КРИПТОГРАФИЯ
И БЕЗОПАСНОСТЬ СЕТЕЙ

Behrouz A. Forouzan

INTRODUCTION TO CRYPTOGRAPHY AND NETWORK SECURITY

Higher Education

Boston Burr Ridge, IL Dubuque, IA New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

Основы информационных технологий

Бехроуз А. Фороузан

КРИПТОГРАФИЯ И БЕЗОПАСНОСТЬ СЕТЕЙ

Учебное пособие

Перевод с английского под редакцией А.Н. Берлина

ЭКОМ

Интернет-Университет
Информационных Технологий
www.intuit.ru

БИНОМ.
Лаборатория знаний
www.lbz.ru

Москва
2010

УДК 003.26(075.8)
ББК 32.973.26-018.2я73-1+32.811.4я73-1
Ф79

Фороузан Б.А.

Ф79 Криптография и безопасность сетей: Учебное пособие / Фороузан Б.А.; пер. с англ. под ред. А.Н. Берлина. — М.: Интернет-Университет Информационных Технологий : БИНОМ. Лаборатория знаний, 2010. — 784 с.: ил., табл. — (Основы информационных технологий).

ISBN 978-5-9963-0242-0

В этом издании известный автор Бехроуз А. Фороузан в доступном стиле рассказывает о концепциях криптографии и безопасности сети.

В представленных лекциях студенты могут освоить необходимую математическую основу для изучения следующего за ним материала; эти математические лекции могут быть пропущены, если студенты обладают соответствующими знаниями.

Для студентов, аспирантов и всех тех, кто хотел бы повысить свою квалификацию в области телекоммуникаций.

Авторизированный перевод с английского издания, опубликованного компанией *McGraw Hill. Higher Education*.

УДК 003.26(075.8)
ББК 32.973.26-018.2я73-1+32.811.4я73-1

По вопросам приобретения обращаться:
«БИНОМ. Лаборатория знаний»
Телефон (499) 157-1902, (499) 157-5272,
e-mail: binom@Lbz.ru, http://www.Lbz.ru

ISBN 978-5-9963-0242-0

О проекте

Интернет-Университет Информационных Технологий – это первое в России высшее учебное заведение, которое предоставляет возможность получить дополнительное образование во Всемирной сети. Web-сайт университета находится по адресу www.intuit.ru.

Мы рады, что вы решили расширить свои знания в области компьютерных технологий. Современный мир – это мир компьютеров и информации. Компьютерная индустрия – самый быстрорастущий сектор экономики, и ее рост будет продолжаться еще долгое время. Во времена жесткой конкуренции от уровня развития информационных технологий, достижений научной мысли и перспективных инженерных решений зависит успех не только отдельных людей и компаний, но и целых стран. Вы выбрали самое подходящее время для изучения компьютерных дисциплин. Профессионалы в области информационных технологий сейчас востребованы везде: в науке, экономике, образовании, медицине и других областях, в государственных и частных компаниях, в России и за рубежом. Анализ данных, прогнозы, организация связи, создание программного обеспечения, построение моделей процессов – вот далеко не полный список областей применения знаний для компьютерных специалистов.

Обучение в университете ведется по собственным учебным планам, разработанным ведущими российскими специалистами на основе международных образовательных стандартов Computer Curricula 2001 Computer Science. Изучать учебные курсы можно самостоятельно по учебникам или на сайте Интернет-Университета, задания выполняются только на сайте. Для обучения необходимо зарегистрироваться на сайте университета. Удостоверение об окончании учебного курса или специальности выдается при условии выполнения всех заданий к лекциям и успешной сдачи итогового экзамена.

Книга, которую вы держите в руках, – очередная в многотомной серии «Основы информационных технологий», выпускаемой Интернет-Университетом Информационных Технологий. В этой серии будут выпущены учебники по всем базовым областям знаний, связанным с компьютерными дисциплинами.

Добро пожаловать в
Интернет-Университет Информационных Технологий!

Анатолий Шкред
anatoli@shkred.ru

Об авторе

Бехроуз А. Фороузан окончил Калифорнийский Университет. В настоящее время преподает в De Anza College, где ведет курсы по компьютерным информационным системам. Кроме того, он работает как системный консультант в различных фирмах. Известен как автор весьма востребованных книг по программированию и построению сетей телекоммуникаций.

Лекции

Оглавление

Предисловие к русскому переводу

Поскольку обмен информацией в современном мире представляет одну из центральных задач. Утечка, искажение, отказ от сообщений часто приводят к катострофическим последствиям. Поэтому современные сети Интернет и мобильной связи не могут использоваться без средств защиты.

Книга Берхоуза Фороузана — одно из наиболее полных изданий. Будучи преподавателем одного из ведущих университетов США, он написал очень содержательную и понятную книгу, несмотря на сложность материала.

Книга представляет собой часть учебной программы, которая содержит следующие курсы лекций.

1. Оконечные устройства и линии абонентского участка информационной сети.
2. Абонентские устройства и технологии высокоскоростных сетей.
3. Телекоммуникационные сети и устройства.
4. Основные протоколы Интернет.
5. Криптография и безопасность сетей.

Читатель получает возможность последовательного и взамосвязанного обучения основам построений современных информационных сетей в указанном выше порядке.

