. Вычисляется

$m_1 = C^{(p+1)/4} \bmod p$

$m_2 = (p - C^{(p+1)/4}) \bmod p$

$m_3 = C^{(q+1)/4} \bmod q$

$m_4 = (q - C^{(q+1)/4}) \bmod q$

Затем выбирается целые числа $a = q(q^{-1} \bmod p)$ и $b = p(p^{-1} \bmod q)$. Четырьмя возможными решениями являются:

$M_1 = (am_1 + bm_3) \bmod n$

$M_2 = (am_1 + bm_4) \bmod n$

$M_3 = (am_2 + bm_3) \bmod n$

$M_4 = (am_2 + bm_4) \bmod n$

Один из четырех результатов, M_1, M_2, M_3 и M_4, равно M. Если сообщение написано по английски, выбрать правильное M_i нетрудно. С другой стороны, если сообщение является потоком случайных битов (скажем, для генерации ключей или цифровой подписи), способа определить, какое M_i - правильное, нет. Одним из способов решить эту проблему служит добавление к сообщению перед шифрованием известного заголовка.

Williams

Хью Вильямс (Hugh Williams) переопределил схему Рабина, чтобы устранить эти недостатки [1601]. В его схеме p и q выбираются так, чтобы

$p \equiv 3 \bmod 8$

$q \equiv 7 \bmod 8$

и

$N = pq$

Кроме того, используется небольшое целое число, S, для которого $J(S,N) = -1$. (J - это символ Якоби - см. раздел I I .3). N и S опубликовываются. Секретным ключом является k, для которого

$k = 1/2\ (1/4\ (p - 1)\ (q - 1) + 1)$

Для шифрования сообщения M вычисляется c_1, такое что $J(M,N) = (-1)^{c_1}$. Затем вычисляется $M' = (S^{c_1} * M) \bmod N$. Как и в схеме Рабина, $C = M'^2 \bmod N$. И $c_2 = M' \bmod 2$. Окончательным шифротекстом сообщения является тройка:

(C, c_1, c_2)

Для дешифрирования C, получатель вычисляет M'' с помощью

$C^k \equiv \pm M'' \pmod N$

Правильный знак M'' определяет c_2. Наконец

$M = (S^{c_1} * (-1)^{c_1} * M'') \bmod N$

Впоследствии Вильямс улучшил эту схему в [1603, 1604, 1605]. Вместо возведения в квадрат открытого текста сообщения, возведите его в третью степени. Большие простые числа должны быть конгруэнтны 1 по модулю 3, иначе открытый и закрытый ключи окажутся одинаковыми. Даже лучше, существует только одна уникальная расшифровка каждого шифрования.

Преимущество схем Рабина и Вильямса перед RSA в том, что доказано, что они также безопасны, как и разложение на множители. Однако перед вскрытием с выбранным шифротекстом они совершенно беззащитны. Если вы собираетесь использовать эти схемы для случаев, когда взломщик может выполнить такое вскрытие (например, алгоритм цифровой подписи, когда взломщик может выбирать подписываемые сообщения), не за-

бывайте использовать перед подписанием однонаправленную хэш-функцию. Рабин предложил другой способ защититься от такого вскрытия: к каждому сообщению перед хэшированием и подписанием добавляется уникальная случайная строка. К несчастью, после добавления однонаправленной хэш-функцией тот факт, что система столь же безопасна, как и разложение на множители, больше не является доказанным [628]. Хотя с практической точки зрения добавление хэширования не может ослабить систему.

Другими вариантами схемы Рабина являются [972, 909, 696, 697, 1439, 989]. Двумерный вариант описан в [866, 889].

19.6 ElGamal

Схему ElGamal [518,519] можно использовать как для цифровых подписей, так и для шифрования, его безопасность основана на трудности вычисления дискретных логарифмов в конечном поле.

Для генерации пары ключей сначала выбирается простое число p и два случайных числа, g и x, оба эти числа должны быть меньше p. Затем вычисляется

$y = g^x \bmod p$

Открытым ключом являются y, g и p. И g, и p можно сделать общими для группы пользователей. Закрытым ключом является x.

Подписи ElGamal

Чтобы подписать сообщение M, сначала выбирается случайное число k, взаимно простое с p-1. Затем вычисляется

$a = g^k \bmod p$

и с помощью расширенного алгоритма Эвклида находится b в следующем уравнении:

$M = (xa + kb) \bmod (p - 1)$

Подписью является пара чисел: a и b. Случайное значение k должно храниться в секрете. Для проверки подписи нужно убедиться, что

$y^a a^b \bmod p = g^M \bmod p$

Каждая подпись или шифрование ElGamal требует нового значения k, и это значение должно быть выбрано случайным образом. Если когда-нибудь Ева раскроет k, используемое Алисой, она сможет раскрыть закрытый ключ Алисы x. Если Ева когда-нибудь сможет получить два сообщения, подписанные или зашифрованные с помощью одного и того же k, то она сможет раскрыть x, даже не зная значение k. Описание ElGamal сведено в 14-й.

Табл. 19-5.
Подписи ElGamal

Открытый ключ:	
p	простое число (может быть общим для группы пользователей)
g	$<p$ (может быть общим для группы пользователей)
y	$=g^x \bmod p$
Закрытый ключ:	
x	$<p$
Подпись:	
k	выбирается случайным образом, взаимно простое с p-1
a	(подпись) $=g^k \bmod p$
b	(подпись), такое что $M = (xa + kb) \bmod (p - 1)$
Проверка:	
Подпись считается правильной, если $y^a a^b \bmod p = g^M \bmod p$	

Например, выберем $p = 11$ и $g = 2$, а закрытый ключ $x = 8$. Вычислим

$y = g^x \bmod p = 2^8 \bmod 11 = 3$

Открытым ключом являются $y = 3$, $g = 2$ и $p = 11$. Чтобы подписать $M = 5$, сначала выберем случайное число k=9. Убеждаемся, что gcd(9, 10)= 1. Вычисляем

$a = g^k \bmod p = 2^9 \bmod 11 = 6$

и с помощью расширенного алгоритма Эвклида находим b:

$M = (xa + kb) \bmod (p - 1)$

$5 = (8*6 + 9*b) \bmod 10$

Решение: $b = 3$, а подпись представляет собой пару: $a = 6$ и $b = 3$.

Для проверки подписи убедимся, что

$y^a a^b \bmod p = g^M \bmod p$

$3^6 6^3 \bmod 11 = 2^5 \bmod 11$

Вариант ElGamal, используемый для подписей, описан в [1377]. Томас Бет (Thomas Beth) изобрел вариант схемы ElGamal, подходящий для доказательства идентичности [146]. Существуют варианты для проверки подлинности пароля [312] и для обмена ключами [773]. И еще тысячи и тысячи других (см. раздел 20.4).

Шифрование ElGamal

Модификация ElGamal позволяет шифровать сообщения. Для шифрования сообщения M сначала выбирается случайное число k, взаимно простое с p - 1. Затем вычисляются

$a = g^k \bmod p$

$b = y^k M \bmod p$

Пара (a,b) является шифротекстом. Обратите внимание, что шифротекст в два раза длиннее открытого текста. Для дешифрирования (a,b) вычисляется

$M = b/a^x \bmod p$

Так как $a^x \equiv g^{kx} \pmod p$ и $b/a^x \equiv y^k M/a^x \equiv g^{xk} M/ g^{kx} = M \pmod p$, то все работает (см. 13-й). По сути это то же самое, что и обмен ключами Диффи-Хеллмана (см. раздел 22.1) за исключением того, что y - это часть ключа, а при шифровании сообщение умножается на y^k.

Табл. 19-6.
Шифрование ElGamal

	Открытый ключ:
p	простое число (может быть общим для группы пользователей)
g	$<p$ (может быть общим для группы пользователей)
y	$=g^x \bmod p$
	Закрытый ключ:
x	$<p$
	Шифрование:
k	выбирается случайным образом, взаимно простое с p-1
a	(шифротекст) $=g^k \bmod p$
b	(шифротекст)$= y^k M \bmod p$
	Дешифрирование:
M (открытый текст)	$= b/a^x \bmod p$

Скорость

Некоторые примеры скорости работы программных реализаций ElGamal приведены в 12-й [918].

Табл. 19-7.

Скорости ElGamal для различных длин модулей при 160-битовом показателе степени (на SPARC II)

	512 битов	768 битов	1024 битов
Шифрование	0.33 с	0.80 с	1.09 с
Дешифрирование	0.24 с	0.58 с	0.77 с
Подпись	0.25 с	0.47 с	0.63 с
Проверка	l.37 с	5.12 с	9.30 с

Патенты

ElGamal незапатентован. Но, прежде чем двигаться вперед и реализовывать алгоритм, нужно знать, что PKP считает, что этот алгоритм попадает под действие патента Диффи-Хеллмана [718]. Однако срок действия патента Диффи-Хеллмана заканчивается 29 апреля 1997 года, что делает ElGamal первым криптографическим плгоритмом с открытыми ключами, пригодным для шифрования и цифровых подписей и несвязанным в Соедине нных Штатах патентами. Я не могу дождаться этого момента.

19.7 McEliece

В 1978 году Роберт МакЭлис (Robert McEliece) разработал криптосистему с открытыми ключами на основе теории алгебраического кодирования [1041]. Этот алгоритм использует существование определенного класса исправляющих ошибки кодов, называемых **кодами Гоппа** (Goppa). Он предлагал создать код Гоппа и замаскировать его как обычный линейный код. Существует быстрый алгоритм декодирования кодов Гоппа, но общая проблема найти слово кода по данному весу в линейном двоичном коде является **NP-полной**. Хорошее описание этого алгоритма можно найти в [1233], см. также [1562]. Ниже приведен только краткий обзор.

Пусть $d_H(x,y)$ обозначает расстояние Хэмминга между x и y. Числа n, k и t служат параметрами системы.

Закрытый ключ состоит из трех частей: G' - это матрица генерации года Гоппа, исправляющего t ошибок. P - это матрица перестановок размером $n*n$. S - это nonsingular матрица размером $k*k$.

Открытым ключом служит матрица G размером $k*n$: $G = SG'P$.

Открытый текст сообщений представляет собой строку k битов в виде k-элементного вектора над полем GF(2).

Для шифрования сообщения случайным образом выбирается n-элементный вектор z над полем GF(2), для которого расстояние Хэмминга меньше или равно t.

$c = mG + z$

Для дешифрирования сообщения сначала вычисляется $c' = cP^{-1}$. Затем с помощью декодирующего алгоритма для кодов Гоппа находится m', для которого $d_H(m'G,c)$ меньше или равно t. Наконец вычисляется $m = m'S^{-1}$.

В своей оригинальной работе МакЭлис предложил значения n = 1024, t = 50 и k = 524. Это минимальные значения, требуемые для безопасности.

Хотя этот алгоритм был одним из первых алгоритмов с открытыми ключами, и вне появлялось публикаций о его успешном криптоаналитическом вскрытии, он не получил широкого признания в криптографическом с ообществе. Схема на два-три порядка быстрее, чем RSA, но у нее есть ряд недостатков. Открытый ключ огромен: 2^{19} битов. Сильно увеличивается объем данных - шифротекст в два раза длиннее открытого текста.

Ряд попыток криптоанализа этой системы можно найти в [8, 943, 1559, 306]. Ни одна из них не достигла успеха для общего случая, хотя сходство между алгоритмом МакЭлиса и алгоритмом рюкзака немного волнует.

В 1991 два русских криптографа заявили, что взломали систему МакЭлиса с некоторыми параметрами [882]. В их статье это утверждение не было обосновано, и большинство криптографов не приняли во внимание этот результат. Еще одно выполненное русскими вскрытие, которое нельзя непосредственно использовать против системы МакЭлиса, описано в [1447, 1448]. Расширения McEliece можно найти в [424, 1227, 976].

Другие алгоритмы, основанные на линейных кодах, исправляющих ошибки

Алгоритм Нидеррейтера (Niederreiter) [1167] очень близок к алгоритму МакЭлиса и считает, что открытый ключ - это случайная матрица проверки четности кода, исправляющего ошибки. Закрытым ключом служит эффективный алгоритм декодирования этой матрицы.

Другой алгоритм, используемый для идентификации и цифровых подписей, основан на декодировании

синдрома [1501], пояснения см. в [306]. Алгоритм [1621], использующий коды, исправляющие ошибки, небезопасен [698, 33, 31, 1560, 32].

19.8 Криптосистемы с эллиптическими кривыми

Эллиптические кривые изучались многие годы, и по этому вопросу существует огромное количество литературы. В 1985 году Нил Коблиц (Neal Koblitz) и В.С. Миллер (V. S. Miller) независимо предложили использовать их для криптосистем с открытыми ключами [867, 1095]. Они не изобрели нового криптографического алгоритма, использующего эллиптические кривые над конечными полями, но реализовали существующие алгоритмы, подобные Diffie-Hellman, с помощью эллиптических кривых.

Эллиптические кривые вызывают интерес, потому что они обеспечивают способ конструирования "элементов" и "правил объединения", образующих группы. Свойства этих групп известны достаточно хорошо, чтобы использовать их для криптографических алгоритмов, но у них нет определенных свойств, облегчающих криптоанализ. Например, понятие "гладкости" неприменимо к эллиптическим кривым. То есть, не существует такого множества небольших элементов, используя которые с помощью простого алгоритма с высокой вероятностью можно выразить случайный элемент. Следовательно, алгоритмы вычисления дискретного логарифма показателя степени не работают work. Подробности см. в [1095].

Особенно интересны эллиптические кривые над полем $GF(2^n)$. Для n в диапазоне от 130 до 200 несложно разработать схему и относительно просто реализовать арифметический процессор для используемого поля. Такие алгоритмы потенциально могут послужить основой для более быстрых криптосистем с открытыми ключами и меньшими размерами ключей. С помощью эллиптических кривых над конечными полями могут быть реализованы многие алгоритмы с открытыми ключами, такие как Diffie-Hellman, EIGamal и Schnorr.

Соответствующая математика сложна и выходит за рамки этой книги. Интересующимся этой темой я предлагаю прочитать две вышеупомянутые работы и отличную книгу Альфреда Менезеса (Alfred Menezes) [1059]. Эллиптические кривые используются двумя аналогами RSA [890, 454]. Другими работами являются [23, 119, 1062, 869, 152, 871, 892, 25, 895, 353, 1061, 26, 913, 914, 915]. Криптосистемы с ключами небольшой длины на базе эллиптических кривых рассматриваются в [701]. Алгоритм Fast Elliptic Encryption (FEE, быстрое эллиптическое шифрование) компании Next Computer Inc. также использует эллиптические кривые [388]. Приятной особенностью FEE является то, что закрытый ключ может быть любой легко запоминающейся строкой. Предлагаются и криптосистемы, использующие гиперэллиптические кривые [868, 870, 1441, 1214].

19.9 LUC

Некоторые криптографы разработали обобщенные модификации RSA, которые используют различные перестановочные многочлены вместо возведения в степень. Вариант, называющийся Kravitz-Reed и использующий неприводимые двоичные многочлены [898], небезопасен [451, 589]. Винфрид Мюллер (Winfried Müller) и Вилфрид Нобауер (Wilfried Nöbauer) используют полиномы Диксона (Dickson) [1127, 1128, 965]. Рудольф Лидл (Rudolph Lidl) и Мюллер обобщили этот подход в [966, 1126] (этот вариант назван схемой Réidi), и Нобауер проанализировал его безопасность в [1172, 1173]. (Соображения по поводу генерации простых чисел с помощью функций Лукаса (Lucas) можно найти в [969, 967, 968, 598].) Несмотря на все предыдущие разработки группе исследователей из Новой Зеландии удалось запатентовать эту схему в 1993 году, назвав ее LUC [1486, 521, 1487].

n-ое число Лукаса, $V_n(P,1)$, определяется как

$V_n(P,1) = PV_{n-1}(P,1) - V_{n-2}(P,1)$

Теория чисел Лукаса достаточно велика, и я ее пропущу. Теория последовательностей Лукаса хорошо изложена в [1307, 1308]. Особенно хорошо математика LUC описана в [1494, 708].

В любом случае для генерации пары открытый ключ/закрытый ключ сначала выбираются два больших числа p и q. Вычисляется n, произведение p и q. Ключ шифрования e - это случайное число, взаимно простое с p-1, q-1, p+1 и q+1. Существует четыре возможных ключа дешифрирования,

$d = e^{-1} \bmod (\text{НОК}(p+1), (q+1)))$

$d = e^{-1} \bmod (\text{НОК}(p+1), (q-1)))$

$d = e^{-1} \bmod (\text{НОК}(p-1), (q+1)))$

$d = e^{-1} \bmod (\text{НОК}(p-1), (q-1)))$

где НОК означает наименьшее общее кратное.

Открытым ключом являются d и n; закрытым ключом - e и n. p и q отбрасываются.

Для шифрования сообщения P (P должно быть меньше n) вычисляется

$C = V_e(P,1) \pmod{n}$

А для дешифрирования:

$P = V_d(P, 1) \pmod{n}$, с соответствующим d

В лучшем случае LUC не безопаснее RSA. А недавние, только что опубликованные результаты показывают, как взломать LUC по крайней мере в нескольких реализациях. Я не доверяю этому алгоритму.

19.10 Криптосистемы с открытым ключом на базе конечных автоматов

Китайский криптограф Тао Ренжи разработал алгоритм с открытым ключом, основанный на использовании конечных автоматов [1301, 1302, 1303, 1300, 1304, 666]. Такой же сложной задачей, как и разложение на множители произведения двух больших простых чисел, является задача разложения на составляющие произведения двух конечных автоматов. Это тем более верно, если один из автоматов нелинеен.

Большая часть работы в этой области была выполнена в Китае в 80-х годах и опубликована на китайском языке. Ренжи начал писать по английски. Его главным результатом было то, что обратное значение некоторых нелинейных (квазилинейных) автоматов является слабым тогда и только тогда, когда эти автоматы обладают определенной ступенчатой матричной структурой. Это свойство исчезает, если они объединены с другим автоматом (хотя бы линейным). В алгоритме с открытым ключом секретный ключ является инвертируемым квазилинейным автоматом, а соответствующий открытый ключ может быть получен с помощью их почленного перемножения. Данные шифруются, проходя через линейный автомат, а дешифрируются, проходя через обратные значения компонентов алгоритма (в некоторых случаях автоматы должны быть установлены в подходящее начальное значение). Эта схема работает и для шифрования, и для цифровых подписей.

О производительности таких систем вкратце можно сказать следующее: они, как и система McEliece, намного быстрее RSA, но требуют использования более длинных ключей. Длина ключа, обеспечивающая, как думают, безопасность, аналогичную 512-битовому RSA, равна 2792 битам, а 1024-битовому RSA - 4152 битам. В первом случае система шифрует данные со скоростью 20869 байт/с и дешифрирует данные со скоростью17117 байт/с, работая на 80486/33 МГц.

Ренжи опубликовал три алгоритма. Первым был FAPKC0. Эта слабая система использует линейные компоненты и, главным образом, является иллюстративной. Каждая из двух серьезных систем, FAPKC1 и FAPKC2, использует один линейный и один нелинейный компонент. Последняя сложнее, она была разработана для поддержки операции проверки подлинности.

Что касается их надежности, в Китае немало занимались этой проблемой (где сейчас свыше 30 институтов, публикующих работы по криптографии и безопасности). Из достаточного количества источников на китайском языке можно видеть, что проблема была изучена.

Привлекательной особенностью FAPKC1 и FAPKC2 является то, что они не ограждены никакими патентами США. Следовательно, так как срок действия патента на алгоритм Diffie-Hellman истекает в 1997 году, эти алгоритмы несомненно являются очень интересными.

Глава 20
Алгоритмы цифровой подписи с открытым ключом

20.1 Алгоритм цифровой подписи (DIGITAL SIGNATURE ALGORITHM, DSA)

В августе 19991 года Национальный институт стандартов и техники (National Institute of Standards and Technology, NIST) предложил для использования в своем Стандарте цифровой подписи (Digital Signature Standard, DSS) Алгоритм цифровой подписи (Digital Signature Algorithm, DSA). Согласно *Federal Register* [538]:

> Предлагается Федеральный стандарт обработки информации (Federal Information Processing Standard, FIPS) для Стандарта цифровой подписи (Digital Signature Standard, DSS). В этом стандарте определяется алгоритм цифровой подписи с открытым ключом (DSA), пригодный для федеральных применений, требующих цифровой подписи. Предложенный DSS использует открытый ключ для проверки получателем целостности полученных данных и личности отправителя. DSS также может быть использован третьей стороной для проверки правильности подписи и связанных с ней данных.
>
> В этом стандарте принимается схема подписи с открытым ключом, использующая пару преобразований для создания и проверки цифрового значения, называемого подписью.

И:

> Предложенный стандарт представляет собой результат оценки различных методик цифровой подписи. Принимая решение, NIST следовал положению раздела 2 Акта о компьютерной безопасности (Computer Security Act) 1987 года о том, что NIST разрабатывает стандарты," . . . обеспечивающие рентабельные безопасность и секретность Федеральной информации, выбирая из технологий, предлагающих сравнимую степень защиты, ту, которая обладает наиболее подходящими рабочими и эксплуатационными характеристиками".
>
> Среди факторов, рассмотренных в процессе принятия решения были уровень обеспечиваемой безопасности, простота аппаратной и программной реализации, простота экспорта за пределы США, применимость патентов, влияние на национальную безопасность и обеспечение правопорядка, а также степень эффективностикак функции подписи, так и функции проверки. Казалось, что обеспечить соответствующую защиту Федеральным системам можно многими способами. Выбранный удовлетворяет следующим требованиям:
>
> NIST ожидает, что его можно будет использовать бесплатно. Широкое использование этой технологии, обусловленной его доступностью, послужит к экономической выгоде правительства и общества.
>
> Выбранная технология обеспечивает эффективное использование операций подписи в приложениях, связанных с использованием интеллектуальных карточек. В этих приложениях операции подписи выполняются в слабой вычислительной среде интеллектуальных карточек, а процесс проверки реализуется в более мощной вычислительной среде, например, на персональном компьютере, в аппаратном криптографическом модуле или на компьютере-мэйнфрейме.

Прежде, чем все совсем запутается, позвольте мне разобраться с названиями: DSA - это алгоритм, а DSS стандарт. Стандарт использует алгоритм. Алгоритм является частью стандарта.

Реакция на заявление

Заявление NIST вызвало поток критических замечаний и обвинений. К сожалению, они были скорее политическими, чем научными. RSA Data Security, Inc., продающая алгоритм RSA, возглавила критиков DSS. Они требовали, чтобы в стандарт использовался алгоритм RSA. RSADSI получило немало денег за лицензирование алгоритма RSA, и стандарт бесплатной цифровой подписи прямо повлиял бы на самую суть ее коммерческих успехов. (Примечание: DSA необязательно не нарушает патенты, мы рассмотрим эту тему позднее.)

До заявления о принятии алгоритма RSADSI вело компанию против "общего модуля," который, возможно, позволит правительству подделывать подписи. Когда было объявлено, что алгоритм не использует общий модуль, критика была продолжена с других позиций [154], как с помощью писем в NIST, так и с помощью заявлений в прессе. (Четыре письма в NIST появилось в [1326]. Читая их, не забывайте, что по крайней мере два автора, Ривест и Хеллман, были финансово заинтересованы в том, чтобы DSS не был принят.)

Многие большие компании, разрабатывающие программное обеспечение, которые уже лицензировали алгоритм RSA, также выступили против DSS. В 1982 году правительство попросило предоставить ему алгоритмы с открытым ключом для выбора одного из них в качестве стандарта [537]. После этого в течение девяти лет от NIST не было никаких известий. Такие компании, как IBM, Apple, Novell, Lotus, Northern Telecom, Microsoft, DEC и Sun потратили много денег, реализуя алгоритм RSA. Они не были заинтересованы в потере инвестиций.

Всего к концу первого периода обсуждения(28 февраля 1992 года) NIST получил 109 замечаний. Рассмотрим по порядку критические замечания в адрес DSA.

1. DSA нельзя использовать для шифрования или распределения ключей.

Правильно, но стандарт и не требует наличия этих возможностей. Это стандарт подписи. NIST подготовить стандарт шифрования с открытым ключом. NIST совершает большую ошибку, оставляя американский народ без стандарта шифрования с открытым ключом. По всей вероятности предложенный стандарт цифровой подписи будет невозможно использовать для шифрования. (Но оказывается, что возможно - см. раздел 23.3.) Это не означает, что стандарт подписи бесполезен.

2. DSA был разработан NSA, и в алгоритме могут быть специальные лазейки.

Большинство первоначальных комментариев были просто параноидальными : "Отрицание NIST существующих алгоритмов без видимых причин не внушает доверия к DSS, а усиливает подозрение, что существует тайная программа, стремящаяся позволить NIST и/или NSA вскрывать национальную криптосистему с открытым ключом" [154]. Серьезный вопрос относительно безопасности DSA был задан Аржаном Ленстрой (Arjen Lenstra) и Стюартом Хабером (Stuart Haber) из Bellcore. Он будет рассмотрен ниже.

3. DSA медленнее RSA [800].

Более или менее справедливо. Скорости генерации подписи примерно одинаковы, но проверка подписи с помощью DSA от 10 до 40 раз медленнее. Однако генерация ключей быстрее. Но эта операция неинтересна, пользователь редко применяет ее. С другой стороны проверка подписи - это наиболее частая операция.

Проблема критики в том, что существует много способов поиграть параметрами тестирования, добиваясь нужных результатов. Предварительные вычисления могут ускорить генерацию подписи DSA, но они не всегда возможны. Сторонники RSA подбирают числа так, чтобы выделить преимущества своего алгоритма, а сторонники DSA используют свой способ оптимизации. В любом случае компьютеры становятся все быстрее и быстрее. Хотя разница в скорости и существует, в большинстве приложений она не будет заметна.

4. RSA - это стандарт *de facto*.

Вот два примера подобных жалоб. Письмо Роберта Фоллета (Robert Follett), директора программы стандартизации компании IBM [570]:

> IBM считает, что NIST предложил стандарт схемы цифровой подписи, отличающийся от принимаемых международных стандартов. Пользователи и организации пользователей убедили нас в том, что поддержка международных стандартов, и использующих RSA, в самом ближайшем будущем станет необходимым условием продажи средств обеспечения безопасности.

Письмо Леса Шроера (Les Shroyer), вице-президента и директора компании Motorola [1444]:

> У нас должен быть единый, надежный, признанный всеми алгоритм цифровой подписи, который можно использовать по всему миру как между американскими и неамериканскими объектами, так и между системами компании Motorola и системами других производителей. Отсутствие других жизнеспособных технологий цифровой подписи за последние восемь лет сделало RSA фактическим стандартом. . . . Motorola и многие другие компании. . . вложили в RSA миллионы долларов. Мы сомневаемся во взаимодействии и возможности поддержки двух различных стандартов , такое положение приведет к росту расходов, задержек развертывания и усложнению систем. . . .

Многим компаниям хотелось, чтобы NIST принял ISO 9796, международный стандарт цифровой подписи, использующий RSA [762.]. Хотя это и серьезный аргумент, он недостаточен, чтобы принять международный стандарт в качестве национального. Бесплатный стандарт лучше отвечал бы общественным интересам Соединенных Штатов.

5. Выбор национального алгоритма не был открытым, не было дано достаточно времени для анализа .

Сначала NIST утверждал, что разработал DSA самостоятельно, затем признал помощь NSA. Наконец NIST подтвердил, что NSA является автором алгоритма. Это многих обеспокоило - NSA не внушает людям доверие. Даже так, алгоритм был опубликован и доступен для анализа, кроме того, NIST продлил время анализа и комментирования алгоритма.

6. DSA может нарушать другие патенты. Это так. Этот вопрос будет рассмотрен в разделе, рассматривающим патенты.

7. Размер ключа слишком мал.

Это единственно справедливая критика DSS. Первоначально предлагалось использовать модуль длиной 512 битов [1149]. Так как безопасность алгоритма определяется сложностью вычисления дискретных логарифмов по заданному модулю, этот вопрос волновал многих криптографов. С тех пор вычисление дискретных логарифмов в конечном поле достигло определенных успехов, и 512 битов слишком мало для долговременной подписи (см. раздел 7.2). Согласно Брайану ЛаМаччиа (Brian LaMacchia) и Эндрю Одлыжко (Andrew Odlyzko), " . . . даже безопасность, обеспечиваемая 512-битовыми простыми числами, по видимому, находится на пределе . . . " [934]. В ответ на эти замечания NIST сделал длину ключа переменной, от 512 до 1024 битов. Немного, но все-таки получше.

19 мая 1994 года был издан окончательный вариант стандарта [1154]. При этом было сказано [542]:

> Этот стандарт может применяться всеми Федеральными департаментами и управлениями для защиты несекретной информации. . . . Этот стандарт будет использован при проектировании и реализации схем подписи с открытыми ключами, которые разрабатывают Федеральные департаменты и управления, или которые разрабатываются по из заказу . Частные и коммерческие организации могут принять и использовать этот стандарт.

Прежде чем пользоваться этим стандартом и реализовывать его, прочтите ниже раздел о патентах.

Описание DSA

DSA, представляющий собой вариант алгоритмов подписи Schnorr и EIGamal, полностью описан в [1154].

Алгоритм использует следующие параметры:

p = простое число длиной L битов, где L принимает значение, кратное 64, в диапазоне от 512 до 1024. (В первоначальном стандарте размер p был фиксирован и равен 512 битам [1149]. Это вызвало множество критических замечаний, и NIST этот пункт алгоритма [1154].)

q = 160-битовой простое число - множитель p-1.

$g = h^{(p-1)/q} \bmod p$, где h - любое число, меньшее p-1, для которого $h^{(p-1)/q} \bmod p$ больше 1.

x = число, меньшее q.

$y = g^x \bmod p$.

В алгоритме также используется однонаправленная хэш-функция: $H(m)$. Стандарт определяет использование SHA, рассмотренного в разделе 18.7.

Первые три параметра, p, q и g, открыты и могут быть общими для пользователей сети. Закрытым ключом является x, а открытым - y. Чтобы подписать сообщение, m:

(1) Алиса генерирует случайное число k, меньшее q

(2) Алиса генерирует

$r = (g^k \bmod p) \bmod q$

$s = (k^{-1} (H(m) + xr)) \bmod q$

Ее подписью служат параметры r и s, она посылает их Бобу.

(3) Боб проверяет подпись, вычисляя

$w = s^{-1} \bmod q$

$u_1 = (H(m) * w) \bmod q$

$u_2 = (rw) \bmod q$

$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$

Если $v = r$, то подпись правильна.

Доказательства математических соотношений можно найти в [1154]. 19th представляет собой краткое описание алгоритма.

Табл. 20-1.
Подписи DSA

	Открытый ключ:
p	простое число длиной от 512 до 1024 битов (может использоваться группой пользователей)
q	160-битовый простой множитель p-1 (может использоваться группой пользователей)
g	$= h^{(p-1)/q} \bmod p$, где h - любое число, меньшее p-1, для которого $h^{(p-1)/q} \bmod p > 1$ (может использоваться группой пользователей)
y	$= g^x \bmod p$ (p-битовое число)
	Закрытый ключ:
x	$< q$ (160-битовое число)
	Подпись:
k	выбирается случайно, меньшее q
r	(подпись) $= (g^k \bmod p) \bmod q$
s	(подпись) $= (k^{-1} (H(m) + xr)) \bmod q$
	Проверка:
w	$= s^{-1} \bmod q$
u_1	$= (H(m) * w) \bmod q$
u_2	$= (rw) \bmod q$

$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$

Если $v = r$, то подпись правильна.

Ускоряющие предварительные вычисления

В 18-й приведены примеры скорости работы программных реализаций DSA [918].

Табл. 20-2.
Скорость DSA для различных длин модулей с 160-битовым показателем степени (на SPARC II)

	512 битов	768 битов	1024 бита
Подпись	0.20 с	0.43 с	0.57 с
Проверка	0.35 с	0.80 с	1.27 с

Практические реализации DSA часто можно ускорить с помощью предварительных вычислений. Обратите внимание, что значение r не зависит от сообщения. Можно создать строку случайных значений k, и затем рассчитать значения r для каждого из них. Можно также вычислить k^{-1} для каждого из этих значений k. Затем, когда приходит сообщение, можно вычислить s для заданных r и k^{-1}.

Эти предварительные вычисления заметно ускоряют DSA. В 17-й приведены сравнения времени вычисления DSA и RSA для конкретной реализации интеллектуальной карточки [1479].

Табл. 20-3.
Сравнение времени вычислений RSA и DSA

	DSA	RSA	DSA с общими p, q, g
Глобальные вычисления	Off-card (P)	N/A	Off-card (P)
Генерация ключа	14 с	Off-card (S)	4с
Предварительные вычисления	14 с	N/A	4 с
Подпись	0.03 с	15 с	0.03 с
Проверка	16 с	1.5 с	10 с
	1-5 с off-card (P)	1-3 с off-card (P)	

Вычисления вне карточки (off-card) выполнялись на персональном компьютере i80386/33 МГц. (P) указывает открытые параметры off-card, а (S) - на закрытые параметры off-card. В обоих алгоритмах используется 512-битовый модуль.

Генерация простых чисел DSA

Ленстра и Хабер указали, что взломать некоторые модули намного легче, чем другие [950]. Если кто-нибудь заставит пользователей сети использовать один из таких слабых модулей, то их подписи будет легче подделать. Тем не менее это не представляет проблемы по двум причинам: такие модули легко обнаружить, и они так редки, что вероятность случайно использовать одного из них пренебрежимо мала, меньше, чем вероятность случайно получить составное число на выходе вероятностной процедуры генерации простых чисел.

В [1154] NIST рекомендовал конкретный метод генерации двух простых чисел, p и q, где q является делителем p-1. Длина простого числа p - между 512 и 1024 и кратна 64 -битам. Пусть L-1= 160n+b, где L - это длина p, а n и b - два числа, причем b меньше 160.

(1) Выберем произвольную последовательность, по крайней мере, 160 битов и назовем ее S. Пусть g - это длина S в битах.

(2) Вычислим $U = \mathrm{SHA}(S) \oplus \mathrm{SHA}((S + 1) \bmod 2^g)$, где SHA описан в разделе 18.7.

(3) Образуем q, установив наибольший и наименьший значащие биты U в 1.

(4) Проверим, является ли q простым.

(5) Если q не является простым, то вернемся на этап (1).

(6) Пусть C=0 и N=2.

(7) Для k=0,l,...,n, пусть V_k=SHA((S+N+k) mod 2^g)

(8) Пусть W - целое число

$W = V_0 + 2^{160}V_1 + \ldots + 2^{160(n-1)}\ V_{n-1} + 2^{160}\,(V_n \bmod 2^b)$

и пусть

$X = W + 2^{L-1}$

Обратите внимание, что X - это L-битовое число.

(9) Пусть $p = X - ((X \bmod 2q) - 1)$. Обратите внимание, что p конгруэнтно 1 mod $2q$.

(10) Если $p < 2^{L-1}$, то перейдем на этап (13).

(11) Проверим, является ли p простым числом.

(12) Если p - простое, перейдем к этапу (15).

(13) Пусть C=C+1 и N=N+n+l.

(14) Если C = 4096, вернемся к этапу (1). В противном случае перейдем на этап (7).

(15) Сохраним значения S и C, использованные для генерации p и q.

В [1154] переменная S называется стартовой, переменная C - счетчиком, а N - смещением.

Смысл этого упражнения в том, что оно является опубликованным способом генерации p и q. Для всех практических применений этот метод позволяет избежать слабых значений p и q. Если кто-то вручит вам какие-то p и q, вас может заинтересовать, как получены эти числа. Однако, если вы получите значения S и C, использованные при генерации случайных p и q, вы сможете повторить всю процедуру самостоятельно. Использование однонаправленной хэш-функции (в стандарте используется SHA) не позволяет получить S и C по значениям p и q.

Эта безопасность лучше, чем обеспечиваемая RSA. В RSA простые числа хранятся в секрете. Любой может генерировать фальшивое простое число или число, форма которого упрощает разложение на множители. Не зная закрытого ключа, это никогда не проверишь. В DSA, даже если закрытый ключ неизвестен, можно убедиться, что p и q генерировались случайным образом.

Шифрование ElGamal с DSA

Утверждалось, что DSA так нравится правительству, потому что его нельзя использовать в качестве алгоритма шифрования. Однако можно использовать вызов функции DSA для шифрования EIGamal. Пусть алгоритм реализован как вызов одной функции

```
DSAsign(p,q,g,k,x,h,r,s)
```

Задав входные значения p, q, g, k, x и h, можно получить параметры подписи: r и s.

Для шифрования сообщения m алгоритмом EIGamal с помощью открытого ключа y выберем случайное число k и вызовем

```
DSAsign(p,p,g,k,0,0,r,s)
```

Возвращенное значение r и будет a из схемы EIGamal. Отбросим s. Затем вызовем, call

```
DSAsign(p,p,y,k,0,0,r,s)
```

Переименуем значение r в u, отбросим s. Вызовем

```
DSAsign(p,p,m,1,u,0,r,s)
```

Отбросим r. Возвращенное значение s и будет b в схеме EIGamal. Теперь у вас есть шифротекст, a и b. Дешифрирование также просто. Используя закрытый ключ x и шифротекст сообщений, a и b, вызовем

```
DSAsign(p,p,a,x,0,0,r,s)
```

Значение r - это a^x mod p. Назовем его e. Затем вызовем

```
DSAsign(p,p,1,e,b,0,r,s)
```

Значение s и будет открытым текстом сообщения, m.

Этот способ работает не со всеми реализациями DSA - в некоторых из них могут быть зафиксированы значения p и q или длины некоторых других параметров. Тем не менее, если реализация является достаточно общей, то можно шифровать сообщение, не используя ничего, кроме функции цифровой подписи.

Шифрование RSA с DSA

Шифрование RSA еще проще. Используя модуль n, сообщение m и открытый ключ e, вызовем

```
DSAsign(n,n,m,e,0,0,r,s)
```

Возвращенное значение r и есть шифротекст. Дешифрирование RSA является точно таким же. Если d - закрытый ключ, то

```
DSAsign(n,n,m,d,0,0,r,s)
```

возвращает открытый текст как значение r.

Безопасность DSA

С 512 битами DSA недостаточно надежен для длительной безопасности, но он вполне надежен при 1024 битах. В своем первом заявлении на эту тему NSA так комментировало утверждение Джо Эбернети (Joe Abernathy) из *The Houston Chronicle* по поводу лазейки в DSS [363]:

> Что касается предполагаемой лазейки в DSS. Мы считаем, что термин "лазейка" вводит в заблуждение, так он предполагает, что через лазейку можно как-то расшифровать (прочитать) зашифрованные сообщения, подписываемые с помощью DSS, без разрешения отправителя.
>
> DSS не шифрует никаких данных. По сути вопросом является, не может ли кто-то при помощи DSS подделать подпись, и, таким образом, дискредитировать всю систему. Мы категорически заявляем, что вероятность, что кто-нибудь - включая NSA - сможет подделать подпись DSS, при правильном использовании стандарта бесконечно мала.
>
> Более того, предположение о чувствительности к лазейке справедливо для *любой* системы проверки подлинности с открытыми ключами, включая RSA. Утверждение, что это влияет только на DSS (аргумент, популярный в прессе), полностью неверно. Вопрос в реализации и способе выбора простых чисел. Мы призываем вас уделить внимание недавней конференции EUROCRYPT, где "за круглым столом" обсуждался вопрос лазеек в DSS. Одним из участников обсуждения был один из исследователей из Bellcore, утверждавший о возможности лазейки, и по нашему пониманию участники дискуссии - включая этого исследователя из Bellcore - пришли к выводу, что вопрос о лазейке в DSS не представляет проблемы. Более того, всеми было признано, что вопрос о лазейке является тривиальным и был раздут прессой. Однако, пытаясь по просьбе NIST ответить на обвинение о лазейке, мы разработали процесс генерации простых чисел, позволяющий избежать выбора одного из относительно небольшого числа слабых простых чисел, использование которых ослабляет DSS. Кроме того, NIST настаивает на использовании модулей большей длины, вплоть до 1024, что позволяет не пользоваться разработанным процессом генерации простых чисел, избегая слабых простых чисел. Очень важным дополнительным моментом, на который часто не обращают внимание, является то, что при использовании DSS простые числа *общедоступны* и, следовательно, могут быть предметом открытого изучения. Не все системы с открытыми ключами способны пройти подобную проверку.
>
> Целостность любой системы защиты информации требует обратить внимание на реализацию. Учитывая уязвимость систем с миллионами равноправных пользователей, NSA по традиции настаивает на использовании централизованных доверенных центров как на способе минимизировать риск в системе. Хотя мы по просьбе NIST и разработали ряд технических модификаций DSS, позволяющих реализовать менее централизованный подход, все же нужно выделить ту часть объявления о DSS в *Federal Register*, в которой говорится:
>
> "Хотя этот стандарт должен обеспечить общие требования безопасности генерации цифровых подписей, соответствие стандарту не обеспечивает безопасность конкретной реализации. Ответственное лицо в каждом департаменте или управлении должно гарантировать, что общая реализация гарантирует приемлемый уровень безопасности. NIST продолжит работу с правительственными пользователями, обеспечивая правильность реализаций."
>
> Наконец мы изучили все утверждения о небезопасности DSS, и они нас не убедили. DSS был тщательно изучен в NSA, что позволило нашему Директору по безопасности информационных систем разрешить использовать этот стандарт для подписи несекретных данных, обрабатываемых в определенных разведывательных системах, и даже для подписи секретных данных в ряде систем. Мы считаем, что подобное признание свидетельствует о невозможности какого-либо вероятного вскрытия безопасности, обеспечиваемой DSS при его правильных реализации и использовании. Основываясь на требованиях правительства США к технике и безопасности цифровых подписей, мы считаем, что DSS является лучшим выбором. В действительности, DSS выступает в качестве пилотного проекта Системы защиты сообщений (Defense Message System), призванного гарантировать подлинность электронных сообщений для жизненно важных команд и контрольной информации. Эта начальная демонстрация включает участие Комитета начальников штабов, военных служб и оборонных ведомств и проводится в кооперации с NIST.

Я не собираюсь комментировать истинность заявления NSA. Принимать его на вру или нет - зависит от вашего к нему отношения.

Вскрытия k

Для каждой подписи нужно новое значение k, которое должно выбираться случайным образом. Если Ева узнает k, которое Алиса использовала для подписи сообщения, может быть воспользовавшись некоторыми свойствами генератора случайных чисел, который выдает k, она сможет раскрыть закрытый ключ Алисы, x. Если Ева добудет два сообщения, подписанных с помощью одного и того же k, то, даже не зная значение k, она сможет раскрыть x. А с помощью x Ева сможет тайно подделывать подпись Алисы. В любой реализации DSA для безопасности системы очень важен хороший генератор случайных чисел [1468].

Опасности общего модуля

Хотя DSS не определяет применение пользователями общего модуля, различные реализации могут воспользоваться такой возможностью. Например, Налоговое управление рассматривает использование DSS для электронной налогов. Что если эта организация потребует, чтобы все налогоплательщики страны использовали общие p и q? Общий модуль слишком легко становится мишенью для криптоанализа. Пока слишком рано обсуждать различные реализации DSS, но причины для беспокойства есть.

Подсознательный канал в DSA

Гус Симмонс (Gus Simmons) открыл в DSA подсознательный канал [1468, 1469] (см. раздел 23.3). Этот подсознательный канал позволяет встраивать в подпись тайное сообщение, которое может быть прочитано только тем, у кого есть ключ. Согласно Симмонсу, это "замечательное совпадение", что "все очевидные недостатки подсознательных каналов, использующих схему ElGamal, могут быть устранены" в DSS, и что DSS "на сегодня обеспечивает наиболее подходящую среду для подсознательных коммуникаций". NIST и NSA не комментировали этот подсознательный канал, никто даже не знает, догадывались ли они о такой возможности. Так как этот подсознательный канал позволяет при недобросовестной реализации DSS передавать с каждой подписью часть закрытого ключа. Никогда на пользуйтесь реализацией DSS, если вы не доверяете разработчику реализации.

Патенты

Дэвид Кравиц (David Kravitz), ранее работавший в NSA, владеет патентом DSA [897]. Согласно NIST [538]:

> NIST в интересах общества стремится сделать технологию DSS доступной бесплатно по всему миру. Мы считаем, что эта технология может быть запатентована, и что никакие другие патенты не применимы к DSS, но мы не можем дать твердых гарантий этого до получения патента.

Несмотря на это, три владельца патентов утверждают, что DSA нарушает их патенты: Diffie-Hellman (см. раздел 2.2.1) [718], Merkle-Hellman (см. раздел 19.2.) [720] и Schnorr (см. раздел 21.3) [1398]. Патент Schnorr является источником наибольших сложностей. Срок действия двух других патентов истекает 1997 году, а патент Schnorr действителен до 2008 года. Алгоритм Schnorr был разработан не на правительственные деньги. В отличие от патентов PKP у правительства США нет прав на патент Schnorr, и Шнорр запатентовал свой алгоритм по всему миру. Даже если суды США вынесут решение в пользу DSA, неясно, какое решение примут суды в других странах. Сможет ли международная корпорация принять стандарт, который законен в одних странах и нарушает патентное законодательство в других? Нужно время, чтобы решить эту проблему, к моменту написания этой книги этот вопрос не решен даже в Соединенных Штатах.

В июне 1993 года NIST предложил выдать PKP исключительную патентную лицензию на DSA [541]. Соглашение провалилось после протестов общественности и стандарт вышел в свет без всяких соглашений. NIST заявил [542]:

> • . . NIST рассмотрел заявления о возможном нарушении патентов и сделал вывод, что эти заявления несправедливы.

Итак стандарт официально принят, в воздухе пахнет судебными процессами, и никто не знает, что делать. NIST заявил, что он поможет защититься людям, обвиненным в нарушении патентного законодательства при использовании DSA в работе по правительственному контракту. Остальные, по видимому, должны заботиться о себе сами. Проект банковского стандарта, использующего DSA, выдвинут ANSI [60]. NIST работает над введением стандарта DSA в правительственном аппарате. Shell Oil сделала DSA своим международным стандартом. О других предложенных стандартах DSA мне неизвестно.

20.2 Варианты DSA

Этот вариант делает проще вычисления, необходимые для подписи, не заставляя вычислять k^{-1} [1135]. Все используемые параметры - такие же, как в DSA. Для подписи сообщения m Алиса генерирует два случайных числа, k и d, меньшие q. Процедура подписи выглядит так

$r = (g^k \bmod p) \bmod q$

$s = (H(m) + xr) * d \bmod q$

$t = kd \bmod q$

Боб проверяет подпись, вычисляя

$w = t/s \bmod q$

$u_1 = (H(m) * w) \bmod q$

$u_2 = (rw) \bmod q$

Если $r = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$, то подпись правильна.

Следующий вариант упрощает вычисления при проверке подписи [1040, 1629]. Все используемые параметры - такие же, как в DSA. Для подписи сообщения m Алиса генерирует случайное число k, меньшее q. Процедура подписи выглядит так

$r = (g^k \bmod p) \bmod q$

$s = k\,(H(m) + xr)^{-1} \bmod q$

Боб проверяет подпись, вычисляя

$u_1 = (H(m) * s) \bmod q$

$u_2 = (sr) \bmod q$

Если $r = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$, то подпись правильна.

Еще один вариант DSA разрешает пакетную проверку, Боб может проверять подписи пакетами [1135]. Если все подписи правильны, то работа Боба закончена. Если одна из них неправильна, то ему еще нужно понять, какая. К несчастью, это небезопасно. Либо подписывающий, либо проверяющий может легко создать набор фальшивых подписей, который удовлетворяет критерию проверки пакета подписей [974].

Существует также вариант генерации простых чисел для DSA, который включает q и используемые для генерации простых чисел параметры внутрь p. Влияет ли эта схема на безопасность DSA, все еще неизвестно.

(1) Выберем произвольную последовательность, по крайней мере, 160 битов и назовем ее S. Пусть g - это длина S в битах.

(2) Вычислим $U = \text{SHA}(S) \oplus \text{SHA}((S + 1) \bmod 2^g)$, где SHA описан в разделе 18.7.

(3) Образуем q, установив наибольший и наименьший значащие биты U в 1.

(4) Проверим, является ли q простым.

(5) Пусть p - это объединение q, S, C и SHA(S). C представляет собой 32 нулевых бита.

(6) $p=p-(p \bmod q)+1$.

(7) $p=p+q$.

(8) Если C в p равно 0x7fffffff, вернемся на этап (1).

(9) Проверим, является ли p простым.

(10) Если p - составное, вернемся на этап (7).

Преимуществом этого варианта является то, что вам не нужно хранить значения C и S, использованные для генерации p и q. Они включены в состав p. Для приложений, работающих в условиях нехватки памяти, например, для интеллектуальных карточек, это может быть важно.

20.3 Алгоритм цифровой подписи ГОСТ

Это русский стандарт цифровой подписи, Официально называемый ГОСТ Р 34.10-94 [656]. Алгоритм очень похож на DSA, и использует следующие параметры

p = простое число, длина которого либо между 509 и 512 битами, либо между 1020 и 1024 битами.

q = простое число - множитель p-1, длиной от 254 до 256 битов.

a = любое число, меньшее p-1, для которого $a^q \bmod p = 1$.

x = число, меньшее q.

$y = a^x \bmod p$.

Этот алгоритм также использует однонаправленную хэш-функцию : $H(x)$. Стандарт определяет использование хэш-функции ГОСТ Р 34.1 1-94 (см. раздел 18.1 1), основанной на симметричном алгоритме ГОСТ (см. раздел 14.1) [657].

Первые три параметра, p, q и a, открыты и могут использоваться совместно пользователями сети. Закрытым ключом служит x, а открытым - y. Чтобы подписать сообщение m

(1) Алиса генерирует случайное число k, меньшее q

(2) Алиса генерирует $I = (\text{a}* \bmod p) \bmod q$ s = (ct + $k(H(m)))$ mod q

$r = (a^k \bmod p) \bmod q$

$s = (xr + k(H(m))) \bmod q$

Если $H(m) \bmod q = 0$, то значение хэш-функции устанавливается равным 1. Если $r = 0$, то выберите другое значение k и начните снова. Подписью служат два числа: $r \bmod 2^{256}$ и $s \bmod 2^{256}$, Алиса посылает их Бобу.

(3) Боб проверяет подпись, вычисляя

$v = H(m)^{q-2} \bmod q$

$z_1 = (sv) \bmod q$

$z_2 = ((q-r)*v) \bmod q$

$u = ((a^{z_1} * y^{z_2}) \bmod p) \bmod q$

Если $u = r$, то подпись правильна.

Различие между этой схемой и DSA в том, что в DSA $s = (k^{-1} (H(m) + xr)) \bmod q$, что дает другое уравнение проверки. Любопытно, однако, что длина q равна 256 битам. Большинству западных криптографов кажется достаточным q примерно 160 битов длиной. Может быть это просто следствие русской привычки играть в сверхбезопасность.

Стандарт используется с начала 1995 года и не закрыт грифом "Для служебного пользования", что бы это не значило.

20.4 Схемы цифровой подписи с использованием дискретных логарифмов

Схемы подписи ElGamal, Schnorr (см. раздел 21.3) и DSA очень похожи. По сути, все они являются тремя примерами общей схемы цифровой подписи, использующей проблему дискретных логарифмов . Вместе с тысячами других схем подписей они являются частью одного и того же семейства [740, 741, 699, 1184].

Выберем p, большое простое число, и q, равное либо p-1, либо большому простому множителю p-1. Затем выберем g, число между 1 и p, для которого $g^q \equiv 1 \pmod p$. Все эти числа открыты, и могут быть совместно использованы группой пользователей. Закрытым ключом является x, меньшее q. Открытым ключом служит $y = g^x \bmod q$.

Чтобы подписать сообщение m, сначала выберем случайное значение k, меньшее q и взаимно простое с ним. Если q тоже простое число, то будет работать любое k, меньшее q. Сначала вычислим

$r = g^k \bmod p$

Обобщенное **уравнение подписи** примет вид

$ak = b + cx \bmod q$

Коэффициенты a, b и c могут принимать различные значения. Каждая строка 16th предоставляет шесть возможностей. Проверяя подпись, получатель должен убедиться, что

$r^a = g^b y^c \bmod p$

Это уравнение называется **уравнением проверки**.

Табл. 20-4.
Возможные перестановки *a*, *b* и *c* ($r' = r \bmod q$)

$\pm r'$	$\pm s$	m
$\pm r' m$	$\pm s$	1
$\pm r' m$	$\pm ms$	1
$\pm m r'$	$\pm r' s$	1
$\pm ms$	$\pm r' s$	1

В 15th перечислены возможные варианты подписи и проверки, полученные только из первой строки возможных значений a, b и c без учета эффектов ±.

Табл. 20-5.
Схемы цифровой подписи с использованием дискретных логарифмов

Уравнение подписи	Уравнение проверки
(1) $r'k=s+mx \bmod q$	$r^{r'}=g^s y^m \bmod p$
(2) $r'k=m+sx \bmod q$	$r^{r'}=g^m y^s \bmod p$
(3) $sk=r'+mx \bmod q$	$r^s=g^{r'} y^m \bmod p$

(4) $sk = m + r'x \bmod q$ $\quad r^s = g^m y^{r'} \bmod p$

(5) $mk = s + r'x \bmod q$ $\quad r^m = g^s y^{r'} \bmod p$

(6) $mk = r' + sx \bmod q$ $\quad r^m = g^{r'} y^s \bmod p$

Это шесть различных схем цифровых подписей. Добавление минуса увеличивает их количество до 24. При использовании всех возможных значения a, b и c число схем доходит 120.

EIGamal [518, 519] и DSA [1154] по существу основаны на уравнении (4). Другие схемы - на уравнении (2) [24, 1629]. Schnorr [1396, 1397], как и другая схема [1183], тесно связан с уравнением (5). А уравнение (1) можно изменить так, чтобы получить схему, предложенную в [1630]. Оставшиеся уравнения - новые.

Далее. Любую из этих схем можно сделать более DSA-подобной, определив r как

$r = (g^k \bmod p) \bmod q$

Используйте то же уравнение подписи и сделайте уравнением проверки

$u_1 = a^{-1} b \bmod q$

$u_2 = a^{-1} c \bmod q$

$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$

$(r \bmod q)^a = g^b y^c \bmod p$

Существуют и две другие возможности подобных преобразований [740, 741]. Такие операции можно проделать с каждой из 120 схем, доведя общее число схем цифровой подписи, использующих дискретные логарифмы, до 480.

Но и это еще не все. Дополнительные обобщения и изменения приводят более, чем к 13000 вариантам (не все из них достаточно эффективны) [740, 741].

Одной из приятных сторон использования RSA для цифровой подписи является свойство, называемое **восстановлением сообщения**. Когда вы проверяете подпись RSA, вы вычисляете m. Затем вычисленное m сравнивается с сообщением и проверяется, правильна ли подпись сообщения. В предыдущих схемах восстановить m при вычислении подписи невозможно, вам потребуется вероятное m, которое и используется в уравнении проверки. Но, оказывается, можно построить вариант с восстановлением сообщения для всех вышеприведенных схем. Для подписи сначала вычислим

$r = mg^k \bmod p$

и заменим m единицей в уравнении подписи. Затем можно восстановит уравнение проверки так, чтобы m могло быть вычислено непосредственно. То же самое можно предпринять для DSA-подобных схем:

$r = (mg^k \bmod p) \bmod q$

Безопасность всех вариантов одинакова, поэтому имеет смысл выбирать схему по сложности вычисления. Большинство схем замедляет необходимость вычислять обратные значения. Как оказывается, одна из этих схем позволяет вычислять и уравнение подписи, и уравнение проверки без использования обратных значений, при этом еще и восстанавливая сообщение. Она называется схемой **p-NEW** [1184].

$r = mg^{-k} \bmod p$

$s = k - r'x \bmod q$

m восстанавливается (и проверяется подпись) с помощью вычисления

$m = g^s y^{r'} r \bmod p$

В ряде вариантов одновременно подписывается по два-три блока сообщения [740], другие варианты можно использовать для слепых подписей [741].

Это значительная область для изучения. Все различные схемы цифровой подписи с использованием дискретных логарифмов были объединены логическим каркасом. Лично я считаю, что это окончательно положит конец спорам между Schnorr [1398] и DSA [897]: DSA не является ни производной Schnorr, равно как и EIGamal. Все три алгоритма являются частными случаями описанной общей схемы, и эта общая схема незапатентована.

20.5 ONG-SCHNORR-SHAMIR

Эта схема подписи использует многочлены по модулю n [1219, 1220]. Выбирается большое целое число (знать разложение n на множители не обязательно). Затем выбирается случайное число k, взаимно простое с n, и вычисляется h, равное

$h = -k^{-2} \bmod n = -(k^{-1})^2 \bmod n$

Открытым ключом служат h и n; а закрытым - k.

Чтобы подписать сообщение M, сначала генерируется случайное число r, взаимно простое с n. Затем вычисляется:

$S_1 = 1/2\ (M/r + r) \bmod n$

$S_2 = 1/2\ (M/r - r) \bmod n$

Пара чисел S_1 и S_2 представляет собой подпись. Проверяя подпись, убеждаются, что

$S_1^2 + h*S_2^2 \equiv M \pmod n$

Описанный здесь вариант схемы основан на квадратичных многочленах. При его опубликовании в [1217] за успешный криптоанализ было предложено вознаграждение в $100. Небезопасность схемы была доказана [1255, 18], но это не остановило ее авторов. Они предложили модификацию алгоритма, основанную на кубических многочленах, также оказавшуюся небезопасной [1255]. Авторы предложили версию на базе многочленов четвертой степени, но была взломана и она [524, 1255]. Вариант, решающий эти проблемы, описан в [1134].

20.6 ESIGN

ESICN -это схема цифровой подписи, разработанная NTT Japan [1205, 583]. Утверждалось, что она не менее безопасна, чем RSA или DSA, и намного быстрее при тех же размерах ключа и подписи. Закрытым ключом служит пара больших простых чисел p и q. Открытым ключом является n, для которого

$n = p^2*q$

H - это хэш-функция, применяемая к сообщению m, причем значение $H(m)$ находится в пределах от 0 до n-1. Используется также параметр безопасности k, который будет вкратце рассмотрен.

(1) Алиса выбирает случайное число x, меньшее pq.

(2) Алиса вычисляет:

w, наименьшее целое, которое больше или равно

$(H(m) - x^k \bmod n)/pq$

$s = x + ((w/kx^{k-1} \bmod p)\ pq$

(3) Алиса посылает s Бобу.

(4) Для проверки подписи Боб вычисляет $s^k \bmod n$. Кроме этого, он вычисляет a, наименьшее целое, которое больше или равно удвоенному числу битов n, деленному на 3. Если $H(m)$ меньше или равна $s^k \bmod n$, и если $s^k \bmod n$ меньше $H(m)+2^a$, то подпись считается правильной.

Выполнив ряд предварительных вычислений, этот алгоритм можно ускорить. Эти вычисления могут быть выполнены в произвольный момент времени и никак не связаны с подписываемым сообщением. Выбрав x, Алиса может разбить этап (2) на два подэтапа. Сначала.

(2a) Алиса вычисляет:

$u = x^k \bmod n$

$v = 1/(kx^{k-1}) \bmod p$

(2b) Алиса вычисляет:

w= наименьшее целое, которое больше или равно

$(H(m) - u)/pq$

$s = x + ((wv \bmod p)\ pq$

Для обычно используемых размеров чисел предварительные вычисления ускоряют процесс подписи на порядок. Почти вся трудная работа выполняется именно на стадии предварительных вычислений. Обсуждение действий модульной арифметики, выполняемых при ускорении ESIGN, можно найти в [1625, 1624]. Этот алго-

ритм можно расширить для работы с эллиптическими кривыми [1206].

Безопасность ESIGN

Когда этот алгоритм был впервые предложен, k было выбрано равным 2 [1215]. Такая схема быстро была взломана Эрни Брикеллом (Ernie Brickell) и Джоном ДеЛаурентисом [261], которые распространили свое вскрытие и на случай $k = 3$. Модифицированная версия этого алгоритма [1203] была взломана Шамиром [1204]. Вариант, предложенный в [1204], был взломан в [1553]. ESIGN - это сегодняшняя реинкарнация алгоритмов из этого семейства. Попытка вскрыть ESIGN [963] оказалась безрезультатной.

В настоящее время авторы рекомендуют использовать следующие значения k: 8, 16, 32, 64, 128, 256, 512 и 1024. Они также рекомендуют, чтобы p и q были не меньше 192 битов каждое, образуя n не менее, чем 576 битов в длину. (Я думаю, что n должно быть еще в два раза больше.) Авторы считают, что с такими значениями параметров, безопасность ESIGN равна безопасности RSA или Rabin. И выполненный ими анализ показывает, что скорость ESIGN намного выше, чем у RSA, EIGamal и DSA [582].

Патенты

ESICN запатентован в Соединенных Штатах [1208], Канаде, Англии, Франции, Германии и Италии. Любой, кто хочет получить лицензию на алгоритм, должен обратиться в Отдел интеллектуальной собственности NTT (Intellectual Property Department, NTT, 1-6Uchisaiwai-cho, 1-chome, Chiyada-ku, 100 Japan).

20.7 Клеточные автоматы

Совершенно новая идея, изученная Папуа Гуамом (Papua Guam) [665], состоит в использовании в криптосистемах с открытыми ключами клеточных автоматов. Эта система все еще слишком нова и не прошла через тщательное изучение, но предварительное исследование показало, что у нее может быть такое же криптографически слабое место, как и у других систем [562]. Тем не менее, это многообещающая область исследований. Свойством клеточных автоматов является то, что даже если они инвертируемы, невозможно вычислить предка произвольного состояния, инвертировав правило нахождения потомка. Это выглядит очень похоже на однонаправленную хэш-функцию с лазейкой.

20.8 Другие алгоритмы с открытым ключом

За эти годы было предложено и вскрыто множество других алгоритмов с открытыми ключами. Алгоритм Matsumoto-Imai [1021] был вскрыт в [450]. Алгоритм Cade был впервые предложен в 1985 году, взломан в 1986 [774], и затем доработан в том же году [286]. Помимо этих вскрытий, существуют общие вскрытия, раскладывающие многочлены над конечными полями [605]. К любому алгоритму, безопасность которого определяется композицией многочленов над конечными полями, нужно относиться со скептицизмом, если не с откровенным подозрением.

Алгоритм Yagisawa объединяет возведение в степень mod p с арифметикой mod p-1 [1623], он был взломан в [256]. Другой алгоритм с открытым ключом, Tsujii-Kurosawa-Itoh-Fujioka-Matsumoto [1548], также оказался небезопасным [948]. Небезопасной [717] была и третья система, Luccio-Mazzone [993]. Схема подписи на базе birational перестановок [1425] была взломана на следующий день после ее представления [381]. Несколько схем подписей предложил Тацуаки Окамото (Tatsuaki Okamoto): было доказано, что одна из них так же безопасна, как проблема дискретного логарифма, а другая - как проблема дискретного логарифма *и* проблема разложения на множители [1206]. Аналогичные схемы представлены в [709].

Густавус Симмонс (Gustavus Simmons) предложил использовать в качестве основы алгоритмов с открытыми ключами J-алгебру [1455, 145]. От этой идеи пришлось отказаться после изобретения эффективных методов разложения многочленов на множители [951]. Также были изучены и специальные полугруппы многочленов [1619, 962], но и это ничего не дало. Харальд Нидеррейтер (Harald Niederreiter) предложил алгоритм с открытым ключом на базе последовательностей сдвиговых регистров [1166]. Другой алгоритм использовал слова Линдона (Lyndon) [1476], а третий - prepositional исчисление [817]. Безопасность одного из недавних алгоритмов с открытыми ключами основывалась на проблеме matrix cover [82]. Тацуаки Окамото и Казуо Ота (Kazuo Ohta) провели сравнение ряда схем цифровой подписи в [1212].

Перспективы создания радикально новых и различных алгоритмов с открытыми ключами неясны. В 1988 году Уитфилд Диффи отметил, что большинство алгоритмов с открытыми ключами основаны на одной из трех трудных проблем [492, 494]:

1. Рюкзак: Дано множество уникальных чисел, найти подмножество, сумма которого равна N.
2. Дискретный логарифм: Если p - простое число, а g и M - целые, найти x, для которого выполняется $g^x \equiv M \pmod{p}$.

3. Разложение на множители: Если N - произведение двух простых чисел, то лиьо

   (a) разложить N на множители,

   (b) для заданных целых чисел M и C найти d, для которого $M^d \equiv C \pmod{N}$,

   (c) для заданных целых чисел e и C найти M, для которого $M^e \equiv C \pmod{N}$,

   (d) для заданного целого числа x определить, существует ли целое число y, для которого $x \equiv y^2 \pmod{N}$.

Согласно Диффи [492, 494], проблема дискретных логарифмов была предложена Дж. Гиллом (J. Gill), проблема разложения на множители - Кнутом , а проблема рюкзака - самим Диффи .

Эта узость математических основ криптографии с открытыми ключами немного беспокоит . Прорыв в решении либо проблемы дискретных логарифмов, либо проблемы разложения на множители сделает небезопасными целые классы алгоритмов с открытыми ключами . Диффи показал [492, 494], что подобный риск смягчается двумя факторами:

> 1. Все операции, на которые сейчас опирается криптография с открытыми ключами - умножение, возведение в степень и разложение на множители - представляют собой фундаментальные арифметические явления . Веками они были предметом интенсивного математического изучения, и рост внимания к ним, вызванный применением в криптосистемах с открытыми ключами, увеличил, а не уменьшил наше доверие.
> 2. Наша возможность выполнять большие арифметические вычисления растет равномерно и сейчас позволяет нам реализовывать системы с числами такого размера, чтобы эти системы были чувствительны только к действительно радикальным прорывам в разложении на множители, дискретных логарифмах или извлечении корней .

Как мы уже видели, не все алгоритмы с открытыми ключами, основанные на этих проблемах, безопасны . Сила любого алгоритма с открытым ключом зависит не только от вычислительной сложности проблемы, лежащей в основе алгоритма. Трудная проблема необязательно реализуется в сильном алгоритме . Ади Шамир объясняет это тремя причинами [1415]:

1. Теория сложности обычно связана с отдельными частными случаями проблемы. Криптоаналитик же часто получает большой набор статистически связанных проблем - различные шифротексты, зашифрованные одним и тем же ключом.
2. Вычислительная сложность проблемы обычно измеряется для худшего или среднего случаев . Чтобы быть эффективной в качестве способа шифрования, проблема должна быть трудной для решения по чти во всех случаях.
3. Произвольную сложную проблему необязательно можно преобразовать в криптосистему, к тому же проблема должна позволить встроить в нее лазейку, знание которой и только оно делает возможным простое решение проблемы.

Глава 21
Схемы идентификации

21.1 FEIGE-FIAT-SHAMIR

Схема цифровой подписи и проверки подлинности, разработанная Амосом Фиатом (Amos Fiat) и Ади Шамиром (Adi Shamir), рассматривается в [566, 567]. Уриель Фейге (Uriel Feige), Фиат и Шамир модифицировали алгоритм, превратив его в доказательство подлинности с нулевым знанием [544, 545]. Это лучшее доказательство подлинности с нулевым знанием.

9 июля 1986 года три автора подали заявку на получение патента США [1427]. Из-за возможного военного применения заявка была рассмотрена военными. Время от времени результатом работы Патентное бюро является не выдача патента, а нечто, называемое секретным распоряжением. 6 января 1987 года, за три дня до истечения шестимесячного периода, по просьбе армии Патентное бюро издало такое распоряжение. Заявило, что " . . . раскрытие или публикация предмета заявки . . . может причинить ущерб национальной безопасности . . ." Авторам было приказано уведомить всех граждан США, которые по тем или иным причинам узнали о проводимых исследованиях, что несанкционированное раскрытие информации может закончиться двумя годами тюремного заключения, штрафом $10,000 или тем и другим одновременно. Более того, авторы должны были сообщить Уполномоченному по патентам и торговым знакам обо всех иностранных гражданах, которые получили доступ к этой информации.

Это было нелепо. В течение второй половины 1986 года авторы представляли свою работу на конференциях в Израиле, Европе и Соединенных Штатах. Они даже не были американским гражданами, и вся работа была выполнена в Институте Вейцмана (Weizmann) в Израиле.

Слухи об этом стали распространяться в научном сообществе и прессе. В течение двух дней секретное распоряжение было аннулировано. Шамир и его коллеги считают, что за отменой секретного распоряжения стояло NSA, хотя никаких официальных комментариев не было. Дальнейшие подробности этой причудливой истории приведены в [936].

Упрощенная схема идентификации Feige-Fiat-Shamir

Перед выдачей любых закрытых ключей арбитр выбирает случайный модуль , n, который является произведением двух больших простых чисел. В реальной жизни длина n должна быть не меньше 512 битов и лучше как можно ближе к 1024 битам. n может общим для группы контролеров. (Использование чисел Блюма (Blum) облегчит вычисления, но не является обязательным для безопасности .)

Для генерации открытого и закрытого ключей Пегги доверенный арбитр выбирает число v, являющееся квадратичным остатком mod n. Другими словами выбирается v так, чтобы уравнение $x^2 \equiv v \pmod{n}$ имело решение, и существовало $v^{-1} \bmod n$. Это v и будет открытым ключом Пегги. Затем вычисляется наименьшее s, для которого $s \equiv \text{sqrt}\ (v^{-1}) \pmod{n}$. Это будет закрытый ключ Пегги. Используется следующий протокол идентификации.

(1) Пегги выбирает случайное r, меньшее n. Затем она вычисляет $x = -r^2 \bmod n$ и посылает x Виктору.

(2) Виктор посылает Пегги случайный бит b.

(3) Если $b = 0$, то Пегги посылает Виктору r. Если $b = 1$, то Пегги посылает Виктору $y = r*s \bmod n$.

(4) Если $b = 0$, Виктор проверяет, что $x = -r^2 \bmod n$, убеждаясь, что Пегги знает значение sqrt(x). Если $b = 1$, Виктор проверяет, что $x = y^2*v \bmod n$, убеждаясь, что Пегги знает значение sqrt(v^{-1}).

Это один этап протокола, называемый **аккредитацией**. Пегги и Виктор повторяют этот протокол t раз, пока Виктор не убедится, что Пегги знает s. Это протокол "разрезать и выбрать". Если Пегги не знает s, она может подобрать r так, что она сможет обмануть Виктора, если он пошлет ей 0, или она может подобрать r так, что она сможет обмануть Виктора, если он пошлет ей 1. Она не может сделать одновременно и то, и другое. Вероятность, что ей удастся обмануть Виктора один раз, равна 50 процентам. Вероятность, что ей удастся обмануть его t раз, равна $1/2^t$.

Виктор может попробовать вскрыть протокол, выдавая себя за Пегги. Он может начать выполнение протокола с с другим контролером, Валерией. На шаге (1) вместо выбора случайного r ему останется просто использовать значение r, которое Пегги использовало в прошлый раз. Однако, вероятность того, что Валерия на шаге (2) выберет то же значение b, которое Виктор использовал в протоколе с Пегги, равна 1/2. Следовательно, вероятность, что он обманет Валерию, равна 50 процентам. Вероятность, что ему удастся обмануть ее t раз, равна $1/2^t$.

Чтобы этот протокол работал, Пегги никогда не должна использовать r повторно. В противном случае, если Виктор на шаге (2) пошлет Пегги другой случайный бит, то он получит оба ответа Пегги. Тогда даже по одному из них он сможет вычислить s, и для Пегги все закончится.

Схема идентификации Feige-Fiat-Shamir

В своих работах [544, 545], Фейге, Фиат и Шамир показали, как параллельная схема может повысить число аккредитаций на этап и уменьшить взаимодействия Пегги и Виктора.

Сначала, как и в предыдущем примере, генерируется n, произведение двух больших простых чисел. Для генерации открытого и закрытого ключей Пегги сначала выбирается k различных чисел: $v_1, v_2, \ldots v_k$, где каждое v_i является квадратичным остатком mod n. Иными словами, v_i выбираются так, чтобы $x^2 \equiv v_i \pmod n$ имело решение, и существовало $v_i^{-1} \bmod n$. Строка, $v_1, v_2, \ldots v_k$, служит открытым ключом. Затем вычисляются наименьшие s_i, для которых $s_i \equiv \text{sqrt}\ (v_i^{-1}) \pmod n$. Строка $s_1, s_2, \ldots s_k$, служит закрытым ключом.

Выполняется следующий протокол:

(1) Пегги выбирает случайное r, меньшее n. Затем она вычисляет $x = -r^2 \bmod n$ и посылает x Виктору.

(2) Виктор посылает Пегги строку из k случайных битов: $b_1, b_2, \ldots b_k$.

(3) Пегги вычисляет $y = r * (s_1^{b_1} * s_2^{b_2} * \ldots * s_k^{b_k}) \bmod n$. (Она перемножает вместе значения s_i, соответствующие b_i=1. Если первым битом Виктора будет 1, то s_1 войдет в произведение, а если первым битом будет 0, то нет, и т.д.) Она посылает y Виктору.

(4) Виктор проверяет, что $x = y^2 * (v_1^{b_1} * v_2^{b_2} * \ldots * v_k^{b_k}) \bmod n$. (Он перемножает вместе значения v_i, основываясь га случайной двоичной строке. Если его первым битом является 1, то v_1 войдет в произведение, а если первым битом будет 0, то нет, и т.д.)

Пегги и Виктор повторяют этот протокол t раз, пока Виктор не убедится, что Пегги знает $s_1, s_2, \ldots s_k$.

Вероятность, что Пегги удастся обмануть Виктор t раз, равна $1/2^{kt}$. Авторы рекомендуют использовать вер оятность мошенничества $1/2^{20}$ и предлагают значения $k = 5$ и $t = 4$. Если у вас склонность к мании преследов ания, увеличьте эти значения.

Пример

Взглянем на работу этого протокола небольших числах. Если $n = 35$ (два простых числа - 5 и 7), то возможными квадратичными остатками являются:

1: $x^2 \equiv 1 \pmod{35}$ имеет решения: $x = 1, 6, 29, 34$.

4: $x^2 \equiv 4 \pmod{35}$ имеет решения: $x = 2, 12, 23, 33$.

9: $x^2 \equiv 9 \pmod{35}$ имеет решения: $x = 3, 17, 18, 32$.

11: $x^2 \equiv 11 \pmod{35}$ имеет решения: $x = 9, 16, 19, 26$.

14: $x^2 \equiv 14 \pmod{35}$ имеет решения: $x = 7, 28$.

15: $x^2 \equiv 15 \pmod{35}$ имеет решения: $x = 15, 20$.

16: $x^2 \equiv 16 \pmod{35}$ имеет решения: $x = 4, 11, 24, 31$.

21: $x^2 \equiv 21 \pmod{35}$ имеет решения: $x = 14, 21$.

25: $x^2 \equiv 25 \pmod{35}$ имеет решения: $x = 5, 30$.

29: $x^2 \equiv 29 \pmod{35}$ имеет решения: $x = 8, 13, 22, 27$.

30: $x^2 \equiv 30 \pmod{35}$ имеет решения: $x = 10, 25$.

Обратными значениями (mod 35) и их квадратными корнями являются:

v	v^{-1}	s=sqrt(v^{-1})
1	1	1
4	9	3
9	4	2
11	16	4

16	11	9
29	29	8

Обратите внимание, что у чисел 14, 15, 21, 25 и 30 нет обратных значений mod 35, так как они не взаимно просты с 35. Это имеет смысл, так как должно быть (5 - 1) * (7 - 1)/4 квадратичных остатков mod 35, взаимно простых с 35: НОД(x, 35) = 1 (см. раздел 11.3).

Итак, Пегги получает открытый ключ, состоящий из k = 4 значений: {4,11,16,29}. Соответствующим закрытым ключом является {3,4,9,8}. Вот один этап протокола.

(1) Пегги выбирает случайное r=16, вычисляет 16^2 mod 35 = 11 и посылает его Виктору.

(2) Виктор посылает Пегги строку случайных битов: {1, 1, 0, 1}

(3) Пегги вычисляет $16*(3^1*4^1*9^0*8^1)$ mod 35 = 31 и посылает его Виктору.

(4) Виктор проверяет, что $31^2*(4^1*11^1*16^0*29^1)$ mod 35 =11.

Пегги и Виктор повторяют этот протокол t раз, каждый раз с новым случайным r, пока Виктор будет убежден.

Небольшие числа, подобные использованным в примере, не обеспечивают реальной безопасности . Но когда длина n равна 512 и более битам, Виктор не сможет узнать о закрытом ключе Пегги ничего кроме того факта, что Пегги действительно знает его.

Улучшения

В протокол можно встроить идентификационные данные. Пусть I - это двоичная строка, представляющая идентификатор Пегги: имя, адрес, номер социального страхования, размер головного убора, любимый сорт прохладительного напитка и другая личная информация. Используем однонаправленную хэш-функцию $H(x)$ для вычисления $H(I,j)$, где j - небольшое число, добавленное к I. Найдем набор j, для которых $H(I,j)$ - это квадратичный остаток по модулю n. Эти значения $H(I,j)$ становятся $v_1, v_2, \ldots v_k$ (j не обязаны быть квадратичными остатками). Теперь открытым ключом Пегги служит I и перечень j. Пегги посылает I и перечень j Виктору перед шагом (1) протокола (или Виктор загружает эти значения с какой-то открытой доски объявлений), И Виктор генерирует $v_1, v_2, \ldots v_k$ из $H(I,j)$.

Теперь, после того, как Виктор успешно завершит протокол с Пегги, он будет убежден, что Трент, которому известно разложение модуля на множители, сертифицировал связь между I и Пегги, выдав ей квадратные корни из v_i, полученные из I. (См. раздел 5.2.) Фейге, Фиат и Шамир добавили следующие замечания [544, 545]:

> Для неидеальных хэш-функций можно посоветовать рандомизировать I, добавляя к нему длинную случайную строку R. Эта строка выбирается арбитром и открывается Виктору вместе с I.
>
> В типичных реализациях k должно быть от 1 до 18. Большие значения k могут уменьшить время и трудности связи, уменьшая количество этапов.
>
> Длина n должна быть не меньше 512 битов. (Конечно, с тех пор разложение на множители заметно продвинулось .)
>
> Если каждый пользователь выберет свое собственное n и опубликует его в файле открытых ключей, то можно обойтись и без арбитра. Однако такой RSA-подобный вариант делает схему заметно менее удобной.

Схема подписи Fiat-Shamir

Превращение этой схемы идентификации в схему подписи - это, по сути, вопрос превращения Виктора в хэш-функциию. Главным преимуществом схемы цифровой подписи Fiat-Shamir по сравнению с RSA является ее скорость: для Fiat-Shamir нужно всего лишь от 1 до 4 процентов модульных умножений, используемых в RSA. В этом протоколе снова вернемся к Алисе и Бобу.

Смысл переменных - такой же, как и в схеме идентификации. Выбирается n - произведение двух больших простых чисел. Генерируется открытый ключ, $v_1, v_2, \ldots v_k$, и закрытый ключ, $s_1, s_2, \ldots s_k$, где $s_i \equiv \text{sqrt}\ (v_i^{-1})$ (mod n).

(1) Алиса выбирает t случайных целых чисел в диапазоне от 1 до n - $r_1, r_2, \ldots, r_t$ - и вычисляет $x_1, x_2, \ldots x_t$, такие что $x_i = r_i^2$ mod n.

(2) Алиса хэширует объединение сообщения и строки x_i, создавая битовый поток: $H(m, x_1, x_2, \ldots x_t)$. Она использует первые $k*t$ битов этой строки в качестве значений b_{ij}, где i пробегает от1 до t, а j от 1 до k.

(3) Алиса вычисляет $y_1, y_2, \ldots y_t$,, где $y_i = r_i * (s_1^{b_{i1}} * s_2^{b_{i2}} * \ldots * s_k^{b_{ik}})$ mod n

(Для каждого i она перемножает вместе значения s_i, в зависимости от случайных значений b_{ij}. Если b_{ij}=1, то s_i участвует в вычислениях, если b_{ij}=0, то нет.)

(4) Алиса посылает Бобу m, все биты b_{ij}, и все значения y_i. У Боба уже есть открытый ключ Алисы: $v_1, v_2, \ldots v_k$.

(5) Боб вычисляет $z_1, z_2, \ldots z_t$, где $z_i = y^2 * (v_1^{b_{i1}} * v_2^{b_{i2}} * \ldots * v_k^{b_{ik}}) \bmod n$

(И снова Боб выполняет умножение в зависимости от значений b_{ij}.) Также обратите внимание, что z_i должно быть равно x_i.

(6) Боб проверяет, что первые $k*t$ битов $H(m, z_1, z_2, \ldots z_t)$ - это значения b_{ij}, которые прислала ему Алиса.

Как и в схеме идентификации безопасность схемы подписи пропорциональна $1/2^{kt}$. Она также зависит от сложности разложения n на множители. Фиат и Шамир показали, что подделка подписи облегчается, если сложность разложения n на множители заметно меньше 2^{kt}. Кроме того, из-за вскрытия методом дня рождения (см. раздел 18.1), они рекомендуют повысить $k*t$ от 20 по крайней мере до 72, предлагая $k = 9$ и $t = 8$.

Улучшенная схема подписи Fiat-Shamir

Сильвия Микали (Silvia Micali) и Ади Шамир улучшили протокол Fiat-Shamir [1088]. Они выбирали $v_1, v_2, \ldots v_k$ так, чтобы они были первыми k простыми числами. То есть

$v_1 = 1$, $v_2 = 3$, $v_3 = 5$, и т.д.

Это открытый ключ. Закрытым ключом, $s_1, s_2, \ldots s_k$, служат случайные квадратные корни, определяемые как

$s_i = \text{sqrt}\ (v_i^{-1}) \pmod n$

В этой версии у каждого участника должен быть свой n. Такая модификация облегчает проверку подписей, не влияя на время генерации подписей и их безопасность.

Другие улучшения

На основе алгоритма Fiat-Shamir существует и N-сторонняя схема идентификации [264]. Два других улучшения схемы Fiat-Shamir в [1218]. Еще один вариант - в [1368].

Схема идентификации Ohta-Okamoto

Этот протокол является вариантом схемы идентификации Feige-Fiat-Shamir, его безопасность основана на трудности разложения на множители [1198, 1199]. Эти же авторы разработали схему с несколькими подписями (см. раздел 23.1), с помощью которой различные люди могут последовательно подписывать [1200]. Эта схема была предложена для реализации на интеллектуальных карточках [850].

Патенты

Fiat-Shamir запатентован [1427]. При желании получить лицензию на алгоритм свяжитесь с Yeda Research and Development, The Weizmann Institute of Science, Rehovot 76100, Israel.

21.2 GUILLOU-QUISQUATER

Feige-Fiat-Shamir был первым практическим протоколом идентификации. Он минимизировал вычисления, увеличивая число итераций и аккредитаций на итерацию. Для ряда реализаций, например, для интеллектуальных карточек, это не слишком подходит. Обмены с внешним миром требуют времени, а хранение данных для каждой аккредитации может быстро исчерпать ограниченные возможности карточки.

Луи Гиллу (Louis Guillou) и Жан-Жак Кискатр (Jean-Jacques Quisquater) разработали алгоритм идентификации с нулевым знанием, который больше подходит для подобных приложений [670, 1280]. Обмены между Пегги и Виктором, а также параллельные аккредитации в каждом обмене сведены к абсолютному минимуму : для каждого доказательства существует только один обмен, в котором - только одна аккредитация. Для достижения того же уровня безопасности при использовании схемы Guillou-Quisquater потребуется выполнить в три раза больше вычислений, чем при Feige-Fiat-Shamir. И, как и Feige-Fiat-Shamir, этот алгоритм идентификации можно превратить в алгоритм цифровой подписи.

Схема идентификации Guillou-Quisquater

Пегги - это интеллектуальная карточка, которая собирается доказать свою подлинность Виктору. Идентификация Пегги проводится по ряду атрибутов, представляющих собой строку данных содержащих название карточки, период действия, номер банковского счета и другие, подтверждаемые ее применимость, данные. Эта битовая строка называется J. (В реальности строка атрибутов может быть очень длинной, и в качестве J используется ее хэш-значение. Это усложнение никак не влияет на протокол.) Эта строка аналогична открытому ключу. Другой открытой информацией, общей для всех "Пегги", которые могут использовать это приложение, является показатель степени v и модуль n, где n - это произведение двух хранящихся в секрете простых чисел. Закрытым

ключом служит B, рассчитываемое так, чтобы $JB^v \equiv 1 \pmod{n}$.

Пегги посылает Виктору свои атрибуты J. Теперь она хочет доказать Виктору, что это именно ее атрибуты. Для этого она должна убедить Виктора, что ей известно B. Вот этот протокол:

(1) Пегги выбирает случайное целое r, находящееся в диапазоне от 1 до n-1. Она вычисляет $T = r^v \bmod n$ и отправляет его Виктору.

(2) Виктор выбирает случайное целое d, находящееся в диапазоне от 0 до v-1. Он посылает d Пегги.

(3) Пегги вычисляет $D = rB^d \bmod n$ и посылает его Виктору.

(4) Виктор вычисляет $T' = D^v J^d \bmod n$. Если $T \equiv T' \pmod{n}$, то подлинность Пегги доказана.

Математика не слишком сложна:

$T' = D^v J^d = (rB^d)^v J^d = r^v B^{dv} J^d = r^v (B^v J)^d = r^v = r' \equiv T \pmod{n}$, так как $JB^v \equiv 1 \pmod{n}$

Схема подписи Guillou-Quisquater

Эту схему идентификации можно превратить в схему подписи, также пригодную для реализации в интелле ктуальных карточках [671, 672]. Открытый и закрытый ключи не меняются. Вот как выглядит протокол:

(1) Алиса выбирает случайное целое r, находящееся в диапазоне от 1 до n-1. Она вычисляет $T = r^v \bmod n$.

(2) Алиса вычисляет $d = H(M,T)$, где M - подписываемое сообщение, а $H(x)$ - однонаправленная хэш-функция. Значение d, полученное с помощью хэш-функции, должно быть в диапазоне от 0 до v-1 [1280]. Если выход хэш-функции выходит за этот диапазон, он должен быть приведен по модулю v.

(3) Алиса вычисляет $D = rB^d \bmod n$. Подпись состоит из сообщения M, двух вычисленных значений, d and D, и ее атрибутов J. Она посылает подпись Бобу.

(4) Боб вычисляет $T' = D^v J^d \bmod n$. Затем он вычисляет $d' = H(M,T')$. Если $d \equiv d'$, то Алиса знает B, и ее подпись действительна.

Несколько подписей

Что если несколько человек захотят подписать один и тот же документ? Проще всего, чтобы они подписали его порознь, но рассматриваемая схема подписи делает это лучше. Пусть Алиса и Боб подписывают документ, а Кэрол проверяет подписи, но в процесс подписания может быть вовлечено произвольное количество людей. Как и раньше, Алиса и Боб обладают уникальными значениями J и B: (J_A,B_A) и (J_B,B_B). Значения n и v являются общими для всей системы.

(1) Алиса выбирает случайное целое r_A, находящееся в диапазоне от 1 до n-1. Она вычисляет $T_A = r_A{}^v \bmod n$ и посылает T_A Бобу.

(2) Боб выбирает случайное целое r_B, находящееся в диапазоне от 1 до n-1. Он вычисляет $T_B = r_B{}^v \bmod n$ и посылает T_B Алисе.

(3) Алиса и Боб, каждый вычисляет $T = (T_A * T_B) \bmod n$.

(4) Алиса и Боб, каждый вычисляет $d = H(M,T)$, где M - подписываемое сообщение, а $H(x)$ - однонаправленная хэш-функция. Значение d, полученное с помощью хэш-функции, должно быть в диапазоне от 0 до v-1 [1280]. Если выход хэш-функции выходит за этот диапазон, он должен быть приведен по модулю v.

(5) Алиса вычисляет $D_A = r_A B_A{}^d \bmod n$ и посылает D_A Бобу.

(6) Боб вычисляет $D_B = r_B B_B{}^d \bmod n$ и посылает D_B Алисе.

(7) Алиса и Боб, каждый вычисляет $D = D_A\, D_B \bmod n$. Подпись состоит из сообщения M, двух вычисленных значений, d and D, и атрибутов обоих подписывающих: J_A и J_B.

(8) Кэрол вычисляет $J = J_A\, J_B \bmod n$.

(9) Кэрол вычисляет $T' = D^v J^d \bmod n$. Затем она вычисляет $d' = H(M,T')$. Если $d \equiv d'$, то множественная подпись действительна.

Этот протокол может быть расширен на любое количество людей. Для этого подписывающие сообщение люди должны перемножить свои значения T_i на этапе (3), и свои значения D_i на этапе (7). Чтобы проверить множественную подпись, нужно на этапе (8) перемножить значения J_i подписывающих (8). Либо все подписи правильны, либо существует по крайней мере одна неправильная подпись.

21.3 SCHNORR

Безопасность схемы проверки подлинности и подписи Клауса Шнорра [1396,1397] опирается на трудность вычисления дискретных логарифмов. Для генерации пары ключей сначала выбираются два простых числа, p и q так, чтобы q было сомножителем p-1. Затем выбирается a, не равное 1, такое что $a^q \equiv 1 \pmod{p}$. Все эти числа могут быть свободно опубликованы и использоваться группой пользователей.

Для генерации конкретной пары ключей выбирается случайное число, меньшее q. Оно служит закрытым ключом, s. Затем вычисляется открытый ключ $v = a^{-s} \bmod p$.

Протокол проверки подлинности

(1) Пегги выбирает случайное число r, меньшее q, и вычисляет $x = a^r \bmod p$. Эти вычисления являются предварительными и могут быть выполнены задолго до появления Виктора.

(2) Пегги посылает x Виктору.

(3) Виктор посылает Пегги случайное число e, из диапазона от 0 до 2^{t-1}. (Что такое t, я объясню чуть позже.)

(4) Пегги вычисляет $y = (r + se) \bmod q$ и посылает y to Виктору.

(5) Виктор проверяет, что $x = a^y v^e \bmod p$.

Безопасность алгоритма зависит от параметра t. Сложность вскрытия алгоритма примерно равна 2^t. Шнорр советует использовать p около 512 битов, q - около 140 битов и t - 72.

Протокол цифровой подписи

Алгоритм Schnorr также можно использовать и в качестве протокола цифровой подписи сообщения M. Пара ключей используется та же самая, но добавляется однонаправленная хэш-функция $H(M)$.

(1) Алиса выбирает случайное число r, меньшее q, и вычисляет $x = a^r \bmod p$. Это стадия предварительных вычислений.

(2) Алиса объединяет M и x и хэширует результат:

$e = H(M,x)$

(3) Алиса вычисляет $y = (r + se) \bmod q$. Подписью являются значения e и y, она посылает их Бобу.

(4) Боб вычисляет $x' = a^y v^e \bmod p$. Затем он проверяет, что хэш-значение для объединения M и x' равно e.

$e = H(M,x')$

Если это так, то он считает подпись верной.

В своей работе Шнорр приводит следующие новые свойства своего алгоритма:

> Большая часть вычислений, нужных для генерации подписи и независящих от подписываемого сообщения, может быть выполнена на стадии предварительных вычислений. Следовательно, эти вычисления могут быть выполнены во время простоя и не влияют на скорость подписания. Вскрытие, направленное против стадии предварительных вычислений, рассматривается в [475], я не думаю, что оно имеет практическую ценность.
>
> При одинаковом уровне безопасности длина подписей для Schnorr короче, чем для RSA. Например, при 140-битовом q длина подписей равна всего лишь 212 битам, меньше половины длины подписей RSA. Подписи Schnorr также намного короче подписей EIGamal.

Конечно, из практических соображений количество битов, используемых в этой схеме, может быть уменьшено: например, для схемы идентификации, в которой мошенник должен выполнить диалоговое вскрытие всего лишь за несколько секунд (сравните со схемой подписи, когда мошенник может годами вести расчеты, чтобы выполнить подлог).

Модификация, выполненная Эрни Брикеллом (Ernie Brickell) и Кевином МакКерли (Kevin McCurley), повысила безопасность этого алгоритма [265].

Патенты

Schnorr запатентован в Соединенных Штатах [1398] и многих других странах. В 1993 году PKP приобрело обще мировые права на этот патент (см. раздел 25.5). Срок действия патента США истекает 19 февраля 2008 года.