Представленный перевод дополняет предыдущие курсы лекций того же автора. По своей структуре книга весьма близка к уже опубликованным и перечисленным выше в этой серии (краткие итоги, список терминов, глоссарий).

Книгу сопровождают наборы вопросов для самопроверки. Чтобы на них ответить, требуется определенный уровень владения материалом.В этом случае при составлении вопросов было учтено мнение студентов (см. форум), которые выразили пожелание усложнить вопросы.

Автор благодарит руководителя издательского отдела Наталью Рахманову и редактора Светлану Перепелкину за внимательную работу над рукописью и доброжелательное отношение.

Моим любимым дочери
Сэйтаре и зятю Шану.
Б. Фороузан

Предисловие

Всемирная сеть Интернет изменила во многом нашу повседневную жизнь. Новые формы коммерческой деятельности позволяют делать покупки в магазине, не выходя из дома. Всемирная Паутина (WWW) дает возможность пользоваться общей информацией. Технология Электронной почты (e-mail) соединяет людей из самых удаленных уголков мира. Но как результат, этот неизбежный прогресс породил зависимость современного общества от Интернета.

Интернет, доступная всем система обмена информацией, создал проблемы информационной безопасности. При обмене важными сведениями необходимы конфиденциальность, целостность и удостоверение их подлинности. Когда люди делают покупки путем диалога в Интернете, они должны быть убеждены, что средства связи обеспечивают конфиденциальность. Они должны быть уверены, что продавцы, ведущие с ними диалог, подлинны. Когда клиенты совершают операции с банками, запрашивают информацию и получают ответ, они хотят точно знать, что целостность сообщения сохранена и цифры не искажены.

Безопасность сети обеспечивается набором протоколов, которые позволяют нам спокойно использовать Интернет, не думая о возможных атаках на нарушение информации. Самый общий инструмент для того, чтобы обеспечить безопасность сети, – криптография. Это старая техника, которая была восстановлена и приспособлена к современной сети. Наша книга сначала представляет читателю принципы криптографии, а затем показывает их применение для описания протоколов безопасности сети.

Особенности книги

Некоторые особенности книги объясняются попыткой облегчить читателю понимание принципов безопасности сети и криптографии.

Структура

Книга использует подход к обучению безопасности сети и криптографии по нарастающей сложности. Она не предполагает у читателя никаких особых и специальных знаний математики типа теории чисел или абстрактной алгебры. Однако криптография и безопасность сети не могут быть поняты без знания некоторых основ в соответсвующих областях математики, и эта базовая информация приводится в лекциях 2, 4 и 9. Читатели, знакомые с этими вопросами, могут пропустить указанные лекции. Лекции 1-15 рассматривают вопросы криптографии, а лекции 16-18 — проблемы безопасности сети.

Визуальный подход

Книга рассматривает сложные технические вопросы, не используя сложных формул, за счет баланса между текстом и рисунками. Текст сопровождают более

чем 400 рисунков, обеспечивающих визуальные и интуитивные возможности для понимания материалов. Рисунки особенно важны при объяснении трудных понятий шифрования и сложных сетевых протоколов безопасности.

Алгоритмы

Алгоритмы играют важную роль в обучении шифрованию. Чтобы сделать изложение независимым от конкретного компьютерного языка, алгоритмы даются в псевдокоде, который может быть легко запрограммирован на современном языке.

Важные понятия выделены шрифтом для справки, а также чтобы привлечь непосредственное внимание читателя.

Примеры

Каждая лекция содержит большое количество примеров, которые иллюстрируют применение понятий, обсуждаемых в этой лекции. Некоторые примеры просто показывают, как использовать понятия и формулы; другие иллюстрируют практическое соотношение шифров; третьи дают дополнительную информацию для лучшего понимания некоторых трудных идей.

Рекомендованная литература

В конце каждой лекции читатель найдет список книг для самостоятельного чтения.

Ключевые термины

Ключевые термины выделены полужирным шрифтом в тексте лекции, и список ключевых терминов приводится в конце каждой лекции. Все ключевые термины также определены в глоссарии, в конце книги.

Каждая лекция заканчивается краткими итогами материала, то есть обзором всех важных пунктов в лекции.

Набор для практики

В конце каждой лекции студенты найдут набор для практики, разработанный с целью закрепить существенные понятия. Набор для практики состоит из двух частей: вопросы и упражнения. Вопросы предназначены для того, чтобы проверить первичное понимание читателем материала, представленного в лекции. Упражнения требуют более глубокого понимания материала.

Приложения

Приложения содержат справочные материалы или краткое изложение материалов, которые необходимы для понимания положений, обсуждаемых в этой книге. Рассмотрение некоторых математических тем также представлено в приложениях — это дает возможность читателю, уже знакомому с этими вопросами, пропустить их чтение.

Доказательства

Математические понятия, упомянутые в лекциях, приводятся без доказательств, только для того, чтобы показать результаты применения этих понятий. Заинтересованный читатель может найти доказательства в приложении Q.

Глоссарий и Список сокращений

В конце текста читатель найдет обширный глоссарий и список сокращений.

Содержание

После вводной лекции 1 книга разделена на четыре части.

Первая часть: Шифрование с помощью симметричных ключей

Первая часть дает представление о криптографии с симметричными ключами (симметричное шифрование) — как о традиционном, так и о современном подходе к процессу шифрования. Лекции в этой части посвящены использованию симметричного шифрования для обеспечения секретности информации. Первая часть включает в себя лекции 2-8.

Вторая часть: Шифрование с помощью асимметричных ключей

Вторая часть рассматривает шифрование с помощью асимметричных ключей (асимметричное шифрование). Лекции в этой части показывают, как асимметричное шифрование может обеспечить безопасность информации. Вторая часть включает в себя лекции 9 и 10.

Третья часть: Целостность, удостоверение подлинности и управление ключами

Третья часть показывает, как криптографические функции хэширования[1] (hashing) могут обеспечить другие функции безопасности, такие как целостность сообщения и удостоверения подлинности сообщения. Лекции в этой части также показывают, как асимметричное шифрование и асимметричное шифрование могут дополнять друг друга.

Третья часть включает в себя лекции 11-15.

Четвертая часть: Безопасность сети

Четвертая часть показывает, как криптография, рассмотренная в первой и третьей частях, может использоваться для создания протоколов безопасности сети на трех уровнях модели организации сети Интернета.

Четвертая часть включает в себя лекции 16-18.

Как использовать эту книгу

Эта книга написана и для студенческой, и для профессиональной аудитории. Заинтересованные профессионалы могут использовать её для самостоятельного обучения. Как учебник она может применяться для чтения лекций в течение одного семестра или целого курса. Ниже приводятся некоторые руководящие принципы изучения.

- Части 1-3 настоятельно рекомендуются для изучения.
- Часть 4 рекомендуется, если курс должен продолжиться за рамками криптографии в области безопасности сети. Для чтения четвертой части следует предварительно изучить курс организации сетей.

Онлайновый Учебный Центр

Онлайновый Учебный Центр McGraw-Hill предоставляет много дополнительных материалов, связанных с темой «Безопасность Сети и Криптография». Сайт доступен по следующей ссылке www.mhhe.com/forouzan.

[1] Функция, представляющая собой отображение фрагментов открытого текста в шифрованную строку фиксированной длины.- *прим. перев.*

Профессора и студенты могут получить доступ к материалам лекций, например, к слайдам Power Point. Студенты могут увидеть решения задач с нечетными номерами, а преподаватели могут использовать пароль для доступа ко всем решениям Дополнительно McGraw-Hill облегчает создание web-сайта (на примере которого читатель сможет лучше усвоить данный курс) с помощью эксклюзивного продукта фирмы McGraw-Hill, который называется PageOut (внешние страницы). Он не требует никакой предварительной подготовки и знания HTML, больших затрат времени, навыков проектирования web-сайтов.

Для этого PageOut предлагает ряд шаблонов. Нужно просто заполнить их информацией вашего курса и щелкнуть «мышкой» на поле одного из 16 проектов. Этот процесс отнимает времени менее часа, и в результате читатель получит профессионально разработанный web-сайт. PageOut предоставляет возможность «мгновенной» разработки законченного web-сайта и обеспечивает мощные функциональности.

Интерактивное содержание курса позволяет вам получить доступ на экране к содержанию вашей лекции. Когда студенты посещают ваш web-сайт PageOut, ваша программа направляет их к компонентам Онлайнового Учебного Центра Фороузана или исключительно к вашему собственному материалу.

Благодарности

Очевидно, что для создания этой книги была необходима поддержка многих людей.

Рецензенты

Самый важный вклад в создание книги внесли рецензенты. Я не могу выразить в словах мою благодарность многочисленным рецензентам, которые затратили многие часы, читая рукопись и обогащая меня полезными комментариями и идеями. Я особенно хотел бы высказать признательность следующим специалистам:

Kaufman, Robert, University of Texas, San Antonio
Kesidis, George, Penn State
Stephens, Brooke, University of Maryland, Baltimore County
Koc, Cetin, Oregon State University
Uminowicz, Bill, Westwood College
Wang, Xunhua, James Madison University
Kak, Subhash, Louisiana State University
Dunigan, Tom, University of Tennessee, Knoxville

Сотрудники издательства McGraw-Hill

Хочу выразить особую благодарность сотрудникам издательства McGraw-Hill, издателю Алану Апту (Alan Apt), который доказал, что может делать невозможное возможным. Мелинда Билецки (Melinda Bilecki), технический редактор, оказывала мне помощь всегда, когда я нуждался в этом. Шейла Франк (Sheila Frank), менеджер проекта, организовала процесс издания с огромным энтузиазмом. Я также благодарю Давида Хэша (David Hash), разработчика проекта, Кара Кудронович (Kara Kudronowicz), работающей в производстве книг, и Венди Нелсон (Wendy Nelson) — редактора печати.

Бехроуз Фороузан (Behrouz A. Forouzan)

Лекция 1. Введение

Цели и содержание

Эта лекция преследует несколько целей.

- Определить три цели информационной безопасности.
- Определить виды атак на безопасность информации, которые угрожают секретности.
- Определить службы безопасности и как они связаны с тремя задачами безопасности.
- Определить механизмы безопасности, которые обеспечивают службы секретности.
- Познакомить с двумя методами шифрования для реализации механизма безопасности — криптографией и стеганографией.