21.4 Преобразование схем идентификации в схемы подписи

Вот стандартный метод преобразования схемы идентификации в схему подписи: Виктор заменяется однонаправленной хэш-функцией. Перед подписанием сообщение не хэшируется, вместо этого хэширование встраив а-

ется в алгоритм подписи. В принципе, такую манипуляцию можно проделать с любой схемой идентификации .

Глава 22 Алгоритмы обмена ключами

22.1 DIFFIE-HELLMAN

Diffie-Hellman, первый в истории алгоритм с открытым ключом, был изобретен 1976 году [496]. Его безопасность опирается на трудность вычисления дискретных логарифмов в конечном поле (в сравнении с легкостью возведения в степень в том же самом поле. Diffie-Hellman может быть использован для распределения ключей - Алиса и Боб могут воспользоваться этим алгоритмом для генерации секретного ключа - но его нельзя использовать для шифрования и дешифрирования сообщений.

Математика несложна. Сначала Алиса и Боб вместе выбирают большие простые числа n и g так, чтобы g было примитивом mod n. Эти два целых числа хранить в секрете необязательно, Алиса и Боб могут договориться об из использовании по несекретному каналу. Эти числа даже могут совместно использоваться группой пользователей. Без разницы. Затем выполняется следующий протокол:

(1) Алиса выбирает случайное большое целое число x и посылает Бобу

$X = g^x \bmod n$

(2) Боб выбирает случайное большое целое число y и посылает Алисе

$Y = g^y \bmod n$

(3) Алиса вычисляет

$k = Y^x \bmod n$

(4) Боб вычисляет

$k' = X^y \bmod n$

И k, и k' равны $g^{xy} \bmod n$. Никто из подслушивающих этот канал не сможет вычислить это значение, им и известно только n, g, X и Y. Пока они не смогут вычислить дискретный логарифм и раскрыть x или y, они не смогут решить проблему. Поэтому, k - это секретный ключ, который Алиса и Боб вычисляют независимо.

Выбор g и n может заметно влиять на безопасность системы. Число $(n-1)/2$ также должно быть простым [1253]. И, самое главное, n должно быть большим: безопасность системы основана на сложности разложения на множители чисел того же размера, что и n. Можно выбирать любое g, которое является примитивом mod n; нет причин, по которым нельзя было бы выбрать наименьшее возможное g - обычно одноразрядное число. (К тому же, на самом деле, g не должно даже быть примитивом, оно только должно генерировать достаточно большую подгруппу мультипликативной группы mod n.)

Diffie-Hellman с тремя и более участниками *

Протокол обмена ключами Diffie-Hellman легко можно расширить на случай с тремя и более участниками. В приводимом примере Алиса, Боб и Кэрол вместе генерируют секретный ключ.

(1) Алиса выбирает случайное большое целое число x и вычисляет

$X = g^x \bmod n$

(2) Боб выбирает случайное большое целое число y и посылает Кэрол

$Y = g^y \bmod n$

(3) Кэрол выбирает случайное большое целое число z и посылает Алисе

$Z = g^z \bmod n$

(4) Алиса посылает Бобу

$Z' = Z^x \bmod n$

(5) Боб посылает Кэрол *

$X' = X^y \bmod n$

(6) Кэрол посылает Алисе

$Y' = Y^z \bmod n$

(7) Алиса вычисляет

$k = Y'^x \bmod n$

(8) Боб вычисляет

$k = Z^{y} \bmod n$

(9) Кэрол вычисляет

$k = X^{e} \bmod n$

Секретный ключ k равен $g^{xyz} \bmod n$, и никто из подслушивающих каналы связи не сможет вычислить это значение. Протокол можно легко расширить для четверых и более участников, просто добавляются участники и этапы вычислений.

Расширенный Diffie-Hellman

Diffie-Hellman также работает в коммутативных кольцах [1253]. З. Шмули (Z. Shmuley) и Кевин МакКерли (Kevin McCurley) изучили вариант алгоритма, в котором модуль является составным числом [1441, 1038]. В.С. Миллер (V. S. Miller) и Нил Коблиц (Neal Koblitz) расширили этот алгоритм, используя эллиптические кривые [1095, 867]. Тахер ЭльДжамаль (Taher ElGamal) использовал основополагающую идею для разработки алгоритма шифрования и цифровой подписи (см. раздел 19.6).

Этот алгоритм также работает в поле Галуа $GF(2^k)$ [1442, 1038]. В ряде реализаций используется именно этот подход [884, 1631, 1632], так как вычисления выполняются намного быстрее. Но и криптоаналитические вычисления выполняются намного быстрее, поэтому важно тщательно выбирать поле, достаточно большое, чтобы обеспечить нужную безопасность.

Hughes

Этот вариант алгоритма Diffie-Hellman позволяет Алисе генерировать ключ и послать его Бобу [745].

(1) Алиса выбирает случайное большое целое число x и генерирует

$k = g^{x} \bmod n$

(2) Боб выбирает случайное большое целое число y и посылает Алисе

$Y = g^{y} \bmod n$

(3) Алиса посылает Бобу

$X = Y^{x} \bmod n$

(4) Боб вычисляет

$z = y^{-1}$

$k' = X^{z} \bmod n$

Если все выполнено правильно, $k = k'$.

Преимуществом этого протокола над Diffie-Hellman состоит в том, что k можно вычислить заранее, до взаимодействия, и Алиса может шифровать сообщения с помощью k задолго до установления соединения с Бобом. Она может послать сообщение сразу множеству людей, а передать ключ позднее каждому по отдельности.

Обмен ключом без обмена ключом

Если у вас сообщество пользователей, каждый может опубликовать открытый ключ, $X = g^{x} \bmod n$, в общей базе данных. Если Алиса захочет установить связь с Бобом, ей понадобится только получить открытый ключ Боба и генерировать их общий секретный ключ. Она может зашифровать сообщение этим ключом и послать его Бобу. Боб извлечет открытый ключ Алисы и вычислит общий секретный ключ.

Каждая пара пользователей может использовать уникальный секретный ключ, не требуется никаких предварительных обменов данными между пользователями. Открытые ключи должны пройти сертификацию, чтобы предотвратить мошеннические вскрытия, и должны регулярно меняться, но в любом случае это очень умная идея

Патенты

Алгоритм обмена ключами Diffie-Hellman запатентован в Соединенных Штатах [718] и Канаде [719]. Группа, называющаяся Public Key Partners (PKP, Партнеры по открытым ключам), получила вместе с другими патентами в области криптографии с открытыми ключами получила лицензию на этот патент (см. раздел 25.5). Срок действия патента США истекает 29 апреля 1997 года.

22.2 Протокол "точка-точка"

Обмен ключами Diffie-Hellman чувствителен к вскрытию "человек в середине". Одним из способов предотвратить это, является необходимость для Алисы и Боба подписывать сообщения, которые они посылают друг другу [500].

Этот протокол предполагает, что у Алисы есть сертифицированный открытый ключ Боба, а у Боба есть сертифицированный открытый ключ Алисы. Эти сертификаты подписаны некоторым заслуживающим доверия органом власти, непосредственно не участвующим в протоколе. Вот как Алиса и Боб генерируют секретный ключ k.

(1) Алиса генерирует случайное число x и посылает его Бобу.

(2) Боб генерирует случайное число y. Используя протокол Diffie-Hellman, он вычисляет общий ключ k на базе x и y. Он подписывает x и y и шифрует подпись ключом k. Затем он посылает получившееся вместе с y Алисе.

$y, E_k(S_B(x,y))$

(3) Алиса также вычисляет k. Она расшифровывает оставшуюся часть сообщения Боба и проверяет его подпись. Затем она посылает Бобу подписанное сообщение, состоящее из x и y, зашифрованных общим ключом k.

$E_k(S_A(x,y))$

(4) Боб расшифровывает сообщение и проверяет подпись Алисы.

22.3 Трехпроходный протокол Шамира

Этот изобретенный Ади Шамиром но никогда не опубликованный протокол позволяет Алисе и Бобу безопасно обмениваться информацией, не используя предварительного обмена ни секретными, ни открытыми ключами [1008]. Он предполагает использование коммутативного симметричного шифра, для которого:

$E_A(E_B(P)) = E_B(E_A(P))$

Секретный ключ Алисы - A, а Боба - B. Алиса хочет послать сообщение M Бобу. Вто этот протокол.

(1) Алиса шифрует M своим ключом и посылает его Бобу

$C_1 = E_A(M)$

(2) Боб шифрует C_1 своим ключом и посылает Алисе

$C_2 = E_B(E_A(M))$

(3) Алиса расшифровывает C_2 своим ключом и посылает Бобу

$C_3 = D_A(E_B(E_A(M))) = D_A(E_A(E_B(M))) = E_B(M)$

(4) Боб расшифровывает C_3 своим ключом, получая M.

Коммутативны и обладают совершенной безопасностью одноразовые блокноты, но с этим протоколом они работать не будут. При использовании одноразового блокнота три шифротекста будут выглядеть следующим образом be:

$C_1 = M \oplus A$

$C_2 = M \oplus A \oplus B$

$C_3 = M \oplus B$

Ева, записав эти три сообщения, которыми обмениваются Алиса и Боб, просто выполнит XOR всех этих шифротекстов и восстановит сообщение:

$C_1 \oplus C_2 \oplus C_3 = (M \oplus A) \oplus (M \oplus A \oplus B) \oplus (M \oplus B) = M$

Очевидно, что такой способ работать не будет.

Шамир (и независимо Джим Омура (Jim Omura)) описал похожий на RSA алгоритм шифрования, который будет работать с этим протоколом. Пусть p будет большим большим простым числом, причем множитель p-1 является большим простым. Выберем ключ шифрования e, взаимно простой с p-1. Вычислим d, для которого выполняется $de = 1 \pmod{p - 1}$. Для шифрования сообщения вычисляем

$C = M^e \bmod p$

Для дешифрирования сообщения вычисляем

$M = C^d \bmod p$

По видимому, у Евы нет способа получить M, не решив проблему дискретного логарифма, но это никогда не было доказано.

Как и Diffie-Hellman, этот протокол позволяет Алисе начать секретный обмен информацией с Бобом, не зная ни одного из его ключей. При использовании алгоритма с открытым ключом Алиса должна знать открытый ключ Боба. Применяя трехпроходный алгоритм Шамира, она просто посылает Бобу шифротекст сообщения . То же действие с помощью алгоритма с открытым ключом выглядит следующим образом :

(1) Алиса запрашивает у Боба (или у KDC) его открытый ключ.

(2) Боб (или KDC) посылает Алисе свой открытый ключ.

(3) Алиса шифрует M открытым ключом Боба и посылает его Бобу.

Трехпроходный алгоритм Шамира не может устоять перед вскрытием "человек в середине" .

22.4 COMSET

COMSET (COMmunications SETup, установление связи) это протокол одновременной идентификации и обмена ключом, разработанный для проекта RIPE [1305] (см. раздел 25.7). С помощью криптографии с открытыми ключами он позволяет Алисе и Бобу идентифицировать друг друга, при этом обмениваясь секретным ключом.

Математической основой COMSET служит схема Rabin [1283] (см. раздел 19.5). Сама схема впервые была предложена в [224]. См. подробности в [1305].

22.5 Обмен зашифрованными ключами

Протокол обмена зашифрованными ключами (Encrypted Key Exchange, EKE) был разработан Стивом Белловином (Steve Bellovin) и Майклом Мерриттом (Michael Merritt) [109]. Он обеспечивает безопасность и проверку подлинности в компьютерных сетях, по новому используя и симметричную криптографию, и криптографию с открытыми ключами: общий секретный ключ используется для шифрования генерированного случайным образом открытого ключа.

Базовый протокол EKE

Алиса и Боб (два пользователя, клиент и сервер, или кто угодно) имеют общий пароль P. Используя следующий протокол, они могут проверить подлинность друг друга и генерировать общий сеансовый ключ K.

(1) Алиса Случайным образом генерирует пару "открытый ключ/закрытый ключ" . Она шифрует открытый ключ K' с помощью симметричного алгоритма, используя P в качестве ключа: $E_P(K')$. Она посылает Бобу

$A, E_P(K')$

(2) Боб знает P. Он расшифровывает сообщение, получая K'. Затем он генерирует случайный сеансовый ключ K шифрует его открытым ключом, который он получил от Алисы, а затем используя P качестве ключа. Он посылает Алисе

$E_P(E_{K'}(K)$

(3) Алиса расшифровывает сообщение, получая K. Она генерирует случайную строку R_A, шифрует ее с помощью K и посылает Бобу

$E_K(R_A)$

(4) Боб расшифровывает сообщение, получая R_A. Он генерирует другую случайную строку, R_B, шифрует обе строки ключом K и посылает Алисе результат.

$E_K(R_A,R_B)$

(5) Алиса расшифровывает сообщение, получая R_A и R_B. Если строка R_A, полученная от Боба, - это та самая строка, которую она послала Бобу на этапе (3), она, используя K, шифрует R_B и посылает ее Бобу.

$E_K(R_B)$

(6) Боб расшифровывает сообщение, получая R_B. Если строка R_B, полученная от Алисы, - это та самая строка, которую он послал ей на этапе (4), завершен. Теперь обе стороны могут обмениваться информацией, используя K в качестве сеансового ключа.

На этапе (3) и Алиса, и Боб знают K' и K. K - это сеансовый ключ, он может быть использован для шифрования всех других сообщений, которыми обмениваются Алиса и Боб. Ева, сидя между Алисой и Бобом, знает только $E_P(K')$, $E_P(E_{K'}(K))$ и несколько сообщений, зашифрованных K. В других протоколах Ева могла бы попробовать угадать P (люди все время любят выбирать плохие пароли, и если Ева достаточно умна, она может этот пароль) и затем проверить свои предположения. В рассматриваемом протоколе Ева не может проверять свои предположения, не вскрыв при этом и алгоритм с открытым ключом. И, если K' и K выбираются случайным образом, то эта проблема будет непреодолимой.

Ответная часть протокола, этапы (3) - (6), обеспечивает подтверждение. Этапы (3) - (5) доказывают Алисе, что Боб знает K, этапы (4) - (6) доказывают Бобу, что Алиса знает K. Обмен метками времени, используемый в протоколе Kerberos, решает ту же задачу.

EKE может быть реализован с множеством алгоритмов с открытыми ключами: RSA, ElGamal, Diffie-Hellman. Проблемы с безопасностью возникают при реализации EKE с алгоритмом рюкзака (даже без учета проблем безопасности, присущих самим алгоритмам рюкзака): нормальное распределение шифротекста сообщений сводит на нет преимущества EKE.

Реализация EKE с помощью RSA

Алгоритм RSA кажется идеальным для такого использования, но есть ряд тонких проблем. Авторы рекомендуют шифровать на этапе (1) только показатель степени, посылая модуль. Объяснение этого совета и другие тонкости, связанные с использованием RSA, можно найти [109].

Реализация EKE с помощью ElGamal

Реализация EKE на базе алгоритма ElGamal проста, можно даже упростить основной протокол. Используя обозначения из раздела 19.6, g и p служат частями открытого ключа, общими для всех пользователей. Закрытым ключом является случайное число r. Открытым - $g^r \bmod p$. На этапе (1) Алиса посылает Бобу следующее сообщение

Алиса, $g^r \bmod p$

Обратите внимание, что этот открытый ключ не нужно шифровать с помощью P. В общем случае это неверно, но это так для алгоритма ElGamal algorithm. Подробности в [109].

Боб выбирает случайное число R (для алгоритма ElGamal, независимо от других случайных чисел, выбираемых для EKE), и сообщение, которое он посылает Алисе на этапе (2), выглядит так

$E_P(g^R \bmod p, Kg^{rR} \bmod p)$

Существующие ограничения на выбор переменных для ElGamal были приведены в разделе 19.6.

Реализация EKE с помощью Diffie-Hellman

При использовании протокола Diffie-Hellman K генерируется автоматически. Окончательный протокол еще проще. Значения g и n определяются для всех пользователей сети.

(1) Алиса выбирает случайное число r_A и посылает Бобу

$A,\ g^{r_A} \bmod n$

При использовании Diffie-Hellman Алисе не нужно шифровать с помощью P свое первое сообщение.

(2) Боб выбирает случайное число r_B и вычисляет

$K = g^{r_A * r_B} \bmod n$

Он генерирует случайную строку R_B, затем вычисляет и посылает Алисе:

$E_P(g^{r_B} \bmod n), E_K(R_B)$

(3) Алиса расшифровывает первую половину сообщения Боба, получая $g^{r_B} \bmod n$. Затем она вычисляет K и использует его для шифрования R_B. Она генерирует другую случайную строку R_A,, шифрует обе строки ключом K и посылает результат Бобу.

$E_K(R_A,,R_B)$

(4) Боб расшифровывает сообщение, получая R_A, и R_B. Если полученная от Алисы строка R_B совпадает с той, которую он посылал ей на этапе (2), он шифрует R_A ключом K и посылает результат Алисе.

$E_K(R_A)$

(5) Алиса расшифровывает сообщение, получая R_A. Если полученная от Боба строка R_A совпадает с той, которую она посылала Бобу на этапе (3), протокол завершается. Теперь стороны могут обмениваться сообщениями, используя K в качестве сеансового ключа .

Усиление EKE

Белловин (Bellovin) и Мерритт (Merritt) предложили улучшение запросно-ответной части алгоритма , которое позволяет избежать возможного вскрытия при обнаружении криптоаналитиком ста рого значения K.

На базовый протокол EKE. На этапе (3) Алиса генерирует другое случайное число S_A и посылает Бобу

$E_K(R_A, S_A)$

На этапе (4), Боб генерирует другое случайное число S_B и посылает Алисе

$E_K(R_A,,R_B,S_B)$

Теперь Алиса и Боб могут вычислить истинный сеансовый ключ , $S_A \oplus S_B$. Этот ключ в дальнейшем используется для сообщений, которыми обмениваются Алиса и Боб, K используется в качестве ключа обмена ключами.

Посмотрим на уровни защиты, предоставляемые EKE. Восстановленное значение S не дает Еве никакой информации о P, так как P никогда не используется для шифрования чего-то такого, что ведет непосредственно к S. Криптоаналитическое вскрытие K также невозможно, K используется только для шифрования случайных данных, а S никогда не шифруется отдельно.

Расширенный EKE

Протокол EKE страдает одним серьезным недостатком: он требует, чтобы обе стороны знали P. В большинстве систем авторизации доступа хранятся значения однонаправленной хэш-функции паролей пользователей, а не сами пароли (см. раздел 3.2). Протокол Расширенный EKE (Augmented EKE, A-EKE) использует в варианте EKE на базе Diffie-Hellman значение однонаправленной хэш-функции пароля пользователя в качестве ключа сверхшифрования. Затем пользователь посылает дополнительное сообщение, основанное на реальном пароле, это сообщение удостоверяет заново выбранный сеансовый ключ .

Вот как это работает. Как и обычно, Алиса и Боб хотят проверить подлинность друг друга и генерировать общий ключ. Они выбирают какую-нибудь схему цифровой подписи, в которой в качестве закрытого ключа может использоваться любое число, а открытый ключ получается из закрытого, а не генерируется отдельно . Прекрасно подходят алгоритмы ElGamal и DSA. Пароль Алисы P (или, может быть, какое-нибудь простое хэш-значение этого пароля) будет использоваться в качестве закрытого ключа и как P'.

(1) Алиса выбирает случайный показатель степени R_a и отправляет

$E_{P'}(g^{r_B} \bmod n)$

(2) Боб, который знает только P' и не может получить из него P, выбирает R_b и посылает

$E_P(g^{r_B} \bmod n)$

(3) Алиса и Боб вычисляют общий сеансовый ключ $K= g^{r_A * r_B} \bmod n$. Наконец Алиса доказывает, что она сама знает P, а не только P', посылая

$E_K(S_P(K))$

Боб, который знает K и P', может расшифровать и проверить подпись. Только Алиса могла прислать это сообщение, так как только она знает P. Самозванец, добывший копию файла паролей Боба, может попытаться P, но он не сможет подписать сеансовый ключ.

Схема A-EKE не работает с вариантом EKE, использующим открытые ключи, так как в этом протоколе одна сторона выбирает сеансовый ключ и навязывает его другой. Это позволяет взломщику, заполучившему P', выполнить вскрытие "человек в середине".

Применения EKE

Белловин и Мерритт предлагают использовать этот протокол для безопасной телефонной связи [109]:

> Предположим, что развернута сеть шифрующих телефонных аппаратов. Если кто-нибудь хочет воспользоваться таким телефоном, то понадобится определенная ключевая информация. Общепринятые решения. . . требуют, чтобы у звонящего был физический ключ. Во многих ситуациях это нежелательно. EKE позволяет использовать короткий, вводимый с клавиатуры пароль, обеспечивая гораздо более длинный сеансовый ключ .
>
> EKE мог бы быть полезен и для сотовой связи. Мошенничество представляет собой большую проблему сотовой телефонии, EKE может помочь защититься от него (и обеспечить закрытость звонка) за счет продажи телефонов, бесполезных без

введения PIN-кода. Так как PIN-код не хранится в телефоне, его невозможно извлечь из украденного экземпляра.

Главная сила EKE состоит в том, что криптография с открытыми ключами и симметричная криптография объединяются и усиливают друг друга:

> В общей перспективе EKE работает как *усилитель секретности*. То есть, его можно использовать для усиления сравнительно слабых симметричных и асимметричных систем, используемых вместе. Рассмотрим, например, размер ключа, необходимый для обеспечения безопасности при использовании обмена ключом - показателем степени. Как показали ЛаМачча (LaMacchia) и Одлыжко (Odlyzko) [934], даже модули с размерами, считавшимися безопасными, (а именно, 192 бита) чувствительны к вскрытию, занимающему несколько минут компьютерного времени. Но их вскрытие становится невозможным, если необходимо перед применением вскрытия угадать пароль.
>
> С другой стороны, сложность вскрытия обмена ключами - показателями степени может быть использована для срыва попыток угадать пароль. Возможность вскрытия угадыванием пароля зависит от скорости проверки каждого предположения. Если для выполнения такой проверки необходимо выполнить обмен ключами - показателями степени, то общее время эффектно возрастает.

EKE запатентован [111].

22.6 Защищенные переговоры о ключе

Эта схема также защищает переговоры о ключе от плохого выбора паролей и вскрытий "человек в середине" [47, 983]. В ней используется хэш-функция двух переменных, обладающая особенным свойством: она часто приводит к столкновениям по первой переменной, и практически никогда - по второй.

$H'(x,y) = H(H(k,x) \bmod 2^m, x)$, где $H(k,x)$ - обычная функция k и x

Вот как выглядит этот протокол. Алиса и Боб используют общий секретный пароль P и уже обменялись секретным ключом K, используя обмен ключом Dime-Hellman. Они используют P для проверки, что их сеансовые ключи одинаковы (и что Ева не предприняла вскрытие "человек в середине"), не позволяя Еве получить P.

(1) Алиса посылает Бобу

$H'(P,K)$

(2) Боб вычисляет $H'(P,K)$ и сравнивает результат со значением, присланным Алисой. Если они совпадают, он посылает Алисе

$H'(H(P,K))$

(3) Алиса вычисляет $H'(H(P,K))$ и сравнивает результат со значением, полученным от Боба.

Если Ева пытается выполнить вскрытие "человек в середине", она использует один ключ, K_1, общий с Алисой, и другой, K_2, общий с Бобом. Чтобы обмануть Боба на этапе (2), ей придется вычислить общий пароль и затем послать Бобу $H'(P,K_2)$. При использовании обычной хэш-функции она может перебирать часто встречающиеся пароли, пока не угадает правильный, и затем успешно проникнуть в протокол. Но при использовании предлагаемой хэш-функции, многие пароли дают одно и то же значение при хэшировании с ключом K_1. Поэтому, когда она находит совпадение, то скорее всего это неправильный пароль, и в этом случае Боба обмануть не удастся.

22.7 Распределение ключа для конференции и секретная широковещательная передача

Алиса хочет передать сообщение M сразу нескольким получателям. Однако она совсем не хочет, чтобы кто угодно смог прочесть его. В действительности, ей нужно, чтобы только получатели из определенного подмножества могли правильно раскрыть M. У всех остальных должна получиться чепуха.

Алиса может использовать для каждого получателя отличный ключ (секретный или открытый). Она шифрует сообщение каким-нибудь случайным ключом K. Затем она шифрует копию K каждым из ключей выбранных получателей сообщения. Наконец она широковещательно посылает зашифрованное сообщение, а затем все зашифрованные K. Слушающий передачу Боб либо пытается расшифровать все K своим секретным ключом, пытаясь найти правильный, либо, если Алиса не забыла перечислить получателей своего сообщения, он ищет свое имя, сопровождаемое зашифрованным ключом. Также будет работать и ранее рассмотренная криптография с несколькими ключами.

Другой способ предлагается в [352]. Сначала каждый из получателей договаривается с Алисой об общем для них двоих ключе, который длиннее любого возможного шифрованного сообщения. Все эти ключи должны быть взаимно простыми. Она шифрует сообщение случайным ключом K. Затем она вычисляет одно целое число R, которое по модулю секретного ключа конгруэнтно K, если этот секретный ключ предполагается использовать для расшифровки сообщения, и конгруэнтно нулю в противном случае.

Например, если Алиса хочет, чтобы секрет получили Боб, Кэрол и Эллен, но не Дэйв и Фрэнк, она шифрует

сообщение ключом K и затем вычисляет такое R, что

$R \equiv K \pmod{K_B}$

$R \equiv K \pmod{K_C}$

$R \equiv 0 \pmod{K_D}$

$R \equiv K \pmod{K_E}$

$R \equiv 0 \pmod{K_F}$

Это простая алгебраическая проблема, которая легко может быть решена Алисой . Когда это сообщение будет принято получателями, они вычислят значение полученного ключа по модулю их секретного ключа . Те, кому предназначалось это сообщение, в результате вычисления получат нужный ключ. В противном случае р езультатом будет 0.

Еще один, третий, путь, использующий пороговую схему (см. раздел 3.7), предлагается в [141]. Как и в других способах каждый потенциальный получатель получает секретный ключ . Этот ключ является тенью в еще не созданной пороговой схеме. Алиса сохраняет ряд секретных ключей для себя, внося некоторую непредсказу емость в систему. Пусть всего существует k возможных получателей. Тогда для широковещательной передачи M Алиса шифрует M ключом K и делает следующее.

(1) Алиса выбирает случайное число j. Это число призвано замаскировать количество получателей сообщения. Оно не должно быть слишком большим и даже может равняться нулю .

(2) Алиса создает пороговую схему $(k + j + 1, 2k + j + 1)$, в которой:

K - это секрет.

Секретные ключи адресатов сообщения служат тенями .

Секретные ключи пользователей, которых нет среди получателей сообщения, не являются тенями .

j теней выбираются случайным образом, не совпадая ни с одним секретным ключом.

(3) Алиса широковещательно передает $k + j$ случайно выбранных теней, ни одна из которых не совпадает с тенями этапа (2).

(4) Каждый из слушателей, принявших широковещательное сообщение, добавляет свою тень к полученным $k + j$ теням. Если добавление своей тени позволяет пользователю вычислить секрет, то ему удалось открыть ключ. В противном случае - не удалось .

Другой подход можно найти в [885, 886, 1194]. И еще один - в [1000].

Распределение ключей для конференции

Этот протокол позволяет группе из n пользователей договориться о секретном ключе, используя только н есекретные каналы. Группа использует два общих больших простых числа p и q, а также генератор g той же длины, что и q.

(1) Пользователь i, где i от 1 до n, выбирает случайное число r_i, меньшее q, и широковещательно отправляет

$z_i = g^{r_i} \bmod p$

(2) Каждый пользователь проверяет, что $z_i^q \equiv 1 \pmod{p}$ для всех i от 1 до n.

(3) i-ый пользователь широковещательно передает

$xi = (z_{i+1}/z_{i-1})^{r_i} \bmod p$

(4) i-ый пользователь вычисляет

$K = (z_{i-1})^{nr_i} * x_i^{n-1} * x_{i+1}^{n-2} * \ldots * x_{i-2} \bmod p$

Все вычисления индексов в приведенном протоколе - i-1, i-2 и i+1 - проводятся mod n. По окончании протокола у всех честных пользователей окажется один и тот же K. А все остальные ничего не получат. Однако этот протокол не может устоять перед вскрытием "человек в середине" . Другой протокол, не такой хороший, приведен в [757].

Tateboyashi-Matsuzaki-Newman

Этот протокол распределения ключей подходит для использования в сетях [1521]. Алиса хочет с помощью Трента, KDC, генерировать ключ для сеанса связи с Бобом. Всем участникам известен открытый ключ Трента

n. Тренту известны два простых множителя n, и, следовательно, он может легко вычислять квадратные корни по модулю n. Следующий протокол не содержит некоторых деталей, но позволяет получить общее представление.

(1) Алиса выбирает случайное число r_A и посылает Тренту

$r_A^3 \bmod n$

(2) Трент сообщает Бобу, что кто-то хочет обменяться с ним ключом.

(3) Боб выбирает случайное число r_B и посылает Тренту

$r_B^3 \bmod n$

(4) Трент, используя свой закрытый ключ, расшифровывает r_A и r_B. Он посылает Алисе

$r_A \oplus r_B$

(5) Алиса вычисляет

$(r_A \oplus r_B) \oplus r_A = r_B$

Она использует r_B для безопасного сеанса связи с Бобом.

Протокол выглядит хорошо, но содержит заметный изъян. Кэрол может подслушать этап(3) и использовать эту информацию, воспользовавшись помощью доверчивого Трента и своего сообщника Дэйва, чтобы раскрыть [1472].

(1) Кэрол выбирает случайное число r_C и посылает Тренту

$r_B^3 r_C^3 \bmod n$

(2) Трент сообщает Дэйву, что кто-то хочет обменяться с ним ключом.

(3) Дэйв выбирает случайное число r_D и посылает Тренту

$r_D^3 \bmod n$

(4) Трент, используя свой закрытый ключ, расшифровывает r_C и r_D. Он посылает Кэрол

$(r_B r_C \bmod n) \oplus r_D$

(5) Дэйв посылает r_D Кэрол.

(6) Кэрол использует r_C и r_D для получения r_B. Она использует r_B для расшифровывания переговоров Алисы и Боба.

Это плохо.

Глава 23
Специальные алгоритмы для протоколов

23.1 Криптография с несколькими открытыми ключами

Это обобщение RSA (см. раздел 19.3) [217, 212]. Модуль n является произведением двух простых чисел p и q. Однако вместо e и d, для которых $ed \equiv 1 \bmod ((p\text{-}1)(q\text{-}1))$, выбирается t ключей K_i, для которых выполняется

$K_1 * K_2 * \ldots * K_t \equiv 1 \bmod ((p\text{-}1)(q\text{-}1))$

Так как

$$M^{K_1 * K_2 * \ldots * K_t} = M$$

то эта схема оказывается схемой с несколькими ключами, описанная в разделе 3.5.

Если, например, используется пять ключей, то сообщение, зашифрованное ключами K_3 и K_5, может быть расшифровано с помощью K_1, K_2 и K_4.

$$C = M^{K_3 * K_5} \bmod n$$

$$M = C^{K_1 * K_2 * K_4} \bmod n$$

Одним из применений этой схемы является подписание документа несколькими людьми. Представим ситуацию, когда для того, чтобы документ был действителен, он должен быть подписан и Алисой, и Бобом. Используются три ключа: K_1, K_2 и K_3. Алиса и Боб получают по одному ключу из первых двух, а третий опубликовывается.

(1) Сначала Алиса подписывает M и посылает его Бобу.

$$M' = M^{K_1} \bmod n$$

(2) Боб может восстановить M по M'.

$$M = M'^{K_3 * K_5} \bmod n$$

(3) Он может также добавить свою подпись.

$$M'' = M'^{K_2} \bmod n$$

(4) Проверить подписи можно при помощи открытого ключа K_3.

$$M = M''^{K_3} \bmod n$$

Обратите внимание, что для работоспособности этой системы нужна заслуживающая доверия сторона, которая установила бы систему и выдала ключи Алисе и Бобу. Та же проблема существует и в схеме [484]. Более тонкая схема описана в [695, 830, 700], Но усилия, предпринимаемые для проверки, пропорциональны количеству подписывающих. Новые схемы [220, 1200], основанные на схемах идентификации с нулевым знанием, преодолевают эти недостатки предшествующих систем.

23.2 Алгоритмы разделения секрета

В разделе 3.7 я рассматривал идею, используемую в схемах разделения секрета. Четыре приведенных ниже различных алгоритма представляют собой частные случаи общего теоретического подхода [883].

Схема интерполяционных многочленов Лагранжа

Для создания пороговой схема Ади Шамир воспользовался уравнениями для многочленов в конечном поле [1414]. Выберем простое число p, которое больше количества возможных теней и больше самого большого из возможных секретов. Чтобы сделать секрет общим, сгенерируем произвольный многочлен степени m-1. Например, если нужно создать пороговую схему (3,n) (для восстановления M потребуется три тени), генерируется квадратичный многочлен

$(ax^2 + bx + M) \bmod p$

где p - это случайное простое число, большее любого из коэффициентов. Коэффициенты a и b выбираются случайным образом, они хранятся в тайне и отбрасываются после того, как распределяются тени. M - это сообщение. Простое число должно быть опубликовано. Тени получаются с помощью вычисления многочлена в n различных точках:

$k_i = F(x_i)$

Другими словами, первой тенью может быть значение многочлена при $x = 1$, второй тенью - значение многочлена при $x = 2$, и т.д.

Так как в квадратичных многочленах три неизвестных коэффициента , a, b и M, для создания трех уравнений можно использовать любые три цели. Одной или двух теней не хватит, а четырех или пяти теней будет много .

Например, пусть M равно 11. Чтобы создать пороговую схему (3, 5), в которой любые трое из пяти человек могут восстановить M, сначала получим квадратичное уравнение (7 и 8 - случайно выбранные числа chosen randomly):

$F(x) = (7x^2 + 5x + 11) \bmod 13$

Пятью тенями являются:

$k_1 = F(1) = 7 + 8 + 11 \equiv 0 \pmod{13}$

$k_2 = F(2) = 28 + 16 + 11 \equiv 3 \pmod{13}$

$k_3 = F(3) = 63 + 24 + 11 \equiv 7 \pmod{13}$

$k_4 = F(4) = 112 + 32 + 11 \equiv 12 \pmod{13}$

$k_5 = F(5) = 175 + 40 + 11 \equiv 5 \pmod{13}$

Чтобы восстановить M по трем теням, например, k_2, k_3 и k_5, решается система линейных уравнений:

$a*2^2 + b*2 + M = 3 \pmod{13}$

$a*3^2 + b*3 + M = 7 \pmod{13}$

$a*5^2 + b*5 + M = 5 \pmod{13}$

Решением будут $a = 7$, $b = 8$ и $M = 11$. Итак, M получено.

Эту схему разделения можно легко реализовать для больших чисел. Если вы хотите разбить сообщение на 30 равных частей так, чтобы восстановить сообщение можно было, объединив любые шесть из них , выдайте каждому из 30 человек значения многочлена пятой степени .

$F(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + M \pmod{p}$

Шесть человек могут шесть неизвестных (включая M), но пятерым не удастся узнать ничего об M.

Наиболее впечатляющим моментом совместного использования секрета является то, что, если коэффициенты выбраны случайным образом, пять человек даже при помощи бесконечных вычислительных мощностей не смогут узнать ничего, кроме длины сообщения (которая и так им известна) . Это также безопасно, как одноразовый блокнот, попытка выполнить исчерпывающий поиск (то есть, перебор всех возможных шестых теней) покажет, что любое возможное сообщение останется секретным . Это справедливо для всех представленных в этой книге схем разделения секрета.

Векторная схема

Джордж Блэкли (George Blakley) изобрел схему, использующую понятие точек в пространстве [182]. Сообщение определяется как точка в m-мерном пространстве. Каждая тень - это уравнение (m-1)-мерной гиперплоскости, содержащей эту точку.

Например, если для восстановления сообщения нужны три тени, то оно является точкой в трехмерном пространстве. Каждая тень представляет собой иную плоскость. Зная одну тень, можно утверждать, что точка находится где-то на плоскости. Зная две тени - что она находится где-то на линии пересечения двух плоскостей . Зная три тени, можно точно определить, что точка находится на пересечении трех плоскостей .

Asmuth-Bloom

В этой схеме используются простые числа [65]. Для (m, n)-пороговой схемы выбирается большое простое число p, большее M. Затем выбираются числа, меньшие p - d_1, d_2, . . . d_n, для которых:

1. Значения d_i упорядочены по возрастанию, $d_i < d_{i+1}$
2. Каждое d_i взаимно просто с любым другим d_i
3. $d_1*d_2* \ldots *d_m > p*d_{n-m+2}*d_{n-m+3}* \ldots *d_n$

Чтобы распределить тени, сначала выбирается случайное число r и вычисляется

$M' = M + rp$

Тенями, k_i, являются

$k_i = M' \bmod d_i$

Объединив любые m теней, можно восстановить M, используя китайскую теорему об остатках, но это невозможно с помощью любых m-1 теней. Подробности приведены в [65].

Karnin-Greene-Hellman

В этой схеме используется матричное умножение [818]. Выбирается n+1 m-мерных векторов, V_0, V_1, . . . V_n, так, что ранг любой матрицы размером m*m, образованной из этих векторов, равен m. Вектор U - это вектор размерности m+1.

M - это матричное произведение $U \cdot V_0$. Тенями являются произведения $U \cdot V_i$, где i меняется от 1 до n.

Любые m теней можно использовать для решения системы линейных уравнений размерности m*m, неизвестными являются коэффициенты U. $U \cdot V_0$ можно вычислить по U. Используя любые m-1 теней, решить систему уравнений и, таким образом, восстановить секрет невозможно.

Более сложные пороговые схемы

В предыдущих примерах показаны только простейшие пороговые схемы : секрет делится на n теней так, чтобы, объединив любые m из них, можно было раскрыть секрет. На базе этих алгоритмов можно создать намного более сложные схемы. В следующих примерах будет использоваться алгоритм Шамира, хотя будут работать и все остальные.

Чтобы создать схему, в которой один из участников важнее других, ему выдается больше теней . Если для восстановления секрета нужно пять теней, и у кого-то есть три тени, а у всех остальных - по одной , этот человек вместе с любыми двумя другими может восстановить секрет . Без его участия для восстановления секрета потребуется пять человек.

По несколько теней могут получить два человека и более . Каждому человеку может быть выдано отличное число теней. Независимо от того, сколько теней было роздано, для восстановления секрета потребуется любые m из них. Ни один человек, ни целая группа не смогут восстановить секрет, обладая только m-1 тенями.

Для других схем представим сценарий с двумя враждебными делегациями . Можно распределить секрет так, чтобы для его восстановления потребовалось двое из 7 участников делегации A и трое из 12 участников делегации B. Создается многочлен степени 3, который является произведением линейного и квадратного выражений . Каждому участнику делегации A выдается тень, которая является значением линейного выражения, а участникам делегации B выдаются значения квадратичного выражения .

Для восстановления линейного выражения достаточны любые две тени участников делегации A, но независимо от того, сколько других теней есть у делегации , ее участники не смогут ничего узнать о секрете . Аналогично для делегации B: ее участники могут сложить три тени, восстанавливая квадратное выражение, но другую информацию, необходимую для восстановления секрета в целом, они получить не смогут . Только перемножив свои выражения, участники двух делегаций смогут восстановить секрет .

В общем случае, может быть реализована любая мыслимая схема разделения секрета . Потребуется только написать систему уравнений, соответствующих конкретной системе. Вот несколько прекрасных статей на тему обобщенных схем разделения секрета [1462, 1463, 1464].

Разделение секрета с мошенниками

Этот алгоритм изменяет стандартную пороговую схему (m, n) для обнаружения мошенников [1529]. Я покажу его использование на базе схемы Лагранжа , но алгоритм работает и с другими схемами . Выбирается простое число p, большее n и большее

$(s - 1)(m - 1)/e + m$

где s - это самый большой возможный секрет, а e - вероятность успеха мошенничества . e можно сделать настолько малым, насколько это необходимо, это просто усложнит вычисления . Постройте тени как раньше, но вместо использования 1, 2, 3, . . . , n для x_i, выберите случайным образом числа из диапазона от 1 до p-1.

Теперь, если Мэллори при восстановлении секрета заменит свою часть подделкой , его тень с высокой вероятностью окажется невозможной . Невозможный секрет, конечно же, окажется подделанным секретом . Математика этой схемы приведена в [1529].

К сожалению, хотя мошенничество Мэллори и будет открыто, ему удастся узнать секрет (при условии, что все остальные нужные тени правильны). От этого защищает другой протокол, описанный в [1529, 975]. Основ-

ной идеей является использование набора из k секретов, так чтобы никто из участников заранее не знал, какой из них правильный. Каждый секрет, за исключением настоящего, больше предыдущего. Участники объединяют свои тени, получая один секрет за другим, пока они не получат наименьшее значение секрета. Этот секрет и будет правильным.

В этой схеме мошенники легко выявляются еще до получения конечного секрета. Существует определенные сложности, если участники предъявляют свои тени по очереди, подробности можно найти в литературе. В следующих работах также рассматриваются обнаружение и предотвращение мошенничества в пороговых схемах [355, 114, 270].

23.3 Подсознательный канал

Ong-Schnorr-Shamir

Этот подсознательный канал (см. раздел 4.2), разработанный Густавусом Симмонсом (Gustavus Simmons) [1458, 1459, 1460], использует схему идентификации Ong-Schnorr-Shamir (см. раздел 20.5). Как и в оригинальной схеме отправитель (Алиса) выбирает общедоступный модуль n и закрытый ключ k так, чтобы n и k были взаимно простыми числами. В отличии от оригинальной схемы k используется совместно Алисой и Бобом, получателем в подсознательном канале. Открытый ключ вычисляется следующим образом:

$h = -k^2 \bmod n$

Если Алисе нужно отправить подсознательное сообщение M в безобидном сообщении M', она сначала проверяет, что пары M' и n, а также M и n являются взаимно простыми числами. Алиса вычисляет

$S_1 = 1/2*((M'/M + M)) \bmod n$

$S_2 = 1/2*((M'/M - M)) \bmod n$

Пара чисел S_1 и S_2 представляет собой подпись в традиционной схеме Ong-Schnortr-Shamir и одновременно является носителем подсознательного сообщения.

Тюремщик Уолтер (помните такого?) может проверить подлинность сообщения, как это принято в Ong-Schnorr-Shamir, но Боб может сделать еще кое-что. Он может проверить подлинность сообщения (Всегда возможно, что Уолтер попытается ему подсунуть поддельное сообщение). Он проверяет, что

$S_1^2 - S_2^2 \equiv M' \pmod n$

Если подлинность сообщения доказана, получатель может извлечь и подсознательное сообщение, используя следующую формулу:

$M=M'/(S_1 + S_2 k^{-1}) \bmod n$

Это работает, но не забывайте, что сама схема Ong-Schnorr-Shamir была взломана.

ElGamal

Другой предложенный Симмонсом подсознательный канал [1459], описанный в [1407, 1473], основан на схеме подписи ElGamal см. раздел 19.6).

Генерация ключа выполняется также, как и в основной схеме подписи ElGamal. Сначала выбирается простое число p и два случайных числа, g и r, меньшие p. Затем вычисляется

$K = g^r \bmod p$

Открытым ключом служат K, g и p. Закрытым ключом является r. Помимо Алисы r известно и Бобу, это число используется не только для подписи безобидного сообщения, но и в качестве ключа для отправки и чт ения подсознательного сообщения.

Чтобы послать подсознательное сообщение M в безобидном сообщении, M', M и p должны быть попарно взаимно простыми, кроме того, взаимно простыми должны быть M и p-1. Алиса вычисляет

$X = g^M \bmod p$

и решает следующее уравнение для Y (с помощью расширенного алгоритма Эвклида):

$M' = rX + MY \bmod (p\text{-}1)$

Как и в базовой схеме ElGamal, подписью является пара чисел: X и Y. Уолтер может проверить подпись ElGamal. Он убеждается, что

$K^X X^Y \equiv g^{M'} \pmod p$

Боб может восстановить подсознательное сообщение. Сначала он убеждается, что

$(g^r)^X X^Y \equiv g^{M'} \pmod p$

Если это так, он считает сообщение подлинным (не подделанным Уолтером). Затем для восстановления M он вычисляет

$M = (Y^{-1} (M' - rX)) \bmod (p - 1)$

Например, пусть $p = 11$, а $g = 2$. Закрытый ключ r выбирается равным 8. Это означает, что открытым ключом, который Уолтер может использовать для проверки подписи, будет $g^r \bmod p = 2^8 \bmod 11 = 3$.

Чтобы отправить подсознательное сообщение $M = 9$, используя безобидное сообщение $M' = 5$, Алиса проверяет, что 9 и 11, а также 5 и 11 попарно взаимно просты. Она также убеждается, что взаимно просты 9 и 11-1=10. Это так, поэтому она вычисляет

$X = g^{M'} \pmod p = 2^9 \bmod 11 = 6$

Затем она решает следующее уравнение для Y:

$5 = 8\ 6 + 9\ Y \bmod 10$

$Y = 3$, поэтому подписью служит пара чисел 6 и 3 (X и Y). Боб убеждается, что

$(g^r)^X X^Y \equiv g^{M'} \pmod p$

$(2^8)^6 6^3 \equiv 2^5 \pmod{11}$

Это так (выполните арифметические действия самостоятельно, если вы мне не верите), поэтому он может раскрыть подсознательное сообщение, вычисляя

$M = (Y^{-1} (M' - rX)) \bmod (p - 1) = 3^{-1}(5 - 8*6) \bmod 10 = 7(7) \bmod 10 = 49 \bmod 10 = 9$

ESIGN

Подсознательный канал можно добавить и к ESIGN [1460] (см. раздел 20.6). В ESIGN секретный ключ является парой больших простых чисел p и q, а открытым ключом служит $n = p^2q$. Использовании подсознательного канала закрытым ключом являются три простых числа p, q и r, а открытым ключом - n, такое что

$n = p^2qr$

Переменная r - это дополнительные данные, нужные Бобу для прочтения подсознательного сообщения .

Чтобы подписать обычное сообщение, Алиса сначала выбирает случайное число x, меньшее pqr, и вычисляет:

w, наименьшее целое, которое больше или равно $(H(m) - x^k \bmod n)/pq$

$s = x + ((w/kx^{k-1} \bmod p)\ pq$

$H(m)$ - это хэш-значение сообщения, а k - параметр безопасности. Подписью является значение s.

Для проверки подписи Боб вычисляет $s^k \bmod n$. Кроме этого, он вычисляет a, наименьшее целое, которое больше или равно удвоенному числу битов n, деленному на 3. Если $H(m)$ меньше или равна $s^k \bmod n$, и если $s^k \bmod n$ меньше $H(m)+2^a$, то подпись считается правильной.

Для отправки подсознательного сообщения M с помощью безобидного сообщения M' Алиса вычисляет s, используя M вместо of $H(m)$. Это означает, что сообщение должно быть меньше, чем p^2qr. Затем она выбирает случайное число u и вычисляет

$x' = M' + ur$

Затем это значение x' используется в качестве "случайного числа" x при подписи M'. Соответствующее значение s посылается в качестве подписи.

Уолтер может проверить, что s (второе s) является правильной подписью M' Точно также проверить подлинность сообщения может и Боб. Но, так как ему известно и r, он может вычислить

$s = x' + ypqr = M + ur + ypqr \equiv M \pmod r$

Эта реализация подсознательного канала намного лучше двух предыдущих. В вариантах Ong-Schnorr-Shamir и ElGamal у Боба должен быть закрытый ключ Алисы. Боб сможет не только читать подсознательные сообщения Алисы, но и выдавать себя за Алису, подписывая обычные документы . Алиса ничего с этим не сможет поделать, устанавливая такой подсознательный канал, ей придется довериться Бобу .

Схема ESICN страдает от этой проблемы. Закрытым ключом Алисы служит набор трех простых чисел: p, q

и r. Секретным ключом Боба является только r. Он знает $n = p^2qr$, но, чтобы раскрыть p и q, ему понадобится разложить на множители это число. Если простые числа достаточно велики, Бобу будет так же трудно выдать себя за Алису, как и Уолтеру или кому-нибудь еще.

DSA

Подсознательный канал существует и в DSA (см. раздел 20.1) [1468, 1469, 1473]. На самом деле их даже может быть несколько. Простейший подсознательный канал включает выбор k. Предполагается, что это будет 160-битовое число. Однако, если Алиса выбирает конкретное k, то Боб, зная закрытый ключ Алисы, сможет раскрыть это k. Алиса посылать Бобу 160-битовое подсознательное сообщение в каждой подписи DSA, а все остальные будут только проверять подпись Алисы. Дополнительное усложнение: Так как k должно быть случайным, Алиса и Боб должны использовать общий одноразовый блокнот и шифровать подсознательное сообщение с помощью этого блокнота, генерируя k.

В DSA есть подсознательные каналы, не требующие передавать Бобу закрытый ключ Алисы. Они также подразумевают выбор конкретных значений k, но не могут передавать по 160 битов информации. Следующая схема, представленная в [1468, 1469], позволяет Алисе и Бобу обмениваться в каждой подписи одним битом подсознательной информации.

(1) Алиса и Боб выбирают случайное простое число P (отличающееся от параметра p в схеме подписи). Это секретный ключ для подсознательного канала.

(2) Алиса подписывает безобидное сообщение M. Если она хочет отправить Бобу подсознательный бит 1, она убеждается, что параметр r подписи является квадратичным остатком по модулю P. Если она хочет отправить ему 0, она проверяет, что параметр r подписи не является квадратичным остатком по модулю P. Она добивается этого, подписывая сообщение с помощью случайных значений k, пока она не получит подпись с нужным ей свойством для r. Так как числа, являющиеся квадратичными остатками и не являющиеся ими, равновероятны, то это не должно быть слишком сложно.

(3) Алиса посылает Бобу подписанное сообщение.

(4) Боб проверяет подпись, убеждаясь в подлинности сообщения. Затем он проверяет, является ли r квадратичным остатком по модулю P и восстанавливает подсознательный бит.

Передача таким образом нескольких битов подразумевает подбор такого r, которое является или не является квадратичным остатком по нескольким модулям. Подробности приведены в [1468, 1469].

Эта схема может быть легко расширена для передачи нескольких подсознательных битов на подпись. Если Алиса и Боб выбирают два случайных числа P и Q, то Алиса может посылать два бита, выбирая случайное k так, чтобы r являлось или не являлось квадратичным остатком $\bmod P$, а также являлось или не являлось квадратичным остатком $\bmod Q$. Случайное значение k с вероятностью 25 процентов позволит получить r с нужными свойствами.

Вот как Мэллори, нечестный реализатор DSA, может создать алгоритм, извлекающий по 10 битов закрытого ключа Алисы из каждой ее подписи.

(1) Мэллори строит свою реализацию DSA базе устойчивой к взлому СБИС, чтобы никто не смог проверить, как она работает. Он создает 14 подсознательных каналов в своей реализации DSA. То есть, он выбирает 14 случайных простых чисел и использует микросхему, которая выбирает значение k так, чтобы r являлось или не являлось квадратичным остатком по модулю каждого из этих 14 простых чисел, в зависимости от подсознательного сообщения.

(2) Мэллори выдает микросхемы Алисе, Бобу и остальным желающим.

(3) Алиса обычным образом подписывает сообщение, используя свой закрытый 160-битовый ключ x.

(4) Микросхема случайным образом выбирает 10-битовый блок x: первые 10 битов, вторые 10 битов, и т.д. Так как существует 16 возможных 10-битовых блоков, то номер блока выражается 4-битовым числом. Этот 4-битовый идентификатор и 10 битов ключа и будут 14-битовым подсознательным сообщением.

(5) Микросхема перебирает случайные значения k, пока не удастся найти то, которое обладает правильными квадратичными остатками, нужными для передачи подсознательного. Вероятность случайного k обладать правильной формой равна 1/16384. Если микросхема может проверить 10000 значений k в секунду, нужное значение будет найдено меньше, чем за пару секунд. Эти вычисления не зависят от сообщения и могут быть вычислены заранее, до того, как Алиса захочет подписать сообщение.

(6) Микросхема обычным образом подписывает сообщение, используя выбранное на этапе (5) значение k.

(7) Алиса посылает цифровую подпись Бобу, или опубликовывает ее в сети, или еще что-нибудь делает.

(8) Мэллори раскрывает r и, так как он знает 14 простых чисел, расшифровывает подсознательное

сообщение.

Страшнее всего, что, даже если Алиса знает, что происходит, она ничего не сможет доказать. Пока 14 простых чисел хранятся в секрете, Мэллори в безопасности.

Уничтожение подсознательного канала в DSA

Подсознательный канал опирается на то, что Алиса может выбирать k для передачи подсознательной информации. Чтобы сделать подсознательный канал невозможным, Алисе не должно быть позволено выбирать k. Однако, выбор k должен быть запрещен и для всех других. Если кому-то другому будет позволено выбирать k, то этот человек получит возможность подделать подпись Алисы. Единственным решением для Алисы является проведение генерации k вместе с другой стороной, Бобом, так, чтобы Алиса не могла контролировать ни один бит k, а Боб не мог определить ни один бит k. На другой стороне протокола у Боба должна быть возможность проверить, что Алиса использовала именно совместно созданное k.

Вот этот протокол [1470, 1472, 1473]

(1) Алиса выбирает k' и посылает Бобу

$u = g^{k'} \bmod p$

(2) Боб выбирает k'' и посылает его Алисе.

(3) Алиса вычисляет $k = k'k'' \bmod (p - 1)$. Она использует k, чтобы подписать свое сообщение M, используя DSA, и посылает Бобу свою подпись: r и s.

(4) Боб проверяет, что $((u = g^{k'} \bmod p) \bmod q) = r$

Если это так, то он знает, что для подписи M использовалось k. После этапа (4) Боб знает, что в r не было включено никакой подсознательной информации. Если он является доверенной стороной, он может проверить, что в подписи Алисы нет подсознательной информации. Другим придется поверить его заявлению, Боб не сможет доказать этот факт третьей стороне, воспроизведя протокол.

Удивительно то, что Боб, если захочет, может использовать этот протокол для создания собственного подсознательного канала. Боб может включить подсознательную информацию в одну из подписей Алисы, выбрав k'' с определенными характеристиками. Когда Симмонс открыл такую возможность, он назвал ее "Каналом кукушки". Подробности работы Канала кукушки, и мешающий этому трехпроходный протокол генерации k, рассматриваются в [1471, 1473].

Другие схемы

Подсознательный канал можно организовать для любой схемы подписи [1458, 1460, 1406]. Описание протокола встраивания подсознательного канала в схемы Fiat-Shamir и Feige-Fiat-Shamir вместе с возможными злоупотреблениями можно найти в [485].

23.4 Неотрицаемые цифровые подписи

Автором этого алгоритма неотрицаемой подписи (см. раздел 4.3) является Дэвид Чаум (David Chaum) [343,327]. Сначала опубликовываются большое простое число p и примитивный элемент g, которые будут совместно использоваться группой подписывающих. У Алисы есть закрытый ключ x и открытый ключ $g^x \bmod p$.

Чтобы подписать сообщение, Алиса вычисляет $z = m^x \bmod p$. Это все, что ей нужно сделать. Проверка подписи немного сложнее.

(1) Боб выбирает два случайных числа, a и b, меньшие p, и отправляет Алисе:

$c = z^a (g^x)^b \bmod p$

(2) Алиса вычисляет $t = x^{-1} \bmod (p-1)$, и отправляет Бобу:

$d = c^t \bmod p$

(3) Боб проверяет, что

$d \equiv m^a g^b \pmod p$

Если это так, он считает подпись истинной.

Представим, что Алиса и Боб выполнили этот протокол, и Боб теперь считает, что Алиса подписала сообщение. Боб хочет убедить в этом Кэрол, поэтому он показывает ей запись протокола. Дэйв, однако, хочет убедить Кэрол, что документ подписан кем-то другим. Он создает поддельную запись протокола. Сначала он генерирует сообщение на этапе (1). Затем на этапе (3) он генерирует d и ложную передачу от другого человека на этапе (2).

Наконец, он создает сообщение этапа (2). Для Кэрол записи Боба и Дэйва одинаковы. Ее невозможно убедить в правильности подписи, пока она не выполнит протокол самостоятельно .

Конечно, если бы она следила из-за плеча Боба за тем, как он выполняет протокол, она была бы убеждена . Кэрол нужно увидеть выполнение этапов по порядку, так, как это делал Боб .

Используя эту схему подписи, можно столкнуться с проблемой, но я не знаю подробностей . Прежде, чем воспользоваться этой схемой, просмотрите литературу .

Другой протокол включает не только протокол подтверждения - Алиса может убедить Боба в правильности своей подписи - но и протокол отрицания. Алиса может с помощью интерактивного протокола с нулевым знанием убедить Боба, что ее подпись неправильна, если это так [329].

Как и предыдущий протокол группа подписывающих использует общедоступное большое простое число p и примитивный элемент g. У Алисы есть закрытый ключ x и открытый ключ $g^x \bmod p$. Чтобы подписать сообщение, Алиса вычисляет $z = m^x \bmod p$. Чтобы проверить подпись:

(1) Боб выбирает два случайных числа, a и b, меньшие p, и отправляет Алисе:

$c = m^a g^b \bmod p$

(2) Алиса выбирает случайное число q, меньшее p, а затем вычисляет и отправляет Бобу:

$s_1 = cg^q \bmod p$, $s_2 = (cg^q)^x \bmod p$

(3) Боб посылает Алисе a и b, чтобы Алиса могла убедиться, что Боб не мошенничал на этапе (1).

(4) Алиса посылает Бобу q, чтобы он мо воспользоваться m^x и восстановить s_1 и s_2. Если

$s_1 \equiv cg^q \bmod p$

$s_2 \equiv (g^x)^{b+q} z^a \pmod p$

то подпись правильна.

Алиса может также отказаться от подписи z под сообщением m. Подробности приведены в [329]. Дополнительные протоколы для неотрицаемых подписей можно найти в [584, 344]. Лейн Харн (Lein Harn) и Шубао Янг (Shoubao Yang) предложили схему групповых неотрицаемых подписей [700].

Преобразуемые неотрицаемые подписи

Алгоритм для **преобразуемых неотрицаемых подписей**, которые можно проверять, отменять и преобразовывать в обычные неотрицаемые подписи, приведен в [213]. Он основан на алгоритме цифровых подписей ElGamal.

Как и в ElGamal, сначала выбираются два простых числа, p и q, так, чтобы q было делителем p-1. Теперь нужно создать число g, меньшее q. В диапазоне от 2 до p-1 выбирается случайное число h и вычисляется

$g=h^{(p-1)/q} \bmod p$

Если g равно 1, выбирается другое случайное h. Если нет, используется полученное значение g.

Закрытыми ключами служат два различных случайных числа, x и z, меньшие q. Открытыми ключами являются p, q, g, y и u, где

$y = g^x \bmod p$

$u=g^я \bmod p$

Для вычисления преобразуемой неотрицаемой подписи сообщения m (которое в действительности является хэш-значением сообщения), сначала диапазоне от 1 до q-1 выбирается случайное число t. Затем вычисляется

$T = g^r \bmod p$

и

$m' = Ttzm \bmod q$.

Теперь вычисляется обычная подпись ElGamal для m'. Выбирается случайное число R, меньшее p-1 и взаимно простое с ним. Затем вычисляется $r = g^R \bmod p$ и, с помощью расширенного алгоритма Эвклида, вычисляется s, для которого

$m' \equiv rx + Rs \pmod q$

Подписью служат подпись ElGamal (r, s) и T. Вот как Алиса подтверждает свою подпись Бобу:

(1) Боб генерирует два случайных числа, a и b, и вычисляет $c = T^{Tma}g^b \mod p$ и посылает результат Алисе.

(2) Алиса генерирует случайное число k и вычисляет $h_1 = cg^k \mod p$ и $h_2 = h_1^z \mod p$, а затем посылает оба числа Бобу.

(3) Боб посылает Алисе a и b.

(4) Алиса проверяет, что $c = T^{Tma}g^b \mod p$. Она посылает k Бобу.

(5) Боб проверяет, что $h_1 = T^{Tma}g^{b+k} \mod p$, и что $h_2 = y^{ra}r^{sa}u^{b+k} \mod p$.

Алиса может преобразовать все свои неотрицаемые подписи в обычные, опубликовав z. Теперь любой может проверить ее подпись без ее помощи.

Схемы неотрицаемых подписей можно объединить со схемами разделения секрета, создав **распределенные преобразуемые неотрицаемые подписи** [1235]. Кто-нибудь может подписать сообщение, а затем распределить возможность подтверждения правильности подписи. Он может, например, потребовать, чтобы в протоколе убеждения Боба в правильности подписи участвовали трое из пяти обладателей возможность подтверждения правильности. В [700, 1369] предложены улучшения, позволяющие отказаться от необходимости доверенного лица - распределителя.

23.5 Подписи, подтверждаемые доверенным лицом

Вот как Алиса может подписать сообщение, а Боб проверить его так, чтобы и Кэрол немного позже могла доказать Дэйву правильность подписи Алисы (см. раздел 4.4) [333].

Сначала опубликовываются большое простое число p и примитивный элемент g, которые будут совместно использоваться группой пользователей. Также опубликовывается n, произведение двух простых чисел. У Кэрол есть закрытый ключ z и открытый ключ $h = g^x \mod p$.

В этом протоколе Алиса может подписать m так, чтобы Боб мог проверить правильность ее подписи, но не мог убедить в этом третью сторону.

(1) Алиса выбирает случайное x и вычисляет

$a = g^x \mod p$

$b = h^x \mod p$

Она вычисляет хэш-значение m, $H(m)$, и хэш-значение объединения a и b, $H(a,b)$, а затем

$j = (H(m) \oplus H(a,b))^{1/3} \mod n$

и посылает a, b и j Бобу.

(2) Боб выбирает два случайных числа, s и t, меньших p, и посылает Алисе

$c = g^s h^t \mod p$

(3) Алиса выбирает случайное q, меньшее p, и посылает Бобу

$d = g^q \mod p$

$e = (cd)^x \mod p$

(4) Боб посылает Алисе s и t.

(5) Алиса проверяет, что

$g^s h^t \equiv c \pmod p$

затем она посылает Бобу q.

(6) Боб проверяет

$d \equiv g^q \mod p$

$e/a^q \equiv a^s b^t \pmod p$

$(H(m) \oplus H(a,b)) = j^{1/3} \mod n$

Если все тождества выполняются, то Боб считает подпись истинной.

Боб не может использовать запись этого доказательства для убеждения Дэйва в истинности подписи, но Дэйв может выполнить протокол с доверенным лицом Алисы, Кэрол. Вот как Кэрол убеждает Дэйва в том, что a и b образуют правильную подпись.

(1) Дэйв выбирает случайные u и v, меньшие p, и посылает Кэрол

$k = g^u a^v \bmod p$

(2) Кэрол выбирает случайное w, , меньшее p, и посылает Дэйву

$l = g^w \bmod p$

$y = (kl)^z \bmod p$

(3) Дэйв посылает Кэрол u и v.

(4) Кэрол проверяет, что

$g^u a^v \equiv k \pmod p$

Затем она посылает Дэйву w.

(5) Дэйв проверяет, что

$g^w \equiv l \pmod p$

$y/h^w \equiv h^u b^v \pmod p$

Если все тождества выполняются, то Дэйв считает подпись истинной.

В другом протоколе Кэрол может преобразовать протокол доверенного лица в обычную цифровую подпись. Подробности в [333].

23.6 Вычисления с зашифрованными данными

Проблема дискретного логарифма

Существует большое простое число p и генератор g. Алиса хочет для конкретного x найти такое e, для которого

$g^e \equiv x \pmod p$

Это трудная проблема, и Алисе не хватает вычислительных мощностей для вычисления результата. У Боба есть такие возможности - он представляет правительство, или мощный вычислительный центр, или еще какую-нибудь влиятельную организацию. Вот как Алиса может получить помощь Боба, не раскрыв ему x [547, 4]:

(1) Алиса выбирает случайное число r, меньшее p.

(2) Алиса вычисляет

$x' = xg^r \bmod p$

(3) Алиса просит Боба решить

$g^{e'} \equiv x' \pmod p$

(4) Боб вычисляет e' и посылает его Алисе.

(5) Алиса восстанавливает e, вычисляя

$e = (e' - r) \bmod (p - 1)$

Аналогичные протоколы для проблем квадратичных остатков и примитивных корней приведены в [3, 4]. (См. также раздел 4.8.)

23.7 Бросание "честной" монеты

Следующие протоколы позволяют Алисе и Бобу бросать честную монету в сети передачи данных (см. раздел 4.9) [194]. Это пример бросания монеты в колодец (см. раздел 4.10). Сначала только Боб узнает результат броска и сообщает его Алисе. Затем Алиса может проверить, что Боб сообщил правильный результат броска.

Бросание "честной" монеты с помощью квадратных корней

Подпротокол бросания честной монеты:

(1) Алиса выбирает два больших простых числа, p и q, и посылает их произведение n Бобу.

(2) Боб выбирает случайное положительное целое число r, меньшее $n/2$. Боб вычисляет

$z = r^2 \bmod n$

и посылает z Алисе.

(3) Алиса вычисляет четыре квадратных корня z (mod n). Она может сделать это, так как она знает разложение n на множители. Назовем их $+x$, $-x$, $+y$ и $-y$. Обозначим как x' меньшее из следующих двух чисел:

x mod n

$-x$ mod n

Аналогично, обозначим как y' меньшее из следующих двух чисел:

y mod n

$-y$ mod n

Обратите внимание, что r равно либо x', либо y'.

(4) Алиса делает пытается угадать, какое из значений равно $r - x'$ или y', и посылает свою догадку Бобу.

(5) Если догадка Алисы правильна, результатом броска монеты является "орел", а если неправильна - "решка". Боб объявляет результат броска монеты.

Подпротокол проверки:

(6) Алиса посылает p и q Бобу.

(7) Боб вычисляет x' и y' и посылает их Алисе.

(8) Алиса вычисляет r.

У Алисы нет возможности узнать r, поэтому она действительно угадывает. Она на этапе (4) сообщает Бобу только один бит своей догадки, не давая Бобу получить и x', и y'. Если Боб получит оба этих числа, он сможет изменить r после этапа (4).

Бросание "честной" монеты с помощью возведения в степень по модулю p

В этом протоколе в качестве однонаправленной функции используется возведение в степень по модулю простого числа p [1306]:

Подпротокол бросания честной монеты:

(1) Алиса выбирает простое число p так, чтобы множители p-1 были известны, и среди них было по крайней мере одно большое простое число.

(2) Боб выбирает два примитивных элемента, h и t, в GF(p). Он посылает их Алисе.

(3) Алиса убеждается, что h и t являются примитивными элементами, и затем выбирает случайное число x, взаимно простое с p-1. Затем она вычисляет одно из двух значений:

$y = h^x \bmod p$, или $y = t^x \bmod p$

Она посылает y Бобу.

(4) Боб пытается угадать, вычислила Алиса y как функцию h или как функцию t, и посылает свое предположение Алисе.

(5) Если догадка Боба правильна, результатом бросания монеты является "орел", в противном случае - "решка". Алиса объявляет результат броска монеты.

Подпротокол проверки:

(6) Алиса раскрывает Бобу значение x. Боб вычисляет $h^x \bmod p$ и $t^x \bmod p$, убеждаясь, что Алиса играла честно и проверяя результат броска. Он также проверяет, что x и p-1 - взаимно простые числа.

Чтобы Алиса могла смошенничать, она должна знать два целых числа, x и x', для которых выполняется $h^x \equiv t^{x'} \bmod p$. Для того, чтобы узнать эти значения, ей нужно вычислить:

$\log_t h = x'x^{-1} \bmod p\text{-}1$ и $\log_t h = xx'^{-1} \bmod p\text{-}1$.

Это трудные проблемы.

Алиса смогла бы сделать это, если бы она знала $\log_t h$, но Боб выбирает h и t на этапе (2). У Алисы нет другого способа кроме, как попытаться вычислить дискретный логарифм. Алиса может также попытаться смошенничать, выбрав x, которое не является взаимно простым с p-1, но Боб обнаружит это на этапе (6).

Боб может смошенничать, если h и t не являются примитивными элементами в поле in GF(p), но Алиса сможет легко проверить это после этапа (2), так как ей известно разложение p-1 на простые множители.

Удачным в этом протоколе является то, что если Алиса и Боб захотят бросить несколько монет, он7и смогут использовать одни и те же значения p, h и t. Алиса просто генерирует новое x, и протокол продолжается с этапа (3).

Бросание "честной" монеты с помощью целых чисел Блюма

В протоколе бросания монеты можно использовать челые числа Блюма .

(1) Алиса генерирует целое число Блюма n, случайное x, взаимно простое с n, $x_0 = x^2 \bmod n$ и $x_1 = x_0^2 \bmod n$. Она посылает Бобу n и x_1.

(2) Боб угадывает, четным или нечетным является x_0.

(3) Алиса посылает x Бобу.

(4) Боб проверяет, что n является целым числом Блюма (Алиса нужно передать Бобу множители n и доказательства того, что они являются простыми, или выполнить некоторый протокол с нулевым знанием, убеждающий Боба, что n - это целое число Блюма), и что $x_0 = x^2 \bmod n$ и $x_1 = x_0^2 \bmod n$. Если все проверки выполняются, и Боб угадал правильно, он выигрывает бросок .

Это важно, чтобы n было числом Блюма. Иначе Алиса сможет найти такое x'_0, что $x'^2_0 \bmod n = x_0^2 \bmod n = x_1$, где x'_0 также является квадратичным остатком. Если бы x_0 был четным, а x'_0 - нечетным (или наоборот), Алиса могла бы мошенничать.

23.8 Однонаправленные сумматоры

Существует простая функция однонаправленного сумматоры [116] (см. раздел 4.12.):

$A(x_i, y) = x_{i-1}^y \bmod n$

Числа n (являющееся произведением двух простых чисел) и x_0 должны быть заранее согласованы. Тогда суммированием y_1, y_2 и y_3 будет

$$((x_0^{y_q} \bmod n)^{y_2} \bmod n)^{y_3} \bmod n$$

Это вычисление не зависит от порядка y_1, y_2 и y_3.

23.9 Раскрытие секретов "все или ничего"

Этот протокол позволяет нескольким сторонам (для работы протокола нужно не меньше двух участников) покупать различные секреты у одного продавца (см. раздел 4.13) [1374, 1175]. Начнем с определения. Возьмем две строки битов, x и y. Фиксированным битовым индексом (fixed bit index, **FBI**) x и y называется последовательность номеров совпадающих битов этих строк.

Например:

$x = 110101001011$

$y = 101010000110$

$FBI(x, y) = \{1, 4, 5, 11\}$

(Мы читаем биты справа налево, считая нулевым крайний правый бит .)

Теперь вот как выглядит протокол. Алиса будет продавцом. Боб и Кэрол - покупателями. У Алисы есть k n-битовых секретов: $S_1, S_2, \ldots S_k$. Боб хочет купить секрет S_b, Кэрол - секрет S_c.

(1) Алиса генерирует пару "открытый ключ/закрытый ключ"и сообщает Бобу (но не Кэрол) открытый ключ. Она генерирует другую пару "открытый ключ/закрытый ключ"и сообщает Кэрол (но не Бобу) открытый ключ.

(2) Боб генерирует k n-битовых случайных чисел, $B_1, B_2, \ldots B_k$, и сообщает их Кэрол. Кэрол генерирует k n-битовых случайных чисел, $C_1, C_2, \ldots C_k$, и сообщает их Бобу.

(3) Боб шифрует C_b (напомним, он хочет купить секрет S_b) открытым ключом, полученным от Алисы. Он вычисляет FBI для C_b и только что зашифрованного результата. Он посылает этот FBI Кэрол.

Кэрол шифрует B_c (напомним, она хочет купить секрет S_c) открытым ключом, полученным от Алисы. Она вычисляет FBI для B_c и только что зашифрованного результата. Она посылает этот FBI Бобу.

(4) Боб берет каждое из n-битовых чисел $B_1, B_2, \ldots B_k$ и заменяет каждый бит, номера которого нет в FBI, полученном от Кэрол, его дополнением. Он посылает этот новый список n-битовых чисел $B'_1, B'_2, \ldots B'_k$

Алисе.

Кэрол берет каждое из n-битовых чисел $C_1, C_2, \ldots C_k$ и заменяет каждый бит, номера которого нет в FBI, полученном от Боба, его дополнением. Она посылает этот новый список n-битовых чисел $C'_1, C'_2, \ldots C'_k$ Алисе.

(5) Алиса расшифровывает все C'_i закрытым ключом Боба, получая k n-битовых чисел $C''_1, C''_2, \ldots C''_k$. Она вычисляет $S_i \oplus C''_i$ для $i = 1, \ldots k$, и посылает результаты Бобу.

Алиса расшифровывает все B'_i закрытым ключом Кэрол, получая k n-битовых чисел $B''_1, B''_2, \ldots B''_k$. Она вычисляет $S_i \oplus B''_i$ для $i = 1, \ldots k$, и посылает результаты Кэрол.

(6) Боб вычисляет S_b, выполняя XOR C_b и b-го числа, полученного от Алисы.

Кэрол вычисляет S_c, выполняя XOR B_c и c-го числа, полученного от Алисы..

Все так сложно. Поясним эти долгие действия на примере.

У Алисы есть для продажи восемь 12-битовых секретов: $S_1 = 1990$, $S_2 = 471$, $S_3 = 3860$, $S_4 = 1487$, $S_5 = 2235$, $S_6 = 3751$, $S_7 = 2546$ и $S_8 = 4043$. Боб хочет купить S_7, а Кэрол - S_2.

(1) Алиса использует алгоритм RSA. В диалоге с Бобом она использует следующую пару ключей: $n = 7387$, $e = 5145$ и $d = 777$, а в диалоге с Кэрол - $n = 2747$, $e = 1421$ и $d = 2261$. Она сообщает Бобу и Кэрол их открытые ключи.

(2) Боб генерирует восемь 12-битовых чисел, $B_1 = 743$, $B_2 = 1988$, $B_3 = 4001$, $B_4 = 2942$, $B_5 = 3421$, $B_6 = 2210$, $B_7 = 2306$ и $B_8 = 222$, и сообщает их Кэрол. Кэрол генерирует восемь 12-битовых чисел, $C_1 = 1708$, $C_2 = 711$, $C_3 = 1969$, $C_4 = 3112$, $C_5 = 4014$, $C_6 = 2308$, $C_7 = 2212$ и $C_8 = 222$, и сообщает их Бобу.

(3) Боб хочет купить S_7, поэтому он открытым ключом, выданным Алисой, шифрует C_7.

$2212^{5145} \bmod 7387 = 5928$

Теперь:

2212 = 0100010100100

5928 = 1011100101000

Следовательно, FBI этих двух чисел равен {0, 1, 4, 5, 6}. Он посылает его Кэрол.

Кэрол хочет купить S_2, поэтому она открытым ключом, выданным Алисой, шифрует B_2 и вычисляет FBI B_2 и результата шифрования. Она посылает Бобу {0, 1, 2, 6, 9, 10}.

(4) Боб берет $B_1, B_2, \ldots B_8$ и заменяет каждый бит, индекс которого отсутствует в наборе {0, 1, 2, 6, 9, 10} его дополнением. Например:

B_2 = 111111000100 = 1988

B'_2 = 011001111100 = 1660

Он посылает $B'_1, B'_2, \ldots B'_8$ Алисе.

Кэрол берет $C_1, C_2, \ldots C_8$ и заменяет каждый бит, индекс которого отсутствует в наборе {0, 1, 4, 5, 6} его дополнением. Например:

C_7 = 0100010100100 = 2212

C'_7 = 1011100101000 = 5928

Она посылает $C'_1, C'_2, \ldots C'_8$ Алисе.

(5) Алиса расшифровывает все C'_i закрытым ключом Боба и выполняет XOR результатов с S_i. Например, для $i = 7$:

$5928^{777} \bmod 7387 = 2212$; $2546 \oplus 2212 = 342$

Она посылает результат Бобу.

Алиса расшифровывает все B'_i закрытым ключом Кэрол и выполняет XOR результатов с S_i. Например, для $i = 2$:

$1660^{2261} \pmod{2747} = 1988$; $471 \oplus 1988 = 1555$

Она посылает результат Кэрол.

(6) Боб вычисляет S_7, выполняя XOR C_7 и седьмого числа, полученного им от Алисы:

2212 ⊕ 342=2546

Кэрол вычисляет S_2, выполняя XOR B_2 и второго числа, полученного ей от Алисы.

1988 ⊕ 1555 = 471

Протокол работает для любого количества покупателей. Если Боб, Кэрол и Дэйв хотят купить секреты, Алиса выдает каждому покупателю два открытых ключа, по одному на каждого другого покупателя. Каждый покупатель получает набор чисел от каждого другого покупателя. Затем они выполняют протокол с Алисой для каждого из своих наборов номеров и выполняют XOR всех полученных от Алисы результатов, получая свои секреты. Более подробно это описано в [1374, 1175].

К сожалению, пара нечестных участников могут смошенничать. Алиса и Кэрол, действуя на пару, могут легко понять, какой секрет получил Боб: если они знают FBI C_b и алгоритм шифрования Боба, они могут подыскать такое b, что у C_b будет правильный FBI. А Боб и Кэрол, действуя вместе, могут легко заполучить все секреты Алисы.

Если вы считаете, что участники честны, можно использовать протокол попроще [389].

(1) Алиса шифрует все секреты RSA и посылает их Бобу:

$C_i = S_i^e \bmod n$

(2) Боб выбирает свой секрет C_b, генерирует случайное число r и посылает Алисе.

$C' = C_b r^e \bmod n$

(3) Алиса посылает Бобу^

$P' = C'^d \bmod n$

(4) Боб вычисляет P'

$S_b = P' r^{-1} \bmod n$

Если участники могут жульничать, Боб может доказать с нулевым знанием, что он знает некоторое r, такое что $C' = C_b r^e \bmod n$, и хранить в b секрете, пока Алиса не передаст ему на этапе (3) P' [246].

23.10 Честные и отказоустойчивые криптосистемы

Честная схема Diffie-Hellman

Честные криптосистемы представляют собой программный способ условного вручения документов (см. раздел 4.14). Этот пример взят из работ Сильвии Микали (Silvia Micali) [1084, 1085]. Он запатентован [1086, 1087].

В базовой схеме Diffie-Hellman группа пользователей использует общее простое число p и генератор g. Закрытым ключом Алисы является s, а ее открытым ключом $t = g^s \bmod p$. Вот как сделать схему Diffie-Hellman честной (в этом примере используется пять доверенных лиц).

(1) Алиса выбирает пять целых чисел, s_1, s_2, s_3, s_4, s_5, меньших p-1. Закрытым ключом Алисы является

$s = (s_1 + s_2 + s_3 + s_4 + s_5) \bmod p\text{-}1$

а ее открытым ключом

$t = g^s \bmod p$

Алиса также вычисляет

$t_i = g^{s_i} \bmod p$, для $i = 1, \ldots 5$.

Открытыми частями Алисы являются t_i, а закрытыми - s_i.

(2) Алиса посылает закрытую и соответствующую открытую части каждому доверенному лицу. Например, она посылает s_1 и t_2 доверенному лицу 1. Она посылает t в KDC.

(3) Каждое доверенное лицо проверяет, что

$t_i = g^{s_i} \bmod p$

Если это так, доверенное лицо подписывает t_i и посылает его в KDC. Доверенное лицо сохраняет s_i в безопасном месте.

(4) Получив все пять открытых частей, KDC проверяет, что

$t=(t_1 * t_2 * t_3 * t_4 * t_5) \bmod p$

Если это так, KDC признает открытый ключ.

В этот момент KDC знает, что у каждого доверенного лица есть правильная часть, и что они при необходимости смогут восстановить закрытый ключ. Однако ни KDC, ни любые четыре доверенных лица не могут восстановить закрытый ключ Алисы.

Работы Микали [1084, 1085] также содержат последовательность действия для создания честного RSA и для объединения пороговой схемы с честной криптосистемой, позволяющей m доверенным лицам из n восстановить закрытый ключ.

Отказоустойчивая схема Diffie-Hellman

Как и в предыдущем протоколе у группы пользователей есть общие простое число p и генератор g. Закрытым ключом Алисы является s, а ее открытым ключом $t = g^s \bmod p$.

(1) KDC выбирает случайное число B из диапазона от 0 до p-2 и вручает B с помощью протокола вручения битов (см. раздел 4.9).

Алиса выбирает случайное число A из диапазона от 0 до p-2. Она посылает KDC $g^A \bmod p$.

(2) Пользователь "разделяет" A с каждым доверенным лицом, используя схему подтверждаемого совместного использования секрета (см. раздел 3.7).

(3) KDC раскрывает B Алисе.

(4) Алиса проверяет вручение этапа (1). Затем она устанавливает свой открытый ключ равным

$t = g^A g^B \bmod p$

а закрытый ключ равным

$s = (A + B) \bmod (p\text{-}1)$

Доверенные лица могут восстановить A. Так как KDC знает B, этого достаточно для восстановления s. И Алиса не сможет использовать никаких подсознательных каналов для передачи несанкционированной информации. Этот протокол, рассмотренный в [946, 833] в настоящее время патентуется.

23.11 ZERO-KNOWLEDGE PROOFS OF KNOWLEDGE

Доказательство с нулевым знанием для дискретного логарифма

Пегги хочет доказать Виктору, что ей известно x, являющееся решением

$A^x \equiv B \pmod p$

где p - простое число, а x - произвольное число, взаимно простое с p-1. Числа A, B и p общедоступны, а x хранится в секрете. Вот как Пегги, не раскрывая значения x, может доказать, что оно ей известно (см. раздел 5.1) [338, 337].

(1) Пегги генерирует t случайных чисел, $r_1, r_2, \ldots r_t$, причем все r_i меньше p-1.

(2) Пегги вычисляет $h_i = A^{r_i} \bmod p$ для всех значений i и посылает их Виктору.

(3) Пегги и Виктор, воспользовавшись протоколом бросания монеты генерируют t битов: $b_1, b_2, \ldots b_t$.

(4) Для всех t битов Пегги выполняет одну из следующих операций:

a) Если $b_i = 0$, она посылает Виктору r_i

b) Если $b_i = 1$, она посылает Виктору $s_i = (r_i - r_j) \bmod (p\text{-}1)$, где j - наименьшее значение индекса, при котором $b_j = 1$

(5) Для всех t битов Виктор проверяет одно из следующих условий:

a) При $b_i = 0$ что $A^{r_i} \equiv h_i \pmod p$

b) При $b_i = 1$ что $A^{s_i} \equiv h_i h_j^{-1} \pmod p$

(6) Пегги посылает Виктору Z, где

$Z = (x - r_j) \bmod (p\text{-}1)$

(7) Виктор проверяет, что $A^Z \equiv B h_j^{-1} \pmod p$

Вероятность удачного мошенничества Пегги равна $1/2^t$.

Доказательство с нулевым знанием для возможности вскрыть RSA

Алиса знает закрытый ключ Кэрол. Может быть она взломала RSA, а может она взломала дверь квартиры Кэрол и выкрала ключ. Алиса хочет убедить Боба, что ей известен ключ Кэрол. Однако она не хочет ни сообщать Бобу ключ, ни даже расшифровать для Боба одно из сообщений Кэрол. Далее приведен протокол с нулевым знанием, с помощью которого Алиса убеждает Боба, что она знает закрытый ключ Кэрол [888]. Пусть открытый ключ Кэрол - e, ее закрытый ключ - d, а модуль RSA - n.

(1) Алиса и Боб выбирают случайное k и m, для которых

$km \equiv e \pmod{n}$

Числа они должны выбирать случайным образом, используя для генерации k протокол бросания монеты, а затем вычисляя m. Если и k, и m больше 3, протокол продолжается. В противном случае числа выбираются заново.

(2) Алиса и Боб генерируют случайный шифротекст C. И снова они должны воспользоваться протоколом бросания монеты.

(3) Алиса, используя закрытый ключ Кэрол, вычисляет

$M = C^d \bmod n$

Затем она вычисляет

$X = M^k \bmod n$

и посылает X Бобу.

(4) Боб проверяет, что $X^m \bmod n = C$. Если это так, то он убеждается в правильности заявления Алисы.

Аналогичный протокол можно использовать для демонстрации возможности вскрытия проблемы дискретного логарифма [888].

Доказательство с нулевым знанием того, что n является числом Блюма

Пока неизвестно никаких действительно практичных доказательств того, что $n = pq$, где p и q - простые числа, конгруэнтные 3 по модулю 4. Однако если n имеет форму $p^r q^s$, где r и s нечетны, то у числа n сохраняются свойства, которые делают числа Блюма полезными для криптографии. И тогда существует доказательство с нулевым знанием того, что n имеет такую форму.

Предположим, что Алисе известно разложение на множители числа Блюма n, где n обладает рассмотренной выше формой. Вот как она может доказать Бобу, что n имеет такую форму [660].

(1) Алиса посылает Бобу число u, чей символ Якоби равен -1 по модулю n.

(2) Алиса и Боб совместно выбирают случайные биты: $b_1, b_2, \ldots b_k$.

(3) Алиса и Боб совместно выбирают случайные числа: $x_1, x_2, \ldots x_k$.

(4) Для каждого $i = 1, 2, \ldots k$ Алиса посылает Бобу квадратный корень по модулю n для одного из четырех чисел: x_i, $-x_i$, ux_i, $-ux_i$. Символ Якоби квадратного корня должен быть равен b_i.

Вероятность удачного мошенничества Алисы равна $1/2^k$.

23.12 Слепые подписи

Понятие слепых подписей (см. раздел 5.3) было придумано Дэвидом Чаумом (David Chaum) [317, 323], который также предложил и первую реализацию этого понятия [318]. Она использует алгоритм RSA.

У Боба есть открытый ключ e, закрытый ключ d и открытый модуль n. Алиса хочет, чтобы Боб вслепую, не читая, подписал сообщение m.

(1) Алиса выбирает случайное число k из диапазона от 1 до n. Затем она маскирует m, вычисляя

$t = mk^e \bmod n$

(2) Боб подписывает t

$t^d = (mk^e)^d \bmod n$

(3) Алиса снимает маскировку с t^d, вычисляя

$s = t^d/k \bmod n$

(4) Результатом является

$s = m^d \bmod n$

Это можно легко показать

$t^d \equiv (mk^e)^d \equiv m^d k \pmod n$, поэтому $t^d/k = m^d k/k \equiv m^d \pmod n$.

Чаум придумал целое семейство более сложных алгоритмов слепой подписи [320, 324], называемых неожиданными слепыми подписями. Схемы этих подписей сложнее, но они дают больше возможностей .

23.13 Передача с забыванием

В этом протоколе, предложенном Майклом Рабином (Michael Rabin) [1286], Алиса с вероятностью 50 процентов удается передать Бобу два простых числа, p и q. Алиса не знает, успешно ли прошла передача (См. раздел 5.5.) (Этот протокол можно использовать для передачи Бобу любого сообщения с 50-процентной вероятностью успешной передачи, если p и q раскрывают закрытый ключ RSA.)

(1) Алиса посылает Бобу произведение двух простых чисел: $n = pq$.

(2) Боб выбирает случайное число x, меньшее n и взаимно простое с n. Он посылает Алисе:

$a = x^2 \bmod n$

(3) Алиса, зная p и q, вычисляет четыре квадратных корня a: x, n-x, y и n-y. Она случайным образом выбирает любой из этих корней и посылает его Бобу.

(4) Если Боб получает y или n-y, он может вычислит наибольший общий делитель x+y и n, которым будет либо p, либо q. Затем, конечно же, $n/p = q$. Если Боб получает x или n-x, он не может ничего вычислить.

У этого протокола может быть слабое место: возможна ситуация, когда Боб может вычислить такое число a, что при известном квадратном корне a он сможет все время раскладывать n на множители.

23.14 Безопасные вычисления с несколькими участниками

Этот протокол взят из [1373]. Алиса знает целое число i, а Боб - целое число j. Алиса и Боб вместе хотят узнать, что правильно - $i \leq j$ или $i > j$, но ни Алиса, ни Боб не хочет раскрыть свое число партнеру. Этот особый случай безопасных вычислений с несколькими участниками (см. раздел 6.2) иногда называют **проблемой миллионера Яо** [162, 7].

В приводимом примере предполагается, что i и j выбираются из диапазона от 1 до 100. У Боба есть открытый и закрытый ключи.

(1) Алиса выбирает большое случайное число x и шифрует его открытым ключом Боба.

$c = E_B(x)$

(2) Алиса вычисляет c-j и посылает результат Бобу.

(3) Боб вычисляет следующие 100 чисел:

$y_u = D_B(c\text{-}i\text{+}u)$, для $1 \leq u \leq 100$

D_B обозначает дешифрирование закрытым ключом Боба.

Он выбирает большое случайное число p. (Размер p должен быть немного меньше x. Боб не знает x, но Алиса может легко сообщить ему размер x.) он вычисляет следующие 100 чисел:

$z_u = (y_u \bmod p)$, для $1 \leq u \leq 100$

Далее он проверяет, что для всех $u \neq v$

$|z_u - z_м| \geq 2$

и что для всех u

$0 < z_u < p\text{-}1$

Если это не так, то Боб выбирает другое простое число и пробует снова.

(4) Боб посылает Алисе эту последовательность чисел, соблюдая их точный порядок:

$z_1, z_2, \ldots z_j, z_{j+1}+1, z_{j+2}+1, \ldots z_{100}+1, p$

(5) Алиса проверяет, конгруэнтен ли i-ый член последовательности $x \bmod p$. Если это так, она делает вывод, что $i \leq j$. В противном случае она решает, что $i > j$.

(6) Алиса сообщает Бобу свои выводы.

Проверка, которую Боб выполняет на этапе (3), должна гарантировать, что ни одно число не появится дважды в последовательности, генерированной на этапе (4). В противном случае, если $z_a = z_b$, Алиса узнает, что $a \leq j < b$.

Недостатком этого протокола является то, что Алиса узнает результаты вычислений раньше Боба. Ничто не помешает ей завершить протокол на этапе (5), отказавшись сообщать Бобу результаты. Она даже может солгать Бобу на этапе (6).

Пример протокола

Пусть они используют RSA. Открытым ключом Боба является 7, а закрытым - 23. $n = 55$. Секретное число Алисы, i, равно 4, секретное число Боба, j - 2. (Предположим, что числа i и j могут принимать только значения 1, 2, 3 и 4.)

(1) Алиса выбирает $x = 39$ и $c = E_B(39) = 19$.

(2) Алиса вычисляет c-i=19-4=15. Она посылает 15 Бобу.

(3) Боб вычисляет следующие четыре числа:

$y_1 = D_B\{15+1) = 26$

$y_2 = D_B\{15+2) = 18$

$y_3 = D_B\{15+3) = 2$

$y_4 = D_B\{15+4) = 39$

Он выбирает $p = 31$ и вычисляет:

$z_1 = (26 \bmod 31) = 26$

$z_2 = (18 \bmod 31) = 18$

$z_3 = (2 \bmod 31) = 2$

$z_4 = (39 \bmod 31) = 8$

Он выполняет все проверки и убеждается, что последовательность правильна.

(4) Боб посылает Алисе эту последовательность чисел, соблюдая их порядок:

26, 18, 2+1, 8+1, 31, т.е., 26, 18, 3, 9, 31

(5) Алиса проверяет, конгруэнтно ли четвертое число $X \bmod p$. Так как $9 \neq 39 \pmod{31}$, то $i > j$.

(6) Алиса сообщает об этом Бобу.

Этот протокол можно использовать для создания намного более сложных протоколов. Группа людей может проводить секретный аукцион по сети. Они логически упорядочивают себя по кругу и, с помощью попарных сравнений, определяют, кто предложил большую цену. Чтобы помешать людям уже изменять сделанные предложения в середине аукциона должен использоваться какой-то протокол вручения битов. Если аукцион проводится по голландской системе, то предложивший наивысшую цену получает предмет за предложенную цену. Если аукцион проводится по английской системе, то он получает предмет за вторую высшую цену. (Это может быть выяснено во время второго круга попарных сравнений.) Аналогичные идеи применимы при заключении сделок, переговорах и арбитраже.