Мы живем в информационную эпоху. Мы должны внимательно сохранять информацию о каждом аспекте нашей жизни. Другими словами, информация — собственность, и, подобно любой другой собственности, имеет важное значение. И в этом качестве информация должна быть защищена от нападений.

Информация должна быть сохранена от неправомочного доступа (*конфиденциальность*), защищена от неправомочного изменения (*целостность*) и доступна только разрешенному объекту, когда это ему необходимо (*готовность*).

Несколько десятилетий назад информация собиралась на физических носителях. Конфиденциальность таких носителей достигалась строгим ограничением доступа, который предоставлялся только людям, имеющим на это право, и тем из них, кому можно было доверить данную информацию. Также нескольким правомочным субъектам разрешалось изменение содержания этих файлов. Готовность была обеспечена тем, что по меньшей мере одному человеку всегда разрешался доступ к носителям.

С появлением компьютеров хранение информации стало электронным.

Информация хранилась уже не в физической неэлектронной среде — она накапливалась в электронной среде (компьютерах). Однако три требования безопасности не изменились. Файлы, записанные в компьютере, должны обладать свойствами конфиденциальности, целостности и готовности. Реализация этих требований возможна различными методами и требует решения сложных задач.

В течение прошлых двух десятилетий компьютерные сети произвели революцию в использовании информации. Информация теперь распределена. Люди при наличии полномочий могут передавать информацию и искать ее на расстоянии, используя компьютерные сети. Но три уже упомянутых требования — конфиденциальности, целостности и готовности — не изменились. Они лишь приобрели некоторые новые аспекты. Теперь недостаточно того, что информация должна быть конфиденциальной, когда она сохраняется в компьютере. Должен также существовать способ поддержки конфиденциальности, когда эта информация передается от одного компьютера к другому.

В этой лекции мы сначала обсуждаем три главных цели поддержки безопасности информации.

Когда будет понятно, какие атаки могут угрожать этим трем целям, тогда можно обсудить службы безопасности, предназначенные для этих целей. Потом определяются механизмы обеспечения службы безопасности и методы, которые могут использоваться, чтобы осуществить эти механизмы.

1.1. Цели поддержки безопасности

Рассмотрим **три цели поддержки информационной безопасности: конфиденциальность, целостность** и **готовность.**

Рис. 1.1. Систематизация целей поддержки безопасности

Конфиденциальность

Конфиденциальность — вероятно, самый общий аспект информационной безопасности. Мы должны защитить нашу конфиденциальную информацию. Организация должна принять меры против тех злонамеренных действий, которые могут нарушить конфиденциальность информации. В военных организациях сохранение секретности важной информации – главная забота руководства. В промышленности сохранение тайны некоторой информации от конкурентов является одним из основных факторов работы организации. В банковском деле должна сохраняться секретность учетных записей клиентов.

Как мы увидим позже в этой лекции, конфиденциальность нужна не только при хранении информации, она также необходима при ее передаче. Когда мы передаем часть информации, которая должна будет храниться в удаленном компьютере, или когда мы отыскиваем информацию, которая находится в удаленном компьютере, мы должны гарантировать ее секретность в течение передачи.

Целостность

Потребители изменяют информацию постоянно. В банке, когда клиент вносит или снимает деньги, баланс на его счету должен быть изменен. **Целостность** означает, что изменения должны быть сделаны только разрешенными объектами и с помощью разрешенных механизмов. Нарушение целостности — не обязательно результат злонамеренного действия; сбой в системе, такой, например, как всплеск или прерывание мощности в первичной сети электропитания, может привести к нежелательным изменениям некоторой информации.

Готовность

Третий компонент информационной безопасности — **готовность.** Информация, созданная и сохраненная организацией, должна быть доступна разрешенным

объектам. Информация бесполезна, если она не доступна. Информация должна постоянно изменяться и поэтому тоже должна быть доступна для разрешенных объектов. Неготовность информации столь же вредна для организации, как отсутствие конфиденциальности или целостности. Вообразите, что случилось бы с банком, если клиенты не могли бы обратиться к своим счетам для снятия или вклада денег.

1.2. Атаки

Нашим трем целям информационной безопасности — конфиденциальности, целостности и готовности — могут угрожать **атаки[1] с целью нарушения** безопасности информации. Хотя в литературе встречаются различные подходы к систематизации атак, мы сначала разделим их на три группы, связанные с целями нарушения информационной безопасности. Позже мы будем делить их на две широких категории, основанные на эффективности их воздействия на систему. Рисунок 1.2 показывает первую систематизацию.

Рис. 1.2. Систематизация атак и соотношение их с целями поддержки информационной безопасности

[1] Атака – попытка злоумышленника вызвать отклонения от нормального протекания информационного процесса.

Атаки, угрожающие конфиденциальности

Вообще, имеется два типа атак, которые угрожают конфиденциальности информации: **вмешательство** и **наблюдение за трафиком и его анализ.**

Вмешательство

Вмешательство относится к неправомочному доступу или перехвату данных. Например, файл, передаваемый через Интернет, может содержать конфиденциальную информацию. Объект, не имеющий полномочий, может прервать передачу и использовать передаваемую информацию для собственной выгоды. Чтобы предотвратить вмешательство, данные могут быть представлены так, что перехвативший не сможет понять их. Для этого применимы методы шифрования, приводимые в этой книге.