23.15 Вероятностное шифрование

Понятие **вероятностного шифрования** было изобретено Шафи Голдвассером (Shafi Goldwasser) и Сильвией Микали [624]. Хотя их теория позволяет создать самую безопасную из изобретенных криптосистем, ранняя реализации была неэффективной [625]. Но более поздние реализации все изменили.

Идеей вероятностного шифрования является устранение утечки информации в криптографии с открытыми ключами. Так как криптоаналитик всегда может расшифровать случайные сообщения открытым ключом, он может получить некоторую информацию. При условии, что у него есть шифротекст $C = E_K(M)$, и он пытается получить открытый текст M, он может выбрать случайное сообщение M' и зашифровать его: $C' = E_K(M')$. Если $C' = C$, то он угадал правильный открытый текст. В противном случае он делает следующую попытку.

Кроме того, вероятностное шифрование позволяет избежать даже частичной утечки информации об оригинальном сообщении. При использовании криптографии с открытыми ключами криптоаналитик иногда может узнать кое-что о битах: XOR 5-го, 17-го и 39-го битов равно 1, и т.п.. При вероятностном шифровании остается скрытой и такая информация.

Таким способом можно извлечь не много информации, но потенциально возможность криптоаналитика расшифровывать случайные сообщения вашим открытым ключом может создать определенные проблемы. Каждый раз, шифруя сообщение, криптоаналитик может извлечь немного информации. Никто не знает, насколько значительна эта информация.

Вероятностное шифрование пытается устранить эту утечку. Цель этого метода состоит в том, чтобы ни вычисления, проводимые над шифротекстом, ни проверка любых других открытых текстов не смогли дать криптоаналитику никакой информации о соответствующем открытом тексте.

При вероятностном шифровании алгоритм шифромания является вероятностным, а не детерминированным. Другими словами, многие шифротексты при расшифровке дают данный открытый текст, и конкретный шифротекст, используемый в любом конкретном шифровании, выбирается случайным образом.

$C_1 = E_K(M),\ C_2 = E_K(M),\ C_3 = E_K(M), \ldots C_i = E_K(M)$

$M = D_K(C_1) = D_K(C_2) = D_K(C_3) = \ldots = D_K(C_i)$

При вероятностном шифровании криптоаналитику больше не удастся шифровать произвольные открытые тексты в поисках правильного шифротекста. Для иллюстрации пусть у криптоаналитика есть шифротекст $C_i = E_K(M)$. Даже если он праильно угадает M, полученный при шифровании $E_K(M)$ результат будет совершенно другим шифротекстом C: C_j. Сравнивая C_i и C_j, он не может по их совпадению определить правильность своей догадки.

Это поразительно. Даже если у криптоаналитика есть открытый ключ шифрования, открытый текст и шифротекст, он не может без закрытого ключа дешифрирования доказать, что шифротекст является результатом шифрования конкретного открытого текста. Даже выполнив исчерпывающий поиск, он может доказать только, что каждый возможный открытый текст является возможным открытым текстом.

В этой схеме шифротекст всегда будет больше открытого текста. Этого невозможно избежать, это является результатом того, что многие шифротексты расшифровываются в один и тот же открытый текст. В первой схеме вероятностного шифрования [625] шифротекст получался настолько больше открытого текста, что он был бесполезным.

Однако Мануэль Блюм (Manual Blum) и Голдвассер (Goldwasser) получили эффективную реализацию вероятностного шифрования с помощью генератора псевдослучайных битов Blum Blum Shub (BBS), описанного в разделе 17.9 [199].

Генератор BBS основан на теории квадратичных остатков. Существуют два простых числа, p и q, конгруэнтных 3 по модулю 4. Это закрытый ключ. Их произведение, $pq = n$, является открытым ключом. (Запомните свои p и q, безопасность схемы опирается на сложность разложения n на множители.)

Для шифрования сообщения M сначала выбирается случайное x, взаимно простое с n. Затем вычисляется

$x_0 = x^2 \bmod n$

x_0 служит стартовой последовательностью для генератора псевдослучайных битов BBS, а выход генератора используется в качестве потокового шифра. Побитно выполняется XOR M с выходом генератора. Генератор выдает биты b_i (младший значащий бит x_i, где $x_i = x_{i-1}^2 \bmod n$), поэтому

$M=M_1, M_2, M_3, \ldots M_t$

с = $M_1 \oplus b_1, M_2 \oplus b_2, M_3 \oplus b_3, \ldots\ M_t \oplus b_t$

где t - это длина открытого текста

Добавьте последнее вычисленное значение, x_t, к концу сообщения, и дело сделано.

Расшифровать это сообщение можно только одним способом - получить x_0 и с этой стартовой последовательностью запустить генератор BBS, выполняя XOR выхода с шифротекстом. Так как генератор BBS безопасен влево, значение x_t бесполезно для криптоаналитика. Только тот, кому известны p и q, может расшифровать сообщение. Вот как на языке C выглядит алгоритм получения x_0 из x_t:

```
int xO (int p, int q, int n, int t, int xt) {
     int a, b, u, v, w, z;
     /* мы уже знаем, что НОД(p,q) == 1 */
     (void)extended_euclidian(p, q, &a, &b);
```

```
    u = modexp ((p+1)/4, t, p-1);
    v = modexp ((q+1)/4, t, q-1);
    w = modexp (xt%p, u, p);
    z = modexp (xt%p, v, q);
    return (b*q*w + a*p*z) % n;
}
```

При наличии x_0 дешифрирование несложно. Просто задайте стартовую последовательность генератора BBS и выполните XOR результата с шифротекстом.

Эту схему можно сделать еще быстрее, используя все известные безопасные биты x_i, а не только младший значащий бит. С таким улучшением вероятностное шифрование Blum-Goldwasser оказывается быстрее RSA и не допускает утечки информации об открытом тексте. Кроме того, можно доказать, что сложность вскрытия этой схемы равна сложности разложения n на множители.

С другой стороны, эта схема совершенно небезопасна по отношению к вскрытию с выбранным шифротекстом. По младшим значащим битам правильных квадратичных остатков можно вычислить квадратный корень любого квадратичного остатка. Если это удастся, то удастся и разложение на множители. Подробности можно найти в [1570, 1571, 35, 36].

23.16 Квантовая криптография

Квантовая криптография вводит естественную неопределенность квантового мира. С ее помощью можно создавать линии связи, которые невозможно послушать, не внося помех в передачу. Законы физики надежно защищают такой квантовый канал, даже если подслушивающий может предпринимать любые действия, даже если он имеет доступ к неограниченной вычислительной мощности, даже если **P = NP**. Шарль Бенне (Charles Bennett), Жиль Брассар (Gilles Brassard), Клод Крепо (Claude Crepeau) и другие расширили эту идею, описав квантовое распределение ключей, квантовое бросание монеты, квантовое вручение бита, квантовую передачу с забыванием и квантовые вычисления с несколькими участниками. Описание их результатов можно найти в [128, 129, 123, 124, 125, 133, 126, 394, 134, 392, 243, 517, 132, 130, 244, 393, 396]. Лучшим обзором по квантовой криптографии является [131]. Другим хорошим нетехническим обзором может служить [1651]. Полную библиографию по квантовой криптографии можно найти в [237].

Эти идеи так и остались бы предметом обсуждения фанатиков криптографии, но Бенне и Брассар разработали действующую модель [127, 121, 122]. Теперь у нас есть *экспериментальная* квантовая криптография.

Итак устройтесь поудобнее, налейте себе чего-нибудь выпить и расслабьтесь. Я попробую объяснить вам, что это такое.

В соответствии с законами квантовой механики частицы на самом деле не находятся в одном месте, а с определенной вероятностью существуют сразу во многих местах. Однако это так только до тех пор, пока не приходит ученый и не обмеряет частицу, "оказавшуюся" в данном конкретном месте. Но измерить все параметры частицы (например, координаты и скорость) одновременно невозможно. Если измерить одну из этих двух величин, сам акт измерения уничтожает всякую возможность измерить другую величину. Неопределенность является фундаментальным свойством квантового мира, и никуда от этого не денешься.

Эту неопределенность можно использовать для генерации секретного ключа. Путешествуя, фотоны колеблются в определенном направлении, вверх-вниз, влево-вправо, или, что более вероятно, под каким-то углом. Обычный солнечный свет неполяризован, фотоны колеблются во всех возможных направлениях. Когда направление колебаний многих фотонов совпадает, они являются **поляризованными**. Поляризационные фильтры пропускают только те фотоны, которые поляризованы в определенном направлении, а остальные блокируются. Например, горизонтальный поляризационный фильтр пропускает только фотоны с горизонтальной поляризацией. Повернем этот фильтр на 90 градусов, и теперь сквозь него будут проходить только вертикально поляризованные фотоны.

Пусть у вас есть импульс горизонтально поляризованных фотонов. Если они попробуют пройти через горизонтальный фильтр, то у них у всех прекрасно получится. Если медленно поворачивать фильтр на 90 градусов, количество пропускаемых фотонов будет становиться все меньше и меньше, и наконец ни один фотон не пройдет через фильтр. Это противоречит здравому смыслу. Кажется, что даже незначительный поворот фильтра должен остановить все фотоны, так как они горизонтально поляризованы. Но в квантовой механике каждая частица с определенной вероятностью может изменить свою поляризацию и проскочить через фильтр. Если угол отклонения фильтра невелик, эта вероятность высока, а если он равен 90 градусам, то вероятность равна нулю. А если угол поворота фильтра равен 45 градусам, вероятность фотона пройти фильтр равна 50 процентам.

Поляризацию можно измерить в любой **системе координат**: двух направлениях, расходящихся под прямым углом. Примерами систем координат являются прямоугольная - горизонтальное и вертикальное направления - и

диагональная - левая и правая диагонали. Если импульс фотонов поляризован в заданной системе координат, то при измерении в той же системе координат вы узнаете поляризацию. При измерении в неправильной системе координат, вы получите случайный результат. Мы собираемся использовать это свойство для генерации секретного ключа:

(1) Алиса посылает Бобу последовательность фотонных импульсов. Каждый из импульсов случайным образом поляризован в одном из четырех направлений: горизонтальном, вертикальном, лево- и праводиагональном.

Например, Алиса посылает Бобу:

| | / — \ — | — /

(2) У Боба есть детектор поляризации. Он может настроить свой детектор на измерение прямоугольной или диагональной поляризации. Одновременно мерить и ту, и другую у него не получится, ему не позволит квантовая механика. Измерение одной поляризации не даст измерить другую. Итак, он устанавливает свои детекторы произвольным образом:

X + + X X X + X + +

Теперь, если Боб правильно настроит свой детектор, он зарегистрирует правильную поляризацию. Если он настроит детектор на измерение прямоугольной поляризации, и импульс будет поляризован прямоугольно, он узнает, какую поляризацию фотонов выбрала Алиса. Если он настроит детектор на измерение диагональной поляризации, а импульс будет поляризован прямоугольно, то результат измерения будет случайным. Боб не сможет определить разницу. В приведенном примере он может получить следующий результат:

/ | — \ / \ — / — |

(3) Боб сообщает Алисе по незащищенному каналу, какие настройки он использовал.

(4) Алиса сообщает Бобу, какие настройки были правильными. В нашем примере детектор был правильно установлен для импульсов 2, 6, 7 и 9.

(5) Алиса и Боб оставляют только правильно измеренные поляризации. В нашем примере они оставляют:

* | * * * \ — * — *

С помощью заранее приготовленного кода Алиса и Боб преобразуют в биты эти результаты измерений поляризации. Например, горизонтальная и леводиагональная могут означать единицу, а вертикальная и праводиагональная - ноль. В нашем примере они оба получат:

0 0 1 1

Итак, Алиса и Боб получили четыре бита. С помощью этой системы они могут генерировать столько битов, сколько им нужно. В среднем Боб правильно угадывает в 50 процентах случаев, поэтому для генерации n битов Алисе придется послать $2n$ фотонных импульсов. Они могут использовать эти биты как секретный ключ симметричного алгоритма или обеспечить абсолютную безопасность, получив достаточно битов для использования в качестве одноразового блокнота.

Замечательным является то, что Ева не сможет подслушать. Как и Бобу, ей нужно угадать тип измеряемой поляризации, и, как и у Боба, половина ее догадок будет неправильной. Так как неправильные измерения изменяют поляризацию фотонов, то при подслушивании она неминуемо вносит ошибки в передачу. Если это так, Алиса и Боб получат различные битовые последовательности. Итак, Алиса и Боб заканчивают протокол подобными действиями:

(6) Алиса и Боб сравнивают несколько битов своих строк. По наличию расхождений они узнают о подслушивании. Если строки не отличаются, то они отбрасывают использованные для сравнения биты и используют оставшиеся.

Улучшения этого протокола позволяют Алисе и Боб использовать свои биты даже в присутствии Евы [133, 134, 192]. Они могут сравнивать только четность битовых подмножеств. Тогда, если не обнаружено расхождений, им придется отбросить только один бит подмножества. Это обнаруживает подслушивание с вероятностью 50 процентов, но если они сверят таким образом n различных битовых подмножеств, вероятность Евы подслушать и остаться незамеченной будет равна $1/2^n$.

В квантовом мире не бывает пассивного подслушивания. Если Ева попытается раскрыть все биты, она обязательно разрушит канал связи.

Бенне и Брассар построили работающую модель квантового распределения ключей и обменялись безопасными битами на оптической скамье. Последнее, что я слышал, было сообщение о том, что в British Telecom по-

сылали биты по 10-километровому оптоволокну [276, 1245, 1533]. Они считают, что достижимо и расстояние в 50 километров. Это поражает воображение.

Часть IV

Реальный мир

Глава 24
Примеры реализаций

Одно дело разрабатывать протоколы и алгоритмы, и совсем другое дело встраивать их в операционные системы. В теории практика и теория не отличимы, но на практике между ними огромные различия. Часто идеи замечательно выглядят на бумаге, но не работают в реальной жизни. Может быть слишком велики требования к скорости канала, может быть протокол слишком медлителен. Некоторые из вопросов использования криптографии рассматриваются в главе 10, в этой главе обсуждаются примеры того, как криптографические алгоритмы реализуются на практике.

24.1 Протокол управления секретными ключами компании IBM

В конце 70-х годов IBM, используя только симметричную криптографию, разработала законченную систему управления ключами для передачи данных и безопасности файлов в компьютерных сетях [515, 1027]. Не так важны реальные механизмы протокола, как его общая философия: за счет автоматизации генерации, распределения, установки, хранения, изменения и разрушения ключей этот протокол далеко продвинулся, обеспечивая безопасность лежащих в его основе криптографических алгоритмов.

Этот протокол обеспечивает три вещи: безопасную связь между сервером и различными терминалами, безопасное хранение файлов на сервере и безопасную связь между серверами. Протокол не обеспечивает настоящего прямого соединения терминал-терминал, хотя его модификация может реализовать такую возможность.

Каждый сервер сети подключен к криптографической аппаратуре, которая выполняет все шифрование и дешифрирование. У каждого сервера есть **Главный ключ** (Master Key), KM_0, и два варианта, KM_1 и KM_2, которые являются упрощенными вариантами KM_0. Эти ключи используются для шифрования других ключей и для генерации новых ключей. У каждого терминала есть **Главный ключ терминала** (Master Terminal Key), *KMT*, который используется для обмена ключами с другими терминалами.

KMT хранятся на серверах, зашифрованные ключом KM_1. Все остальные ключи, например, используемые для шифрования файлов ключей (они называются *KNF*), хранятся в зашифрованной форме, закрытые ключом KM_2. Главный ключ KM_0 хранится в энергонезависимом модуле безопасности. Сегодня это может быть либо ключ в ПЗУ, либо магнитная карточка, или ключ может вводиться пользователем с клавиатуры (возможно как текстовая строка, преобразуемая в ключ). KM_1 и KM_2 не хранятся где-нибудь в системе, а, когда понадобится, вычисляются по KM_0. Сеансовые ключи для связи между серверами генерируются на сервере с помощью псевдослучайного процесса. Аналогичным образом генерируются ключи для шифрования хранимых файлов (*KNF*).

Сердцем протокола служит устойчивый к вскрытию модуль, называемый **криптографической аппаратурой** (cryptographic facility). И на сервере, и на терминале все шифрование и дешифрирование происходит именно в этом модуле. В этом модуле хранятся самые важные ключи, используемые для генерации действительных ключей шифрования. После того, как эти ключи записаны, считать их становится невозможным. Кроме того, они помечены для конкретного использования: ключ, предназначенный для решения одной задачи, не может случайно быть использован для решения другой. Эта концепция **векторов управления ключами** возможно является самым значительным достижением этой системы. Дональд Дэвис (Donald Davies) Вильям Прайс (William Price) подробно рассматривают этот протокол управления ключами в [435].

Модификация

Модификацию этой схемы главного и сеансовых ключей можно найти в [1478]. Она построена на базе сетевых узлов с аппаратурой проверки подлинности ключей, которая обслуживает локальные терминалы. Эта модификация была разработана, чтобы:

— Обезопасить дуплексный канал между двумя пользовательскими терминалами.

— Обезопасить связь с помощью шифрованной почты.

— Обеспечить защиту личных файлов.

— Обеспечить возможность цифровой подписи.

Для связи и передачи файлов между пользователями в этой схеме используются ключи, генерированные в аппаратуре проверки подлинности ключей, отправляемые пользователям после шифрования с помощью главного ключа. Информация о личности пользователя встраивается в ключ, предоставляя доказательство того, что сеансовый ключ используется конкретной парой пользователей. Возможность **проверки подлинности ключей** является главной в этой системе. Хотя в системе не используется криптография с открытыми ключами, она поддерживает возможность, похожую на цифровую подпись: ключ может быть прислан только из конкретного источника и прочитан только в конкретном месте назначения.

24.2 MITRENET

Одной из самых ранних реализаций криптографии с открытыми ключами была экспериментальная система MEMO (MITRE Encrypted Mail Office, Шифрованное почтовое отделение). MITRE - это была команда умных парней, работающая по заказу Министерства обороны. MEMO служила системой безопасной электронной почты для пользователей сети MITRENET и использовала криптографию с открытыми ключами для обмена ключами и DES для шифрования файлов.

В системе MEMO все открытые ключи хранятся в Центре распределения открытых ключей (Public Key Distribution Center), который является отдельным узлом сети. Ключи хранятся в стираемом перепрограммируемом ПЗУ, чтобы не дать изменить их. Закрытые ключи генерируются пользователями системы.

Чтобы пользователь мог отправлять безопасные сообщения, система сначала устанавливает безопасное соединение с Центром распределения открытых ключей. Пользователь запрашивает в Центре файл всех открытых ключей. Если пользователь проходит идентификацию с использованием его закрытого ключа, Центр пересылает запрошенный список на рабочую станцию пользователя. Для обеспечения целостности список шифруется с помощью DES.

Для шифрования сообщений используется DES. Для шифрования файлов система генерирует случайный ключ DES, пользователь шифрует файл ключом DES, а ключ DES - открытым ключом получателя. Зашифрованный файл и ключ отправляются получателю.

MEMO не предусматривает мер предосторожности против потерь ключей. Существуют некоторые средства проверки целостности сообщений с использованием контрольных сумм. В систему не встроены средства проверки подлинности.

Прежде, чем система была реализована, была доказана небезопасность конкретной реализации системы открытых ключей в MEMO - обмена ключами по схеме Diffie-Hellman над $GF(2^{127})$ (см. раздел 11.6), хотя нетрудно изменить систему, чтобы можно было использовать большие числа. MEMO была изобретена главным образом для экспериментальных целей и никогда не использовалась в реальной системе MITRENET.

24.3 ISDN

Bell-Northern Research разработала прототип безопасного телефонного терминала ISDN (Integrated Services Digital Network, Цифровая сеть с интегрированием услуг) [499, 1192, 493, 500]. Как телефонный аппарат, терминал остался на уровне прототипа. В результате появился Уровень безопасности пакетов данных (Packet Data Security Overlay). Терминал использует схему обмена ключами Diffie-Hellman, цифровые подписи RSA и DES для шифрования данных. Он может передавать и принимать речь и данные со скоростью 64 Кбит/с.

Ключи

В телефон встроена пара "открытый ключ/закрытый ключ" для длительного использования. Закрытый ключ хранится в устойчивом от вскрытия модуле телефона. Открытый ключ служит для идентификации телефона. Эти ключи являются частью самого телефонного аппарата и не могут быть изменены.

Кроме того, в телефоне хранятся еще два открытых ключа. Одним из них является открытый ключ владельца аппарата. Этот ключ используется для проверки подлинности команд владельца, он может быть изменен по команде, подписанной владельцем. Так пользователь может передать кому-то другому право владения аппаратом.

В телефоне также хранится открытый ключ сети. Он используется для проверки подлинности команд аппаратуры управления сетью и проверки подлинности вызовов от других пользователей сети. Этот ключ также можно изменить командой, подписанной владельцем. Это позволяет владельцу менять сеть, к которой подключен его аппарат.

Эти ключи рассматриваются как ключи длительного пользования - они меняются редко, если вообще меняются. В телефоне также хранится пара "открытый ключ/закрытый ключ" для краткосрочного использования. Они встроены в сертификат, подписанный центром управления ключами. Два телефона обмениваются сертификатами при установлении соединения. Подлинность этих сертификатов удостоверяется открытым ключом сети.

Обмен сертификатами и их проверка выполняются только при установлении безопасного соединения между аппаратами. Для установления безопасного соединения между людьми протокол содержит дополнительный компонент. В аппаратном **ключе зажигания**, который вставляется в телефон владельцем, хранится закрытый ключ владельца, зашифрованный секретным паролем, известным только владельцу (его не знает ни телефонный аппарат, ни центр управления сетью, ни еще кто-нибудь). Ключ зажигания также содержит сертификат, подписанный центром управления сетью, в который включены открытый ключ владельца и некоторая идентификационная информация (имя, компания, специальность, степень допуска, любимые сорта пиццы, сексуальная ориентация и прочее). Все это также зашифровано. Для дешифрирования этой информации и ввода ее в телефон

пользователь вводит свой секретный пароль с клавиатуры аппарата. Телефонный аппарат использует эту информацию для соединения, но она удаляется после того, как пользователь извлечет свой ключ зажигания.

В телефоне также хранится набор сертификатов, выданных центром управления сетью. Эти сертификаты удостоверяют право конкретных пользователей пользоваться конкретными телефонными аппаратами.

Вызов

Вызов Боба Алисой происходит следующим образом.

(1) Алиса вставляет в телефон свой ключ зажигания и вводит свой пароль.

(2) Телефон опрашивает ключ зажигания, чтобы определить личность Алисы и выдать ей сигнал "линия свободна".

(3) Телефон проверяет свой набор сертификатов, проверяя, что Алиса имеет право использовать этот аппарат.

(4) Алиса набирает номер, телефон определяет адресата звонка.

(5) Два телефона используют протокол обмена ключами на базе криптографии с открытыми ключами, чтобы генерировать уникальный и случайный сеансовый ключ. Все последующие этапы протокола шифруются с помощью этого ключа.

(6) Телефон Алисы передает свой сертификат и идентификатор пользователя.

(7) Телефон Боба проверяет подписи сертификата и идентификатора пользователя, используя открытый ключ сети.

(8) Телефон Боба инициирует последовательность запросов/ответов. Для этого необходимо в реальном времени (не позднее заданной задержки) отправлять подписанные ответы на запросы. (Это помешает злоумышленнику использовать сертификаты, скопированные из предыдущего обмена.) Один ответ должен быть подписан закрытым ключом телефона Алисы, а другой - закрытым ключом Алисы.

(9) Если Боба нет у телефона, то его телефон звонит.

(10) Если Боб дома, он вставляет в телефон свой ключ зажигания. Его телефон опрашивает ключ зажигания и проверяет сертификат Боба, как на этапах (2) и (3).

(11) Боб передает свой сертификат и идентификатор пользователя.

(12) Телефон Алисы проверяет подписи Боба, как на этапе (7) и инициирует последовательность запросов/ответов, как на этапе (8).

(13) Оба телефона выводят на свои экраны личность и номер телефона другого пользователя.

(14) Начинается безопасный разговор.

(15) Когда одна из сторон вешает трубку, удаляются сеансовый ключ, а также сертификаты, которые телефон Боба получил от телефона Алисы, и сертификаты, которые телефон Алисы получил от телефона Боба.

Каждый ключ DES уникален для каждого звонка. Он существует только внутри двух телефонных аппаратов и только в течение разговора, а после его окончания немедленно уничтожается. Если злоумышленник добудет один или оба участвовавших в разговоре аппарата, он не сможет расшифровать ни один предшествующий разговор, в котором участвовали эти два аппарата.

24.4 STU-III

STU обозначает "Secure Telephone Unit" (Безопасный телефонный модуль), разработанный в NSA безопасный телефон. По размерам и форме этот модуль почти такой же, как и обычный телефон, и может быть использован также, как и обычный телефон. Аппараты устойчивы к взлому, без ключа они работают как несекретные. Они также включают порт передачи данных и помимо передачи речи могут быть использованы для безопасной передачи данных по модемному каналу [1133].

Уитфилд Диффи описал STU-III в [494]:

> Чтобы позвонить, используя STU-III, звонящий сначала обычным образом звонит на другой STU-III, затем вставляет похожее на ключ устройство, содержащее криптографическую переменную, и нажимает кнопку "секретные переговоры" ("go secure"). Спустя примерно 15 секунд задержки, нужной для криптографической настройки, каждый телефон выводит на экран информацию о личности и допуске другой стороны, и разговор может начинаться.
>
> Беспрецедентным шагом был объявление Уолтера Дили (Walter Deeley), заместителя директора NSA по безопасности коммуникаций, о STU-III или будущей системе безопасной голосовой связи в эксклюзивном интервью, данном *The New York Times* [282]. Главной целью новой системы было предоставить Министерству обороны США и его подрядчикам средства безопасной передачи речи и безопасной низкоскоростной передачи данных. В интервью не было много сказано о работе системы, но постепенно информация начала появляться. В новой системе используются открытые ключи.

О новом подходе к распределению ключей было рассказано в [68], в одной статье говорилось о телефонах, "перепрограммируемых раз в год по безопасному телефонному каналу", что весьма вероятно предполагает использование протокола проверки сертификатов, аналогичного описанному [в разделе 24.3], который минимизирует для телефонов необходимость общаться с центром управления ключами. Последние известия были более информативными, в них рассказывалось о системе управления ключами, названной FIREFLY, которая [1341] "разработана на базе технологии открытых ключей и используется для распределения ключей шифрования попарного трафика". И это описание, и свидетельские показания, данные Конгрессу США Ли Ньювиртом (Lee Neuwirth) из Cylink [1164] предполагают использование комбинации обмена ключами и сертификатами, аналогичного используемому в безопасных телефонах ISDN. Весьма вероятно, что FIREFLY также основана на возведении в степень.

STU-III производятся AT&T и GE. За 1994 год было выпущено 300000-400000 штук. Новая версия, Secure Terminal Equipment (STE, Безопасный терминал), будет работать по линиям ISDN.

24.5 KERBEROS

Kerberos представляет собой разработанный для сетей TCP/IP протокол проверки подлинности с доверенной третьей стороной. Служба Kerberos, работающая в сети, действует как доверенный посредник, обеспечивая безопасную сетевую проверку подлинности, дающую пользователю возможность работать на нескольких машинах сети. Kerberos на симметричной криптографии (реализован DES, но вместо него можно использовать и другие алгоритмы). При общении с каждым объектом сети Kerberos использует отличный общий секретный ключ, и знание этого секретного ключа равносильно идентификации объекта.

Kerberos был первоначально разработан в МТИ для проекта Афина. Модель Kerberos основана на протоколе Needham-Schroeder с доверенной третьей стороной (см. раздел 3.3) [1159]. Оригинальная версия Kerberos, Версия 4, определена в [1094, 1499]. (Версии с 1 по 3 были внутренними рабочими версиями.) Версия 5, модификация Версии 4, определена в [876, 877, 878]. Лучшим обзором по Kerberos является [1163]. Другие обзорные статьи - [1384, 1493], использование Kerberos в реальном мире хорошо описано в [781, 782].

Модель Kerberos

Базовый протокол Kerberos был схематично описан в разделе 3.3. В модели Kerberos существуют расположенные в сети объекты - клиенты и серверы. Клиентами могут быть пользователи, но могут и независимые программы, выполняющие следующие действия: загрузку файлов, передачу сообщений, доступ к базам данных, доступ к принерам, получение административных привилегий, и т.п.

Kerberos хранит базу данных клиентов и их секретных ключей. Для пользователей-людей секретный ключ является зашифрованным паролем. Сетевые службы, требующие проверки подлинности, и клиенты, которые хотят использовать эти службы, регистрируют в Kerberos свои секретные ключи.

Так как Kerberos знает все секретные ключи, он может создавать сообщения, убеждающие один объект в подлинности другого. Kerberos также создает сеансовые ключи, которые выдаются клиенту и серверу (или двум клиентам) и никому больше. Сеансовый ключ используется для шифрования сообщений, которыми обменив аются две стороны, и уничтожается после окончания сеанса.

Для шифрования Kerberos использует DES. Kerberos версии 4 обеспечивал нестандартный, слабый режим проверки подлинности - он не мог определить определенный изменения шифротекста (см. раздел 9.10). Kerberos версии 5 использует режим CBC.

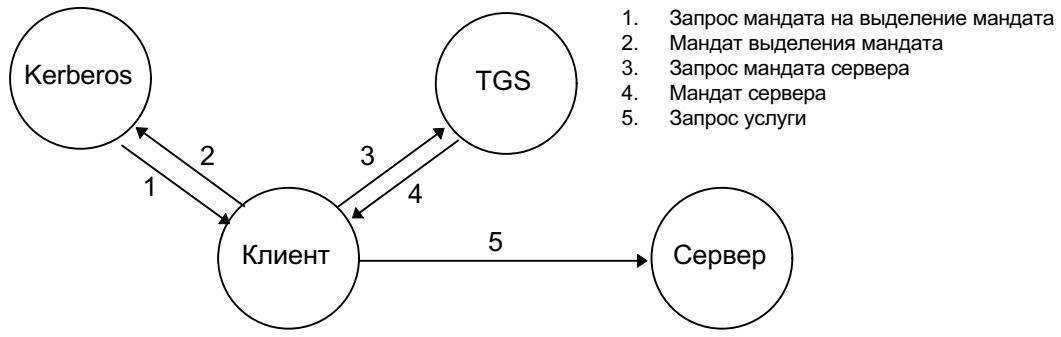


Рис. 24-1. Этапы проверки подлинности Kerberos

Как работает Kerberos

В этом разделе рассматривается Kerberos версии 5. Ниже я обрисую различия между версиями 4 и 5. Протокол Kerberos прост (см. 23rd). Клиент запрашивает у Kerberos мандат на обращение к **Службе выделения мандатов** (Ticket-Granting Service, TGS). Этот мандат, зашифрованный секретным ключом клиента, посылается

клиенту. Для использования конкретного сервера клиент запрашивает у TGS мандат на обращение к серверу. Если все в порядке, TGS посылает мандат клиенту. Затем клиент предъявляет серверу этот мандат вместе с удостоверением. И снова, если атрибуты клиента правильны, сервер предоставляет клиенту доступ к услуге.

Табл. 24-1.
Таблица сокращений Kerberos

c	= клиент
s	= сервер
a	= сетевой адрес клиента
v	= начало и окончание времени действия мандата
t	= метка времени
K_x	= секретный ключ x
$K_{x,y}$	= сеансовый ключ для x и y
$(m)K_x$	= m, шифрованное секретным ключом x
$T_{x,y}$	= мандат x на использование y
$A_{x,y}$	= удостоверение x для y

Атрибуты

Kerberos использует два типа атрибутов: **мандаты** и **удостоверения**. (В дальнейшем в этом разделе будет использоваться нотация, используемая в документах Kerberos - см. 23-й.) Мандат используется для безопасной передачи серверу личности клиента, которому выдан этот мандат. В нем также содержится информация, которую сервер может использовать для проверки того, что клиент, использующий мандат, - это именно тот клиент, которому этот мандат был выдан. Удостоверение - это дополнительный атрибут, предъявляемый вместе с мандатом. Мандат Kerberos имеет следующую форму:

$T_{c,s} = s, \{c, a, v, K_{c,s}\}K_s.$

Мандат хорош для одного сервера и одного клиента. Он содержит имя клиента, его сетевой адрес, имя сервера, метку времени и сеансовый ключ. Эта информация шифруется секретным ключом сервера. Если клиент получил мандат, он может использовать его для доступа к серверу много раз - пока не истечет срок действия мандата. Не может расшифровать мандат (он не знает секретного ключа сервера), но он может предъявить его серверу в зашифрованной форме. Прочитать или изменить мандат при передаче его по сети невозможно. Удостоверение Kerberos имеет следующую форму:

$A_{c,s} = \{c, t, ключ\}K_{c,s}$

Клиент создает его каждый раз, когда ему нужно воспользоваться услугами сервера. Удостоверение содержит имя клиента, метку времени и необязательный дополнительный сеансовый ключ, все эти данные шифруются сеансовым ключом, общим для клиента и сервера. В отличие от мандата удостоверение используется только один раз. Однако это не проблема, так как клиент может генерировать удостоверения по мере надобности (ему известен общий секретные ключ).

Использование удостоверения преследует две цели. Во первых, оно содержит некоторый открытый текст, зашифрованный сеансовым ключом. Это доказывает, что клиенту известен ключ. Что не менее важно, зашифрованный открытый текст включает метку времени. Злоумышленник, которому удалось записать и мандат, и удостоверение, не сможет использовать их спустя два дня.

Сообщения Kerberos версии 5

В Kerberos версии 5 используется пять сообщений (см. 23-й):

1. Клиент-Kerberos: c,tgs
2. Kerberos-клиент: $\{K_{c,tgs}\}K_c$, $\{T_{c,tgs}\}K_{tgs}$
3. Клиент-TGS: $\{A_{c,s}\}K_{c,tgs}\{T_{c,tgs}\}\ K_{tgs,s}$
4. TGS-клиент: $\{K_{c,s}\}K_{c,tgs}\{T_{c,s}\}K_s$
5. Клиент-сервер: $\{A_{c,s}\}K_{c,s}\ \{T_{c,s}\}K_s$

Теперь рассмотрим использование этих сообщений подробно.

Получение первоначального мандата

У клиента есть часть информации, доказывающей его личность - его пароль. Понятно, что не хочется заставлять клиента передавать пароль по сети. Протокол Kerberos минимизирует вероятность компрометации пароля, но при этом не позволяет пользователю правильно идентифицировать себя, если он не знает пароля.

Клиент посылает сообщение, содержащее его имя и имя его сервера TGS на сервер проверки подлинности Kerberos. (может быть несколько серверов TGS.) На практике пользователь, скорее всего, просто вводит свое имя и программа входа в систему посылает запрос.

Сервер проверки подлинности Kerberos ищет данные о клиенте в своей базе данных. Если информация о клиенте есть в базе данных, Kerberos генерирует сеансовый ключ, который будет использоваться для обмена данными между клиентом и TGS. Он называется **Мандатом на выделение мандата** (Ticket Granting Ticket, TGT). Kerberos шифрует этот сеансовый ключ секретным ключом клиента. Затем он создает для клиента TGT, доказывающий подлинность клиента TGS, и шифрует его секретным ключом TGS. Сервер проверки подлинности посылает эти два зашифрованных сообщения клиенту.

Теперь клиент расшифровывает первое сообщение и получает сеансовый ключ. Секретный ключ является однонаправленной хэш-функцией клиентского пароля, поэтому у законного пользователя не будет никаких проблем. Самозванец не знает правильного пароля и, следовательно, не может расшифровать ответ сервера проверки подлинности. Доступ запрещается, и самозванный клиент не может получить мандат или сеансовый ключ.

Клиент сохраняет TGT и сеансовый ключ, стирая пароль и хэш-значение. Эта информация уничтожается для уменьшения вероятности компрометации. Если враг попытается скопировать память клиента, он получит только TGT и сеансовый ключ. Эти данные важны, но только на время жизни TGT. Когда срок действия TGT истечет, эти сведения станут бессмысленными. Теперь в течение времени жизни TGT клиент может доказывать TGS свою подлинность.

Получение серверных мандатов

Клиенту требуется получить отдельный мандат для каждой нужной ему услуги. TGS выделяет мандаты для отдельных серверов.

Когда клиенту нужен мандат, которого у него пока нет, он посылает запрос к TGS. (На практике программа, скорее всего, делает это автоматически и незаметно для пользователя.)

TGS, получив запрос, расшифровывает TGT своим секретным ключом. Затем TGS использует включенный в TGT сеансовый ключ, чтобы расшифровать удостоверение. Наконец TGS сравнивает информацию удостоверения с информацией мандата, сетевой адрес клиента с адресом отправителя запроса и метку времени с текущим временем. Если все совпадает, TGS разрешает выполнение запроса.

Проверка меток времени предполагает, что часы всех компьютеров синхронизированы, по крайней мере с точностью до нескольких минут. Если время, указанное в запросе, отстоит от текущего момента слишком далеко в прошлое или в будущее, TGS считает запрос попыткой повторения предыдущего запроса. TGS должна также отслеживать правильность сроков действия удостоверений, так как услуги сервера могут запрашиваться несколько раз последовательно с одним мандатом, но разными удостоверениями. Другой запрос с тем же мандатом и уже использованной меткой времени удостоверения будет отвергнут.

В ответ на правильный запрос TGS возвращает правильный мандат, который клиент может предъявить серверу. TGS также создает новый сеансовый ключ для клиента и сервера, зашифрованный сеансовым ключом, общим для клиента и TGS. Оба этих сообщения отправляются клиенту. Клиент расшифровывает сообщение и извлекает сеансовый ключ.

Запрос услуги

Теперь клиент может доказать свою подлинность серверу. Он создает сообщение, очень похожее на то, которое посылалось TGS (и это понятно, так как TGS - тоже услуга).

Клиент создает удостоверение, состоящее из его имени, сетевого адреса и метки времени, зашифрованное сеансовым ключом, который был генерирован TGS для сеанса клиента и сервера. Запрос состоит из мандата, полученного от Kerberos (уже зашифрованного секретным ключом сервера) и зашифрованного идентификатора.

Сервер расшифровывает и проверяет мандат и удостоверение, как уже обсуждалось, а также проверяет адрес клиента и метку времени. Если все в порядке, то сервер уверен, что, согласно Kerberos, клиент - именно тот, за кого он себя выдает.

Если приложение требует взаимной проверки подлинности, сервер посылает клиенту сообщение, состоящее из метки времени, зашифрованной сеансовым ключом. Это доказывает, что серверу известен правильный секретный ключ, и он может расшифровать мандат и удостоверение.

При необходимости клиент и сервер могут шифровать дальнейшие сообщения общим ключом. Так как этот ключ известен только им, они оба могут быть уверены, что последнее сообщение, зашифрованное этим ключом, отправлено другой стороной.

Kerberos версии 4

В предыдущих разделах рассматривался Kerberos версии 5. Версия 4 немного отличается сообщениями и конструкцией мандатов и удостоверений. В Kerberos версии 4 используются следующие пять сообщений:

1. Клиент-Kerberos: *c*,*tgs*
2. Kerberos-клиент: $\{K_{c,tgs}\{T_{c,tgs}\}K_{tgs}\}K_c$,
3. Клиент-TGS: $\{A_{c,s}\}K_{c,tgs}\{T_{c,tgs}\}\ K_{tgs,s}$
4. TGS-клиент: $\{K_{c,s}\{T_{c,s}\}K_s\}K_{c,tgs}$
5. Клиент-сервер: $\{A_{c,s}\}K_{c,s}\ \{T_{c,s}\}K_s$

$$T_{c,s} = \{s, c, a, v, l, K_{c,s}\}K_s$$

$$A_{c,s} = \{c, a, t\}\ K_{c,s}$$

Сообщения 1,3 и 5 не изменились. Двойное шифрование мандата на этапах 2 и 4 в версии 5 было устранено. Мандаты версии 5 дополнительно включают возможность использовать несколько адресов, а поле "время жи зни", *l*, заменено временем начала и окончания. В удостоверение версии пять добавлена возможность включения дополнительного ключа.

Безопасность Kerberos

Стив Белловин (Steve Bellovin) и Майкл Мерритт (Michael Merritt) проанализировали некоторые потенциальные уязвимые места Kerberos [108]. Хотя эта работа была написана про протоколы версии 4, многие ее замечания применимы и к версии 5.

Возможно кэширование и повторное использование старых удостоверений. Хотя метки должны предотвратить иакую возможность, удостоверения могут использоваться повторно в течение времени жизни мандата. Предполагается, что серверы хранят все правильные мандаты, чтобы обнаружить повторы, но это не всегда возможно. Кроме того, время жизни бывает достаточно большим, часто до восьми часов.

Использование удостоверений основаны на том, что все часы сети более или менее синхронизированы. Если время компьютера будет установлено неправильно, то старое удостоверение может быть использовано без проблем. Большинство сетевых протоколов поддержки единого времени небезопасны, поэтому такая возможность представляет собой серьезную проблему.

Kerberos также чувствителен к вскрытиям с угадыванием пароля. Злоумышленник может записать мандаты и затем попытаться их расшифровать. Не забудем, что средний пользователь редко выбирает хороший пароль. Если Мэллори добудет достаточно мандатов, у него появятся неплохие шансы раскрыть пароль.

Возможно самым опасным является вскрытие, использующее специальное программное обеспечение. Протоколы Kerberos подразумевают, что программному обеспечению можно доверять. Нет способа помешать Мэллори исподтишка заменить все клиентское программное обеспечение Kerberos такой версией, которая помимо выполнения протоколов Kerberos записывает пароли. Это является проблемой для любого криптографического программного пакета, работающего на небезопасном компьютере, но широко распространенное использование Kerberos в подобных средах делает его особенно привлекательной мишенью.

Ведутся работы над улучшением Kerberos, включая модернизацию управления ключами с помощью криптографии с открытыми ключами и интерфейса интеллектуальных карточек.

Лицензии

Kerberos не является общедоступным, но код МТИ доступен свободно. Действительная реализация в работающих системах UNIX - это совсем другая история. Ряд компаний продает версии Kerberos, но можно получить хорошую версию бесплатно от Cygnus Support, 814 University Ave., Pale Alto, CA, 94301; (415) 32,2.-3811; fax: (415) 32.2.-3270.

24.6 KRYPTOKNIGHT

KryptoKnight (КриптоРыцарь) является системой проверки подлинности и распределения ключей, разработанной в IBM. Это протокол с секретным ключом, использующий либо DES в режиме CBC (см. раздел 9.3) или модифицированную версию MD5 (см. раздел 18.5). KryptoKnight поддерживает четыре сервиса безопасности:

— Проверка подлинности пользователя (называемая единственной подписью - single sign-on)

— Двусторонняя проверка подлинности

— Распределение ключей

— Проверка подлинности содержания и происхождения данных

С точки зрения пользователя, KryptoKnight похож на Kerberos. Вот некоторые отличия:

— Для проверки подлинности и шифрования мандатов KryptoKnight использует хэш-функцию.

— KryptoKnight не использует синхронизированных часов, используются только текущие запросы (см. раздел 3.3).

— Если Алисе нужно связаться с Бобом, одна из опций KryptoKnight позволяет Алисе послать сообщение Бобу, а затем позволяет Бобу начать протокол обмена ключами.

KryptoKnight, как и Kerberos, использует мандаты и удостоверения. Он содержит и TGS, но в KryptoKnight называются серверами проверки подлинности. Разработчики KryptoKnight потратили немало усилий, минимизируя количество сообщений, их размер и объем шифрования. О KryptoKnight читайте в [1110, 173, 174, 175].

24.7 SESAME

SESAME означает Secure European System for Applications in a Multivendor Environment - Безопасная европейская система для приложений в неоднородных средах. Это проект Европейского сообщества, на 50 процентов финансируемый RACE (см. раздел 25.7), главной целью которой является разработка технологии для проверки подлинности пользователя при распределенном контроле доступа. Эту систему можно рассматривать как европейский вариант Kerberos. Проект состоит из двух частей: на первой стадии разрабатывается базовая архитектура, а вторая стадия представляет собой ряд коммерческих проектов. Следующие три компании принимают наибольшее участие в разработке системы - ICL в Великобритании, Siemens в Германии и Bull во Франции.

SESAME представляет собой систему проверки подлинности и обмена ключами [361, 1248, 797, 1043]. Она использует протокол Needham-Schroeder, применяя криптографию с открытыми ключами для связи между различными безопасными доменами. В системе есть ряд серьезных изъянов. Вместо использования настоящего алгоритма шифрования в этой системе применяется XOR с 64-битовым ключом. Что еще хуже, в SESAME используется XOR в режиме CBC, который оставляет незашифрованным половину открытого текста. В защиту разработчиков надо сказать, что они собирались использовать DES, но французское правительство выразило неудовольствие по этому поводу. Они утвердили код с DES, но затем убрали его. Эта система меня не впечатлила.

Отождествление в SESAME является функцией первого блока, а не всего сообщения. В результате этого тождественность сообщений будет проверена по словам "Dear Sir", а не по всему содержанию сообщений. Генерация ключей состоит из двух вызовов функции rand операционной системы UNIX, которая совсем не случайна. В качестве однонаправленных хэш-функций SESAME использует crc32 и MD5. И конечно, SESAME подобно Kerberos чувствительна к угадыванию паролей.

24.8 Общая криптографическая архитектура IBM

Общая криптографическая архитектура (Common Cryptographic Architecture, CCA) была разработана компанией IBM, чтобы обеспечить криптографические примитивы для конфиденциальности, целостности, управления ключами и обработки персонального идентификационного кода (PIN) [751, 784, 1025, 1026, 940, 752]. Управление ключами происходит с помощью векторов управления (control vector, CV) (см. раздел 8.5). Каждому ключу соответствует CV, с которым ключ объединен операцией XOR. Ключ и CV разделяются только в безопасном аппаратном модуле. CV представляет собой структуру данных, обеспечивающую интуитивное понимание привилегий, связанных с конкретным ключом.

Отдельные биты CV обладают конкретным смыслом при использовании каждого ключа, применяемого в CGA. CV передаются вместе с зашифрованным ключом в структурах данных, называемых ключевыми маркерами (key token). Внутренние ключевые маркеры используются локально и содержат ключи, шифрованные локальным главным ключом (master key, MK). Внешние ключевые маркеры используются для шифрованными ключами между системами. Ключи во внешних ключевых маркерах зашифрованы ключами шифрования ключей (key-encrypting key, KEK). Управление KEK осуществляется с помощью внутренних ключевых маркеров. Ключи разделяются на группы в соответствии с их использованием.

Длина ключа также задается при помощи битов CV. Ключи одинарной длины - 56-битовые - используются для таких функций, как обеспечение конфиденциальности и сообщений. Ключи двойной длины - 112-битовые - применяются для управления ключами, функций PIN и других специальных целей. Ключи могут быть DOUBLE-ONLY (только двойные), правые и левые половины которых должны быть различны, DOUBLE (двойные) половины которых могут случайно совпасть, SINGLE-REPLICATED (одинарные-повторенные), в которых правые и левые половины равны, или SINGLE (одинарные), содержащие только 56 битов. CGA определяет аппаратную реализацию определенных типов ключей, используемых для некоторых операций.

CV проверяется в безопасном аппаратном модуле: для каждой функции CGA вектор должен соответствовать определенным правилам. Если CV успешно проходит проверку, то при помощи XOR KEK или MK с CV получается вариант KEK или MK, и извлеченный ключ для дешифрирования открытого текста сообщения использ уется только при выполнении функции CGA. При генерации новых ключей CV задает способ использования созданного ключа. Комбинации типов ключей, которые могут быть использованы для вскрытия системы, не со здаются в CGA-совместимых системах и не импортир уются в них.

Для распределения ключей CGA применяет комбинацию криптографии с открытыми ключами и криптогр афии с секретными ключами. KDC шифрует сеансовый ключ для пользователя секретным главным ключом, ра зделяемым с этим пользователем. Распределение главных ключей происходит с помощью криптографии с о ткрытыми ключами.

Разработчики системы выбрали такой гибридный подход по двум причинам. Первой из них является эффе ктивность. Криптография с открытыми ключами требует больших вычислительных ресурсов, если сеансовые ключи распределяются с помощью криптографии с открытыми ключами, система может повиснуть. Второй причиной является обратная совместимость, система может быть с минимальными последствиями установлена поверх существующих схем с секретными ключами.

CGA-системы проектировались так, чтобы они могли взаимодействовать с различными другими системами. При контакте с несовместимыми системами функция трансляции вектора управления (Control Vector Translate, CVXLT) позволяет системам обмениваться ключами. Инициализация функции CVXLT требует контроля с обе их сторон. Каждая из них должна независимо установить нужные таблицы трансляции. Такой двойной контроль обеспечивает высокую степень надежности, касающейся целостности и происхождения ключей, импортируемых в систему.

Тип ключа DATA поддерживается для совместимости с другими системами. Ключ типа DATA хранится вместе с соответствующим CV, указывающим, что это ключ типа DATA. Ключи типа DATA могут использоваться достаточно широко, и поэтому к ним нужно относиться с подозрением и использовать их с осторожн остью. Ключи типа DATA нельзя использовать ни для каких функций управления ключ ами.

Аппаратура закрытия коммерческих данных (Commercial Data Masking Facility, CDMF) представляет собой экспортируемую версию CGA. Ее особенностью является уменьшение эффективной длины ключей DES до разрешенных к экспорту 40 битов (см. раздел 15.5) [785].

24.9 Схема проверки подлинности ISO

Для использования в схеме проверки подлинности ISO, также известной как протоколы X.509, рекомендуется криптография с открытыми ключами [304]. Эта схема обеспечивает проверку подлинности по сети. Хотя конкретный алгоритм не определен ни для обеспечения безопасности, ни для проверки подлинности, специф икация рекомендует использовать RSA. Однако возможно использование нескольких алгоритмов и хэш-функций. Первоначальный вариант X.509 был выпущен в 1988 г. После открытого изучения и комментирования он был пересмотрен в 1993 году, чтобы исправить некоторые изъяны в безопасности [1100, 750].

Версия
Последовательный номер
Идентификатор алгоритма - Алгоритм - Параметры
Выдавшая организация
Время действия - начало действия - гонец действия
Субъект
Открытый ключ субъекта - Алгоритм - Параметры - Открытый ключ

Подпись

Рис. 24-2. Сертификат X.509.

Сертификаты

Наиболее важной частью X.509 используемая им структура сертификатов открытых ключей. Имена всех пользователей различны. Доверенный Орган сертификации (Certification Authority, CA) присваивает каждому пользователю уникальное имя и выдает подписанный сертификат, содержащий имя и открытый ключ пользов ателя. Структура сертификата X.509 показана на 22-й [304].

Поле версии определяет формат сертификата. Последовательный номер уникален для конкретного CA. Следующее поле определяет алгоритм, использованный для подписи сертификата, вместе со всеми необходимыми параметрами. Выдавшей организацией является CA. Срок действия представляет собой пару дат, сертификат действителен в промежутке между этими двумя датами. Субъект - это имя пользователя. Информация об открытом ключе включает название алгоритма, все необходимые параметры и открытый ключ. Последним полем является подпись CA.

Если Алиса хочет связаться с Бобом, она сначала извлекает из базы данных его сертификат и проверяет его достоверность. Если у них общий CA, то все просто. Алиса проверяет подпись CA на сертификате Боба.

Если они пользуются различными CA, то все гораздо сложнее. Представьте себе древовидную структуру, в которой одни CA сертифицируют другие CA и пользователей. На самом верху находится главный CA. У каждого CA есть сертификаты, подписанные вышестоящим CA и нижестоящим CA. При проверке сертификата Боба Алиса использует эти сертификаты.

Такая схема продемонстрирована на 21-й. Сертификат Алисы заверен CA_A, сертификат Боба заверен CA_B. Алиса знает открытый ключ CA_A. У CA_C есть сертификат, подписанный CA_A, поэтому Алиса может проверить это. У CA_C есть сертификат, подписанный CA_D. И сертификат Боба подписан CA_D. Подымаясь по дереву сертификации до общей точки, в данном случае CA_D, Алиса может проверить сертификат Боба.

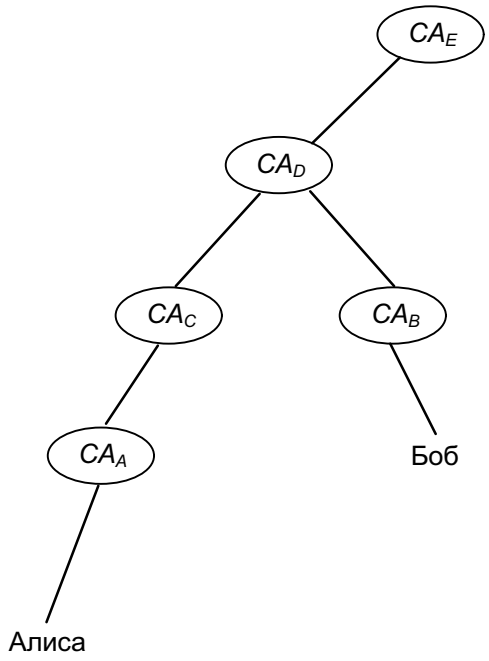


Рис. 24-3. Пример иерархии сертификации.

Сертификаты могут храниться в базах данных на различных узлах сети. Пользователи могут посылать их друг другу. Истечении срока действия сертификата он должен быть удален из всех общедоступных каталогов. Однако CA, выдавший сертификат, должен продолжать хранить его копию, которая может потребоваться при разрешении возможных споров.

Сертификаты также могут быть отозваны, либо из-за компрометации ключа пользователя, либо из-за того, что CA больше не хочет подтверждать сертификат данного пользователя. Каждый CA должен поддерживать список всех отозванных сертификатов, срок действия которых еще не закончился. Когда Алиса получает новый сертификат, она должна проверить, не был ли он отозван. Она может проверить базу данных отозванных ключей по сети, но скорей всего она проверит локально кэшируемый перечень отозванных сертификатов. В такой системе определенно вероятны злоупотребления, отзыв сертификатов возможно является самой слабой частью

этой схемы.

Протоколы проверки подлинности

Алисе нужно связаться с Бобом. Сначала она извлекает из базы данных **последовательность сертификации** от Алисы до Боба и открытый ключ Боба. В этот момент Алиса может инициировать однопроходный, двухпроходный или трехпроходный протокол проверки подлинности.

Однопроходный протокол представляет собой простую передачу данных Бобу Алисой. Протокол устанавливает личности и Алисы, и Боба, а также целостность информации, передаваемой Бобу Алисой. Кроме того, он обеспечивает защиту от вскрытия линии связи с помощью повтора.

В двухпроходном протоколе добавлен ответ Боба. Протокол устанавливает, что именно Боб, а не какой-то самозванец, посылает ответ. Он также обеспечивает безопасность обеих передач и защищает от вскрытия повтором.

И в однопроходных, и в двухпроходных алгоритмах используются метки времени. В трехпроходном протоколе добавляется еще одно сообщение Алисы Бобу и позволяет избежать меток времени (и, следовательно, правильного единого времени).

Однопроходный протокол:

(1) Алиса генерирует случайное число R_A.

(2) Алиса создает сообщение, $M = (T_A, R_A, I_B, d)$, где T_A - метка времени Алисы, I_B - идентификатор Боба, d - произвольные данные. Для безопасности данные могут быть зашифрованы открытым ключом Боба E_B.

(3) Алиса посылает Бобу $(C_A, D_A(M))$. (C_A - это сертификат Алисы, D_A - это общий узел дерева сертификации.)

(4) Боб проверяет C_A и получает E_A. Он проверяет, что срок действия этих ключей еще не истек. (E_A - это открытый ключ Алисы.)

(5) Боб использует E_A для дешифрирования $D_A(M)$. Этим действием он проверяет и подпись Алисы, и целостность подписанной информации.

(6) Боб для точности проверяет I_B в M.

(7) Боб проверяет T_A в M и убеждается, что сообщение является текущим.

(8) Дополнительно Боб может проверить R_A в M по базе данных старых номеров, чтобы убедиться, что сообщение не является повторяемым старым сообщением.

Двухпроходный протокол состоит из однопроходного протокола и последующего аналогичного однопроходного протокола от Боба к Алисе. После выполнения этапов (1)-(8) однопроходного протокола двухпроходный протокол продолжается следующим образом:

(9) Боб генерирует случайное число R_B.

(10) Боб создает сообщение $M' = (T_B, R_B, I_A, R_A, d)$, где T_B - метка времени Боба, I_A- идентификатор Алисы, а d - произвольные данные. Для безопасности данные могут быть зашифрованы открытым ключом Алисы E_A. R_A - случайное число Алисы, созданное на этапе (1).

(11) Боб посылает Алисе sends $D_B(M')$.

(12) Алиса использует E_B, чтобы расшифровать $D_B(M')$. Таким образом одновременно проверяются подпись Боба и целостность подписанной информации.

(13) Алиса для точности проверяет I_A в M'.

(14) Алиса проверяет T_B в M' и убеждается, что сообщение является текущим.

(15) Дополнительно Алиса может проверить R_B в M', чтобы убедиться, что сообщение не является повторяемым старым сообщением.

Трехпроходный протокол решает ту же самую задачу, но без меток времени. Этапы (1) - (15) такие же, как в двухпроходном алгоритме, но $T_A = T_B = 0$.

(16) Алиса сверяет полученную версию R_A с R_A, которое было отправлено Бобу на этапе (3).

(17) Алиса посылает Бобу $D_A(R_B)$.

(18) Боб использует E_A, чтобы расшифровать $D_A(R_B)$. Таким образом одновременно проверяются подпись Алисы и целостность подписанной информации.

(19) Алиса сверяет полученную версию R_B с R_B, которое было отправлено Алисе на этапе (10).

24.10 Почта с повышенной секретностью PRIVACY-ENHANCED MAIL (PEM)

Почта с повышенной секретностью (Privacy-Enhanced Mail, PEM) представляет собой стандарт Internet для почты с повышенной секретностью, одобренный Советом по архитектуре Internet (Internet Architecture Board, IAB) для обеспечения безопасности электронной почты в Internet. Первоначальный вариант был разработан Группой секретности и безопасности (Privacy and Security Research Group, PSRG) Internet Resources Task Force (IRTF), а затем их разработка была передана в Рабочую группу PEM Internet Engineering Task Force (IETF) PEM Working Group. Протоколы PEM предназначены для шифрования, проверки подлинности, проверки цел остности сообщения и управления ключами.

Полностью протоколы PEM сначала были детально описаны в ряде RFC (Requests for Comment, Запросы комментариев) в [977] и затем пересмотрены в [978]. Третья итерация протоколов [979, 827, 980] сведена в [177, 178]. Протоколы были изменены и улучшены, и окончательные протоколы детально описываются в др угом наборе RFC [981, 825, 76, 802]. В другой статье Мэтью Бишопа (Matthew Bishop) [179] подробно описаны все изменения. Попытки реализации PEM рассматриваются в [602, 1505, 1522, 74, 351, 1366, 1367]. См. также [1394].

PEM является расширяемым стандартом. Процедуры и протоколы PEM разработаны так, чтобы быть совместимыми со множеством подходов к управлению ключами, включая симметричную схему и использование открытых ключей для шифрования ключей шифрования данных. Симметричная криптография применяется для шифрования текста сообщений. Для контроля целостности сообщения используются криптографические спос обы хэширования. Другие документы поддерживают механизмы управления ключами с помощью сертификатов открытых ключей, алгоритмов, режимов и связанных идентификаторов, а также и электронные подробности, инфраструктуру и процедуры управления ключами.

PEM поддерживает только определенные алгоритмы, но позволяет добавлять и более поздние алгоритмы. Сообщения шифруются алгоритмом DES в режиме CBC. Проверка подлинности, обеспечиваемая средством **Проверки целостности сообщения** (Message Integrity Check, MIC), использует MD2 или MD5. Симметричное управление ключами может применять либо DES в режиме , либо тройной DES с двумя ключами (так называемый режим EDE). Для управления ключами PEM также поддерживает сертификаты открытых ключей, испол ьзуя RSA (длина ключа до 1024 битов) и стандарт X.509 для структуры сертификатов.

PEM обеспечивает три сервиса повышения секретности: конфиденциальность, проверка подлинности и ко нтроль целостности сообщений. К электронной постовой системе не предъявляется никаких специальных треб ований. PEM может быть встроены выборочно, в определенные узлы или у определенных пользователей, не вл ияя на работу остальной сети.

Документы PEM

PEM определяется в следующих четырех документах:

— RFC 1421: Часть I, Процедуры шифрования и проверки подлинности сообщений. В этом документе опре-
  деляются процедуры шифрования и проверки подлинности сообщений, которые должны обеспечить
  функции почты с повышенной секретностью для передачи электронной почты в Internet.

— RFC 1422: Часть II, Управление ключами с помощью сертификатов. В этом документе определяется ар-
  хитектура и инфраструктура управления ключами, которые основаны на методе сертификатов открытых
  ключей, предоставляющих информацию о ключах отправителям и получателям сообщений.

— RFC 1423: Часть III, Алгоритмы, режимы и идентификаторы. Этот документ содержит определения,
  форматы, ссылки и цитаты для криптографических алгоритмов, режимов использования и связанных
  идентификаторов и параметров.

— RFC 1424: Часть IV, Сертификация ключей и родственные функции. В этом документе описываются три
  типа функций, поддерживаемых PEM: сертификация ключей, хранение и извлечение списка отозванных
  сертификатов (certificate revocation list, CRL).

Сертификаты

PEM совместим со схемой проверки подлинности, описанной в [304], см. также [826]. PEM представляет собой надмножество X.509, определяя процедуры и соглашения для инфраструктуры управления ключами, и спользуемой с PEM и в будущем другими протоколами (включая стеки TCP/IP и OSI).

Инфраструктура управления ключами использует общий корень для всей сертификации Internet. Центр регистрационной политики (Internet Policy Registration Authority, IPRA) определяет глобальную стратегию, приме нимую ко всей иерархии. Ниже корня - IPRA - находятся Центры сертификационной политики (Policy Certification Authorities, PCA), каждый из которых определяет и опубликовывает свою стратегию регистрации пользов ателей и организаций. Каждый PCA сертифицирован IPRA. Следом за PCA идут CA, сертифицирующие пользо-

вателей и и управляющие организационными подразделениями (департаментами, офисами, дочерними компаниями). Первоначально предполагалось, что большинство пользователей будет регистрироваться в качестве членов организаций.

Как ожидается, ряд PCA будет обеспечивать сертификацию пользователей, не входящих ни в одну организацию. Предполагается выделить один или несколько PCA для регистрации пользователей, желающих воспользоваться преимуществами секретности PEM и сохранить анонимность. Стратегия этих PCA будет позволять регистрировать пользователей, не желающих раскрывать свои личности.

Сообщения PEM

Сердцем PEM является формат сообщений. На 20-й показано зашифрованное сообщение при симметричном управлении ключами. На 19-й показано подписанное и зашифрованное сообщение при управлении ключами на базе открытых ключей, и на Figure 24.6 показано подписанное (но незашифрованное) сообщение при управлении ключами на базе открытых ключей.

```
   -----BEGIN PRIVACY-ENHANCED MESSAGE-----
   Proc-Type: 4,ENCRYPTED
   Content-Domain: RFC822
   DEK-Info: DES-CBC,F8143EDE5960C597
   Originator-ID-Symmetric: schneler@counterpane.com,,
   Recipient-ID-Symmetric: schneler@chinet.com,ptf-kmc,3
   Key-Info                                                                :
DES-ECB,RSA-MD2,9FD3AAD2F2691B9A,B70665BB9BF7CBCDA60195DB94F727D3
   Recipient-lD-Symmetric: penl-dev@tis.com,ptf-kmc,4
   Key-Info :
DES-ECB,RSA-MD2,161A3F75DC82EF26,E2EF532C65CBCFF79F83A2658132DB47
LLrHBOeJzyhP+/fSStdH8okeEnv47jxe7SJ/iN72ohNcUk2jHEUSoHlnvNSIHE9M
8tEjmF/zxB+bATMtPjCUHbz8Er9wloxIkjHUlBEpvXROUrUzYbkNpkOagV2IzUpk
J6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz5rDqUcMlKlZ6720dcBHGGsDLpTpSCnpot
   dXd/H5LMDHnonNvPCmQUHt==
   -----END PRIVACY ENHANCED MESSAGE-----
```

Рис. 24-4. Пример встроенного сообщения (симметричный случай)

```
   -----BEGIN PRIVACY-ENHANCED MESSAGE-----
   Proc-Type: 4,ENCRYPTED
   ContentDomain: RFC822
   DEK-Info: DESCBC,BFF968AA74691AC1
   0riginator - Certificate :
MIIBlTCCAScCAWuwDQYJKoZIhvcNAQECBQAwUTELMAkGAlUEBhMCVVMxIDAeBgNV
BAoTFIJTQSBEYXRhIFNIY3VyaXR5LCBJbmMuMQ8wDQYDVQQLEmZCZXRhIDExDzAN
BgNVBAsTBk5PVEESWTAeFw05MTA5MDQxODM4MTdaFmO5MzA5MDMxODM4MTZaMEUx
CzAJBgNVBAYTAlVTMSAmHgYDVQQKExdSUOEgRGFOYSBTZWNlcmlOeSwgSW5jLjEU
MBIGAIUEAxMLVGVzdCBVc2VyIDEmHTAKBgRVCAEBAglCAANLADBIAkEAwHZHI7i+
yJcqDtjJComzTdBJrdAiLAnSC+CnnjOJEEyuQiBgkGrgIh3j8/xOfM+YrsyFlu3F
LZPVtzlrldhYFJQI DAQABMAOGCSqGSIb3DQEBAgUAAIkACKrOPqphJYwlj+YPtc
iWlFPuN5jJ79Khfg7ASFxskYkEMjRNZV/HZDZQEhtVaU7Jxfzs2mfX5byMp2X3U/
   5XIJXGx7qlJsDgHQGs7Jk9H8CHlfuSHUgN4w==
   Key-Info: RSA,
I3rRIGXUGWAF8js5wCzRTkdh034PTHdRZY9Tuvm03M+NM7fx6qc5uIxps2LrlgO+
   wGrtiUm/ovtKdlnzeZQ/aQ==
   Issuer-Certificate:
MIIB3DCCAUgCAQowDQYJKoZIhvcNAQECBQAwTzELMAkGAIUEBhMCVVMxIDAeBgNV
BAoTFIJTQSBEYXRhIFNIY3VyaXR5LCBJbmMuMQ8mDQYDVQQEEwZCZXRhIDExDTAE
BgNVBAsTBFRMQOEwHhcNOTEmOTAxMDgwMDAwWhcNOTlwOTAxMDclOTU5HjBRMQsw
CQYDVQQGEwJVUzEgMB4GAIUEChMXUINBIERhdGEgU2VjdXJpdHksIEIuYy4xDzAN
BgNVBAsTBkJIdGEgMTEPMAOGAIUECxMGTk9UQVJZMHAmCgYEVQgBAQICArmDYgAm
XwJYCsnp61QCxYykNIODwutF7jMJ3kE+3PjYyHOwk+79rLg6X65B/ED4bJHt05XH
cqAz/7R7XhjYCmOPcqbdzoACZtIIETrKrcJiDYoP+DkZ8klgCk7hQHpbImIDAQAB
MAOGCSqGSIb3DQEBAgUAA38AAICPv4f9Gx/tY4+p+4DB7MV+tKZnvBoy8zgoMGOx
dD2jMZ/3HsyWKHgSFOeH7AJB3qr9zosG47pyMnTf3aSy2nBO7CMxpUHRBcXUpE+x
EREZd9++32ofGBIXaIaInOgVUnOOzSYgugiQO77nJLDUjOhQehCizEs5wUJ35a5h
   MIC-Info: RSA-MD5,RSA,
UdFJR8u/TIGhfH651eeme210H4tooa3vZCvVNGBZirf/7nrgzHDABz8m9NsXSexv
   AjRFbHoNPzBuxwnlOAFeAOHJszE4yBvhG
   Recipient-ID-Asymmetric :
MFExCzAJBgNVBAYTAIVTMSAmHgYDVQQKExdSUOEgRGFOYSBTZHNlcmIOeSwgSH5j
   LjEPMAOGAIUECxMGQftiVOYSAxMQ81tfDQYDVQQLEwZOTIRBUIk=,
   66
   Key-Info: RSA,
06BSIww9CTyHPtS3bMLD+EOhejdvX6QvlHK2ds2sQPEaXhX8EhvVphHYTjmekdHv
   7xOZ3Jx2vTAhOYHMcqqCjA==
qeWlj/YJ2Uf5ng9yznPbtDOmYloSwIuV9FRYx+gzY+81Xd/NQrXHfi6/MhPfPF3d
   jIqCJAxvld2xgqQimUzoSla4r7kQQ5c/Iua4LqKeq3clFzEv7MbZhA==
   -----END PRIVACY ENHANCED MESSAGE-----
```

Рис. 24-5. Пример встроенного шифрованного (ENCRYPTED) сообщения (асимметричный случай).

Первым полем является "Proc-Type", идентификатор типа обработки, которой подверглось сообщение. Существует три возможных типа сообщений. Спецификатор "ENCRYPTED" обозначает, что сообщение зашифро-

вано и подписано. Спецификатор "MIC-ONLY" и "MIC-CLEAR" указывают, что сообщение подписано, но не зашифровано. Сообщения MIC-CLEAR не кодируются и могут быть прочитаны с помощью другого, не входящего в PEM программного обеспечения. Для преобразования сообщений MIC-ONLY в удобочитаемую форму необходимо программное обеспечение PEM. Сообщение PEM подписывается всегда, а шифрование не является обязательным.

Следующее поле, "Content-Domain", задает тип почтового сообщения. Оно не влияет на безопасность. Поле "DEK-Info" содержит информацию о **ключе обмена данными** (Data Exchange Key, DEK), алгоритме, используемом для шифрования текста, и параметрах, связанных с алгоритмом шифрования . В настоящее время определен единственный алгоритм - DES в режиме CBC, "DES-CBC" Второе подполе содержит IV. В будущем для PEM могут быть определены и другие алгоритмы, их использование будет запротоколировано в поле DEK-Info и других полях, определяющих алгоритм.

В сообщениях с симметричным управлением ключами (см. 20th) следующим полем будет "Originator-ID-Symmetric" с тремя подполями. Первое подполе с помощью уникального адреса электронной почты определяет отправителя. Второе поле не является обязательным и определяет орган, выдавший заменяемый ключ . Третьим является необязательное подполе Версия/Окончание срока .

Далее, при использовании симметричного управления ключами, у каждого получателя есть два поля : "Recipient-ID-Symmetric" и "Key-Info." Поле "Recipient-ID-Symmetric" содержит три подполя, которые определяют получателя также, как подполя поля "Originator- ID-Symmetric" определяют отправителя.

Поле "Key-Info" задает параметры управления ключами. У этого поля четыре подполя. Первое определяет алгоритм, использованный для шифрования DEK. Так как в рассматриваемом сообщении применяется симметричное управление ключами, то отправитель и получатель используют общий ключ . Он называется **заменяемым ключом** (Interchange Key, IK) и используется для шифрования DEK. DEK может быть зашифрован либо с помощью DES в режиме ECB (этот способ обозначается "DES-ECB"), либо тройным DES ("DES-EDE"). Второе подполе определяет алгоритм MIC. Может использоваться MD2 (обозначается "RSA-MD2") или MD5 ("RSA-MD5"). Третье подполе, DEK, и четвертое подполе, MIC, шифруются с помощью IK.

На 19-й и 18-й показаны сообщения, в которых используется управление ключами с помощью открытых ключей (в перечне PEM такой способ называется асимметричным). Заголовки изменяются. В сообщениях ENCRYPTED после поля "DEK-Info" идет поле "Originator-Certificate". Форма сертификата соответствует стандарту X.509 (см. раздел 24.9). Следующим полем является "Key-Info" с двумя подполями. Первое подполе определяет алгоритм с открытым ключом, использованный для шифрования DEK, в настоящее время поддерживается только RSA. Следующее подполе - DEK, зашифрованный открытым ключом отправителя. Это необязательное поле, которое позволяет отправителю расшифровать свое собственное сообщение, возвращенное почтовой системой. Следующим полем является "Issuer-Certificate", сертификат организации, подписавшей сертификат отправителя ("Originator-Certificate").

Далее при асимметричном управлении ключами следует поле "MIC-Info". Первое подполе задает алгоритм вычисления MIC, а второе - алгоритм, использованный для подписи MIC. Третье подполе содержит MIC, подписанный закрытым ключом отправителя.

```
   -----BEGIN PRIVACY-ENHANCED MESSAGE-----
   Proc-Type: 4,MIC-ONLY
   Content-Domain: RFC822
   0riginator - Certificate :
MIIBITCCAScCAHUwDQYJKoZIhvcNAQECBQAwTzELMAkGAIUEBhMCVVMxlDAeBgNV
BAoTFIJTQSBEYXRhIFNIY3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDzAN
BgNVBAsTBk5PVEFSHTAeEw05MTA5MDQxODM4MTdaFw05MzA5MDMxODM4MTZaMEUx
CzAJBgNVBAYTAIVTMSAwHgYDVQQKExdSUOEgRGFOYSBTZMNlcmlOeSwgSW5jLjEU
MEIGAIUEAxMLVGVzdCBVc2VylDEwHTAKBgRVCAEBAgICAANLADBIAkEAmHZHI71+
yJcqDtjJCowzTdBJrdAiLAnSC+CnnjOJELyuQiBgkGrgIh3j8/xOfM+YrsyFlu3F
LZPVtzlndhYFJQIDAQABMAOGCSqGSIb3DQEBAgUAAIkACKrOPqphJYwlj+YPtcIq
ItJIFPuN5jJ79Khfg7ASFxskYkEMjRNZV/HZDZQEhtVaU7Jxfzs2mfX5bMp2X3U/
   5XUXGx7qusDgHQGs7Jk9W8CW1fuSI4UgN4w==
   Issuer-Certificate:
MIIB3DCCAUgCAQowDQYJKoZIhvcNAQECBQAkTzELMAkGAIUEBhMCVVMxlDAeBgNV
BAoTFIJTQSBEYXRhIFNIY3VyaXR5LCBJbmMuMQ8wDQYDVQQLEIZCZXRhIDExDTAE
BgNVBAsTBFRMQOEwHhcNOTEwOTAxMDgwMDAmkmcNOTImOTAxMDclOTU5HjBRMQsw
CQYDVQQGEwJVUzEgMB4GAIUEChMXUINBIERhdGEgU2VjdXJpdHksIEIuYywxDzAN
BgNVBAsTBkJIdGEgMTEPMAOGAIUECxMGTk9UQVJZMHAwCgYEVQgBAQICArwDYgAw
XwJYCsnpelQCxYkNIODwlltF/jMJ3kE+3PjYjHOwk+/9rEg6X65B7LD4bJHt05XH
cqAz77R7XhjYCmOPcqbdzoACZIlETrKrcJIDYoP+DkZ8klgCk7hQHpbltflDAQAB
MAOGCSqGSIb3DQEBAgUAA38AAICPv4f9Gx7tY4+p+4DB7MV+tKZrlvBo8zgoMGOx
dD2jMZ73Hs*kl4gSFOeH7AJB3qr9zosG47pyMnTf3aS2nBO7CMxpUkfRBcXUpE+x
EREZd9++32otGBIXaIalnOgVUnOOzSYgljglQO77nJEDUOhQehCizEs5mUJ35a5h
   MIC-Info: RSA-MD5,RSA,
jV20fH+nnXHU8bnE8kPAad7mSQITDZIbVuxvZAOVRZ5q5+EjI5bQvqNeqOUNQjr6
   EtE7K2QDeVMCj/XsdJIA8fA==
LSBBIGIIc3NhZ2UgZrti9lHVzZSBpbBOZXNOaH5nLgOKESBGb2xsb3dpbmcgaXMg
   YSBibGFuaj/Bsakf51OgOKDQpUaGIzIGIzIHRoZSBIbrnQuDQo=
   -----END PRIVACY-ENHANCED MESSAGE-----
```

Рис. 24-6. Пример встроенного MIC-ONLY сообщения (асимметричный случай).

Следующие поля связаны с получателями. Каждому получателю соответствуют два поля : "Recipient-ID-Asymmetric" и "Key-Info". У поля"Recipient-ID-Asymmetric" два подполя. Первое определяет орган, выдавший открытый ключ получателя, а вторым является необязательное подполе Версия/Окончание срока . Поле "Key-Info" задает параметры управления ключами: первое подполе определяет алгоритм, использованный для ши фрования сообщения, а вторым подполем служит DEK, зашифрованный открытым ключом получателя.

Безопасность PEM

Длина ключей RSA, используемых в PEM, может меняться в диапазоне от 508 до 1024 битов. Этого достаточно практически для любого уровня безопасности. Более вероятно, что вскрытие будет направлено против протоколов управления ключами. Мэллори может украсть ваш закрытый ключ - не записывайте его нигде - или попытаться подсунуть вам фальшивый открытый ключ. Процедуры сертификации ключей в PEM делают это невозможным, если все пользователи строго следуют соответствующим процедурам , но, как известно, люди часто неаккуратны.

Мэллори может поступить хитрее и модифицировать реализацию PEM, работающую в вашей системе. Эта измененная версия может тайком пересылать Мэллори всю вашу почту, зашифровав ее его открытым ключом. Ему может быть послана даже копия вашего закрытого ключа . Если измененная реализация будет работать х орошо, то вы никогда не узнаете, что случилось.

Реального способа предотвратить такое вскрытие не существует. Вы можете использовать однонаправле нную хэш-функцию и получить контрольную сумму исполняемого кода PEM. Затем, при каждом запуске программного обеспечения вы можете проверять контрольную сумму, чтобы вовремя обнаружить изменения . Но Мэллори точно также может изменить и код контрольной суммы при изменении кода PEM. Можно сохранить контрольную сумму контрольной суммы, но Мэллори может изменить и ее. Если у Мэллори есть доступ к вашему компьютеру, он может разрушить безопасность PEM.

Мораль в том, что вы не должны доверять никакому элементу программного обеспечения, если вы не мож ете доверять аппаратуре, на которой работает это программное обеспечение . Для большинства такие опасения покажутся необоснованными. Но для некоторых людей они вполне реальны .

TIS/PEM

Доверенные информационные системы (TIS, Trusted Information Systems), частично поддерживаемые Управлением по передовым научным проектам правительства Соединенных Штатов , включают реализацию PEM (TIS/PEM). Разработанные для платформ UNIX, они были также перенесены на VMS, DOS и Windows.

Хотя спецификации PEM определяют для Internet один главный сертификационный центр, TIS/PEM поддерживает существование нескольких иерархий сертификации. Узлы могут определить набор сертификатов, которые будут считаться действительными, включая все сертификаты, выданные узлами. Для того, чтобы пользоваться TIS/PEM узлу не нужно присоединяться к иерархии Internet.

Все организации и граждане США и Канады при желании могут получить доступ к TIS/PEM, которая распространяется в виде исходного кода. Заинтересованные лица должны обращаться по следующему адресу: Privacy-Enhanced Mail, Trusted Information Systems, Inc., 3060 Washington Road IRte. 97), Glenwood, MD 2,1738; (301) 854-6889; fax: (301) 854-5363; Internet: pern-info@tis.com.

RIPEM

RIPEM - это программа, написанная Марком Риорданом (Mark Riordan) и реализующая протоколы PEM. Хотя эта программа не является свободно доступной, ей можно воспользоваться бесплатно для частного, некоммерческого использования. Лицензия на ее использование входит в документацию.

Код не может быть экспортирован. Конечно, законы правительства США не действуют за пределами Соединенных Штатов, и ряд людей игнорирует экспортные ограничения. Код RIPEM доступен по всему миру на электронных досках объявлений. Разрешена для экспорта версия, называемая RIPEM/SIC, реализующая только цифровые подписи.

К моменту написания этих строк RIPEM не полностью реализовала протоколы PEM, в ней нет возможности использовать сертификаты проверки подлинности ключей.

До RIPEM Риордан написал похожую программу RPEM. Подразумевалось, что это будет общедоступная программа электронной почты. Пытаясь обойти патентные проблемы, Риордан использовал алгоритм Rabin (см. раздел 19.5). Public Key Partners заявила, что их патенты распространяются на всю криптографию с открытыми ключами. Под угрозой судебного процесса Риордан прекратил распространение программы.

Сейчас RPEM не используется. Она не совместима с RIPEM. Так как можно использовать RIPEM, не встречая препятствий со стороны Public Key Partners, нет повода возвращаться к RPEM.

24.11 Протокол безопасности сообщений

Протокол безопасности сообщений (Message Security Protocol, MSP) - это военный эквивалент PEM. Он был разработан NSA в конце 80-х годов при работе по программе создания Безопасной системы передачи данных по сети (Secure Data Network System, SDNS) program. Это совместимый с X.400 протокол уровня приложения для закрытия электронной почты. MSP планируется использовать в разрабатываемой сети оборонных сообщений (Defense Message System, DMS) Министерства обороны.

Предварительный протокол безопасности сообщений (Preliminary Message Security Protocol, PMSP), который предполагается использовать для "несекретных, но важных" сообщений, представляет собой адаптированную для использования с X.400 и TCP/IP версию MSP. Этот протокол также называют Mosaic.

Как и PEM, программные реализации MSP и PMSP достаточно гибки, их конструкция позволяет подстроиться под использование различных алгоритмов для осуществления функций безопасности, таких как подпись, хэширование и шифрование. PSMP будет работать с микросхемой Capstone (см. раздел 24.17).

24.12 PRETTY GOOD PRIVACY (PGP)

Pretty Good Privacy (PGP, весьма хорошая секретность) - это свободно распространяемая программа безопасной электронной почты, разработанная Филипом Циммерманном (Philip Zimmermann) [1652]. Для шифрования данных она использует IDEA, для управления ключами и цифровой подписи - RSA (длина ключа до 2047 битов), а для однонаправленного хэширования - MD5.

Для получения случайных открытых ключей PGP использует вероятностную проверку чисел на простоту, используя для получения стартовых последовательностей интервалы между нажатиями пользователем клавиш на клавиатуре. PGP генерирует случайные ключи IDEA с помощью метода, в ANSI X9.17, Appendix C (см. раздел 8.1) [55], используя вместо DES в качестве симметричного алгоритма IDEA. PGP также шифрует закрытый ключ пользователя с помощью хэшированной парольной фразы, а не пароля непосредственно.

Сообщения, зашифрованные PGP, имеют несколько уровней безопасности. Единственная вещь, известная криптоаналитику о зашифрованном сообщении, - это получатель сообщения при условии, что криптоаналитику известен ID ключа получателя. Только расшифровав сообщение, получатель узнает, кем оно подписано, если оно подписано. Это резко отличается от сообщения PEM, в заголовке которого немало информации об отправителе, получателе и самом сообщении хранится в незашифрованном виде.

Самой интересной особенностью PGP является распределенный подход к управлению ключами (см. раздел 8.12). Центров сертификации ключей нет, вместо этого в PGP поддерживается "сеть доверия". Каждый пользователь сам создает и распространяет свой открытый ключ. Пользователи подписывают ключи друг друга, создавая взаимосвязанное сообщество пользователей PGP.

Например, Алиса может физически передать Бобу свои открытый ключ. Боб лично знает Алису, поэтому он

подписывает ее открытый ключ. Одну подписанную копию он возвращает Алисе, а другую оставляет. Когда Алисе нужно связаться с Кэрол, она посылает Кэрол подписанную Бом копию ключа. Кэрол, у которой каким-то образом уже есть ключ Боба (она получила его раньше), и которая доверяет Бобу заверить ключ другого человека, проверяет его подпись под ключом Алисы и убеждается, что она правильна. Таким образом, Боб знакомит Алису и Кэрол.

PGP не определяет стратегию установки доверительных связей, пользователи сами решают, кому верить, а кому нет. PGP обеспечивает механизмы для поддержки ассоциативного доверия открытым ключам и для использования доверия. Каждый пользователь хранит набор подписанных открытых ключей в виде файла **кольца открытых ключей** (public-key ring). Каждый ключ кольца обладает полем законности ключа, определяющим уровень доверия к ключу конкретного пользователя. Чем больше уровень доверия, тем больше пользователь уверен в законности ключа. Поле доверия к подписи измеряет, насколько пользователь верит тому, кто подписал открытые ключи других пользователей. И наконец поле доверия к владельцу ключа задает уровень, определяющий, насколько конкретный пользователь верит владельцу ключа, подписавшему другие открытые ключи. Это поле вручную устанавливается пользователем. PGP непрерывно обновляет эти поля по мере появления новой информации.

На 17-й показано, как выглядит эта модель для конкретного пользователя, Алисы. Ключ Алисы находится в самом верху иерархии, владелец ключа абсолютно надежен. Алиса подписывает ключи Боба, Кэрол, Дэйва, Элен и Фрэнка. Она доверяет Бобу и Кэрол подписывать открытые ключи других людей, кроме того, она частично доверяет Дэйву и Элен подписывать открытые ключи других людей. И она доверяет Гейл подписывать открытые ключи других людей, хотя сама не подписывала ключ Гейл.

Двух частично доверяемых подписей может оказаться достаточным для сертификации ключа. Алиса считает, что ключ Курта законен, так как Дэйв и Элен подписали его. Уровень доверия устанавливается в PGP вручную, Алиса может выбрать устраивающую ее степень паранойи.

Алиса не должна автоматически доверять ключам других людей только потому, что они подписаны ключом, который она считает правильным. Алиса Она не доверяет Фрэнку Она подписывать другие ключи, хотя она собственноручно подписывала его ключ. Кроме того, она не доверяет подписи Ивана под ключом Мартина или подписи Курта под ключом.

Ключ Оуэна вообще на входит в сеть, может быть, Алиса получила его от сервера. PGP не считает ключ автоматически правильным, Алиса должна либо объявить о правильности ключа, либо решиться поверить одному из тех, кто подписал ключ.

Конечно, ничто не мешает Алисе использовать ключи, которым она не доверяет. Задача PGP - предупредить Алису о подозрительности ключа, а не помешать ей устанавливать соединения.

Самым слабым звеном этой системы является отзыв ключей: гарантировать, что кто-нибудь не воспользуется скомпрометированным ключом, невозможно. Если закрытый ключ Алисы украден, она может послать некий **сертификат отзыва ключа** (key revocation certificate), но, так как некое распределение ключей уже произошло, нельзя гарантировать, что это сообщение будет получено всеми, использующими ее открытый ключ в своем кольце ключей. И так как Алиса должна будет подписать свой сертификат отзыва ключа своим закрытым ключом, то если она потеряет ключ, она не сможет и отозвать его.

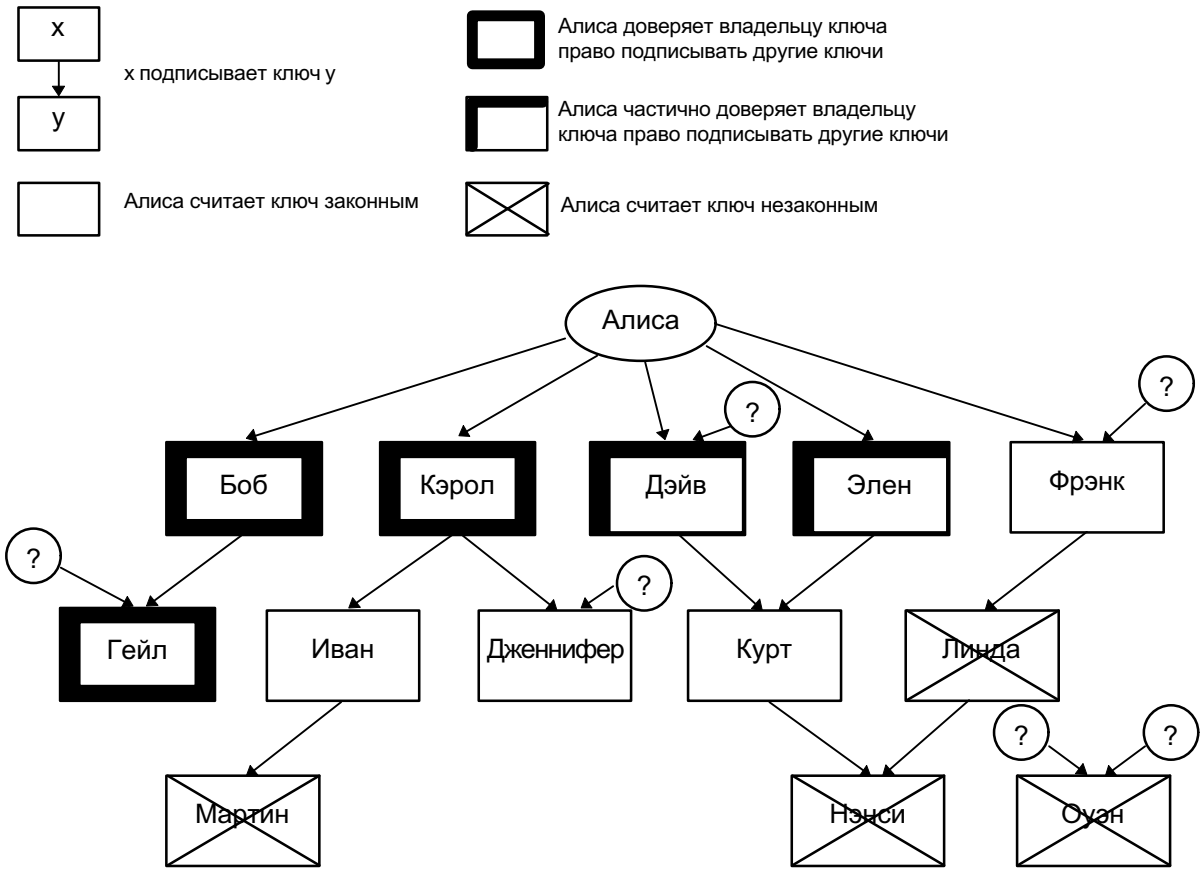


Рис. 24-7. Модель доверия в PGP.

Текущей версией PGP является 2.6.2. Появление новой версии, PGP 3.0, ожидается к концу 1995 года. В 3.0 включены опции тройного DES, SHA, другие алгоритмы с открытыми ключами, разделение пар "открытый ключ/закрытый ключ" для шифрования и для подписи, расширенные процедуры отзыва ключей, улучшенные функции управления кольцом ключей, API для интегрирования PGP в другие программы и полностью переписанные исполняемые модули.

PGP доступна для MS-DOS, UNIX, Macintosh, Amiga и Atari. В личных, некоммерческих целях ее можно использовать свободно, скачав со многих узлов ftp в Internet. Чтобы скопировать PGP с узла MIT с помощью telnet подключитесь к net-dist.mit.edu, войдите в систему как getpgp, ответьте на вопросы, затем используйте ftp для соединения с net-dist.mit.edu и перейдите в каталог, указанный в сессии telnet. Эту программу также можно получить ftp.ox.ac.uk, ftp.dsi.unimi.it, ftp.funet.fi, ftp.demon.co.uk, CompuServe, AOL, и т.п. Для коммерческого использования в США PGP можно приобрести - полностью, вместе с лицензиями - примерно за 100 долларов в компании ViaCrypt, 9033 N 24th Ave., Phoenix, AZ, 85021; (602) 944-0773; viacrypt@acm.org. Существуют различные средства, помогающие интегрировать PGP в MS-DOS, Microsoft Windows, Macintosh и UNIX.

О PGP написано несколько книг [601,1394,1495]. Исходный код был даже опубликован в печатном виде в [1653] при попытке обойти Госдепартамент США, который продолжает считать, что исходный код можно экспортировать только в бумажном, а не в электронном виде. Если вы доверяете IDEA, PGP позволит вам приблизиться к военному уровню шифрования.

24.13 Интеллектуальные карточки

Интеллектуальная карточка представляет собой пластиковую карточку, по размеру и форме как кредитная карточка, с встроенной компьютерной микросхемой. Идея стара - первые патенты были выданы лет 20 тому назад - но из-за практических ограничений возможность реализовать такие карточки появилась только приме рно пять лет назад. С тех пор они стали популярны, главным образом в Европе. Во многих странах интеллектуальные карточки используются для оплаты за телефоны. Существуют интеллектуальные кредитные карточки, интеллектуальные дебитные карточки, интеллектуальные карточки для чего угодно. Американские компании по выпуску кредитных карточек работают над технологией, и через несколько лет даже захудалые американцы будут носить интеллектуальные карточки в своих бумажниках.

Интеллектуальная карточка содержит маленький компьютер (обычно 8-битовый микропроцессор), ОЗУ (четверть килобайта), ПЗУ (примерно 6-8 килобайт), и несколько килобайт либо EPROM (стираемое программируемое ПЗУ) или EEPROM (электронно стираемое программируемое ПЗУ). Объем памяти в интеллектуаль-

ных карточках следующего поколения наверняка возрастет, но определенные физические ограничения затру днят такие расширения. Карточка использует свою операционную систему, программы и данные . (Чего в ней нет, так это источника питания, электроэнергия подается, когда карточку вставляют в считыватель .) Карточка безопасна. В нашем меняющемся мире, когда нельзя доверять чужому компьютеру, телефону, да чему угодно, вы можете быть уверены в своей карточке, которая хранится в вашем бумажнике .

В интеллектуальных карточках могут использоваться различные криптографические протоколы и алгоритмы. Они могут быть электронным кошельком, давая возможность тратить и получать электронные н аличные. Карточки могут использоваться в протоколах проверки подлинности с нулевым знанием, они могут обладать собственными ключами шифрования. Возможно, они позволяют подписывать документы или снимать блокировку с компьютерных программ.

Некоторые интеллектуальные карточки считаются устойчивыми к взлому, таким образом себя часто защ ищают организации, эмитировавшие карточки. Банк вовсе не хочет, чтобы вы могли влезть в их интеллектуал ьную карточку и начислить себе побольше денег.

Интеллектуальные карточки - это очень интересная тема, на которую написано множество литературы . Хорошей обзорной статьей по криптографии в интеллектуальных карточках может служить [672]. Ежегодно проводятся конференции: CARTES в октябре в Париже и CardTech в апреле в Вашингтоне, округ Колумбия. Труды двух других конференций по интеллектуальным карточкам можно найти в [342, 382]. В области интеллектуальных карточек существуют сотни патентов, частью принадлежащие европейским компаниям. Интересные вопросы будущего использования интеллектуальных карточек - проверка целостности, аудиторский контроль, защита от копирования, электронные наличные, оплата почтовых расходов - описаны в [1628].

24.14 Стандарты криптографии с открытыми ключами

Стандарты криптографии с открытыми ключами (Public-Key Cryptography Standards, PKCS) - это попытка компании RSA Data Security, Inc обеспечить промышленный стандарт для криптографии с открытыми ключами. По традиции такими делами занимался ANSI, но, учитывая текущую ситуацию в криптографической политике, RSADSI решила, что лучше они все сделают сами. Работая со множеством компаний, RSADSI разработала набор стандартов. Некоторые из них совместимы с другими стандартами, а некот орые - нет.

Эти стандарты не являются стандартами в общепринятом смысле этого слова, никто не собирался и не гол осовал за PKCS. По своим собственным словам RSADSI "будет единственной организацией, правомочной пр инимать решения о каждом стандарте, и будет пересматривать эти стандарты по мере необходимости " [803].

Даже это уже совсем. Если вы неуверены, какие структуры данных и синтаксис использовать при програ ммировании криптографии с открытыми ключами, эти стандарты не хуже каких-либо других. К тому же, так как это не настоящие стандарты, вы можете подстроить их под свои нужды .

Далее приведено краткое описание каждого PKCS (PKCS #2 и PKCS #4 были включены в PKCS #l).

PKCS #l [1345] описывает способ шифрования и дешифрирования RSA, главным образом для создания ци фровых подписей и цифровых конвертов, описанных в PKCS #7. Для цифровых подписей сообщение хэшируется, а затем хэш-значение шифруется закрытым ключом подписывающего. Совместное представление сообщения и хэш-значения подробно описано в in PKCS #7. Для цифровых конвертов (шифрованные сообщения) сообщение сначала шифруется симметричным алгоритмом, а затем ключ сообщений шифруется открытым ключом получ ателя. Совместное представление шифрованного сообщения и шифрованного ключа должно соответствовать PKCS #7. Эти два метода совместимы со стандартами PEM. Для структуры сертификатов (или их подобия) о ткрытых и закрытых ключей RSA и трех алгоритмов подписи - MD2 и RSA, MD4 и RSA, MD5 и RSA - PKCS #l также описывает синтаксис, идентичный синтаксису X.509 и PEM.

PKCS #3 [1346] описывает способ реализации обмена ключами по схеме Diffie-Hellman.

PKCS #5 [1347) описывает способ шифрования сообщений секретным ключом, полученным из пароля. Стандарт использует MD2 или MD5 для получения ключа из пароля и шифрует сообщения с помощью DES в режиме CBC. Этот метод предназначен главным образом для шифрования закрытых ключей при их передаче от одной компьютерной системы другой, но может быть использован и для шифрования сообщений .

PKCS #6 [1348] описывает стандартный синтаксис сертификатов открытых ключей. Синтаксис является надмножеством сертификата X.509, при необходимости можно извлечь и сертификат X.509. Дополнительные атрибуты не ограничивают процесс сертификации только открытым ключом. Они содержат и другую информ ацию, например, адрес электронной почты.

PKCS # 7 [1349] представляет собой общий синтаксис для подписываемых или шифруемых данных, напр имер, цифровых подписей или цифровых конвертов. Синтаксис является рекурсивным, поэтому можно организ овать вложенность конвертов или поставить чью-то подпись под ранее зашифрованными данными. Синтаксис также разрешает вместе с содержанием сообщения проверку подлинности других атрибутов, например, меток

времени. PKCS #7 с PEM, поэтому подписанные и зашифрованные сообщения могут быть преобразованы в с о-общения PEM, и наоборот, без дополнительных криптографических операций. Для управления ключами с по-мощью сертификатов PKCS #7 может поддерживать множество архитектур - одной из них является PEM.

PKCS #8 [1350] описывает синтаксис информации о закрытых ключах, включая закрытый ключ и набор а т-рибутов, и синтаксис шифрованных закрытых ключей. Для шифрования информации о закрытых ключах мож-но использовать PKCS #5.

PKCS #9 [1351] определяет избранные типы атрибутов для расширенных сертификатов PKCS #6, сообщений с цифровой подписью PKCS #7 и информации о закрытых ключах PKCS #8.

PKCS #10 [1352,] описывает стандартный синтаксис запросов сертификации. Сертификация включает инди-видуальное имя, открытый ключ и (необязательно) набор атрибутов, которые подписаны лицом, приславшим запрос. Запросы сертификации присылаются в сертифицирующий орган, который преобразует запрос либо в сертификат открытого ключа X.509, либо в сертификат PKCS #6.

PKCS #11 [1353], Стандарт API криптографической метки (Cryptographic Token API Standard), определяет интерфейс программирования, называемый "Cryptoki", для портативных криптографических устройств всех типов. Cryptoki представляет собой обобщенную логическую модель, позволяющую приложениям выполнять криптографические операции на портативных устройствах, не зная деталей используемой технологии. Этот стандарт также определяет профили приложения: наборы алгоритмов, которые может поддерживать устройство.

PKCS #12 [1354] описывает синтаксис хранения в программном обеспечении открытых ключей пользоват е-лей, защищенных закрытых ключей, сертификатов и другой связанной криптографической информации. Целью этого является стандартизация единого фала ключей, используемого многими приложениями.

Эти стандарты всесторонни, но не всеобъемлющи. Многие вопросы остались за пределами этих стандартов: проблема присвоения имен, некриптографические вопросы, касающиеся сертификации, длины ключей и усл о-вия для различных параметров. PKCS призваны обеспечить формат передачи данных, основанной на крипт о-графии с открытыми ключами, и инфраструктуру, поддерживающую такую передачу.

24.15 Универсальная система электронных платежей

Универсальная система электронных платежей (Universal Electronic Payment System, UEPS) представляет собой банковское приложение, использующее интеллектуальные карточки, первоначально разработанное для сельской Южной Африки, но позднее принятое основными банковскими группами этой страны. К началу 1995 года в ЮАР было выпущено около 2 миллионов карточек. Эта система также принята в Намибии, и разверт ы-вается по крайней мере одним российским банком.

Система позволяет использовать безопасные дебитные карточки, подходящие для регионов, в которых пл о-хая телефонная сеть делает невозможной диалоговую проверку. Карточки есть и покупателей, и у продавцов, покупатели могут использовать свои карточки для перевода денег продавцам. Продавец может воспользоваться своей карточкой, чтобы позвонить в банк и поместить деньги на свой банковский счет, покупатель может во с-пользоваться своей карточкой, чтобы позвонить в банк и перевести деньги на свою карточку. Нет необходимо-сти заботиться об анонимности, нужно обеспечить только защиту от мошенничества.

Вот как выглядит протокол связи между покупателем Алисой и продавцом Бобом (В действительности, Али-са и Боб просто вставляют свои карточки в машину и ожидают выполнения транзакции.) Когда Алиса впервые получает свою карточку, она получает и пару ключей, K_1 и K_2, банк вычисляет их, используя ее имя и некот о-рую секретную функцию. Только в карточки продавцов встроены секретные средства, необходимые для вычи с-ления ключей пользователей.

(1) Алиса посылает Бобу свое имя, A, его имя, B, и случайное число R_A, шифруя их с помощью DES: сначала ключом K_2, затем K_1. Она также посылает свое имя открытым текстом.

$$A, E_{K_1}(E_{K_2}(A,B,R_A))$$

(2) Боб вычисляет K_1 и K_2 по имени Алисы. Он расшифровывает сообщение, убеждается, что A и B правиль-ны, затем шифрует незашифрованную вторую половину сообщения Алисы ключом K_2.

$$E_{K_2}(A,B,R_A)$$

Боб не посылает это сообщение Алисе, 56 битов шифротекста становятся ключом K_3. Боб посылает Алисе свое имя, ее имя и случайное число, R_B, шифруя их с помощью DES: сначала ключом K_3, затем K_1.

$$E_{K_1}(E_{K_3}(B,A,R_B))$$

(3) Алиса аналогичным образом вычисляет K_3 и расшифровывает сообщение Боба, убеждаясь, что A и B пра-

вильны, затем шифрует незашифрованную вторую половину сообщения Боба ключом K_3.

$$E_{K_3}(B,A,R_B)$$

Алиса не посылает это сообщение Бобу, 56 битов шифротекста становятся ключом K_4. Затем Алиса посылает Бобу свое имя, его имя проверочное значение C. Это проверочное значение содержит имена отправителя и получателя, дату, контрольную сумму, количество и два MAC. Все это шифруется DES: сначала ключом K_4, затем K_1. Один из MAC может быть проверен банком Алисы, а второй может быть проверен только расчетно-кассовым центром. Алиса уменьшает свой счет на соответствующее значение.

$$E_{K_1}(E_{K_4}(A,B,C))$$

(4) Боб аналогичным образом вычисляет K_4. При условии, что все имена совпадают, и правильно выполнена проверка, он принимает платеж.

Великолепным нововведением в этом протоколе является то, что каждое сообщение зависит от предыдущего. Каждое сообщение выступает удостоверением *всех* предыдущих сообщений. Это означает, что повторить старое сообщение никому не удастся, получатель просто никогда не расшифрует его. Мне нравится эта идея, и я уверен, что она получит широкое применение, как только станет широко известна.

Другой разумной вещью в этом протоколе - навязывание правильной реализации. Если разработчик приложения неправильно реализует протокол, он просто не будет работать.

Обе карточки сохраняют записи каждой транзакции. Когда карточки рано или поздно установят диалоговое соединение с банком (продавец - положить деньги на счет, а покупатель - снять со счета), банк извлечет эти записи для последующего контроля.

Аппаратура изготавливается устойчивой к взлому, чтобы помешать любому из участников испортить данные. Алиса не сможет изменить значение своей карточки. Подробная запись обеспечивает данные для обнаружения и запрещения мошеннических транзакций. В карточках используются универсальные секреты - ключи MAC в карточках покупателей, функции для преобразования имен пользователей в K_1 и K_2 - но считается, что решение обратной задачи для этих секретов достаточно трудно.

Эта схема, конечно же, несовершенна, но она безопаснее бумажных чеков и обычных дебитных карточек. Источником угрозы мошенничества являются не военные враги, а покупатели и продавцы. UEPS предоставляет защиту от таких злоупотреблений.

Обмен сообщения является прекрасным примером устойчивого протокола: В каждом сообщении присутствую имена обеих сторон, включая информацию, уникальную для сообщения, каждое сообщение явным образом зависит от всех предыдущих.

24.16 CLIPPER

Микросхема Clipper (известная также как MYK-78T) - это разработанная в NSA, устойчивая к взлому микросхема, предназначенная для шифрования переговоров голосом. Это одна из двух схем, реализующих правительственный Стандарт условного шифрования (Escrowed Encryption Standard, EES) [1153]. VLSI Technologies, Inc. изготовила микросхему, а Mykotronx, Inc. запрограммировала ее. Сначала все микросхемы Clipper будут входить в Безопасное телефонное устройство Model 3600 AT&T (см. раздел 24.18). Микросхема реализует алгоритм шифрования Skipjack (см. раздел 13.12,), разработанный NSA секретный алгоритм с шифрованием секретным ключом, только в режиме OFB.

Самым противоречивым моментом микросхемы Clipper, и EES в целом, является протокол условного вручения ключей (см. раздел 4.14). У каждой микросхемы есть специальный, ненужный для сообщений, ключ. Этот ключ используется для шифрования копии ключа сообщений каждого пользователя. В ходе процесса синхронизации передающая микросхема Clipper генерирует и посылает принимающей Поле доступа для выполнения закона (Law Enforcement Access Field, LEAF). LEAF содержит копию текущего сеансового ключа, зашифрованного специальным ключом (называемым **ключом модуля**). Это позволяет правительственным прослушивателям получить сеансовый ключ и раскрыть открытый текст разговора.

По словам директора NIST [812]:

Предусматривается, что система "с условно врученным ключом" обеспечит использование микросхемы Clipper для защиты законопослушных американцев. В каждом устройстве, содержащем микросхему будет два уникальных "ключа", два числа, которые понадобятся уполномоченным правительственным органам для дешифрирования сообщений, зашифрованных устройством. При изготовлении устройства оба ключа будут помещены порознь в двух базах данных " условно врученных ключей", контролируемых Генеральным прокурором. Доступ к этим ключам будет разрешен только правительственным чиновникам с законным разрешением подключить подслушивающее устройство.

Правительство также собирается поощрять широкое распространение таких телефонных аппаратов, но никто не знает, что может произойти с базами данных условно врученных ключей.

Помимо политических аспектов, стоит поговорить и о внутренней структуре LEAF [812, 1154, 1594, 459, 107, 462]. LEAF - это строка, включающая достаточно информации, чтобы при обеспечении правопорядка можно было раскрыть сеансовый ключ K_s при условии, что два **условно получивших ключи учреждения** будут действовать сообща. LEAF содержит 32-битовый идентификатор модуля U, уникальный для каждой микросхемы Clipper. Оно также содержит текущий 80-битовый сеансовый ключ, зашифрованный уникальным ключом модуля микросхемы K_U, и 16-битовую контрольную сумму C, называемую идентификатором условного вручения. Контрольная сумма представляет собой функцию сеансового ключа, IV и возможно другой информации. Эти три поля шифруются фиксированным общим ключом K_F, общим для всех взаимодействующих микросхем Clipper. Общий ключ, используемые режимы шифрования, детали контрольной суммы и точная структура LEAF засекречены. Возможно это поле похоже на что-то подобное:

$$E_{K_F}(E_{K_U}(K_s, U, C))$$

K_U вводится в микросхемы Clipper при изготовлении. Этот ключ затем разделяется (см. раздел 3.5) и хранится в двух базах данных условно врученных ключей, охраняемых двумя различными учреждениями.

Чтобы Ева могла извлечь K_s из LEAF, она должна сначала расшифровать LEAF ключом K_F и получить U. Затем она должна получить постановление суда для каждого из учреждений условного вручения, каждое из которых возвращает половину K_U для данного U. Ева выполняет XOR обеих половин и получает K_U, затем она использует K_U для получения K_s, и K_s - для подслушивания разговора.

Контрольная сумма должна помешать нарушению этой схемы, принимающая микросхема Clipper не может выполнить дешифрирование, если контрольная сумма неправильна. Однако существует лишь 2^{16} возможных значений контрольной суммы, и фальшивое LEAF с правильной контрольной суммой, но неправильным ключом, может быть найдено примерно за 42 минуты [187]. Но это не очень поможет подслушать разговор, ведущийся с помощью Clipper. Так как протокол обмена ключами не является частью микросхемы Clipper, 42-минутное вскрытие грубой силой должно быть выполнено после обмена ключами, оно не может быть выполнено до телефонного звонка. Такое вскрытие может работать при передаче факсов или при использовании карточки Fortezza (см. раздел 24.17).

Предположительно микросхема Clipper должна противостоять инженерному вскрытию, выполненному "изощренным, хорошо" [1154], но по слухам в Sandia National Laboratories успешно провели исследование одной из микросхем. Даже если эти слухи ложны, я подозреваю, что самым крупным мировым производителям такое инженерное вскрытие вполне по силам, и его срок является только вопросом ресурсов и морали.

С этой темой связано множество вопросов о тайне личности. Многочисленные группы защиты гражданских свобод ведут активную компанию против любого механизма условного вручения ключей, который даст правительству право подслушивать граждан. Вся подлость в том, что, ходя эта схема никогда не проходила через Конгресс, NIST опубликовал EES в качестве FIPS [1153], обойдя болезненный законодательный процесс. Сейчас все выглядит, как если бы EES тихо и медленно умирал, но стандарты способны продолжать свою ползучую деятельность.

В 22-й перечислены различные организации, участвующие в этой программе. Как насчет идеи, чтобы оба учреждения условного вручения относились только к исполнительной ветви власти? Что вы скажете об учреждениях условного вручения, которые по сути ничего не знают о заявках на подслушивание и могут только слепо одобрять их? И что насчет идее о принятии правительством секретного алгоритма в качестве коммерческого стандарта?

Табл. 24-2.
Организации, участвующие в EES.

Министерство юстиции - Спонсор системы, владелец общего ключа
NIST - Руководство программой, хранитель условно врученной части ключа
FBI - Пользователь-дешифровщик, владелец общего ключа
Министерство финансов - Хранитель условно врученной части ключа
NSA - Разработчик программы

В любом случае, использование Clipper породит немало проблем при обращении в суд. Не забывайте, Clipper работает только в режиме OFB. Что бы вам иное не говорили, этот режим не обеспечивает целостности или проверке подлинности. Предположим, что Алиса предстала перед судом, и частью доказательств является телефонный разговор, зашифрованный микросхемой Clipper. Алиса утверждает, что она никогда не звонила, и голос - не ее. Алгоритм сжатия речи настолько плох, что опознать голос Алисы трудно, но обвинение утверждает, что, так как расшифровать разговор можно только с помощью условно врученного ключа Алисы, этот звонок был

сделан с ее телефона.

Алиса заявляет, что разговор был подделан в соответствии с [984, 1339]: даны шифротекст и открытый текст, объединив их с помощью XOR, можно получить ключевой поток. Затем этот ключевой поток можно объединить с помощью XOR с абсолютно другим открытым текстом, получая фальшивый шифротекст, который затем может быть преобразован в фальшивый открытый текст, который подается на дешифратор микросхемы. Правдив он или нет, этот довод может легко посеять сомнение в жюри присяжных, которые не сочтут телефонный разговор доказательством.

Другой способ вскрытия, называемый Втискиванием (Squeeze), позволяет Алисе выдать себя за Боба. Вот как это происходит [575]: Алиса звонит Бобу, используя Clipper. Она сохраняет копию его LEAF вместе с сеансовым ключом. Затем она звонит Кэрол (про которую известно, что ее подслушивают). При установке ключа Алиса делает сеансовый ключ идентичным тому, который она использовала для разговора с Бобом. Для этого потребуется взломать телефон, но это нетрудно. Затем вместо того, чтобы послать свое LEAF, она посылает LEAF Боба. Это правильное LEAF, поэтому телефон Кэрол ничего не заметит. Теперь она может говорить Кэрол все, что захочет - когда полиция расшифрует LEAF, она обнаружит, что оно принадлежит Бобу. Даже если Алисе не удастся выдать себя за Боба, ему придется доказывать свою невиновность в суде, что вполне может оправдать применение подобной схемы.

Органы охраны правопорядка Соединенных Штатов не должны тратить свое время, занимаясь сбором информации в уголовных расследованиях, которую нельзя использовать в суде. Даже если условное вручение ключей и являлось бы неплохой идеей, Clipper - это не лучший способ реализации этой идеи.

24.17 CAPSTONE

Capstone (известный также как MYK-80) - это другая разработанная NSA СБИС, реализующая Стандарт условного шифрования правительства США [1153]. Capstone реализует следующие функции [1155, 462]:

— Алгоритм Skipjack в любом из четырех основных режимов: ECB, CBC, CFB и OFB.

— Алгоритм обмена ключами (Key Exchange Algorithm, KEA) на базе открытых ключей, скорее всего Diffie-Hellman.

— Алгоритм цифровой подписи (Digital Signature Algorithm, DSA). *

— Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA). j

— Алгоритм возведения в степень для общего назначения.

— Генератор случайных чисел с использованием истинно шумового источника.

Capstone обеспечивает криптографические возможности, необходимые для безопасной электронной торговли и других компьютерных приложений. Первым применением Capstone является карточка PCMCIA, названная Fortezza. (Сначала она называлась Tessera, пока на это не пожаловалась компания Tessera, Inc..)

NSA изучило возможность удлинения контрольной суммы LEAF в Capstone в версиях для карточек для того, чтобы помешать ранее рассмотренному вскрытию LEAF. Вместо этого была добавлена возможность выполнять перезапуск карточки после 10 неправильных LEAF. Меня это не впечатлило - время поиска правильного LEAF только на 10 процентов, до 46 минут.

24.18 Безопасный телефон AT&T MODEL 3600 TELEPHONE SECURITY DEVICE (TSD)

Безопасный телефон AT&T (Telephone Security Device, TSD) - это телефон с микросхемой Clipper. На самом деле существует четыре модели TSD. Одна содержит микросхему Clipper, другая - экспортируемый фирменный алгоритм шифрования AT&T третья - фирменный алгоритм для использования внутри страны плюс экспортируемый алгоритм, а четвертая включает Clipper, внутренний и экспортируемый алгоритмы.

Для каждого телефонного звонка TSD используют отличный сеансовый ключ. Пара TSD генерирует сеансовый ключ с помощью схемы обмена ключами Diffie-Hellman, независящей от микросхемы Clipper. Так как Diffie-Hellman не включает проверки подлинности, TSD использует два метода для предотвращения вскрытия "человек в середине".

Первым является экран. TSD хэширует сеансовый ключ и выводит хэш-значение на маленьком экране в в иде четырех шестнадцатиричных цифр. Собеседники проверяют, что на их экраны выведены одинаковые цифры. Качество голоса достаточно хорошо, чтобы они могли узнать друг друга по голосу.

Все же Ева может вскрыть эту схему. Пусть ей удалось вклиниться в линию между Бобом и Алисой. Она использует TSD на линии с Алисой и модифицированный TSD на линии с Бобом. Посередине она сопрягает два

телефонных звонка. Алиса пытается сделать разговор безопасным. Она обычным образом генерирует ключ, но общается с Евой, выдающей себя за Боба. Ева раскрывает ключ и с помощью модифицированного TSD делает так, чтобы ключ, который она сгенерировала для Боба, имел такое же хэш-значение. Это вскрытие на вид не очень реально, но для его предотвращения в TSD используется блокировка.

TSD генерирует случайные числа, используя источник шума и хаотичный усилитель с цифровой обратной связью. Он генерирует битовый поток, который пропускается через постотбеливающий фильтр на базе цифрового процессора.

Несмотря на все это в справочном руководстве TSD нет ни слова о безопасности. На самом деле там написано [70]:

> AT&T не гарантирует, что TSD защитит от вскрытия зашифрованной передачи правительственным учреждением, его агентами или третьей стороной. Более того, AT&T не гарантирует, что TSD защитит от вскрытия передаваемой информации с помощью методов, обходящих шифрование.

Глава 25
Политика

25.1 Агентство национальной безопасности (NSA)

NSA - это Агентство национальной безопасности (National Security Agency, когда-то расшифровывалось шутниками как "No Such Agency" (никакое агентство) или "Never Say Anything" (никогда ничего не скажу), но теперь они более открыты), официальный орган правительства США по вопросам безопасности. Агентство было создано в 1952 году президентом Гарри Труменом в подчинении Министерства безопасности, и многие годы в секрете хранилось сам факт его существования. NSA воспринималось как электронная разведка, в его задачи входило подслушивать и расшифровывать все иностранные линии связи в интересах Соединенных Штатов.

Следующие абзацы взяты из оригинального положения о NSA, подписанного в 1952 году президентом Труменом и рассекреченного спустя много лет [1535]:

> В задачи COMINT Агентства национальной безопасности (NSA) должны входить эффективные организация и управление разведывательной деятельности Соединенных Штатов в области телекоммуникаций, проводимой против иностранных правительств, чтобы обеспечить целостную действенную политику и соответствующие меры. Используемый в этой директиве термин "электронная разведка" ("communications intelligence") или "COMINT" обозначает все действия и методы, используемые для перехвата телекоммуникаций, исключая зарубежные прессу и радиовещание, и получения информации, предназначенной для приема другим получателем, но исключает цензуру, а также производство и распространение полученной разведывательной информации.
>
> Специальная природа действий COMINT требует, чтобы они во всех отношениях проводились отдельно от другой или общей разведывательной деятельности. Приказы, директивы, указания или рекомендации любого органа исполнительной власти, касающиеся сбора, получения, безопасности, обработки, распространения или использования разведывательной информации неприменимы в отношении действий COMINT, если это не оговорено особо, и документы не будут изданы компетентным представителем агентства, входящим в правительство. Другие директивы Национального совета безопасности директору ЦРУ и связанные директивы, изданные директором ЦРУ, не должны применяться к действиям COMINT, если это не будет специальная директива Национального совета безопасности, касающаяся COMINT.

NSA ведет исследования в области криптологии, занимаясь как разработкой безопасных алгоритмов для защиты коммуникаций Соединенных Штатов, так и криптоаналитические методы для прослушивания коммуникаций за пределами США research. Известно, что NSA является крупнейшим в мире работодателем для математиков. Оно также является крупнейшим в мире покупателем компьютерной аппаратуры. Возможно криптографический опыт NSA на много лет оторвался от состояния дел в открытой науке (в части алгоритмов, но вряд ли в части протоколов). Несомненно Агентство может взломать многие из используемых сегодня систем. Но, из соображений национальной безопасности, почти вся информация о NSA - даже ее бюджет - засекречена. (По слухам бюджет Агентства составляет около 13 миллиардов долларов в год - включая военное финансирование проектов NSA и оплату персонала - и, по слухам, в нем работает 16 тысяч человек.)

NSA использует свою власть, чтобы ограничить открытую доступность криптографии и помешать национальным врагам использовать слишком сильные методы шифрования, чтобы Агентство могло их взломать. Джеймс Массей (James Massey) анализирует эту борьбу между научными и военными исследованиями в криптографии [1007]:

> Если считать, что криптология является прерогативой правительства, то, конечно, большая часть криптологических исследований должна вестись за закрытыми дверями. Без всякого сомнения количество людей, занятых сегодня криптологическими исследованиями, намного больше, чем количество людей, работающих в открытой криптологии. Открытые криптологические исследования широко ведутся только последние 10 лет. Между этими двумя исследовательскими сообществами были и будут конфликты. Открытые исследования представляют собой обычный поиск знания, для которого жизненно важен открытый обмен идеями с помощью конференций, презентаций и публикаций в научных журналах. Но может ли правительственная организация, ответственная за вскрытие шифров других государств, приветствовать публикацию шифра, который нельзя взломать? Может ли исследователь с чистой совестью публиковать подобный шифр, которые может свести на нет все усилия взломщиков кода, находящихся на службе его правительства? Можно настаивать, что публикация доказано безопасного шифра заставит все правительства вести себя подобно "джентльменам" Стимсона, но необходимо помнить, что открытые исследования в криптографии полны политических и этических мотивов гораздо более серьезных, чем во многих других областях науки. Удивляться надо не тому, что правительственные организации на почве криптологии конфликтуют с независимыми исследователями, а тому, что эти конфликты (по крайней мере те, о которых нам известно) так незначительны и так сглажены.

Джеймс Бэмфорд (James Bamford) написал увлекательную книгу про NSA: *The Puzzle Palace* [79], (*Дворец головоломок*), недавно доработанную вместе с Вэйной Медсен (Wayne Madsen) [80].

Коммерческая программа сертификации компьютерной безопасности

Коммерческая программа сертификации компьютерной безопасности (Commercial COMSEC Endorsement Program (CCEP)), кодовое имя Overtake, - это предложение, сделанное NSA в 1984 году и призванное облегчить разработку компьютеров и средств связи с встроенными криптографическими возможностями [85, 1165]. Обычно всю разработку таких изделий оплачивали военные, и это обходилось им недешево. NSA посчитало, что если компании могут продавать аппаратуру и армии, и корпорациям, даже иностранным, это позволит уменьшить расходы к всеобщей выгоде. Агентству больше не требовалось бы проверять совместимость обору-

дования с Федеральным стандартом 102.7, и затем CCEP предоставила бы доступ к одобренному правительством криптографическому оборудованию [419].

NSA разработало ряд криптографических модулей различного назначения. В этих модулях для различных приложений используются различные алгоритмы, и производители получают возможность извлечь один модуль и вставить другой в зависимости от желаний клиента. Существуют модули для военного использования (Тип I), модули для "несекретного, но важного" правительственного использования (Тип II), модули для корпоративного использования (Тип III) и модули для экспортирования (Тип IV). Различные модули, их применение и названия сведены в 24-й.

Табл. 25-1.
Модули CCEP

Применение	Тип I	Тип II
Речь/низкоскоростная передача данных	Winster	Edgeshot
Компьютер	Tepache	Bulletproof
Высокоскоростная передача данных	Foresee	Brushstroke
Следующее поколение	Countersign I	Countersign II

Эта программа все еще действует, но она не вызвала энтузиазма ни у кого кроме правительства. Все модули были защищены от вскрытия, все алгоритмы были засекречены, а пользователи должны были получать ключи от NSA. Корпорации никогда реально не верили в идею использования секретных алгоритмов, навязанных правительством. Казалось бы, NSA получило заметный урок, чтобы больше не докучать применением Clipper, Skipjack и микросхем шифрования с условным вручением ключей.

25.2 Национальный центр компьютерной безопасности (NCSC)

Национальный центр компьютерной безопасности (National Computer Security Center, NCSC), отделение NSA, отвечает за доверенную правительственную компьютерную программу. В настоящее время центр проводит оценку продуктов компьютерной безопасности (программных и аппаратных), финансирует исследования и публикует их резаультаты, разрабатывает технические руководства и обеспечивает общую поддержку и обучение.

NCSC издает скандально известную "Оранжевую книгу" [465]. Ее настоящее название - *Department of Defense Trusted Computer System Evaluation Criteria* (Критерии оценки департамента оборонных доверенных компьютерных систем), но это так трудно выговаривать, и к тому же у книги оранжевая обложка. Оранжевая книга пытается определить требования к безопасности, дает производителям компьютеров объективный способ измерить безопасность их систем и указывает им, что необходимо встраивать в безопасные продукты. Книга посвящена компьютерной безопасности, о криптографии в ней по сути говорится не очень много.

Оранжевая книга определяет четыре широких категории защиты безопасности. В ней также определяются классы защиты внутри некоторых из этих категорий. Они сведены в 23-й.

Табл. 25-2.
Классификация Оранжевой книги

D: Minimal Security (Минимальная безопасность)
C: Discretionary Protection (Защита по усмотрению)
    C1: Discretionary Security Protection (Защита безопасности по усмотрению)
    C2: Controlled Access Protection (Защита управляемого доступа)
B: Обязательная защита
    B1: Labeled Security Protection
    B2: Structured Protection (Структурная защита)
    B3: Security Domains (Области безопасности)
A: Verified Protection (Достоверная защита)
    A1: Verified Design (Достоверная разработка)

Иногда производители любят говорить "мы обеспечиваем безопасность C2". В виду они имеют классификацию Оранжевой книги. За более подробной информацией обращайтесь к [1365]. Модель компьютерной безопасности, используемая в этих критериях, называется моделью Bell-LaPadula [100, 101, 102, 103].

NCSC издал целую серию книг по компьютерной безопасности, иногда называемую Радугой книг (все обложки имеют различные цвета). Например, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria* [1146] (Интерпретация критериев оценки доверенных компьютерных систем в отношении

доверенных сетей), иногда называемая Красной книгой, толкует положения Оранжевой книги по отношению к сетям и сетевому оборудованию. *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria* [1147] (Интерпретация критериев оценки доверенных компьютерных систем в отношении систем управления базами данных) - я даже не пытаюсь описать цвет обложки - делает то же самое для баз данных. Сегодня существует свыше 30 таких книг, цвет обложек некоторых из них отвратителен.

За полным комплектом Радуги книг обращайтесь по адресу Director, National Security Agency, INFOSEC Awareness, Attention: C81, 9800 Savage Road, Fort George G. Meade, MD 2,0755-6000; (301) 766-8729. Не говорите им, что вас послал я.

25.3 Национальный институт стандартов и техники

NIST - это Национальный институт стандартов и техники (National Institute of Standards and Technology), подразделение Министерства торговли США. Ранее он назывался Национальным бюро стандартов (NBS, National Bureau of Standards) и изменил имя в 1988 году. С помощью своей Лаборатории компьютерных систем (Computer Systems Laboratory, CSL), NIST продвигал открытые стандарты взаимодействия, которые, как он надеялся, ускорят развитие основанных на компьютерах отраслях промышленности. К настоящему времени NIST выпустил стандарты и руководства, которые, как он считает, будут приняты всеми компьютерными системами Соединенных Штатов. Официальные стандарты опубликованы как издания FIPS (Федеральные стандарты обработки информации.

Если вам нужны копии любого из FIPS (или других изданий NIST), свяжитесь с Национальной службой технической информации Министерства торговли США - National Technical Information Service (NTIS), U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161; (703) 487-4650; или посетите gopher://csrc.ncsl.nist.go*

Когда в 1987 году Конгресс принял Акт о компьютерной безопасности (Computer Security Act), NIST был уполномочен определять стандарты, обеспечивающие безопасность важной, но не секретной информации в правительственных компьютерных. (Секретная информация и данные Предупреждающей поправки находятся в сфере юрисдикции NSA.) Акт разрешает NIST в ходе оценки предлагаемых технических стандартов сотрудничать с другими правительственными организациями и частными предприятиями.

NIST издает стандарты криптографических функций. Организации правительства США обязаны использовать их для важной, но несекретной информации. Часто эти стандарты принимаются и частным сектором. NIST выпустил DES, DSS, SHS и EES.

Все эти алгоритмы разработаны с некоторой помощью NSA, начиная от анализа DES до проектирования DSS, SHS и алгоритма Skipjack в EES. Некоторые критикуют NIST за то, что NSA в большой степени может контролировать эти стандарты, хотя интересы NSA могут не совпадать с интересами NIST. Неясно, как действительно NSA может повлиять на проектирование и разработку алгоритмов. Но при ограничениях на персонал, бюджет и ресурсы NIST привлечение NSA кажется разумным. NSA обладает большими возможностями, включая лучшую в мире компьютерные средства.

Официальный "Меморандум о взаимопонимании" ("Memorandum of Understanding", MOU) между двумя организациями гласит:

> МЕМОРАНДУМ О ВЗАИМОПОНИМАНИИ МЕЖДУ ДИРЕКТОРОМ НАЦИОНАЛЬНОГО ИНСТИТУТА СТАНДАРТОВ И ТЕХНИКИ И ДИРЕКТОРОМ АГЕНТСТВА НАЦИОНАЛЬНОЙ БЕЗОПАСНОСТИ ОТНОСИТЕЛЬНО ПРИМЕНЕНИЯ ПУБЛИЧНОГО ЗАКОНА 100-235
>
> Сознавая, что:
>
> A. В соответствии с разделом 2 Акта о компьютерной безопасности от 1987 года (Публичный закон 100-235), (Акт), на Национальный институт стандартов и техники (NIST) как часть Федерального правительства возлагается ответственность за:
>
> 1. Разработку технических, административных, физических стандартов, стандартов управления и руководств для рентабельных безопасности и защищенности важной информации Федеральных компьютерных систем, определенных в Акте; и,
>
> 2. Разработку руководств по технической безопасности соответствующих компьютерных систем Агентства национальной безопасности (NSA).
>
> B. В соответствии с разделом 2 Акта NIST обязан работать в тесном взаимодействии с другими организациями, включая NSA, обеспечивая:
>
> 1. Максимальное использование всех существующих и планируемых программ, материалов, исследований и отчетов, касающихся безопасности и защищенности компьютерных систем, чтобы избежать недужного и дорогого дублирования работ; и,
>
> 2. Эти стандарты, разработанные NIST в соответствии с Актом, в максимально возможной степени должны быть согласованы и совместимы со стандартами и процедурами, разработанными для защиты секретной информации в Федеральных компьютерных системах.
>
> C. В соответствии с Актом в обязанности Министра торговли, которые он перепоручает NIST, входит назначение членов Консультативного комитета по безопасности и защищенности компьютерных систем (Computer System Security and Privacy Advisory Board), по крайней мере члена, представляющего NSA.
>
> Следовательно, для обеспечения целей данного MOU Директор NIST и Директор NSA настоящим признают следующее:
>
> I. NIST будет:
>
> 1. Назначать в Консультативный комитет по безопасности и защищенности компьютерных систем по крайней мере одного представителя, замещающего Директора NSA.

2. Опираться на разработанные NSA руководства по технической безопасности компьютерных систем до той степени, в которой NIST определяет, что эти руководства отвечают требованиям, предъявляемым к защите важной информации в Федеральных компьютерных системах.

3. Признавать сертифицированный NSA рейтинг доверенных систем в соответствии с Программой критериев оценки безопасности доверенных компьютеров без дополнительной экспертизы.

4. Разрабатывать стандарты безопасности телекоммуникаций для защиты важных несекретных компьютерных данных, максимально опираясь на результаты экспертизы и разработки Агентства национальной безопасности, чтобы выполнять эти обязанности своевременно и эффективно.

5. По возможности избегать дублирования, разграничив совместные работы с NSA для получения помощи NSA.

6. Запрашивать помощи NSA по всем вопросам, связанным с криптографическими алгоритмами и криптографическими методами, включая исследования, оценку разработки, одобрение, но не ограничиваясь этими действиями.

II. NSA будет:

1. Обеспечивать NIST техническими руководствами по доверенным технологиям, безопасности телекоммуникаций и идентификации личности, которые могут быть использованы в рентабельных системах защиты важных компьютерных данных.

2. Проводить или инициировать исследовательские и проектные программы по доверенным технологиям, безопасности телекоммуникаций, криптографическим методам и методам идентификации личности.

3. По просьбам NIST оказывать помощь в отношении всех вопросов, связанных с криптографическими алгоритмами и криптографическими методами, включая исследования, оценку разработки, одобрение, но не ограничиваясь этими действиями.

4. Устанавливать стандарты и одобрять изделия для применения в безопасных системах, охватываемых 10 USC раздел 2315 (Поправка Уорнера).

5. По требованию федеральных организаций, их подрядчиков и других финансируемых правительством субъектов проводить оценку возможности вражеской разведывательной деятельности в отношении федеральных информационных систем, а также обеспечивать техническое содействие и рекомендовать изделия, одобренные для применения в безопасных системах, чтобы противостоять такой угрозе.

III. NIST и NSA будут:

1. Координировать свои планы по обеспечению безопасности и защищенности компьютерных систем, за которые NIST и NSA несут ответственность в соответствии с разделом 6(b) Акта.

2. Обмениваться техническими стандартами и руководствами, если это необходимо для достижения целей Акта.

3. Совместно работать над достижением целей этого меморандума с максимальной эффективностью, избегая ненужного дублирования усилий.

4. Поддерживать непрерывный диалог, гарантирующий, что каждая из организаций будет находиться на одинаковом уровне современных технологий и вопросов, влияющих на безопасность автоматизированных информационных компьютерных систем.

5. Организовывать техническую рабочую группу для обзора и анализа областей совместных интересов, касающихся защиты систем, обрабатывающих важную или другую несекретную информацию. Эта Группа будет состоять из шести федеральных служащих, по трое от NIST и NSA, и при необходимости может быть увеличена за счет представителей других организаций. Темы работы группы могут определяться либо заместителем директора NSA по информационной безопасности, либо заместителем директора NIST, либо могут инициироваться самой группой с последующим одобрением заместителем директора NSA по информационной безопасности или заместителем директора NIST. В течение нескольких дней после постановки перед Группой вопроса либо заместителем директора NSA по информационной безопасности, либо заместителем директора NIST Группа должна представить отчет о выполнении работ по этому вопросу и, при необходимости, план дальнейшего анализа.

6. На ежегодной основе обмениваться планами работы по всем исследовательским и конструкторским проектам, связанным с защитой систем, обрабатывающих важную или другую несекретную информацию, включая доверенные технологии, защиту целостности и доступности данных, безопасности телекоммуникаций и методов идентификации личности. Обмен информацией по проектам должен происходить ежеквартально, и обзор состояния проектов должен любой из сторон предоставляться по запросу другой стороны.

7. Проверять обзоры технической рабочей группы до опубликования всех вопросов, касающихся техники обеспечения безопасности систем, разрабатываемых для использования при защите важной информации в федеральных компьютерных системах, чтобы гарантировать совместимость раскрытия этих тем с национальной безопасностью Соединенных Штатов. Если NIST и NSA не смогут решить подобный вопрос в течение 60 дней, любая из организаций может поднять этот вопрос перед Министром обороны и Министром торговли. Признается, что данный вопрос с помощью NSC может быть передан для решения Президенту. Никакие действия не должны предприниматься до окончательного решения в опроса.

8. Определять дополнительные рабочие соглашения, заключенные между NSA и NIST, как приложения к этому MOU.

IV. Любая из сторон может прекратить действие этого MOU письменным уведомлением, направленным за шесть месяцев до прекращения действия. Этот MOU считается действительным при наличии обеих подписей.

/подписано/

РЭЙМОНД. ДЖ. КАММЕР
Исполнительный Директор, Национальный институт стандартов и техники, 24 марта 1989 года

У. О. СТЬЮДМЕН
Вице-адмирал, ВМС США, Директор, Агентство национальной безопасности, 23 марта 1989 года

25.4 RSA Data Security, Inc.

RSA Data Security, Inc. (RSADSI) была основана в 1982 году для разработки, лицензирования и коммерческого использования патента RSA. У компании есть ряд коммерческих продуктов, включая отдельный пакет безопасности электронной почты, и различные криптографические библиотеки (доступные в виде исходных текстов или объектного кода). RSADSI также предлагает на рынке симметричные алгоритмы RC2 и RC4 (см. раздел 11.8). RSA Laboratories, исследовательская лаборатория, связанная с RSADSI, выполняет фундаментальные криптографические исследования и оказывает консультационные услуги.

При заинтересованности в лицензиях или продуктах нужно обращаться к директору по продажам (Director of Sales, RSA Data Security, Inc., 100 Marine Parkway, Redwood City, CA 94065; (415) 595-8782; факс: (415) 595-1873).

25.5 PUBLIC KEY PARTNERS

Пять патентов, перечисленных в 22-й, принадлежат Public Key Partners (PKP) из Саннивэйла (Sunnyvale), Калифорния, партнерству RSADSI и Care-Kahn, Inc. - родительской компании Cylink. (RSADSI получает 65 процентов прибыли, а Care-Kahn 35 процентов.) PKP утверждает, что эти патенты и 4218582 особенно применимы ко *всем способам использования* криптографии с открытыми ключами.

Табл. 25-3.
Патенты Public Key Partners

№ патента	Дата	Изобретатели	Название патента
4200770	29.3.80	Hellman, Diffie, Merkle	Обмен ключами Diffie-Hellman
4218582	19.8.80	Hellman, Merkle	Рюкзаки Merkle-Hellman
4405829	20.9.83	Rivest, Shamir, Adleman	RSA
4424414	3.3.84	Hellman, Pohlig	Pohlig-Hellman
4995082	19.2.91	Schnorr	Подписи Schnorr

В [574], PKP писала:

> Эти патенты [4200770, 4218582, 4405829 и 4424414] охватывают все известные методы использования искусства открытых ключей, включая варианты, обобщенно известные как ElGamal.
>
> Благодаря широкому распространению цифровых подписей RSA в международном сообществе Public Key Partners решительно одобряет их включение в стандарт цифровой подписи. Мы заверяем все заинтересованные стороны, что Public Key Partners подчинится всем решениям ANSI и IEEE, касающимся доступности лицензирования этого искусства. Особенно для поддержки любых принимаемых стандартов, использующих цифровую подпись RSA. Public Key Partners настоящим заверяет, что лицензии на использование подписей RSA будут предоставляться в разумные сроки, на разумных условиях и без какой-либо дискриминации.

Правда ли это, зависит от того, с кем вы говорите. Лицензии PKP, как правило, секретны, поэтому способа проверить, отличается ли данная лицензия от других, не существует. Хотя компания утверждает, что никому не отказала в выдаче лицензии, по крайней мере две компании говорят о том, что им лицензия выдана не была. PKP тщательно охраняет свои патенты, угрожая всем, кто использует без лицензирования криптографию с открытыми ключами. Частично это реакция на патентное законодательство США. Если владельцу патента не удается наказать нарушителя патента, он может потерять свой патент. Было много разговоров о законности этих патентов, но дальше разговоров дело не пошло. Все законные претензии к патентам PKP были урегулированы до суда.

Я не собираюсь в этой книге давать юридические советы. Может быть патент RSA не устоит перед судом. Может быть эти патенты не применимы ко всей криптографии с открытыми ключами. (Честно говоря, я не понимаю, как они охватывают ElGamal или криптосистемы с эллиптическими кривыми.) Может кому-то удастся выиграть процесс против PKP или RSADSI. Но не забывайте, что корпорации с огромными юридическими отделами, например, IBM, Microsoft, Lotus, Apple, Novell, Digital, National Semiconductor, AT&T и Sun, лицензировали RSA для использования в своих продуктах, а не обращались в суд. Boeing, Shell Oil, DuPont, Raytheon и Citicorp - все лицензировали RSA для своего внутреннего использования.

В одном случае PKP возбудило процесс против TRW Corporation по поводу использования без лицензирования алгоритма ElGamal. TRW утверждала, что ей не нужна лицензия. PKP и TRW достигли соглашения в июне 1992. Подробности урегулирования конфликта неизвестны, но среди них - согласие TRW получить лицензию на патенты. Это не предвещает ничего хорошего. TRW могла позволить себе хороших юристов. Я могу только предположить, что, если бы TRW была уверена, что сможет выиграть процесс, не потратив невероятного количества денег, она бы не отказалась от борьбы.

Тем не менее в PKP существуют свои внутренние проблемы. В июне 1994 года Care-Kahn подала в суд на RSADSI, заявив, среди всего остального, что патент RSA неправилен и неприменим [401]. Оба партнера попытались разорвать свое партнерство. Законны патенты или нет? Нужно ли будет пользователям получать лицензию от Care-Kahn, чтобы пользоваться алгоритмом RSA? Кому будет принадлежать патент Schnorr? Возможно это дело будет урегулировано к моменту выхода этой книги.

Патенты действительны лишь в течение Patents 17 лет и не могут быть возобновлены. 29 марта 1997 года обмен ключами Diffie-Hellman (и алгоритм ElGamal) станут общедоступными. 20 сентября 2000 года станет общедоступным и RSA. Пометьте на своих календарях.

25.6 Международная ассоциация криптологических исследований

Международная ассоциация криптологических исследований (International Association for Cryptologic Research, IACR) - это всемирная криптографическая исследовательская организация. Ее целью является развитие теории и практики криптологии и связанных областей. Ее членом может стать любой. Ассоциация выступает

спонсором двух ежегодных конференций, Crypto (проводится в августе в Санта-Барбаре) и Eurocrypt (проводится в в Европе), и ежеквартально издает *The Journal of Cryptology* и *IACR Newsletter*.

Адрес штаб-квартиры IACR меняется вместе со сменой президента. Текущий адрес: IACR Business Office, Aarhus Science Park, Custav Wieds Vej 10, DK-8000 Aarhus C, Denmark.

25.7 Оценка примитивов целостности RACE (RIPE)

Программа исследования и развития передовых средств связи в Европе (Research and Development in Advanced Communication Technologies in Europe, RACE) была инициирована Европейским сообществом для поддержки предварительной проработки телекоммуникационных стандартов и технологий, поддерживающих Интегрированные высокоскоростные средства связи (Integrated Broadband Communication, IBC). В качестве части этой работы RACE учредило консорциум для Оценки примитивов целостности RACE (RACE Integrity Primitives Evaluation, RIPE), чтобы собрать в одно целое пакет технологий, соответствующих возможным требованиям к безопасности IBC.

Консорциум RIPE образовали шесть ведущих европейских криптографических исследовательских групп: Центр по математике и компьютерным наукам (Center for Mathematics and Computer Science), Амстердам; Siemens AG; Philips Crypto BV; Royal PTT Nederland NV, PTT Research; Katholieke Univesiteit Leuven и Aarhus Universitet. После объявлений о приеме алгоритмов в 1989 и 1991 годах [1564], подачи 32 заявок, присланных со всего мира, и собственно оценивающего проекта длительностью 350 человеко-месяцев, консорциум опубликовал *RIPE Integrity Primitives* [1305, 1332]. Отчет содержит введение, несколько основных концепций целостности и их примитивы: MDC-4 (см. раздел 14.11), RIPE-MD (см. раздел 14.8), RIPE-MAG (см. раздел 14.14), IBC-HASH, SKID (см. раздел 3.2), RSA, COMSET (см. раздел 16.1) и генерацию ключей RSA.

25.8 Условный доступ для Европы (CAFE)

Условный доступ для Европы (Conditional Access for Europe, CAFE) - это проект в рамках программы ESPRIT Европейского сообщества [204, 205]. Работа началась в декабре 1992 года и по плану должна закончиться к концу 1995 года. Образованный консорциум состоит из груп социальных исследований и исследований рынка (Cardware, Institut fur Sozialforschung), изготовителей программного обеспечения и аппаратуры (DigiCash, Cemplus, Ingenico, Siemens), а также криптографов (CWI Amsterdam, PTT Research Netherlands, SPET, Sintef Delab Trondheim, Universities of Arhus, Hildesheim and Leuven).

Целью проекта является разработка системы условного доступа, особенно для цифровых платежных систем. Платежные системы должны обеспечивать надежность для каждого пользователя и требовать как можно меньше веры в себя - надежность не должна зависеть от устойчивости устройств к взлому.

Основным устройством CAFE служит электронный бумажник: маленький компьютер, очень похожий на карманный калькулятор. У него есть батарейка, клавиатура, экран и инфракрасный канал для связи с другими бумажниками. У каждого пользователя свой собственный бумажник, который обеспечивает его права и гарантирует его безопасность.

У устройства с клавиатурой и экраном есть определенное преимущество перед интеллектуальной картой - оно может работать независимо от терминала. Пользователь может непосредственно ввести свой пароль и сумму платежа. Отличие от кредитной карты пользователю не нужно отдавать свой бумажник кому-то, чтобы выполнить транзакцию. Дополнительными возможностями являются:

— Автономные транзакции. Система предназначена для замены обращения небольших сумм наличных, диалоговая система была бы слишком громоздка.

— Устойчивость к потерям. Если пользователь потеряет свой бумажник, или бумажник сломается, или его украдут, пользователь не потеряет свои деньги.

— Поддержка различных валют.

— Открытая архитектура и открытая система. Пользователь должен иметь возможность заплатить за произвольные услуги, например, покупки в магазине, телефон, общественный транспорт, предоставляемые различными поставщиками. Система должна обеспечивать взаимодействие любого количества эмитентов электронных денег, а также взаимодействие бумажников различных типов и производителей.

— Низкая стоимость.

К моменту написания этой книги существует только программная версия системы, и консорциум плотно работает над аппаратным прототипом.

25.9 ISO/IEC 9979

В середине 80-х ISO стандартизировать DES, который уже использовался в качестве FIPS и стандарта ANSI. После некоторой политической возни ISO решило не стандартизировать криптографические алгоритмы, а регистрировать их. Зарегистрировать можно только алгоритмы шифрования, регистрировать хэш-функции и схемы подписи нельзя. Зарегистрировать алгоритм может любая национальная организация.

В настоящее время поданы заявки на регистрацию трех алгоритмов (см. 21-й). Подача заявки включает информацию об использовании, параметрах, реализациях, режимах и тестовых векторах. Подробное описание необязательно, можно подавать на регистрацию и секретные алгоритмы.

Факт регистрации алгоритма ничего не говорит о его качестве. Регистрация не является и одобрением алгоритма ISO/IEC, она просто показывает, что одна из национальных организаций хочет зарегистрировать алгоритм, независимо от критериев, используемых данной организацией.

Меня не впечатлила эта идея. Регистрация мешает процессу стандартизации. Вместо того, чтобы принять несколько алгоритмов, ISO регистрирует любой алгоритм. При таком контроле можно зарегистрировать все, что угодно, и далее с полным правом сопровождать свой алгоритм звучной добавкой "Зарегистрирован ISO/IEC 9979 ". В любом случае реестр ведет National Computer Centre Ltd., Oxford Road, Manchester, MI 7ED, United Kingdom.

Табл. 25-4.
Зарегистрированные алгоритмы ISO/IEC 9979

Регистрационный номер	Название
0001	B-CRYPT
0002	IDEA
0003	LUC

25.10 Профессиональные и промышленные группы, а также группы защитников гражданских свобод

Информационный центр по электронной тайне личности (EPIC)

Информационный центр по электронной тайне личности (Electronic Privacy Information Center, EPIC) был учрежден в 1994 году для привлечения общественного внимания к возникающим вопросам тайн личности, связанным с Национальной информационной инфраструктурой, таких как микросхемы Clipper, предложения по цифровой телефонии, национальные системы идентификационных номеров, тайны историй болезни и продажа сведений о потребителях. EPIC ведет судебные процессы, спонсирует конференции, публикует отчеты, издает *EPIC Alert* и проводит кампании по вопросам тайны личности. Желающие присоединиться могут обратиться по адресу Anyone interested in joining should contact Electronic Privacy Information Center, 666 Pennsylvania Avenue SE, Suite 301, Washington, D.C. 20003 (202,) 544-9240; факс: (202) 547-5482; Internet: info@epic.org.

Фонд электронного фронтира (EFF)

Фонд электронного фронтира (Electronic Frontier Foundation, EFF) посвятил себя защите гражданских прав в киберпространстве. Рассматривая криптографическую политику США, EFF считает, что информация и доступ к криптографии являются фундаментальными правами, и поэтому с них должны быть сняты правительственные ограничения. Фонд организовал рабочую группу по цифровой безопасности и тайне личности (Digital Privacy and Security Working Croup), которая является коалицией 50 организаций. Группа противодействует закону о цифровой телефонии и инициативе Clipper. EFF также содействует ведению процессов против контроля за экспортом криптографии [143]. Желающие присоединиться к EFF могут связаться с Electronic Frontier Foundation, 1001 C Street NW, Suite 950E, Washington, D.C. 20001; (202) 347 5400, факс: (202) 393-5509; Internet: eff@eff.org.

Ассоциация по вычислительной технике (ACM)

Ассоциация по вычислительной технике (Association for Computing Machinery, ACM) - это международная компьютерная промышленная организация. В 1994 году Комитет общественной политики ACM США представил прекрасный отчет о криптографической политике США [935]. Его стоит прочитать каждому, кто интересуется политикой в криптографии. Его можно получить с помощью анонимного ftp с info.acm.org в /reports/acm. crypt_study/acm_crypto_study.ps.

Институт инженеров по электричеству и радиоэлектронике (IEEE)

Институт инженеров по электричеству и радиоэлектронике (Institute of Electrical and Electronics Engineers , IEEE) - это другая профессиональная организация . Отделение в США изучает вопросы, связанные с тайной личности, включая криптографическую политику, идентификационные номера, и защита тайн в Internet, и разрабатывает соответствующие рекомендации .

Ассоциация производителей программного обеспечения (SPA)

Ассоциация производителей программного обеспечения (Software Publishers Association , SPA) - это торговая ассоциация, в которую входят свыше 1000 компаний, разрабатывающих программное обеспечение для перс о-нальных компаний. Они выступают за ослабление экспортного контроля в криптографии и поддерживают пер е-чень коммерчески доступных зарубежных продуктов .

25.11 Sci.crypt

Sci.crypt - это телеконференция Usenet по криптологии. Ее читают примерно 100000 человек по всему миру . Большинство сообщений - обычная чепуха, перебранка ли и то, и другое одновременно. Некоторые сообщения касаются политики, а большинство остальных - просьбы предоставить сведения или общие . Иногда в этой телеконференции случайно попадаются различные самородки и некоторая полезная информация . Если читать sci.crypt регулярно, можно узнать, как использовать нечто, называемое файлом-убийцей .

Другой телеконференцией Usenet является sci.crypt.research, более умеренная телеконференция, посвященная обсуждению криптологических исследований . В ней меньше сообщений, и они гораздо интереснее .

25.12 Шифропанки

Шифропанки (Cypherpunks) - это неформальная группа людей, заинтересованных в обучении и изучении криптографии. Они также экспериментируют с криптографией, пытаясь ввести ее в обиход . По их мнению все криптографические исследования не принесли обществу ничего хорошего, так как оно не воспользовалось до с-тижениями криптографии.

В "Манифесте шифропанков" Эрик Хьюз (Eric Hughes) пишет [744]:

> Мы, Шифропанки, стремимся создать анонимные системы . Мы защищаем наши тайны с помощью криптографии, с п о-мощью систем анонимной отправки почты, с помощью цифровых подписей и электронных денег .
>
> Шифропанки пишут код. Мы знаем, что кто-то должен написать программное обеспечение, защищающее тайны личн о-сти, и так как пока это не сделано, мы не сможем обеспечить сохранение своих тайн, мы собираемся написать такие програ м-мы. Мы публикуем наш код, чтобы наши друзья Шифропанки могли попрактиковаться и поиграть с ним . Наш код свободно может использовать кто угодно и где угодно . Нас не очень волнует, нравятся ли вам программы, которые мы пишем . Мы зна-ем, что программное обеспечение невозможно разрушить, и что невозможно прекратить работу рассеянных систем .

Те, кто хочет присоединиться к списку рассылки шифропанков в Internet, должны отправлять почту в адрес majordomo@toad.com. Список рассылки хранится на ftp.csua.berkeley.edu в /pub/cypherpunks.

25.13 Патенты

Вопрос о программных патентах невозможно втиснуть в рамки этой книги . Хороши они или нет, они суще-ствуют. В Соединенных Штатах можно патентовать алгоритмы , в том числе и криптографические . IBM владеет патентами DES [514]. IDEA запатентован. Запатентованы почти все алгоритмы с открытыми ключами . NIST даже запатентовал DSA. Действие ряда криптографических патентов было блокировано вмешательством NSA, в соответствии с Актом о секретности изобретений (Invention Secrecy Act) от 1940 года и Актом о национальной безопасности (National Security Act) от 1947 года. Это означает, что вместо патента изобретатель получает се к-ретное постановление, и ему запрещается обсуждать его изобретение с кем-нибудь еще .

У NSA есть особые возможности при патентовании . Агентство может обратиться за патентом и затем бл о-кировать его выдачу . Снова появляется секретное постановление, но теперь NSA одновременно и изобретатель, и издатель постановления . Когда спустя некоторое время секретное постановление отменяется , регистрационная контора выдает патент, действующий стандартные 17 лет years. Это более явно защищает изобретение, чем хранение его в секрете . Если кому-нибудь удастся изобрести то же самое , NSA уже подало заявку на патент . Если никому другому не удастся изобрести то же самое, изобретение остается се кретным.

Несмотря на то, что процесс патентования должен не только защищать изобретения, но и раскрывать их, благодаря этой уловке NSA может держать патент дольше 17 лет. Отсчет 17-летнего срока начинается с моме н-та выдачи патента, а не подачи заявки . Пока неясно, как все может измениться в связи с ратификацией договора о GATT Соединенными Штатами .

25.14 Экспортное законодательство США

Согласно правительству США криптография относится к военному снаряжению. Это означает, что криптография подчиняется тем же законам, что и ракета TOW или танк M1 Абрамс. Если вы продаете криптографический продукт без соответствующей экспортной лицензии, то вы - международный контрабандист оружием. Если вы не хотите испортить ваше резюме строкой о пребывании в федеральной тюрьме, обратите внимание на законодательство.

С началом в 1949 году холодной войны все страны НАТО (кроме Исландии), а затем Австралия, Япония и Испания, образовали КОКОМ - Координационный комитет для многостороннего контроля за экспортом (CoCom, Coordinating Committee for Multilateral Export Controls). Это неофициальная организация, призванная координировать национальные ограничения, касающиеся экспорта важных военных технологий в Советский Союз, другие страны Варшавского Договора и Китайскую Народную Республику. Примерами контролируемых технологий являются компьютеры, станки для металлопроката и криптография. Целью этой организации являлось замедление передачи технологий в указанные страны, и сдерживание, таким образом, их военного потенциала.

С концом холодной войны страны КОКОМ осознали, что выполняемый ими контроль большей частью устарел. В настоящее время, по видимому, идет процесс формирования "Нового форума", другой международной организации, которая собирается остановить поток военных технологий в страны, которые не нравятся членам организации.

В любом случае экспортная политика США в отношении стратегических товаров регулируется Правительственным актом об экспорте (Export Administration Act), Актом о контроле над экспортом вооружения (Arms Export Control Act), Актом об атомной энергии (Atomic Energy Act) и Актом о нераспространении ядерных вооружений (Nuclear Non-Proliferation Act). Контроль, установленный этим законодательством, реализуется с помощью многих подзаконных актов, ни один из них не координирует другой. Свыше дюжины организаций, включая военные службы, осуществляют контроль, часто их деятельность перекрывается и конфликтует.

Подконтрольные технологии фигурируют в нескольких списках. Криптография, по традиции относящаяся к вооружению, появляется в Перечне вооружений США (U.S. Munitions List, USML), Международном перечне вооружений (International Munitions List, IML), Перечне контроля за торговлей (Commerce Control List, CCL) и Международном промышленном перечне (International Industrial List, IIL). Госдепартамент отвечает за USML, он публикуется как часть Регулирования международного трафика оружия (International Traffic in Arms Regulations, ITAR) [466, 467].

Экспорт криптографии в США контролируется двумя правительственными организациями. Одной является Комитет по управлению экспортом (Bureau of Export Administration, BXA) в Министерстве торговли, уполномоченный Правилами регулирования экспорта (Export Administration Regulations, EAR). Другая - это Управление по регулированию продажи средств обороны (Office of Defense Trade Controls, DTC) в Государственном департаменте, уполномоченное ITAR. По опыту требования BXA из Министерства торговли менее строги, но сначала весь криптографический экспорт просматривается DTC из Госдепартамента (которое получает советы по технике и национальной безопасности от NSA и, кажется, всегда следует этим советам), которое может отказать передать право решения BXA.

ITAR регулирует этот процесс. (До 1990 года Управление DTC называлось Управлением по контролю над вооружением, возможно, эти усилия в области "паблик рилейшнз" направлены на то, чтобы мы забыли, что мы имеем дело с бомбами и пушками.) Исторически DTC сопротивлялось выдаче экспортных лицензий на средства шифрования сильнее определенного уровня - хотя о том, каков этот уровень, никогда не сообщалось.

Следующие разделы взяты из ITAR [466, 467]:

> § 120.10 Технические данные.
> Технические данные - это, в настоящем подпункте:
> (1) Информация, отличная от программного обеспечения, определенного в 120.10(d), которая нужна для проектирования, разработки, производства, обработки, изготовления, сборки, работы, ремонта, поддержки или модификации средств обороны. Это, например, информация в форме светокопий, чертежей, фотографий, планов, инструкций и документации;
> (2) Секретная информация, касающаяся средств обороны и оборонной деятельности;
> (3) Информация, охватываемая постановлением о засекречивании изобретения;
> (4) Программное обеспечение, определенное в разделе 121.8(f) и непосредственно связанное со средствами обороны
> (5) Это определение не включает информацию, касающуюся общенаучных, математических или инженерных принципов, обычно изучаемых в общедоступных школах, колледжах и университетах, как определено в § 120.11. Оно также не включает базовую рыночную информацию о функции, назначении или общесистемном описании средств обороны.
>
> § 120.11 Открытый доступ.
> Открытый доступ обозначает информацию, которая опубликовывается и может быть общедоступной:
> (1) С помощью продажи в киосках и книжных магазинах;
> (2) С помощью подписки, которая доступна без ограничений для любого, кто хочет получить или приобрести опубликованную информацию;
> (3) С помощью почтовых привилегий второго класса, выданных правительством США;
> (4) В библиотеках, открытых для публики, или в которых публика может получить документы;
> (5) С помощью патентов, доступных в любой патентной конторе;

(6) С помощью неограниченного распространения на конференции, встрече, семинаре, презентации или выставке, доступных обычной публике в Соединенных Штатах;

(7) С помощью сообщений для печати (т.е., неограниченное распространение) в любой форме (например, необязательно опубликованной), одобренных компетентными органами США (см. также § 125.4(b)(13)).

(8) С помощью фундаментальных исследований в науке и технике в аккредитованных высших учебных заведениях США, где полученная информация обычно публикуется и широко распространяется в научном сообществе. Фундаментальными называются базовые и прикладные исследования в науке и технике, когда полученная информация обычно публикуется и широко распространяется в научном сообществе в отличие от исследований, результаты которых не разглашаются из-за прав собственности или определенного контроля доступа и распространения правительством США. Университетские исследования не считаются фундаментальными, если:

(i) Университет или его исследователи соглашаются с другими ограничениями на публикацию научно-технической информации, полученной в результате работы над проектом, или

(ii) Исследования финансируются правительством США, а доступ к результатам исследований и их распространение находится ограничены с целью защиты информации.

§ 120.17 Экспорт.

Под экспортом понимается:

(1) Передача или вывоз средств обороны за пределы Соединенных Штатов любым способом, кроме путешествия за пределы Соединенных Штатов лица, чьи личные знания включают технические данные; или

(2) Передача иностранному лицу прав регистрации, управления или собственности на любой самолет, судно или спутник, присутствующий в Перечне вооружений США, в Соединенных Штатах или за их пределами; или

(3) Раскрытие (в том числе устное или визуальное) или передача в Соединенных Штатах любых средств обороны посольству, учреждению или подразделению иностранного правительства (например, дипломатическим миссиям); или

(4) Раскрытие (в том числе устное или визуальное) или передача технических данных иностранному лицу в Соединенных Штатах или за их пределами; или

(5) Выполнение оборонной деятельности от имени или для выгоды иностранного лица в Соединенных Штатах или за их пределами.

(6) Запускаемый аппарат или полезная нагрузка не должны, при запуске такого аппарата, рассматриваться как экспорт. Однако для определенных целей (см § 126.1 этого подпункта), положения этого подпункта применимы к продажам и другим способам передачи средств обороны или продуктов оборонительной деятельности.

Часть 121- Перечень вооружений США

§ 121.1 Общие положения. Перечень вооружений США

Category XIII—Дополнительное военное снаряжение

(1) Криптографические (включая управление ключами) системы, аппаратура, конструкции, модули, интегральные схемы, компоненты или программное обеспечение с возможностью поддержки секретности или конфиденциальности информации или информационных систем, кроме следующего криптографического оборудования и программного обеспечения:

(i)Специально спроектированное для выполнения защищенным от копирования программным обеспечением только функций дешифрирования при условии, что управление дешифрированием недоступно пользователю.

(ii) Специально спроектированное, разработанное или модифицированное для использования в машинах для банковских операций или денежных транзакций, которое можно использовать только для таких транзакций. Машины для банковских операций или денежных транзакций включают автоматические кассовые аппараты, самообслуживаемые печатающие устройства, торговые терминалы или оборудование для шифрования межбанковских транзакций.

(iii) Использующее только аналоговые методы для криптографической обработки, которая обеспечивает безопасность информации в следующих приложениях. . . .

(iv) Персональные интеллектуальные карточки, использование которых возможно только в оборудовании или системах, не попадающих под регулирование USML.

(v) С ограничением доступа, такие как автоматические кассовые аппараты, самообслуживаемые печатающие устройства или торговые терминалы, которые обеспечивают защиту паролей или персональных идентификационных номеров (PIN) или аналогичных данных, чтобы предотвратить несанкционированный доступ к средствам, но не могут шифровать файлы или тексты, не посредственно не связанные с защитой паролей или PIN.

(vi) Осуществляющее только проверку подлинности данных с помощью вычисления кода проверки подлинности сообщения (MAC) или аналогичной функции для проверки, что в текст не было внесено изменений, или для проверки подлинности пользователей, но которое нельзя использовать для шифрования данных, текста или другой информации помимо необходимой для проверки подлинности.

(vii) Использующее только фиксированные методы сжатия и кодирования данных.

(viii) Используемое только для радиовещания, платного телевидения или аналогичных телевизионных систем с ограниченной аудиторией, без цифрового шифрования, и в которых цифровое дешифрирование ограничено только видео- и аудиофункциями или управлением.

(ix) Программное обеспечение, спроектированное или модифицированное для защиты от злоумышленных компьютерных повреждений, (например, вирусов).

(2) Криптографические (включая управление ключами) системы, аппаратура, конструкции, модули, интегральные схемы, компоненты или программное обеспечение с возможностью генерации распространяемых кодов для большого количества систем или устройств:

(3) Криптографические системы, аппаратура, конструкции, модули, интегральные схемы, компоненты или программное обеспечение.

§ 125.2 Экспорт несекретных технических данных.

(a) Общие положения. Для экспорта несекретных технических данных необходима лицензия (DSP-5), если эти данные не исключены из лицензирующих требований данного подпункта. В случае планового визита детали предполагаемых дискуссий должны быть переданы в Управление по регулированию продажи средств обороны для экспертизы технических данных. Должно быть предоставлено семь копий технических данных или тем ди скуссий.

(b) Патенты. При экспорте технических данных требуется лицензия, выданная Управлением по регулированию продажи средств обороны, если данные превышают необходимые для заполнения внутренней патентной заявки или для заполнения иностранной патентной заявки, если внутренняя заявка не была заполнена. Заявки на патентование за рубежом, выполнение в таких патентах улучшений, модификаций или дополнений должны регулироваться Управлением по патентам и торговым знакам США в соответствии с 37 CFR, часть 5. Экспорт технических данных, необходимых для патентования в других странах, является субъектом норм, издаваемых Управлением по патентам и торговым знакам США, в соответствии с 35 U.S.C. 184.

(c) Раскрытия. Для устного, визуального или документального раскрытия технических данных гражданами США иностранным лицам требуется лицензия, если в данном подпункте не оговорено иное. Лицензия требуется независимо от формы передачи технических данных (например, лично, по телефону, в переписке, электронными средствами, и т.д.). Лицензия тре-

буется для таких раскрытий, делаемых гражданами США при посещении иностранных дипломатических миссий и консульств.

И так далее. В этом документе намного больше информации. Если вы собираетесь экспортировать криптографию, я советую вам добыть его копию и воспользоваться услугами юриста, который во всем этом разбирается.

В действительности экспорт криптографических продуктов контролируется NSA. Если вам нужно получить свидетельство о признании вашего продукта предметом общего потребления (Commodity Jurisdiction, CJ), вы должны представить ваш продукт на одобрение в NSA и подать в Государственный департамент заявку на получение CJ. После одобрения в Госдепартаменте дело попадает под юрисдикцию Министерства торговли, которое никогда особенно не интересовалось криптографией. Однако Государственный департамент никогда не выдаст CJ без одобрения NSA.

В 1977 году Джозеф А. Мейер (Joseph A. Meyer), служащий NSA, написал письмо - несанкционированное, в соответствии с официальной историей инцидента - в IEEE, предупреждающее, что планируемое представление оригинальной работы RSA нарушит ITAR. Из *The Puzzle Palace*:

> Вот его точка зрения. ITAR охватывает всю "несекретную информацию, которая может быть использована, или адаптирована для использования, при проектировании, производстве, изготовлении, ремонте, капитальном ремонте, переработке, конструировании, разработке, действии, поддержке или восстановлении" перечисленных материалов, также как и "любую технологию, которая развивает определенное умение или создает новое в области, которая имеет важное военное применение в Соединенных Штатах." И экспорт действительно включал передачу информации как в письменном виде, так и с помощью устных или визуальных средств, включая краткие обсуждения и симпозиумы, на которых были представлены иностранцы.
>
> Но, буквально следуя туманному, часто слишком пространному законодательству, кажется, требуется, чтобы каждый, кто собирается написать или заявить что-то на тему, касающуюся Перечня вооружений, сначала получил бы одобрение Государственного департамента - эта унылая перспектива явно противоречит Первой поправке и требует подтверждения Верховным судом.

В конце концов NSA признало действия Мейера несанкционированными, и работа по RSA была опубликована, как планировалось. Против изобретателей не было предпринято никаких действий, хотя может быть доказано, что их работа увеличила возможности зарубежной криптографии гораздо больше, чем что-нибудь, опубликованное до того.

Экспорт криптографии обсуждается в следующем заявлении NSA [363]:

> Криптографические технологии считаются жизненно важными для интересов национальной безопасности, включая экономические интересы, военные интересы и интересы внешней политики.
>
> Мы не согласны с заявлениями, сделанными 7 мая 1992 года на слушаниях Судебного комитета, и последними газетными статьями, которые заявляют, что экспортные законы США мешают американским фирмам изготавливать и использовать современное шифровальное оборудование. Нам неизвестно ни об одном случае, когда из-за экспортных ограничений США американской фирме помешали изготавливать и использовать аппаратуру шифрования внутри страны, или американской фирме либо ее дочерней компании помешали использовать аппаратуру шифрования за пределами США. В действительности, NSA всегда поддерживало использование шифрования в американском бизнесе для защиты важной информации как дома, так и за границей.
>
> Для экспорта в другие страны NSA, являющееся частью Министерства обороны, (вместе с Государственным департаментом и Министерством торговли просматривает экспортные лицензии в поисках технологий информационной безопасности, попадающих под действие Экспортного правительственного законодательства или Регулирования международного трафика оружия. Аналогичная система контроля экспорта действует во всех странах КОКОМ и во многих других странах, так как эти технологии повсеместно считаются важными. Не существует общего запрета на экспорт подобных технологий, каждый случай рассматривается отдельно. При этом может потребоваться получить лицензии на такие системы, при получении которых анализируется влияние экспорта этой системы на интересы национальной безопасности - включая интересы экономической, военной и политической безопасности. Экспортные лицензии выдаются или не выдаются в зависимости от типа задействованного оборудования, предполагаемого использования и предполагаемого пользователя.
>
> Наш анализ показывает, что США лидирует в мировом производстве и экспорте технологий информационной безопасности. NSA одобряет для экспорта свыше 90% криптологических продуктов, направленных в NSA Государственным департаментом для лицензирования. Экспортные лицензии на продукты информационной безопасности, попадающие под юрисдикцию Министерства торговли, выдаются без участия NSA или Министерства обороны. Среди них - продукты, использующие такие методы, как DSS и RSA, обеспечивающие проверку подлинности и контроль доступа к компьютерам и сетям. На самом деле, в прошлом NSA играло главную роль в успешном отстаивании ослабления экспортного контроля над RSA и близкими технологиями для проверки подлинности. Эти методы особенно важны при решении проблемы хакеров и несанкционированного использования ресурсов.

Итак, заявлено, что NSA ограничивает экспорт только продуктов шифрования, но не проверки подлинности. Если вы собираетесь экспортировать продукт только для проверки подлинности, получение разрешение ограничится демонстрацией того, что ваш продукт нельзя без значительных переделок использовать для шифрования. Более того, бюрократическая процедура для продуктов проверки подлинности намного проще, чем для продуктов шифрования. Для системы проверки подлинности получать одобрение Госдепартамента (CJ), система шифрования требует повторного одобрения для каждой версии продукта или даже при каждой продаже.

Без CJ вам придется запрашивать разрешение на экспорт всякий раз, когда вы захотите экспортировать продукт. Государственный департамент не разрешает экспортировать продукты с сильным шифрованием, даже использующие DES. Отдельные исключения были сделаны для дочерних фирм американских компаний для возможности закрытой связи с, для некоторых банковских приложений и экспорт для военных пользователей США. Ассоциация производителей программного обеспечения (SPA) вела переговоры с правительством об ослаблении ограничений на экспортные. Соглашение, заключенное SPA и Госдепартаментом в 1992 году, облег-

чило правила выдачи экспортных лицензий для двух алгоритмов, RC2 и RC4, при условии, что длина используемого ключа не превысит 40 битов. Подробности можно найти в разделе 7.1.

В 1993 году в Палате представителей Мария Кантвелл (Maria Cantwell) (D-WA) по просьбе компаний-разработчиков программного обеспечения внесла законопроект, ослабляющий экспортный контроль за программами. Сенатор Пэтти Мюррей (Patty Murray) (D-WA) внесла соответствующий билль в сенате. Законопроект Кантвелл был добавлен к общему закону о контроле над экспортом, проходящему через Конгресс, но был удален Комитетом по разведке под сильным давлением NSA. Когда NSA что-нибудь делает, оно прикладывает все усилия - комитет единодушно проголосовал за удаление формулировки. За последнее время я не припомню другого случая, чтобы группа законодателей что-то сделала единодушно.

В 1995 году Дан Бернштейн (Dan Bernstein) при поддержке EFF подал в суд на правительство США, пытаясь помешать правительству ограничивать публикации криптографических документов и программного обеспечения [143]. В иске утверждалось, что законы об экспортном контроле неконституционны и вносят "непозволительные априорные ограничения высказываний в нарушение Первой поправки". Конкретно в иске утверждалось, что современный процесс контроля над экспортом:

— Позволяет бюрократам ограничивать публикации без решения суда.

— Обеспечивает слишком мало процедурных возможностей защиты прав в соответствии с Первой поправкой.

— Требует от издателей регистрироваться в правительстве, создавая эффект "лицензированной прессы".

— Отказывает в общих публикациях, требуя идентифицировать каждого получателя.

— Достаточно запутан, чтобы простые люди не могли знать, какое поведение правильно, а какое - нет.

— Слишком пространен, так как запрещает поведение, которое явно защищается (например, разговор с иностранцами внутри Соединенных Штатов).

— Применяется слишком широко, запрещая экспорт программного обеспечения не содержащего криптографии, исходя из соображений, что криптография может быть добавлена позже.

— Явно нарушает Первую поправку, запрещая частные беседы по криптографии, так как правительство желает вместо этого навязывать публике свои криптографические взгляды.

— Многими способами превышает полномочия, предоставленные как Конгрессом в экспортном законодательстве, так и Конституцией.

Можно предвидеть, что решение этого дела займет несколько лет, но предвидеть, чем оно закончится, невозможно.

Тем не менее, Консультативный комитет по безопасности и защищенности (Computer Security and Privacy Advisory Board), официальный консультант NIST, в марте 1992 года проголосовал за то, чтобы пересмотреть в национальной политике криптографические вопросы, включая экспортную политику. Было заявлено, что экспортная политика определяется только организациями, отвечающими за национальную безопасность, без учета точки зрения организаций, связанных с развитием торговли. Эти связанные с национальной безопасностью организации делают все возможно, чтобы ничего не изменилось, но необходимость перемен уже назрела.

25.15 Экспорт и импорт криптографии за рубежом

В других странах существует свое экспортное и импортное право [311]. Приведенный обзор неполон и возможно устарел. Страны могут издать законы и не обращать на них внимания, или не иметь законов, но каким-то образом ограничивать экспорт, импорт и использование.

— Австралия требует наличия сертификата у импортируемого криптографического продукта только по требованию страны-экспортера.

— В Канаде нет контроля импорта, а контроль экспорта аналогичен американскому. Экспорт продуктов из Канада может быть ограничен, если они включены в Перечень контроля экспорта, соответствующий Акту разрешений экспорта и импорта. В отношении криптографических технологий Канада следует ограничениям КОКОМ. Шифровальные устройства описаны под категорией пять, части два канадских правил экспорта. These provisions аналогичны категории пять в Правительственных правилах экспорта в США.

— Китай использует схему лицензирования импортируемых продуктов, экспортеры должны заполнить заявку в Министерстве зарубежной торговли. На основе китайского Перечня запрещенного и ограниченного экспорта и импорта, принятого в 1987 году, Китай ограничивает импорт и экспорт устройств кодирования речи.

— Во Франции нет специального законодательства относительно импорта криптографии, но существуют за-

коны, касающиеся продажи и использования криптографии в стране. Продукты должны быть сертифицированы: либо они должны соответствовать опубликованным спецификациям, либо фирменная спецификация компании должна быть предоставлена правительству. Правительство может также затребовать два устройства для собственного использования. У компаний должна быть лицензия на продажу криптографии во Франции, в лицензии указывается рыночное назначение. У пользователей должна быть лицензия на покупку и использование криптографии, в лицензию включено положение о том, что пользователи должны быть готовы передать свои ключи правительству в течение четырех месяцев после использования. Это ограничение иногда допускает исключения: банков, больших компаний, и т.д. Для криптографии, экспортируемой из США, лицензионные требования отсутствуют.

— Германия следует положениям КОКОМ, требуя лицензировать экспорт криптографии. Проводится специальный контроль общедоступного криптографического программного обеспечения.

— В Израиле есть ограничения импорта, но, по видимому, никто не знает какие.

— Бельгия, Италия, Япония, Нидерланды и Великобритания следуют положениям КОКОМ, требуя лицензировать экспорт криптографии.

— В Бразилии, Индии, Мексике, России, Саудовской Аравии, Испании, Южной Африке, Швеции и Швейцарии контроль экспорта или импорта криптографии отсутствует.

25.16 Правовые вопросы

Являются ли цифровые подписи настоящими подписями? Будут ли они признаны судом? Некоторые предварительные правовые исследования привели к мнению, что цифровые подписи будут соответствовать требованиям законных обязующих подписей для большей части применений, включая коммерческое использование, определенное в Едином своде законов о торговле (Uniform Commercial Code, UCC). Решение Управления по общей бухгалтерии (GAD, General Accounting Office), вынесенное по просьбе NIST, утверждает, что цифровые подписи соответствуют правовым стандартам для рукописных подписей [362].

Акт о цифровых подписях штата Юта вступил в действие 1 мая 1995 года, обеспечивая законную основу использования цифровых подписей в системе судопроизводства. Калифорния рассматривает соответствующий законопроект, а в Орегоне и Вашингтоне разрабатывают свои законы. Техас и Флорида дышат им в затылок. К моменту издания книги большинство штатов пройдет этот путь.

Американская юридическая ассоциация (Отдел EDI и информационных технологий секции науки и техники) разработала образец акта, который может быть использован штатами в процессе законотворчества. Акт пытается вписать цифровые подписи в существующую для подписей правовую инфраструктуру: Единый свод законов о торговле, Законы Федеральной резервной системы Соединенных Штатов, общее право о контрактах и подписях, Конвенция ООН по контрактам для международной продажи товаров и Конвенция ООН по международным законам о комитетах по биржам и долговым обязательствам. В акт включены положения об ответственности и обязанностях сертифицирующих органов, вопросы ответственности, а также ограничения и политика.

В Соединенных Штатах законы о подписях, контрактах и торговых операциях находятся в юрисдикции штатов, поэтому этот акт-образец разработан для штатов. Окончательной целью является федеральный акт, но если все начинается на уровне штатов, у NSA меньше возможностей все испоганить.

Даже при этом, пока правильность цифровых подписей не будет оспорена в суде, их правовой статус останется неопределенным. Для того, чтобы цифровые подписи обладали теми же идентификационными возможностями, что и рукописные подписи, они сначала должны быть использованы для подписания законного, затем оспорены в суде одной из сторон. Тогда суд рассмотрит безопасность схемы подписи и вынесет решение. Спустя некоторое время, когда повторится подобный случай, решения о том, какие методы цифровой подписи и какие размеры ключей понадобятся, чтобы цифровая подпись была признана законной, будет вынесено на основе предыдущих решений. Возможно для этого потребуются годы.

До тех пор, если два человека хотят использовать цифровые подписи для контракта(для заявок на покупку, для приказов по работе, и т.д.), рекомендуется, чтобы они подписали на бумаге контракт, с котором они согласятся в будущем признавать любые документы, подписанные их цифровыми подписями [1099]. В этом документе должны определяться алгоритм, размер ключа и все остальные параметры. В нем должен, к тому же, быть определен способ разрешения споров.

Послесловие Мэтта Блейза

Одним из самых опасных моментов криптологии (и, следовательно, данной книги), является то, что вам почти удается измерить ее. Знание длины ключей, способов разложения на множители и криптоаналитических методов позволяет оценить (в отсутствии настоящей теории проектирования шифров) " коэффициент работы", необходимый для вскрытия конкретного шифра. Слишком велик соблазн неправильно использовать эти оценки в качестве общей меры безопасности систем. В реальном мире у взломщика есть куда больше возможностей, чем использование одного криптоанализа. Часто успех достигается с помощью вскрытий протоколов, троянских коней, вирусов, электромагнитного контроля, физической компрометации, шантажа и запугивания владельцев ключа, ошибок операционной системы и прикладных программ, аппаратных ошибок, ошибок пользователей, физического подслушивания, прикладной социологии, анализ содержимого свалок, и это далеко не все.

Высококачественные шифры и протоколы являются важными средствами, но сами по себе они не заменяют реалистичных, критических размышлений о том, что действительно нужно защитить, и как могут быть взломаны различные уровни обороны (взломщики, в конце концов, редко ограничиваются чистыми, хорошо определенными моделями научного мира). Росс Андерсон (Ross Anderson) приводит примеры криптографически сильных систем (в банковской индустрии), которые не устояли перед угрозами реального мира [43, 44]. Даже когда у взломщика есть доступ только к шифротексту, через кажущиеся незначительными бреши в других частях системы может просочиться достаточно информации, чтобы сделать хорошую криптосистему бесполезной . Союзники во второй мировой войне взломали трафик немецкой Энигмы, главным образом тщательно используя ошибки операторов [1587].

NSA в ответ на вопрос, может ли правительство вскрывать DES, язвительно заметило, что реальные системы настолько небезопасны, что об этом даже не стоит беспокоиться. К сожалению, не существует простых рецептов, как сделать систему безопасной, заменить тщательное проектирование и критический анализ невозможно. Хорошие криптосистемы делают жизнь взломщика намного труднее, чем жизнь законного пользователя, но это не так в отношении почти всех остальных аспектов безопасности компьютеров и систем связи . Рассмотрим следующие (наверняка не все) "Десять главных угроз безопасности реальных систем ", каждую из которых легче осуществить, чем предотвратить.

1. Печальное состояние программного обеспечения. Всем известно, что никто не знает, как писать программное обеспечение. Современные системы сложны, включают сотни тысяч строк кода, любая из них может повредить безопасности. Из программных модулей, связанных с безопасностью извлекать ошибки еще труднее.

2. Неэффективная защита против вскрытий с отказом от услуг. В некоторых криптографических протоколах допускается анонимность. Использование анонимных протоколов может быть особенно опасным, если они увеличивают возможность неопознанного вандала нарушить предоставление услуги Поэтому анонимные системы должны быть особенно устойчивы к вскрытиям с отказом от услуг . В устойчивых сетях поддерживать анонимность может быть легче - ведь вряд ли кого-то сильно волнует наличие миллионов анонимных входных точек в большинстве устойчивых сетей, таких как телефонная сеть или почтовая система, где отдельному пользователю относительно трудно (или дорого) вызвать крупномасштабные аварии.

3. Нет места для хранения секретов. Криптосистемы защищают большие секреты малыми (ключами) . К сожалению, современные компьютеры не особенно хороши для защиты даже маленьких секретов . Многопользовательские сетевые рабочие станции могут быть взломаны, а их память - скомпромет ирована. Отдельно стоящие, однопользовательские машины могут быть украдены или скомпрометированы вирусами, которые организуют асинхронную утечку секретов . Удаленные серверы, где может и не быть пользователя, вводящего парольную фразу (но см. угрозу №5), представляют собой особенно трудную проблему.

4. Плохая генерация случайных чисел. Для ключей и сеансовых переменных нужны хорошие источники непредсказуемых битов. Энтропия работающего компьютера велика, но редкое приложение в состоянии правильно использовать ее. Было предложено множество методов получать истинно случайные числа программным образом (используются непредсказуемость времени выполнения операций ввода вывода, расхождения тактовой частоты и таймера, и даже турбулентность воздуха внутри корпуса твердого диска), но все они очень чувствительны к незначительным изменениям сред, в которых они используются.

5. Слабые парольные фразы. Большинство криптографического программного обеспечения решает проблемы хранения и генерации ключей на основе создаваемых пользователем парольных фраз , которые считаются достаточно непредсказуемыми для генерации хорошего ключевого материала, и которые также легко запоминаются и поэтому не требуют безопасного хранения . В то время, как словарные вскрытия являются хорошо известной проблемой для коротких паролей , о способах вскрытия ключей,

созданных на основе выбранных пользователями парольных фраз, известно мало. Шеннон показал, что энтропия английского текста чуть больше 1 бита на символ, что, по видимому, позволяет использовать против парольных фраз грубую силу. Однако пока не вполне понятно, для этого как упорядочивать парольные фразы. Пока мы не разберемся как следует, как вскрывать парольные фразы, мы не поймем, насколько они слабы или сильны.

6. Неправильное доверие. Почти все доступное криптографическое программное обеспечение предполагает, что пользователь находится в непосредственном контакте с системой ли пользуется надежным способом доступа. Например, интерфейсы к программам, подобным PGP, предполагают, что их парольные фразы поступают от пользователя по надежному пути, например, с локальной консоли. Но это не всегда так, рассмотрим проблему чтения вами шифрованной почты при подключении по сети. То, что проектировщик системы считает надежным, может не соответствовать потребностям или ожиданиям реальных пользователей, особенно когда программным обеспечением можно управлять удаленно по небезопасным каналам.

7. Плохо понимаемое взаимодействие протоколов и услуг. С ростом и усложнением систем часто происходят странные вещи, и бывает трудно что-нибудь понять что-нибудь, даже когда произойдет какая-нибудь авария. Червь Internet распространялся с помощью туманного и с виду вполне невинного средства программы передачи почты. Сколько еще возможностей и в каком количестве программ обладают неожиданными следствиями, которые только ждут своего о ткрытия?

8. Нереалистичная оценка угрозы и риска. Эксперты по безопасности стремятся сконцентрировать свои усилия на угрозах, которые известно как моделировать и предотвращать. К сожалению, взломщики выполняют вскрытия на базе собственных знаний, и две эти области редко совпадают. Слишком много "безопасных" систем было спроектировано без учета реально возможных действий взломщика.

9. Интерфейсы, которые делают безопасность дорогой и неудобной. Если нужно использовать средства обеспечения безопасности, то они должны быть удобными и достаточно прозрачными, чтобы люди действительно пользовались ими. Нетрудно спроектировать механизмы шифрования, которые работают только за счет производительности или простоты использования, и еще легче создать механизм, который провоцирует ошибки. Безопасность должно быть труднее выключить, чем включить; к несчастью, лишь немногие системы действительно так р аботают.

10. Слишком всеобъемлющие требования к безопасности. Эта проблема хорошо известна почти всем, чье счастье связано с продажей продуктов и услуг безопасности. Пока существует широко распространенное требование всеобъемлющей безопасности, средства и инфраструктура, обеспечивающие его реализацию, будут дороги и недоступны для многих приложений. Частично это проблема понимания и раскрытия угроз и опасностей в реальных приложениях, а частично проблема проектирования систем, в которых безопасность не закладывается изначально, а добавляется позже.

Более полный список и обсуждение подобных угроз может легко заполнить книгу такого же размера, при этом проблема будет лишь едва затронута. Что делает их особенно трудными и опасными, так это то, что не существует никакого магического способа избавиться от них, кроме хорошего анализа и хорошей инженерной работы. Честолюбивый криптограф должен ощущать границы искусства.