Наблюдение за трафиком и его анализ

Хотя шифрование данных может сделать их непонятными для злоумышленника, он, анализируя сетевой трафик, может получить некоторую другую информацию. Например, он может найти электронный адрес (адрес электронной почты) передатчика или приемника. Он может также собрать пары запросов и ответов, что поможет ему понять активность действий наблюдаемой организации.

Атаки, угрожающие целостности

Целостности данных можно угрожать несколькими видами атак, такими как **модификация, имитация источника, повторная передача информации** и **отказ от сообщения.**

Модификация

После прерывания или доступа к информации атакующий изменяет информацию с определенной выгодой для себя. Например, клиент передает сообщение банку, чтобы провести некоторую операцию. Атакующий прерывает сообщение и изменяет тип операции, чтобы принести этим пользу себе. Обратите внимание, что иногда атакующий просто удаляет или задерживает сообщение, чтобы навредить системе или извлечь выгоду из самого факта задержки операции.

Имитация источника

Имитация источника (spoofing) заключается в том, что атакующий имитирует кого-то, кто имеет право на производимые действия. Например, атакующий захватывает банковскую кредитную карточку и PIN-код клиента банка. Он может действовать как настоящий клиент. Иногда атакующий имитирует приемник. Например, пользователь хочет войти в контакт с банком, но ему предоставляется другой сайт, который имитирует, что это – банк, и атакующий получает необходимую ему информацию от пользователя.

Повторная передача информации

Другой вид атаки — повторная передача информации (атака воспроизведения). Атакующий получает копию сообщения, передаваемого пользователем, и

передает эту копию с целью дезорганизации процесса или попыток повторить его. Например, человек передает запрос банку, чтобы оплатить работу своего сотрудника. Атакующий перехватывает сообщение и передает его снова, чтобы получить оплату этой работы от этого банка еще раз (повторно).

Отказ от сообщения

Этот тип атак отличается от других, потому что действие может быть выполнено одной из двух сторон связи: передатчиком или приемником. Передатчик сообщения может отрицать факт передачи. Другой вариант — когда приемник сообщения отрицает, что он получил сообщение.

Пример отказа от сообщения передатчика: клиент банка передал запрос банку на перевод некоторой суммы денег третьему лицу, но впоследствии отрицает, что он сделал такой запрос.

Пример опровержения сообщения приемником: человек, который покупает изделие у изготовителя, платит за это с помощью электроники, но изготовитель позже отрицает, что получил оплату, и просит оплатить покупку.

Атаки, угрожающие готовности

Рассмотрим только одну атаку, угрожающую готовности: **отказ в обслуживании.**

Отказ в обслуживании

Отказ в обслуживании (Denial of Service — DoS) — очень общее название атаки. Она может замедлить или полностью прервать обслуживание системы. Возможны несколько стратегий, чтобы достигнуть этого. Атакующий может передать так много фиктивных запросов серверу, что это приведет к сбою сервера из-за высокой нагрузки. Атакующий может также прервать и удалить ответ сервера клиенту, порождая у клиента впечатление, что сервер не отвечает. Атакующий может прервать запросы от клиентов, порождая у клиента уверенность, что сервер не отвечает. Атакующий может прерывать запросы клиентов, заставляя клиентов передать запросы много раз и перезагружать систему.

Пассивные и активные атаки

Давайте теперь разделим атаки на две группы: **пассивные** и **активные**. Таблица 1.1 показывает соотношения между этим делением и предыдущей классификацией.

Пассивные нападения

При пассивном нападении цель атакующего состоит в том, чтобы только получить информацию. Это означает, что нападение не изменяет данные и не повреждает систему. Система продолжает нормально работать, однако атака может нанести вред передатчику или приемнику сообщения. Атаки, которые угрожают конфиденциальности — вмешательство и наблюдение за трафиком плюс его анализ, — являются пассивными. Раскрытие информации может вредить передатчику или приемнику сообщения, но систему не затрагивает. По этой причине труд-

но обнаружить такой тип нападения, пока передатчик или приемник не узнают об утечке конфиденциальной информации. Пассивные нападения, однако, могут быть предотвращены шифрованием данных.

Таблица 1.1. Классификация пассивных и активных атак

Атаки	Пассивные / Активные	Угроза
Вмешательство Наблюдение за трафиком и его анализ	Пассивные	Конфиденциальности
Модификация Имитация источника Повторная передача информации Отказ от сообщения	Активные	Целостности
Отказ в обслуживании	Активные	Готовности

Активные атаки

Активные атаки изменяют данные или повреждают систему. Атаки, которые угрожают целостности или готовности, — активные. Активные атаки обычно легче обнаруживаются, чем предотвращаются, потому что атакующий может начинать их разнообразными методами.

1.3. Услуги и механизмы

Международный Союз Электросвязи, Секция Стандартов по телекоммуникации (ITU-T) (см. приложение В) разработал стандарты некоторых служб безопасности и некоторые механизмы[1] для осуществления этих услуг. Службы информационной безопасности и механизмы близко связаны, потому что механизм или комбинация механизмов применяются, чтобы обеспечить обслуживание. Механизм может использоваться в одной или нескольких услугах. Здесь эти механизмы кратко обсуждаются, чтобы понять их общую идею. Далее они будут рассмотрены более подробно.