Мэтт Блейз
Нью-Йорк

Часть V

Исходные коды

1. **DES**
2. **LOKI91**
3. **IDEA**
4. **GOST**
5. **BLOWFISH**
6. **3-WAY**
7. **RC5**
8. **A5**
9. **SEAL**

DES

```
#define EN0   0      /* MODE == encrypt */
#define DE1   1      /* MODE == decrypt */

typedef struct {
        unsigned long ek[32];
        unsigned long dk[32];
} des_ctx;

extern void deskey(unsigned char *, short);
/*                  hexkey[8]     MODE
 * Sets the internal key register according to the hexadecimal
 * key contained in the 8 bytes of hexkey, according to the DES,
 * for encryption or decryption according to MODE.
 */

extern void usekey(unsigned long *);
/*                cookedkey[32]
 * Loads the internal key register with the data in cookedkey.
 */

extern void cpkey(unsigned long *);
/*               cookedkey[32]
 * Copies the contents of the internal key register into the storage
 * located at &cookedkey[0].
 */

extern void des(unsigned char *, unsigned char *);
/*                from[8]         to[8]
 * Encrypts/Decrypts (according to the key currently loaded in the
 * internal key register) one block of eight bytes at address 'from'
 * into the block at address 'to'.  They can be the same.
 */

static void scrunch(unsigned char *, unsigned long *);
static void unscrun(unsigned long *, unsigned char *);
static void desfunc(unsigned long *, unsigned long *);
static void cookey(unsigned long *);
```

```
static unsigned long KnL[32] = { 0L };
static unsigned long KnR[32] = { 0L };
static unsigned long Kn3[32] = { 0L };
static unsigned char Df_Key[24] = {
       0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
       0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10,
       0x89,0xab,0xcd,0xef,0x01,0x23,0x45,0x67 };

static unsigned short bytebit[8]    = {
       0200, 0100, 040, 020, 010, 04, 02, 01 };

static unsigned long bigbyte[24] = {
       0x800000L,    0x400000L,     0x200000L,    0x100000L,
       0x80000L,     0x40000L,      0x20000L,     0x10000L,
       0x8000L,      0x4000L,       0x2000L,      0x1000L,
       0x800L,              0x400L,               0x200L,              0x100L,
       0x80L,               0x40L,                0x20L,               0x10L,
       0x8L,         0x4L,          0x2L,         0x1L   };

/* Use the key schedule specified in the Standard (ANSI X3.92-1981). */

static unsigned char pc1[56] = {
       56, 48, 40, 32, 24, 16,  8,   0, 57, 49, 41, 33, 25, 17,
        9,  1, 58, 50, 42, 34, 26,  18, 10,  2, 59, 51, 43, 35,
       62, 54, 46, 38, 30, 22, 14,   6, 61, 53, 45, 37, 29, 21,
       13,  5, 60, 52, 44, 36, 28,  20, 12,  4, 27, 19, 11,  3 };

static unsigned char totrot[16] = {
       1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28 };

static unsigned char pc2[48] = {
       13, 16, 10, 23,  0,  4,       2, 27, 14,  5, 20,  9,
       22, 18, 11,  3, 25,  7,      15,  6, 26, 19, 12,  1,
       40, 51, 30, 36, 46, 54,      29, 39, 50, 44, 32, 47,
       43, 48, 38, 55, 33, 52,      45, 41, 49, 35, 28, 31 };

void deskey(key, edf)       /* Thanks to James Gillogly & Phil Karn! */
unsigned char *key;
short edf;
{
       register int i, j, l, m, n;
       unsigned char pc1m[56], pcr[56];
       unsigned long kn[32];

       for ( j = 0; j < 56; j++ ) {
              l = pc1[j];
              m = l & 07;
              pc1m[j] = (key[l >> 3] & bytebit[m]) ? 1 : 0;
              }
       for( i = 0; i < 16; i++ ) {
              if( edf == DE1 ) m = (15 - i) << 1;
              else m = i << 1;
              n = m + 1;
              kn[m] = kn[n] = 0L;
              for( j = 0; j < 28; j++ ) {
                     l = j + totrot[i];
                     if( l < 28 ) pcr[j] = pc1m[l];
                     else pcr[j] = pc1m[l - 28];
                     }
              for( j = 28; j < 56; j++ ) {
                  l = j + totrot[i];
```

```
                if( l < 56 ) pcr[j] = pc1m[l];
                else pcr[j] = pc1m[l - 28];
                }
            for( j = 0; j < 24; j++ ) {
                if( pcr[pc2[j]] ) kn[m] |= bigbyte[j];
                if( pcr[pc2[j+24]] ) kn[n] |= bigbyte[j];
                }
            }
        cookey(kn);
        return;
}

static void cookey(raw1)
register unsigned long *raw1;
{
        register unsigned long *cook, *raw0;
        unsigned long dough[32];
        register int i;

        cook = dough;
        for( i = 0; i < 16; i++, raw1++ ) {
            raw0 = raw1++;
            *cook   = (*raw0 & 0x00fc0000L) << 6;
            *cook  |= (*raw0 & 0x00000fc0L) << 10;
            *cook  |= (*raw1 & 0x00fc0000L) >> 10;
            *cook++       |= (*raw1 & 0x00000fc0L) >> 6;
            *cook   = (*raw0 & 0x0003f000L) << 12;
            *cook  |= (*raw0 & 0x0000003fL) << 16;
            *cook  |= (*raw1 & 0x0003f000L) >> 4;
            *cook++       |= (*raw1 & 0x0000003fL);
            }
        usekey(dough);
        return;
}

void cpkey(into)
register unsigned long *into;
{
        register unsigned long *from, *endp;

        from = KnL, endp = &KnL[32];
        while( from < endp ) *into++ = *from++;
        return;
}

void usekey(from)
register unsigned long *from;
{
        register unsigned long *to, *endp;

        to = KnL, endp = &KnL[32];
        while( to < endp ) *to++ = *from++;
        return;
}

void des(inblock, outblock)
unsigned char *inblock, *outblock;
{
        unsigned long work[2];

        scrunch(inblock, work);
        desfunc(work, KnL);
```

```
        unscrun(work, outblock);
        return;
}

static void scrunch(outof, into)
register unsigned char *outof;
register unsigned long *into;
{
        *into   = (*outof++ & 0xffL) << 24;
        *into  |= (*outof++ & 0xffL) << 16;
        *into  |= (*outof++ & 0xffL) << 8;
        *into++ |= (*outof++ & 0xffL);
        *into   = (*outof++ & 0xffL) << 24;
        *into  |= (*outof++ & 0xffL) << 16;
        *into  |= (*outof++ & 0xffL) << 8;
        *into  |= (*outof   & 0xffL);
        return;
}

static void unscrun(outof, into)
register unsigned long *outof;
register unsigned char *into;
{
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >>  8) & 0xffL;
        *into++ =  *outof++           & 0xffL;
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >>  8) & 0xffL;
        *into   =  *outof     & 0xffL;
        return;
}

static unsigned long SP1[64] = {
        0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
        0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
        0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
        0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
        0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
        0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
        0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
        0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
        0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
        0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
        0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
        0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
        0x01000004L, 0x00000404L, 0x00010404L, 0x01010400L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00000000L,
        0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L };

static unsigned long SP2[64] = {
        0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
        0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
        0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
        0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
        0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
        0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
        0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
        0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
        0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
```

```
        0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
        0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
        0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
        0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
        0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
        0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
        0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L };

static unsigned long SP3[64] = {
        0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
        0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
        0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
        0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
        0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
        0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
        0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
        0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
        0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
        0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
        0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
        0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
        0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
        0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
        0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
        0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L };

static unsigned long SP4[64] = {
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
        0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
        0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
        0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
        0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
        0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
        0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
        0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
        0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L };

static unsigned long SP5[64] = {
        0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
        0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
        0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
        0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
        0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
        0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
        0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
        0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
        0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
        0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
        0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
        0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
        0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
        0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
        0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
        0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L };
```

```
static unsigned long SP6[64] = {
        0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
        0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
        0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
        0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
        0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
        0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
        0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
        0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
        0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
        0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
        0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
        0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
        0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
        0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L };

static unsigned long SP7[64] = {
        0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
        0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
        0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
        0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
        0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
        0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
        0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
        0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
        0x04000000L, 0x00200800L, 0x00200000L, 0x04000802L,
        0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
        0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
        0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
        0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
        0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
        0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
        0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L };

static unsigned long SP8[64] = {
        0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
        0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
        0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
        0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
        0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
        0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
        0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
        0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
        0x00041040L, 0x00040000L, 0x10041000L, 0x00001000L,
        0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
        0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
        0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
        0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
        0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
        0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
        0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L };

static void desfunc(block, keys)
register unsigned long *block, *keys;
{
        register unsigned long fval, work, right, leftt;
        register int round;

        leftt = block[0];
        right = block[1];
```

```
work = ((leftt >> 4) ^ right) & 0x0f0f0f0fL;
right ^= work;
leftt ^= (work << 4);
work = ((leftt >> 16) ^ right) & 0x0000ffffL;
right ^= work;
leftt ^= (work << 16);
work = ((right >> 2) ^ leftt) & 0x33333333L;
leftt ^= work;
right ^= (work << 2);
work = ((right >> 8) ^ leftt) & 0x00ff00ffL;
leftt ^= work;
right ^= (work << 8);
right = ((right << 1) | ((right >> 31) & 1L)) & 0xffffffffL;
work = (leftt ^ right) & 0xaaaaaaaaL;
leftt ^= work;
right ^= work;
leftt = ((leftt << 1) | ((leftt >> 31) & 1L)) & 0xffffffffL;

for( round = 0; round < 8; round++ ) {
       work  = (right << 28) | (right >> 4);
       work ^= *keys++;
       fval  = SP7[ work             & 0x3fL];
       fval |= SP5[(work >>  8) & 0x3fL];
       fval |= SP3[(work >> 16) & 0x3fL];
       fval |= SP1[(work >> 24) & 0x3fL];
       work  = right ^ *keys++;
       fval |= SP8[ work             & 0x3fL];
       fval |= SP6[(work >>  8) & 0x3fL];
       fval |= SP4[(work >> 16) & 0x3fL];
       fval |= SP2[(work >> 24) & 0x3fL];
       leftt ^= fval;
       work  = (leftt << 28) | (leftt >> 4);
       work ^= *keys++;
       fval  = SP7[ work             & 0x3fL];
       fval |= SP5[(work >>  8) & 0x3fL];
       fval |= SP3[(work >> 16) & 0x3fL];
       fval |= SP1[(work >> 24) & 0x3fL];
       work  = leftt ^ *keys++;
       fval |= SP8[ work             & 0x3fL];
       fval |= SP6[(work >>  8) & 0x3fL];
       fval |= SP4[(work >> 16) & 0x3fL];
       fval |= SP2[(work >> 24) & 0x3fL];
       right ^= fval;
       }

right = (right << 31) | (right >> 1);
work = (leftt ^ right) & 0xaaaaaaaaL;
leftt ^= work;
right ^= work;
leftt = (leftt << 31) | (leftt >> 1);
work = ((leftt >> 8) ^ right) & 0x00ff00ffL;
right ^= work;
leftt ^= (work << 8);
work = ((leftt >> 2) ^ right) & 0x33333333L;
right ^= work;
leftt ^= (work << 2);
work = ((right >> 16) ^ leftt) & 0x0000ffffL;
leftt ^= work;
right ^= (work << 16);
work = ((right >> 4) ^ leftt) & 0x0f0f0f0fL;
leftt ^= work;
right ^= (work << 4);
```

```
        *block++ = right;
        *block = leftt;
        return;
}

/* Validation sets:
 *
 * Single-length key, single-length plaintext -
 * Key    : 0123 4567 89ab cdef
 * Plain  : 0123 4567 89ab cde7
 * Cipher : c957 4425 6a5e d31d
 *
 ********************************************************************/

void des_key(des_ctx *dc, unsigned char *key){
        deskey(key,EN0);
        cpkey(dc->ek);
        deskey(key,DE1);
        cpkey(dc->dk);
}

/* Encrypt several blocks in ECB mode.  Caller is responsible for
   short blocks. */
void des_enc(des_ctx *dc, unsigned char *data, int blocks){
        unsigned long work[2];
        int i;
        unsigned char *cp;

        cp = data;
        for(i=0;i<blocks;i++){
                scrunch(cp,work);
                desfunc(work,dc->ek);
                unscrun(work,cp);
                cp+=8;
        }
}

void des_dec(des_ctx *dc, unsigned char *data, int blocks){
        unsigned long work[2];
        int i;
        unsigned char *cp;

        cp = data;
        for(i=0;i<blocks;i++){
                scrunch(cp,work);
                desfunc(work,dc->dk);
                unscrun(work,cp);
                cp+=8;
        }
}

void main(void){
        des_ctx dc;
        int i;
        unsigned long data[10];
        char *cp,key[8] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
        char x[8] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xe7};


        cp = x;

        des_key(&dc,key);
```

```
        des_enc(&dc,cp,1);
        printf("Enc(0..7,0..7) = ");
        for(i=0;i<8;i++) printf("%02x ", ((unsigned int) cp[i])&0x00ff);
        printf("\n");

        des_dec(&dc,cp,1);

        printf("Dec(above,0..7) = ");
        for(i=0;i<8;i++) printf("%02x ",((unsigned int)cp[i])&0x00ff);
        printf("\n");

        cp = (char *) data;
        for(i=0;i<10;i++)data[i]=i;

        des_enc(&dc,cp,5); /* Enc 5 blocks. */
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                  i/2,data[i],data[i+1]);

        des_dec(&dc,cp,1);
        des_dec(&dc,cp+8,4);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                  i/2,data[i],data[i+1]);

}
```

LOKI91

```
#include <stdio.h>

#define LOKIBLK        8             /* No of bytes in a LOKI data-block
*/
#define ROUNDS         16            /* No of LOKI rounds
*/

typedef unsigned long          Long;   /* type specification for aligned
LOKI blocks */

extern Long    lokikey[2];     /* 64-bit key used by LOKI routines          */
extern char    *loki_lib_ver;         /* String with version no. &
copyright       */

#ifdef __STDC__                       /* declare prototypes for library
functions  */
extern void enloki(char *b);
extern void deloki(char *b);
extern void setlokikey(char key[LOKIBLK]);
#else                          /* else just declare library functions extern
*/
extern void enloki(), deloki(), setlokikey();
#endif __STDC__

char P[32] = {
        31, 23, 15, 7, 30, 22, 14, 6,
        29, 21, 13, 5, 28, 20, 12, 4,
        27, 19, 11, 3, 26, 18, 10, 2,
        25, 17, 9, 1, 24, 16, 8, 0
        };
```

```
typedef        struct {
        short  gen;            /* irreducible polynomial used in this field */
        short  exp;            /* exponent used to generate this s function */
        } sfn_desc;

sfn_desc sfn[] = {
        { /* 101110111 */ 375, 31}, { /* 101111011 */ 379, 31},
        { /* 110000111 */ 391, 31}, { /* 110001011 */ 395, 31},
        { /* 110001101 */ 397, 31}, { /* 110011111 */ 415, 31},
        { /* 110100011 */ 419, 31}, { /* 110101001 */ 425, 31},
        { /* 110110001 */ 433, 31}, { /* 110111101 */ 445, 31},
        { /* 111000011 */ 451, 31}, { /* 111001111 */ 463, 31},
        { /* 111010111 */ 471, 31}, { /* 111011101 */ 477, 31},
        { /* 111100111 */ 487, 31}, { /* 111110011 */ 499, 31},
        { 00, 00}      };

typedef struct {
        Long loki_subkeys[ROUNDS];
} loki_ctx;

static Long    f();                    /* declare LOKI function f */
static short   s();                    /* declare LOKI S-box fn s */

#define ROL12(b) b = ((b << 12) | (b >> 20));
#define ROL13(b) b = ((b << 13) | (b >> 19));

#ifdef  LITTLE_ENDIAN
#define bswap(cb) {                                \
        register char   c;                         \
        c = cb[0]; cb[0] = cb[3]; cb[3] = c;       \
        c = cb[1]; cb[1] = cb[2]; cb[2] = c;       \
        c = cb[4]; cb[4] = cb[7]; cb[7] = c;       \
        c = cb[5]; cb[5] = cb[6]; cb[6] = c;       \
}
#endif

void
setlokikey(loki_ctx *c, char *key)
{
        register       i;
        register Long  KL, KR;

#ifdef LITTLE_ENDIAN
        bswap(key);                    /* swap bytes round if little-endian */
#endif
        KL = ((Long *)key)[0];
        KR = ((Long *)key)[1];

        for (i=0; i<ROUNDS; i+=4) {           /* Generate the 16 subkeys */
            c->loki_subkeys[i] = KL;
            ROL12 (KL);
            c->loki_subkeys[i+1] = KL;
            ROL13 (KL);
            c->loki_subkeys[i+2] = KR;
            ROL12 (KR);
            c->loki_subkeys[i+3] = KR;
            ROL13 (KR);
        }

#ifdef LITTLE_ENDIAN
        bswap(key);                    /* swap bytes back if little-endian */
#endif
```

```
}

void
enloki (loki_ctx *c, char *b)
{
        register       i;
        register Long  L, R;          /* left & right data halves  */

#ifdef LITTLE_ENDIAN
        bswap(b);                     /* swap bytes round if little-endian */
#endif

        L = ((Long *)b)[0];
        R = ((Long *)b)[1];

        for (i=0; i<ROUNDS; i+=2) {           /* Encrypt with the 16 subkeys
*/
            L ^= f (R, c->loki_subkeys[i]);
            R ^= f (L, c->loki_subkeys[i+1]);
        }

        ((Long *)b)[0] = R;           /* Y = swap(LR) */
        ((Long *)b)[1] = L;

#ifdef LITTLE_ENDIAN
        bswap(b);                     /* swap bytes round if little-endian */
#endif
}

void
deloki(loki_ctx *c, char *b)
{
        register       i;
        register Long  L, R;                  /* left & right data halves  */

#ifdef LITTLE_ENDIAN
        bswap(b);                     /* swap bytes round if little-endian */
#endif

        L = ((Long *)b)[0];                   /* LR = X XOR K */
        R = ((Long *)b)[1];

        for (i=ROUNDS; i>0; i-=2) {                   /* subkeys in reverse
order */
            L ^= f(R, c->loki_subkeys[i-1]);
            R ^= f(L, c->loki_subkeys[i-2]);
        }

        ((Long *)b)[0] = R;                   /* Y = LR XOR K */
        ((Long *)b)[1] = L;
}

#define MASK12         0x0fff                 /* 12 bit mask for expansion E
*/

static Long
f(r, k)
register Long  r;      /* Data value R(i-1) */
Long           k;      /* Key     K(i)   */
{
        Long   a, b, c;               /* 32 bit S-box output, & P output */
```

```
        a = r ^ k;                      /* A = R(i-1) XOR K(i) */

        /* want to use slow speed/small size version */
        b = ((Long)s((a         & MASK12))      ) | /* B = S(E(R(i-1))^K(i))
*/
            ((Long)s(((a >>  8) & MASK12)) <<  8) |
            ((Long)s(((a >> 16) & MASK12)) << 16) |
            ((Long)s((((a >> 24) | (a << 8)) & MASK12)) << 24);

        perm32(&c, &b, P);              /* C = P(S( E(R(i-1)) XOR K(i))) */

        return(c);                      /* f returns the result C */
}

static short s(i)
register Long i;        /* return S-box value for input i */
{
        register short r, c, v, t;
        short  exp8();                  /* exponentiation routine for GF(2^8) */

        r = ((i>>8) & 0xc) | (i & 0x3);             /* row value-top 2 &
bottom 2 */
        c = (i>>2) & 0xff;                          /* column
value-middle 8 bits */
        t = (c + ((r * 17) ^ 0xff)) & 0xff;         /* base value for Sfn */
        v = exp8(t, sfn[r].exp, sfn[r].gen);        /* Sfn[r] = t ^ exp
mod gen */
        return(v);
}

#define        MSB     0x80000000L          /* MSB of 32-bit word */

perm32(out, in , perm)
Long    *out;           /* Output 32-bit block to be permuted              */
Long    *in;            /* Input  32-bit block after permutation           */
char    perm[32];       /* Permutation array                               */
{
        Long   mask = MSB;                  /* mask used to set bit in
output    */
        register int   i, o, b;         /* input bit no, output bit no, value */
        register char  *p = perm;       /* ptr to permutation array  */

        *out = 0;                       /* clear output block */
        for (o=0; o<32; o++) {                  /* For each output bit
position o */
                i =(int)*p++;                   /* get input bit permuted to
output o */
                b = (*in >> i) & 01;            /* value of input bit i */
                if (b)                  /* If the input bit i is set */
                        *out |= mask;                   /*  OR in mask to
output i */
                mask >>= 1;                             /* Shift mask to next
bit    */
        }
}

#define SIZE 256                /* 256 elements in GF(2^8) */

short mult8(a, b, gen)
short   a, b;           /* operands for multiply */
short   gen;            /* irreducible polynomial generating Galois Field */
{
```

```
        short  product = 0;           /* result of multiplication */

        while(b != 0) {                       /* while multiplier is
non-zero */
                if (b & 01)
                        product ^= a;         /*   add multiplicand if LSB
of b set */
                a <<= 1;              /*   shift multiplicand one place */
                if (a >= SIZE)
                        a ^= gen;     /*   and modulo reduce if needed */
                b >>= 1;              /*   shift multiplier one place  */
        }
        return(product);
}

short exp8(base, exponent, gen)
short   base;           /* base of exponentiation      */
short   exponent;       /* exponent                    */
short   gen;            /* irreducible polynomial generating Galois Field */
{
        short  accum = base;          /* superincreasing sequence of base */
        short  result = 1;     /* result of exponentiation       */

        if (base == 0)                /* if zero base specified then      */
                return(0);      /* the result is "0" if base = 0    */

        while (exponent != 0) {       /* repeat while exponent non-zero */
                if (( exponent & 0x0001) == 0x0001)           /* multiply if
exp 1 */
                        result = mult8(result, accum, gen);
                exponent >>= 1;               /* shift exponent to next
digit */
                accum = mult8(accum, accum, gen);             /* & square  */
        }
        return(result);
}

void loki_key(loki_ctx *c, unsigned char *key){
        setlokikey(c,key);
}

void loki_enc(loki_ctx *c, unsigned char *data, int blocks){
        unsigned char *cp;
        int i;

        cp = data;
        for(i=0;i<blocks;i++){
                enloki(c,cp);
                cp+=8;
        }
}

void loki_dec(loki_ctx *c, unsigned char *data, int blocks){
        unsigned char *cp;
        int i;

        cp = data;
        for(i=0;i<blocks;i++){
                deloki(c,cp);
                cp+=8;
        }
}
```

```
void main(void){
        loki_ctx lc;
        unsigned long data[10];
        unsigned char *cp;
        unsigned char key[] = {0,1,2,3,4,5,6,7};
        int i;

        for(i=0;i<10;i++) data[i]=i;

        loki_key(&lc,key);

        cp = (char *)data;
        loki_enc(&lc,cp,5);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx\n",
                        i/2,data[i],data[i+1]);
        loki_dec(&lc,cp,1);
        loki_dec(&lc,cp+8,4);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx\n",
                        i/2,data[i],data[i+1]);


}
```

IDEA

```
typedef unsigned char boolean;      /* values are TRUE or FALSE */
typedef unsigned char byte; /* values are 0-255 */
typedef byte *byteptr;      /* pointer to byte */
typedef char *string;/* pointer to ASCII character string */
typedef unsigned short word16;      /* values are 0-65535 */
typedef unsigned long word32;       /* values are 0-4294967295 */

#ifndef TRUE
#define FALSE 0
#define TRUE (!FALSE)
#endif /* if TRUE not already defined */

#ifndef min   /* if min macro not already defined */
#define min(a,b) ( (a)<(b) ? (a) : (b) )
#define max(a,b) ( (a)>(b) ? (a) : (b) )
#endif /* if min macro not already defined */

#define IDEAKEYSIZE 16
#define IDEABLOCKSIZE 8

#define IDEAROUNDS 8
#define IDEAKEYLEN (6*IDEAROUNDS+4)

typedef struct{
       word16 ek[IDEAKEYLEN],dk[IDEAKEYLEN];
}idea_ctx;

/* End includes for IDEA.C */
#ifdef IDEA32        /* Use >16-bit temporaries */
#define low16(x) ((x) & 0xFFFF)
typedef unsigned int uint16;/* at LEAST 16 bits, maybe more */
#else
#define low16(x) (x) /* this is only ever applied to uint16's */
typedef word16 uint16;
```

```
#endif

#ifdef SMALL_CACHE
static uint16
mul(register uint16 a, register uint16 b)
{
	register word32 p;

	p = (word32)a * b;
	if (p) {
		b = low16(p);
		a = p>>16;
		return (b - a) + (b < a);
	} else if (a) {
		return 1-b;
	} else {
		return 1-a;
	}
} /* mul */
#endif /* SMALL_CACHE */

static uint16
mulInv(uint16 x)
{
	uint16 t0, t1;
	uint16 q, y;

	if (x <= 1)
		return x;	/* 0 and 1 are self-inverse */
	t1 = 0x10001L / x;	/* Since x >= 2, this fits into 16 bits */
	y = 0x10001L % x;
	if (y == 1)
		return low16(1-t1);
	t0 = 1;
	do {
		q = x / y;
		x = x % y;
		t0 += q * t1;
		if (x == 1)
			return t0;
		q = y / x;
		y = y % x;
		t1 += q * t0;
	} while (y != 1);
	return low16(1-t1);
} /* mukInv */

static void
ideaExpandKey(byte const *userkey, word16 *EK)
{
	int i,j;

	for (j=0; j<8; j++) {
		EK[j] = (userkey[0]<<8) + userkey[1];
		userkey += 2;
	}
	for (i=0; j < IDEAKEYLEN; j++) {
		i++;
		EK[i+7] = EK[i & 7] << 9 | EK[i+1 & 7] >> 7;
		EK += i & 8;
		i &= 7;
	}
```

```
} /* ideaExpandKey */

static void
ideaInvertKey(word16 const *EK, word16 DK[IDEAKEYLEN])
{
	int i;
	uint16 t1, t2, t3;
	word16 temp[IDEAKEYLEN];
	word16 *p = temp + IDEAKEYLEN;

	t1 = mulInv(*EK++);
	t2 = -*EK++;
	t3 = -*EK++;
	*--p = mulInv(*EK++);
	*--p = t3;
	*--p = t2;
	*--p = t1;

	for (i = 0; i < IDEAROUNDS-1; i++) {
		t1 = *EK++;
		*--p = *EK++;
		*--p = t1;

		t1 = mulInv(*EK++);
		t2 = -*EK++;
		t3 = -*EK++;
		*--p = mulInv(*EK++);
		*--p = t2;
		*--p = t3;
		*--p = t1;
	}
	t1 = *EK++;
	*--p = *EK++;
	*--p = t1;

	t1 = mulInv(*EK++);
	t2 = -*EK++;
	t3 = -*EK++;
	*--p = mulInv(*EK++);
	*--p = t3;
	*--p = t2;
	*--p = t1;
/* Copy and destroy temp copy */
	memcpy(DK, temp, sizeof(temp));
	for(i=0;i<IDEAKEYLEN;i++)temp[i]=0;
} /* ideaInvertKey */

#ifdef SMALL_CACHE
#define MUL(x,y) (x = mul(low16(x),y))
#else /* !SMALL_CACHE */
#ifdef AVOID_JUMPS
#define MUL(x,y) (x = low16(x-1), t16 = low16((y)-1), \
		t32 = (word32)x*t16 + x + t16 + 1, x = low16(t32), \
		t16 = t32>>16, x = (x-t16) + (x<t16) )
#else /* !AVOID_JUMPS (default) */
#define MUL(x,y) \
	((t16 = (y)) ? \
		(x=low16(x)) ? \
			t32 = (word32)x*t16, \
			x = low16(t32), \
			t16 = t32>>16, \
			x = (x-t16)+(x<t16) \
```

```
                : \
                        (x = 1-t16) \
        : \
                (x = 1-x))
#endif
#endif

static void
ideaCipher(byte *inbuf, byte *outbuf, word16 *key)
{
        register uint16 x1, x2, x3, x4, s2, s3;
        word16 *in, *out;
#ifndef SMALL_CACHE
        register uint16 t16; /* Temporaries needed by MUL macro */
        register word32 t32;
#endif
        int r = IDEAROUNDS;

        in = (word16 *)inbuf;
        x1 = *in++;  x2 = *in++;
        x3 = *in++;  x4 = *in;
#ifndef HIGHFIRST
        x1 = (x1 >>8) | (x1<<8);
        x2 = (x2 >>8) | (x2<<8);
        x3 = (x3 >>8) | (x3<<8);
        x4 = (x4 >>8) | (x4<<8);
#endif
        do {
                MUL(x1,*key++);
                x2 += *key++;
                x3 += *key++;
                MUL(x4, *key++);

                s3 = x3;
                x3 ^= x1;
                MUL(x3, *key++);
                s2 = x2;
                x2 ^= x4;
                x2 += x3;
                MUL(x2, *key++);
                x3 += x2;

                x1 ^= x2;  x4 ^= x3;

                x2 ^= s3;  x3 ^= s2;
        } while (--r);
        MUL(x1, *key++);
        x3 += *key++;
        x2 += *key++;
        MUL(x4, *key);

        out = (word16 *)outbuf;
#ifdef HIGHFIRST
        *out++ = x1;
        *out++ = x3;
        *out++ = x2;
        *out = x4;
#else /* !HIGHFIRST */
        *out++ = (x1 >>8) | (x1<<8);
        *out++ = (x3 >>8) | (x3<<8);
        *out++ = (x2 >>8) | (x2<<8);
        *out = (x4 >>8) | (x4<<8);
```

```
#endif
} /* ideaCipher */

void idea_key(idea_ctx *c, unsigned char *key){
       ideaExpandKey(key,c->ek);
       ideaInvertKey(c->ek,c->dk);
}

void idea_enc(idea_ctx *c, unsigned char *data, int blocks){
       int i;
       unsigned char *d = data;

       for(i=0;i<blocks;i++){
              ideaCipher(d,d,c->ek);
              d+=8;
       }
}

void idea_dec(idea_ctx *c, unsigned char *data, int blocks){
       int i;
       unsigned char *d = data;

       for(i=0;i<blocks;i++){
              ideaCipher(d,d,c->dk);
              d+=8;
       }
}

#include <stdio.h>

#ifndef BLOCKS
#ifndef KBYTES
#define KBYTES 1024
#endif
#define BLOCKS (64*KBYTES)
#endif

int
main(void)
{      /* Test driver for IDEA cipher */
       int i, j, k;
       idea_ctx c;
       byte userkey[16];
       word16 EK[IDEAKEYLEN], DK[IDEAKEYLEN];
       byte XX[8], YY[8], ZZ[8];
       word32 long_block[10]; /* 5 blocks */
       long l;
       char *lbp;

       /* Make a sample user key for testing... */
       for(i=0; i<16; i++)
              userkey[i] = i+1;

       idea_key(&c,userkey);

       /* Make a sample plaintext pattern for testing... */
       for (k=0; k<8; k++)
              XX[k] = k;

       idea_enc(&c,XX,1); /* encrypt */

       lbp = (unsigned char *) long_block;
```

```
        for(i=0;i<10;i++) long_block[i] = i;
        idea_enc(&c,lbp,5);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                  i/2,long_block[i],long_block[i+1]);

        idea_dec(&c,lbp,3);
        idea_dec(&c,lbp+24,2);

        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                  i/2,long_block[i],long_block[i+1]);

        return 0;       /* normal exit */
} /* main */
```

GOST

```
typedef unsigned long u4;
typedef unsigned char byte;

typedef struct {
        u4 k[8];
        /* Constant s-boxes -- set up in gost_init(). */
        char k87[256],k65[256],k43[256],k21[256];
} gost_ctx;

/* Note:  encrypt and decrypt expect full blocks--padding blocks is
          caller's responsibility.  All bulk encryption is done in
          ECB mode by these calls.  Other modes may be added easily
          enough.                                                    */
void gost_enc(gost_ctx *, u4 *, int);
void gost_dec(gost_ctx *, u4 *, int);
void gost_key(gost_ctx *, u4 *);
void gost_init(gost_ctx *);
void gost_destroy(gost_ctx *);

#ifdef __alpha  /* Any other 64-bit machines? */
typedef unsigned int word32;
#else
typedef unsigned long word32;
#endif


kboxinit(gost_ctx *c)
{
        int i;

        byte k8[16] = {14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6,
                       12,  5,  9,  0,  7 };
        byte k7[16] = {15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2,
                       13, 12,  0,  5, 10 };
        byte k6[16] = {10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,
                        7, 11,  4,  2,  8 };
        byte k5[16] = { 7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,
                        5, 11, 12,  4, 15 };
        byte k4[16] = { 2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3,
                       15, 13,  0, 14,  9 };
        byte k3[16] = {12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,
                        4, 14,  7,  5, 11 };
```

```
        byte k2[16] = { 4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,
                        7,  5, 10,  6,  1 };
        byte k1[16] = {13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3,
                       14,  5,  0, 12,  7 };

     for (i = 0; i < 256; i++) {
                c->k87[i] = k8[i >> 4] << 4 | k7[i & 15];
                c->k65[i] = k6[i >> 4] << 4 | k5[i & 15];
                c->k43[i] = k4[i >> 4] << 4 | k3[i & 15];
                c->k21[i] = k2[i >> 4] << 4 | k1[i & 15];
     }
}

static word32
f(gost_ctx *c,word32 x)
{
        x = c->k87[x>>24 & 255] << 24 | c->k65[x>>16 & 255] << 16 |
            c->k43[x>> 8 & 255] <<  8 | c->k21[x & 255];

     /* Rotate left 11 bits */
     return x<<11 | x>>(32-11);
}

void gostcrypt(gost_ctx *c, word32 *d){
     register word32 n1, n2; /* As named in the GOST */

        n1 = d[0];
        n2 = d[1];

     /* Instead of swapping halves, swap names each round */
        n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
        n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
        n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
        n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

        n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
        n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
        n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
        n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

        n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
        n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
        n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
        n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

        n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
        n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        d[0] = n2; d[1] = n1;
}

void
gostdecrypt(gost_ctx *c, u4 *d){
     register word32 n1, n2; /* As named in the GOST */

        n1 = d[0]; n2 = d[1];

        n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
        n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
        n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
```

```
        n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

        n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
        n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
        n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
        n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        d[0] = n2; d[1] = n1;
}

void gost_enc(gost_ctx *c, u4 *d, int blocks){
        int i;

        for(i=0;i<blocks;i++){
                gostcrypt(c,d);
                d+=2;
        }
}

void gost_dec(gost_ctx *c, u4 *d, int blocks){
        int i;

        for(i=0;i<blocks;i++){
                gostdecrypt(c,d);
                d+=2;
        }
}

void gost_key(gost_ctx *c, u4 *k){
        int i;
        for(i=0;i<8;i++) c->k[i]=k[i];
}

void gost_init(gost_ctx *c){
        kboxinit(c);
}

void gost_destroy(gost_ctx *c){
        int i;
        for(i=0;i<8;i++) c->k[i]=0;
}

void main(void){
        gost_ctx gc;
        u4 k[8],data[10];
        int i;

        /* Initialize GOST context. */
        gost_init(&gc);

        /* Prepare key--a simple key should be OK, with this many rounds! */
        for(i=0;i<8;i++) k[i] = i;
```

```
        gost_key(&gc,k);

        /* Try some test vectors. */
        data[0] = 0; data[1] = 0;
        gostcrypt(&gc,data);
        printf("Enc of zero vector:  %08lx %08lx\n",data[0],data[1]);
        gostcrypt(&gc,data);
        printf("Enc of above:        %08lx %08lx\n",data[0],data[1]);
        data[0] = 0xffffffff; data[1] = 0xffffffff;
        gostcrypt(&gc,data);
        printf("Enc of ones vector:  %08lx %08lx\n",data[0],data[1]);
        gostcrypt(&gc,data);
        printf("Enc of above:        %08lx %08lx\n",data[0],data[1]);

        /* Does gost_dec() properly reverse gost_enc()?  Do
           we deal OK with single-block lengths passed in gost_dec()?
           Do we deal OK with different lengths passed in? */

        /* Init data */
        for(i=0;i<10;i++) data[i]=i;

        /* Encrypt data as 5 blocks. */
        gost_enc(&gc,data,5);

        /* Display encrypted data. */
        for(i=0;i<10;i+=2) printf("Block %02d = %08lx %08lx\n",
                                  i/2,data[i],data[i+1]);

        /* Decrypt in different sized chunks. */
        gost_dec(&gc,data,1);
        gost_dec(&gc,data+2,4);
        printf("\n");

        /* Display decrypted data. */
        for(i=0;i<10;i+=2) printf("Block %02d = %08lx %08lx\n",
                                  i/2,data[i],data[i+1]);

        gost_destroy(&gc);
}
```

BLOWFISH

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef little_endian   /* Eg: Intel */
   #include <alloc.h>
#endif

#include <ctype.h>

#ifdef little_endian   /* Eg: Intel */
   #include <dir.h>
   #include <bios.h>
#endif
```

```
#ifdef big_endian
   #include <Types.h>
#endif

typedef struct {
        unsigned long S[4][256],P[18];
} blf_ctx;


#define MAXKEYBYTES 56          /* 448 bits */
// #define little_endian 1              /* Eg: Intel */
#define big_endian 1            /* Eg: Motorola */

void Blowfish_encipher(blf_ctx *,unsigned long *xl, unsigned long *xr);
void Blowfish_decipher(blf_ctx *,unsigned long *xl, unsigned long *xr);

#define N               16
#define noErr            0
#define DATAERROR         -1
#define KEYBYTES         8

FILE*         SubkeyFile;


unsigned long F(blf_ctx *bc, unsigned long x)
{
   unsigned short a;
   unsigned short b;
   unsigned short c;
   unsigned short d;
   unsigned long  y;

   d = x & 0x00FF;
   x >>= 8;
   c = x & 0x00FF;
   x >>= 8;
   b = x & 0x00FF;
   x >>= 8;
   a = x & 0x00FF;
   //y = ((S[0][a] + S[1][b]) ^ S[2][c]) + S[3][d];
   y = bc->S[0][a] + bc->S[1][b];
   y = y ^ bc->S[2][c];
   y = y + bc->S[3][d];

   return y;
}

void Blowfish_encipher(blf_ctx *c,unsigned long *xl, unsigned long *xr)
{
   unsigned long  Xl;
   unsigned long  Xr;
   unsigned long  temp;
   short          i;

   Xl = *xl;
   Xr = *xr;

   for (i = 0; i < N; ++i) {
      Xl = Xl ^ c->P[i];
      Xr = F(c,Xl) ^ Xr;

      temp = Xl;
```

```
      Xl = Xr;
      Xr = temp;
   }

   temp = Xl;
   Xl = Xr;
   Xr = temp;

   Xr = Xr ^ c->P[N];
   Xl = Xl ^ c->P[N + 1];

   *xl = Xl;
   *xr = Xr;
}

void Blowfish_decipher(blf_ctx *c, unsigned long *xl, unsigned long *xr)
{
   unsigned long  Xl;
   unsigned long  Xr;
   unsigned long  temp;
   short          i;

   Xl = *xl;
   Xr = *xr;

   for (i = N + 1; i > 1; --i) {
      Xl = Xl ^ c->P[i];
      Xr = F(c,Xl) ^ Xr;

      /* Exchange Xl and Xr */
      temp = Xl;
      Xl = Xr;
      Xr = temp;
   }

   /* Exchange Xl and Xr */
   temp = Xl;
   Xl = Xr;
   Xr = temp;

   Xr = Xr ^ c->P[1];
   Xl = Xl ^ c->P[0];

   *xl = Xl;
   *xr = Xr;
}

short InitializeBlowfish(blf_ctx *c, char key[], short keybytes)
{
   short          i;
   short          j;
   short          k;
   short          error;
   short          numread;
   unsigned long  data;
   unsigned long  datal;
   unsigned long  datar;

unsigned long ks0[] = {
0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96,
0xba7c9045, 0xf12c7f99, 0x24a19947, 0xb3916cf7, 0x0801f2e2, 0x858efc16,
0x636920d8, 0x71574e69, 0xa458fea3, 0xf4933d7e, 0x0d95748f, 0x728eb658,
```

```
0x718bcd58, 0x82154aee, 0x7b54a41d, 0xc25a59b5, 0x9c30d539, 0x2af26013,
0xc5d1b023, 0x286085f0, 0xca417918, 0xb8db38ef, 0x8e79dcb0, 0x603a180e,
0x6c9e0e8b, 0xb01e8a3e, 0xd71577c1, 0xbd314b27, 0x78af2fda, 0x55605c60,
0xe65525f3, 0xaa55ab94, 0x57489862, 0x63e81440, 0x55ca396a, 0x2aab10b6,
0xb4cc5c34, 0x1141e8ce, 0xa15486af, 0x7c72e993, 0xb3ee1411, 0x636fbc2a,
0x2ba9c55d, 0x741831f6, 0xce5c3e16, 0x9b87931e, 0xafd6ba33, 0x6c24cf5c,
0x7a325381, 0x28958677, 0x3b8f4898, 0x6b4bb9af, 0xc4bfe81b, 0x66282193,
0x61d809cc, 0xfb21a991, 0x487cac60, 0x5dec8032, 0xef845d5d, 0xe98575b1,
0xdc262302, 0xeb651b88, 0x23893e81, 0xd396acc5, 0x0f6d6ff3, 0x83f44239,
0x2e0b4482, 0xa4842004, 0x69c8f04a, 0x9e1f9b5e, 0x21c66842, 0xf6e96c9a,
0x670c9c61, 0xabd388f0, 0x6a51a0d2, 0xd8542f68, 0x960fa728, 0xab5133a3,
0x6eef0b6c, 0x137a3be4, 0xba3bf050, 0x7efb2a98, 0xa1f1651d, 0x39af0176,
0x66ca593e, 0x82430e88, 0x8cee8619, 0x456f9fb4, 0x7d84a5c3, 0x3b8b5ebe,
0xe06f75d8, 0x85c12073, 0x401a449f, 0x56c16aa6, 0x4ed3aa62, 0x363f7706,
0x1bfedf72, 0x429b023d, 0x37d0d724, 0xd00a1248, 0xdb0fead3, 0x49f1c09b,
0x075372c9, 0x80991b7b, 0x25d479d8, 0xf6e8def7, 0xe3fe501a, 0xb6794c3b,
0x976ce0bd, 0x04c006ba, 0xc1a94fb6, 0x409f60c4, 0x5e5c9ec2, 0x196a2463,
0x68fb6faf, 0x3e6c53b5, 0x1339b2eb, 0x3b52ec6f, 0x6dfc511f, 0x9b30952c,
0xcc814544, 0xaf5ebd09, 0xbee3d004, 0xde334afd, 0x660f2807, 0x192e4bb3,
0xc0cba857, 0x45c8740f, 0xd20b5f39, 0xb9d3fbdb, 0x5579c0bd, 0x1a60320a,
0xd6a100c6, 0x402c7279, 0x679f25fe, 0xfb1fa3cc, 0x8ea5e9f8, 0xdb3222f8,
0x3c7516df, 0xfd616b15, 0x2f501ec8, 0xad0552ab, 0x323db5fa, 0xfd238760,
0x53317b48, 0x3e00df82, 0x9e5c57bb, 0xca6f8ca0, 0x1a87562e, 0xdf1769db,
0xd542a8f6, 0x287effc3, 0xac6732c6, 0x8c4f5573, 0x695b27b0, 0xbbca58c8,
0xe1ffa35d, 0xb8f011a0, 0x10fa3d98, 0xfd2183b8, 0x4afcb56c, 0x2dd1d35b,
0x9a53e479, 0xb6f84565, 0xd28e49bc, 0x4bfb9790, 0xe1ddf2da, 0xa4cb7e33,
0x62fb1341, 0xcee4c6e8, 0xef20cada, 0x36774c01, 0xd07e9efe, 0x2bf11fb4,
0x95dbda4d, 0xae909198, 0xeaad8e71, 0x6b93d5a0, 0xd08ed1d0, 0xafc725e0,
0x8e3c5b2f, 0x8e7594b7, 0x8ff6e2fb, 0xf2122b64, 0x8888b812, 0x900df01c,
0x4fad5ea0, 0x688fc31c, 0xd1cff191, 0xb3a8c1ad, 0x2f2f2218, 0xbe0e1777,
0xea752dfe, 0x8b021fa1, 0xe5a0cc0f, 0xb56f74e8, 0x18acf3d6, 0xce89e299,
0xb4a84fe0, 0xfd13e0b7, 0x7cc43b81, 0xd2ada8d9, 0x165fa266, 0x80957705,
0x93cc7314, 0x211a1477, 0xe6ad2065, 0x77b5fa86, 0xc75442f5, 0xfb9d35cf,
0xebcdaf0c, 0x7b3e89a0, 0xd6411bd3, 0xae1e7e49, 0x00250e2d, 0x2071b35e,
0x226800bb, 0x57b8e0af, 0x2464369b, 0xf009b91e, 0x5563911d, 0x59dfa6aa,
0x78c14389, 0xd95a537f, 0x207d5ba2, 0x02e5b9c5, 0x83260376, 0x6295cfa9,
0x11c81968, 0x4e734a41, 0xb3472dca, 0x7b14a94a, 0x1b510052, 0x9a532915,
0xd60f573f, 0xbc9bc6e4, 0x2b60a476, 0x81e67400, 0x08ba6fb5, 0x571be91f,
0xf296ec6b, 0x2a0dd915, 0xb6636521, 0xe7b9f9b6, 0xff34052e, 0xc5855664,
0x53b02d5d, 0xa99f8fa1, 0x08ba4799, 0x6e85076a};
unsigned long ks1[] = {
0x4b7a70e9, 0xb5b32944, 0xdb75092e, 0xc4192623, 0xad6ea6b0, 0x49a7df7d,
0x9cee60b8, 0x8fedb266, 0xecaa8c71, 0x699a17ff, 0x5664526c, 0xc2b19ee1,
0x193602a5, 0x75094c29, 0xa0591340, 0xe4183a3e, 0x3f54989a, 0x5b429d65,
0x6b8fe4d6, 0x99f73fd6, 0xa1d29c07, 0xefe830f5, 0x4d2d38e6, 0xf0255dc1,
0x4cdd2086, 0x8470eb26, 0x6382e9c6, 0x021ecc5e, 0x09686b3f, 0x3ebaefc9,
0x3c971814, 0x6b6a70a1, 0x687f3584, 0x52a0e286, 0xb79c5305, 0xaa500737,
0x3e07841c, 0x7fdeae5c, 0x8e7d44ec, 0x5716f2b8, 0xb03ada37, 0xf0500c0d,
0xf01c1f04, 0x0200b3ff, 0xae0cf51a, 0x3cb574b2, 0x25837a58, 0xdc0921bd,
0xd19113f9, 0x7ca92ff6, 0x94324773, 0x22f54701, 0x3ae5e581, 0x37c2dadc,
0xc8b57634, 0x9af3dda7, 0xa9446146, 0x0fd0030e, 0xecc8c73e, 0xa4751e41,
0xe238cd99, 0x3bea0e2f, 0x3280bba1, 0x183eb331, 0x4e548b38, 0x4f6db908,
0x6f420d03, 0xf60a04bf, 0x2cb81290, 0x24977c79, 0x5679b072, 0xbcaf89af,
0xde9a771f, 0xd9930810, 0xb38bae12, 0xdccf3f2e, 0x5512721f, 0x2e6b7124,
0x501adde6, 0x9f84cd87, 0x7a584718, 0x7408da17, 0xbc9f9abc, 0xe94b7d8c,
0xec7aec3a, 0xdb851dfa, 0x63094366, 0xc464c3d2, 0xef1c1847, 0x3215d908,
0xdd433b37, 0x24c2ba16, 0x12a14d43, 0x2a65c451, 0x50940002, 0x133ae4dd,
0x71dff89e, 0x10314e55, 0x81ac77d6, 0x5f11199b, 0x043556f1, 0xd7a3c76b,
0x3c11183b, 0x5924a509, 0xf28fe6ed, 0x97f1fbfa, 0x9ebabf2c, 0x1e153c6e,
0x86e34570, 0xeae96fb1, 0x860e5e0a, 0x5a3e2ab3, 0x771fe71c, 0x4e3d06fa,
0x2965dcb9, 0x99e71d0f, 0x803e89d6, 0x5266c825, 0x2e4cc978, 0x9c10b36a,
0xc6150eba, 0x94e2ea78, 0xa5fc3c53, 0x1e0a2df4, 0xf2f74ea7, 0x361d2b3d,
```

```
0x1939260f, 0x19c27960, 0x5223a708, 0xf71312b6, 0xebadfe6e, 0xeac31f66,
0xe3bc4595, 0xa67bc883, 0xb17f37d1, 0x018cff28, 0xc332ddef, 0xbe6c5aa5,
0x65582185, 0x68ab9802, 0xeecea50f, 0xdb2f953b, 0x2aef7dad, 0x5b6e2f84,
0x1521b628, 0x29076170, 0xecdd4775, 0x619f1510, 0x13cca830, 0xeb61bd96,
0x0334fe1e, 0xaa0363cf, 0xb5735c90, 0x4c70a239, 0xd59e9e0b, 0xcbaade14,
0xeecc86bc, 0x60622ca7, 0x9cab5cab, 0xb2f3846e, 0x648b1eaf, 0x19bdf0ca,
0xa02369b9, 0x655abb50, 0x40685a32, 0x3c2ab4b3, 0x319ee9d5, 0xc021b8f7,
0x9b540b19, 0x875fa099, 0x95f7997e, 0x623d7da8, 0xf837889a, 0x97e32d77,
0x11ed935f, 0x16681281, 0x0e358829, 0xc7e61fd6, 0x96dedfa1, 0x7858ba99,
0x57f584a5, 0x1b227263, 0x9b83c3ff, 0x1ac24696, 0xcdb30aeb, 0x532e3054,
0x8fd948e4, 0x6dbc3128, 0x58ebf2ef, 0x34c6ffea, 0xfe28ed61, 0xee7c3c73,
0x5d4a14d9, 0xe864b7e3, 0x42105d14, 0x203e13e0, 0x45eee2b6, 0xa3aaabea,
0xdb6c4f15, 0xfacb4fd0, 0xc742f442, 0xef6abbb5, 0x654f3b1d, 0x41cd2105,
0xd81e799e, 0x86854dc7, 0xe44b476a, 0x3d816250, 0xcf62a1f2, 0x5b8d2646,
0xfc8883a0, 0xc1c7b6a3, 0x7f1524c3, 0x69cb7492, 0x47848a0b, 0x5692b285,
0x095bbf00, 0xad19489d, 0x1462b174, 0x23820e00, 0x58428d2a, 0x0c55f5ea,
0x1dadf43e, 0x233f7061, 0x3372f092, 0x8d937e41, 0xd65fecf1, 0x6c223bdb,
0x7cde3759, 0xcbee7460, 0x4085f2a7, 0xce77326e, 0xa6078084, 0x19f8509e,
0xe8efd855, 0x61d99735, 0xa969a7aa, 0xc50c06c2, 0x5a04abfc, 0x800bcadc,
0x9e447a2e, 0xc3453484, 0xfdd56705, 0x0e1e9ec9, 0xdb73dbd3, 0x105588cd,
0x675fda79, 0xe3674340, 0xc5c43465, 0x713e38d8, 0x3d28f89e, 0xf16dff20,
0x153e21e7, 0x8fb03d4a, 0xe6e39f2b, 0xdb83adf7};
unsigned long ks2[] = {
0xe93d5a68, 0x948140f7, 0xf64c261c, 0x94692934, 0x411520f7, 0x7602d4f7,
0xbcf46b2e, 0xd4a20068, 0xd4082471, 0x3320f46a, 0x43b7d4b7, 0x500061af,
0x1e39f62e, 0x97244546, 0x14214f74, 0xbf8b8840, 0x4d95fc1d, 0x96b591af,
0x70f4ddd3, 0x66a02f45, 0xbfbc09ec, 0x03bd9785, 0x7fac6dd0, 0x31cb8504,
0x96eb27b3, 0x55fd3941, 0xda2547e6, 0xabca0a9a, 0x28507825, 0x530429f4,
0x0a2c86da, 0xe9b66dfb, 0x68dc1462, 0xd7486900, 0x680ec0a4, 0x27a18dee,
0x4f3ffea2, 0xe887ad8c, 0xb58ce006, 0x7af4d6b6, 0xaace1e7c, 0xd3375fec,
0xce78a399, 0x406b2a42, 0x20fe9e35, 0xd9f385b9, 0xee39d7ab, 0x3b124e8b,
0x1dc9faf7, 0x4b6d1856, 0x26a36631, 0xeae397b2, 0x3a6efa74, 0xdd5b4332,
0x6841e7f7, 0xca7820fb, 0xfb0af54e, 0xd8feb397, 0x454056ac, 0xba489527,
0x55533a3a, 0x20838d87, 0xfe6ba9b7, 0xd096954b, 0x55a867bc, 0xa1159a58,
0xcca92963, 0x99e1db33, 0xa62a4a56, 0x3f3125f9, 0x5ef47e1c, 0x9029317c,
0xfdf8e802, 0x04272f70, 0x80bb155c, 0x05282ce3, 0x95c11548, 0xe4c66d22,
0x48c1133f, 0xc70f86dc, 0x07f9c9ee, 0x41041f0f, 0x404779a4, 0x5d886e17,
0x325f51eb, 0xd59bc0d1, 0xf2bcc18f, 0x41113564, 0x257b7834, 0x602a9c60,
0xdff8e8a3, 0x1f636c1b, 0x0e12b4c2, 0x02e1329e, 0xaf664fd1, 0xcad18115,
0x6b2395e0, 0x333e92e1, 0x3b240b62, 0xeebeb922, 0x85b2a20e, 0xe6ba0d99,
0xde720c8c, 0x2da2f728, 0xd0127845, 0x95b794fd, 0x647d0862, 0xe7ccf5f0,
0x5449a36f, 0x877d48fa, 0xc39dfd27, 0xf33e8d1e, 0x0a476341, 0x992eff74,
0x3a6f6eab, 0xf4f8fd37, 0xa812dc60, 0xa1ebddf8, 0x991be14c, 0xdb6e6b0d,
0xc67b5510, 0x6d672c37, 0x2765d43b, 0xdcd0e804, 0xf1290dc7, 0xcc00ffa3,
0xb5390f92, 0x690fed0b, 0x667b9ffb, 0xcedb7d9c, 0xa091cf0b, 0xd9155ea3,
0xbb132f88, 0x515bad24, 0x7b9479bf, 0x763bd6eb, 0x37392eb3, 0xcc115979,
0x8026e297, 0xf42e312d, 0x6842ada7, 0xc66a2b3b, 0x12754ccc, 0x782ef11c,
0x6a124237, 0xb79251e7, 0x06a1bbe6, 0x4bfb6350, 0x1a6b1018, 0x11caedfa,
0x3d25bdd8, 0xe2e1c3c9, 0x44421659, 0x0a121386, 0xd90cec6e, 0xd5abea2a,
0x64af674e, 0xda86a85f, 0xbebfe988, 0x64e4c3fe, 0x9dbc8057, 0xf0f7c086,
0x60787bf8, 0x6003604d, 0xd1fd8346, 0xf6381fb0, 0x7745ae04, 0xd736fccc,
0x83426b33, 0xf01eab71, 0xb0804187, 0x3c005e5f, 0x77a057be, 0xbde8ae24,
0x55464299, 0xbf582e61, 0x4e58f48f, 0xf2ddfda2, 0xf474ef38, 0x8789bdc2,
0x5366f9c3, 0xc8b38e74, 0xb475f255, 0x46fcd9b9, 0x7aeb2661, 0x8b1ddf84,
0x846a0e79, 0x915f95e2, 0x466e598e, 0x20b45770, 0x8cd55591, 0xc902de4c,
0xb90bace1, 0xbb8205d0, 0x11a86248, 0x7574a99e, 0xb77f19b6, 0xe0a9dc09,
0x662d09a1, 0xc4324633, 0xe85a1f02, 0x09f0be8c, 0x4a99a025, 0x1d6efe10,
0x1ab93d1d, 0x0ba5a4df, 0xa186f20f, 0x2868f169, 0xdcb7da83, 0x573906fe,
0xa1e2ce9b, 0x4fcd7f52, 0x50115e01, 0xa70683fa, 0xa002b5c4, 0x0de6d027,
0x9af88c27, 0x773f8641, 0xc3604c06, 0x61a806b5, 0xf0177a28, 0xc0f586e0,
0x006058aa, 0x30dc7d62, 0x11e69ed7, 0x2338ea63, 0x53c2dd94, 0xc2c21634,
0xbbcbee56, 0x90bcb6de, 0xebfc7da1, 0xce591d76, 0x6f05e409, 0x4b7c0188,
```

```
0x39720a3d, 0x7c927c24, 0x86e3725f, 0x724d9db9, 0x1ac15bb4, 0xd39eb8fc,
0xed545578, 0x08fca5b5, 0xd83d7cd3, 0x4dad0fc4, 0x1e50ef5e, 0xb161e6f8,
0xa28514d9, 0x6c51133c, 0x6fd5c7e7, 0x56e14ec4, 0x362abfce, 0xddc6c837,
0xd79a3234, 0x92638212, 0x670efa8e, 0x406000e0};
unsigned long ks3[] = {
0x3a39ce37, 0xd3faf5cf, 0xabc27737, 0x5ac52d1b, 0x5cb0679e, 0x4fa33742,
0xd3822740, 0x99bc9bbe, 0xd5118e9d, 0xbf0f7315, 0xd62d1c7e, 0xc700c47b,
0xb78c1b6b, 0x21a19045, 0xb26eb1be, 0x6a366eb4, 0x5748ab2f, 0xbc946e79,
0xc6a376d2, 0x6549c2c8, 0x530ff8ee, 0x468dde7d, 0xd5730a1d, 0x4cd04dc6,
0x2939bbdb, 0xa9ba4650, 0xac9526e8, 0xbe5ee304, 0xa1fad5f0, 0x6a2d519a,
0x63ef8ce2, 0x9a86ee22, 0xc089c2b8, 0x43242ef6, 0xa51e03aa, 0x9cf2d0a4,
0x83c061ba, 0x9be96a4d, 0x8fe51550, 0xba645bd6, 0x2826a2f9, 0xa73a3ae1,
0x4ba99586, 0xef5562e9, 0xc72fefd3, 0xf752f7da, 0x3f046f69, 0x77fa0a59,
0x80e4a915, 0x87b08601, 0x9b09e6ad, 0x3b3ee593, 0xe990fd5a, 0x9e34d797,
0x2cf0b7d9, 0x022b8b51, 0x96d5ac3a, 0x017da67d, 0xd1cf3ed6, 0x7c7d2d28,
0x1f9f25cf, 0xadf2b89b, 0x5ad6b472, 0x5a88f54c, 0xe029ac71, 0xe019a5e6,
0x47b0acfd, 0xed93fa9b, 0xe8d3c48d, 0x283b57cc, 0xf8d56629, 0x79132e28,
0x785f0191, 0xed756055, 0xf7960e44, 0xe3d35e8c, 0x15056dd4, 0x88f46dba,
0x03a16125, 0x0564f0bd, 0xc3eb9e15, 0x3c9057a2, 0x97271aec, 0xa93a072a,
0x1b3f6d9b, 0x1e6321f5, 0xf59c66fb, 0x26dcf319, 0x7533d928, 0xb155fdf5,
0x03563482, 0x8aba3cbb, 0x28517711, 0xc20ad9f8, 0xabcc5167, 0xccad925f,
0x4de81751, 0x3830dc8e, 0x379d5862, 0x9320f991, 0xea7a90c2, 0xfb3e7bce,
0x5121ce64, 0x774fbe32, 0xa8b6e37e, 0xc3293d46, 0x48de5369, 0x6413e680,
0xa2ae0810, 0xdd6db224, 0x69852dfd, 0x09072166, 0xb39a460a, 0x6445c0dd,
0x586cdecf, 0x1c20c8ae, 0x5bbef7dd, 0x1b588d40, 0xccd2017f, 0x6bb4e3bb,
0xdda26a7e, 0x3a59ff45, 0x3e350a44, 0xbcb4cdd5, 0x72eacea8, 0xfa6484bb,
0x8d6612ae, 0xbf3c6f47, 0xd29be463, 0x542f5d9e, 0xaec2771b, 0xf64e6370,
0x740e0d8d, 0xe75b1357, 0xf8721671, 0xaf537d5d, 0x4040cb08, 0x4eb4e2cc,
0x34d2466a, 0x0115af84, 0xe1b00428, 0x95983a1d, 0x06b89fb4, 0xce6ea048,
0x6f3f3b82, 0x3520ab82, 0x011a1d4b, 0x277227f8, 0x611560b1, 0xe7933fdc,
0xbb3a792b, 0x344525bd, 0xa08839e1, 0x51ce794b, 0x2f32c9b7, 0xa01fbac9,
0xe01cc87e, 0xbcc7d1f6, 0xcf0111c3, 0xa1e8aac7, 0x1a908749, 0xd44fbd9a,
0xd0dadecb, 0xd50ada38, 0x0339c32a, 0xc6913667, 0x8df9317c, 0xe0b12b4f,
0xf79e59b7, 0x43f5bb3a, 0xf2d519ff, 0x27d9459c, 0xbf97222c, 0x15e6fc2a,
0x0f91fc71, 0x9b941525, 0xfae59361, 0xceb69ceb, 0xc2a86459, 0x12baa8d1,
0xb6c1075e, 0xe3056a0c, 0x10d25065, 0xcb03a442, 0xe0ec6e0e, 0x1698db3b,
0x4c98a0be, 0x3278e964, 0x9f1f9532, 0xe0d392df, 0xd3a0342b, 0x8971f21e,
0x1b0a7441, 0x4ba3348c, 0xc5be7120, 0xc37632d8, 0xdf359f8d, 0x9b992f2e,
0xe60b6f47, 0x0fe3f11d, 0xe54cda54, 0x1edad891, 0xce6279cf, 0xcd3e7e6f,
0x1618b166, 0xfd2c1d05, 0x848fd2c5, 0xf6fb2299, 0xf523f357, 0xa6327623,
0x93a83531, 0x56cccd02, 0xacf08162, 0x5a75ebb5, 0x6e163697, 0x88d273cc,
0xde966292, 0x81b949d0, 0x4c50901b, 0x71c65614, 0xe6c6c7bd, 0x327a140a,
0x45e1d006, 0xc3f27b9a, 0xc9aa53fd, 0x62a80f00, 0xbb25bfe2, 0x35bdd2f6,
0x71126905, 0xb2040222, 0xb6cbcf7c, 0xcd769c2b, 0x53113ec0, 0x1640e3d3,
0x38abbd60, 0x2547adf0, 0xba38209c, 0xf746ce76, 0x77afa1c5, 0x20756060,
0x85cbfe4e, 0x8ae88dd8, 0x7aaaf9b0, 0x4cf9aa7e, 0x1948c25c, 0x02fb8a8c,
0x01c36ae4, 0xd6ebe1f9, 0x90d4f869, 0xa65cdea0, 0x3f09252d, 0xc208e69f,
0xb74e6132, 0xce77e25b, 0x578fdfe3, 0x3ac372e6};

/* Initialize s-boxes without file read. */
        for(i=0;i<256;i++){
                c->S[0][i] = ks0[i];
                c->S[1][i] = ks1[i];
                c->S[2][i] = ks2[i];
                c->S[3][i] = ks3[i];
        }


        j = 0;
        for (i = 0; i < N + 2; ++i) {
                data = 0x00000000;
                for (k = 0; k < 4; ++k) {
```

```
                        data = (data << 8) | key[j];
                        j = j + 1;
                        if (j >= keybytes) {
                                j = 0;
                        }
                }
        c->P[i] = c->P[i] ^ data;
        }

     datal = 0x00000000;
        datar = 0x00000000;

        for (i = 0; i < N + 2; i += 2) {
                Blowfish_encipher(c,&datal, &datar);

                c->P[i] = datal;
                c->P[i + 1] = datar;
        }

        for (i = 0; i < 4; ++i) {
                for (j = 0; j < 256; j += 2) {

                        Blowfish_encipher(c,&datal, &datar);

                        c->S[i][j] = datal;
                        c->S[i][j + 1] = datar;
                }
        }
}

void blf_key(blf_ctx *c, char *k, int len){
        InitializeBlowfish(c,k,len);
}

void blf_enc(blf_ctx *c, unsigned long *data, int blocks){
        unsigned long *d;
        int i;

        d = data;
        for(i=0;i<blocks;i++){
                Blowfish_encipher(c,d,d+1);
                d += 2;
        }
}

void blf_dec(blf_ctx *c, unsigned long *data, int blocks){
        unsigned long *d;
        int i;

        d = data;
        for(i=0;i<blocks;i++){
                Blowfish_decipher(c,d,d+1);
                d += 2;
        }
}

void main(void){
        blf_ctx c;
        char key[]="AAAAA";
        unsigned long data[10];
        int i;
```

```
        for(i=0;i<10;i++) data[i] = i;

        blf_key(&c,key,5);
        blf_enc(&c,data,5);
        blf_dec(&c,data,1);
        blf_dec(&c,data+2,4);
        for(i=0;i<10;i+=2) printf("Block %01d decrypts to: %08lx %08lx.\n",
                                  i/2,data[i],data[i+1]);
}
```

3-WAY

```
#define   STRT_E   0x0b0b /* round constant of first encryption round */
#define   STRT_D   0xb1b1 /* round constant of first decryption round */
#define     NMBR       11 /* number of rounds is 11                   */

typedef   unsigned long int  word32 ;
                /* the program only works correctly if long = 32bits */
typedef unsigned long u4;
typedef unsigned char u1;

typedef struct {
        u4 k[3],ki[3], ercon[NMBR+1],drcon[NMBR+1];
} twy_ctx;

/* Note:  encrypt and decrypt expect full blocks--padding blocks is
          caller's responsibility.  All bulk encryption is done in
          ECB mode by these calls.  Other modes may be added easily
          enough.                                                   */

/* destroy:  Context. */
/* Scrub context of all sensitive data. */
void twy_destroy(twy_ctx *);

/* encrypt:  Context, ptr to data block, # of blocks. */
void twy_enc(twy_ctx *, u4 *, int);

/* decrypt:  Context, ptr to data block, # of blocks. */
void twy_dec(twy_ctx *, u4 *, int);

/* key:  Context, ptr to key data. */
void twy_key(twy_ctx *, u4 *);

/* ACCODE--------------------------------------------------------- */
/* End of AC code prototypes and structures.                      */
/* ------------------------------------------------------------- */

void mu(word32 *a)       /* inverts the order of the bits of a */
{
int i ;
word32 b[3] ;

b[0] = b[1] = b[2] = 0 ;
for( i=0 ; i<32 ; i++ )
   {
   b[0] <<= 1 ; b[1] <<= 1 ; b[2] <<= 1 ;
   if(a[0]&1) b[2] |= 1 ;
   if(a[1]&1) b[1] |= 1 ;
```

```
   if(a[2]&1) b[0] |= 1 ;
   a[0] >>= 1 ; a[1] >>= 1 ; a[2] >>= 1 ;
   }

a[0] = b[0] ;      a[1] = b[1] ;      a[2] = b[2] ;
}


void gamma(word32 *a)   /* the nonlinear step */
{
word32 b[3] ;

b[0] = a[0] ^ (a[1]|(~a[2])) ;
b[1] = a[1] ^ (a[2]|(~a[0])) ;
b[2] = a[2] ^ (a[0]|(~a[1])) ;

a[0] = b[0] ;      a[1] = b[1] ;      a[2] = b[2] ;
}


void theta(word32 *a)    /* the linear step */
{
word32 b[3];

b[0] = a[0] ^  (a[0]>>16) ^ (a[1]<<16) ^     (a[1]>>16) ^ (a[2]<<16) ^
               (a[1]>>24) ^ (a[2]<<8)  ^     (a[2]>>8)  ^ (a[0]<<24) ^
               (a[2]>>16) ^ (a[0]<<16) ^     (a[2]>>24) ^ (a[0]<<8)  ;
b[1] = a[1] ^  (a[1]>>16) ^ (a[2]<<16) ^     (a[2]>>16) ^ (a[0]<<16) ^
               (a[2]>>24) ^ (a[0]<<8)  ^     (a[0]>>8)  ^ (a[1]<<24) ^
               (a[0]>>16) ^ (a[1]<<16) ^     (a[0]>>24) ^ (a[1]<<8)  ;
b[2] = a[2] ^  (a[2]>>16) ^ (a[0]<<16) ^     (a[0]>>16) ^ (a[1]<<16) ^
               (a[0]>>24) ^ (a[1]<<8)  ^     (a[1]>>8)  ^ (a[2]<<24) ^
               (a[1]>>16) ^ (a[2]<<16) ^     (a[1]>>24) ^ (a[2]<<8)  ;

a[0] = b[0] ;      a[1] = b[1] ;      a[2] = b[2] ;
}

void pi_1(word32 *a)
{
a[0] = (a[0]>>10) ^ (a[0]<<22);
a[2] = (a[2]<<1)  ^ (a[2]>>31);
}

void pi_2(word32 *a)
{
a[0] = (a[0]<<1)  ^ (a[0]>>31);
a[2] = (a[2]>>10) ^ (a[2]<<22);
}

void rho(word32 *a)    /* the round function       */
{
theta(a) ;
pi_1(a) ;
gamma(a) ;
pi_2(a) ;
}

void rndcon_gen(word32 strt,word32 *rtab)
{                           /* generates the round constants */
int i ;

for(i=0 ; i<=NMBR ; i++ )
```

```
   {
   rtab[i] = strt ;
   strt <<= 1 ;
   if( strt&0x10000 ) strt ^= 0x11011 ;
   }
}

/* Modified slightly to fit the caller's needs. */
void encrypt(twy_ctx *c, word32 *a)
{
char i ;
for( i=0 ; i<NMBR ; i++ )
   {
   a[0] ^= c->k[0] ^ (c->ercon[i]<<16) ;
   a[1] ^= c->k[1] ;
   a[2] ^= c->k[2] ^ c->ercon[i] ;
   rho(a) ;
   }
a[0] ^= c->k[0] ^ (c->ercon[NMBR]<<16) ;
a[1] ^= c->k[1] ;
a[2] ^= c->k[2] ^ c->ercon[NMBR] ;
theta(a) ;
}

/* Modified slightly to meet caller's needs. */
void decrypt(twy_ctx *c, word32 *a)
{
char i ;

mu(a) ;
for( i=0 ; i<NMBR ; i++ )
   {
   a[0] ^= c->ki[0] ^ (c->drcon[i]<<16) ;
   a[1] ^= c->ki[1] ;
   a[2] ^= c->ki[2] ^ c->drcon[i] ;
   rho(a) ;
   }
a[0] ^= c->ki[0] ^ (c->drcon[NMBR]<<16) ;
a[1] ^= c->ki[1] ;
a[2] ^= c->ki[2] ^ c->drcon[NMBR] ;
theta(a) ;
mu(a) ;
}

void twy_key(twy_ctx *c, u4 *key){
        c->ki[0] = c->k[0] = key[0];
        c->ki[1] = c->k[1] = key[1];
        c->ki[2] = c->k[2] = key[2];
        theta(c->ki);
        mu(c->ki);
        rndcon_gen(STRT_E,c->ercon);
        rndcon_gen(STRT_D,c->drcon);

}

/* Encrypt in ECB mode. */
void twy_enc(twy_ctx *c, u4 *data, int blkcnt){
        u4 *d;
        int i;

        d = data;
        for(i=0;i<blkcnt;i++) {
```

```
                encrypt(c,d);
                d +=3;
        }
}


/* Decrypt in ECB mode. */
void twy_dec(twy_ctx *c, u4 *data, int blkcnt){
        u4 *d;
        int i;

        d = data;
        for(i=0;i<blkcnt;i++){
                decrypt(c,d);
                d+=3;
        }
}


/* Scrub sensitive values from memory before deallocating. */
void twy_destroy(twy_ctx *c){
        int i;

        for(i=0;i<3;i++) c->k[i] = c->ki[i] = 0;
}

void printvec(char *chrs, word32 *d){
        printf("%20s : %08lx %08lx %08lx \n",chrs,d[2],d[1],d[0]);
}


main()
{
twy_ctx gc;
word32 a[9],k[3];
int i;

/* Test vector 1. */

k[0]=k[1]=k[2]=0;
a[0]=a[1]=a[2]=1;
twy_key(&gc,k);

printf("**********\n");
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);


/* Test vector 2. */

k[0]=6;k[1]=5;k[2]=4;
a[0]=3;a[1]=2;a[2]=1;
twy_key(&gc,k);

printf("**********\n");
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);
```

```
/* Test vector 3. */

k[2]=0xbcdef012;k[1]=0x456789ab;k[0]=0xdef01234;
a[2]=0x01234567;a[1]=0x9abcdef0;a[0]=0x23456789;
twy_key(&gc,k);

printf("**********\n");
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);


/* Test vector 4. */

k[2]=0xcab920cd;k[1]=0xd6144138;k[0]=0xd2f05b5e;
a[2]=0xad21ecf7;a[1]=0x83ae9dc4;a[0]=0x4059c76e;
twy_key(&gc,k);

printf("**********\n");
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);

/*  TEST VALUES

key        : 00000000 00000000 00000000
plaintext  : 00000001 00000001 00000001
ciphertext : ad21ecf7 83ae9dc4 4059c76e

key        : 00000004 00000005 00000006
plaintext  : 00000001 00000002 00000003
ciphertext : cab920cd d6144138 d2f05b5e

key        : bcdef012 456789ab def01234
plaintext  : 01234567 9abcdef0 23456789
ciphertext : 7cdb76b2 9cdddb6d 0aa55dbb

key        : cab920cd d6144138 d2f05b5e
plaintext  : ad21ecf7 83ae9dc4 4059c76e
ciphertext : 15b155ed 6b13f17c 478ea871

*/

/* Enc/dec test: */
for(i=0;i<9;i++) a[i]=i;
twy_enc(&gc,a,3);
for(i=0;i<9;i+=3) printf("Block %01d encrypts to %08lx %08lx %08lx\n",
                         i/3,a[i],a[i+1],a[i+2]);

twy_dec(&gc,a,2);
twy_dec(&gc,a+6,1);

 for(i=0;i<9;i+=3) printf("Block %01d decrypts to %08lx %08lx %08lx\n",
                         i/3,a[i],a[i+1],a[i+2]);
}
```

RC5

```
#include <stdio.h>

/* An RC5 context needs to know how many rounds it has, and its subkeys. */
typedef struct {
        u4 *xk;
        int nr;
} rc5_ctx;

/* Where possible, these should be replaced with actual rotate instructions.
   For Turbo C++, this is done with _lrotl and _lrotr. */

#define ROTL32(X,C) (((X)<<(C))|((X)>>(32-(C))))
#define ROTR32(X,C) (((X)>>(C))|((X)<<(32-(C))))

/* Function prototypes for dealing with RC5 basic operations. */
void rc5_init(rc5_ctx *, int);
void rc5_destroy(rc5_ctx *);
void rc5_key(rc5_ctx *, u1 *, int);
void rc5_encrypt(rc5_ctx *, u4 *, int);
void rc5_decrypt(rc5_ctx *, u4 *, int);


/* Function implementations for RC5. */

/* Scrub out all sensitive values. */
void rc5_destroy(rc5_ctx *c){
        int i;
     for(i=0;i<(c->nr)*2+2;i++) c->xk[i]=0;
     free(c->xk);
}

/* Allocate memory for rc5 context's xk and such. */
void rc5_init(rc5_ctx *c, int rounds){
     c->nr = rounds;
     c->xk = (u4 *) malloc(4*(rounds*2+2));
}

void rc5_encrypt(rc5_ctx *c, u4 *data, int blocks){
        u4 *d,*sk;
        int h,i,rc;

     d = data;
        sk = (c->xk)+2;
        for(h=0;h<blocks;h++){
                d[0] += c->xk[0];
                d[1] += c->xk[1];
                for(i=0;i<c->nr*2;i+=2){
                        d[0] ^= d[1];
                        rc = d[1] & 31;
                        d[0] = ROTL32(d[0],rc);
                        d[0] += sk[i];
               d[1] ^= d[0];
                        rc = d[0] & 31;
                        d[1] = ROTL32(d[1],rc);
                        d[1] += sk[i+1];
/*printf("Round %03d : %08lx %08lx  sk= %08lx %08lx\n",i/2,
                                d[0],d[1],sk[i],sk[i+1]);*/
                }
          d+=2;
        }
```

```
}

void rc5_decrypt(rc5_ctx *c, u4 *data, int blocks){
     u4 *d,*sk;
        int h,i,rc;

     d = data;
        sk = (c->xk)+2;
     for(h=0;h<blocks;h++){
                 for(i=c->nr*2-2;i>=0;i-=2){
/*printf("Round %03d: %08lx %08lx  sk: %08lx %08lx\n",
        i/2,d[0],d[1],sk[i],sk[i+1]); */
                         d[1] -= sk[i+1];
                         rc = d[0] & 31;
                         d[1] = ROTR32(d[1],rc);
                         d[1] ^= d[0];

                         d[0] -= sk[i];
                         rc = d[1] & 31;
                         d[0] = ROTR32(d[0],rc);
                d[0] ^= d[1];
                 }
                 d[0] -= c->xk[0];
                 d[1] -= c->xk[1];
        d+=2;
     }
}

void rc5_key(rc5_ctx *c, u1 *key, int keylen){
     u4 *pk,A,B; /* padded key */
     int xk_len, pk_len, i, num_steps,rc;
     u1 *cp;

     xk_len = c->nr*2 + 2;
     pk_len = keylen/4;
     if((keylen%4)!=0) pk_len += 1;

     pk = (u4 *) malloc(pk_len * 4);
     if(pk==NULL) {
          printf("An error occurred!\n");
          exit(-1);
     }

     /* Initialize pk -- this should work on Intel machines, anyway.... */
     for(i=0;i<pk_len;i++) pk[i]=0;
     cp = (u1 *)pk;
     for(i=0;i<keylen;i++) cp[i]=key[i];

     /* Initialize xk. */
     c->xk[0] = 0xb7e15163; /* P32 */
     for(i=1;i<xk_len;i++) c->xk[i] = c->xk[i-1] + 0x9e3779b9; /* Q32 */

     /* TESTING */
     A = B = 0;
     for(i=0;i<xk_len;i++) {
          A = A + c->xk[i];
          B = B ^ c->xk[i];
     }

     /* Expand key into xk. */
     if(pk_len>xk_len) num_steps = 3*pk_len;else num_steps = 3*xk_len;
```

```
     A = B = 0;
     for(i=0;i<num_steps;i++){
          A = c->xk[i%xk_len] = ROTL32(c->xk[i%xk_len] + A + B,3);
          rc = (A+B) & 31;
          B = pk[i%pk_len] = ROTL32(pk[i%pk_len] + A + B,rc);

     }

     /* Clobber sensitive data before deallocating memory. */
     for(i=0;i<pk_len;i++) pk[i] =0;

     free(pk);
}

void main(void){
     rc5_ctx c;
     u4 data[8];
     char key[] = "ABCDE";
     int i;

     printf("-------------------------------------------------\n");

        for(i=0;i<8;i++) data[i] = i;
     rc5_init(&c,10); /* 10 rounds */
     rc5_key(&c,key,5);

        rc5_encrypt(&c,data,4);
        printf("Encryptions:\n");
        for(i=0;i<8;i+=2) printf("Block %01d = %08lx %08lx\n",
                                 i/2,data[i],data[i+1]);
        rc5_decrypt(&c,data,2);
     rc5_decrypt(&c,data+4,2);
        printf("Decryptions:\n");
        for(i=0;i<8;i+=2) printf("Block %01d = %08lx %08lx\n",
                                 i/2,data[i],data[i+1]);

}
```

A5

```
typedef struct {
        unsigned long r1,r2,r3;
} a5_ctx;

static int threshold(r1, r2, r3)
unsigned int r1;
unsigned int r2;
unsigned int r3;
{
int total;

  total = (((r1 >>  9) & 0x1) == 1) +
          (((r2 >> 11) & 0x1) == 1) +
          (((r3 >> 11) & 0x1) == 1);

  if (total > 1)
    return (0);
  else
```

```
    return (1);
}

unsigned long clock_r1(ctl, r1)
int ctl;
unsigned long r1;
{
unsigned long feedback;

  ctl ^= ((r1 >> 9) & 0x1);
  if (ctl)
  {
    feedback = (r1 >> 18) ^ (r1 >> 17) ^ (r1 >> 16) ^ (r1 >> 13);
    r1 = (r1 << 1) & 0x7ffff;
    if (feedback & 0x01)
      r1 ^= 0x01;
  }
  return (r1);
}

unsigned long clock_r2(ctl, r2)
int ctl;
unsigned long r2;
{
unsigned long feedback;

  ctl ^= ((r2 >> 11) & 0x1);
  if (ctl)
  {
    feedback = (r2 >> 21) ^ (r2 >> 20) ^ (r2 >> 16) ^ (r2 >> 12);
    r2 = (r2 << 1) & 0x3fffff;
    if (feedback & 0x01)
      r2 ^= 0x01;
  }
  return (r2);
}

unsigned long clock_r3(ctl, r3)
int ctl;
unsigned long r3;
{
unsigned long feedback;

  ctl ^= ((r3 >> 11) & 0x1);
  if (ctl)
  {
    feedback = (r3 >> 22) ^ (r3 >> 21) ^ (r3 >> 18) ^ (r3 >> 17);
    r3 = (r3 << 1) & 0x7fffff;
    if (feedback & 0x01)
      r3 ^= 0x01;
  }
  return (r3);
}

int keystream(key, frame, alice, bob)
unsigned char *key;   /* 64 bit session key              */
unsigned long frame;  /* 22 bit frame sequence number    */
unsigned char *alice; /* 114 bit Alice to Bob key stream */
unsigned char *bob;   /* 114 bit Bob to Alice key stream */
{
unsigned long r1;   /* 19 bit shift register */
unsigned long r2;   /* 22 bit shift register */
```

```
unsigned long r3;   /* 23 bit shift register */
int i;              /* counter for loops     */
int clock_ctl;      /* xored with clock enable on each shift register */
unsigned char *ptr; /* current position in keystream */
unsigned char byte; /* byte of keystream being assembled */
unsigned int bits;  /* number of bits of keystream in byte */
unsigned int bit;   /* bit output from keystream generator */

  /* Initialise shift registers from session key */

  r1 = (key[0] | (key[1] << 8) | (key[2] << 16) ) & 0x7ffff;
  r2 = ((key[2] >> 3) | (key[3] << 5) | (key[4] << 13) | (key[5] << 21)) &
0x3fffff;
  r3 = ((key[5] >> 1) | (key[6] << 7) | (key[7] << 15) ) & 0x7fffff;


  /* Merge frame sequence number into shift register state, by xor'ing it
   * into the feedback path
   */

  for (i=0;i<22;i++)
  {
    clock_ctl = threshold(r1, r2, r2);
    r1 = clock_r1(clock_ctl, r1);
    r2 = clock_r2(clock_ctl, r2);
    r3 = clock_r3(clock_ctl, r3);
    if (frame & 1)
    {
      r1 ^= 1;
      r2 ^= 1;
      r3 ^= 1;
    }
    frame = frame >> 1;
  }

  /* Run shift registers for 100 clock ticks to allow frame number to
   * be diffused into all the bits of the shift registers
   */

  for (i=0;i<100;i++)
  {
    clock_ctl = threshold(r1, r2, r2);
    r1 = clock_r1(clock_ctl, r1);
    r2 = clock_r2(clock_ctl, r2);
    r3 = clock_r3(clock_ctl, r3);
  }

  /* Produce 114 bits of Alice->Bob key stream */

  ptr = alice;
  bits = 0;
  byte = 0;
  for (i=0;i<114;i++)
  {
    clock_ctl = threshold(r1, r2, r2);
    r1 = clock_r1(clock_ctl, r1);
    r2 = clock_r2(clock_ctl, r2);
    r3 = clock_r3(clock_ctl, r3);

    bit = ((r1 >> 18) ^ (r2 >> 21) ^ (r3 >> 22)) & 0x01;
    byte = (byte << 1) | bit;
    bits++;
```

```
    if (bits == 8)
    {
      *ptr = byte;
      ptr++;
      bits = 0;
      byte = 0;
    }
  }
  if (bits)
    *ptr = byte;

  /* Run shift registers for another 100 bits to hide relationship between
   * Alice->Bob key stream and Bob->Alice key stream.
   */

  for (i=0;i<100;i++)
  {
    clock_ctl = threshold(r1, r2, r2);
    r1 = clock_r1(clock_ctl, r1);
    r2 = clock_r2(clock_ctl, r2);
    r3 = clock_r3(clock_ctl, r3);
  }

  /* Produce 114 bits of Bob->Alice key stream */

  ptr = bob;
  bits = 0;
  byte = 0;
  for (i=0;i<114;i++)
  {
    clock_ctl = threshold(r1, r2, r2);
    r1 = clock_r1(clock_ctl, r1);
    r2 = clock_r2(clock_ctl, r2);
    r3 = clock_r3(clock_ctl, r3);

    bit = ((r1 >> 18) ^ (r2 >> 21) ^ (r3 >> 22)) & 0x01;
    byte = (byte << 1) | bit;
    bits++;
    if (bits == 8)
    {
      *ptr = byte;
      ptr++;
      bits = 0;
      byte = 0;
    }
  }
  if (bits)
    *ptr = byte;

  return (0);

}

void a5_key(a5_ctx *c, char *k){
        c->r1 = k[0]<<11|k[1]<<3 | k[2]>>5          ; /* 19 */
        c->r2 = k[2]<<17|k[3]<<9 | k[4]<<1 | k[5]>>7; /* 22 */
        c->r3 = k[5]<<15|k[6]<<8 | k[7]             ; /* 23 */
}

/* Step one bit in A5, return 0 or 1 as output bit. */
int a5_step(a5_ctx *c){
        int control;
```

```
        control = threshold(c->r1,c->r2,c->r3);
        c->r1 = clock_r1(control,c->r1);
        c->r2 = clock_r2(control,c->r2);
        c->r3 = clock_r3(control,c->r3);
        return( (c->r1^c->r2^c->r3)&1);
}

/* Encrypts a buffer of len bytes. */
void a5_encrypt(a5_ctx *c, char *data, int len){
        int i,j;
        char t;

        for(i=0;i<len;i++){
                for(j=0;j<8;j++) t = t<<1 | a5_step(c);
                data[i]^=t;
        }
}

void a5_decrypt(a5_ctx *c, char *data, int len){
        a5_encrypt(c,data,len);
}

void main(void){
        a5_ctx c;
        char data[100];
        char key[] = {1,2,3,4,5,6,7,8};
        int i,flag;

        for(i=0;i<100;i++) data[i] = i;

        a5_key(&c,key);
        a5_encrypt(&c,data,100);

        a5_key(&c,key);
        a5_decrypt(&c,data,1);
        a5_decrypt(&c,data+1,99);

        flag = 0;
        for(i=0;i<100;i++) if(data[i]!=i)flag = 1;
        if(flag)printf("Decrypt failed\n"); else printf("Decrypt
succeeded\n");
}
```

SEAL

```
#undef SEAL_DEBUG

#define ALG_OK 0
#define ALG_NOTOK 1
#define WORDS_PER_SEAL_CALL 1024

typedef struct {
     unsigned long t[520]; /* 512 rounded up to a multiple of 5 + 5*/
     unsigned long s[265]; /* 256 rounded up to a multiple of 5 + 5*/
     unsigned long r[20];  /* 16 rounded up to multiple of 5 */
        unsigned long counter; /* 32-bit synch value. */
        unsigned long ks_buf[WORDS_PER_SEAL_CALL];
        int ks_pos;
} seal_ctx;
```

```
#define ROT2(x) (((x) >> 2) | ((x) << 30))
#define ROT9(x) (((x) >> 9) | ((x) << 23))
#define ROT8(x) (((x) >> 8) | ((x) << 24))
#define ROT16(x) (((x) >> 16) | ((x) << 16))
#define ROT24(x) (((x) >> 24) | ((x) << 8))
#define ROT27(x) (((x) >> 27) | ((x) << 5))

#define WORD(cp)  ((cp[0] << 24)|(cp[1] << 16)|(cp[2] << 8)|(cp[3]))

#define F1(x, y, z) (((x) & (y)) | ((~(x)) & (z)))
#define F2(x, y, z) ((x)^(y)^(z))
#define F3(x, y, z) (((x) & (y)) | ((x) & (z)) | ((y) & (z)))
#define F4(x, y, z) ((x)^(y)^(z))

int g(in, i, h)
unsigned char *in;
int i;
unsigned long *h;
{
unsigned long h0;
unsigned long h1;
unsigned long h2;
unsigned long h3;
unsigned long h4;
unsigned long a;
unsigned long b;
unsigned long c;
unsigned long d;
unsigned long e;
unsigned char *kp;
unsigned long w[80];
unsigned long temp;

     kp = in;
     h0 = WORD(kp); kp += 4;
     h1 = WORD(kp); kp += 4;
     h2 = WORD(kp); kp += 4;
     h3 = WORD(kp); kp += 4;
     h4 = WORD(kp); kp += 4;

     w[0] = i;
     for (i=1;i<16;i++)
          w[i] = 0;
     for (i=16;i<80;i++)
          w[i] = w[i-3]^w[i-8]^w[i-14]^w[i-16];

     a = h0;
     b = h1;
     c = h2;
     d = h3;
     e = h4;

     for (i=0;i<20;i++)
     {
          temp = ROT27(a) + F1(b, c, d) + e + w[i] + 0x5a827999;
          e = d;
          d = c;
          c = ROT2(b);
          b = a;
          a = temp;
     }
```

```
	for (i=20;i<40;i++)
	{
		temp = ROT27(a) + F2(b, c, d) + e + w[i] + 0x6ed9eba1;
		e = d;
		d = c;
		c = ROT2(b);
		b = a;
		a = temp;
	}
	for (i=40;i<60;i++)
	{
		temp = ROT27(a) + F3(b, c, d) + e + w[i] + 0x8f1bbcdc;
		e = d;
		d = c;
		c = ROT2(b);
		b = a;
		a = temp;
	}
	for (i=60;i<80;i++)
	{
		temp = ROT27(a) + F4(b, c, d) + e + w[i] + 0xca62c1d6;
		e = d;
		d = c;
		c = ROT2(b);
		b = a;
		a = temp;
	}
	h[0] = h0+a;
	h[1] = h1+b;
	h[2] = h2+c;
	h[3] = h3+d;
	h[4] = h4+e;

	return (ALG_OK);
}

unsigned long gamma(a, i)
unsigned char *a;
int i;
{
unsigned long h[5];

	(void) g(a, i/5, h);
	return h[i % 5];
}

int  seal_init(seal_ctx *result, unsigned char *key)
{
int i;
unsigned long h[5];

	for (i=0;i<510;i+=5)
		g(key, i/5, &(result->t[i]));
	/* horrible special case for the end */
	g(key, 510/5, h);
	for (i=510;i<512;i++)
		result->t[i] = h[i-510];
	/* 0x1000 mod 5 is +1, so have horrible special case for the start */
	g(key, (-1+0x1000)/5, h);
	for (i=0;i<4;i++)
		result->s[i] = h[i+1];
	for (i=4;i<254;i+=5)
```

```
          g(key, (i+0x1000)/5, &(result->s[i]));
     /* horrible special case for the end */
     g(key, (254+0x1000)/5, h);
     for (i=254;i<256;i++)
          result->s[i] = h[i-254];
     /* 0x2000 mod 5 is +2, so have horrible special case at the start */
     g(key, (-2+0x2000)/5, h);
     for (i=0;i<3;i++)
          result->r[i] = h[i+2];
     for (i=3;i<13;i+=5)
          g(key, (i+0x2000)/5, &(result->r[i]));
     /* horrible special case for the end */
     g(key, (13+0x2000)/5, h);
     for (i=13;i<16;i++)
          result->r[i] = h[i-13];
     return (ALG_OK);
}

int seal(seal_ctx *key, unsigned long in, unsigned long *out)
{
int i;
int j;
int l;
unsigned long a;
unsigned long b;
unsigned long c;
unsigned long d;
unsigned short p;
unsigned short q;
unsigned long n1;
unsigned long n2;
unsigned long n3;
unsigned long n4;
unsigned long *wp;

     wp = out;

     for (l=0;l<4;l++)
     {
          a = in ^ key->r[4*l];
          b = ROT8(in) ^ key->r[4*l+1];
          c = ROT16(in) ^ key->r[4*l+2];
          d = ROT24(in) ^ key->r[4*l+3];

          for (j=0;j<2;j++)
          {
               p = a & 0x7fc;
               b += key->t[p/4];
               a = ROT9(a);

               p = b & 0x7fc;
               c += key->t[p/4];
               b = ROT9(b);

               p = c & 0x7fc;
               d += key->t[p/4];
               c = ROT9(c);

               p = d & 0x7fc;
               a += key->t[p/4];
               d = ROT9(d);
```

```
  }
  n1 = d;
  n2 = b;
  n3 = a;
  n4 = c;

  p = a & 0x7fc;
  b += key->t[p/4];
  a = ROT9(a);

  p = b & 0x7fc;
  c += key->t[p/4];
  b = ROT9(b);

  p = c & 0x7fc;
  d += key->t[p/4];
  c = ROT9(c);

  p = d & 0x7fc;
  a += key->t[p/4];
  d = ROT9(d);

/* This generates 64 32-bit words, or 256 bytes of keystream. */
  for (i=0;i<64;i++)
  {
       p = a & 0x7fc;
       b += key->t[p/4];
       a = ROT9(a);
       b ^= a;

       q = b & 0x7fc;
       c ^= key->t[q/4];
       b = ROT9(b);
       c += b;

       p = (p+c) & 0x7fc;
       d += key->t[p/4];
       c = ROT9(c);
       d ^= c;

       q = (q+d) & 0x7fc;
       a ^= key->t[q/4];
       d = ROT9(d);
       a += d;

       p = (p+a) & 0x7fc;
       b ^= key->t[p/4];
       a = ROT9(a);

       q = (q+b) & 0x7fc;
       c += key->t[q/4];
       b = ROT9(b);

       p = (p+c) & 0x7fc;
       d ^= key->t[p/4];
       c = ROT9(c);

       q = (q+d) & 0x7fc;
       a += key->t[q/4];
       d = ROT9(d);

       *wp = b + key->s[4*i];
```

```
                wp++;
                *wp = c ^ key->s[4*i+1];
                wp++;
                *wp = d + key->s[4*i+2];
                wp++;
                *wp = a ^ key->s[4*i+3];
                wp++;

                if (i & 1)
                {
                     a += n3;
                     c += n4;
                }
                else
                {
                     a += n1;
                     c += n2;
                }

          }
     }
     return (ALG_OK);
}

/* Added call to refill ks_buf and reset counter and ks_pos. */
void seal_refill_buffer(seal_ctx *c){
        seal(c,c->counter,c->ks_buf);
        c->counter++;
        c->ks_pos = 0;
}

void seal_key(seal_ctx *c, unsigned char *key){
        seal_init(c,key);
        c->counter = 0;  /* By default, init to zero. */
        c->ks_pos = WORDS_PER_SEAL_CALL;
                /* Refill keystream buffer on next call. */
}

/* This encrypts the next w words with SEAL. */
void seal_encrypt(seal_ctx *c, unsigned long *data_ptr, int w){
        int i;

        for(i=0;i<w;i++){
                if(c->ks_pos>=WORDS_PER_SEAL_CALL) seal_refill_buffer(c);
                data_ptr[i]^=c->ks_buf[c->ks_pos];
                c->ks_pos++;
        }
}

void seal_decrypt(seal_ctx *c, unsigned long *data_ptr, int w) {
        seal_encrypt(c,data_ptr,w);
}

void seal_resynch(seal_ctx *c, unsigned long synch_word){
        c->counter = synch_word;
        c->ks_pos = WORDS_PER_SEAL_CALL;
}


void main(void){
        seal_ctx sc;
        unsigned long buf[1000],t;
```

```
        int i,flag;
        unsigned char key[] =
                {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};

        printf("1\n");
        seal_key(&sc,key);

        printf("2\n");
        for(i=0;i<1000;i++) buf[i]=0;
        printf("3\n");
        seal_encrypt(&sc,buf,1000);
        printf("4\n");
        t = 0;
        for(i=0;i<1000;i++) t = t ^ buf[i];
                printf("XOR of buf is %08lx.\n",t);

        seal_key(&sc,key);
        seal_decrypt(&sc,buf,1);
        seal_decrypt(&sc,buf+1,999);
        flag = 0;
        for(i=0;i<1000;i++) if(buf[i]!=0)flag=1;
        if(flag) printf("Decrypt failed.\n");
        else printf("Decrypt succeeded.\n");