Услуги информационной безопасности

ITU-T (X.800) определил пять услуг, связанных с целями информационной безопасности и атаками, типы которых мы определили в предыдущих секциях. Рисунок 1.3 показывает классификацию пяти общих услуг.

Чтобы предотвратить атаки на информационную безопасность, о которых мы говорили, надо просто иметь одну или больше показанных ниже услуг для одного или большего количества целей информационной безопасности.

[1] Механизм – алгоритм или процедура, основанные на выполнении ряда шагов, которые следуют друг за другом *(прим. ред.)*.

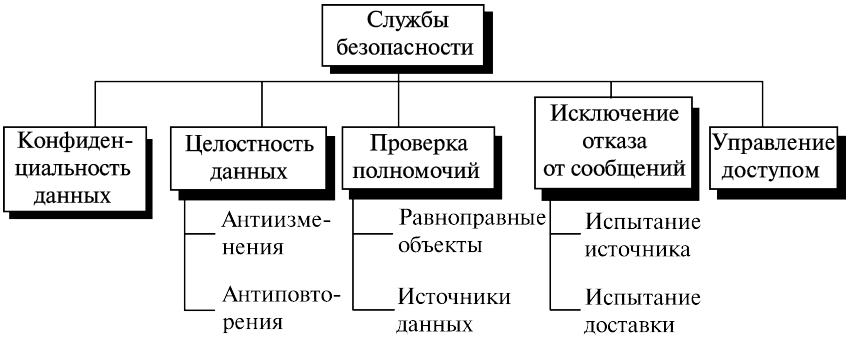


Рис. 1.3. Услуги информационной безопасности

Конфиденциальность данных

Конфиденциальность данных разработана, чтобы защитить данные от попытки их раскрытия. Эта широкая услуга, определенная в рекомендации ITU-T(X.800). Она может охватывать конфиденциальность целого сообщения или его части, а также защищает от наблюдения за трафиком и его анализа — собственно, она создана для предотвращения вмешательства и наблюдения за трафиком.

Целостность данных

Целостность данных разработана для защиты данных от модификации, вставки, удаления и повторной передачи информации противником. Она может защищать целое сообщение или часть сообщения.

Установление подлинности (аутентификация)

Эта услуга обеспечивает **установление подлинности (аутентификацию)** оператора на другом конце линии. При соединении, ориентированном на подключение, она гарантирует установление подлинности передатчика или приемника в течение установления соединения (установление подлинности объектов равного уровня). При соединении без установления подключения она подтверждает подлинность источника данных (установление подлинности происхождения данных).

Исключение отказа от сообщений

Услуга **исключение отказа от сообщений** защищает от отказа от сообщения передатчиком или приемником данных. При исключении отказа от сообщения передатчиком приемник данных может потом доказать происхождение сообщения, используя опознавательный код (идентификатор) передатчика. При исключении отказа от сообщений приемником передатчик, используя подтверждение доставки, может потом доказать, что данные доставлены предназначенному получателю.

Управление доступом

Управление доступом обеспечивает защиту против неправомочного доступа к данным. *Доступ* в этом определении — термин очень широкий и может включать чтение, запись, изменение данных, запуск выполнения программы и так далее.

Механизмы безопасности

Для обеспечения услуг информационной безопасности стандарт ITU-T (X.800) рекомендует некоторые **механизмы безопасности**, определенные в предыдущей секции. Рисунок 1.4 дает классификацию этих механизмов.

Шифрование

Засекречивая или рассекречивая данные, можно гарантировать конфиденциальность. Шифрование также дополняет другие механизмы, которые обеспечивают другие услуги. Сегодня для шифрования используются два метода: криптография и стеганография[1] — тайнопись (steganography). Мы коротко обсудим их в дальнейшем.

Целостность данных

Механизм **целостности данных** добавляет в конце данных короткий контрольный признак (check value), который создается определенным процессом отдельно от данных. Приемник получает данные и контрольный признак. На основании полученных данных он создает новый контрольный признак и сравнивает только что созданный с полученным. Если эти два контрольных признака совпадают, целостность данных была сохранена.

Цифровая подпись

Цифровая подпись — средство, которым отправитель может с помощью электроники подписать данные, а приемник может с помощью компьютера проверить подпись. Отправитель использует процесс, который может указать, что эта подпись имеет частный ключ, выбранный из общедоступных ключей, которые были объявлены публично для общего пользования. Приемник использует общедоступный ключ отправителя, чтобы доказать, что сообщение действительно подписано отправителем, который утверждает, что послал сообщение.

Обмен сообщениями для опознавания

При **обмене сообщениями для опознавания** два объекта обмениваются некоторыми сообщениями, чтобы доказать, что эти объекты известны друг другу. Например, одно юридическое лицо может доказать, что оно знает тайный признак, который только оно может знать (скажем, последнее место встречи с партнером).

Заполнение трафика

Заполнение трафика означает возможность вставлять в трафик данных некоторые фиктивные данные, чтобы сорвать попытки злоумышленников использовать его для анализа.

---

[1] Стеганография (тайнопись) – наука о скрытой передаче информации путем **сокрытия самого факта передачи**. В отличие от криптографии, которая скрывает содержание сообщения, стеганография скрывает существование сообщения.