}
```

References

1. ABA Bank Card Standard, "Management and Use of Personal Information Numbers, " Aids from ABA, Catalog no. 207213, American Bankers Association, 1979.

2. ABA Document 4.3, "Key Management Standard," American Bankers Association, 1980.

3. M. Abadi, J. Feigenbaum, and J. Kilian, "On Hiding Information from an Oracle," Proceedings of the 19th ACM Symposium on the Theory of Computing, 1987, pp. 195-203.

4. M. Abadi, J. Feigenbaum, and J. Kilian, "On Hiding Information from an Oracle," Journal of Computer and System Sciences, v.39, n.1, Aug 1989, pp.21-50.

5. M. Abadi and R. Needham, "Prudent Engineering Practice for Cryptographic Protocols," Research Report 125, Digital Equipment Corp Systems Research Center, Jun 1994.

6. C.M. Adams, "On Immunity Against Biham and Shamir's Differential Cryptanalysis,' " Information Processing Letters, v. 41, 14 Fob 1992, pp. 77-80.

7. C.M. Adams, "Simple and Effective Key Scheduling for Symmetric Ciphers, " Workshop on Selected Areas in Cryptography Workshop Record, Kingston, Ontario, 5-6 May 1994, pp.129-133.

8. C.M. Adams and H. Mailer, "Security Related Comments Regarding McEliece's Public-Key Cryptosystem, " Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 224-230.

9. C.M. Adams and S.E. Tavares, "The Structured Design of Cryptographically Good SBoxes," journal of Cryptology v. 3, n. 1, 1990, pp. 27-41.

10. C.M. Adams and S.E. Tavares, "Designing S-Boxes for Ciphers Resistant to Differential Cryptanalysis," Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography Rome, Italy, 15-16 Feh 1993, pp. 181-190.

11. W. Adams and D. Shanks, "Strong Primality Tests That Are Not Sufficient, " Mathematics of Computation, v. 39, 1982, pp. 255-300.

12. W.W Adams and L.J. Goldstein, Introduction to Number Theory, Englewood Cliffs, N.J.: Prentice-Hall, 1976.

13. B.S. Adiga and P. Shankar, "Modified LuLee Cryptosystem," Electronics Letters, v 21, n. 18, 29 Aug 1985, pp. 794-795.

14. L.M. Adleman, "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," Proceedings of the IEEE 20th Annual Symposium of Foundations of Computer Science, 1979, pp.55-60.

15. L.M. Adleman, "On Breaking Generalized Knapsack Public Key Cryptosystems, " Proceedings of the 15th ACM Symposium on Theory of Computing, 1983, pp. 402412.

16. L.M. Adleman, "Factoring Numbers Using Singular Integers," Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing, 1991, pp. 64 71.

17. L.M. Adleman, "Molecular Computation of Solutions to Combinatorial Problems," Science, v. 266, n. 11, Nov 1994, p. 1021.

18. L.M. Adleman, D. Estes, and K. McCurley, "Solving Bivariate Quadratic Congruences in Random Polynomial Time," Mathematics of Computation, v. 48, n. 177, Jan 1987, pp. 17-28.

19. L.M. Adleman, C. Pomerance, and R.S. Rumeley, "On Distinguishing Prime Numbers from Composite Numbers, " Annals of Mathematics, v. 117, n. 1, 1983, pp. 173-206.

20. L.M. Adleman and R.L. Rivest, "How to Break the Lu-Lee {COMSAT) Public-Key Cryptosystem, " MIT Laboratory for Computer Science, Jul 1979.

21. G.B. Agnew, "Random Sources for Cryptographic Systems, " Advances in Cryptology EUROCRYPT '8 7 Proceedings, Springer-Verlag, 1988, pp. 77-81.

22. G.B. Agnew, R.C. Mullin, I.M. Onyszchuk, and S.A. Vanstone, "An Implementation for a Fast Public-Key Cryptosystem," Journal of Cryptology, v. 3, n. 2, 1991, pp. 63-79.

23. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "A Fast Elliptic Curve Cryptosystem," Advances in Cryptology EUROCRYPT '89 Proceedings, Spnnger-Verlag, 1990, pp. 706-708.

24. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "Improved Digital Signature Scheme Based on Discrete Exponentiation, " Electronics Letters, v. 26, n. 14, 5 Jul 1990, pp. 1024 1025.

25. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "On the Development of a Fast Elliptic Curve Cryptosystem," Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 482

26. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "An Implementation of Elliptic Curve Cryptosystems over F:155," IEEE Selected Areas of Communications, v. 11, n. 5, Jun 1993, pp. 804-813.

27. A. Aho, J. Hopcroft, and J. Ullman. The 40. Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

28. S.G. Akl, "Digital Signatures: A Tutorial Survey." Computer, v. 16, n. 2, Feb 1983, pp. 15-24.

29. S.G. Akl, "On the Security of Compressed Encodings," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 209-230.

30. S.G. Akl and H. Meijer, "A Fast Pseudo-Random Permutation Generator with Applications to Cryptology," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 269-275.

31. M. Alabbadi and S.B. Wicker, "Security of Xinmei Digital Signature Scheme," Electronics Letters, v. 28, n. 9, 23 Apr 1992, pp. 890-89 1.

32. M. Alabbadi and S.B. Wicker, "Digital Signature Schemes Based on Error-Correcting Codes," Proceedings of the 1993 IEEE-ISIT, IEEE Press, 1993, p. 199.

33. M. Alabbadi and S.B. Wicker, "Cryptanalysis of the Harn and Wang Modification of the Xinmei Digital Signature Scheme, " Electronics Letters, v. 28, n. 18, 27 Aug 1992, pp. 1756-1758.

34. K. Alagappan and J. Tardo, "SPX Guide: Prototype Public Key Authentication Service, " Digital Equipment Corp.. May 1991.

35. W. Alexi, B.-Z. Chor, O. Goldreich, and C.R Schnorr, "RSA and Rabin Functions: Certain Parts Are as Hard as the Whole," Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science, 1984, pp. 449-457.

36. W. Alexi, B.-Z. Chor, O. Goldreich, and C.R Schnorr, "RSA and Rabin Functions: Certain Parts are as Hard as the Whole," SIAM 1ournal on Computing, v. 17, n. 2, Apr 1988, pp. 194 209.

37. Ameritech Mobile Communications et al., "Cellular Digital Packet Data System Specifications: Part 406: Airlink Security," CDPD Industry Input Coordinator. Costa Mesa, Calif.. Jul 1993.

38. H.R. Amirazizi, E.D. Karnin, and J.M. Reyneri, "Compact Knapsacks are Polynomial Solvable," ACM SIGACT News, v.15, 1983, pp. 20-22.

39. R.J. Anderson, "Solving a Class of Stream Ciphers," Cryptologia, v. 14, n. 3, Jul 1990, pp. 285-288.

40. R.J. Anderson, "A Second Generation Electronic Wallet," ESORICS 92, Proceedings of the Second European Symposium on Research in Computer Security, Springer 54. Verlag, 1992, pp. 411 418.

41. R.J. Anderson, "Faster Attack on Certain Stream Ciphers, " Electronics Letters, v. 29, n. 15, 22 Jul 1993, pp. 1322-1323.

42. R.J. Anderson! "Derived Sequence Attacks on Stream Ciphers, " presented at the rump session of CRYPTO '93, Aug 1993.

43. R.J. Anderson, "Why Cryptosystems Fail," lst ACM Conference on Computer and Communications Security ACM Press, 1993, pp. 215-227.

44. R.J. Anderson, "Why Cryptosystems Fail," Communications of the ACM, v. 37, n. 11, Nov 1994, pp. 32 40.

45. R.J. Anderson, "On Fibonacci Keystream 58. Generators, " K. U. Lezzven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

46. R.J. Anderson, "Searching for the Optimum Correlation Attack, " K. U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995. to appear.

47. R.J. Anderson and T.M.A. Lomas, "Fortifying Key Negotiation Schemes with Poorly Chosen Passwords," Electronics Letters, v. 30, n. 13, 23 Jun 1994, pp. 1040-1041.

48. R.J. Anderson and R. Needham, "Robustness Principles for Public Key Protocols," Advances in Cryptology CRYPTO '95 Proceedings, Springer-Verlag, 1995, to appear,

49. D. Andleman and J. Reeds, "On the Cryptanalysis of Rotor Machines and Substitution-Permutation Networks," IEEE Trans actions on Information Theory, v. IT-28, n. 4, Jul 1982, pp. 578-584.

50. ANSI X3.92, "American National Standard for Data Encryption Algorithm (DEA)," Ameriean National Standards Institute, 1981.

51. ANSI X3.105, "American National Standard for Information Systems Data Link Encryption, " Ameriean National Standards Institute, 1983.

52. ANSI X3.106, "American National Standard for Information Systems Data Encryption Algorithm Modes of Operation," Ameriean National Standards Institute, 1 983.

53. ANSI X9.8, "American National Standard for Personal Information Number (PIN) Management and Security, " American Bankers Association, 1982.

54. ANSI X9.9 (Revised, "American National Standard for Financial Institution Message Authentication (Wholesales), " American Bankers Asso ciation, 1986.

55. ANSI X9.17 (Revised. "American National Standard for Financial Institution Key Management (Wholesales)" American Bankers Assoeiation, 1985.

56. ANSI X9.19, "American National Standard for Retail Message Authentication," Ameriean Bankers Assoeiation, 1985.

57. ANSI X9.23, "American National Standard for Financial Institution Message Encryption, " American Bankers Assoeiation, 1988.

58. ANSI X9.24, "Draft Proposed Ameriean National Standard for Retail Key Management," Ameriean Bankers Assoeiation, 1988.

59. ANSI X9.26 (Revised). "American National Standard for Financial Institution Sign-On Authentication for Wholesale Financial Transaction," American Bankers Association, 1990.

60. ANSI X9.30, "Working Draft: Public Key Cryptography Using irreversible Algorithms for the Financial Services Industry'" Ameriean Bankers Association, Aug 1994.

61. ANSI X9.31, "Working Draft: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry," Ameriean Bankers Association, Mar 1993.

62. K. Aoki and K. Ohta, "Differential-Linear Cryptanalysis of FEAL-8," Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS by), Innyama, Japan, 24-27 Jan 1995, pp. A3.4.1-11. (In Japanese)

63. K. Araki and T. Sekine, "On the Conspiracy Problem of the Generalized Tanaka's Cryptosystem," IEICE Transactions, v. E74, n. 8, Aug 1991, pp. 2176-2178.

64. S. Araki, K. Aoki, and K. Ohta, "The Best Linear Expression Search for FEAL," Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95), Inuyama, Japan, 24-27 Jan 1995, pp. A4.4.1-10.

65. C. Asmuth and J. Bloom, "A Modular Approach to Key Safeguarding," IE EE Transactions on Information Theory, v. IT- 29, n. 2, Mar 1983, pp. 208 210.

66. D. Atkins, M. Graff, A.K. Lenstra, and RC. Leyland, "The Magic Words are Squeamish Ossifrage, " Advances in Cryptology ASIA CRYPT '94 Proceedings, Springer- Verlag, 1995, pp. 263-277.

67. AT&T, "T7001 Random Number Generator," Data Sheet, Aug 1986.

68. AT&LT, "ATTEST Readying New Spy-Proof Phone for Big Military and Civilian Markets," The Report on ATTEST, 2 Jun 1986, pp. 6-7.

69. AT&T, "T7002/T7003 Bit Slice Multiplier," product announcement, 1987.

70. AT&T, "Telephone Security Device TSD 3600 User's Manual, " ATTEST, 20 Sep 1992.

71. Y. Aumann and U. Feige, "On Message Proof Systems with Known Space Verifiers," Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 85-99.

72. R.G. Ayoub, An Introduction to the Theory of Numbers, Providence, Rl: American Mathematical Society, 1963.

73. A. Aziz and W. Diffie, "Privacy and Authentication for Wireless Local Area Networks," IEEE Personal Communications, v. l, n. 1, 1994, pp. 25-31.

74. A. Bahreman and J.D. Tygar, "Certified Electronic Mail," Proceedings of the Internet Society 1994 Workshop on Network and Distributed System Secunty, The Internet Society, 1994, pp. 3-19.

75. D. Balenson, "Automated Distribution of Cryptographic Keys Using the Financial Institution Key Management Standard, " IEEE Communications Magazine, v. 23, n. 9. Sep 1985, pp. 41-46.

76. D. Balenson, "Privacy Enhancement for Internet Electronic Mail: Part 111: Algo 91. rithms, Modes, and Identifiers, " RFC 1423, Feb 1993.

77. D. Balenson, C.M. Ellison, S.B. Lipner, and S.T. Walker, "A New Approach to Software Key Escrow Encryption," TIS Report #520, Trusted Information Systems, Aug 94

78. R. Ball, Mathematical Recreations and Essays, New York: MacMillan, 1960.

79. J. Bamford, The Puzzle Palace, Boston: 93. Houghton Mifflin, 1982.

80. J. Bamford and W. Madsen, The Puzzle Palace, Second Edition, Penguin Books, 1995.

81. S.K. Banerjee, "High Speed Implementa- tion of DES," Computers ed Security, v. l, 1982, pp. 261-267.

82. Z. Baodong, "MC-Veiled Linear Transform Public Key Cryptosystem," Acta Electron- ica Sinica, v. 20, n. 4, Apt 1992, pp. 21-24. {In Chinese 1

83. P.H. Bardell, "Analysis of Cellular Automata Used as Pseudorandom Pattern Generators," Proceedings of 1990 Interna- tional Test Conference, pp. 762-768.

84. T. Baritaud, H. Gilbert, and M. Girault, "FFT Hashing is not Collision-Free, " Advances in Cryptology EUR OCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 35-44.

85. C. Barker, "An Industry Perspective of the CCEP, " 2nd Annual AIAA Computer Security Conference Proceedings, 1986.

86. W.G. Barker, Cryptanalysis of the Hagelin Cryptograph, Aegean Park Press, 1977.

87. R Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor," Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp 311-323.

88. T.C. Bartee and D.l. Schneider, "Computation with Finite Fields," Information and Control, v. 6, n. 2, Jun 1963, pp. 79-98.

89. U. Baum and S. Blackburn, "Clock Controlled Pseudorandom Generators on Finite Groups," K.U Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

90. K.R. Bauer, T.A. Bersen, and R.J. Feiertag, "A Key Distribution Protocol Using Event Markers," ACM Transactions on Computer Systems, v. 1, n. 3, 1983, pp. 249-255.

91. F. Bauspiess and F. Damm, "Requirements for Cryptographic Hash Functions," Computers Security, v. l1, n. 5, Sep 1992, pp. 427 437.

92. D. Bayer, S. Haber, and W.S. Stornetta, "Improving the Efficiency and Reliability of Digital Time-Stamping, " Sequences '91: Methods in Communication, Security, and Computer Science, Springer-Verlag, 1992, pp. 329-334.

93. R. Bayer and J.K. Metzger, "On the Encipherment of Search Trees and Random Access Files," ACM Transactions on Data base Systems, v. l, n. 1, Mar 1976, pp. 37-52.

94. M. Beale and M.F. Monaghan, "Encrytion Using Random Boolean Functions," Cryptography and Coding, H.J. Beker and F.C. Piper, eds., Oxford: Clarendon Press, 1989, pp. 219-230.

95. P. Beauchemin and G. Brassard, "A Gener- alization of Hellman's Extension to Shannon's Approach to Cryptography," lournal of Cryptology, v. 1, n. 2, 1988, pp. 129-132.

96. R Beauchemin, G. Brassard, C. Crepeau, C. Goutier, and C. Pomerance, "The Generation of Random Numbers that are Probably Prime, " Journal of Cryptology, v. 1, n. 1, 1988, pp. 53-64.

97. D. Beaver, J. Feigenbaum, and V Shoup, "Fliding Instances in Zero-Knowledge Proofs, " Advances in Cryptology CR YPTO '90 Proceedings, Springer-Verlag, 1991, pp. 326-338.

98. H. Beker, J. Friend, and P. Halliden, "Simplifying Key Management in Electronic Funds Transfcr Points of Sale Systems," Electronics Letters, v. 19, n. 12, Jun 1983, pp. 442 444.

99. H. Beker and F. Piper, Cipher Systems: The Protection of Communications, London: Northwood Books, 1982.

100. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Mathematical Foundations, " Report ESD-TR-73-275, MITRE Corp., 1973.

101. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: A Mathematical Model," Report MTR-2547, MITRE Corp., 1973.

102. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: A Refinement of the Mathematical Model," Report ESD-TR-73-278, MITRE Corp., 1974.

103. D.E. Bell and L.J. LaPadula. "Secure Com- puter Systems: Unified Exposition and Multics Interpretation," Report ESD-TR- 75-306, MITRE Corp., 1976.

104. M. Bellare and S. Goldwasser, "New Paradigms for Digital Signatures and Message Authentication Based on Non- interactive Zero Knowledge Proofs, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 194-211.

105. M. Bellare and S. Micali, "Non-interactive Oblivious Transfer and Applications, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp.547-557.

106. M. Bellare, S. Micali, and R. Ostrovsky, "Perfect Zero-Knowledge in Constant Rounds," Proceedings of the 22nd ACM Symposium on the Theory of Computing, 1990, pp. 482-493.

107. S.M. Bellovin, "A Preliminary Technical Analysis of Clipper and Skipjack," unpublished manuscript, 20 Apr 1993.

108. S.M. Bellovin and M. Merritt, "Limitations of the Kerberos Protocol, " Winter 1991 USENIX Conference Proceedings, USENIX Association, 1991, pp. 253-267.

109. S.M. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks," Pro ceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy, 1992, pp. 72-84.

110. S.M. Bellovin and M. Merritt, "An Attack on the Interlock Protocol When Used for Authentication, " IEEE Transactions on Information Theory, v. 40, n. 1, Jan 1994, pp. 273-275.

111. S.M. Bellovin and M. Merritt, "Cryptographic Protocol for Secure Communications, " U.S. Patent #5,241,599, 31 Aug 93.

112. J. Ben-Aroya and E. Biham, "Differential Cryptanalysis of Lucifer, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 187-199.

113. J.C. Benaloh, "Cryptographic Capsules: A Disjunctive Primitive for Interactive Protocols," Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, 213-222.

114. J.C. Benaloh, "Secret Sharing Homorphisms: Keeping Shares of a Secret Secret, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987. pp. 251-260.

115. J.C. Benaloh, "Verifiable Secret-Ballot Elections, " Ph.D. dissertation, Yale University, YALEU/DCS/TR-561, Dec 1987.

116. J.C. Benaloh and M. de Mare, "One-Way Accumulators: A Decentralized Alternative to Digital Signatures," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 274 285.

117. J.C. Benaloh and D. Tuinstra, "Receipt Free Secret Ballot Elections," Proceedings of the 26th ACM Symposium on the Theory of Computing, 1994, pp. 544-553.

118. J.C. Benaloh and M. Yung, "Distributing the Power of a Government to Enhance the Privacy of Voters, " Proceedings of the 130. 5th ACM Symposium on the Principles in Distributed Computing, 1986, pp. 52-62.

119. A. Bender and G. Castagnoli, "On the Implementation of Elliptic Curve Cryptosystems, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 186-192.

120. S. Bengio, G. Brassard, Y.G. Desmedt, C. Goutier, and J.-J. Quisquater, "Secure Implementation of Identification Systems, " Journal of Cryptology, v. 4, n. 3, 1991, pp. 175-184.

121. C.H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental Quantum Cryptography, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 253-265.

122. C.H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental Quantum Cryptography, "Journal of Cryptology, v. 5, n. 1, 1992, pp. 3-28.

123. C.H. Bennett and G. Brassard, "Quantum Cryptography: Public Key Distribution and Coin Tossing, " Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Banjalore, India, Dec 1984, pp. 175-179.

124. C.H. Bennett and G. Brassard, "An Update on Quantum Cryptography," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 475-480.

125. C.H. Bennett and G. Brassard, "Quantum Public-Key Distribution System, " IBM Technical Disclosure Bulletin, v. 28, 1985, pp. 3153-3163.

126. C.H. Bennett and G. Brassard, "Quantum Public Key Distribution Reinvented, " S IGACT News, v. 18, n.4, 1987, pp. 51-53.

127. C.H. Bennett and G. Brassard, "The Dawn of a New Era for Quantum Cryptography: The Expenmental Prototype is Working!" SIGACT News, v. 20, n. 4, Fall 1989, pp. 78-82.

128. C.H. Bennett, G. Brassard, and S. Breidbart, Quantum Cryptography 11: How to Re-Use a One-Time Pad Safely Even if P=NP, unpublished manuscript, Nov 1982.

129. C.H. Bennett, G. Brassard, S. Breidbart, and S. Weisner, "Quantum Cryptography, or Unforgeable Subway Tokens," Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp. 267-275.

130. C.H. Bennett, G. Brassard, C. Crepeau, and M.-H. Skubiszewska, "Practical Quantum Oblivious Transfer, " Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 351-366.

131. C.H. Bennett, G. Brassard, and A.K. Ekert, "Quantum Cryptography," Scientific American, v. 267, n.4, Oct 1992, pp. 50-57.

132. C.H. Bennett, G. Brassard, and N.D. Mermin, "Quantum Cryptography Without Bell's Theorem, " Physical Review Letters, v.68, n.5, 3 Feb 1992, pp. 557-559.

133. C.H. Bennett, G. Brassard, and J.-M. Robert, "How to Reduce Your Enemy's Information," Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 468-476.

134. C.H. Bennett, G. Brassard, and J.-M. Robert, "Privacy Amplification by Public Discussion," SIAM Journal on Computing, v. 17, n.2, Apr 1988, pp. 210-229.

135. J. Bennett, "Analysis of the Encryption Algorithm Used in WordPerfect Word Processing Program," Cryptologia, v. 1 l, n. 4, Oct 1987, pp. 206-210.

136. M. Ben-Or, S. Goldwasscr, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation, " Proceedings of the 20th ACM Symposium on the Theory of Computing, 1988, pp. 1-10.

137. M. Ben-Or, O. Goldreich, S. Goldwasser, J. Hastad, J. Kilian, S. Micali, and R Rogaway, "Everything Provable is Provable in Zero-Knowledge," Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 37-56.

138. M. Ben-Or, O. Goldreich, S. Micali, and R.L. Rivest, "A Fair Protocol for Signing Contracts," IEEE Transactions on Information Theory, v. 36, n. 1, Jan 1990, pp. 40 46.

139. H.A. Bergen and W.J. Caelli, "File Security in WordPerfect 5.0," Cryptologia, v. 15, n. 1, Jan 1991, pp. 57-66.

140. E.R. Berlekamp, Algebraic Coding Theory, Aegean Park Press, 1984.

141. S. Berkovits, "How to Broadcast a Secret," Advances i n Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 535-541.

142. S. Berkovits, J. Kowalchuk, and B. Schanning, "Implementing Public-Key Scheme, " IEEE Communications Magazine, v. 17, n. 3, May 1979, pp. 2-3.

143. D.J. Bernstein, Bernstein vs. U.S. Depart- ment of State et al., Civil Action No. C95-0582-MHP, United States District Court for the Northern District of California, 21 Feb 1995.

144. T. Berson, "Differential Cryptanalysis Mod 232 with Applications to MD5, " Advances in Cryptology EUROCRYPT '92 Proceedings, 1992, pp. 71-80.

145. T. Beth, Verfahren der schnellen Fourier-Transformation, Teubner, Stuttgart, 1984. (In German.)

146. T. Beth, "Efficient Zero-Knowledge Identification Scheme for Smart Cards," Advances in Cryptology EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 77-84.

147. T. Beth, B.M. Cook, and D. Gollmann, "Architectures for Exponentiation in GF|2n|," Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 302-310.

148. T. Beth and Y. Desmedt, "Identification Tokens or: Solving the Chess Grandmastcr Problem," Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 169-176.

149. T. Beth and C. Ding, "On Almost Nonlinear Permutations, " Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 65-76.

150. T. Beth, M. Frisch, and G.J. Simmons, eds., Lecture Notes in Computer Science 578; Public Key Cryptography: State of the Art and Future Directions, Springer-Verlag, 1992.

151. T. Beth and F.C. Piper, "The Stop-and-Go Generator," Advances in Cryptology: Procedings of EUROCRYPT 84, Springer-Verlag, 1984, pp. 88-92.

152. T. Beth and F. Schaefer, "Non Supersingular Elliptic Curves for Public Key Cryptosystems," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 316-327.

153. A. Beutelspacher, "How to Say 'No', " Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 491-96.

154. J. Bidzos, letter to NIST regarding DSS, 20 Sep 1991.

155. J. Bidzos, personal communication, 1993. 169.

156. R Bieber, "A Logic of Communication in a Hostile Environment," Proceedings of the Computer Security Foundations Workshop, IEEE Computer Society Press, 1990, pp. 14-22.

157. E. Biham, "Cryptanalysts of the Chaotic- Map Cryptosystem Suggested at EUROCRYPT '91, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 532-534.

158. E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys, " Technical Report #753, Computer Science Department, Technion Israel Institute of Technology, Sep 1992.

159. E. Biham, "On the Applicability of Differential Cryptanalysis to Hash Functions," lecture at EIES Workshop on Cryptographic Hash Functions, Mar 1992.

160. E. Biham, personal communication, 1993.

161. E. Biham, "Higher Order Differential Cryptanalysis, " unpublished manuscript, Jan 1994.

162. E. Biham, "On Modes of Operation," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994,pp. 116-120.

163. E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," Jo urnal of Cryptology, v. 7, n. 4, 1994, pp. 229-246.

164. E. Biham, "On Matsui's Linear Cryptanalysis, " Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, pp. 398 -412.

165. E. Biham and A. Biryukov, "How to Strengthen DES Using Existing Hardware, " Advances in Cryptology ASIACKYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

166. E. Biham and P.C. Kocher, "A Known Plaintext Attack on the PKZIP Encryption," K.U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

167. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," Advances in Cryptology- CRYPTO 90 Proceedings, Springer- Verlag, 1991, pp. 2-21.

168. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," Journal of Cryptology, v. 4, n. 1, 1991, pp 3-72.

169. E. Biham and A. Shamir, "Differential Cryptanalysis of Feal and N-Hash, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 181. 1-16.

170. E. Biham and A. Shamir, "Differential Cryptanalysis of Snefru, Khafre, REDOC- II, LOKI, and Lucifer," Advances in Cryptology CRYPTO '91 Proceedings, 1992, pp. 156-171.

171. E. Biham and A. Shamir, "Differential Cryptanalysis of the Full 16-Round DES," Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, 487- 496.

172. E. Biham and A. Shamir, Differential Cryptanalysis of the Data Encryption Standard, Springer-Verlag, 1993.

173. R. Bird, I. Gopal, A. Herzberg, R Janson, S. Kutten, R. Molva, and M. Yung, "Systematic Design of Two-Party Authentication Protocols, " Advances in Cryptology CRYPTO'91 Proceedings, Springer-Verlag, 1992, pp. 44-61.

174. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "System- atic Design of a Family of Attack-Resistant Authentication Protocols, " IEEE journal of Selected Areas in Communication, to appear.

175. R. Bird, I. Gopal, A. Herzberg R Janson, S. Kutten, R. Molva, and M. Yung, "A Modular Family of Secure Protocols for Authentication and Key Distribution," IEEE/ACM Transactions on Networking, to appear.

176. M. Bishop, "An Application for a Fast Data Encryption Standard Implementation, " Computing Systems, v. 1, n. 3, 1988, pp. 221-254.

177. M. Bishop, "Privacy-Enhanced Electronic Mail," Distributed Computing and Cryptography, J. Feigenbaum and M. Merritt, eds., American Mathematical Society, 1991, pp. 93-106.

178. M. Bishop, "Privacy-Enhanced Electronic Mail, " Internetworking: Research and Experience, v. 2, n. 4, Dec 1991, pp. 199-233.

179. M. Bishop, "Recent Changes to Privacy Enhanced Electronic Mail," Internetworking: Research and Experience, v. 4, n. 1, Mar 1993, pp. 47-59.

180. I.F. Blake, R. Fuji-Hara, R.C. Mullin, and S.A. Vanstone, "Computing Logarithms in Finite Fields of Characteristic Two, " SIAM Journal on Algebraic Discrete Methods, v. 5, 1984, pp. 276-285.

181. I.F. Blake, R.C. Mullin, and S.A. Vanstone, "Computing Logarithms in GF (2n), " Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 73-82.

182. G.R. Blakley, "Safeguarding Cryptographic Keys," Proceedings of the National Computer Conference, 1979, American Federation of Information Processing Societies, v. 48. 1979, pp. 313-317.

183. G.R. Blakley, "One-Time Pads are Key Safeguarding Schemes, Not Cryptosystems Fast Key Safeguarding Schemes (Threshold Schemes Exist), " Proceedings of the 1980 Symposium on Security and Privacy, IEEE Computer Society. Apr 1980, pp. 108-113.

184. G.R. Blakley and I. Borosh, "Rivest-Shamir-Adleman Public Key Cryptosystems Do Not Always Conceal Messages," Computers and Mathematics with Applications, v. 5, n. 3, 1979, pp. 169-178.

185. G.R. Blakley and C. Meadows, "A Database Encryption Scheme which Allows the Computation of Statistics Using Encrypted Data," Proceedings of the 1985 Symposium on Security and Privacy, IEEE Computer Society, Apr 1985, pp. 116-122.

186. M. Blaze, "A Cryptographic File System for UNIX," 1st ACM Conference on Computer and Communications Security, ACM Press, 1993, pp. 9-16.

187. M. Blaze, "Protocol Failure in the Escrowed Encryption Standard, " 2nd ACM Conference on Computer and Communications security, ACM Press, 1994, pp. 59-67.

188. M. Blaze, "Key Management in an Encrypting File System, " Proceedings of the Summer 94 USENIX Conference, USENIX Association, 1994, pp. 27-35.

189. M. Blaze and B. Schneier, "The MacGuffin Block Cipher Algorithm, " K. U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

190. U. Blocher and M. Dichtl, "Fish: A Fast Software Stream Cipher," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 41-44.

191. R. Blom, "Non-Public Key Distribution," Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp. 231-236.

192. K.J. Blow and S.J.D. Phoenix, "On a Fundamental Theorem of Quantum Cryptography, " Journal of Modern Optics, v. 40, n. 1, Jan 1993, pp. 33-36.

193. L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," SIAM Journal on Computing, v. 15, n. 2, 1986, pp. 364-383.

194. M. Blum, "Coin Flipping by Telephone: A Protocol for Solving Impossible Problems," Proceedings of the 24th IEEE Computer Conference (CompCon), 1982, pp. 133-137.

195. M. Blum, "How to Exchange Secret Keys, " ACM Transactions on Computer Systems, v. 1, n. 2, May 1983, pp. 175-193.

196. M. Blum, "How to Prove a Theorem So No Onc Else Can Claim It," Proceedings of the International Congress of Mathematicians, Berkeley, CA, 1986, pp. 1444-1451.

197. M. Blum, A. De Santis, S. Micali, and G. Persiano, "Noninteractive Zero-Knowledge, " SIAM Journal on Computing, v. 20, n. 6, Dec 1991. pp. 1084-1118.

198. M. Blum, P. Feldman, and S. Micali, "Non Interactivc Zero-Knowledge and Its Applications, " Proceedings of the 20th ACM Symposilzm on Theory of Computing, 1988, pp. 103-112.

199. M. Blum and S. Goldwasser, "An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information," Advances in Cryptology: Proceedings of C RYPTO 84, Springer-Verlag, 1985, pp. 289-299.

200. M. Blum and S. Micali, "How to Generate Cryptographically-Strong Sequences of Pseudo-Random Bits," SIAM Journal on Computing, v. 13, n. 4, Nov 1984, pp. 850-864.

201. B. den Boer. "Cryptanalysts of F.E.A.L.," Advances in Cryptology E UROCRYPT '88 Proceeding.s, Springer-Verlag, 1988, pp. 293-300.

202. B. den Boer and A. Bosselaers, "An Attack on the Last Two Rounds of MD4, " Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 1 94-203.

203. B. den Boer and A. Bosselaers, "Collisions for the Compression Function of M D5," Advances in Cryptology EUROCRYPT 93 Proceedings, Springer-Verlag, 1994, pp. 293

204. J.-P. Boly, A. Bosselaers, R. Cramer, R. Michelsen, S. Mjolsnes, F. Muller, T. Pedersen, B. Pfitzmann, R de Rooij, B. Schoenmakers, M. Schunter, L. Vallee, and M. Waidner, "Digital Payment Systems in the ESPRIT Project CAFE, " Securicom 94, Paris, France, 2-6 Jan 1994, pp. 35-45.

205. J.-R Boly, A. Bosselaers, R. Cramer, R. Michelsen, S. Mjolsnes, F. Muller, T. Pcdersen, B. Pfitzmann, P. de Rooij, B. Schoen makers, M. Schunter, L. Vallee, and M. Waidner, "The ESPRIT Project CAFE High Security Digital Payment System," Computer Security ESORICS 94, Springer-Verlag, 1994, pp. 217-230.

206. D.J. Bond, "Practical Primality Testing," Proceedings of IKE International Conference on Secure Communications Systems, 22-23 Feb 1984, pp. 50-53.

207. H. Bonnenberg, Secure Testing of VSLI Cryptographic Equipment, Series in Microelectronics, Vol. 25, Konstanz: Hartung Gorre Verlag, 1993.

208. H. Bonnenberg, A. Curiger, N. Felber, H. Kacslin, and X. Lai, "VLSI Implementation of a New Block Cipher," Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD 91), Oct 1991, pp. 510 -513.

209. K.S. Booth, "Authentication of Signatures Using Public Key Encryption," Commu nications of the ACM, v. 24, n. 11, Nov 1981, pp. 772-774,

210. A. Bosselaers, R. Govaerts, and J. Vanderwalle, Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 175-186.

211. D.R Bovet and P. Crescenzi, Introduction to the Theory of CompiexiLy, Englewood Cliffs, N.J.: Prenticc-Hall, 1994.

212. J. Boyar, "Inferring Scqucnccs Produced by a Linear Congruential Generator Missing Low-Order Bits." Journal of Cryptology, v. 1, n. 3, 1989, pp. 177-184.

213. J. Boyar, D. Chaum, and I. Damgard, "Convertible Undeniable Signatures," Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 189-205.

214. J. Boyar, K. Fricdl, and C. Lund, "Practical Zero-Knowledge Proofs: Giving Hints and Using Deficiencies, " Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 155-172.

215. J. Boyar, C. Lund, and R. Peralta, "On the Communication Complexity of Zero Knowledge Proofs, " Journal of Cryptology, v.6, n.2, 1993, pp.65-85.

216. J. Boyar and R. Peralta, "On the Concrete Complexity of Zero-Knowledge Proofs, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag 1990, pp. 507-525.

217. C. Boyd, "Some Applications of Multiple Key Ciphers," Advances in Cryptology EUROCRYPT '88 Proceedings, Springer Verlag, 1988, pp. 455-467.

218. C. Boyd, "Digital Multisignatures," Cryptography and Coding, H.J. Beker and F.C. Piper, eds., Oxford: Clarendon Press, 1989, pp. 241-246.

219. C. Boyd, "A New Multiple Key Cipher and an Improved Voting Scheme," Advances in Cryptology EUROCRYPT '89 Proceed ings, Springer-Verlag, 1990, pp. 617 625.

220. C. Boyd, "Multisignatures Revisited," Cryptography and Coding, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 21-30.

221. C. Boyd and W. Mao, "On the Limitation of BAN Logic, " Advances in Cryptology EUROCRYPT '93 Proceedings, Springer Verlag, 1994, pp. 240-247.

222. C. Boyd and W. Mao, "Designing Secure Key Exchange Protocols," Computer Secu rity ESORICS 94, Springer-Verlag, 1994, pp. 217-230.

223. B. O. Brachtl, D. Coppersmith, M.M. Hyden, S.M. Matyas, C.H. Meyer, J. Oseas, S. Pilpel, and M. Schilling, "Data Authentication Using Modification Detection Codes Based on a Public One Way Function," U.S. Patent #4,908,861, 13 Mar 1990.

224. J. Brandt, I.B. Damgard, R Landrock, and T. Pederson, "Zero-Knowledge Authentication Scheme with Secret Key Exchange," Advances in Cryptology CRYPTO '88, Springer-Verlag, 1990, pp. 583-588.

225. S.A. Brands, "An Efficient Off-Line Electronic Cash System Based on the Representation Problem," Report CS-R9323, Computer Science / Department of Algorithms and Architecture, CWI, Mar 1993.

226. S.A. Brands, "Untraceable Off-line Cash in Wallet with Observers," Advances in Cryptology CRYPTO '93, Springer Verlag, 1994, pp. 302-318.

227. S.A. Brands, "Electronic Cash on the Internet," Proceedings of the Internet Society Symposium on Network and Distributed Systems Secunty, IEEE Computer Society Press 1995, pp 64-84.

228. D.K. Branstad, "Hellman's Data Does Not Support His Conclusion," IEEE Spectrum, v. 16, n. 7, Jul 1979, p. 39.

229. D.K. Branstad, J. Gait, and S. Katzke, "Report on the Workshop on Cryptography in Support of Computer Security, " NBSIR 77-1291, National Bureau of Standards, Sep 21-22, 1976, September 1977.

230. G. Brassard, "A Note on the Complexity of Cryptography, " IEEE Transactions on Information Theory, v. IT-25, n. 2, Mar 1979, pp. 232-233.

231. G. Brassard, "Relativized Cryptography," Proceedings of the IEEE 20th Annual Symposium on the Foundations of Computer Science, 1979, pp. 383-391.

232. G. Brassard, "A Time-Luck Trade-off in Relativized Cryptography, " Proceedings of the IEEE 21st Annual Symposium on the Foundations of Computer Science, 1980, pp. 380-386.

233. G. Brassard, "A Time-Luck Tradeoff in Relativized Cryptography," Journal Of Computer and System Sciences, v. 22, n.3, Jun 1981, pp. 280-311.

234. G. Brassard, "An Optimally Secure Relativized Cryptosystem," SIGACT News, v. 15, n. 1, 1983, pp. 28-33.

235. G. Brassard, "Relativized Cryptography," IEEE Transactions on Information Theory, v. IT-29, n. 6, Nov 1983, pp. 877-894.

236. G. Brassard, Modern Cryptology: A Tutorial, Springer-Verlag, 1988.

237. G. Brassard, "Quantum Cryptography: A Bibliography," SIGACT News, v. 24, n. 3, Oct 1993, pp. 16-20.

238. G. Brassard, D. Chaum, and C. Crepeau, "An Introduction to Minimum Disclosure," CWI Quarterly v. 1, 1988, pp. 3-17.

239. G. Brassard, D. Chaum, and C. Crepeau, "Minimum Disclosure Proofs of Knowledge," Journal of Computer and System Sciences, v. 37, n.2, Oct 1988, pp. 156-189.

240. G. Brassard and C. Crepeau, "Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond," Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 188-195.

241. G. Brassard and C. Crepeau, "Zero- Knowledge Simulation of Boolean Circuits," Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 22 5-233.

242. G. Brassard and C. Crcpeau, "Sorting Out Zcro-Knowlcdge, " Advances in Cryptology EUROCRYPT '89 Proceedings, Springcr-Vcrlag, 1990, pp. 181-191.

243. G. Brassard and C. Crcpcau, "Quantum Bit Commitment and Coin Tossing Protocols, " Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 49-61.

244. G. Brassard, C. Crepeau, R. Jozsa, and D. Langlois, "A Quantum Bit Commitment Schcmc Provably Unbreakable by Both Parties, " Proceedings of the 34th IEEE Symposium on Foundations of computer Science, 1993, pp. 362-371.

245. G. Brassard, C. Crepeau, and J.-M. Robert, "Information Theoretic Reductions Among Disclosure Problems, " Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 168-173.

246. G. Brassard, C. Crcpeau, and J.-M. Robert, "All-or-Nothing Disclosure of Secrets, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 234-238.

247. G. Brassard, C. Crepeau, and M. Yung, "Everything in NP Can Be Argued in Perfect Zero-Knowledge in a Bounded Number of Rounds," Proceedings on the 16th Inter national Colloquium on Automata, Languages, and Programming, Springer-Verlag, 1989, pp. 123-136.

248. R.P. Brent, "An Improved Monte-Carlo Factorization Algorithm," BIT v. 20, n. 2, 1980, pp. 176-184.

249. R.P. Brent, "On the Periods of Generalized 261. Fibonacci Recurrences, Mathematics of Computation, v 63, n. 207, Jul 1994, pp. 389-401.

250. R.R Brent, "Parallel Algorithms for Integer Factorization," Research Report CMA-R49-89, Computer Science Laboratory The Australian National University, Oct 1989.

251. D.M. Bressotid, Factorization and Primality Testing, Springer-Verlag, 1989.

252. E.F. Brickcll, "A Fast Modular Multiplication Algorithm with Applications to Two Key Cryptography," Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1982, pp. 51-60.

253. E.F. Brickell, "Are Most Low Density Polynomial Knapsacks Solvable in Polynomial Timer" Proceedings of the 14th Southeastern Conference on Combinatorics, Graph Theory, and Computing, 1983.

254. E.F. Brickell, "Solving Low Density Knapsacks," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 25-37.

255. E.F. Brickell, "Breaking Iterated Knapsacks," Advances in Cryptology: Proceedings of Crypto 84, Springer-Verlag, 1985, pp. 342-358.

256. E.F. Brickell, "Cryptanalysts of the Uagisawa Public Key Cryptosystem," Abstracts of Papers, EUROCRYPT '86, 20-22 May 1986.

257. E.F. Brickell, "The Cryptanalysis of Knapsack Cryptosystems, " Applications of Discrete Mathematics, R.D. Ringeisen and F.S. Roberts, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988, pp. 3-23.

258. E.F. Brickell, "Survey of Hardware Implementations of RSA, " Advances in Cryptology CRYPTO '89 Proceedings, Springcr-Verlag, 1990, pp. 368-370.

259. E.F. Brickell, D. Chaum, I.B. Damgard, and J. van de Graff, "Gradual and Verifiable Release of a Secret," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 156-166.

260. E.F. Brickell, J.A. Davis, and G.J. Simmons, "A Preliminary Report on the Cryptanalysis of Merkle-Hellman Knapsack, " Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp. 289-303.

261. E.F. Brickell and J. DeLaurentis, "An Attack on a Signature Scheme Proposed by Okamoto and Shiraishi, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 28-32.

262. E.F. Brickell, D.E. Denning, S.T. Kent, D.R Maher, and W. Tuchman, "SKIPJACK Review Interim Report, " unpublished manuscript, 28 Jul 1993.

263. E.F. Brickell, J.C. Lagarias, and A.M. Odlyzko, "Evaluation of the Adleman Attack of Multiple Iterated Knapsack Cryptosystems," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 39-42.

264. E.F. Brickell, RJ. Lee, and Y. Yacobi, "Secure Audio Teleconference," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 418 426.

265 . E. F. Brickell and K. S. McCurley, "An Interactive Identification Scheme Based on Discrete Logarithms and Factoring, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 63-71.

266. E.F. Brickell, J.H. Moore, and M.R. Purtill, "Structure in the S-Boxes of the DES," Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 3-8.

267. E.F. Brickell and A.M. Odlyzko, "Crypt analysis: A Survey of Recent Results," Pro ceedings of the IEEE, v. 76, n. 5, May 1988, 279. pp. 578-593.

268. E.F. Brickell and A.M. Odlyzko, "Crypt analysis: A Survey of Recent Results," Contemporary Cryptology: The Science of Information Integnty, G.J. Simmons, ed., IEEE Press, 1991, pp. 501-540.

269. E.F. Brickell and G.J. Simmons, "A Status Report on Knapsack Based Public Key Cryptosystems, " Congressus Numeran tium, v. 7, 1983, pp. 3-72.

270. E.F. Brickell and D.R. Stinson, "The Detection of Cheaters in Threshold Schemes," Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 564-577.

271. A.G. Broscius and J.M. Smith, "Exploiting Parallelism in Hardware Implementation of the DES, " Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 367-376.

272. L. Brown, M. Kwan, J. Pieprzyk, and J. Seberry, "Improving Resistancc to Differential Cryptanalysis and the Redesign of LOKI, " Advances in Cryptology ASIACRYPT '91 Proceedings, Springer-Verlag, 1993, pp. 36-50.

273. L. Brown, J. Pieprzyk, and J. Seberry, "LOKI: A Cryptographic Primitive for Authentication and Secrecy Applications," Advances in Cryptology AUSCRYPT ' 90 Proceedings, Springer Verlag, 1990, pp. 229-236.

274. L. Brown, J. Pieprzyk, and J. Seberry, "Key Scheduling in DES Type Cryptosystems," Advances in Cryptology A IJSCRYPT '90 Proceedings, Springer-Verlag, 1990, pp. 221-228.

275. L. Brown and J. Seberry, "On the Design of Permutation P in DES Type Cryptosystems," Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 696-705.

276. W. Brown, "A Quantum Leap in Secret Communications, " New Scientist, n. 1585, 30 Jan 1993, p. 21.

277. J.O. Bruer, "On Pseudo Random Sequences as Crypto Generators," Proceedings of the International Zurich Seminar on Digital Communication, Switzerland, 1984.

278. L. Brynielsson "On the Linear Complexity of Combined Shift Register Sequences," Advances in Cryptology EUROCRYPT '85, Springer-Verlag, 1986, pp. 156-166.

279. J. Buchmann, J. Loho, and J. Zayer, "An Implementation of thc General Number Field Sieve, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 159-165.

280. M. Burmester and Y. Desmedt, "Broadcast Interactive Proofs," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 81-95.

281. M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System, " Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

282. D. Burnham, "NSA Seeking 500,000 'Secure' Telephones," The New York Times, 6 Oct 1994.

283. M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication, " Research Report 39, Digital Equipment Corp. Sys- tems Research Center, Feb 1989.

284. M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," ACM Trans- actions on Computer Systems, v. 8, n. 1, Feb 1990, pp. 18-36.

285. M. Burrows, M. Abadi, and R. Needham, "Rejoinder to Nessett," Operating System Review, v. 20, n. 2, Apr 1990, pp. 39 40.

286. J.J. Cadc, "A Modification of a Broken Pu blic-Key Cipher," Advances in Cryptology - CRYPTO '86 Proceedillgs, Springer- Verlag, 1987, pp. 64-83.

287. T.R. Cain and A.T. Sherman, "How to Break Gifford's Cipher, " P roceedings of the 2nd Annual ACM Conference Computer and Communications Security 300 ACM Press, 1994, pp. 198-209.

288. C. Calvelli and V Varadharajan, "An Analysis of Some Delegation Protocols for Distributed Systems, " Proceedings of the Computer Security Foundations Workshop V, IEEE Computer Society Press, 1992, pp. 92-110.

289. J.L. Camenisch, J.-M. Piveteau, and M.A. Stadler, "An Efficient Electronic Payment System Protecting Privacy," Computer Security ESOR ICS 94, Springer-Verlag, 1994, pp. 207-215,

290. P. Camion and J. Patarin, "The Knapsack Hash Function Proposed at Crypto '89 Can Be Broken," Advances in Cryptology EUROCRYPT '91, Springer-Verlag, 1991, pp. 39-53.

291. C.M. Campbell, "Design and Specification of Cryptographic Capabilities," IEEE Computer Society Magazine, v. 16, n. 6, Nov 1978, pp. 15 19.

292. E.A. Campbell, R. Safavi-Naini, and PA. Pleasants, "Partial Belief and Probabilistic Reasoning in the Analysis of Secure Protocols," Proceedings of the Computer Security Foundations Workshop V, IEEE Computer Society Press, 1992, pp. 92-110.

293. K.W. Campbell and M.J. Wiener, "DES Is Not a Group," Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, pp. 512-520.

294. Z.F. Cao and G. Zhao, "Some New MC Knapsack Cryptosystems, " CHINACRYPT 307 . '94, Xidian, China, 11-15 Nov 1994, pp. 70-75. (In Chinese.)

295. C. Carlet, "Partially-Bent Functions, " Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 280 -291.

296. C. Carlet, "Partially Bent Functions," Designs, Codes and Cryptography. v. 3, 1993, pp. 135-145.

297. C. Carlet, "Two New Classes of Bent Functions" Advances in Cryptology EU ROCRYPT '93 Proceedings, Springer Vcrlag, 1994, pp. 77-101.

298. C. Carlet, J. Seberry, and X.M. Zhang, "Comments on Generating and Counting Binary Bent Sequences,' " IEEE Transac tions on Information Theory v. IT-40, n. 2, Mar 1994, p. 600.

299. J.M. Carroll, Computer Security, 2nd cdition, Butterworths 1987.

300. J.M. Carroll, "The Three Faces of Information Security," Advances in Cryptology AUSCRYPT '90 Proceedings, Springer-Verlag, 1990, pp. 433 -450.

301. J.M. Carroll, "'Do-it-yourself' Cryptography," Computers & Security v. 9, n. 7, Nov 1990, pp. 613-619.

302. T.R. Caron and R.D. Silverman, "Parallel Implementation of the Quadratic Scheme," Journal of Supercomputing, v. 1, n. 3, 1988, pp. 273-290.

303. CCITT, Draft Recommendation X.509, "The Directory Authentication Framework," Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1987.

304. CCITT, Recommendation X.509, "The Directory Authentication Framework, " Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.

305. CCITT, Recommendation X.800, "Security Architecture for Open Systems Interconnection for CCITT Applications, " International Telephone and Telegraph. International Telecommunications Union, Geneva, 1991.

306. F. Chabaud, "On the Security of Some Cryptosystems Based on Error-Correcting Codes," Advances in Cryptology EURO- CRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

307. F. Chabaud and S. Vaudenay, "Links Between Differential and Linear Cryptanalysis, " Advances in Cryptology- EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

308. W.G. Chambers and D. Gollmann, "Generators for Sequences with Near-Maximal Linear Equivalence," IKE l'roceedings, V. 135, Pt. E, n. 1, Jan 1988, pp. 67-69.

309. W.G. Chambers and D. Gollmann, "Lock-In Effect in Cascades of Clock-Controlled Shi ft Registers, " Advances in Cryptology EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 331-343.

310. A. Chan and R. Games, "On the Linear Span of Binary Sequences from Finite Geometries, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 405-417.

311. J.R Chandler, D.C. Arrington, D.R. Berkel- hammer, and W.L. Gill, "Identification and Analysis of Foreign Laws and Regulations Pertaining to the Use of Commercial Encryption Products for Voice and Data Communications, " National Intellectual Property Law Institute, George Washing- ton University, Washington, D.C., Jan 1994.

312. C.C. Chang and S.J. Hwang, "Cryptographic Authentication of Passwords, " Proceedings of the 25th Annual 1991 IEEE International Carnahan Conference on Security Technology, Taipei, Taiwan, 1-3 Oct 1991, pp. 126-130.

313. C.C. Chang and S.J. Hwang, "A Strategy for Transforming Public-Key Cryptosystems into Identity-Based Cryptosystems." Proceedings of the 25th Annual 1991 IEEE International Carnahan Conference on Security Technology, Taipei, Taiwan, 1-3 Oct 1991, pp. 68-72.

314. C.C. Chang and C.H. Lin, "An ID-Based Signature Scheme Based upon Rabin's Public Key Cryptosystem, " Proceedings of the 25th Annual 1991 IEEE International Carahan Conference on Secunty Technology, Taipei, Taiwan, 1-3 Oct 1991, pp. 139-141.

315. C. Charnes and J. Pieprzyk, "Attacking the SL2 Hashing Scheme," Advances in Cryptology ASIACRYPT '94 Proceedings, Springer-Verlag, 1995, pp. 322-330.

316. D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms, " Communications of the ACM,v.24,n.2,Febl981,pp.84 88.

317. D. Chaum, "Blind Signatures for Untraceable Payments," Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp. 199-203.

318. D. Chaum, "Security Without Identification: Transaction Systems to Make Big Brother Obsolete, " Communications of the ACM, v. 28, n. 10, Oct 1985, pp. 1030-1044.

319. D. Chaum, "Demonstrating that a Public Predicate Can Be Satisfied without Revealing Any Information about How, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 159-199.

320. D. Chaum, "Blinding for Unanticipated Signatures," Advances in Cryptology EUROCRYPT '87 Proceedings, Springer-Vcrlag, 1988, pp. 227-233.

321. D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Receiver Untraceability, " Journal of Cryptology, v. 1, n. 1, 1988, pp. 65-75.

322. D. Chaum, "Elections with Unconditionally Secret Ballots and Disruptions Equivalent to Breaking RSA," Advances in Cryptology EUROCRYPT '88 Proceedings. Springer-Verlag, 1988, pp. 177-181.

323. D. Chaum, "Blind Signature Systems, " U.S. Patent #4,759,063, 19 Jul 1988.

324. D. Chaum, "Blind Unanticipated Signature Systems," U.S. Patent #4,759,064, 19 Jul 1988.

325. D. Chaum, "Online Cash Checks, " Advances in Cryptology EUROCRYPT '89 Proceedings, Springcr-Verlag, 1990, pp. 288-293.

326. D. Chaum, "One-Show Blind Signature Systems," U.S. Patent #4,914,698, 3 Apr 1990.

327. D. Chaum, "Undeniable Signature Systems," U.S. Patent #4,947,430, 7 Aug 1990.

328. D. Chaum, "Returned-Value Blind Signature Systems," U.S. Patent #4,949,380, 14 Aug 1990.

329. D. Chaum, "Zero-Knowledge Undeniable Signatures, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 458-464.

330. D. Chaum, "Group Signatures," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 2.57-265.

331. D. Chaum, "Unpredictable Blind Signature Systems," U.S. Patent #4,991,210, 5 Feb 1991.

332. D. Chaum, "Achieving Electronic Privacy," Scientific American, v. 267, n. 2, Aug 1992, pp. 96-101.

333. D. Chaum, "Designated Confirmer Signatures," Advances in Cryptology E UROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

334. D. Chaum, C. Crepeau, and I.B. Damgard, "Multiparty Unconditionally Secure Protocols, " Proceedings of the 20th ACM Symposium on the Theory of Computing, 1988, pp. 11-19.

335. D. Chaum, B. den Boer, E. van Heyst, S. Mjolsnes, and A. Steenbeek, "Efficient Offline Electronic Checks," Advances in Cryptology E UK OCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 2Y4-301.

336. D. Chaum and J.-H. Evertse, "Cryptanalysis of DES with a Reduced Number of Rounds; Scqucnces of Linear Factors in Block Ciphers, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 192-211.

337. D. Chaum, J.-H. Evertse, and J. van de Graff, "An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations, " Advances in Cryptology EUROCRYPT '87 Proceedings, Springer-Verlag, 1988, pp. 127 141.

338. D. Chaum, J.-H. Evertse, J. van de Graff, and R. Peralta, "Demonstrating Possession of a Discrete Logarithm without Revealing It, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 200-212.

339. D. Chaum, A. Fiat, and M. Naor, "Untraceable Electronic Cash," Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 319-327.

340. D. Chaum and T. Pedersen, "Transferred Cash Grows in Size," Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 391-407.

341. D. Chaum and T. Pedersen, "Wallet Databases with Observers," Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 89-105.

342. D. Chaum and J. Schaumuller-Bichel, eds., Smart Card 2000, North Holland: Elsevier Science Publishers, 1989.

343. 1). Chaum and H. van Antwcrpen, "Und eniable Signaturcs," Advances in Cryptology CRYPTO '89 Proceedings, Springcr-Verlag, 1990, pp. 212-216.

344. D. Chaum, E. van Heijst, and B. Pfitzmann, "Cryptographically Strong Undeniab le Signatures, Unconditionally Secure for thc Signer, " Advances in Cryptology CRYPTO '91 Proceedings. Springer-Verlag, 1992, pp. 470-484.

345. T.M. Chee, "The Cryptanalysis of a New Public-Key Cryptosystem Based on Modular Knapsacks, " Advances in Cryptology CKYP TO '91 Proceedings, Springer-Verlag, 1992, pp. 204-212.

346. L Chen, "Oblivious Signatures," Computer Security ESORICS 94, Springcr-Verlag, 19 94, pp. 161-172,

347. L. Chen and M. Burminster, "A Practical Secret Voting Scheme which Allows Voters to Abstain," CHINACRYPT '94, Xidian, China, 11-15 Nov 1994, pp. 100-107.

348. L. Chen and T.P Pedersen "New Group Signature Schemes, " Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

349. J. Chenhui, "Spectral Characteristics of Partially-Bent Functions," CHINACRYPT '94, Xidian, China, 11-15 Nov 1994, pp. 48-51.

350. V. Chepyzhov and B. Smeets, "On a Fast Correlation Attack on Certain Stream Ciphers," Advances in Cryptology EUR OCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 176-185.

351. T.C. Cheung, "Management of PEM Public Key Certificates Using X.500 Directory Service: Some Problems and Solutions," Proceedings of the lnternet Society 1994 Workshop on Network and Distnhuted System Security. The Internet Society, 1994, pp. 35 42.

352. G.C. Chiou and W.C. Chen, "Secure Broadcasting Using the Secure Lock, " IEEE Transactions on Software Engineenng, v. SE-15, n. 8, Aug 1989, pp. 929-934.

353. Y.J. Choie and H.S. Hwoang, "On the Cryptosystem Using Elliptic Curves, " Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryp tography, Seoul, Korca, 24-26 Oct 1993, pp. 105-113.

354. B. Chor and O. Goldreich, "RSA/Rabin Least Significant Bits are 1/2+1/ ... Secure, " Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 303-313.

355. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults," Proceedings of the 26th Amlual IEEE Symposium on the Foundations of Computer Science, 1985, pp. 383-395.

356. B. Chor and R.L. Rivcst, "A Knapsack Typc Public Key Cryptosystem Based on Arithmctic in Finitc Fields," Advallces ill Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 54-65.

357. R Christoffersson, S.-A. Ekahll, V. Fak, S. Herda, R Mattila, W. Price, and H.-O. Widman, Crypto Users Handbook: A Guide for Implementors of Cryptograph hic Protection in Computer Systems, North Holland Elscvicr Scicncc Publishcrs, 1988.

358. R. Cleve, "Controlled Gradual Disclosure Schemes for Random Bits and Their Applications, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 572-588.

359. J.D. Cohen, "Improving Privacy in Cryptographic Elections," Yale University Computer Science Department Technical Report YALEU/DCS/TR-454, Feb 1986.

360. J.D. Cohen and M.H. Fischer, "A Robust and Verifiable Cryptographically Secure Election Scheme, " Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985. pp. 372-382.

361. R. Cole, "A Model for Security in Dis tributed Systems," Computers and Secu rity, v.9 , n.4, Apr 1990, pp.319-330.

362. Comptroller General of the United States, "Matter of National Institute of Standards and Technology Use of Electronic Data Interchange Technology to Create Valid Obligations," File B-245714, 13 Dec 1991.

363. M.S. Conn, letter to Joe Abernathy, National Security Agency, Ser: Q43-111 92, 10 Jun 1992.

364. C. Connell, "An Analysis of NewDES: A Modified Version of DES," Cryptologia, v. 14, n. 3, Jul 1990, pp. 217-223.

365. S.A. Cook, "The Complexity of Theorem Proving Procedures," Proceedings of the 3rd Annual ACM Symposium on the The oryofComputing, 1971,pp. 151-158.

366. R.H. Cooper and W. Patterson, "A Generalization of the Knapsack Method Using Galois Fields," Cryptologia, v. 8, n. 4, Oct 1984, pp. 343-347.

367. R.H. Cooper and W. Patterson, "RSA as a Benchmark for Multiprocessor Machines, " Advances in Cryptology AUSCRYPT'90 Proceedings, Springer-Verlag, 1990, pp. 356-359.

368. D. Coppersmith, "Fast Evaluation of Logarithms in Fields of Characteristic Two," IEEE Transactions on Information Theory, v.30,n.4,Jull984,pp.587-594.

369. D. Coppersmith, "Another Birthday Attack, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 14-17.

370. D. Coppersmith, "Cheating at Mental Poker, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 104-107.

371. D. Coppersmith, "The Real Reason for Rivest's Phenomenon, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 535-536.

372. D. Coppersmith, "Two Broken Hash Functions," Research Report RD 18397, IBM T.J. Watson Center, Oct 1992.

373. D. Coppersmith, "The Data Encryption Standard (DES) and Its Strength against Attacks," Technical Report RC 18613, IBM T.J. Watson Center, Dec 1992.

374. D. Coppersmith, "The Data Encryption Standard (DES) and its Strength against Attacks, " IBM /ournal of Research and Development, v. 38, n. 3, May 1994, pp. 243-250.

375. D. Coppersmith, "Attack on the Cryptographic Scheme NIKS-TAS," Advances in Cryptology CRYPTO '94 Proceedings, Springer-Verlag, 1994, pp. 294-307.

376. D. Coppersmith, personal communication, 1994.

377. D. Coppersmith and E. Grossman, "Generators for Certain Alternating Groups with Applications to Cryptography, " SIAM Journal on Applied Mathematics, v. 29, n. 4, Dec 1975, pp. 624-627.

378. D. Coppersmith, H. Krawczyk, and Y. Mansour, "The Shrinking Generator, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 22-39.

379. D. Coppersmith, A. Odlykzo, and R. Schroeppel, "Discrete Logarithms in GF (p)," Algorithmica, v. 1, n. 1, 1986, pp. 1-16.

380. D. Coppersmith and R Rogaway, "Software Efficient Pseudo Random Function and the Use Thereof for Encryption," U.S. Patent pending, 1995.

381. D. Coppersmith, J. Stern, and S. Vaudenay, "Attacks on the Birational Signature Schemes, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 435-443.

382. V. Cordonnier and J.-J. Quisquater, eds.. CARD1S '94 Proceedings of the First Smart Card Research and Advanced Application Conference, Lille, France, 24-26 Oct 1994.

383. C. Couvreur and J.-J. Quisquater, "An Introduction to Fast Generation of Large Prime Numbers," Philips /ournal Research, v. 37. n. 5 6, 1982, pp. 231-264.

384. C. Couvreur and J.-J. Quisquater, "An Introduction to Fast Generation of Large Prime Numbers," Philips journal Research, v. 38, 1983, i' 77

385. C. Coveyou and R.D. MacPherson, "Fourier Analysis of Uniform Random Number Gcncrators," lournal of the ACM, v. 14, n. 1, 1967, pp. 100-119.

386. T.M. Cover and R.C. King, "A Convergent Gambling Estimate of the Entropy of English," IEEE Tran.saction.s on Informa- tion Theory, v. IT-24, n. 4, Jul 1978, pp. 413-421.

387. R.J.F. Cramer and T.R Pedersen, "Improved Privacy in Wallets with Observers," Advances in Cryptology EZJROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 329-343.

388. R.E. Crandell, "Method and Apparatus for Public Key Exchange in a Cryptographic System," U.S. Patent #5,159,632, 27 Oct 1992.

389. C. Crepeau, "A Secure Poker Protocol That Minimizes the Effect of Player Coalitions," Advances in Cryptology CKYP'I'O '85 Proceedings, Springer-Verlag, 1986, pit. 73-86.

390. C. Crepcau, "A Zcro-Knowlcdge Poker Protocol that Achieves Confidentiality of the Players' Strategy, or How to Achieve an Electronic Poker Face, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 23Y-247,

391. C. Crepeau, "Equivalence Between Two Flavours of Oblivious Transfer," Advances in Cryptology CRYPTO 87 Proceedings, Springer-Ver lag, 1988, pp. 350-354.

392. C. Crepeau, "Correct and Private Reductions among Oblivious Transfers," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1990.

393. C. Crcpcau, "Quantum Oblivious Transfcr, " journal of Modern Optics, v. 41, n. 12, Dec 1994, pp. 2445-2454.

394. C. Crepeau and J. Kilian, "Achieving Oblivious Transfer Using Weakened Security Assumptions, " Proceedings of the 29th Amllzal Symposiurn on the Foundatijns of Computer Science, 1988, pp. 42-.32.

395. C. Crepeau and J. Kilian, "Weakening Security Assumptions and Oblivious Transfer, " Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 2-7.

396. C. Crepeau and L. Salvail, "Quantum Obliv ious Mutual Identification, " Advances in Cryptology EUROCRYPT '95 Proceed- ings, Springer-Verlag 1995, pp. 133-146.

397. A. Curiger, H. Bonnenberg, R. Zimmermann, N. Felber, H. Kaeslin and W. Fichtner, "VINCI: VLSI Implementation of the New Block Cipher IDEA," Proceedings of IEEE CICC '93, San Diego, CA, May 1993, pp. 15.5.1-15.5.4.

398. A. Curiger and B. Stuber, "Specification for the IDEA Chip, " Technical Report No. 92/03, Institut fur Integrierte Systeme, ETH Zurich, Feb 1992.

399. T. Cusick, "Boolean Functions Satisfying a Higher Order Strict Avalanche Criterion," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 102-117.

400. T.W. Cusick and M.C. Wood, "The REDOC-II Cryptosystem," Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 545-563.

401. Cylink Corporation, Cylink Corporation vs. RSA Data Security, Inc., Civil Action No. C94-02332-CW, United States District Court for the Northern District of Califor- nia, 30 Jun 1994.

402. J. Daeman, "Cipher and Hash Function Design, " Ph.D. Thesis, Katholieke Univer- siteit Leuven, Mar 95.

403. J. Daeman, A. Bosselaers, R. Govaerts, and J. Vandewalle, "Collisions for Schnorr's Hash Function FFT-Hash Presented at Crypto '91," Advances in Cryptology ASIA CRYPT '91 Proceedings, Springer- Verlag, 1993, pp. 477-480.

404. J. Daeman, R. Govaerts, and J. Vandewalle, "A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgard's Onc-Way Function Based on Cellular Automata, " Advances in Cryp- tology ASIA CRYPT '91 Proceedings, Springer-Verlag, 1993, pp. 82-96.

405. J. Daeman, R. Govaerts, and J. Vandewalle, "A Hardware Design Model for Crypto- graphic Algorithms, " ESORICS 92, Pro- ceedings of the Second European Sympo- sium on Research in Computer Security, Springer-Verlag, 1992, pp. 419 434.

406. J. Daemcn, R. Govacrts, and J. Vandewalle, "Block Ciphers Based on Modular Arith-
metic, " Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography, Rome, Italy, 15-16 Feb 1993, pp. 418. 80-89.

407. J. Daemen, R. Govaerts. and J. Vandewalle, "Fast Hashing Both in Hardware and Soft ware," presented at the rump session of CRYPTO '93, Aug 1993.

408. J. Daeman, R. Govaerts, and J. Vandewalle, "Resynchronization Weaknesses in Syn chronous Stream Ciphers," Advances in Cryptology E UR O CRYPT '93 Proceed ings, Springer-Verlag, 1994, pp. 159-167.

409. J. Daeman, R. Govaerts, and J. Vandewalle, "Weak Keys for IDEA, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 224-230.

410. J. Daemen, R. Govaerts, and J. Vandewalle, "A New Approach to Block Cipher Design," Fast Software Encryption, Cam bridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 18-32.

411. Z.-D. Dai, "Proof of Rueppel's Linear Complexity Conjecture," IEEE Transactions on Information Theory, v. IT-32, n. 3, May 1986, pp. 440 443.

412. I.B. Damgard, "Collision Free Hash Functions and Public Key Signature Schemes," Advances in Cryptology EUROCRYPT '87 Proceedings, Springer-Verlag, 1988, pp. 203-216.

413. I.B. Damgard, "Payment Systems and Credential Mechanisms with Provable Secu rity Against Abuse by Individuals, " Advances in Cryptology CRYPTO '88 I'roceedings, Springer-Verlag, 1990, pp. 328-335

414. I.B. Damgard, "A Design Principle for Hash Functions, " Advances in Cryptol428. ogy CRYPTO '89 Proceedings, Springer Verlag, 1990, pp. 416 427.

415. I.B. Damgard, "Practical and Provably Secure Release of a Secret and Exchangc of Signatures, " Advances in Cryptology EUROCRYPT '93 Proceedings, Springer Verlag, 1994, pp. 200-217.

416. 1.B. Damgard and L.R. Knudsen, "The Rreaking of the AR Hash Function, " Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 286-292.

417. I.B. Damgard and R Landrock, "Improved Bounds for the Rabin Primality Test, " 431. Cryptography and Coding III, M.J. Ganley, e d., Oxford: Clarendon Press, 1993, pp. 117-128.

418. I.B. Damgard, P. Landrock and C. Pomerance, "Average Case Error Estimates for the Strong Probablc Prime Test," Mathematics of Computation, v. 61, n. 203, Jul 1993, pp. 177-194.

419. H.E. Daniels, Jr., letter to Datapro Research Corporation regarding CCEP, 23 Dec 1985.

420. H. Davenport, The Higher Arithmetic, Dover Books, 1983.

421. G.I. Davida, "Inverse of Elements of a Galois Field," Electronics Letters, v. 8, n. 21, 19 Oct 1972, pp. 518-520.

422. G.I. Davida, "Hellman's Scheme Breaks DES in Its Basic Form," IEEE Spectrum, v. 16, n. 7, Jul 1979, p. 39.

423. G.I. Davida, "Chosen Signature Cryptanalysis of the RSA iMITJ Public Key Cryptosystem," Technical Report TR-CS-82-2, Department of EECS, University of Wis- consin, 1982.

424. G.I. Davida and G.G. Walter, "A Public Key Analog Cryptosystem," Advances in Cryptology E UR O CRYPT '8 7 Proceedings, Springer-Verlag, 1988, pp. 143-147.

425. G.I. Davida, D. Wells, and J. Kam, "A Database Encryption System with Subkeys," ACM Transactions on Database Systems,v.6,n.2,Junl981,pp.312-328.

426. D.W. Davies, "Applying the RSA Digital Signature to Electronic Mail," Computer, v. 16, n. 2, Feb 1983, pp. 55-62.

427. D.W. Davies, "Some Regular Properties of the DES," Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp. 89-96.

428. D.W. Davics, "A Message Authentication Algorithm Suitable for a Mainframe Computer," Advances in Cryptology: Proceed- ings of Crypto 82, Springer-Verlag, 1985, pp. 393 400.

429. D.W. Davies and S. Murphy, "Pairs and Triplets of DES S-boxes, " Cryptologia, v. 8, n. 1, 1995, pp. 1-25.

430. D.W. Davies and G.I.P. Parkin, "The Average Size of thc Key Stream in Output Feedback Encipherment, " Cryptography Proceedings of the Workshop of cryplograpy Burg Feuer.stein, Germany, March 29-April 2, 1982, Springer-Verlag, 1983, pp. 263-279.

431 D.W. Davies and G.I.R Parkin, "The Averag e Size of the Key Stream in Output Feedback Mode, " Advances in Cryptology: Pro ceedings of Crypto 82, Plenum Press, 1983, pp. 97-98.

432. D.W. Davies and W. L. Price, "The Applica tion of Digital Signatures Based on Public- Key Cryptosystems, " Proceedings of the Fifth International Computer Communications Conference, Oct 1980, pp. 525-530.

433. D.W. Davies and W.L. Price, "The Applica- tion of Digital Signatures Based on Public-Key Cryptosystems, " National Physical Laboratory Report DNACS 39/80, Dec 1980.

434. D.W. Davies and W.L. Price, "Digital Sig- nature An Update," Proceedings of International Conference on Computer Com- munications, Sydney, Oct 1984, North Holland: Elsevier, 1985, pp. 843-847.

435. D.W. Davies and W.L. Price, Security for Computer Networks, second edition, John Wiley & Sons, 1989.

436. M. Davio, Y. Desmedt, M. Fosseprez, R. Govaerts, J. Hulsbrosch, R Neutjens, R Piret, J - l Quisquater, J Vandewalle, and S. Wouters, "Analytical Characteristics of the Data Encryption Standard," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 171-202.

437. M. Davio, Y. Desmedt, l Goubert, F. Hoor- naert, and l -J Quisquater, "Efficient Hard- ware and Software Implementation of the DES," Advances in Cryptology: Proceed- ings of CRYPTO 84, Springer-Verlag, 1985, pp. 144 146.

438. M. Davio, Y. Desmedt, and l-l Quisquater, "Propagation Characteristics of the DES, " Advances in Cryptology: Pro- ceedings of EUROCRYPT 84, Springer- Verlag, 1985, 62- 73.

439. D. Davis, R. Ihaka, and R Fenstermacher, "Cryptographic Randomness from Air Turbulence in Disk Drives," Advances in Cryptology CRYPTO '94 Proceedings, Springer-Verlag, 1994, pp. 114 120.

440. J.A. Davis, D. B. Holdbridge, and G.l. Sim- mons, "Status Report on Factoring tat the Sandia National Laboratoriesi," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 183-215.

441. R.M. Davis, "The Data Encryption Stan- dard in Perspective," Computer Secunty and the Data Encryption Standard, National Bureau of Standards Special Pub- lication 500-27, Feb 1978.

442. E. Dawson and A. Clark, "Cryptanalysts of Universal Logic Sequences," Advances in Cryptology EUROCRYPT '93 Proceed- ings, Springer-Verlag, to appear.

443. M.H. Dawson and S.E. Tavares, "An Expanded Set of Design Criteria for Substi- tution Boxes and Their Use in Strengthen- ing DES-Like Cryptosystems, " IEEE Pacific Rim Conference on Communica- tions, Computers, and Signal Processing, Victoria, BC, Canada, 9-10 May 1991, pp. 191-195.

444. M.H. Dawson and S.E. Tavares, "An Expanded Set of S-Box Design Criteria Based on Information Theory and Its Relation to Differential-like Attacks," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 352-367.

445. C.A. Deavours, "Unicity Points in Cryptanalysis," Cryptologia, v. 1, n. 1, 1977, pp. 46-68.

446. C.A. Deavours, "The Black Chamber: A Column; How the British Broke Enigma." Cryptologia, v. 4, n. 3, lU1 1980, pp. 129- 132.

447. C.A. Deavours, "The Black Chamber: A Column; La Methode des Batons," Cryp- tologia, v. 4, n. 4, Oct 1980, pp. 240-247.

448. C.A. Deavours and L. Kruh, Machine Cryptography and Modern Cryptanalysis, Norwood MA: Artech House, 1985.

449. l.M. DeLaurentis, "A Further Weakness in the Common Modulus Protocol for the RSA Cryptosystem," Cryptologia, v. 8, n. 3, lul 1984, pp. 253-259.

450. R Delsarte, Y. Desmedt, A. Odlyzko, and P. Piret, "Fast Cryptanalysis of the Matsumoto-lmai Public-Key Scheme, " Advances in Cryptology: Proceedings of EUROCRYPT 84, Spunger-Verlag, 1985, pp. 142-149.

451. R Delsarte and R Piret, "Comment on 'Extension of RSA Cryptostructure: A Galois Approach'," Electronics Letters, v. 18, n. 13, 24 Jun 1982, pp. 582-583.

452. R. DeMillo, N. Lynch, and M. Merritt, "Cryptographic Protocols," Proceedings of the 14th Annual Symposium on the The- ory of Computing, 1982, pp. 383-400.

453. R. DeMillo and M. Merritt, "Protocols for Data Security," Computer, v. 16, n. 2, Feb 1983, pp. 39-50.

454. N. Demytko, "A New Elliptic Curve Based Analogue of RSA," Advances in Cryptolgy EUROCRYPT 93 Proceedings, Springer-Verlag, 1994, pp. 40-49.

455. D.E. Denning, "Secure Personal Comput- ing in an Insecure Network," Communi- cations of the ACM, v. 22, n. 8, Aug 1979, pp. 476-482.

456. D.E. Denning, Cryptography and Data Security, Addison-Wesley, 1982.

457. D.E. Denning, "Protecting Public Keys and Signature Keys," Computer. v. 16, n. 2, Feb 1983, pp. 27-35.

458. D.E. Denning, "Digital Signatures with RSA and Other Public-Key Cryptosys471. tems," Communications of the ACM, v. 27, n. 4, Apr 1984, pp. 388-392.

459. D.E. Denning, "The Data Encryption Standard: Fifteen Years of Public Scrutiny, " Proceedings of the Sixth Annual Com puter Security Applications Conference, IEEE Computer Society Press, 1990.

460. D.E. Denning, "The Clipper Chip: A Tech nical Summary, " unpublished manuscript, 21 Apr 1993.

461. D.E. Denning and G.M. Sacco, "Time stamps in Key Distribution Protocols, " Communications of the ACM, v. 24, n. 8, Aug 1981, pp. 533-536.

462. D.E. Denning and M. Smid, "Key Escrow ing Today," IEEE Communications Maga zine, v. 32, n. 9, Sep 1994, pp. 58-68.

463. T. Denny, B. Dodson, A.K. Lenstra, and M.S. Manasse, "On the Factorization of RSA-120," Advances in Cryptology CRYPTO 93 Proceedings, Springer-Verlag, 1994, pp. 166-174.

464. W.F. Denny, "Encryptions Using Linear and Non-Linear Codes: Implementations and Security Considerations," Ph.D. dis sertation, The Center for Advanced Com puter Studies, University of Southern Louisiana, Spring 1988.

465. Department of Defense, "Department of Defense Trusted Computer System Evalu478. ation Criteria," DOD 5200.28-STD, Dec 1985.

466. Department of State, "International Traf fic in Arms Regulations SITARS," 22 CFR 120-130, Office of Munitions Control, 479. Nov 1989.

467. Department of State, "Defense Trade Reg ulations," 22 CFR 120-130, Office of Defense Trade Controls, May 1992.

468. Department of the Treasury, "Electronic Funds and Securities Transfer Policy, " Department of the Treasury Directives Manual, Chapter TD 81, Section 80, Department of the Treasury, 16 Aug 1984.

469. Department of the Treasury, "Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Deci- sions for Federal E.F.T. Use," Department of the Treasury, 1 May 198.~.

470. Department of the Treasury, "Electronic Funds and Securitics Transfer Policy Message Authentication and Enhanced Security," Order No. 106-09, Department of the Treasury, 2 Oct 1986.

471. H. Dobbertin, "A Survey on the Construc- tion of Bent Functions," K.U. Le uven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

472. B. Dodson and A.K. Lenstra, "NFS with Four Large Primes: An Explosive Experiment," draft manuscript.

473. D. Dolev and A. Yao, "On the Security of Public-Key Protocols, " Communications of the ACM, v. 29, n. 8, Aug 1983, pp. 198-208.

474. J. Domingo-Ferrer, "Probabilistic Authentication Analysis," CARDIS 94 Proceed- ings of the First Smart Card Research and Applications Conference, Lille, France, 24-26 Oct 1994, pp. 49-60.

475. R de Rooij, "On the Security of the Schnorr Scheme Using Preprocessing, " Advances in Cryptology EUR(9CRYPT 91 I'roceed- ings, Springer-Verlag, 1991, pp. 71-80.

476. A. De Santis, G. Di Crescenzo, and G. Per- siano, "Secret Sharing and Perfect Zero Knowledge, " Advances in Cryptology CKYPTO 93 I'roceedings, Springer-Verlag, 1994, pp. 73-84.

477. A. De Santis, S. Micali, and G. Persiano, "Non-interactive Zero-Knowledge Proof Systems," Advances in Cryptology CRYPTO '87 Pro cee dings , Springer Verlag, 1988, pp. 52-72.

478. A. De Santis, S. Micali, and G. Persiano, "Non-Interactive Zero-Knowledge with Preprocessing," Advances in Cryptology CRYPTO 88 Proceedings, Springer-Verlag, 1990, pp. 269-282.

479. Y. Desmedt, "What Happened with Knapsack Cryptographic Schemes" Performance limits in Communication, Theory and P'ractice, NATO ASI Series E: Applied Sciences, v. 142, Kluwer Academic Publishers, 1988, pp. 113-134.

480. Y. Desmedt, "Subliminal-Free Authentication and Signature," Advances in Cryptology EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 23-33.

481. Y. Desmedt, "Abuses in Cryptography and How to Fight Them," Advances in Cryptology CRYPTO '8~3 Proceedings, Springer Verlag, 1990, pp.375-389.

482. Y. Desmedt and M. Burmester, "An Effi cient Zero-Knowledge Scheme for the Discrete Logarithm Based on Smooth Numbers, " Advances in Cryptology ASIA CRYPT '91 Proceedings, Springer Verlag, 1993, pp. 360-367.

483. Y. Desmedt and Y. Frankel, "Threshold 496. Cryptosystems, " Advances in Cryptol ogy CRYPTO '89 Proceedings, Springer Verlag, 1990, pp. 307-315.

484. Y. Desmedt and Y. Frankel, "Shared Gen eration of Authentication and Signatures, " Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 457 469.

485. Y. Desmedt, C. Goutier, and S. Bengio, "Special Uses and Abuses of the Fiat Shamir Passport Protocol," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 21-39.

486. Y. Desmedt and A.M. Odlykzo, "A Chosen Text Attack on the RSA Cryptosystem and Some Discrete Logarithm Problems, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 516-522.

487. Y. Dcsmedt, J.-J. Quisquater, and M. Davio, "Dependence of Output on Input in DES: Small Avalanche Characteristics," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 359-376.

488. Y. Desmedt, J. Vandewalle, and R. Go vaerts, "Critical Analysis of the Security of Knapsack Public Key Algorithms," IEEE Transactions on Information Theory, v. IT 30,n.4,Jull984,pp.601-611.

489. Y. Desmedt and M. Yung, "Weaknesses of Undeniable Signature Schemes, " Ad vances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 205-220.

490. W. Diffie, lecture at IEEE Information The ory Workshop, Ithaca, N.Y., 1977.

491. W. Diftie, "Cryptographic Technology: Fif teen Year Forecast," BNR Inc., Jan 1981.

492. W. Diffie, "The First Ten Years of Public Key Cryptography, " Proceedings of the IEEE, v 76, n. 5, May 1988, pp. 560-577.

493. W. Diffie, "Authenticated Key Exchange and Secure Interactive Communication," Proceedings of SECURICOM'90, 1990.

494. W. Diffie, "The First Ten Years of Public- Key Cryptography, " in Contemporary Cryptology: The Science of Information Integrity, G.J. Simmons. ed., IEEE Press, 1992, pp. 135-175.

495. W. Diffie and M.E. Hellman, "Multiuser Cryptographic Techniques, " Proceedings of AFIPS National Computer Conference, 1976, pp. 109-112.

496. W. Diffie and M.E. Hellman, "New Direc- tions in Cryptography, " IEEE Transactions on Information Theory, v. IT-22, n. 6, Nov 1976, pp. 644~54.

497. W. Diffie and M.E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," Computer, v. 10, n. 6, Jun 1977, pp. 74-84.

498. W. Diffie and M.E. Hellman, "Privacy and Authentication: An Introduction to Cryp- tography," Proceedings of the IEEE, v. 67, n. 3, Mar 1979, pp. 397-427.

499. W. Diffie, L. Strawczynski, B. O'Higgins, and D. Steer, "An ISDN Secure Telephone Unit," Proceedings of the National Tele- communications Forum, v 41, n. 1, 1987, pp. 473 477.

500. W. Diffie, RC. van Oorschot, and M.J. Wiener, "Authentication and Authenti- cated Key Exchanges," Designs, Codes and Cryptography, v. 2, 1992, 107-125.

501. C. Ding, "The Differential Cryptanalysis and Design of Natural Stream Ciphers," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer- Verlag, 1994, pp. 101-115.

502. C. Ding, G. Xiao, and W. Shan, The Stahility Theory of Stream Ciphers, Springer- Verlag, 1991.

503. A. Di Porto and W. Wolfewicz, "VINO: A Block Cipher Including Variable Permuta- tions, " Fast Software Encryption, Cambridge Secunty Workshop l'roceedings, Springer- Verlag, 1994, pp. 205-210.

504. B. Dixon and A.K. Lenstra, "Factoring Inte- gers Using SIMD Sieves, " Advances in Cryptology E UR O CRYPT '93 Proceed- ings, Springer-Verlag, 1994, pp. 28-39.

505. J.D. Dixon, "Factorization and Primality Tests," American Mathematical Monthly, v.91,n.6, 1984,pp.333-352.

506. D. Dolev and A. Yao, "On the Security of Public Key Protocols," Proceedings ol the 22nd Annual Symposium on the Founda- tions of Computer Science, 1981, pp. 350- 357.

507. L.X. Duan and C.C. Nian, "Modified Lu- Lee Cryptosystems," Electronics Letters, v. 25, n. 13, 22 Jun 1989, p. 826.

508. R. Durstenfeld, "Algorithm 235: Random Permutation, " Communications of the ACM, v. 7, n. 7, Jul 1964, p. 420.

509. S. Dusse and B. Kaliski, Jr., "A Cryptographic Library for the Motorola DSP56000, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer- Verlag, 1991, pp. 230-244.

510. C. Dwork and L. Stockmeyer, "Zero- Knowledge with Finite State Verifiers, " Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 71-75.

511. D.E. Eastlake, S.D. Crocker, and J.I. Schiller, "Randomness Requirements for Security," RFC 1750, Dec 1994.

512. H. Eberle, "A High-Speed DES Implementation for Network Applications, " Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, pp. 521-539.

513. T. Edwards, "Implementing Electronic Poker: A Practical Exercise in Zero Knowledge Interactive Proofs, " Master's thesis, Department of Computer Science, University of Kentucky, May 1994.

514. W.F. Ehrsam, C.H.W. Meyer, R.L. Powers, J L. Smith, and W.L. Tuchman, "Product Block Cipher for Data Security, " U.S. Patent #3,962,539, 8 Jun 1976.

515. W.F. Ehrsam, C.H.W. Meyer, and W.L. Tuchman, "A Cryptographic Key Manage ment Scheme for Implementing the Data Encryption Standard," IBM Systems lour nal, v. 17, n. 2, 1978, pp. 106-125.

516. R. Eier and H. Lagger, "Trapdoors in Knap sack Cryptosystems, " Lecture Notes in Computer Science 149; Cryptography Proceedings, Burg Feuerstein 1982, Springer-Verlag, 1983, pp. 316-322.

517. A.K. Ekert, "Quantum Cryptography Based on Bell's Theorem, " Physical Review Letters, v. 67, n. 6, Aug 1991, pp. 529. 661-663.

518. T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, " Advances in Cryptology: Proceedings of CRYPTO 84, Springer" Verlag, 1985, pp. 1~18.

519. T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," IEEE Transactions on Infor- mation Theory, v. IT-31, n. 4, 1985, pp. 469-472.

520. T. ElGamal, "On Computing Logarithms Over Finite Fields," Advances in Cryptol- ogy CR YP TO '85 Pro cee dings, Springe r - Verlag, 1986, pp. 396 402.

521. T. ElGamal and B. Kaliski, letter to the edi- tor regarding LUC, Dr. Dobb's /ournal, v. 18,n.5,Mayl993,p. 10.

522. T. Eng and T. Okamoto, "Single-Term Divisible Electronic Coins," Advances in Cryptology EUROCRYPT '94 Proceed- ings, Springer-Verlag, 1995, to appear.

523. M.H. Er, D.J. Wong, A.A. Sethu, and K.S. Ngeow, "Design and Implementation of RSA Cryptosystem Using Multiple DSP Chips," 1991 IEEE International Sympo- sium on Circuits and Systems, v. 1, Singa- pore, 11-14 Jun 1991, pp. 49-52.

524. D. Estes, L.M. Adleman, K. Konpella, K.S. McCurley, and G.L. Miller, "Breaking the Ong-Schnorr-Shamir Signature Schemes for Quadratic Number Fields," Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 3-13.

525. ETEBAC, "Echanges Telematiques Entre Les Banques et Leurs Clients," Standard ETEBAC 5, Comite Fran,cais d'Organisa- tion et de Normalisation Bancaires, Apr 1989. IIn French.~

526. A. Evans, W. Kantrowitz, and E. Weiss, "A User Identification Scheme Not Requiring Secrecy in the Computer," Communica- tions of the ACM, v. 17, n. 8, Aug 1974, pp. 437-472.

527. S. Even and O. Goldreich, "DES-Like Functions Can Generate the Alternating Group, " IEEE Transactions on Informa- tion Theory, v. IT-29, n. 6, Nov 1983, pp. 863-865.

528. S. Even and O. Goldreich, "On the Power of Cascade Ciphers," ACM Transactions on Computer Systems, v. 3, n. 2, May 1985, pp. 108-116.

529. S. Even, O. Goldreich, and A. Lempel, "A Randomizing Protocol for Signing Con- tracts," Communications of the ACM, v. 28, n. 6, [un 1985, pp. 637-647.

530. S. Even and Y. Yacobi, "Cryptography and NP-Completeness," I'roceedings of the 7th International CoRoquium on Automata, Languages, and Programming, Springer -Verlag, 1980, pp. 195-207.

531. H.-H. Evertse, "Linear Structures in Block Ciphers, " Advances in Cryptology EUROCRYPT '87 Proceedings, Springer Verlag, 1988, pp. 249-266.

532. R Fahn and M.J.B. Robshaw, "Results from the RSA Factoring Challenge," Technical Report TR-501, Version 1.3, RSA Laboratories, Jan 1995.

533. R.C. Fairfield, A. Matusevich, and J. Plany, "An LSI Digital Encryption Processor (DEP)," Advances in Cryptology: Proceed ings of CRYPTO 84, Springer-Verlag 1985, pp. 115-143.

534. R.C. Fairfield, A. Matusevich, and J. Plany, "An LSI Digital Encryption Processor (DEPJ," IEEE Communications, v. 23. n. 7, Jul 1985, pp. 30-41.

535. R.C. Fairfield, R.L. Mortenson, and K.B. Koulthart, "An LSI Random Number Gen erator (RNG~," Advances in Cryptology: Proceedings of CRYPTO 84, Springer Verlag, 1985, pp. 203-230.

536. "International Business Machines Corp. License Under Patents," Federal Register, v. 40, n. 52, 17 Mar 1975, p. 12067.

537. "Solicitation for Public Key Cryptographic Algorithms," Federal Register, v. 47, n. 126, 30 Jun 1982, p. 28445.

538. "Proposed Federal Information Processing Standard for Digital Signature Standard (DSSi," Federal Register, v. 56, n. 169, 30 Aug 1991, pp. 42980-42982.

539. "Proposed Federal Information Processing Standard for Secure Hash Standard," Fed eral Register, v. 57, n. 21, 31 Jan 1992, pp. 3747-3749.

540. "Proposed Reaffirmation of Federal Infor mation Processing Standard (FIPS) 46-1, Data Encryption Standard (DES)," Federal Register, v. 57, n. 177, 11 Sep 1992, p. 41727.

541. "Notice of Proposal for Grant of Exclusive Patent License," [ederal Register, v. 58, n. 108, 8 Jun 1993, pp. 23105-23106.

542. "Approval of Federal Information Process ing Standards Publication 186, Digital Sig nature Standard (DSS)," Federal Register, v. 58, n. 96, 19 May 1994, pp. 26208-26211.

543. "Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard, " Federal Register, v. 59, n. 131, 11 Jul 1994, pp. 35317-35318.

544. U. Feige, A. Fiat, and A. Shamir, "Zero Knowledge Proofs of Identity," Proceed- ings of the lPth Annual ACM Symposium on the Theory of Computing, 1987, pp. 210-217.

545. U. Feige, A. Fiat, and A. Shamir, "Zero Knowledge Proofs of Identity," Journal of Cryptology v. 1, n. 2, 1988, pp. 77-94.

546. U. Feige and A. Shamir, "Zero Knowledge Proofs of Knowledge in Two Rounds, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 526-544.

547. J. Feigenbaum, "Encrypting Problem Instances, or, ..., Can You Take Advan- tage of Someone Without Having to Trust Him, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 477-488.

548. J. Feigenbaum, "Overview of Interactive Proof Systems and Zero-Knowledge, " in Contemporary Cryptology: The Science of Information Integrity, G.J. Simmons, ed., IEEE Press, 1992, pp. 423 439.

549. J. Feigenbaum, M.Y. Liberman, E. Grosse, and J.A. Reeds, "Cryptographic Protection of Membership Lists," Newsletter of the International Association of Cryptologic Research, v. 9, 1992, pp. 16-20.

550. J. Feigenbaum, M.Y. Liverman, and R.N. Wright, "Cryptographic Protection of Databases and Software, " Distnbuted Computing and Cryptography, 1 Feigen- baum and M. Merritt, eds., American Mathematical Society, 1991, pp. 161-172.

551. H. Feistel, "Cryptographic Coding for Data-Bank Privacy," RC 2827, Yorktown Heights, NY: IBM Research, Mar 1970.

552. H. Feistel, "Cryptography and Computer Privacy, " Scientific American, v. 228, n. 5, May 1973, pp. 15-23.

553. H. Feistel, "Block Cipher Cryptographic System," U.S. Patent #3,798,359, 19 Mar 1974.

554. H. Feistel, "Step Code Ciphering System," U.S. Patent #3,798,360, 19 Mar 1974.

555. H. Feistel, "Centralized Verification Sys- tem," U.S. Patent #3,798,605, 19 Mar 1974.

556. H. Feistel, W.A. Notz, and J.L. Smith, "Cryptographic Techniques for Machine to Machine Data Communications," RC 3663, Yorktown Heights, N.Y.: IBM Research, Dec 1971.

557. H. Feistel, W.A. Notz, and J.L. Smith, "Some Cryptographic Techniques for Machine to Machine Data Communica tions," Proceedings of the IEEE, v. 63, n. 11, Nov 1975, pp. 1545-1554.

558. R Feldman, "A Practical Scheme for Non interactive Verifiable Secret Sharing," Proceedings of the 28th Annual Symposium on the Foundations of Computer Science, 1987, pp. 427 437.

559. R.A. Feldman, "Fast Spectral Test for Mea suring Nunrandomness and the DES, " Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 243-254.

560. R.A. Feldman, "A New Spectral Test for Nonrandomness and the DES, " IEEE 573. Transactions on Software Engineering, v. 16, n. 3, Mar 1990, pp. 261-267.

561. D.C. Feldmeier and RR. Karn, "UNIX Password Security Ten Years Later, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 44-63.

562. H. Fell and W. Diffie, "Analysis of a Public Key Approach Based on Polynomial Sub stitution, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 427-437.

563. N.T. Ferguson, "Single Term Off-Line Coins," Report CS-R9318, Computer Sci577. ence/Department of Algorithms and Architecture, CWI, Mar 1993.

564. N.T. Ferguson, "Single Term Off-Line Coins," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 318-328.

565. N.T. Ferguson, "Extensions of Single-term Coins," Advances in Cryptology 579. CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 292-301.

566. A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Idcntifica tion and Signature Problems," Advances in Cryptology C RYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 186-194.

567. A. Fiat and A. Shamir, "Unforgeable Proofs of Identity," Proceedings of sec uricom 87, Paris, 1987, pp. 147-15~3.

568. P. Finch, "A Study of the Blowfish Encryp tion Algorithm," Ph.D. dissertation, Department of Computer Science, City University of New York Graduate School and University Center, Feb 1995.

569. R. Flynn and A.S. Campasano, "Data Dependent Keys for Selective Encryption Terminal," Proceedings of NCC, vol. 47, AFIPS Press, 1978, pp. 1127-1129.

570. R.H. Follett, letter to NIST regarding DSS, 25 Nov 1991.

571. R. Forre, "The Strict Avalanche Criterion: Spectral Properties and an Extended D efi- nition, " Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 450-468.

572. R. Forre, "A Fast Correlation Attack or Nonlinearity Feedforward Filtered Shift Register Sequences, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 568-595.

573. S. Fortune and M. Merritt, "Poker Protocols," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 454-464.

574. R.B. Fougner, "Public Key Standards and Licenses," RFC 1170, Jan 1991.

575. Y. Frankel and M. Yung, "Escrowed Encryption Systems Visited: Threats. Attacks, Analysis and Designs," Advances in Cryptology CKYPTO '95 Proceedings'. Springer-Verlag, 1995, to appear.

576. W.F. Friedman, Methods for the Solution of Running-Key Ciphers, Riverbank Publication No. 16, Riverbank Labs, 1918.

577. W.F. Friedman, The Index of Coincidence and Its Applications in Cryptography, Riverbank Publication No. 22, Rivcrhank Labs, 1920. Reprinted by Aegean Park Press, 1987.

578. W.F. Friedman, Elements of Cryptanalysis, Laguna Hills, CA: Aegean Park Press, 1976.

579. W.F. Friedman, "Cryptology," Encyclopedia Britannica, v. 6, pp. 844-851, 1967.

580. A.M. Frieze, J. Hastad, R. Kannan, J.C. Lagarias, and A. Shamir, "Reconstructing Truncated Integer Variables Satisfying Linear Congru enccs," SIAM Journal on Computing, v. 17, n. 2, Apr 1988, pp. 262-280.

581. A.M. Frieze, R. Kannan, and J.C. Lagarias, "Linear Congruential Generators loo not Produce Random Sequences," Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, 1984, pp. 480 484.

582. E. Fujiaski and T. Okamoto, "On Comparison of Practical Digitial Signature Schemes," Proceedings of the l992 Symposium on Cryptography and Information Security (SCIS 92), Tateshina, Japan, 2 4 Apr 1994, pp. lA.1-12.

583. A. Fujioka, T. Okamoto, and S. Miyaguchi, "ESIGN: An Efficient Digital Signature Implementation for Smart Cards, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 446 457.

584. A. Fujioka, T. Okamoto, and K. Ohta, "Interactive Bi-Proof Systems and Undeniable Signature Schemes," Advances in Cryptology EU ROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 243-256.

585. A. Fujioka, T. Okamoto, and K. Ohta, "A Practical Secret Voting Scheme for Large Scale Elections, " Advances in Cryptology AUSCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 244-251.

586. K. Gaardner and E. Snekkenes, "Applying a Formal Analysis Technique to the CCITT X.509 Strong Two-Way Authentication Protocol," lournal of Cryptology v. 3,n.2, l991,pp.81-98.

587. H.E Gaines, Cryptanalysis, American Photographic Press, 1937. Reprintcd by Dover Publications, 1956.

588. J. Gait, "A New Nonlinear Pseudorandom Number Generator, " IEEE Transactions on Software Engineering, v. SE-3, n. 5, Sep 1977, pp. 359-363.

589. J. Gait, "Short Cycling in the Kravitz-Reed Public Key Encryption System," Electron- ics Letters, v. 18, n. 16, 5 Aug 1982, pp. 706-707.

590. Z. Galil, S. Haber, and M. Yung, "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryp- tosystems," Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 360-371.

591. Z. Galil, S. Haber, and M. Yung, "Crypto- graphic Computation: Secure Fault- Tolerant Protocols and the Public-Key Model, " Advances in Cryptology CRYPTO '87Proceedings, Springer-Verlag, 1988, pp. 135-155.

592. Z. Galil, S. Haber, and M. Yung, "Mini- mum-Knowledge Interactive Proofs for Decision Problems, " SIAM lournal on Computing, v. 18, n. 4, 1989, pp. 711-739.

593. R.G. Gallager, Information Theory and Reliable Communications, New York:John Wiley & Sons, 1968.

594. P. Gallay and E. Depret, "A Cryptography Microprocessor," 1988 IEEE International Solid-State Circuits Conference Digest of Technical Papers, 1988, pp. 148-149.

595. R.A. Games, "There are no de Bruijn Sequences of Span n with Complexity 2n + n + 1," Journal of Combinatorical Theory, Series A, v. 34, n. 2, Mar 1983, pp. 248-251.

596. R.A. Games and A.H. Chan, "A Fast Algo- rithm for Determining the Complexity of a Binary Sequence with 2n,'t IEEE Transactions on Information Theory, v. IT-29, n. 1, Jan 1983, pp. 144-146.

597. R.A. Games, A.H. Chan, and E.L. Key, "On the Complexity of de Brui jn Sequences," ournal of Combinatorical Theory, Series A, v. 33, n. 1, Nov 1982, pp. 233-246.

598. S.H. Gao and G.L. Mullen, "Dickson Polynomials and Irreducible Polynomials over Finite Fields," Journal of Number Theory, v. 49, n. 1, Oct 1994, pp. 18-132.

599. M. Gardner, "A New Kind of Cipher That Would Take Millions of Years to Break," Scientific American, v. 237, n. 8, Aug 1977, pp. 120-124.

600. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.

601. S.L. Garfinkel, POP: Pretty Good Privacy, Sebastopol, CA: O'Reilly and Associates, 1995.

602. C.W. Gardiner, "Distributed Public Key Certificate Management," Proceedings of the Privacy and Security Research Group 1993 Workshop on Network and Distributed System Security, The Internet Society, 1993, pp. 69-73.

603. G. Garon and R. Outerbridge, "DES Watch: An Examination of the Sufficiency of the Data Encryption Standard for Finan- cial Institution Information Security in the 1990's," Cryptologia, v. 15, n. 3, Jul 1991, pp. 177-193.

604. M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson, "The Digital Distributed Systems Security Architecture," Proceedings of the 12th National Computer Seezznty Conference, NIST, 1989, pp. 305-319.

605. J. von zur Gathen, D. Kozen, and S. Lan- dau, "Functional Decomposition of Polynomials, " Proceedings of the 28tl1 IEEE Symposium on the Poundations of Computer Science, IEEE Press, 1987, pp. 127617. 131.

606. RR. Geffe, "How to Protect Data With Ciphers That are Really Hard to Break," Electronics, v. 46, n. 1, Jan 1973, pp. 99-101.

607. D.K. Gifford, D. Heitmann, D.A. Segal, R.G. Cote, K. Tanacea, and D.E. Burmas- ter, "Boston Community Information Sys- tem 1986 Experimental Test Results, " MIT/LCS/TR-397, MIT Laboratory for Computer Science, Aug 1987.

608. D.K. Gifford, J.M. Lucassen, and S.T. Berlin, "The Application of Digital Broadcast Communication to Large Scale Information Systems, " IEEE Journal on Selected Areas in Communications, v. 3, n. 3, May 1985, pp. 457-467.

609. D.K. Gifford and D.A. Segal, "Boston Community Information System 1987-1988 Experimental Test Results," MIT/LCS/ TR-422, MIT Laboratory for Computer Science, May 1989.

610. H. Gilbert and G. Chase, "A Statistical Attack on the Feal-8 Cryptosystem, " Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 22-33

611. H. Gilbert and R Chauvaud, "A Chosen Plaintext Attack of the 16-Round Khufu Cryptosystem," Advances in Cryptol- ogy CRYPTO '94 Proceedings, Springer-Verlag, 1994, pp. 259-268.

612. M. Girault, "Hash-Functions Using Mod- ulo-N Operations," Advances in Cryptology EUROCRYPT '87 Proceedings, Springer-Verlag, 1988, pp. 217-226.

613. J. Gleick, "A New Approach to Protecting Secrets is Discovered, " The New York Times, 18 Feb 1987, pp. C1 and C3.

614. J.-M. Goethals and C. Couvreur, "A Crypt analytic Attack on the Lu-Lee Public-Key Cryptosystem, " Philips lournal of Re search, v. 35, 1980, pp. 301-306.

615. O. Goldreich, "A Uniform-Complexity Treatment of Encryption and Zero Knowledge, Journal of Cryptology, v. 6, n. 1, 1993, pp. 21-53.

616. O. Goldreich and H. Krawczyk, "On the Composition of Zero Knowledge Proof Systems," Proceedings on the 17th Inter national Colloquium on Automata, Languages, and Programming, Springer" Verlag, 1990, pp. 268-282.

617. O. Goldreich and E. Kushilevitz, "A Perfect Zero-Knowledge Proof for a Problem Equivalent to Discrete Logarithm, " Advances in Cryptology CRYPTO '88 I'roceedings, Springer-Verlag, 1990, pp. 58-70.

618. O. Goldreich and E. Kushilevitz, "A Per- fect Zero-Knowledge Proof for a Problem Equivalent to Discrete Logarithm," lour- nal of Cryptology, v. 6, n. 2, law, pp. 97-116.

619. O. Goldreich, S. Micali, and A. Wigderson, "Proofs That Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design," Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science, 1 986, pp. 174-187.

620. O. Goldreich, S. Micali, and A. Wigderson, "How to Prove All NP Statements in Zero Knowledge and a Methodology of Cryptographic Protocol Design, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 171-185.

621. O. Goldreich, S. Micali, and A. Wigderson, "How to Play Any Mental Game," Pro- ceedings of the 19th ACM Symposium on the Theory of Computing, 1987, pp. 218-229.

622. O. Goldreich, S. Micali, and A. Wigdcrson, "Proofs That Yield Nothing but Their Validity and a Methodology of Crypto- graphic Protocol Design," Journal of the ACM, v. 38, n. 1, Jul 1991, pp. 691-729.

623. S. Goldwasser and J. Kilian, "Almost All Primes Can Be Quickly Certified, " Pro- ceedings of the 18th ACM Symposium on the Theory of Computing, 1986, pp. 316- 329.

624. S. Goldwasser and S. Micali, "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information," Proceedings of the 14th ACM Symposium on the Theory of Computing, 1982, pp. 270-299.

625. S. Goldwasser and S. Micali, "Probabilistic Encryption," Journal of Computer and System Sciences. v. 28, n. 2, Apr 1984, pp. 270-299.

626. S. Goldwasscr, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interac- tive Proof Systems, " Proceedings of the 17th ACM Symposium on Theory of Com- puting, 1985, pp. 291-304.

627. S.Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interac tive Proof Systems, " SIAM lournal on Computing, v. 18, n. 1, Feb 1989, pp. 186 208.

628. S. Goldwasser, S. Micali, and R.L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," SIAM lournal on Computing, v. 17,n.2,Aprl988,pp.281-308.

629. S. Goldwasser, S. Micali, and A.C. Yao, "On Signatures and Authentication," Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp. 211-215.

630. J.D. Golic, "On the Linear Complexity of Functions of Periodic GFIq~ Sequences," IEEE Transactions on Information Theory, v. IT-35, n. 1. Jan 1989, pp. 69-75.

631. J.D. Golic, "Linear Cryptanalysis of Stream Ciphers," K.U. Le uven Workshop on Cryptographic Algorithms, Springer Verlag, 1995, pp. 262-282.

632. J.D. Golic, "Towards Fast Correlation Attacks on Irregularly Clocked Shift Registers," Advances in Cryptology EURO CRYPT '95 Proceedings, Springer-Verlag, 1995, to appear.

633. J.D. Golic and M.J. Mihajlevic, "A Gener alized Correlation Attack on a Class of Stream Ciphers Based on the Levenshtein Distance, " journal of Cryptology, v. 3, n.3, 1991, pp. 201-212.

634. J.D. Golic and L. O'Connor, "Embedding and Probabilistic Correlation Attacks on Clock-Controlled Shift Registers," Advances in Cryptology EUROCRYPT '94 I'roceedings, Springer-Verlag, 1995, to appear.

635. R. Golliver, A.K. Lenstra, K.S. McCurley, "Lattice Sieving and Trial Division," Pro ceedings of the Algorithmic Number Theory Symposium, Cornell, 1994, to appear.

636. D. Gollmann, "Kaskadenschaltungen takt gesteuerter Schicberegister als Pseudozu fallszahlengencratoren," Ph.D. disserta tion, Universitat Linz, 1983. (In German).

637. D. Gollmann, "Pseudo Random Properties of Cascade Connections of Clock Con trolled Shift Registers," Advances in Cryp tology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, pp. 93-98.

638. D. Gollmann, "Correlation Analysis of Cascaded Sequences," Cryptography and Coding, H.J. Beker and F.C. Piper, eds., Oxford: Clarendon Press, 1989, pp. 289-297.

639. D. Gollmann, "Transformation Matrices of Clock-Controlled Shift Registers, " Cryptography and Coding 111, M.J. Ganley, e d., Oxford: Clarendon Press, 1993, pp. 197-210.

640. D. Gollmann and W.G. Chambers, "Lock-In Effect in Cascades of Clock-Controlled Shift-Registers, " Advances in Cryptology EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 331-343.

641. D. Gollmann and WG. Chambers, "Clock-Controlled Shift Registers: A Review, " IEEE lournal on Selected Areas in Communications, v. 7, n. 4, May 1989, pp. 525-533.

642. D. Gollmann and W.G. Chambers, "A Cryptanalysis of Step~-cascades," Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 680-687.

643. S.W. Golomb, Shift Register Sequences, San Francisco: Holden-Day, 1967. (Reprinted by Aegean Park Press, 1982.

644. L. Gong, "A Security Risk of Depending on Synchronized Clocks, " Operating Systems Review, v. 26, n. 1, Jan 1992, pp. 49-53.

645. L. Gong, R. Needham, and R. Yahalom, "Reasoning About Belief in Cryptographic Protocols," Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, 1991, pp. 234-248.

646. R.M. Goodman and A.J. McAuley, "A New Trapdoor Knapsack Public Key Cryptosystem," Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, pp. 150-158.

647. R.M. Goodman and A.J. McAuley, "A New Trapdoor Knapsack Public Key Cryptosystem," IKE Proceedings, v. 132, pt. E, n. 6, Nov 1985, pp. 289-292.

648. D.M. Gordon, "Discrete Logarithms Using the Number Field Sieve," Preprint, 28 Mar 1991.

649. D.M. Gordon and K.S. McCurley, "Computation of Discrete Logarithms in Fields of Characteristic Two," presented at the rump session of CRYPTO'91, Aug 1991.

650. D.M. Gordon and K.S. McCurley, "Massively Parallel Computation of Discrete Logarithms, " Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 661. 1993, pp. 312-323.

651. J.A. Gordon, "Strong Primes are Easy to Find," Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, pp. 216-223.

652. J.A. Gordon, "Very Simple Method to Find the Minimal Polynomial of an Arbitrary Non-Zero Element of a Finite Field, " Electronics Letters, v. 12, n. 25, 9 Dec 1976, pp. 663-664.

653. J.A. Gordon and R. Retkin, "Are Big S- Boxes Best7" Cryptograph y Proceedings of the Workshop on Cryptography, Burg Feuerstein, Germany, March 29-April 2, 1982, Springer-Verlag, 1983, pp. 257-262.

654. M. Goresky and A. Klapper, "Feedback Registers Based on Ramified Extension of the 2-adic Numbers," Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

655. GOST, Gosudarstvennyi Standard 28147-89, "Cryptographic Protection for Data Processing Systems," Government Committee of the USSR for Standards, 1989. (in Russian.}

656. GOST R 34.10-94, Gosudarstvennyi Standard of Russian Federation, "Information technology. Cryptographic Data Security. Produce and check procedures of Electronic

Digital Signature based on Asymmetric Cryptographic Algorithm. " Government Committee of the Russia for Standards, 1994. (In Russian.)

657. GOST R 34.11-94, Gosudarstvennyi Standard of Russian Federation, " Information technology. Cryptographic Data Security. Hashing function." Government Committee of the Russia for Standards, 1994. (In Russian.)

658. R. Gottfert and H. Niederreiter, "On the Linear Complexity of Products of Shift-Register Sequences," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 151-158.

659. R. Gottfert and H. Niederreiter, "A General Lower Bound for the Linear Complexity of the Product of Shift-Register Sequences, " Advances in Cryptology EUROCRYPT '94 Proceedings, Springer- Verlag, 1995, to appear.

660. J. van de Graaf and R. Peralta, "A Simple and Secure Way to Show the Validity of Your Public Key," Advances in Cryptology CRYPTO '87 Proceedings, Springer- Verlag, 1988, pp. 128-134.

661. J. Grollman and A.L. Selman, "Complexity Measures for Public-Key Cryptosystems," Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science, 1984, pp. 495-503

662. GSA Federal Standard 1026, "Telecommunications: General Security Requirements for Equipment Using the Data Encryption Standard, " General Services Administration, Apr 1982.

663. GSA Federal Standard 1027, "Telecommunications: Interoperability and Security Requirements for Use of the Data Encryption Standard in the Physical and Data Link Layers of Data Communications, " General Services Administration, Jan 1983.

664. GSA Federal Standard 1028, "Intcroperability and Security Requirements for Use of the Data Encryption Standard with CCITT Group 3 Facsimile Equipment, " General Services Administration, Apr 1985.

665. R Guam, "Cellular Automaton Public Key Cryptosystems," Complex Systems, v. 1, 1987, pp. 51-56.

666. H. Guan, "An Analysis of the Finite Automata Public Key Algorithm, " CHINACRYPT'94, Xidian, China, 11-15 Nov 1994, pp. 120-126. (In Chinese)

667. G. Guanella, "Means for and Method for Secret Signalling," U.S. Patent #2,405,500, 6 Aug 1946.

668. M. Gude, "Concept for a High-Performance Random Number Generator Based on Physical Random Phenomena," Frequenz, v. 39, 1985, pp. 187-190.

669. M. Gude, "Ein quasi-idealer Gleichverteil-ungsgenerator basierend auf physikalischen Zufallsphinomenen," Ph.D. dissertation, Aachen University of Technology, 1987. (In German.)

670. L.C. Guillou and J.-J. Quisquater, "A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory," Advances in Cryptology EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 123-128.

671. L.C. Guillou and J. Quisquater, "A 'Paradoxical' Identity-Based Signature Scheme Resulting from Zero-Knowledge," Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 216- 231.

672. L.C. Guillou, M. Ugon, and J. -J. Quisquater, "The Smart Card: A Standardized Security Device Dedicated to Public Cryptology," contemporary Cryptology: The Science of Information Integrity G. Simmons, ed., IEEE Press, 1992, pp. 561-613.

673. C.G. Gunther, "Alternating Step Generators Controlled by de Bruijn Sequences," Advances in Cryptology EUROCRYPT '87 Proceedings, Springer-Verlag, 1988, pp. 5-14.

674. C.G. Gunther, "An Identity-based Key-exchange Protocol, " Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 29-37.

675. H. Gustafson, E. Dawson, and B. Caelli, "Comparison of Block Ciphers, " Advances in Cryptology AUSCRYPT '90 Proceed- ings, Springer-Verlag, 1990, pp. 208-220.

676. P. Gutmann, personal communication, 1993.

677. H. Gutowitz, "A Cellular Automaton Cryptosystem: Specification and Call for Attack," unpublished manuscript, Aug 1992.

678. H. Gutowitz, "Method and Apparatus for Encryption, Decryption, and Authentication Using Dynamical Systems," U.S. Patent #5,365,589, 15 Nov 1994.

679. H. Gutowitz, "Cryptography with Dynamical Systems, " Cellular Automata and Cooperative Phenomenon, Kluwer Academic Press, 1993.

680. R.K. Guy, "How to Factor a Number, " Fifth Manitoba Conference on Numeral Mathematics Congressus Numerantium, v. 16, 1976, pp. 49-89.

681. R.K. Guy, Unsolved Problems in Number Theory, Springer-Verlag, 1981.

682. S. Haber and W.S. Stornetta, "How to Time-Stamp a Digital Document, " Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 437-455.

683. S. Haber and W.S. Stornetta, "How to Time-Stamp a Digital Document, " journal of Cryptology, v. 3, n. 2, 1991, pp. 99-112.

684. S. Haber and W.S. Stornetta, "Digital Document Time-Stamping with Catenate Certificate," U.S. Patent #5,136,646, 4 Aug 1992.

685. S. Haber and W.S. Stornetta, "Method for Secure Time-Stamping of Digital Documents," U.S. Patent #5,136,647, 4 Aug 1992.

686. S. Haber and W.S. Stornetta, "Method of Extending the Validity of a Cryptographic Certificate," U.S. Patent #5,373,561, 13 Dec 1994.

687. T. Habutsu, Y. Nishio, I. Sasase, and S. Mori, "A Secret Key Cryptosystem by Iterating a Chaotic Map," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. E73, n. 7,1ul 1990, pp. 1041-1044.

688. T. Habutsu, Y. Nishio, I. Sasase, and S. Mori, "A Secret Key Cryptosystem by Iterating a Chaotic Map, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 127-140.

689. S. Hada and H. Tanaka, "An Improvement Scheme of DES against Differential Cryptanalysis," Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94), Lake Biwa, Japan, 27-29 Jan 1994, pp 14A. I-l l. fin Japanese. l

690. B.C.W. Hagelin, "The Story of the Hagelin Cryptos," Cryptologia, v. 18, n.3, Jul 1994, pp. 204-242.

691. T. Hansen and G.L. Mullen, "Primitive Polynomials over Finite Fields," Mathematics of Computation, v. 59, n. 200, Oct 1992, pp. 639-643.

692. S. Harada and S. Kasahara, "An ID-Based Key Sharing Scheme Without Preliminary Communication," IEICE Japan, Technical Report, ISEC89-38, 1989. (In Japanese).

693. S. Harari, "A Correlation Cryptographic Scheme," EUROCODE '90 International Symposium on Coding Theory, Springer-Verlag, 1991, pp. 180-192.

694. T. Hardjono and J. Seberry, "Authentication via Multi-Service Tickets in the Kuperee Server, " Computer Security ESORICS 94, Springer-Verlag, 1994, pp. 144 160.

695. L. Harn and T. Kiesler, "New Scheme for Digital Multisignatures," Electronics Letters, v. 25, n. 15, 20 Jul 198Y, pp. 1002- 1003.

696. L. Harn and T. Kiesler, "Improved Rabin's Scheme with High Efficiency, " Electronics Letters, v. 25, n. 15, 20 Jul 1989, p. 1016.

697. L. Harn and T. Kiesler, "Two New Efficient Cryptosystems Based on Rabin's Scheme, " Fifth Annual Computer Secunty Applications Conference, IEEE Computer Society Press, 1990, pp. 263-270.

698. L. Harn and D.-C. Wang "Cryptanalysts and Modification of Digital Signature Scheme Based on Error-Correeting Codes, " Electronics Letters, v. 28. n. 2, 10 Jan 1992, p. 157-159.

699. L. Harn and Y. Xu, "Design of Generalized ElGamal Type Digital Signature Schemes Based on Discrete Logarithm, " Electronics Letters, v. 30, n. 24. 24 Nov 1994, p. 2025-2026.

700. L. Harn and S. Yang, "Group-Oriented Undeniable Signature Schemes without the Assistance of a Mutually Trusted Party," Advances in Cryptology AUSCRYPT '92 Proceedings, Springer- Verlag, 1993, pp. 133-142.

701. G. Harper, A. Menezes, and S. Vanstone, "Public-Key Cryptosystems with Very Small Key Lengths," Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag 1993, pp. 163-173.

702. C. Harpes, "Notes on High Order Differen- tial Cryptanalysis of DES, " internal report, Signal and Information Processing Labora- tory, Swiss Federal Institute of Technology, Aug 1993.

703. G.W. Hart, "To Decode Short Cryptograms," Communications of the ACM, 717. v. 37, n. 9, Sep 1994, pp. 102-108.

704. J. Hastad, "On Using RSA with Low Exponent in a Public Key Network," Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag 1986, pp. 403-408.

705 1 Hastad and A. Shamir, "The Cryptographic Secunty of Truncated Linearly Related Variables, " Proceedings of the 1 7th Annual ACM Symposium on the Theory of Computing, 1985, pp. 356-362.

706. R.C. Hauser and E.S. Lee, "Verification and Modelling of Authentication Protocols, " ESORICS 92, Proceedings of the Second European Symposium on Research in Computer Security, Springer-Verlag 1992, pp. 131-154.

707. B. Hayes, "Anonymous One-Time Signatures and Flexible Untraceable Electronic Cash," Advances in Cryptology AUSCRYPT '90 Proceedings, Springer Verlag, 1990, pp. 294-305.

708. D.K. He, "LUC Public Key Cryptosystem and its Properties," CHINACRYPT '94, Xidian, China, 11-15 Nov 1994, pp. 60-69. (In Chinese.)

709. J. He and T. Kiesler, "Enhancing the Security of ElGamal's Signature Scheme," IKE Proceedings on Computers and Digital Techniques, v. 141, n.3, 1994. pp.193-195.

710. E.H. Hebern, "Electronic Coding Machine, " U.S. Patent #1,510,441, 30 Sep 1924.

711. N. Heintze and J.D. Tygar, "A Model for Secure Protocols and their Compositions, "

Proceedings of the 1994 IEEE Computer Society Symposium on Research in Secu rity and Privacy 1994, pp. 2-13.

712. M.E. Hellman, "An Extension of the Shannon Theory Approach to Cryptography," IEEE Transactions on Information Theory, v. IT-23, n. 3, May 1977, pp. 289-294.

713. M.E. Hellman, "The Mathematics of Public-Key Cryptography," Scientihc American, v. 241, n. 8, Aug 1979, pp. 146-157.

714. M.E. Hellman, "DES Will Be Totally Insecure within Ten Years, " IEEE Spectrum, v. 16, n. 7, Jul 1979, pp. 32-39.

715. M.E. Hellman, "On DES-Based Synchronous Encryption," Dept. of Electrical Engineering Stanford University, 1980.

716. M.E. Hellman, "A Cryptanalytic Time- Memory Trade Off," IEEE Transactions on Information Theory, v. 26, n. 4, Jul 1980, pp. 401-406.

717. M.E. Hellman, "Another Cryptanalytic Attack on Cryptosystem for Multiple Communications', " Information Processing Letters, v. 12, 1981. pp. 182-183.

718. M.E. Hellman, W. Diffie, and R.C. Merkle, "Cryptographic Apparatus and Method," U.S. Patent #4,200,770, 29 Apr 1980.

719. M.E. Hellman, W. Diffie, and R.C. Merkle, "Cryptographic Apparatus and Method," Canada Patent #1,121,480, 6 Apr 1982.

720. M.E. Hellman and R.C. Merkle, "Public Key Cryptographic Apparatus and Method," U.S. Patent #4,218,582, 19 Aug 1980.

721. M.E. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig, and R Schweitzer, "Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard," Technical Report SEL 76-042, Information Systems Lab, Department of Electrical Engineering Stanford University, 1976.

722. M.E. Hellman and S.C. Pohlig, "Exponentiation Cryptographic Apparatus and Method," U.S. Patent #4,424,414, 3 Jan 1984.

723. M.E. Hellman and J.M. Reyneri, "Distribution of Drainage in the DES," Advances in Cryptology: Proceedings of Crypto 82, Plenum Prcss, 1983, pp. 129-131.

724. E Hendessi and M.R. Arcf, "A Successful Attack Against the DES, " Third Canadian Workshop on Information Theory and Applications, Springer-Verlag, 1994, pp. 78-90.

725. T. Herlestam, "Critical Remarks on Some 738. Public-Key Cryptosystems, " BIT, v. 18, 1978, pp. 493-496.

726. T. Herlestam, "On Ftmctions of Linear Shift Register Sequences", Advances in Cryptology EUROCRYPT '85, Springer Verlag, 1986, pp. 119-129.

727. T. Herlestam and R. Iohannesson, "On Computing Logarithms over GF (2P)," BIT, 740. v. 21, 1981, pp. 326-334.

728. H.M. Heys and S.E. Tavares, "On thc Security of the CAST Encryption Algorithm," Proceedings of the Canadian Conference on Electrical and Computer Engineenng, Halifax, Nova Scotia, Sep 1994, pp. 332-335.

729. H.M. Heys and S.E. Tavares, "The Design of Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis," Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security, ACM Press, 1994, pp. 148-155.

730. E. Heyst and T.P. Pederson, "How to Make Fail-Stop Signatures," Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag 1993, pp. 366-377.

731. E. Heyst, T.R Pederson, and B. Pfitzmann, "New Construction of Fail-Stop Signatures and Lower Bounds," Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag 1993, pp. 15-30.

732. L.S. Hill, "Cryptography in an Algebraic Alphabet," American Mathematical Monthly, v. 36, Jun-Jul 1929, pp. 306-312.

733. P.J.M. Hin, "Channel-Error-Correcting Privacy Cryptosystems," Ph.D. dissertation, Delft University of Technology, 1986. (In Dutch).

734. R. Hirschield, "Making Elcctronic Refunds Safer, " Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 106 -112.

735. A. Hodges, Alan Turing: The Enigma of Intelligence, Simon and Schuster, 1983.

736. W. Hohl, X. Lai, T. Meier, and C. Waldvogel, "Security of Iterated Hash Functions Based on Block Ciphers, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 379-390.

737. F. Hoornaert, M. Decroos, J. Vandewalle, and R. Govaerts, "Fast RSA-Hardware: Dream or Reality?" Advances in Cryptology E UROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 257-264.

738. F. Hoornaert, J. Goubert, and Y. Desmedt, "Efficient Hardware Implementation of the DES," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 147-173.

739. E. Horowitz and S. Sahni, Fundamentals of Computer Algonthms, Rockville, MD: Computer Science Press, 1978.

740. R Horster, H. Petersen, and M. Michels, "Meta-EIGamal Signature Schemes," Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security, ACM Press, 1994, pp. 96-107.

741. R Horster, H. Petersen, and M. Michels, "Meta Message Recovery and Meta Blind Signature Schemes Based on the Discrete Logarithm Problem and their Applications, " Advances in Cryptology ASIACRYPT '94 Proceedings, Springer-Verlag, 1995, pp. 224-237.

742. L.K. Hua, Introduction to Number Theory, Springer-Verlag, 1982.

743. K. Huber, "Specialized Attack on Chor-Rivest Public Key Cryptosystem, " Electronics Letters, v.27, n. 23, 7 Nov 1991, pp. 2130-2131.

744. E. Hughes, "A Cypherpunk's Manifesto, " 9 Mar 1993.

745. E. Hughes, "An Encrypted Key Transmission Protocol," presented at the rump s ession of CRYPTO '94, Aug 1994.

746. H. Hule and W.B. Muller, "On the RSA- Cryptosystem with Wrong Keys," Contributions to General Algebra 6, Vienna: Verlag Holder-Pichler-Tempsky, 1988, pp. 103-109.

747. H.A. Hussain, J.W.A. Sada, and S.M. Kalipha, "New Multistage Knapsack Public-Key Cryptosystem," International Journal of Systems Science, v. 22, n. 11, Nov 1991, pp. 2313-2320.

748. T. Hwang, "Attacks on Okamoto and Tanaka's One-Way ID-Based Key Distribution System," Information Processing Letters,v.43,n.2,Augl992, pp.83-86.

749. T. Hwang and T.R.N. Rao, "Secret Error- Correcting Codes (SECC)." Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 540-563.

750. C. Ianson and C. Mitchell, "Security Defects in CCITT Recommendation X.509 the Directory Authentication Framework," Computer Communications Review, v. 20, n. 2, Apr 1990, pp. 30-34.

751. IBM, "Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference," SC40-1675-1, IBM Corp., Nov 1990.

752. IBM, "Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference Public Key Algorithm," IBM Corp., Mar 1993.

753. R. Impagliazzo and M. Yung, "Direct Minimum-Knowledge Computations, " Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 40-51.

754. I. Ingemarsson, "A New Algorithm for the Solution of the Knapsack Problem," Lecture Notes in Computer Science 149; Cryptography: Proceedings of the Workshop on Cryptography, Springer-Verlag, 1983, pp. 309-315.

755. I. Ingemarsson, "Delay Estimation for Truly Random Binary Sequences or How to Measure the Length of Rip van Winkle's Sleep," Communications and Cryptography: Two Sides of One Tapestry, R.E. Blahut ct al., eds., Kluwer Adademic Pub- lishcrs, 1994, pp. 179-186.

756. I. Ingemarsson and G.J. Simmons, "A Protocol to Set Up Shared Secret Schemes without the Assistance of a Mutually Trusted Party," Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 266-282.

757. I. Ingemarsson, D.T. Tang, and C.K. Wong, "A Conference Key Distribution System," IEEE Transactions on Information Theory, v. IT-28, n. 5, Sep 1982, pp. 714-720.

758. ISO DIS 8730, "Banking Requirements for Message Authentication Wholesale," Association for Payment Clearing Services, London, Jul 1987.

759. ISO DIS 8781-1, "Banking Approved Algorithms for Message Authentication Part 1: DEA" Association for Payment Clearing Services, London, 1987.

760. ISO DIS 8731-2, "Banking Approved Algorithms for Message Authentication Part 2: Message Authenticator Algorithm, " Association for Payment Clearing Services, London, 1987.

761. ISO DIS 8732, "Banking Key Management (Wholesaled) " Association for Payment Clearing Services, London, Dec 1987.

762. ISO/IEC 9796, "Information Technology Security Techniques . Digital Signature Scheme Giving Message Recovery," International Organization for Standardization, Jul 1991.

763. ISO/IEC 9797, "Data Cryptographic Techniques . Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm," International Organization for Standardization, 1989.

764. ISO DIS 10118 DRAFT, "Information Technology Security Techniques . Hash Functions" International Organization for Standardization, 1989.

765. ISO DIS 10118 DRAFT, "Information Technology Security Techniques . Hash Functions" International Organization for Standardization, April 1991.

766. ISO N98, "Hash Functions Using a Pseudo Random Algorithm," working document, ISO-IEC/JTC 1 /SC27/WG2, International Organization for Standardization, 1992.

767. ISO N179, "AR Fingerprint Function," working document, ISO-IEC/JTC1/SC27/ WG2, International Organization for Standardization, 1992.

768. ISO/IEC 10118, "Information Technology Security Techniques . Hash Functions Part 1: General and Part 2: Hash-Functions Using an e-Bit Block Cipher Algorithm, " International Organization for Standardization, 1993.

769. K. Ito, S. Kondo, and Y. Mitsuoka, "SXAL8/MBAL Algorithm," Technical Report, ISEC93-68, IEICE Japan, 1993. (In Japancsc.)

770. K.R. Iversen, "The Application of Cryptographic Zero-Knowledge Techniques in Computerized Secret Ballot Election Schemes, " Ph.D. dissertation, IDT-report 1991:3, Norwegian Institute of Technology, Feb 1991.

771. K.R. Iversen, "A Cryptographic Scheme for Computerized General Elections, " Advances in Cryptology CRYPTO '91 Proceedings, Springcr-Vcrlag, 1992, pp. 405 -419.

772. K. Iwamura, T. Matsumoto, and H. Imai, "An Implcmcutation Method for RSA Cryptosystem with Parallel Processing ", Transactions of the Institute of Electronics, Information, and Communication Engineers, v. J75-A, n. 8, Aug 1992, pp. 1301-1311.

773. W.J. Jaburek, "A Generalization of ElGamal's Public Key Cryptosystem, " Advances in Cryptology EUROCRYPT '89 Proceedings, 1990, Springer-Verlag, pp. 23-28.

774. N.S. James, R. Lidl, and H. Niederreiter, "Breaking the Cade Cipher," Advances in Cryptology CRYPTO '86 Proceedings, 1987, Springer-Verlag, pp. 60-63.

775. C.J.A. Jansen, "On the Key Storage Requirements for Secure Terminals'" Computers and Security, v. 5, n. 2, Jun 788. 1986, pp. 145-149.

776. C.J.A. Jansen, "Investigations on Nonlin ear Strcamcipher Systems: Construction and Evaluation Methods," Ph.D. dissertation, Technical University of Delft, 1989.

777. C.J.A. Jansen and D.E. Boekee, "Modes of Blockcipher Algorithms and their Protection against Active Eavesdropping, " Advances in Cryptology EUROCRYPT '87 Proceedings, Springer-Verlag, 1988, pp. 281-286.

778. S.M. Jennings, "A Special Class of Binary Sequences," Ph.D. dissertation, University of London, 1980.

779. S.M. Jennings, "Multiplexed Sequences: Some Properties of the Minimum Polynomial," Lecture Notes in Computer Science 149; Cryptography: Proceedings of the Workshop on Cryptography, Springer-Verlag, 1983, pp. 189-206.

780. S.M. Jennings, "Autocorrelation Function of the Multiplexed Sequence, " IKE Pro ceedings, v. 131, n. 2, Apr 1984, pp. 169-172.

781. T. Jin, "Care and Feeding of Your Three Headed Dog, " Document Number IAG-90 -011, Hewlett-Packard, May 1990.

782. T. Jin, "Living with Your Three-Headed Dog," Document Number IAG-90-012, Hewlett-Packard, May 1990.

783. A. Jiwa, J. Seberry. and Y. Zheng, "Beacon Based Authentication," Computer Security ESORICS 94, Springer-Verlag, 1994, pp. 125-141.

784. D.B. Johnson, G.M. Dolan, M.J. Kelly, A.V . Le, and S.M. Matyas, "Common Cryptographic Architecture. Cryptographic Application Programming Interface," IBM Systems journal, v. 30, n.2, 1991, pp.130-150.

785. D.B. Johnson, S.M. Matyas, A.V. Le. and J.D. Wilkins, "Design of the Commercial Data Masking Facility Data Privacy Algorithm," 1st ACM Conference on Computer and Communications Security, ACM Press, 1993, pp. 93-96.

786. J.R Jordan, "A Variant of a Public-Key Cryptosystem Based on Goppa Codes," Sigact News, v. 15, n. 1, 1983, pp. 61-66.

787. A. Joux and L. Granboulan, "A Practical Attack Against Knapsack Based Hash Functions" Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

788. A. Joux and J. Stern, "Cryptanalys is of Another Knapsack Cryptosystem, " Advances in Cryptology ASIACRYPT '91 Proceedings, Springer-Verlag, 1993, pp. 470-476.

789. R.R. Jueneman, "Analysis of Certain Aspects of Output-Feedback Mode, " Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp.99-127.

790. R.R. Jueneman, "Electronic Document Authentication, " IEEE Network Magazine, v. 1, n.2, Apr 1978, pp. 17-23.

791. R.R. Jueneman, "A High Speed Manipulation Detection Code," Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 327-346.

792. R.R. Jueneman, S.M. Matyas, and C.H. Meyer, "Message Authentication with Manipulation Detection Codes," Proceedings of the 1983 IEEE Computer Society Symposium on Research in Security and Privacy, 1983, pp. 733-54.

793. R.R. Jucncman, S.M. Matyas, and C.H. Meyer, "Message Authentication, " IEEE Communications Magazine, v. 23, n. 9, Sep 1985, pp. 29 40.

794. D. Kahn, The Codebreakers: The Story of Secret Writing, New York: Macmillan Publishing Co., 1967.

795. D. Kahn, Kahn on Codes, New York: Macmillan Publishing Co., 1983.

796. D. Kahn, Seizing the Enigma, Boston: Houghton Mifflin Co., 1991.

797. P. Kaijser, T. Parker, and D. Pinkas, "SESAME: The Solution to Security for Open Distributed Systems," Journal of Computer communications, V. 17, n. 4, Jul 1994, pp. 501-518.

798. R. Kailar and V.D. Gilgor, "On Belief Evolution in Authentication Protocols," Proceedings of the Computer Security Foundations Workshop IV, IEEE Computer Society Press, 1991, pp. 102-116.

799. B.S. Kaliski, "A Pseudo Random Bit Generator Based on Elliptic Logarithms," Master's thesis, Massachusetts Institute of Technology, 1987.

800. B.S. Kaliski, letter to NIST regarding DSS, 4 Nov 1991.

801. B.S. Kaliski, "The MD2 Message Digest Algorithm," RFC 1319, Apr 1992.

802. B.S. Kaliski, "Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certificates and Related Services," RFC 1424, Feb 1993.

803. B.S. Kaliski, "An Overview of the PKCS Standards," RSA Laboratories, Nov 1993.