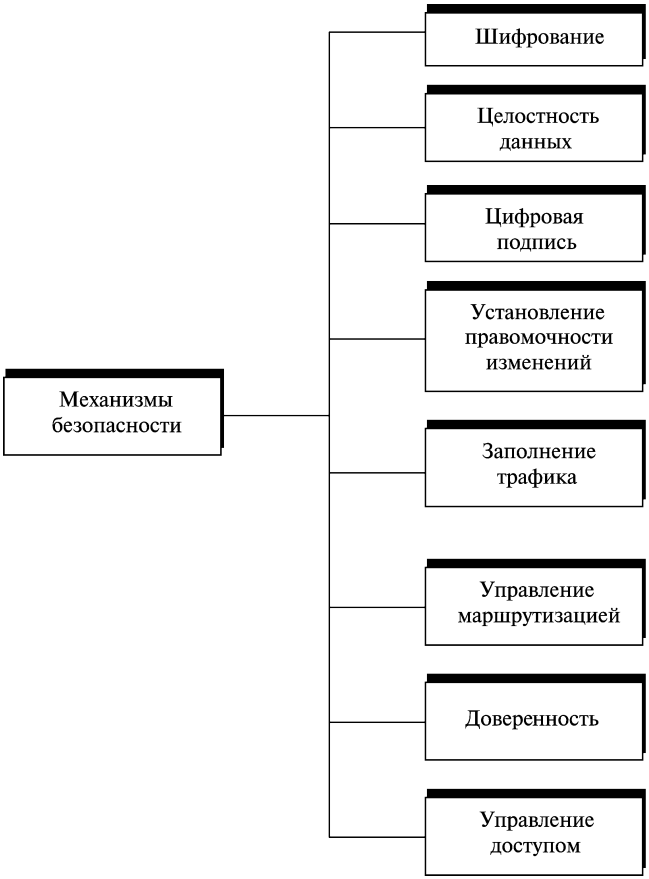


Рис. 1.4. Механизмы информационной безопасности

Управление маршрутизацией

Управление маршрутизацией означает выбор и непрерывное изменение различных доступных маршрутов между отправителем и приемником для того, чтобы препятствовать противнику в перехвате информации на определенном маршруте.

Доверенность

Доверенность означает выбор третьей стороны, с целью доверить ей контроль обменом между двумя объектами. Это может быть сделано, например, для того, чтобы предотвратить отказ от сообщения. Приемник может вовлечь третью сторону, которой можно доверить хранение запросов отправителя и тем самым предотвратить последующее отрицание отправителем факта передачи сообщения.

Управление доступом

Управление доступом идоказывает своими методами, что пользователь имеет право доступа к данным или ресурсам, принадлежащим системе. Примеры такого доказательства — пароли и PIN-коды.

Соотношение между услугами и механизмами

Таблица 1.2 показывает соотношение между услугами и механизмами информационной безопасности. Приведены три механизма (шифрование, цифровая подпись и обмен сообщениями для опознавания), которые могут использоваться для того, чтобы обеспечить удостоверение подлинности. Таблица также показывает, что шифрование может использоваться при трех услугах (конфиденциальности данных, целостности данных и аутентификации).

Таблица 1.2. Соотношение между услугами безопасности и механизмами обеспечения безопасности

Услуга безопасности	**Механизм обеспечения безопасности**
Конфиденциальность данных	Шифрование и управление маршрутизацией
Целостность данных	Шифрование, цифровая подпись, контрольные признаки целостности данных
Проверка полномочий	Шифрование, цифровая подпись, установление правомочности изменений
Исключение отказа от сообщений	Цифровая подпись, целостность данных и доверенность
Управление доступом	Механизм управления доступом

1.4. Методы

Механизмы, которые мы рассмотрели в предыдущих секциях, — это только теоретические рецепты. Для реализации информационной безопасности требуется небольшое число методов. Два из них наиболее распространены: один общий (криптография) и один специфический (стеганография).

Криптография

Некоторые механизмы информационной безопасности, перечисленные в предыдущей секции, могут быть реализованы с помощью криптографии. Криптография, слово с греческим происхождением, означает «тайна написанного». Однако мы используем этот термин, чтобы обозначить науку и искусство преобразования сообщений, которые делают их безопасными и придают иммунитет к атакам. Хотя в прошлом криптография заключалась только в шифровании и дешифровании сообщений с применением секретных ключей, сегодня она определяется как совокупность трех различных механизмов: шифрование симметричными ключами, шифрование асимметричными ключами и хэширование. Ниже кратко рассмотрим эти три механизма.

Шифрование симметричными ключами

Шифрование симметричными ключами иногда называют шифрованием с секретным ключом или криптографией с секретным ключом. Например, объект, назовем его Алиса, может передать сообщение другому объекту, который называется Боб, по опасному каналу, для того, чтобы ее противник, который именуется Ева, не смог понять содержание сообщения, просто подслушав его по каналу. Алиса зашифровала сообщение, используя алгоритм шифрования; Боб расшифровывает сообщение, используя алгоритм расшифровки. В шифровании симметричными ключами применяется единственный **ключ засекречивания** и для шифрования, и для расшифровки. Шифрование/дешифрование можно представить как электронный замок. При шифровании симметричным ключом Алиса помещает сообщение в блок и закрывает блок, используя совместный ключ засекречивания; Боб отпирает замок другим экземпляром того же ключа и извлекает сообщение.