804. B.S. Kaliski, "A Survey of Encryption Standards, IEEE Micro, v. 13, n. 6, Dec 1993, pp. 74-81.

805. B.S. Kaliski, personal communication, 1993.

806. B.S. Kaliski, "On the Security and Performance of Several Triple-DES Modes," RSA Laboratories, draft manuscript, Jan 1994.

807. B.S. Kaliski, R.L. Rivest, and A.T. Sherman, "Is the Data Encryption Standard a Group?", Advances in Cryptology EUROCRYPT '85, Springer-Verlag, 1986, pp. 81-95.

808. B.S. Kaliski, R.L. Rivest, and A.T. Sherman, "Is the Data Encryption Standard a Pure Cipher? Results of More Cycling Experiments in DESK, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 212-226.

809. B.S. Kaliski, R.L. Rivest, and A.T. Sherman, "Is the Data Encryption Standard a Group? (Results of Cycling Experiments on DESK," Journal of Cryptology v. 1, n. 1, 1988, pp. 3-36.

810. B.S. Kaliski and M.J.B. Robshaw, "Fast Block Cipher Proposal," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 33 40.

811. B.S. Kaliski and M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations," Advances in Cryptology CRYPTO '94 Proceedings, Springer-Verlag, 1994, pp. 26-39.

812. B.S. Kaliski and M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations and PEAL," K.U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

813. R.G. Kammer, statement before the U.S. government Subcommittee on Telecommunications and Finance, Committee on Energy and Commerce, 29 Apr 1993.

814. T. Kaneko, K. Koyama, and R. Terada, "Dynamic Swapping Schemes and Differential Cryptanalysis, Proceedings of the 1993 Korea- Japan Workshop on Information Security and Cryptography Seoul, Korea, 24-26 Oct 1993, pp. 292-301.

815. T. Kaneko, K. Koyama, and R. Terada, "Dynamic Swapping Schemes and Differential Cryptanalysis," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. E77-A, n. 8, Aug 1994, pp. 1328-1336.

816. T. Kaneko and H. Miyano, "A Study on the Strength Evaluation of Randomized DES-Like Cryptosystems against Chosen Plaintext Attacks, " Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93), Shozenji, Japan, 28-30 Jan 1993, pp. 15C.1-10.

817. J. Karl, "A Cryptosystem Based on Propositional Logic," Machines, Languages, and Complexity: 5th International Meeting of Young Computer Scientists, Selected Contributions, Springer-Verlag, 1989, pp. 210-219.

818. E.D. Karnin, J.W. Greene, and M.E. Hellman, "On Sharing Secret Systems," IEEE Transactions on Information Theory v. IT- 29, 1983, pp. 35 41.

819. F.W Kasiski, Die Geheimschriften and die Dechiffrir-kunst, E.S. Miller und Sohn, 1863. In German.

820. A. Kehne, J. Schonwalder, and H. Langendorfer, "A Nonce-Based Protocol for Multiple Authentications," Operating Systems Review, v. 26, n. 4, Oct 1992, pp. 84-89.

821. J. Kelsey, personal communication, 1994.

822. R. Kemmerer, "Analyzing Encryption Protocols Using Formal Verification Techniques, " IEEE Journal on Selected Areas in Communications, v. 7, n. 4, May 1989, pp. 448 457.

823. R. Kemmerer, C.A. Meadows, and J. Millen, "Three Systems for Cryptographic Protocol Analysis," Journal of Cryptology v. 7, n. 2, 1994, pp. 79-130.

824. S.T. Kent, "Encryption-Based Protection Protocols for Interactive User-Computer Communications, " MIT/LCS/TR162, MIT Laboratory for Computer Science, May 1976.

825. S.T. Kent, "Privacy Enhancement for Internct Electronic Mail: Part II: Certificate Based Key Management," RFC 1422, Feb 1993.

826. S.T. Kent, "Understanding the Internet Certification System, " Proceedings of INET '93, The Internet Society, 1993, pp. BAB 1 -BAB 10.

827. S.T. Kent and J. Linn, "Privacy Enhaneement for Internet Electronic Mail: Part II: Certificate-Based Key Management," RFC 1114, Aug 1989.

828. V. Kessler and G. Wedel, "AUTOLOG An Advanced Logic of Authentication," Proceedings of the Computer Security Foundations Workshop, IEEE Computer Society Press, 1994, pp. 90-99.

829. E.L. Key, "An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators," IEEE Transactions on Information Theory v. IT-22, n. 6, Nov 1976, pp. 732-736.

830. T. Kiesler and L. Harn, "RSA Blocking and Multisignature Schemes with No Bit Expansion," Electronics Letters, v. 26, n. 18, 30 Aug 1990, pp. 1490-1491.

831. J. Kilian, Crises of Randomness in Algorithms and Protocols, MIT Press, 1990.

832. J. Kilian, "Achieving Zero-Knowledge Robustly, " Advances in Cryptology CRYPTO 90 Proceedings, Springer-Verlag, 1991, pp. 313-325.

833. J. Kilian and T. Leighton, "Failsafe Key Escrow," MIT/LCS/TR-636, MIT Laboratory for Computer Science, Aug 1994.

834. K. Kim, "Construction of DES-Like S-Boxes Based on Boolean Functions Satisfying the SAC, " Advances in Cryptology, ASIACRYPT 91 Proceedings, Springer-Verlag, 1993, pp. 59-72.

835. K. Kim, S. Lee, and S. Park, "Necessary Conditions to Strengthen DES S-Boxes Against Linear Cryptanalysis," Proceedings of the 1994 Symposium on Cryptography and Information Secunty (SCIS 94), Lake Biwa, Japan, 27-29 Jan 1994, pp. 15D.1-9.

836. K. Kim, S. Lee, and S. Park, "How to Strengthen DES against Differential Attack, " unpublished manuscript, 1994.

837. K. Kim, S. Lee, S. Park, and D. Lee, "DES Can Be Immune to Differential Cryptanalysis, " Workshop on Selected Areas in Cryptography Workshop Record, Kingston, Ontario, 5-6 May 1994, pp. 70-81.

838. K. Kim, S. Park, and S. Lee, "How to Strengthen DES against Two Robust Attacks," Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography Inuyama, Japan, 24-27 Jan 1995, 173-182.

839. K. Kim, S. Park, and S. Lee, "Reconstruction of s2DES S-Boxes and their Immunity to Differential Cryptanalysis, " Proceedings of the 1993 Korea- Japan Workshop on Information Security and Cryptography, Seoul, Korea, 24-26 Oct l993, pp.282-291.

840. S. Kim and B.S. Um, "A Multipurpose Membership Proof System Based on Discrete Logarithm," Proceedings of the 1993 Korea- Japan Workshop on Information Security and Cryptography, Seoul, Korea, 24-26 Oct 1993, pp. 177-183.

841. P. Kinnucan, "Data Encryption Gurus: Tuchman and Meyer," Cryptologia, v. 2, n. 4, Oct 1978.

842. A. Klapper, "The Vulnerability of Geometric Sequences Based on Fields of Odd Characteristic," Journal of cryptology v. 7, n. 1, 1994, pp. 33-52.

843. A. Klapper, "Feedback with Carry Shift Registers over Finite Fields," K. U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

844. A. Klapper and M. Goresky, "2-adic Shift Registers, " Fast Software Encryption, Cambridge Security Workshop l'roceed- ings, Springer-Verlag, 1994, pp. 174-178.

845. A. Klapper and M. Goresky, "2-adic Shift Registers," Technical Report #239-93, Department of Computer Science, University of Kentucky, 19 Apr 1994.

846. A. Klapper and M. Goresky, "Large Period Nearly de Bruijn FCSR Sequences, " Advances in Cryptology EUROCRYPT 95 Proceedings, Springer-Verlag, 1995, pp. 263-273.

847. D.V. Klein, "'Foiling the Cracker: A Survey of, and Implications to, Password Security," Proceedings of the USENIX UNIX Security Workshop, Aug 1990, pp. 5-14.

848. D.V Klein, personal communication, 1994.

849. C.S. Kline and G.J. Popek, "Public Key vs. Conventional Key Cryptosystems," Proceedings of AFIPS National Computer Conference, pp. 831-837.

850. H.-J. Knobloch, "A Smart Card Implementation of the Fiat-Shamir Identification Scheme," Advances in Cryptology EUROCRPYT '88 Proceedings, Springer-Verlag, 1988, pp. 87-95.

851. T. Knoph, J. Fropl, W. Beller, and T.Giesler, "A Hardware Implementation of a Modified DES Algorithm," Microprocessing and Microprogramming, v. 30, 1990, pp. 59-66.

852. L.R. Knudsen, "Cryptanalysts of LOKI," Advances in Cryptology ASIACRYPT '91 Proceedings, Springer-Verlag, 1993, pp. 22-35.

853. L.R. Knudsen, "Cryptanalysts of LOKI," Cryptography and Coding 111, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 223-236.

854. L.R. Knudsen, "Cryptanalysts of LOKI91," Advances in Cryptology AUSCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 196-208.

855. L.R. Knudsen, "Iterative Characteristics of DES and sZDES," Advances in Cryptology CRYPTO '92, Springer-Verlag, 1993, pp. 497-511.

856. L.R. Knudsen, "An Analysis of Kim, Park and Lee's DES-Like S-Boxes," unpublished manuscript, 1993.

857. L.R. Knudsen, "Practically Secure Feistel Ciphers," Fast Software Encryption, Cambridge Secunty Workshop Proceedings, Springer-Verlag, 1994, pp. 211-221.

858. L.R. Knudsen, "Block Ciphers Analysis, Design, Applications," Ph.D. dissertation, Aarhus University, Nov 1994.

859. L.R. Knudsen, personal communication, 1994.

860. L.R. Knudsen, "Applications of Higher Order Differentials and Partial Differentials, " K. U Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

861. L.R. Knudsen and X. Lai, "New Attacks on All Double Block Length Hash Functions of Hash Rate 1, Including the Parallel-DM," Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

862. L.R. Knudsen, "A Weakness in SAFER K-64, " Advances in Cryptology-CRYPT O '95 Proceedings, Springer-Verlag, 1995, to appear.

863. D. Knuth, The Art of Computer Programming: Volume 2, Seminumerical Algo rithms, 2nd edition, Addison-Wesley, 1981.

864. D. Knuth, "Deciphering a Linear Congruential Encryption," IEEE Transactions on Information Theory, v. IT-31, n. 1, Jan 1985, pp. 49-52.

865. K. Kobayashi and L. Aoki, "On Linear Cryptanalysis of MBAL, " Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95, Innyama, Japan, 24-27 Jan 1995, pp. A4.2.1-9.

866. K. Kobayashi, K. Tamura, and Y. Nemoto, "Two-dimensional Modified Rabin Cryptosystem," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. J72-D, n. 5, May 1989, pp. 850-851. (In Japanese.)

867. N. Koblitz, "Elliptic Curve Cryptosystems, " Mathematics of Computation, v. 48, n. 177, 1987, pp. 203-209.

868. N. Koblitz, "A Family of Jacobians Suitable for Discrete Log Cryptosystems, " Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag 1990, pp. 94 -99.

869. N. Koblitz, "Constructing Elliptic Curve Cryptosystems in Characteristic 2," Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag 1991, pp. 15 6-167.

870. N. Koblitz, "Hyperelliptic Cryptosystems," Journal of Cryptology, v. 1, n. 3, 1989, pp. 129-150.

871. N. Koblitz, "CM-Curves with Good Cryptographic Properties, " Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 279-287.

872. C.K. Koc, "High-Speed RSA Implementation," Version 2.0, RSA Laboratories, Nov 1994.

873. M.J. Kochanski, "Remarks on Lu and Lee's Proposals," Crypto logia, v. 4, n. 4, 1980, pp. 204-207.

874. M.J. Kochanski, "Developing an RSA Chip," Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 350-357.

875. J.T. Kohl, "The Use of Encryption in Kerberos for Network Authentication," Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp.35 -43.

876. J.T. Kohl, "The Evolution of the Kerberos Authentication Service," Eur opean Conference Proceedings, May 1991, pp. 295-313.

877. J.T. Kohl and B.C. Neuman, "The Kerberos Network Authentication Service, " RFC 1510, Sep 1993.

878. J.T. Kohl, B.C. Neuman, and T. Ts'o, "The Evolution of the Kerberos Authentication System," Distributed Open Systems, IEEE Computer Society Press, 1994, pp. 78-94.

879. Kohnfelder, "Toward a Practical Public Key Cryptosystem, " Bachelor's thesis, MIT Department of Electrical Engineering, May 1978.

880. A. G. Konheim, Cryptography: A Primer, New York: John Wiley & Sons, 1981.

881. A.G. Konheim, M.H. Mack, R.K. McNeill, B. Tuckerman, and G. Waldbaum, "The IPS Cryptographic Programs," IBM Systems journal, v. 19, n. 2, 1980, pp. 253-283.

882. V.I. Korzhik and A.I. Turkin, "Cryptanalysis of McEliece's Public-Key Cryptosystem," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 68-70.

883. S.C. Kothari, "Generalized Linear Threshold Scheme," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 231-241.

884. J. Kowalchuk, B.R Schanning, and S. Powers, "Communication Privacy: Integration of Public and Secret Key Cryptography," Proceedings of the National Telecommunication Conference, IEEE Press, 1980, pp. 49.1.1 49.1.5.

885. K. Koyama, "A Master Key for the RSA Public-Key Cryptosystem," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. J65-D, n. 2, Feb 1982, pp. 163-170.

886. K. Koyama, "A Cryptosystem Using the Master Key for Multi-Address Communications," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. J65-D, n. 9, Sep 1982, pp. 1151-1158.

887. K. Koyama, "Demonstrating Membership of a Group Using the Shizuya-Koyama -Itoh (SKI) Protocol," Proceedings of the 1989 Symposium on Cryptography and Information Security (SCIS 89), Gotenba, Japan, 1989.

888. K. Koyama, "Direct Demonstration of the Power to Break Public-Key Cryptosystems, " Advances in Cryptology AUSCRYPT '90 Proceedings, Springer-Verlag, 1990, pp. 14-21.

889. K. Koyama, "Security and Unique Decipherability of Two-dimensional Public Key Cryptosystems," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. E73, n. 7, Jul 1990, pp. 1057-1067.

890. K. Koyama, U.M. Maurer, T. Okamoto, and S.A. Vanstone, "New Public-Key Schemes Based on Elliptic Curves over the Ring Zn" Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 252-266.

891. K. Koyama and K. Ohta, "Identity-based Conference Key Dist ribution System," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp.175-184.

892. K. Koyama and T. Okamoto, "Elliptic Curve Cryptosystems and Their Applications, " IEICE Transactions on Information and Systems, v. E75-D, n. 1, Jan 1992, pp. 50-57.

893. K. Koyama and R. Terada, " How to Strengthen DES-Like Cryptosystems against Differential Cryptanalysis," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. E76-A, n. 1, Jan 1993, pp. 63-69.

894. K. Koyama and R. Terada, "Probabilistic Swapping Schemes to Strengthen DES against Differential Cryptanalysis, " Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93), Shuzenji, Japan, 28-30 Jan 1993, pp. 15D.1-12.

895. K. Koyama and Y. Tsuruoka, "Speeding up Elliptic Cryptosystems Using a Singled Binary Window Method, " Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 345-357.

896. E. Kranakis, Pnmality and Cryptography, Wiler-Teubner Series in Computer Science, 1986.

897. D. Kravitz, "Digital Signature Algorithm," U.S. Patent #5,231,668, 27 Jul 1993.

898. D. Kravitz and L. Reed, "Extension of RSA Cryptostructure: A Galois Approach, " Electronics Letters, v. 18, n. 6, 18 Mar 1982, pp. 255-256.

899. H. Krawczyk, "How to Predict Congruential Generators," Advances in Cryptology CRYPTO'89 Proceedings, Springer- Verlag, 1990, pp. 138-153.

900. H. Krawczyk, "How to Predict Congruential Generators," Journal of Algorithms, v. 13, n. 4, Dec 1992, pp. 527-545.

901. H. Krawczyk, "The Shrinking Generator: Some Practical Considerations," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 45 46.

902. G.J. Kuhn, "Algorithms for Self-Synchronizing Ciphers," Proceedings of COMSIG 88, 1988.

903. G.J. Kuhn, F. Bruwer, and W. Smit, "'n Vinnige Veeldoelige Enkripsievlokkie, " Proceedings of Infosec 90, 1990. (In Afrikaans.)

904. S. Kullback, Statistical Methods in Cryptanalysis, U.S. Government Printing Office, 1935. Reprinted by Aegean Park Press, 1976.

905. RV. Kumar, R.A. Scholtz, and L.R. Welch, "Generalized Bent Functions and their Properties," Journal of Combinational Theory, Series A, v. 40, n. 1, Sep 1985, pp. 90-107.

906. M. Kurosaki, T. Matsumoto, and H. Imai, "Simple Methods for Multipurpose Certification," Proceedings of the 1989 Symposium on Cryptography and Information Security (SCIS 89), Gotenba, Japan, 1989.

907. M. Kurosaki, T. Matsumoto, and H. Imai, "Proving that You Belong to at Least One of the Specified Groups," Proceedings of the 1990 Symposium on Cryptography and Information Security (SCIS 90), Hihondaira, Japan, 1990.

908. K. Kurosawa, "Key Changeable ID-Based Cryptosystem," Electronics Letters, v. 25, n. 9, 27 Apr 1989, pp. 577-578.

909. K. Kurosawa, T. Ito, and M. Takeuchi, "Public Key Cryptosystem Using a Reciprocal Number with the Same Intractability as Factoring a Large Number," Cryptologia, v. 12, n. 4, Oct 1988, pp. 225-233.

910. K. Kurosawa, C. Park, and K. Sakano, "Group Signer/Verifier Separation Scheme," Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography, Inuyama, Japan, 24-27 Jan 1995, 134-143.

911. G.C. Kurtz, D. Shanks, and H.C. Williams, "Fast Primality Tests for Numbers Less than 50*10" Mathematics of Computation, v. 46, n. 174, Apr 1986, pp. 691-701.

912. K. Kusuda and T. Matsumoto, "Optimization of the Time-Memory Trade-Off Cryptanalysis and Its Application to Block Ciphers, " Proceedings of the 1995 Symposium on

Cryptography and Information Security (SCIS 95), Inuyama, Japan, 24-27 Jan 1995, pp. A3.2.1-11. (In Japanese.)

913. H. Kuwakado and K. Koyama, "Security of RSA-Type Cryptosystems Over Elliptic Curves against Hastad Attack," Electronics Letters, v. 30, n. 22, 27 Oct 1994, pp. 1843-1844.

914. H. Kuwakado and K. Koyama, "A Ncw RSA-Type Cryptosystem over Singular Elliptic Curves, " IMA Conference on Applications of Finite Fields, Oxford University Press, to appear.

915. H. Kuwakado and K. Koyama, "A New RSA-Type Scheme Based on Singular Cubic Curves," Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography, Inuyama, Japan, 24-27 Jan 1995, pp. 144-151.

916. M. Kwan, "An Eight Bit Weakness in the LOKI Cryptosystem, " technical report, Australian Dcfensc Force Academy, Apr 1991.

917. M. Kwan and J. Pieprzyk, "A General Purpose Technique for Locating Key Scheduling Weakness in DES-Like Cryptosystcms, " Advances in Cryptology ASIACRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 237-246.

918. J.B. Lacy, D.P. Mitchell, and W.M. Schell, "CryptoLib: Cryptography in Software," UNIX Security Symposium Proceedings, USENIX Association, 1993, pp. 1-17.

919. J.C. Lagarias, "Knapsack Public Key Cryptosystems and Diophantine Approximations," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 3-23.

920. J.C. Lagarias, "Performance Analysis of Shamir's Attack on the Basic Merkle-Hellman Knapsack Cryptosystem, " Lec- ture Notes in Computer Science 172; Proceedings of the 11 th International Colloquium on Automata, Languages, and Programming (ICALP), Springer- Verlag, 1984, pp. 312-323.

921. J.C. Lagarias and A.M. Odlyzko, "Solving Low-Density Subset Sum Problems," Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, I 983, pp. 1-10.

922. J.C. Lagarias and A.M. Odlyzko, "Solving Low-Density Subset Sum Problems," Journal of the ACM, v. 32, n. 1, Jan 1985, pp. 229-246.

923. J.C. Lagarias and J. Reeds, "Unique Extraporation of Polynomial Recurrences, " SIAM Journal on Computing, v. 17, n. 2, Apr 1988, pp. 342-362.

924. X. Lai, Detailed Description and a Software Implementation of the IPES Cipher, unpublished manuscript, 8 Nov 1991.

925. X. Lai, On the Design and Secunty of Block Ciphers, ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.

926. X. Lai, personal communication, 1993.

927. X. Lai, "Higher Order Derivatives and Differential Cryptanalysis," Communications and Cryptography: Two Sides of One Tapestry, R.E. Blahut et al., eds., Kluwer Adademic Publishers, 1994, pp. 227-233.

928. X. Lai and L. Knudsen, "Attacks on Double Block Length Hash Functions," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 157-165.

929. X. Lai and J. Massey, "A Proposal for a New Block Encryption Standard, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 389 404.

930. X. Lai and J. Massey, "Hash Functions Based on Block Ciphers, " Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag, 1992, pp. 55-70.

931. X. Lai, J. Massey, and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 17-38.

932. X. Lai, R.A. Rueppel, and J. Woollven, "A Fast Cryptographic Checksum Algorithm Based on Stream Ciphers," Advances in Cryptology AUSCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 339-348.

933. C.S. Laih, J.Y. Lee, C.H. Chen, and L. Harn, "A New Scheme for ID-based Cryptosys tems .md Signatures," Tournal of the Chinese Institute of Engineers, v. 15, n. 2, Sep 1992, pp. 605-610.

934. B.A. LaMacchia and A.M. Odlyzko, "Computation of Discrete Logarithms in Prime Fields, " Designs, Codes, and Cryptography, v. 1, 1991, pp. 46-62.

935. L. Lamport, "Password Identification with Insecure Communications," Communications of the ACM, v. 24, n. 11, Nov 1981, pp. 770-772.

936. S. Landau, "Zero-Knowledge and the Department of Defense," Notices of the American Mathematical Society, v. 35, n. 1, Jan 1988, pp. 5-12.

937. S. Landau, S. Kent, C. Brooks, S. Charney, D. Denning, W. Diffie, A. Lauck, D. Mikker, P. Neumann, and D. Sobel, "Codes, Keys, and Conflicts: Issues in U.S. Crypto Policy," Report of a Special Panel of the ACM U.S. Public Policy Committee (USACM), Association for Computing Machinery, Jun 1994.

938. S.K. Langford and M.E. Hellman, "Cryptanalysis of DES," presented at 1994 RSA Data Security conference, Redwood Shores, CA, 12-14 Jan 1994.

939. D. Lapidot and A. Shamir, "Publicly Verifiable Non-Interactive Zero-Knowledge Proofs, " Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 353-365.

940. A.V. Le. S.M. Matyas, D.B. Johnson, and J.D. Wilkins, "A Public-Key Extension to the Common Cryptographic Architecture, " IBM Systems Journal, v. 32, n. 3, 1993, pp. 461-485.

941. P. L'Ecuyer, "Efficient and Portable Combined Random Number Generators, " Communications of the ACM, v. 31, n. 6, Jun 1988, pp. 742-749, 774.

942. R L'Ecuyer, "Random Numbers for Simulation," Communications of the ACM, v. 33, n. 10, Oct 1990, pp. 85-97.

943. P.J. Lee and E.E Brickcll, "An Observation on the Security of McEliece's Public-Key Cryptosystem," Advances in Cryptology EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 275-280.

944. S. Lee, S. Sung, and K. Kim, "An Efficient Method to Find the Linear Expressions for Linear Cryptanalysis," Proceedings of the 1995 Korea- Japan Workshop on Information Security and Cryptography, Inuyama, Japan, 24-26 Jan 1995, pp. 183-190.

945. D.J. Lehmann, "On Primality Tests, " SIAM lournal on Computing, v. 11, n. 2, May 1982, pp. 374-375.

946. T. Leighton, "Failsafe Key Escrow Systems, " Technical Memo 483, MIT Laboratory for Computer Science, Aug 1994.

947. A. Lcmpel and M. Cohn, "Maximal Families of Bent Sequences," IEEE Transactions on Information 'Fheory, v. IT-28, n. 6, Nov 963. 1982, pp. 865-868.

948. A. K. Lenstra. " Factoring Multivariate Polynomials Over Finite Fields," Journal of Computer System Science, v. 30, n. 2, 964. Apr 1985, pp. 235 -248.

949. A.K. Lenstra, personal communication, 1995.

950. A.K. Lenstra and S. Haber, letter to NIST Regarding DSS, 26 Nov 1991.

951, A.K. Lcnstra, H.W. Lenstra Jr., and L.Lovacz, "Factoring Polynomials with Rational Coefficients," Mathematische Annalen, v. 261, n. 4, 1982, pp. 515-534.

952. A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, and J.M. Pollard, "The Number Field Sieve," Proceedings of the 22nd ACM Symposium on the Theory of Computing, 1990, pp. 574-672.

953. A.K. Lenstra and H.W. Lenstra, Jr., eds., Lecture Notes in Mathematics 1554: The 967. Development of the Number Field Sieve, Springer-Verlag, 1993.

954. A.K. Lenstra, H.W. Lcnstra, Jr., M.S. Manasse, and J.M. Pollard' "The Factorization of the Ninth Fermat Number," Mathematics of Computation. v. 61, n. 203, 1993, pp. 319-349.

955. A.K. Lenstra and M.S. Manasse, "Factoring by Electronic Mail," Advances in Cryptology EUROCRYPT '89 Proceedings, Springcr-Verlag, 1990, pp. 355-371.

956. A.K. Lenstra and M.S. Manasse, "Factoring with Two Large Primes," Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 72-82.

957. H.W. Lenstra Jr. "Elliptic Curves and N umber-Theoretic Algorithms, " Report 86-19, Mathematisch Instituut, Universiteit van Amsterdam, 1986.

958. H.W. Lenstra Jr. "On the Chor-Rivest Knapsack Cryptosystem, " Journal of Cryptology, v. 3, n. 3, 1991, pp. 149-155.

959. W.J. LeVequc, Fundamentals of Number Theory, Addison-Wesley. 1 977.

960. L.A. Levin, "One-Way Functions and Pseudo-Randolll Generators." Proceedi ngs of the 17th ACM Symposium on Theory of Comp uting, 1985, pp. 363-365.

961. Lexar Corporation, "An Evaluation of thc DES," Scp 1976.

962. D.-X. Li, "Cryptanalysts of Public-Kcy Distribution Systcms Based on Dickson Polynomials," Electronics Letters, v. 27, n. 3, 1991, pp. 228-229.

963. F. -X. Li, "How to Break Okamoto's Cryptosystems by Continued Fraction Algorithm," ASIACRYPT '91 Abstracts, 1991, pp. 285-289.

964. Y.X. Li and X.M. Wang, "A Coins Authcntication and Encryption Schemc Bascd on Algebraic Coding Theory," Applied Algebra, Algebraic Algonthms and Error Correcting Codes 9, Springer-Vcrlag, 1991, pp. 241-245.

965. R. Lidl, G.L. Mullen, and G. Turwald, Pitman Monographs and Surveys in Pure and Applied Mathematics 65: Dickson Polynomials, London: Longman Scicntific and Technical, 1993.

966. R. Lidl and W.B. Muller, "Permutation Polynomials in RSA-Cryptosystems, " Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 293-301.

967. R. Lidl and W.B. Mullcr, "Generalizations of the Fibonacci Pseudoprimes Test," Discrete Mathematics, v. 92, 1991, pp. 211-220.

968. R. Lidl and W.B. Muller, "Primality Testing with Lucas Functions," Advances in Cryptology A USCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 539-542.

969. R. Lidl, W.B. Muller, and A. Oswald, "Some Remarks on Strong Fibonacci Pscudoprimes," Applicable Algebra in Engineering, Communication and Computing, v. 1, n. 1, 1990, pp. 59 65.

970. R. Lidl and H. Niederreiter, "Finite Fields," Encyclopedia of Mathematics and its Application.s, v. 20, Addison-Wesley, 1983.

971. R. Lidl and H. Niederreiter, Introduction to Finite Fields and Their Applicatio ns. London: Cambridge Univcrsity Press, 1986.

972. K. Lieberherr, "Unifonn Complexity and Digital Signaturcs," Theoretical Computer Science, v. 16, n. 1, Oct 1981, pp. Y9-] 10.

973. C.H. Lim and RJ. Lee, "A Practical Electronic Cash System for Smart Cards," Proceedings of the 1993 Korea-Japan Workshop on Informatio n Security and Cryptography Seoul, Korea, 24-26 Oct 1993, pp. 34-47.

974. C.H. Lim and P.J. Lee, "Security of interactive RSA Batch Verification," Electronics Letters, v. 30, n. 19, 15 Sep 1994, pp. 1592-1593.

975. H.-Y. Lin and L. Harn, "A Generalized Secret Sharing Scheme with Cheater Detection," Advances in Cryptology ASIACRYPT '91 Proceedings, Springer-Verlag, 1993, pp. 149-158.

976. M.-C. Lin, T.-C. Chang, and H.-L. Fu, "Information Rate of McEliece's Public key Cryptosystem," Electronics Letters, v. 990. 26, n. 1, 4 Jan 1990, pp. 16-18.

977. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I Message Encipherment and Authentication Procedures, " RFC 989, Feb 1987.

978. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I Message Encipherment and Authentication Procedures, " RFC 1040, Jan 1988.

979. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I Message Encipher ment and Authentication Procedures, " RFC 1113, Aug 1989.

980. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part III Algorithms, Modes, and Identifiers," RFC 1115, Aug 1989.

981. J. Linn, " Privacy Enhancement for Internet Electronic Mail: Part I Message Encipherment and Authentication Procedures, " RFC 1421, Feb 1993.

982. S. Lloyd, "Counting Binary Functions with Certain Cryptographic Properties, " Journal of Cryptology, v. 5, n. 2, 1992, pp. 107-131.

983. T.M.A. Lomas, "Collision-Freedom, Considered Harmful, or How to Boot a Computer," Proceedings of the 1995 Korea-Japan Workshop on Information Secu rity and Cryptography, Inuyama, Japan, 24-26 Jan 1995, pp. 35-42.

984. T.M.A. Lomas and M. Roe, "Forging a Clipper Message, " Communications of the ACM, v. 37, n. 12, 1994, p. 12.

985. D.L. Long, "The Security of Bits in the Discrete Logarithm, " Ph.D. dissertation, Princeton University, Jan 1984.

986. D.L. Long and A. Wigderson, "How Discrete Is the Discrete Log," Proceedings of the 15th Annual ACM Syposium on the Theory of Computing, Apr 1983.

987. D. Longlcy and S. Rigby, "An Automatic Search for Security Flaws in Key Management Schemes," Computers and Security v. 11, n. 1, Jan 1992. pp. 75-89.

988. S.H. Low, N.F. Maxemchuk, and S. Paul, "Anonymous Credit Cards," Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security, ACM Press, 1994, pp. 108-117.

989. J.H. Loxton, D.S.P Khoo, G.J. Bird, and J. Seberry, "A Cubic RSA Code Equivalent to Factorization," Journal of Cryptology, v. 5, n. 2, 1992, pp. 139-150.

990. S.C. Lu and L.N. Lee, "A Simple and Effective Public-Key Cryptosystem," COMSAT Technical Review, 1979, pp. 15-24.

991. M. Luby, S. Micali and C. Rackoff, "How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin, " Proceedings of the 24nd Annual Symposium on the Foundations of Computer Science, 1983, pp. 11-22.

992. M. Luby and C. Rackoff, "How to Construct Pseudo-Random Permutations from Pseudorandom Functions," SIAM lournal on Computing, Apr 1988, pp. 373-386.

993. F. Luccio and S. Mazzone, "A Cryptosystem for Multiple Communications, " Information Processing Letters, v. 10, 1980, pp. 180-183.

994. V Luchangco and K. Koyama, "An Attack on an ID-Based Key Sharing System, Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography, Seoul, Korea, 24-26 Oct 1993, pp. 262-271.

995. D.J.C. MacKay, "A Free Energy Minimization Framework for Inferring the State of a Shift Register Given the Noisy Output Sequence, " K. U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

996. M.D. MacLaren and G. Marsaglia, "Uniform Random Number Generators," Journal of the ACM v. 12, n. 1, Jan 1965, pp. 83-89.

997. D. MacMillan, "Single Chip Encrypts Data at 14M b/s," Electronics, v. 54, n. 12, 16 June 1981, pp. 161-165.

998. R. Madhavan and L.E. Peppard, "A Multiprocessor GaAs RSA Cryptosystem," Proceedings CCVLSI-89: Canadian Conference on Very Large Scale Integration, Vancouver, BC, Canada, 22-24 Oct 1989, pp. 115-122.

999. W.E. Madryga, "A High Performance Encryption Algorithm," Computer Secu rity: A Global Challenge, Elsevier Science Publishers, 1984, pp. 557-570.

1000. M. Mambo, A. Nishikawa, S. Tsujii, and E. Okamoto, "Efficient Secure Broadcast Communication System," Proceedings of the 1993 Korea- Japan Workshop on Information Security and Cryptography, Seoul, Korea, 24-26 Oct 1993, pp. 23 -33.

1001. M. Mambo, K. Usuda, and E. Okamoto, "Proxy Signatures," Proceedings of the 1995 Sympusium on Cryptography and Information Secunty (SCIS 95), Inuyama, Japan, 24-27 Jan 1995, pp. B1.1.1-17.

1002. W. Mao and C. Boyd, "Towards Formal Analysis of Security Protocols," Proceedings of the Computer Security Foundations Workshop Vl, IEEE Computer Society Press, 1993, pp. 147-158.

1003. G. Marsaglia and T.A. Bray, "On-Line Random Number Generators and their Use in Combinations, " Communications of the ACM, v. 11, n. 11, Nov 1968, p. 757-759.

1004. K.M. Martin, "Untrustworthy Participants in Perfect Secret Sharing Schemes," Cryptography and Coding 111, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 255-264.

1005. J.L. Massey, "Shift-Register Synthesis and BCH Decoding," IEEE Transactions on Information Theory, v. IT-15, n. 1, Jan 1969, pp. 122-127.

1006. J.L. Massey, "Cryptography and System Theory," Proceedings of the 24th Allerton Conference on Communication, Control, and Computers, 1-3 Oct 1986, pp. 1-8.

1007. J.L. Massey, "An Introduction to Contemporary Cryptology, " Proceedings of the IEEE, v. 76, n. 5., May 1988, pp. 533-549.

1008. J.L. Massey, "Contemporary Cryptology: An Introduction," in Contemporary Cryptology: The Science of Information Integrity, G.J. Simmons, ed., IEEE Press, 1992, pp. 1-39.

1009. J.L. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 1-17.

1010. J.L. Massey, "SAFER K-64: One Year Later," K. U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

1011. J.L. Massey and I. Ingemarsson, "The Rip Van Winkle Cipher A Simple and Provably Computationally Secure Cipher with a Finite Key," IEEE International Symposium on Information Theory, Brighton, UK, May 1985.

1012. J.L. Massey and X. Lai, "Device for Converting a Digital Block and the Use Thereof, " International Patent PCT/ CH91/00117, 28 Nov 1991.

1013. J.L. Massey and X. Lai, "Device for the Conversion of a Digital Block and Usc of Same," U.S. Patent #5,214,703, 25 May 1993.

1014. J.L. Massey and R.A. Rueppel, "Linear Ciphers and Random Sequence Generators with Multiple Clocks, " Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, pp. 74-87.

1015. M. Matsui, "Linear Cryptanalysis Method for DES Cipher, " Advances in C ryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 386-397.

1016. M. Matsui, "Linear Cryptanalysis of DES Cipher," Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93), Shuzenji, Japan, 28-30 Jan 1993, pp. 3C.1-14. (In Japanese.)

1017. M. Matsui, "Linear Cryptanalysis Method for DES Cipher " Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94), Lake Biwa, Japan, 27-29 Jan 1994, pp. 4A.1-11. (In Japanese.)

1018. M. Matsui, "On Correlation Between the Order of the S-Boxes and the Strength of DES," Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-V erlag, 1995, to appear.

1019. M. Matsui, "The First Experimental Cryptanalysis of thc Data Encryption Standard," Advances in Cryptology CRYPTO ' 94 Proceedings , Springer -Verlag, 1994, pp. 1-11.

1020. M. Matsui and A. Yamagishi, "A New Method for Known Plaintext Attack of FEAL Cipher," Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 81 -91.

1021. T. Matsumoto and H. Imai, "A Class of Asymmetric Crypto-Systems Based on Polynomials Over Finite Rings, " IEEE International Symposium on Information Theory, 1983, pp. 131-132.

1022. T. Matsumoto and H. Imai, "On the Key Production System: A Practical Solution to the Key Distribution Problem," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 185-193.

1023. T. Matsumoto and H. Imai, "On the Security of Some Key Sharing Schemes (Part 2)," IEICE Japan, Technical Report, ISEC90-28, 1990.

1024. S.M. Matyas, "Digital Signatures. An Overview, " Computer Networks, v. 3, n.2, Apr 1979, pp. 87-94.

1025. S.M. Matyas, "Key Handling with Control Vectors," IBM Systems journal, v. 30, n. 2, 1991, pp. 151-174.

1026. S.M. Matyas, A.V. Le. and D.G. Abraham, "A Key Management Scheme Based on Control Vectors," IBM Systems journal, v .30, n. 2, 1991, pp. 175-191.

1027. S.M. Matyas and C.H. Meyer, "Generation, Distribution, and Installation of Cryptographic Keys," IBM Systems Journal, v. 17, n. 2, 1978, pp. 126-137.

1028. S.M. Matyas, C.H. Meyer, and J. Oseas, "Generating Strong One-Way Functions with Cryptographic Algorithm, " IBM Technical Disclosure Bulletin, v. 27, n. 10A, Mar 1985, pp. 5658-5659.

1029. U.M. Maurer, "Provable Security in Cryptography," Ph.D. dissertation, ETH No. 9260, Swiss Federal Institute of Technology, Zurich, 1990.

1030. U.M. Maurer, "A Provable-Secure Strongly-Randomized Cipher," Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1990, pp. 361-373.

1031. U.M. Maurer, "A Universal Statistical Test for Random Bit Generators, " Advances in Cryptology CRYPTO '90, Proceedings, Springer-Verlag, 1991, pp. 409-420.

1032. U.M. Maurer, "A Universal Statistical Test for Random Bit Generators," Journal of Cryptology, v. 5, n. 2, 1992, pp. 89-106.

1033. U.M. Maurer and J.L. Massey, "Cascade Ciphers: The Importance of Being First," Journal of Cryptology, v. 6, n. 1, 1993, pp. 55-61.

1034. U.M. Maurer and J.L. Massey, "Perfect Local Randomness in Pseudo-Random Sequences, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 110-112.

1035. U.M. Maurer and Y. Yacobi, "Non interactive Public Key Cryptography, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 498-507.

1036. G. Mayhew, "A Low Cost, High Speed Encryption System and Method," Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, 1994, pp. 147-154.

1037. G. Mayhew, R. Frazee, and M. Bianco, "The Kinetic Protection Device, " Proceedings of the 15th National Computer Security Conference, NIST, 1994, pp. 147-154.

1038. K.S. McCurley, "A Key Distribution System Equivalent to Factoring," Journal of Cryptology, v. 1, n. 2, 1988, pp. 95-106.

1039. K.S. McCurley, "The Discrete Logarithm Problem," Cryptography and Computational Number Theory (Proceedings of the Symposium on Applied Mathematics), American Mathematics Society, 1990, pp. 49-74.

1040. K.S. McCurley, open letter from the Sandia National Laboratories on the DSA of the NIST, 7 Nov 1991.

1041. R.J. McEliece, "A Public-Key Cryptosystem Based on Algebraic Coding Theory," Deep Space Network Progress Report 42-44, Jet Propulsion Laboratory, California Institute of Technology, 1978, pp. 114-116.

1042. R.J. McEliece, Finite Fields for Computer Scientists and Engineers, Boston: Kluwer Academic Publishers, 1987.

1043. P. McMahon, "SESAME V2 Public Key and Authorization Extensions to Kerberos, " Proceedings of the Internet Society 1Y95 Symposium on Network and Distributed Systems Security, IEEE Computer Society Press, 1995, pp. 114-131.

1044. C.A. Meadows, "A System for the Specification and Analysis of Key Management Protocols," Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, 1991, pp. 182-195.

1045. C.A. Meadows, "Applying Formal Methods to the Analysis of a Key Management Protocol," Journal of Computer Security. v. I, n. 1, 1992,pp.5-35.

1046. C.A. Meadows, "A Model of Computation for the NRL Protocol Analyzer, " Proceedings of the Computer Security Foundations Workshop VII, IEEE Computer Society Press, 1994, pp. 84-89.

1047. C.A. Meadows, "Formal Verification of Cryptographic Protocols: A Survey," Advances in Cryptology ASIACRYPT '94 Proceedings, Springer-Verlag, 1995, pp. 133-150.

1048. G. Medvinsky and B.C. Neuman, "Net Cash: A Design for Practical Electronic Currency on the Internet," Proceedings of the 1st Annual ACM Conference on Computer and Communications Security, ACM Press, 1993, pp. 102-106.

1049. G. Medvinsky and B.C. Neuman, "Electronic Currency for the Internet," Electro nic Markets, v 3, n. 9/10, Oct 1993, pp. 23-24.

1050. W. Meier, "On the Security of the IDEA Block Cipher," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer -Verlag, 1994, pp. 371-385.

1051. W. Meier and O. Staffelbach, "Fast Correlation Attacks on Stream Ciphers," Journal of Cryptology v I n. 3, 1989, pp. 159-176.

1052. W. Meier and O. Staffelbach, "Analysis of Pseudo Random Sequences Generated by Cellular Automata, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 186-199.

1053. W. Meier and O. Staffelbach, "Correlation Properties of Combiners with Memory in Stream Ciphers, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 204-213.

1054. W. Meier and O. Staffelbach, "Correlation Properties of Combiners with Memory in Stream Ciphers, " Journal of Cryptology, v. 5, n. 1, 1992, pp. 67-86.

1055. W. Meier and O. Staffelbach, "The Self-Shrinking Generator," Communications and Cryptography: Two Sides of One Tapestry R.E. Blahut et al., eds., Kluwer Adademic Publishers, 1994, pp. 287-295.

1056. J. Meijers, "Algebraic-Coded Cryptosystems," Master's thesis, Technical University Eindhoven, 1990.

1057. J. Meijers and J. van Tilburg, "On the Rao-Nam Private-Key Cryptosystem Using Linear Codes," International Symposium on Information Theory, Budapest, Hun gary, 1991.

1058. J. Meijers and J. van Tilburg, "An Improved 5T-Attack on the Rao-Nam Private-Key Cryptosystem," International Conference on Finite Fields, Coding Theory, and Advances in Communications and Computing, Las Vegas, NV, 1991.

1059. A. Menezes, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Pub lishers, 1993.

1060. A. Menezes, ed., Applications of Finite Fields, Kluwer Academic Publishers, 1993.

1061. A. Menezes and S.A. Vanstone, "Elliptic Curve Cryptosystems and Their Implementations," Journal of Cryptology, v. 6, n. 4, 1993, pp. 209-224.

1062. A. Menezes and S.A. Vanstone, "The Implementation of Elliptic Curve Cryptosystems, " Advances in Cryptology AUSCRYPT '90 Proceedings, Springer-Verlag, 1990, pp. 2-13.

1063. R. Menicocci, "Short Gollmann Cascade Generators May Be Insecure," Codes and Ciphers, Institute of Mathematics and its Applications, 1995, pp. 281-297.

1064. R.C. Merkle, "Secure Communication Over Insecure Channels," Communications of the ACM, v. 21, n. 4, 1978, pp. 294-299.

1065. R.C. Merkle, "Secrecy, Authentication, and Public Key Systems," Ph.D. dissertation, Stanford University, 1979.

1066. R.C. Merkle, "Method of Providing Digital Signatures," U.S. Patent #4,309,569, 5 Jan 1982

1067. R.C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 369-378.

1068. R.C. Merkle, "A Certified Digital Signature," Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 218-238.

1069. R.C. Merkle, "One Way Hash Functions and DES, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 428 446.

1070. R.C. Merkle, "A Fast Software One-Way Hash Function," Journal of Cryptology, v. 3, n. 1, 1990, pp. 43-58.

1071. R.C. Merkle, "Fast Software Encryption Functions, " Advances in Cryptology CRYPTO '90, Proceedings, Springer-Verlag, 1991, pp. 476-501.

1072. R.C. Merkle, "Method and Apparatus for Data Encryption," U.S. Patent #5,003,597, 26 Mar 1991.

1073. R.C. Merkle, personal communication, 1993.

1074. R.C. Merkle and M. Hellman, "Hiding information and Signatures in Trapdoor Knapsacks," IEEE Transactions on Information Theory, v. 24, n. 5, Sep 1978, pp. 525-530.

1075. R.C. Merkle and M. Hellman, "On the Security of Multiple Encryption," Communications of the ACM, v. 24, n. 7, 1981, pp. 465-467.

1076. M. Merritt, "Cryptographic Protocols," Ph.D. dissertation, Georgia Institute of Technology, GIT-ICS-83/6, Feb 1983.

1077. M. Merritt, "Towards a Theory of Cryptographic Systems: A Critique of Crypto Complexity," Distributed Computing and Cryptography, J. Feigenbaum and M. Merritt, eds., American Mathematical Society, 1991, pp. 203-212.

1078. C.H. Meyer, "Ciphertext/Plaintext and Ciphertext/Key Dependencies vs. Number of Rounds for Data Encryption Standard," AFIPS Conference Proceedings, 47, 1978, pp. 1119-1126.

1079. C.H. Meyer, "Cryptography A State of the Art. Review, " Proceedings of CompEuro '89, VLSI and Computer Peripherals, 3rd Annual European Computer Conference, IEEE Press, 1989, pp. 150-154.

1080. C.H. Meyer and S.M. Matyas, Cryptography: A New Dimension in Computer Data Security, New York: John Wiley & Sons, 1982.

1081. C.H. Meyer and M. Schilling, "Secure Program Load with Manipulation Detection Code, " Proceedings of Securicom '88, 1988, pp. 111-130.

1082. C.H. Meyer and W.L. Tuchman, "Pseudo-Random Codes Can Be Cracked, " Electronic Design, v. 23, Nov 1972.

1083. C.H. Meyer and W.L. Tuchman, "Design Considerations for Cryptography, " Proceedings of the NCC, v. 42, Montvale, NJ: AFIPS Press, Nov 1979, pp. 594-597.

1084. S. Micali, "Fair Public-Key Cryptosystems, " Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 113-138.

1085. S. Micali, "Fair Cryptosystems," MIT/LCS/TR-579.b, MIT Laboratory for Computer Science, Nov 1993.

1086. S. Micali, "Fair Cryptosystems and Methods for Use," U.S. Patent #5,276,737, 4 Jan 1994.

1087. S. Micali, "Fair Cryptosystems and Methods for Use," U.S. Patent #5,315,658, 24 May 1994.

1088. S. Micali and A. Shamir, "An Improvemcnt on the Fiat-Shamir Identification and Signature Scheme," Advances in Cryptollgy CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 244-247.

1089. M.J. Mihajlevic, "A Correlation Attack on the Binary Sequence Generators with Time-Varying Output Function, " Advances in Cryptology ASIACRYPT'94 , Proceedings, Springer-Verlag, 1995, pp. 67-79.

1090. M.J. Mihajlevic and J.D. Golic, "A Fast Iterative Algorithm for a Shift Register Internal State Reconstruction Given the Noisy Output Sequence, " Advances in Cryptology AUSCRYPT '90 Proceedings, Springer-Verlag, 1990, pp. 165-175.

1091. M.J. Mihajlevic and J.D. Golic, "Convergence of a Bayesian Iterative Error-Correction Procedure to a Noisy Shift Register Sequence," Advances in Cryptology , EUROCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 124-137.

1092. J.K. Millen, S.C. Clark, and S.B. Freedman, "The Interrogator: Protocol Security Analysis," IEEE Transactions on Software Engineering, v. SE-13, n.2, Feb 1987, pp.274 - 288.

1093. G.L. Miller, "Riemann's Hypothesis and Tests for Primality," Journal of Computer Systems Science, v. 13, n. 3, Dec 1976, pp. 300-317.

1094. S.R Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer, "Section E.2.1: Kerberos Authentication and Authorization System," MIT Project Athena, Dec 1987.

1095. V.S. Miller, "Use of Elliptic Curves in Cryptography, " Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 417-426.

1096. M. Minsky, Computation: Finite and Infinite Machines, Englewood Cliffs, NJ: Prentice-Hall, 1967.

1097. C.J. Mitchell, "Authenticating Multi-Cast Internet Electronic Mail Messages Using a Bidirectional MAC Is Insecure, " draft manuscript, 1990.

1098. C.J. Mitchell, "Enumerating Boolean Functions of Cryptographic Significance," Journal of Cryptology, v. 2, n. 3, 1990, pp. 155-170.

1099. C.J. Mitchell, F. Piper, and P. Wild, "Digital Signatures, " Contemporary Cryptology: The Science of Information Integtity, G.J. Simmons, ed., IEEE Press, 1991, pp. 325-378.

1100. C.J. Mitchell, M. Walker, and D. Rush, "CCITT/ISO Standards for Secure Messagc Handling," IEEE Journal on Selected Areas in Communications, v. 7, n. 4, May 1989, pp. 517-524.

1101. S. Miyaguchi, "Fast Encryption Algorithm for the RSA Cryptographic System," Proceedings of Compcon 82, IEEE Press, pp. 1115. 672-678.

1102. S. Miyaguchi, "The FEAL-8 Cryptosystem and Call for Attack, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 624-627.

1103. S. Miyaguchi, "Expansion of the FEAL Cipher," NTT Review, v. 2, n. 6, Nov 1990.

1104. S. Miyaguchi, "The FEAL Cipher Family," Advances in Cryptology CKYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 627-638.

1105. S. Miyaguchi, K. Ohta, and M. Iwata, " 128- bit Hash Function IN-Hashl," Proceedings of SECURICOM '90, 1990, pp. 127-137.

1106. S. Miyaguchi, K. Ohta, and M. Iwata, " 128- bit Hash Function (N-Hash)," NTT Review, v. 2, n. 6, Nov 1990, pp. 128-132.

1107. S. Miyaguchi, K. Ohta, and M. Iwata, "Confirmation that Some Hash Functions Are Not Collision Free," Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 326-343.

1108. S. Miyaguchi, A. Shiraishi, and A. Shimizu, "Fast Data Encipherment Algorithm FEAL-8," Review of tile Electrical Communication Laboratories, v. 36, n. 4, 1988.

1109. H. Miyano, "Differential Cryptanalysis on CALC and Its Evaluation," Proceedings of the 1992 Symposium on Cryptography and Information Security ISCIS 92, Tateshina, Japan, 2-4 Apt 1992, pp. 7B.1-8.

1110. R. Molva, G. Tsudik, E. van Hcrreweghen, and S. Zatti, "KryptoKnight Authentication and Key Distribution System," Proceedings of European Symposium on Research in completer Security, Toulouse, France, Nov 1992.

1111. P.L. Montgomery, "Modular Multiplication without Trial Division," Mathematics of computation, v. 44, n. 170, 1985, pp. 51Y-521.

1112. RL. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," Mathematics of Computation, v.48, n. 177, Jan 19R7, pp. 243-264.

1113. P.L. Montgomery and R. Silverman, "An FFT Extension to the p- l Factoring Algorithm," Mathematics of Computation, v. 54, n. 190, 1990, pp. 839-854.

1114. J.H. Moore, "Protocol Failures in Cryptosystems," Proceedings of the IEEE, v. 76, n. 5, May 1988.

1115. J.H. Moore, "Protocol Failures in Cryptosystems," in Contemporary Cryptology: The Science of Information Integrity, G.J. Simmons, ed., IEEE Press, 1992, pp. 541-558.

1116. J.H. Moore and G.J. Simmons, "Cycle Structure of the DES with Weak and Semi -Weak Keys, " Advances in Cryptology CRYPTO '86 Proceedings, SpringerVerlag, 1987, pp. 3-32.

1117. T. Moriyasu, M. Mor iai, and M. Kasahara, "Nonlinear Pseudorandom Number Generator with Dynamic Structure and Its Properties," Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94), Biwako, Japan, 27-29 Jan 1994, pp. 8A.l-ll.

1118. R. Morris, "The Data Encryption Standard Retrospective and Prospects," IEEE Communications Magazine, v. 16, n. 6, Nov 1978, pp. 11-14.

1119. R. Morris, remarks at the 1993 Cambridge Protocols Workshop, 1993.

1120. R. Morris, N.J.A. Sloane, and A.D. Wyner, "Assessment of the NBS Proposed Data Encryption Standard," Cryptologia, v. 1, n. 3, Jul 1977, pp. 281-2 91.

1121. R. Morris and K. Thompson, "Password Security: A Case History," Communications of the ACM, v. 22, n. 11, Nov 1979, pp. 594-597.

1122. S.B. Morris, "Escrow Encryption," lecture at MIT Laboratory for Computer Science, 2 Jun 1994.

1123. M.N. Morrison and J. Brillhart, "A Method of Factoring and the Factorization of F7," Mathematics of Computation, v. 29, n. 129, Jan 1975, pp. 183-205.

1124. L.E. Moser, "A Logic of Knowlcdgc and Belief for Reasoning About Computer Security, "Proceedings of the Computer Security Foundations Workshop 11, IEEE Computer Society Press, 1989, pp. S7 63.

1125. Motorola Government Electronics Division, Advanced Techniques i n Network security' Scottsdale, AZ, 1977.

1126. W.B. Muller, "Polynomial Functions in Modern Cryptology," contrib utions to General Algebra 3: Proceedings of the Vienna Conference, Vienna: Verlag H older-Pichler-Tempsky' 1985, pp. 7-32.

1127. W.B. Muller and W. Nobauer, "Some Remarks on Public-Key Cryptography, " Studia Scientiarum Mathematicarum Hungarica, v. 16, 1981, pp. 71-76.

1128. W.B. Muller and W. Nobauer, "Cryptanalysis of the Dickson Scheme," Advances in Cryptology EUROCRYPT '85 Proceedings, Springer-Verlag, 1986, pp. 50-61.

1129. C. Muller-Scholer, "A Microprocessor-Based Cryptoprocessor," IEEE Micro, Oct 1983, pp. 5-15.

1130. R.C. Mullin, E. Nemeth, and N. Weidenhofer, "Will Public Kcy Cryptosystems Live Up to Their Expectations? HEP Implementation of the Discrete Log Codebreaker," ICPP 85, pp. 193-196.

1131. Y. Murakami and S. Kasahara, "An ID-Based Key Distribution Scheme, " IEICE Japan, Technical Report, ISEC90-26, 1990.

1132. S. Murphy, "The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts, " Journal of Cryptology, v. 2, n. 3, 1990, pp. 145-154.

1133. E.D. Mycrs, "STU-III Multilevel Secure Computer Interface," Proceedings of the Tenth Annual Computer Security Applications Conference, IEEE Computer Society Press, 1994, pp. 170-179.

1134. D. Naccache, "Can O.S.S. be Repaired? Proposal for a New Practical Signature Scheme," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 233-239.

1135. D. Naccache, D. M'Raihi, D. Raphacli, and S. Vaudenay, "Can D.S.A. be Improved: Complexity Trade-Offs with the Digital Signature Standard, " Advances in Cryptology EUKOCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

1136. Y. Nakao, T. Kaneko, K. Koyama, and R. Terada, "A Study on the Security of RDES-Cryptosystem against Linear Cryptanalysis," Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography, Inuyama, Japan, 24 -27 Jan 1995, pp. 163-172.

1137. M. Naor, "Bit Commitmcnt Using Pseudo-Randomness," Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 128-136.

1138. M. Naor and M. Yung, "Universal One-Way Hash Functions and Their Cryptographic Application," Proceedings of the 21stAnnual ACM Symposium on the Theory of Computing, 1989, pp. 33 43.

1139. National Bureau of Standards, "Report of the Workshop on Estimation of Significant Advances in Computer Technology, " NBSIR 76-1189, National Bureau of Standards, U.S. Department of Commercc, 21-22 Sep 1976, Dec 1977.

1140. National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard, " National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

1141. National Bureau of Standards, NBS FIPS PUB 46-1, "Data Encryption Standard," U.S. Department of Commerce, Jan 1988.

1142. National Bureau of Standards, NBS FIPS PUB 74, "Guidelines for Implementing and Using the NBS Data Encryption Standard, " U.S. Department of Commerce, Apr 1981.

1143. National Bureau of Standards, NBS FIPS PUB 81, "DES Modes of Operation," U.S. Department of Commerce, Dec 1980.

1144. National Bureau of Standards, NBS FIPS PUB 112, "Password Usage," U.S. Department of Commerce, May 1985.

1145. National Bureau of Standards, NBS FIPS PUB 113, "Computer Data Authentication," U.S. Department of Commerce, May 1985.

1146. National Computer Security Center, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria," NCSC-TG-005 Version 1, Jul 1987.

1147. National Computer Security Centcr, "Trusted Datahase Management System Interpretation of the Trusted Computer System Evaluation Criteria, " NCSC-TG-021 Version 1, Apr 1 991.

1148. National Computer Security Center, "A Guide to Understanding Data Rememberance in Automated Information Systems," NCSC-TG-025 Version 2, Sep 1991.

1149. National Institute of Standards and Technology, NIST FIPS PUB XX, "Digital Signature Standard," U.S. Department of Commcrce, DRAFT, 19 Aug 1991.

1150. National Institute of Standards and Technology, NIST FIPS PUB 46-2, "Data Encryption Standard," U.S. Departm ent of Commcrcc, Dec 93.

1151. National Institute of Standards and Technology, NIST FIPS PUB 171, "Key Management Using X9.17," U.S. Departmcnt of Commcrce, Apr 92.

1152. National Institute of Standards and Technology, NIST FIPS PUB 180, "Secure Hash Standard, " U.S. Department of Commerce, May 93.

1153. National Institute of Standards and Technology, NIST FIPS PUB 185, "Escrowed Encryption Standard," U.S. Department of Commerce, Feb 94.

1154. National Institute of Standards and Technology, NIST FIPS PUB 186, "Digital Signature Standard, " U.S. Department of Commerce, May 1994.

1155. National Institute of Standards and Technology," Clipper Chip Technology," 30 Apr 1993.

1156. National Institute of Standards and Technology," Capstone Chip Technology," 30 Apr 1993.

1157. J. Nechvatal, "Public Key Cryptography, " NIST Special Publication 800-2, National Institute of Standards and Technology, U.S. Department of Commerce, Apr 1991.

1158. l. Nechvatal, "Public Key Cryptography," Contemporary Cryptology: The Science of Information Integrity, G.J. Simmons, ed., IEEE Press, 1992, pp. 177-288.

1159. R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," Communications of the ACM, v. 21, n. 12, Dec 1978, pp. 993-999.

1160. R.M. Ncedham and M.D. Schroeder, "Authentication Revisited," Operating Systems Review, v. 21, n. 1, 1987, p. 7.

1161. D.M. Nessett, "A Critique of the Burrows, Abadi, and Needham Logic," Operating System Review, v. 20, n. 2, Apr 1990, pp. 35-38.

1162. B.C. Ncuman and S. Stubblebine, "A Note on the Use of Timestamps as Nonces, " Operating Systems Review, v. 27, n. 2, Apr 1993, pp. 10-14.

1163. B.C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," IEEE Communications Magazine, v. 32, n. 9, Sep 1994, pp. 33-38.

1164. L. Neuwirth, "Statement of Lee Nenwirth of Cylink on HR145," submitted to congressional committees considering HR145, Feb 1987.

1165. D.B. Newman, Jr. and R.L. Pickholtz, "Cryptography in the Private Sector," IEEE Communications Magazine, v. 24, n. 8, Aug 1986, pp.7-10.

1166. H. Niederreiter, "A Public-Key Cryptosystem Based on Shift Register Sequences," Advances in Cryptology EZJROCRYPT '85 Proceedings, Springer-Verlag, 1986, pp. 35-39.

1167. H. Niederreiter, "Knapsack-Type Cryptosystems and Algebraic Coding Theory," Problems of Control and Information Theory, v. 15, n. 2, 1986, pp. 159-166.

1168. H. Niederreiter, "The Linear Complexity Profile and the Jump Complexity of Keystream Sequences, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 174-188.

1169. V. Niemi, "A New Trapdoor in Knapsacks," Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 405 -411.

1170. V. Niemi and A. Renvall, "How to Prevent Buying of Voters in Computer Elections," Advances in Cryptology ASIACRYPT '94 Proceedings, Springer-Verlag, 1995, pp. 164-170.

1171. I. Niven and H.A. Zuckerman, An Introduction to the Theory of Numbers, New York: John Wiley & Sons, 1972.

1172. R. Nobauer, "Cryptanalysts of the Redei Scheme," Contributions to General Algebra 3: Proceedings of the Vienna Conference, Verlag Holder-Pichler-Tempsky, Vienna, 1985, pp. 255-264.

1173. R. Nobauer, "Cryptanalysts of a Public- Key Cryptosystem Based on Dickson-Polynomials," Mathematica Slovaca, v. 38, n. 4, 1988, pp. 309-323.

1174. K. Nogochi, H. Ashiya, Y. Sano, and T. Kaneko, "A Study on Differential Attack of MBAL Cryptosystem," Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS' 94), Lake Biwa, Japan, 27-29 Jan 1994, pp. 14B.1-7. (In Japanese.)

1175. H. Nurmi, A. Salomaa, and L. Santean, "Secret Ballot Elections in Computer Networks," Computers & Security, v. 10, 1991, pp. 553-560.

1176. K. Nyberg, "Construction of Bent Functions and Difference Sets," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 151-160.

1177. K. Nyberg, "Perfect Nonlinear S-Boxes," Advances in Cryptology EUROCRYPT '91 Proceedings. Springcr-Verlag, 1991, pp. 378-386.

1178. K. Nyberg, "On the Construction of Highly Nonlinear Permutations, " Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag 1991, pp. 92-98.

1179. K. Nyberg, "Differentially Uniform Mappings for Cryptography," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 55-64.

1180. K. Nyberg, "Provable Security against Differential Cryptanalysis," presented at the rump session of Eurocrypt '94, May 1994.

1181. K. Nyberg and L.R. Knudsen, "Provable Secu rity against Differential Cryptanalysis," Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 566-574.

1182. K. Nyberg and L.R. Knudsen, "Provable Security against Differential Cryptanalysis," Journal of Cryptology, v. 8, n. 1, 1995, pp. 27-37.

1183. K. Nyberg and R.A. Rueppel, "A New Signature Scheme Based on the DSA Giving Message Recovery," 1st ACM Conference on Computer and Communications Secu rity, ACM Press, 1993, pp. 58-61.

1184. K. Nyberg and R.A. Rueppel, "Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem," Advances in Cryptology EUROCRYPT '94 Proceedings, Spnnger-Verlag, 1995, to appear.

1185. L. O'Connor, "Enumerating Nondegenerate Permutations," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 368-377.

1186. L. O'Connor, "On the Distribution of Characteristics in Bijective Mappings, " Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 360-370.

1187. L. O'Connor, "On the Distribution of Characteristics in Composite Permutations, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 403-412.

1188. L. O'Connor and A. Klapper, "Algebraic Nonlinearity and Its Application to Cryp- tography," Journal of Cryptology, v. 7, n.3, 1994, pp. 133-151.

1189. A. Odlyzko, "Discrete Logarithms in Finite Fields and Their Cryptographic Sig nificance," Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, pp. 224-314.

1190. A. Odlyzko, "Progress in Integer Factorization and Discrete Logarithms, " unpublished manuscript, Feb 1995.

1191. Office of Technology Assessment, U.S. Congress, "Defending Secrets, Sharing Data: New Locks and Keys for Electronic Communication, " OTA-CIT-310, Washington, D.C.: U.S. Government Printing Office, Oct 1987.

1192. B. O'Higgins, W. Diffie, L. Strawczynski, and R. de Hoog, "Encryption and ISDN a Natural Fit," Proceedings of the 1987 International Switching Symposium. 1987, pp. 863-869.

1193. Y. Ohnishi, "A Study on Data Security," Master's thesis, Tohuku University, Japan, 1988. (In Japanese.)

1194. K. Ohta, "A Secure and Efficient Encrypted Broadcast Communication System Using a Public Master Key," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. J70-D, n. 8, Aug 1987, pp. 1616-1624.

1195. K. Ohta, "An Electrical Voting Scheme Using a Single Administrator, " IEICE Sp ring National Convention, A-294, 1988, v. 1, p. 296. (In Japanese.)

1196. K. Ohta, "Identity-based Authentication Schemes Using the RSA Cryptosystem," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. J72D-II, n. 8, Aug 1989, pp. 612-620.

1197. K. Ohta and M. Matsui, "Differential Attack on Message Authentication Codes," Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 200-223.

1198. K. Ohta and T. Okamoto, "Practical Extension of Fiat-Shamir Scheme," Electronics Letters, v. 24, n. 15, 1988, pp. 955-956.

1199. K. Ohta and T. Okamoto, "A Modification of the Fiat-Shamir Scheme," Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 232-243.

1200. K. Ohta and T. Okamoto, "A Digital Multisignature Scheme Bascd on the Fiat-Shamir Scheme, " Advances in Cryptology ASIA CRYPT '91 Rroceedillys, Springer-Verlag, 1993, pp. 139-148.

1201. K. Ohta, T. Okamoto and K. Koyama, "Membership Authentication for Hierarchy Multigroups Using thc Extended Fiat - Shamir Scheme, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 446-457.

1202. E. Okamoto and K. Tanaka, "Key Distribution Based on Identification Information, " IEEE journal on Selected Areas in Com- munication, v. 7, n. 4, May 1989, pp. 481-4 90.

1203. T. Okamoto, "Fast Public-Key Cryptosystems Using Congruent Polynomial Equa tions, " Electronics Letters, v. 22, n. 11, 1986, pp. 581-582.

1204. T. Okamoto, "Modification of a Public -Kcy Cryptosystem," Electronics Letters, v. 23, n. 16, 1987, pp. 814-81S.

1205. T. Okamoto, "A Fast Signature Scheme Based on Congruential Polynomial Operations," IEEE Transactions on Information Theory, v. 36, n. 1, 1990, pp. 47-53.

1206. T. Okamoto, "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes, " Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 31-53.

1207. T. Okamoto, A. Fujioka, and E. Fujisaki, "An Efficient Digital Signature Scheme Based on Elliptic Curve over the Ring Z/p" Advances in Cryptology CRYPTO '92 Proceeding.s, Springer-Verlag, 1993, pp. 54-65.

1208. T. Okamoto, S. Miyaguchi, A. Shiraishi, and T. Kawoaka, "Signed Document Transmission System," U.S. Patent #4,625,076, 25 Nov 1986.

1209. T. Okamoto and K. Ohta, ``Disposablc Zero-Knowlcdge Authentication and Their Applications to Untraceable Electronic Cash," Advances in Cryptology CRYPT O '89 Proceedings, Springer-Verlag, 1990, pp. 134-149.

1210. T. Okamoto and K. Ohta, "How to Utilize the Randomness of zero-Knowlcdgc Proofs," Advances in Cryptology CRYPTO '90 Proceedings, Springcr-Verlag, 1991, pp. 456 475.

1211. T. Okamoto and K. Ohta, "Universal Electronic Cash," Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 324-337.

1212. T. Okamoto and K. Ohta, "Survey of Digital Signature Schemes, " Proceedings of the Third Symposium on State and Progress of Research in Cryptography, Fon dazone Ugo Bordoni, Rome, 1993, pp. 17-29.

1213. T. Okamoto and K. Ohta, "Designated Confirmer Signatures Using Trapdoor Functions," Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94), Lake Biwa, Japan, 27-29 Jan 1994, pp. 16B.l-ll.

1214. T. Okamoto and K. Sakurai, "Efficient Algorithms for the Construction of Hyper-elliptic Cryptosystems," Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 267 278.

1215. T. Okamoto and A.Shiraishi, "A Fast Signature Scheme Based on Quadratic Inequalities," Proceedings of the 1985 Symposium on Security and Privacy, IEEE, Apr 1985, pp. 123-132.

1216. J.D. Olsen, R.A. Scholtz, and L. Welch, "Bent Function Sequences," IEEE Transactions on Information Theory, v. IT-28, n. 6, Nov 1982, pp. 858-864.

1217. H. Ong and C.P Schnorr, "Signatures through Approximate Representations by Quadratic Forms," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984.

1218. H. Ong and C.R Schnorr, "Fast Signature Generation with a Fiat Shamir-Like Scheme," Advances in Cryptology EUKOCKYPT '9() Proceedings, Springer-Verlag, 1991, pp. 432-440.

1219. H. Ong, C.R Schnorr, and A. Shamir, "An Efficient Signature Scheme Based on Polynomial Equations, " Proceedings of the 16th Annual Symposium on the Theory of Computing, 1984, pp. 208 216.

1220. H. Ong, C.P. Schnorr, and A. Shamir, "Efficient Signature Schemes Based on Polynomial Equations," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 37-46.

1221. Open Shop Information Services, OSIS Security Aspects, OS1.S European Working Group, WGI, final report, Oct 1985.

1222. G.A. Orton, M.R Roy, PA. Scott, L.E. Peppard, and S.E. Tavares, "VLSI Implementa tion of Public-Key Encryption Algorithms, " Advances in Cryptology CRYPTO '86 Proceedings, Springcr-Verlag, 1987, pp. 277-301.

1223. H. Orup, E. Svendsen, and E. Anclreasen, "VICTOR An Efficient RSA Hardware Implementation," Advances in Cryptol ogy EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 245-252.

1224. D. Otway and O. Rees, "Efficient and Timely Mutual Authentication'" Operating Systems Review, v. 21, n. 1, 1987, pp. 8-10.

1225. G. Pagels-Fick, " Implementation Issues for Master Key Distribution and Protected Keyload Procedures, " Computers and Security: A Global Challenge, Proceedings of IFIP/SEC '83, North Holland: Elsevier Science Publishers, 1984, pp. 381-390.

1226. C.M. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.

1227. C.S. Park, "Improving Code Rate of McEliece's Public-key Cryptosystem, " Electronics Letters, v. 25, n. 21, 12 Oct 1989, pp. 1466-1467.

1228. S. Park, Y. Kim, S. Lee, and K. Kim, "Attacks on Tanaka's Non-interactive Key Sharing Scheme," Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95), Inuyama, Japan, 24-27 Jan 1995, pp. B3.4.1-4.

1229. S.J. Park, K.H. Lee, and D.H. Won, "An Entrusted Undeniable Signature, " Proceedings of the 1995 Iapan-Korea Workshop on Information Secunty and Cryptography, Inuyama, Japan, 24-27 Jan 1995, pp. 120-126.

1230. S.J. Park, K.H. Lee, and D.H. Won, "A Practical Group Signature," Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography, Inuyama, Japan, 24-27 Jan 1995, pp. 127-133.

1231. S.K. Park and K.W. Miller, "Random Number Generators: Good Ones Are Hard to Find," Communications of the ACM, v. 31, n. 10, Oct 1988, pp. 1192-1201.

1232. J. Patarin, "How to Find and Avoid Collisions for the Knapsack Hash Function," Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag 1994, pp. 305-317.

1233. W. Patterson, Mathematical Cryptology for Computer Scientists and Mathematicians, Totowa, N.J.: Rowman & Littlefield, 1987.

1234. W.H. Payne, "Public Key Cryptography Is Easy to Break," William H. Payne, unpublished manuscript, 16 Oct 90.

1235. T.R Pederson, "Distributed Provers with Applications to Undeniable Signatures, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag 1991, pp. 221-242.

1236. S. Peleg and A. Rosenfield, "Breaking Substitution Ciphers Using a Relaxation Algorithm " Communications of the ACM, v. 22, n. 11, Nov 1979, pp. 598-605.

1237. R. Peralta, "Simultaneous Security of Bits in the Discrete Log " Advances in Cryptology EUROCRYPT '85, Springer-Verlag, 1986, pp. 62-72.

1238. I. Peterson, "Monte Carlo Physics: A Cautionary Lesson," Science News, v. 142, n. 25, 19 Dec 1992, p. 422.

1239. B. Pfitzmann, "Fail-Stop Signatures: Principles and Applications," Proceedings of COMPUSEC '91, Eighth World Conference on Computer Security, Audit, and Control, Elsevier Science Publishers, 1991, pp. 125-134.

1240. B. Pfitzmann and M. Waidner, "Formal Aspects of Fail-Stop Signatures," Fakultat fur Informatik, University Karlsruhe, Report 22/90, 1990.

1241. B. Pfitzmann and M. Waidner, "Fail-Stop Signatures and Their Application, " Securicom '91,1991, pp. 145-160.

1242. B. Pfitzmann and M. Waidner, "Unconditional Concealment with Cryptographic Ruggedness," VIS '91 Verlassliche Informationsysteme Proceedings, Darmstadt, Germany, 13-15 March 1991, pp. 3-2-320. (In German.)

1243. B. Pfitzmann and M. Waidner, "How to Break and Repair a 'Provably Secure' Untraceable Payment System," Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, 1992, pp. 338-350.

1244. C.R Pfleeger, Security in Computing, Englewood Cliffs, N.J.: Prentice-Hall, 1989.

1245. S.J.D. Phoenix and RD. Townsend, "Quantum Cryptography and Secure Optical Communication," BT Technology Journal, v. 11, n. 2, Apr 1993, pp. 65-75.

1246. J. Pieprzyk, "On Public-Key Cryptosystems Built Using Polynomial Rings, " Advances in Cryptology EUROCRYPT '85, Springer-Verlag 1986, pp. 73-80.

1247. J. Pieprzyk, "Error Propagation Property and Applications in Cryptography, " IKE Proceedings-E, Computers and Digital Techniques, v. 136, n. 4, Jul 1989, pp. 262-270.

1248. D. Pinkas, T. Parker, and R Kaijser, "SESAME: An Introduction," Issue 1.2, Bull, JCL, and SNI, Sep 1993.

1249. F. Piper, "Stream Ciphers," Elektrotechnic und Maschinenbau, v. 104, n. 12, 1987, pp. 564-668.

1250. V.S. Pless, "Encryption Schemes for Computer Confidentiality," IEEE Transactions on Computing, v. C-26, n. 11, Nov 1977, pp. 1133-1136.

1251. J.B. Plumstead, "Inferring a Sequence Generated by a Linear Congruence," Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science, 1982, pp. 153-159.

1252. R. Poet, "The Design of Special Purposc Hardware to Factor Large Integers, " Computer Physics Communications, v. 37, 1985, pp. 337-341.

1253. S.C. Pohlig and M.E. Hellman, "An Improved Algorithm for Computing Logarithms in GF(p) and Its Cryptographic Significance," IEEE Transactions on Information Theory, v. 24, n. 1, Jan 1978, pp. 106-111.

1254. J.M. Pollard. "A Monte Carlo Method for Factorization," BIT v. 15, 1975, pp.331-334.

1255. J.M. Pollard and C.P. Schnorr, "An Efficient Solution of the Congruence x+ky= m (mod n)" IEEE Transactions on Infor- mation Theory, v. IT-33, n. 5, Sep 1987, pp. 702-709.

1256. C. Pomerance, "Recent Developments in Primality Testing," The Mathematical Intelligencer, v. 3, n. 3, 1981, pp. 97-105.

1257. C. Pomerance, "The Quadratic Sieve Factoring Algorithm," Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, 169-182.

1258. C. Pomerance, "Fast, Rigorous Factorization and Discrete Logarithm Algorithms," Discrete Algorithms and Complexity, New York: Academic Press, 1987, pp. 119-143.

1259. C. Pomerance, I W. Smith, and R. Tuler, "A Pipe-Line Architecture for Factoring Large Integers with the Quadratic Sieve Algorithm," SIAM lournal on Computing, v.17, n.2, Apr l988, pp. 387-403.

1260. G.J. Popek and C.S. Kline, "Encryption and Secure Computer Networks," ACM Computing Surveys, v 11, n. 4, Dec 1979, pp. 331-356.

1261. F. Pratt, Secret and Urgent, Blue Ribbon Books, 1942.

1262. B. Preneel, "Analysis and Design of Cryptographic Hash Functions, " Ph.D. dissertation, Katholieke Universiteit Leuven, Jan 1993.

1263. B. Preneel, "Differential Cryptanalysis of Hash Functions Based on Block Ciphers, " Proceedings of the 1st ACM Conference on Computer and Communications Security, 1993, pp. 183-188.

1264. B. Preneel, "Cryptographic Hash Functions," European Transactions on Telecommunications, v 5, n. 4, Jul/Aug 1994, pp. 431 -448.

1265. B. Preneel, personal communication, 1995.

1266. B. Preneel, A. Bosselaers, R. Govaerts, and J. Vandewalle, "Collision-Free Hash Functions Based on Block Cipher Algorithms," Proceedings of the 1989 Carnahan Conference on Security Technology 1989, pp. 203-210.

1267. B. Preneel, R. Govaerts, and J. Vandewalle, "An Attack on Two Hash Functions by Zheng-Matsumoto-Imai, " Advances in Cryptology ASIACRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 535-538.

1268. B. Preneel, R. Govaerts, and J. Vandewalle, "Hash Functions Based on Block Ciphers: A Synthetic Approach, " Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp.368-378.

1269. B. Preneel, M. Nuttin, V. Rijmen, and J. Buelens, "Cryptanalysts of the CFB mode of the DES with a Reduced Number of Rounds," Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 212-223.

1270. B. Preneel and V. Rijmen, "On Using Maximum Likelihood to Optimize Recent Cryptanalytic Techniques, " presented at the rump session of EUROCRYPT '94, May 1994.

1271. B. Preneel, W. Van Leekwijck, L. Van Linden, R. Govaerts, and J. Vandewalle, "Propagation Characteristics of Boolean Functions, " Advances in Cryptology EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 161-173.

1272. W.H. Press, B.R Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, 1988.

1273. W. Price, "Key Management for Data Encipherment, " Security: Proceedings of IFIP/SEC '83, North Holland: Elsevier SciencePublishers 1983.

1274. G.R Purdy, "A High-Security Log-in Proce dure," communications of the ACM, v 17, n. 8, Aug 1974, pp. 442-445.

1275. J.-J. Quisquater, "Announcing the Smart-Card with RSA Capability, " Proceedings of the Conference: IC Cards and Applications, Today and Tomorrow, Amsterdam, 1989.

1276. J.-J. Quisquater and C. Couvreur, "Fast Decipherment Algorithm for RSA Public Key Cryptosystem," Electronic Letters, v. 18, 1982, pp. 155-168.

1277. J.-J. Quisquater and J.-R Delescaille, "Other Cycling Tests for DES," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 255-256.

1278. J.-J. Quisquater and Y.G. Desmedt, "Chinese Lotto as an Exhaustive Code-Breaking Machine," Computer. v. 24, n. 11, Nov 1991, pp. 14-22.

1279. J.-J. Quisquater and M. Girault, "2p-bit Hash Functions Using e-bit Symmetric Block Cipher Algorithms, Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 102-109.

1280. J.-J. Quisquater and L.C. Guillou, "Des Procedes d'Authentification Bases sur une Publication de Problemes Complexes et Personnalises dont les Solutions Maintenues

Secretes Constituent autant d'Accreditations, " Proceedings of SECURICOM '89: 7th Worldwide Congress on Computer and Communications Security and Protection, Societe d'Edition et d'Organisation d'Expositions Professionnelles, 1989, pp. 149-158. (In French.)

1281. J.-J., Myriam, Muriel, and Michael Quisquater; L., Marie Annick, Gaid, Anna, Gwenole, and Soazig Guillou; and T. Berson, "How to Explain Zero-Knowledge Protocols to Your Children," Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag 1990, pp. 628-631.

1282. M.O. Rabin, "Digital Signatures," Foundations of Secure Communication, New York: Academic Press, 1978, pp. 155-168.

1283. M.O. Rabin, "Digital Signatures and Public-Key Functions as Intractable as Factorization, " MIT Laboratory for Computer Science, Technical Report, MlT/LCS/TR 212, Jan 1979.

1284. M.O. Rabin, "Probabilistic Algorithm for Testing Primality," Journal of Number Theory, v. 12, n. 1, Feb 1980, pp. 128-138.

1285. M.O. Rabin, "Probabilistic Algorithms in Finite Fields," SIAM Journal on Computing, v.9, n.2, May 1980, pp.273-280.

1286. M.O. Rabin, "How to Exchange Secrets by Oblivious Transfer," Technical Memo TR 81, Aiken Computer Laboratory, Harvard University, 1981.

1287. M.O. Rabin, "Fingerprinting by Random Polynomials, " Technical Report TR15-81, Center for Research in Computing Technology, Harvard University, 1981.

1288. T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority," Proceedings of the 21st ACM Symposium on the Theory of Computing, 1989, pp. 73-85.

1289. RAND Corporation, A Million Random Digits with 100,000 Normal Deviates, Glencoe, IL: Free Press Publishers, 1955.

1290. T.R.N. Rao, "Cryposystems Using Algebraic Codes," International Conference on Computer Systems and Signal Processing, Bangalore, India, Dec 1984.

1291. T.R.N. Rao, "On Struit-Tilburg Cryptanalysis of Rao-Nam Scheme," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 458-460.

1292. T.R.N. Rao and K.H. Nam, "Private-Key Algebraic-Coded Cryptosystems, " Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp.35-48.

1293. T.R.N. Rao and K.H. Nam, "Private-Key Algebraic-Code Encryptions," IEEE Transactions on Information Theory, v. 35, n. 4, Jul 1989, pp. 829-833.

1294. J.A. Reeds, "Cracking Random Number Generator," Cryptologia, v. 1, n. 1, Jan 1977, pp. 20-26.

1295. J.A. Reeds, "Cracking a Multiplicative Congruential Encryption Algorithm, " in Information Linkage Between Applied Mathematics and Industry, P.C.C. Wang, ed., Academic Press, 1979, pp. 467 472.

1296. J.A. Reeds, "Solution of Challenge Cipher," Cryptologia, v. 3, n. 2, Apr 1979, pp. 83-95.

1297. J.A. Reeds and J.L. Manferdelli, "DES Has No Per Round Linear Factors," Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 377-389.

1298. J.A. Reeds and N.J.A. Sloane, "Shift Register Synthesis (Modulo m)," SIAM Journal on Computing, v. 14, n. 3, Aug 1985, pp. 505-513.

1299. J.A. Reeds and P.J. Weinberger, "File Security and the UNIX Crypt Command, " AT &T Technical Journal, v. 63, n. 8, Oct 1984, pp. 1673-1683.

1300. T. Renji, "On Finite Automaton One-Key Cryptosystems," Fast Software Encryption, Cambridge Security Workshop Proceed ings, Springer-Verlag, 1994, pp. 135-148.

1301. T. Renji and C. Shihua, "A Finite Automaton Public Key Cryptosystems and Digital Signature, " Chinese Journal of Computers, v. 8, 1985, pp. 401 -409. (In Chinese.)

1302. T. Renji and C. Shihua, "Two Varieties of Finite Automaton Public Key Cryptosystems and Digital Signature, " Journal of Computer Science and Tecnology, v. 1, 1986, pp. 9-18. (In Chinese.)

1303. T. Renji and C. Shihua, "An Implementation of Identity-based Cryptosystems and Signature Schemes by Finite Automaton Public Key Cryptosystems," Advances in Cryptology CHINACRYPT '92, Bejing: Science Press, 1992, pp.87-104. (In Chinese.)

1304. T. Renji and C. Shihua, "Note on Finite Automaton Public Key Cryptosystems, " CHINACRYPT '94, Xidian, China, 11-15 Nov 1994, pp. 76-80.

1305. Research and Development in Advanced Communication Technologies in Europe, RIPE Integrity Primitives: Final Report of RACE Integrity Primitives Evaluation (R1040), RACE, June 1992.

1306. J.M. Reyneri and E.D. Karnin, "Coin Flipping by Telephone," IEEE Transactions on Information Theory, v. IT-30, n. 5, Sep 1984, pp. 775-776.

1307. P. Ribenboim, The Book of Prime Number Records, Springer-Verlag, 1988.

1308. P. Ribenboim, The Little Book of Big Primes, Springer-Verlag, 1991.

1309. M. Richter, "Fin Rauschgenerator zur Gewinnung won quasi-idealen Zufallszahlen fur die stochastische Simulation," Ph.D. dissertation, Aachen University of Technology, 1992. (In German.)

1310. R.F. Rieden, J.B. Snyder, R.J. Widman, and W.J. Barnard, "A Two-Chip Implementation of the RSA Public Encryption Algorithm," Proceedings of GOMAC (Government Microcircuit Applications Conference), Nov 1982, pp. 24 27.

1311. H. Riesel, Prime Numbers and Computer Methods for Factorization, Boston: Birkhauscr, 1985.

1312. K. Rihaczek, "Data Interchange and Legal Security Signature Surrogates," Computers & Security, v. 13, n. 4, Sep 1994, pp. 287-293.

1313. V. Rilmen and B. Preneel, "Improved Characteristics for Differential Cryptanalysis of Hash Functions Based on Block Ciphers," K.U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

1314. R.L. Rivest, "A Description of a Single-Chip Implementation of the RSA Cipher, " LAMBDA Magazine, v. 1, n. 3, Fall 1980, pp. 14-18.

1315. R.L. Rivest, "Statistical Analysis of the Hagelin Cryptograph," Cryptologia, v. 5, n. 1, Jan 1981, pp. 27-32.

1316. R.L. Rivest, "A Short Report on the RSA Chip, " Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, p. 327.

1317. R.L. Rivest, "RSA Chips (Past/Present/ Future), " Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer- Verlag, 1985, pp. 159-168.

1318. R.L. Rivest, "The MD4 Message Digest Algorithm," RFC 1186, Oct 1990.

1319. R.L. Rivest, "The MD4 Message Digest Algorithm," Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 303-311.

1320. R.L. Rivest, "The RC4 Encryption Algorithm, " RSA Data Security, Inc., Mar 1992.

1321. R.L. Rivest, "The MD4 Message Digest Algorithm," RFC 1320, Apr 1992.

1322. R.L. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, Apt 1992.

1323. R.L. Rivest, "Dr. Ron Rivest on the Difficulty of Factoring," Ciphertext: The RSA Newsletter, v. 1, n. 1, Fall 1993, pp. 6, 8.

1324. R.L. Rivest, "The RC5 Encryption Algorithm," Dr. Dobb's Journal, v. 20, n. 1, Jan 95, pp. 146-148.

1325. R.L. Rivest, "The RC5 Encryption Algorithm, " K. U. Leuven Workshop on CryptographicAlgorithms, Springer-Verlag, 1995, to appear.

1326. R.L. Rivest, M.E. Hcllman, J.C. Anderson, and J.W. Lyons, "Responses to NIST's Proposal," Communications of the ACM, v. 35, n. 7, Jul 1992, pp. 41-54.

1327. R.L. Rivest and A. Shamir, "How to Expose an Eavesdropper," Communications of the ACM, v.27, n.4, Apr 1984, pp.393-395.

1328. R.L. Rivest, A. Shamir, and L.M. Adleman, "A Method for Obtaining Digital Signatures and Publie-Key Cryptosystems," Communications of the ACM, v. 21, n. 2, Fe b 1978, pp. 120-126.

1329. R.L. Rivest, A. Shamir, and L.M. Adlcman, "On Digital Signatures and Public Key Cryptosystems," MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212, Jan 1979.

1330. R.L. Rivest, A. Shamir, and L.M. Adleman, "Cryptographic Communications System and Method," U.S. Patent #4,405,829, 20 Sep 1983.

1331. M.J.B. Robshaw, "Implementations of the Search for Pseudo-Collisions in MD5, " Technical Report TR-103, Version 2.0, RSA Laboratories, Nov 1993.

1332. M.J.B. Robshaw, "The Final Report of RACE 1040: A Technical Summary," Technical Report TR-9001, Version 1.0, RSA Laboratories, Jul 1993.

1333. M.J.B. Robshaw, "On Evaluating the Linear Complexity of a Sequence of Least Period 2n,", Designs, Codes and Cryptography, v. 4, n. 3, 1994, pp. 263-269.

1334. M.J.B. Robshaw, "Block Ciphers," Technical Report TR-601, RSA Laboratories, Jul 1994.

1335. M.J.B. Robshaw, "MD2, MD4, MD5, SHA, and Other Hash Functions, " Technical Report TR-101, Version 3.0, RSA Laboratories, Jul 1994.

1336. M.J.B. Robshaw, "On Pseudo-Collisions in MD5," Technical Report TR-102, Version 1.1, RSA Laboratories, Jul 1994.

1337. M.J.B. Robshaw, "Security of RC4," Technical Report TR-401, RSA Laboratories, Jul 1994.

1338. M.J.B. Robshaw, personal communication, 1995.

1339. M. Roe, "Reverse Engineering of an EES Device," K. U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995, to appear.

1340. P. Rogaway and D. Coppersmith, "A Software-Oriented Encryption Algorithm, " Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 56-63.

1341. H.L. Rogers, "An Overview of the Cand-ware Program, " Proceedings of the 3rd Annual Symposium on Physical/Electronic Security, Armed Forces Communications and Electronics Association, paper 31, Aug 1987.

1342. J. Rompel, "One-Way Functions Are Necessary and Sufficient for Secure Signatures," Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing, 1990, pp. 387-394.

1343. T. Rosati, "A High Speed Data Encryption Processor for Public Key Cryptography, " Proceedings of the IEEE Custom Integrated Circuits Conference, 1989, pp. 12.3.1-12.3.5.

1344. O.S. Rothaus, "On Bent'Functions," Journal of Combinational Theory Series A, v. 20, n. 3, 1976, pp. 300-305.

1345. RSA Laboratories, "PKCS #1: RSA Encryption Standard," version 1.5, Nov 1993.

1346. RSA Laboratories, "PKCS #3: Diffie-Hellman Key-Agreement Standard, " version 1.4, Nov 1993.

1347. RSA Laboratories, "PKCS #5: Password-Based Encryption Standard," version 1.5, Nov 1993.

1348. RSA Laboratories, "PKCS #6: Extended-Certificate Syntax Standard," version 1.5, Nov 1993.

1349. RSA Laboratories, "PKCS #7: Cryptographic Message Syntax Standard," version 1.5, Nov 1993.

1350. RSA Laboratories, "PKCS #8: Private Key Information Syntax Standard, " version 1.2, Nov 1993.

1351. RSA Laboratories, "PKCS #9: Selected Attribute Types," version 1.1, Nov 1993.

1352. RSA Laboratories, "PKCS #10: Certification Request Syntax Standard, " version 1.0, Nov 1993.

1353. RSA Laboratories, "PKCS #11 : Cryptographic Token Interface Standard, " version 1.0, Apr 95.

1354. RSA Laboratories, "PKCS #12: Public Key User Information Syntax Standard," version 1.0, 1995.

1355. A.D. Rubin and P. Honeyman, "Formal Methods for the Analysis of Authentication Protocols," draft manuscript, 1994.

1356. F. Rubin, "Decrypting a Stream Cipher Based on J-K Flip-Flops, " IEEE Transactions on Computing. v. C-28, n. 7, Jul l 97Y, pp. 483 487.

1357. R.A. Rueppel, Analysis and Design of Stream Ciphers, Springer-Verlag, 1986.

1358. R.A. Rueppel, "Correlation Immunity and the Summation Combiner," Advances in Cryptology EUROCRYPT '85, Springer-Verlag, 1986, pp. 260-272.

1359. R.A. Rueppel, "When Shift Registers Clock Themselves," Advances in Cryptology EUROCRYPT '87 Proceedings, Springer-Verlag, 1987, pp. 53-64.

1360. R.A. Rueppel, "Security Models and Notions for Stream Ciphers," Cryptography and Coding 11, C. Mitchell, ed., Oxford: Clarendon Press, 1992, pp. 213 230.

1361. R.A. Rueppel, "On the Security of Schnorr's Pseudo-Random Sequence Generator," Advances in Cryptology EUROCRYPT 89 Proceedings, Springer-Verlag, 1990, pp. 423-428.

1362. R.A. Rueppel, "Stream Ciphers," Contemporary Cryptology: The Science of Information Integrity, G.J. Simmons, ed., IEEE Press, 1 992, pp. 65-134.

1363. R.A. Rueppel and J.L. Massey, "The Knapsack as a Nonlinear Function," IEEE International Symposium on Information Theory, Brighton, UK, May 1985.

1364. R. A. Rueppel and O. J. Staffelbaeh, " Products of Linear Recurring Sequences with Maximum Complexity, " IEEE Transactions on Information Theory, v. IT-33, n. 1, Jan 1987, pp. 124-131.

1365. D. Russell and G.T. Gangemi, Computer Security Basics, O'Reilly and Associates, Inc., 1991.

1366. S. Russell and P. Craig, "Privacy Enhanced Mail Modules for ELM," Proceedings of the Internet Society 1994 Workshop on Network and Distributed System Security, The Internet Society, 1994, pp. 21-34.

1367. D.F.H. Sadok and J. Kelner, "Privacy Enhanced Mail Design and Implementation Perspectives," Computer Communications Review, v. 24, n. 3, Jul 1994, pp. 38 -46.

1368. K Sakano, "Digital Signatures with User Flexible Reliability," Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93), Shuzenji, Japan, 28-30 Jan 1993, pp. 5C.1-8.

1369. K. Sakano, C. Park, and K. Kunsawa, "Threshold Undeniable Signature Scheme," Proceedings of the 1993 Korea Japan Workshop on Information Security and Cryptography, Seoul, Korea, 24-26 Oct 1993, pp. 184-193.

1370. K. Sako, "Electronic Voting Schemes Allowing Open Objection to the Tally," Transactions of the Institute of Electron ics, Information, and Communication Engineers, v. E77-A, n. 1, 1994, pp. 24-30.

1371. K. Sako and J. Kilian, "Secure Voting Using Partially Compatible Homomorphisms," Advances ill Cryptology CRYPTO '94 Proceedings, Springer-Verlag, 1994, p. 411-424.

1372. K. Sako and J. Kilian, "Receipt-Free Mix-Type Voting Scheme A Practical Solution to the Implementation of a Voting Booth," Advances in Cryptology EUROCRYPT '95 Proceedings, Springer-Verlag, 1'995, pp. 393 -403.

1373. A. Salomaa, Public-Key Cryptography, Springer-Verlag, 1990.

1374. A. Salomaa and L. Santean, "Secret Selling of Secrets with Many Buyers," ETACS Bulletin, v. 42, 1990, pp. 178-186.

1375. M. Santha and U.V Vazirani, "Generating Quasi-Random Sequences from Slightly Random Sources," Proceedings of the 25th Annual Symposium on the Fo undations of Computer Science, 1984, pp. 434-440.

1376. M. Santha and U.V Vazirani, "Generating Quasi-Random Sequences from Slightly Random Sources, " 70lzrnal of Computer and System Sciences, v.33, 1986, pp. 75-87.

1377. S. Saryazdi, "An Extension to EIGamal Public Key Cryptosystem with a New Signature Scheme," Proceedings of the 1990 Bilkent International Conference O n New Trends in Communication, Control, and Signal Processing, North Holland: Elsevier Science Publishers, 1990, pp. 195-198.

1378. J.E. Savage, "Some Simple Self- Synchronizing Digital Data Scramblers." Bell System Technical Journal, v. 46, n. 2, Feb 1967, pp. 448 -487.

1379. B.P Sehanning, "Applying Public Key Distribution to Local Area Networks, " Computers & Security, v. 1, n. 3, Nov 1982, pp. 268-274.

1380. B.P Schanning, S.A. Powers, and J. Kowalchuk, "MEMO: Privacy and Authentication for the Automated Office, " Proceethngs of the 5th Conference on Local Computer Networks, IEEE Press, 1980, pp. 21-30.

1381. L. Schaumuller-Bichl, "Zur Analyse des Data Encryption Standard und Synthese Verwandter Chiffriersysteme," Ph.D. dissertation, Linz University, May 1981. (In German.)

1382. Sehaumuller-Bichl, "On the Design and Analysis of New Cipher Systems Related to the DES," Technical Report, Linz University, 1983.

1383. A. Scherbius, "Ciphering Machine," U.S. Patent #1,657,411, 24 Jan 1928.

1384. J.I. Schiller, "Secure Distributed Computing," Scientific American, v. 271, n.5, Nov 1994, pp. 72-76.

1385. R. Schlafly, "Complaint Against Exclusive Federal Patent License," Civil Action File No. C-93 20450, United States District Court for the Northern District of California.

1386. B. Schneier, "One-Way Hash Functions," Dr. Dobb's journal, v. 16, n. 9, Sep 1991, pp. 148-151.

1387. B. Schneier, "Data Guardians," MacWorld, v. 10, n. 2, Feb 1993, pp. 145-151.

1388. B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," Fast Software Encryption, Cambridge Secunty Workshop Proceedings, Springer-Verlag, 1994, pp. 191-204.

1389. B. Schneier, "The Blowfish Encryption Algorithm," Dr. Dobb's Journal, v. 19, n. 4, Apr 1994, pp. 38-40.

1390. B. Schneier. Protect Your Macintosh, Peachpit Press, 1994.

1391. B. Schneier, "Designing Encryption Algorithms for Real People, " Proceedings of the 1994 ACM SIGSAC New Secunty Paradigms Workshop, IEEE Computer Society Press, 1994, pp. 63-71.

1392. B. Schneier, "A Primer on Authentication and Digital Signatures," Computer Secu rity lournal, v. 10, n. 2, 1994, pp. 38-40.

1393. B. Schneier, "The GOST Encryption Algorithm," Dr. Dobb's journal, v. 20, n. 1, Jan 95, pp. 123-124.

1394. B. Schneier, E-Mail Secu rity (with POP and SEM) New York: John Wiley & Sons, 1995.

1395. C.P Schnorr, "On the Construction of Random Number Generators and Random Function Generators," Advances in Cryptology EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 225-232.

1396. C.P Schnorr, "Efficient Signature General tion for Smart Cards," Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 239-252.

1397. C.P. Schnorr, "Efficient Signature Generation for Smart Cards," Journal of Cryptology,v.4,n.3, 1991,pp. 161-174.

1398. C.P Schnorr, "Method for Identifying Subscribers and for Generating and Verifying Electronic Signatures in a Data Exchange System," U.S. Patent #4,995,082, 19 Feb 1991.

1399. C.P. Schnorr, "An Efficient Cryptographic Hash Function, " presented at the rump session of CRYPTO '91, Aug 1991.

1400. C.P. Schnorr, "FFT-Hash II, Efficient Cryptographic Hashing, " Advances in Cryptology EUROCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 45-54.

1401. C.P. Schnorr and W. Alexi, "RSA-bits are 0.5 + E Secure," Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, pp. 113-126.

1402. C.R Schnorr and S. Vaudenay, "Parallel FFT-Hashing," Fast Software Encryption, Cambridge Secunty Workshop Proceedings, Springer-Verlag, 1994, pp. 149-156.

1403. C.P. Schnorr and S. Vaudenay, "Black Box Cryptanalysis of Hash Networks Based on Multipermutations, " Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

1404. W. Schwartau, Information Warfare: Chaos on the Electronic Superhighway, New York: Thunders Mouth Press, 1994.

1405. R. Scott, "Wide Open Encryption Design Offers Flexible Implementations," Cryptologia, v. 9, n. 1, Jan 1985, pp. 75-90.

1406. J. Seberry, "A Subliminal Channel in Codes for Authentication without Secrecy, " Ars Combinatorica, v. 19A, 1985, pp. 337-342.

1407. J. Seberry and J. Pieprzyk, Cryptography: An Introduction to Computer Security, Englewood Cliffs, N.l.: Prentice-Hall, 1989.

1408. J. Seberry, X.-M. Zhang, and Y. Zheng, "Nonlinearly Balanced Boolean Functions and Their Propagation Characteristics, " Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1994, pp. 49-60.

1409. H. Sedlack, "The RSA Cryptography Processor: The First High Speed One-Chip Solution, " Advances in Cryptology EUROCRYPT '87 Proceedings, Springer- Verlag, 1988, pp. 95-105.

1410. H. Sedlack and U. Golze, "An RSA Cryptography Processor," Microprocessing and Microprogramming, v. 18, 1986, pp. 583-590.

1411. E.S. Selmer, Linear Recurrence over Finite Field, University of Bergen, Norway, 1966.

1412. J.O. Shallit, "On the Worst Case of Three Algorithms for Computing the Jacobi Symbol," Journal of Symbolic Computation, v. 10, n. 6, Dec 1990, pp. 593-610.

1413. A. Shamir, "A Fast Signature Scheme," MIT Laboratory for Computer Science, Technical Memorandum, MIT/LCS/TM 107, Massachusetts Institute of Technology, Jul 1978.

1414. A. Shamir, "How to Share a Secret," Communications of the ACM, v. 24, n. 11, Nov 1979, pp. 612-613.

1415. A. Shamir, "On the Cryptocomplexity of Knapsack Systems, " Proceedings of the 11th ACM Symposium on the Theory of Computing, 1979, pp. 118-129.

1416. A. Shamir, "The Cryptographic Security of Compact Knapsacks, " MIT Library for Computer Science, Technical Memorandum, MIT/LCS/TM164, Massachusetts Institute of Technology, 1980.

1417. A. Shamir, "On the Generation of Cryptographically Strong Pseudo-Random Sequences, " Lecture Notes in Computer Science 8th International Colloquium On Automata, Languages, and Programming, Springer-Verlag, 1981.

1418. A. Shamir, "A Polynomial Time Algorithm for Breaking the Basic Merkle -Hellman Cryptosystem," Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, 1983, pp. 279-288.

1419. A. Shamir, "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem, " Proceedings of the 23rd IEEE Sym posium on the Foundations of Computer Science, 1982,pp. 145-152.

1420. A. Shamir, "On the Generation of Cryptographically Strong Pseudo-Ranclom Sequences," ACM Transactions on Computer Systems, v. 1, n. l, Fe b 1983, pp. 38-44.

1421. A. Shamir, "A Polynomial Time Algorithm for Breaking the Basic Merkle Hellman Cryptosystem, " IEEE Transactions on Information Theory, v. IT-30, n. 5, Sep 1984, pp. 699-704.

1422. A. Shamir, "Identity-Based Cryptosystems and Signature Schemes, " Advances in Cryptology: Proceedings of CRYPTO '84. Springer-Verlag, 1985, pp. 47-53.

1423. A. Shamir, "On the Security of OES," Advances in Cryptology C RYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 280-281.

1424. A. Shamir, lecture at SECURICOM '89.

1425. A. Shamir, "Efficient Signature Schemes Based on Birational Permutations," Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 1-12.

1426. A. Shamir, personal communication, 1993.

1427. A. Shamir and A. Fiat, "Method, Apparatus and Article for Identification and Signature," U.S. Patent #4,748,668, 31 May 1988.

1428. A. Shamir and R. Zippel, "On the Security of the Merkle-Hellman Cryptographic Scheme," IEEE Transactions on Information Theory, v. 26, n. 3, May 1980, pp. 339-340.

1429. M. Shand, R Bertin, and J. Vuillemin, "Hardware Speedups in Long Integer Multiplication," Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, 1990, pp. 138-145.

1430. D. Shanks, Solved and Unsolved Problems in Number Theory, Washington D.C.: Spartan, 1962.

1431. C.E. Shannon, "A Mathematical Theory of Communication," Bell System Technical Journal. v. 27, n. 4, 1948, pp. 379-423, 623-656.

1432. C.E. Shannon, "Communication Theory of Secrecy Systems," Bell System Technical Journal. v. 28, n. 4, 1949, pp. 656-715

1433. C.E. Shannon, Collected Papers: Claude Elmwood Shannon, N.J.A. Sloane and A.D. Wyner, eds., New York: IEEE Press, 1993.

1434. C.E. Shannon, "Predication and Entropy in Printed English," Bell System Technical journal, v. 30, n. 1, 1951, pp. 50-64.

1435. A. Shimizu and S. Miyaguchi, "Fast Data Encipherment Algorithm FEAL," Transactions of IEICE of Japan, v. J70-1, n. 7, Jul 87, pp. 1413-1423. (In Japanese.)

1436. A. Shimizu and S. Miyaguchi, "Fast Data Encipherment Algorithm FEAL, " Advances in Cryptology EUROCRYPT 87 Proceedings, Springer-Vcrlag, 1988, pp. 267-278.

1437. A. Shimizu and S. Miyaguchi, "FEAL Fast Data Encipherment Algorithm," Systems and Computers in Japan, v. 19, n. 7, 1988, pp. 20-34, 104-106.

1438. A. Shimizu and S. Miyaguchi, "Data Randomization Equipment," U.S. Patent #4,850,019, 18 Jul 1989.

1439. M. Shimada, "Another Practical Public key Cryptosystem, " Electronics Letters, v. 28, n. 23, 5 Nov 1992, pp. 2146-2147.

1440. K. Shirriff, personal communication, 1993.

1441. H. Shizuya, T. Itoh, and K. Sakurai, "On the Complexity of Hyperelliptic Discrete Logarithm Problem," Advances in Cryptology EUROCRYPT '91 Proceedings, Springer-Verlag, 1991, pp. 337-351.

1442. Z. Shmuley, "Composite Diffie-Hellman Publie-Key Generating Systems Are Hard to Break," Computer Science Department, Technion, Haifa, Israel, Technical Report 356, Feb 1985.

1443. PW. Shor, "Algorithms for Quantum Computation: Discrete Log and Factoring," Proceedings of the 35th Symposium on Foundations of Computer Science, 1994, pp. 124-134.

1444. L. Shroyer, letter to NIST regarding DSS, 17 Feb 1992.

1445. C. Shu. T. Matsumoto, and H. Imai, "A Multi-Purpose Proof System, Transactions of the Institute of Electronics, Information, and Communication Engineers, v. E75-A, n. 6, Jun 1992, pp. 735-743.

1446. E.H. Sibley, "Random Number Generators: Good Ones Are Hard to Find," Communications of the ACM, v. 31, n. l0, Oct 1988, pp. 1192-1201.

1447. VM. Sidenikov and S.O. Shestakov, "On Encryption Based on Generalized Reed-Solomon Codes," Diskretnaya Math, v. 4, 1992, pp. 57-63. (In Russian.)

1448. V.M. Sidenikov and S.O. Shestakov, "On Insecurity of Cryptosystems Based on Generalized Reed-Solomon Codes, " unpublished manuscript, 1992.

1449. D.P Sidbu, "Authentication Protocols for Computer Networks, " Computer Networks and ISDN Systems, v. 11, n. 4, Apr 1986, pp. 297-310.

1450. T. Siegenthaler, "Gorrelation-Immunity of Nonlinear Combining Functions for Cryptographic Applications, " IEEE Transactions on Information Theory, v. IT-30, n. 5, Sep 1984, pp. 776-780.

1451. T. Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphcrtext Only," IEEE Transactions on Computing, v. C-34, Jan 1985, pp. 81-85.

1452. T. Siegenthaler, "Cryptanalyst's Rcpresentation of Nonlinearity Filtered ml-sequenccs," Advances in Cryptology EUROCRYYT '85, Springer-Verlag, 1986, pp. 103-110.

1453. R.D. Silverman, "The Multiple Polynomial Quadratie Sieve," Mathematics of Compu tation, v. 48, n. 177, Jan 1987, pp. 329-339.

1454. G.J. Simmons, "Authentication without Secrecy: A Secure Communication Pro blem Uniquely Solvable by Asymmetric Encryption Techniques, " Proceedings of IEEE EASCON '79, 1979, pp. 661-662.

1455. G.J. Simmons, "Some Number Theoretic Questions Arising in Asymmetric Encryption Techniques," Annual Meeting of the Ame rican Mathematical .Society, AMS Abstract 763.94.1, 1979, pp. 136-151.

1456. G.J. Simmons, "High Speed Arithmetic Using Redundant Number Systems," Pro ceedings of the National Telecommunications Conference, 1980, pp. 49.3.1 -49.3.2.

1457. G.J. Simmons, "A 'Weak' Privacy Protocol Using the RSA Cryptosystem," Cryptologia, v.7, n,2, Apr 1983, pp.180-182.

1458. G.J. Simmons, "The Prisoner's Problem and the Subliminal Channel," Advances in Cryptology: Proceedings of CRYPTO '83, Plenum Press, 1984, pp. 51-67.

1459. G.J. Simmons, "The Subliminal Channel and Digital Signatures," Advances in Cryptology: Proceedings of EUROCRYPT 84, Springer-Verlag, 1985, pp. 364-378.

1460. G.J. Simmons, "A Secure Subliminal Channel?," Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 33 -41.

1461. G.J. Simmons, "Cryptology," Encyclopedia Britannica, 16th edition, 1986, pp. 913-924B.

1462. G.J. Simmons, "How to 'Really' Share a Secret, " Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 390 448.

1463. G.J. Simmons, "Prepositioned Se cret Sharing Schemes and/or Shared Control Schemes, " Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 436 - 467.

1464. G.J. Simmons, "Geometric Shares Secret and/or Shared Control Schemes, " Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 216-241.

1465. G.J. Simmons, ed., Contemporary Cryptology: The Science o f information Integrity, IEEE Press, 1992.

1466. G.J. Simmons, "An Introduction to Shared Secret and/or Shared Control Schemes and Their Application, " in Contemporary Cryptology: The Science of Information Integrity G.J. Simmons, ed., IEEE Press, 1992, pp. 441 -497.

1467. G.J. Simmons, "How to Insure that Data Acquired to Verify Treaty Compliance Are Trustworthy," in Contemporary Cryptology: The Science of Information Integrity, G.J. Simmons, ed., IEEE Press, 1992, pp. 615-630.

1468. G.J. Simmons, "The Subliminal Channels of the U.S. Digital Signature Algorithm (DSA)," Proceedings of the Third Symposium on: State and Progress of Research in Cryptography, Rome: Fondazone Ugo Bordoni, 1993, pp. 35-54.

1469. G.J. Simmons, "Subliminal Communica tion is Easy Using the USA, " Advances in Cryptology EUROCRYPT '93 Proceedings, Springer-Verlag, 1994, pp. 218-232.

1470. G.J. Simmons, "An Introduction to the Mathematics of Trust in Security Protocols," Proceedings: Computer Security Foundations Workshop VI, IEEE Computer Society Press, 1993, pp. 121-127.

1471. G.J. Simmons, "Protocols that Ensure Fairness," Codes and Ciphers, Institute of Mathematics and its Applications, 1995, pp. 383-394.

1472. G.J. Simmons, "Cryptanalysts and Protocol Failures," Communications of the ACM, v.37, n.11, Nov 1994, pp.56-65.

1473. G.J. Simmons, "Subliminal Channels: Past and Present, " European Transactions on Telecommuncations, v. 4, n. 4, Jul/Aug 1994, pp. 459-473.

1474. G.J. Simmons and M.J. Norris, How to Cipher Fast Using Redundant Number Systems, SAND-80-1886, Sandia National Laboratories, Aug 1980.

1475. A. Sinkov, Elementary Cryptanalysis, Mathematical Association of America, 1966.

1476. R. Siromoney and L. Matthew, "A Public Key Cryptosystem Based on Lyndon Words," Information Processing Letters, v. 35, n. 1, 15 Jun 1990, pp. 33-36.

1477. B. Smeets, "A Note on Sequences Generated by Clock-Controlled Shift Registers," Advances in Cryptology EUKOCRYPT '85, Springer-Vcrlag, 1986, pp. 40 42.

1478. M.E. Smid, "A Key Notarization System for Computer Networks, " NBS Special Report 500-54, U.S. Department of Commerce, Oct 1979.

1479. M.E. Smid, "The DSS and the SHS," Federal Digital Signature Applications Symposium, Rockville, MD, 17-18 Feb 1993.

1480. M.E. Smid and D.K. Branstad, "The Data Encryption Standard: Past and Future, " Proceedings of the IEEE, v. 76, n. 5., May 1988, pp. 550-559.

1481. M.E. Smid and D.K. Branstad, "The Data Encryption Standard: Past and Future," in Contemporary Cryptology: The Science of Information Integrity, G. L. Simmons, ed., IEEE Press, 1992, pp. 43-64.

1482. J.L. Smith, "The Design of Lucifer, A Cryptographic Device for Data Communications, " IBM Research Report RC3326, 1971.

1483. J.L. Smith, "Recirculating Block Cipher Cryptographic System," U.S. Patent #3,796,830, 12 Mar 1974.

1484. J.L. Smith, W.A. Notz, and P.R. Osseck, "An Experimental Application of Cryptography to a Remotely Accessed Data System," Proceedings of the ACM Annual Conference, Aug 1972, pp. 282-290.

1485. K. Smith, "Watch Out Hackers, Public Encryption Chips Are Coming," Electronics Week, 20 May 1985, pp. 30-31.

1486. R Smith, "LUC Public-Key Encryption," Dr. Dobb's journal, v. 18, n. l, Jan 1993, pp. 44-49.

1487. P. Smith and M. Lennon, "LUC: A New Public Key System," Proceedings of the Ninth International Conference on Infor- mation Security, IFlP/Sec 1993, North Holland: Elsevier Science Publishers, 1993, pp. 91-111.

1488. E. Snekkenes, "Exploring the BAN Approach to Protocol Analysis," Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, 1991, pp. 171-181.

1489. B. Snow, "Multiple Independent Binary Bit Stream Generator," U.S. Patent #5,237,615, 17 Aug 1993.

1490. R. Solovay and V. Strassen, "A Fast Monte-Carlo Test for Primality, " SIAM journal on Computing, v. 6, Mar 1977, pp. 84-85; erratum in ibid, v. 7, 1978, p. 118.

1491. T. Sorimachi, T. Tokita, and M. Matsui, "On a Cipher Evaluation Method Based on Differential Cryptanalysis," Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94), Lake Biwa, Japan, 27-29 Jan 1994, pp. 4C.l-9. (In Japanese.)

1492. A. Sorkin, "Lucifer, a Cryptographic Algorithm," Cryptologia, v. 8, n. 1, Jan 1984, pp. 22-41.

1493. W. Stallings, "Kerberos Keeps the Ethernet Secure, " Data Communications, Oct 1994, pp. 103-111.

1494. W. Stallings, Network and Internetwork Secunty, Englewood Cliffs, N.J.: Prentice Hall, 1995.

1495. W. Stallings, Protect Your Privacy: A Guide for POP Users, Englewood Cliffs, N. J.: Prentice-Hall, 1995.

1496. Standards Association of Australia, "Australian Standard 2805.4 1985: Electronic Funds Transfer Requirements for Interfaces: Part 4 Message Authentication, " SAA, North Sydney, NSW, 1985.

1497. Standards Association of Australia, "Australian Standard 2805.5 1985: Electronic Funds Transfer Requirements for Interfaces: Part 5 Data Encipherment Algorithm," SAA, North Sydney, NSW, 1985.

1498. Standards Association of Australia, "Australian Standard 2805.5.3: Electronic Data Transfer Requirements for Interfaces: Part 5.3 Data Encipherment Algorithm 2," SAA, North Sydney, NSW, 1992.

1499. J.G. Steiner, B.C. Neuman, and J. J. Schiller, "Kerberos: An Authentication Service for Open Network Systems," USENIX Conference Proceedings, Feb 1988, pp. 191-202.

1500. J. Stern, "Secret Linear Congruential Generators Are Not Cryptographically Secure," Proceedings of the 28th Symposium on Foundations of Computer Science, 1987, pp. 421-426.

1501. J. Stern, "A New Identification Scheme Based on Syndrome Decoding," Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 13-21.

1502. A. Stevens, "Hacks, Spooks, and Data Encryption," Dr. Dobb's journal, v. 15, n. 9, Sep 1990, pp. 127-134, 147-149.

1503. R. Struik, "On the Rao-Nam Private-Key Cryptosystem Using Non-Linear Codes," IEEE 1991 Symposium on Information Theory, Budapest, Hungary, 1991.

1504. R. Struik and J. van Tilburg, "The Rao- Nam Scheme Is insecure against a Chosen-Plaintext Attack, " Advances in Ctyptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 445-457.

1505. S.G. Stubblebine and V.G. Gligor, "Protecting the Integrity of Privacy-Enhanced Mail with DES-Based Authentication Codes," Proceedings of the Privacy and Secunty Research Group 1993 Workshop on Network and Distributed System Security, The Internet Society, 1993, pp. 75-80.

1506. R. Sugarman, "On Foiling Computer Crime," IEEE Spectrum, v. 16, n. 7, Jul 79, pp.31 - 32.

1507. H.N. Sun and T. Hwang, "Public-key ID- Based Cryptosystem," Proceedings of the 25th Annual 1991 IEEE International Carnahan Conference on Security Technology, Taipei, Taiwan, 1 -3 Oct 1991, pp. 142-144.

1508. RF. Syverson, "Formal Semantics for Logics of Computer Protocols, " Proceedings of the Computer Secunty Foundations Workshop III, IEEE Computer Society Press, 1990, pp. 32 41.

1509. RF. Syverson, "The Use of Logic in the Analysis of Cryptographic Protocols," Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, 1991,pp. 156-170.

1510. RF. Syverson, "Knowledge, Belief, and Semantics in the Analysis of Cryptographic Protocols," journal of Computer Secu rity, v. 1, n. 3, 1992, pp. 317-334.

1511. RF. Syverson, "Adding Time to a Logic Authentication," 1st ACM Conference on Computer and Communications Security, ACM Press, 1993, pp. 97-106.

1512. RF. Syverson and C.A. Meadows, "A Logical Language for Specifying Cryptographic Protocol Requirements, " Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy, 1993, pp. 14-28.

1513. RE Syverson and C.A. Meadows, "Formal Requirements for Key Distribution Protocols," Advances in Cryptology EUROCRYPT '94 Proceedings, Springer-Verlag, 1995, to appear.

1514. RF. Syverson and RC. van Oorschot, "On Unifying Some Cryptographic Protocol Logics," Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, 1994, pp. 165-177.

1515. H. Tanaka, "A Realization Scheme for the Identity-Based Cryptosystem," Advances in Cryptology CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 340-349.

1516. H. Tanaka, "A Realization Scheme for the Identity- based Cryptosystem," Electronics and communications in Japan, Part 3 (Fundamental Electronic Science), v. 73, n. 5, May 1990, pp. 1-7.

1517. H. Tanaka, "Identity-Based Noninteractive Common-Key Generation and Its Application to Cryptosystems," Transactions of the Institute of Electronics, Information, and Communication Engineers, v. J75-A, n. 4, Apr 1992, pp. 796-800.

1518. J. Tardo and K. Alagappan, "SPX: Global Authentication Using Public Key Certificates," Proceedings of the 1991 IEEE Computer Society symposiLlm on Security and Privacy, 1991, pp. 232-244.

1519. J. Tardo, K. Alagappan, and R. Pitkin, "Public Key Based Authentication Using Internet Certificates, " USENIX Security 11 Workshop Proceedings, 1990, pp. 121-123.

1520. A. Tardy-Corfdir and H. Gilbert, "A Known Plaintext Attack of FEAL-4 and FEAL-6, " Advances in Cryptology CRYPTO'91 Proceedings, Springer-Verlag, 1992, pp. 172- 182.

IS21. M. Tatebayashi, N. Matsuzaki, and D.B. Newman, "Key Distribution Protocol for Digital Mobile Communication System," Advances in Cryptology CRYPTO '89 Proceedings, Springcr-Verlag, 1990, pp. .324- 333.

1522. M. Taylor, "Implementing Privacy Enhanced Mail on VMS," Proceedings of the Privacy and Security Research Group 1993 Workshop o n Network and Distributed System Security, The Internet Society, 1993, pp. 63-68.

1523. R. Taylor, "An Integrity Cheek Value Algorithm for Stream Ciphers," Advances in Cryptology CRYPTO '93 Proceedings, Springer-Verlag, 1994, pp. 40-48.

1524. T. Tedrick "Fair Exchange of Secrets, " Advances in Cryptology: Proceedings of CRYPTO '84, Springer-Verlag, 1985, pp. 434- 448.

1525. R. Terada and P.G. Pinheiro, "How to Strengthen FEAL against Differential Cryptanalysis, " Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography, Innyama, Japan, 24-27 Jan 1995, pp. 153-162.

1526. J.-P. Tillich and G. Nemor, "Hashing with Sly," Advances in Cryptology CRYPTO '94 Proceedings, Springer-Verlag, 1994, pp. 40 49.

1527. T. Tokita, T. Sorimachi, and M. Matsui, "An Efficient Search Algorithm for the Best Expression on Linear Cryptanalysis." IEICE Japan, Technical Report, ISEC93-97, 1994.

1528. M. Tompa and F. Woll, "Random Self Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information," Proceedings of the 28th IEEE Sy mposium on the Foundations of Computer Science, 1987, pp. 472-482.

1529. M. Tompa and H. Woll, "How to Share a Secret with Cheaters," journal of Cryptology, v. 1, n. 2, 1988, pp. 133-138.

1530. M.-J. Toussaint, "Verification of Cryptographic Protocols, " Ph.D. dissertation, Universite de Liege, 1991.

1531. M.-J. Toussaint, "Deriving the Complete Knowledge of Participants in Cryptographic Protocols," Advances in Cryptology CRYPTO '91 Proceedings, SpringerVerlag, 1992, pp. 24-43.

1532. M.-J. Toussaint, "Separating the Specification and Implementation Phases in Cryptology," ESORICS 92, Proceedings of the Second European Symposium on Research in Computer Security, Springcr-Verlag, 1992, pp. 77-101.

1533. P.D. Townsend, J.G. Rarity, and RR. Tapstcr, "Enhanced Single Photon Fringe Visibility in a 10 km-Long Prototype Quantum Cryptography Channel," Electronics Letters, v. 28, n. 14, S Jul 1993, pp. 1291

1534. S.A. Tretter, "Properties of PN2 Sequences," IEEE Transactions on Information Theory, v. IT-20, n. 2, Mar 1974, pp. 295-297.

1535 H. Truman, "Memorandum for: The Secretary of State, The Secretary of Defensc," A 20707 5/4/54/OSO, NSA TS CONTL. NO 73- 00405, 24 Oct 1952.

1536. Y.W. Tsai and T. Hwang, "ID Based Public Key Cryptosystem Based on Okamoto and Tanaka's ID Based Onc-Way Communications Scheme," Electronics Letters, v. 26, n. 10, 1 May 1990, pp. 666- 668.

1537. G. Tsudik, "Message Authentication with One-Way Hash Functions," ACM Comp uter Communications Review, v. 22, n. 5, 1992, pp. 29 - 38.

1560. J. van Tilburg, "Cryptanalysts of the Xinmei Digital Signature Scheme," Electronics Letters, v. 28, n. 20, 24 Sep 1992, pp. 1935-1938.

1561. J. van Tilburg, "Two Chosen-Plaintext Attacks on the Li Wang Joing Authentication and Encryption Scheme, " Applied Algebra, Algebraic Algorithms and Error Correcting Codes 10, Springer-Verlag, 1993, pp. 332-343.

1562. J. van Tilburg, "Security-Analysis of a Class of Cryptosystems Based on Linear Error-Correcting Codes, " Ph.D. dissertation, Technical University Eindhoven, 1994.

1563. A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and RG. Jespers, "A Single Chip 1024 Bits RSA Processor," Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 219-236.

1564. J. Vanderwalle, D. Chaum, W. Fumy, C. Jansen, P. Landroek, and G. Roelofsen, "A European Call for Cryptographic Algorithms: RIPE; RACE Integrity Primitives Evaluation, " Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, pp. 267-271.

1565. V. Varadharajan, "Verification of Network Security Protocols," Computers and Security, v. 8, n. 8, Aug 1989, pp. 693-708.

1566. V. Varadharajan, "Use of a Formal Description Technique in the Specification of Authentication Protocols, " Computer Standards and Interfaces, v. 9, 1990, pp. 203-215.

1567. S. Vaudenay, "FFT-Hash-II Is not Yet Collision-Free," Advances in Cryptology CRYPTO '92 Proceedings, Springer-Verlag, pp. 587-593.

1568. S. Vaudenay, "Differential Cryptanalysis of Blowfish, " unpublished manuscript, 1995.

1569. U.V. Vazirani and V.V. Vazirani, "Trapdoor Pseudo-Random Number Generators with Applications to Protocol Design, " Proceedings of the 24th IEEE Symposium on the Foundations of Computer Science, 1983, pp. 23-30.

1570. U.V. Vazirani and V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation," Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science, 1984, pp. 458 463.

1571. U.V. Vazirani and V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation," Advances in Cryptology: Proceedings of CR YP TO '84, Springer -Verlag, 1985, pp. 193-202.

1572. I. Verbauwhede, F. Hoornaert, J. Vanderwalle, and H. De Man, "ASIC Cryptographical Processor Based on DES," Euro ASIC '91 Proceedings, 1991, pp. 2 92-295.

1573. I. Verbanwhede, F. Hoornaert, J. Vanderwalle, H. De Man, and R. Govaerts, "Security Considerations in the Design and Implementation of a New DES Chip, " Advances in Cryptology EUROCRYPT '87 Proceedings, Springcr-Verlag, 1988, pp. 287-300.

1574. R. Vogel, "On the Linear Complexity of Caseaded Sequences," Advances in Cryptology: Proceedings oi EUROCRYPT 84, Springer-Verlag, 1985, pp. 99- 109.

1575. S. von Solms and D. Naccache, "On Blind Signatures and Perfect Crimes," Computers & Security, v. 11, 1992, pp. 581-583.

1576. V.L. Voydock and S.T. Kent, "Security Meehanisms in High-Level Networks, " ACM Computing Surveys, v. 15, n. 2, Jun 1983, pp. 135-171.

1577. N.R. Wagner, RS. Putter, and M.R. Cain, "Large-Scale Randomization Techniques," Advances in Cryptology CRYPTO '86 Proceedings, Springer-Verlag, 1987, pp. 393 -404.

1578. M. Waidner and B. Pfitzmann, "The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability," Advances in Cryptology EUROCRYPT '89 Proceedings, Springer-Verlag, 1990, p. 690.

1579. S.T. Walker, "Software Key Escrow A Better Solution for Law Enforcement's Needs ?" TIS Report #533, Trusted Information Systems, Aug 1994.

1580. S.T. Walker, "Thoughts on Key Eserow Acceptability, " TIS Report #534D, Trusted Information Systems, Nov 1994.

1581. S.T. Walker, S.B. Lipner, C.M. Ellison, D.K. Branstad, and D.M. Balenson, "Commercial Key Escrow Something for Everyone Now and for the Future," TIS Report #541, Trusted Information Systems, Jan 1995.

1582. M.Z. Wang and J.L. Massey, "The Characteristics of All Binary Sequences with Perfect Linear Complexity Profiles," Abstracts of Papers, EUROCRYPT '86. 20-22 May 1986.

1583. E.J. Watson, "Primitive Polynomials (Mod 2)," Mathematics of Computation, v. 16, 1962, p. 368.

1584. P. Wayner, "Mimic Functions," Cryptologia, v. 16, n. 3, Jul 1992, pp. 193-214.

1585. P. Wayner, "Mimic Functions and Tractability, " draft manuscript, 1993.

1586. A.F. Webster and S.E. Tavares, "On the Design of S-Boxes," Advances in Cryptology CRYPTO '85 Proceedings, Springer-Verlag, 1986, pp. 523-534.

1587. G. Welchman, The Hut Six Story: Breaking the Enigma Codes, New York: McGraw-Hill, 1982.

1588. A.L. Wells Jr., "A Polynomial Form for Logarithms Modulo a Prime," IEEE Transactions on Information Theory Nov 1984, pp. 845-846.

1589. D.J. Wheeler, "A Bulk Data Encryption Algorithm," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 127-134.

1590. D.J. Wheeler, personal communication, 1994.

1591. D.J. Wheeler and R. Needham, "A Large Block DES-Like Algorithm," Technical Report 35S, "Two Cryptographic Notes," Computer Laboratory, University of Cambridge, Dec 1994, pp. 1-3.

1592. D.J. Wheeler and R. Needham, "TEA, A Tiny Encryption Algorithm, " Technical Report 355, "Two Cryptographic Notes," Computer Laboratory, University of Cambridge, Dee 1994, pp. 1-3.

1593. S.R. White, "Covert Distributed Processing with Computer Viruses, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 616-619.

1594. White House, Office of the Press Secretary, "Statement by the Press Secretary," 16 Apr 1993.

1595. B.A. Wichman and I.D. Hill, "An Efficient and Portable Pseudo-Random Number Generator," Applied Statistics, v. 31, 1982, pp. 188-190.

1596. M.J. Wiener, "Cryptanalysts of Short RSA Secret Exponents," IEEE Transactions on Information Theory, v. 36, n. 3, May 1990, pp. 553-5.58.

1597. M.J. Wiener, "Efficient DES Key Search." presented at the rump session of CRYPTO '93, Aug 1993.

1598. M.J. Wiener, "Efficient DES Key Search," TR-244, School of Computer Science, Car leton University, May 1994.

1599. M.V. Wilkes, Time-Sharing Computer Systems, New York: American Elsevier, 1968.

1600. E.A. Williams, An Invitation to C ryptograms, New York: Simon and Schuster, 1959.

1601. H.C. Williams, "A Modification of the RSA Public-Key Encryption Procedure, " IEEE Transactions on Information Theory, v. IT-26, n. 6, Nov 1980, pp. 726-729.

1602. H.C. Williams, "An Overview of Factoring," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 71-80.

1603. H.C. Williams, "Some Public-Key Crypto-Functions as Intractable as Factorization, " Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, pp. 66-70.

1604. H.C. Williams, "Some Public-Key Crypto-Functions as Intractable as Factorization," Cryptologia, v. 9, n. 3, Jul 1985, pp. 223-237.

1605. H.C. Williams "An M3 Public-Key Encryption Scheme," Advances in Cryptology CRYPTO 85, Springer-Verlag, 1986, pp. 358-368.

1606. R.S. Winternitz, "Producing One-Way Hash Functions from DES," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, 1984, pp. 203-207.

1607. R.S. Winternitz, "A Secure One-Way Hash Function Built from DES," Proceedings of the 1984 Symposium on Security and Privacy, 1984, pp. 88-90.

1608. S. Wolfram, "Random Sequence Generation by Cellular Automata," Advances in Applied Mathematics, v. 7, 1986, pp. 123-164.

1609. S. Wolfram, "Cryptography with Cellular Automata, " Advances in Cryptology CRYPTO '85 Proceedings , SpringerVerl ag, 1986, pp. 429 -432.

1610. T.Y.C. Woo and S.S. Lam, "Authentication for Distributed Systems," Computer, v. 25, n. 1, Jan 1992, pp. 39-52.

1611. T.Y.C. Woo and S.S. Lam, "'Authentication Revisited," Computer, v. 25, n.3, Mar 1992, p. 10.

1612. T.Y.C. Woo and S.S. Lam, "A Semantic Model for Authentication Protocols," Procee dings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy 1993, pp. 178-194.

1613. M.C. Wood, technical report, Cryptech, Inc., Jamestown, NY, Jul 1990.

1614. M.C. Wood, "Method of Cryptographically 1628. Transforming Electronic Digital Data from One Form to Another," U.S. Patent #5,003,596, 26 Mar 1991.

1615. M. C. Wood, personal communication, 1993.

1616. C.K. Wu and X.M. Wang, "Determination of the True Value of the Euler Totient Function in the RSA Cryptosystem from a Set of Possibilities," Electronics Letters, v. 29, n. 1, 7 Jan 1993, pp. 84-85.

1617. M.C. Wunderlich, "Recent Advances in the Design and Implementation of Large Integer Factorization Algorithms, " Proceedings of 1983 Symposium on Security and Privacy, IEEE Computer Society Press, 1983, pp. 67-71.

1618. Xerox Network System (XNS) Authentication Protocol, XSIS 098404, Xerox Corporation, Apr 1984.

1619. Y.Y. Xian, "New Public Key Distribution System," Electronics Letters, v. 23, n. 11, 1987, pp. 560-561.

1620. L.D. Xing and L.G. Sheng, "Cryptanalysts of New Modified Lu-Lee Cryptosystems," Electronics Letters, v. 26, n. 19, 13 Sep 1990, p. 1601-1602.

1621. W. Xinmei, "Digital Signature Scheme Based on Error-Correcting Codes, " Electronics Letters, v. 26, n. 13, 21 Jun 1990, p. 1634. 898-899.

1622. S.B. Xu, INK. He, and X.M. Wang, "An Implementation of the GSM General Data Encryption Algorithm A5, " CHINACRYPT '94, Xidian, China, 11-15 Nov 1994, pp. 287-291. (In Chinese.)

1623. M. Yagisawa, "A New Method for Realizing Public-Key Cryptosystem," Cryptologia, v. 9, n. 4, Oct 1985, pp. 360-380.

1624. C.H. Yang, "Modular Arithmetic Algorithms for Smart Cards, " IEICE Japan, Technical Report, ISEC92-16, 1992.

1625. C.H. Yang and H. Morita, "An Efficient Modular-Multiplication Algorithm for Smart-Card Software Implementation, " IEICE Japan, Technical Report, ISEC91-58, 1991.

1626. J.H. Yang, K.C. Zeng, and Q.B. Di, "On the Construction of Large S-Boxcs, " CHINACRYPT'94, Xidian, China, 11-15 Nov 1994, pp. 24-32. (In Chinese.)

1627. A.C.-C. Yao, "Protocols for Secure Computations, " Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science, 1982, pp. 160 -164.

1628. B. Yee, "Using Secure Coprocessors, " Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, May 1994.

1629. S.-M. Yen, "Design and Computation of Public Key Cryptosystems," Ph. D. dissertation, National Cheng Hung University, Apr 1994.

1630. S.-M. Yen and C.-S. Lai, "New Digital Signature Scheme Based on the Discrete Logarithm," Electronics Letters, v. 29, n. 12, 1993, pp. 1120-1121.

1631. K. Yin and K. Peterson, "A Single-Chip VLSI Implementation of the Discrete Exponential Public-Key Distribution System, " IBM Systems journal, v. 15, n. 1, 1982, pp. 102-116.

1632. K. Yiu and K. Peterson, "A Single-Chip VLSI Implementation of the Discrete Exponential Public-Key Distribution System," Proceedings of Government Microcircuit Applications Conference, 1982, pp. 18-23.

1633. H.Y. Youm, S.L. Lee, and M.Y. Rhee, "Practical Protocols for Electronic Cash," Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography Seoul, Korea, 24-26 Oct 1993, pp. 10-22.

1634. M. Yung, "Cryptoprotocols: Subscriptions to a Public Key, the Secret Blocking, and the Multi-Player Mental Poker Game, " Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, 1985, 439-453.

1635. G. Yuval, "How to Swindle Rabin," Cryptologia, v. 3, n. 3, Jul 1979, pp. 187-190.

1636. K.C. Zeng and M. Huang, "On the Linear Syndrome Method in Cryptanalysis, " Advances in Cryptology CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 469-478.

1637. K.C. Zeng, M. Huang, and T.R.N. Rao, "An Improved Linear Algorithm in Cryptanalysis with Applications, " Advances in Cryptology CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 34-47.

1638. K.C. Zeng, C.-H. Yang, and T.R.N. Rao, "On the Linear Consistency Test ILCTl in Cryptanalysis with Applications," Advances in Cryptology CRYPT O '89 Proceedings, Springer-Verlag, 1990, pp. 164-174.

1639. K.C. Zeng, C.-H. Yang, L. Wei, and T. R.N. Rao, "Pseudorandom Bit Generators in Stream-Cipher Cryptography, " IEEE Computer, v. 24, n. 2, Feb libel, pp. 5-17.

1640. M. Zhang, S.E. Tavares, and L.L. Campbell, "Information Leakage of Boolean Functions and Its Relationship to Other Cryptographic Criteria," Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security, ACM Press, 1994, pp. 156-165.

1641. M. Zhang and G. Xiao, "A Modified Dcsign Criterion for Stream Ciphers," CHINACRYPT'94, Xidian, China, 11-15 Nov 1994, pp. 201-209. (In Chinese.)

1642. Y. Zheng, T. Matsumoto, and H. Imai, "Duality between two Cryptographic Primitives," Papers of Technical Group for Information Security, IEICE of Japan, Mar 1989, pp. 47-57.

1643. Y. Zhcng, T. Matsumoto, and H. Imai, "Impossibility and Optimality Results in Constructing Pseudorandom Permutations," Advances in Cryptology EURO CRYPT '89 Proceedings, Springer-Verlag. 1990, pp. 412-422.

1644. Y. Zheng, T. Matsumoto, and H. Imai, "On the Construction of Block Ciphers Provably Sccurc and Not Relying on Any Unproved Hypotheses, " Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, 1990, pp. 461-480.

1645. Y. Zheng, T. Matsumoto, and H. Imai, "Duality hctwccn two Cryptographic Primitives," Proceedings of the 8th International Conference on Applied Algebra. Algebraic Algorithms and Error-Correcting Codes, Springer-Verlag, 1991, pp. 379-390.

1646. Y. Zheng, J. Pieprzyk, and J. Seberry, "HAVAL A One-Way Hashing Algorithm with Variable Length of Output, " Advances in Crytology AUSCRYPT '92 Proceedings, Springer-Verlag, 1993, pp. 83-104.

1647. N. Zierler, "Linear Recurring Sequences," Journal Soc. Indust. Appl. Math., v. 7, n 1, Mar 1959, pp. 31 48.

1648. N. Zierler, "Primitive Trinomials Whose Degree Is a M ersenne Exponent," Information and Control, v. 15, 1969, pp. 67-69.

1649. N. Zierler and J. Brillhart, "On Primitive Trinomials (mod 2)," Information and Control, v. 13, n. 6, Dec 1968, pp. 541-544.

1650. N. Zicrlcr and W.H. Mills, "Products of Linear Recurring Scqucnces," Journal of Algebra, v. 27, n. 1, Oct 1973, pp. 147-157.

1651. C. Zimmer, "Perfect Gibberish," Discover, v. 13, n. 12, Dec 1992, pp. 92-99.

1652. P. Zimmermann, The Official PGP User's Guide, Boston: MIT Press, 1995.

1653. P. Zimmermann, PGP Source Code and Internals, Boston: MIT Press, 1995.