Шифрование асимметричными ключами

При шифровании асимметричными ключами (иногда называемом шифрованием с открытыми ключами или криптографией с открытыми ключами) ситуация почти такая же, что и при шифровании симметричными ключами, но с небольшой разницей. Мы имеем два ключа вместо одного: из них один **открытый ключ (public key)**, другой — индивидуальный или **секретный (private key)**.

Для того чтобы передать защищенное сообщение Бобу, Алиса сначала зашифровала сообщение, используя открытый ключ Боба. Чтобы расшифровывать сообщение, Боб использует свой собственный секретный ключ.

Хэширование

При **хэшировании** из сообщения переменной длины может быть создан дайджест[1] фиксированной длины, обычно намного меньшего размера, чем исходное сообщение. Сообщение и дайджест нужно передать Бобу. Дайджест используется, чтобы обеспечить проверку целостности данных, которая обсуждалась раньше.

Стеганография

Хотя эта книга базируется на криптографии как методике реализации механизмов безопасности, но все же кратко рассмотрим другую методику, которая в прошлом использовалась для засекречивания связи. В настоящее время она — стеганография — снова восстанавливается. Это слово происходит от греческого названия и означает «закрытую запись», в отличие от криптографии, означающей «секретную запись». Криптография скрывает содержание сообщение путем шифрования. Стеганография скрывает само сообщение непосредственно, закрывая его чем-нибудь.

Исторические примеры использования

История полна фактов и мифов об использовании стеганографии. В Китае военные сообщения писались на кусках тончайшего шелка и закатывались в ма-

[1] Дайджест (digest) — короткое шифрованное сообщение, передаваемое по каналу связи вместе с незашифрованным сообщением и играющее роль цифровой подписи.

ленький шар, который глотал посыльный. В Риме и Греции сообщения вырезались на кусочках древесины, которые потом опускались в воск, чтобы закрыть запись. Также использовались невидимые чернила (такие как луковый сок или соли аммиака) для записи секретного сообщения между строками безобидного текста или в конце бумаги; секретное сообщение выступало, когда эта бумага нагревалась или обрабатывалась каким-то веществом.

Недавно были изобретены другие методы. В некоторые безобидные письма могли бы быть вписаны сообщения поверх письма, карандашом, след которого видим только тогда, когда текст помещен под яркий источник света под углом. Нулевые шифры использовались, чтобы скрыть секретное сообщение в безвредном сообщении. Например, секретное сообщение можно составить, если первая или вторая буква в каждом слове бесполезна, а только закрывает истинное сообщение. Микроточки также применялись для этой цели. Секретные сообщения были сфотографированы и уменьшены до размера точки и встраивались в простые сообщения или периодически вставлялись в конце предложения.

Современное использование

Сегодня любая форма данных, такая как текст, изображение, аудио- или видеоинформация, может быть переведена в цифровую форму, и во время преобразования в цифровую форму или обработки можно в общие данные вставить секретную двоичную информацию. Такая скрытая информация не обязательно используется для сохранения тайны. Она может также применяться как пометка, чтобы защитить авторское право, предотвратить вмешательство или внести дополнительную информацию, комментирующую текст для некоего получателя.

Скрывающие тексты. Для незаметной передачи секретных данных может быть задействован обычный текст. Есть несколько путей. Один из них — вставка двоичных символов. Например, мы можем использовать пробел между словами. Чтобы представить двоичную цифру 0, используется одиночный пробел — и два пробела, чтобы представить двоичную цифру 1. Представленное ниже короткое сообщение скрывает двоичное 8-битовое представление буквы A (01000001) в коде ASCII.

Это учебник по изучению криптографии, а не по стеганографии
□ □□□ □ □□□ □□
0 1 0 0 0 0 0 1

В сообщении, которое приведено выше, два пробела содержатся между словами «учебник» и «по» и между «по» и «стеганография». Конечно, усложненное программное обеспечение может вставить пробелы, которые различаются минимально, чтобы скрыть код от непосредственного визуального распознавания.

Другой, более эффективный метод использует словарь слов, организованных согласно их грамматическим значениям (частям речи). Мы можем, например, иметь словарь, содержащий 2 местоимения, 16 глаголов, 32 существительных. За каждым представителем этого словаря закреплен код. Предположим, что первый бит двоичных данных может быть представлен местоимением, каждое из которых имеет код (например, «Я» — это 0, а «мы» — это 1). Следующие пять би-

тов могут быть представлены существительным (подлежащим в предложении). В нашем примере можно обозначить код 10010 словом «шофер». Следующие четыре бита могут быть представлены глаголом, (в примере — словом «веду», которое представляет код 0001), и последние пять бит — другим существительным (дополнение). В нашем примере «машину» — означает код 001001. Тогда можно договориться использовать скрывающий текст, который всегда применяет предложения *местоимение — существительное — глагол — существительное.* Секретные двоичные данные могут быть разделены на куски на 16 битов. Например, секретное сообщение «Hi», которое в ASCII отображается 0 10010 0001 001001, могло быть засекречено следующим предложением:

Я шофер веду машину
0 10010 0001 001001

Это — очень тривиальный пример. Реальный подход использует более усложненный алгоритм и большее разнообразие применяемых слов для одного и того же кода.

Методы скрытия, использующие изображения. Данные могут быть скрыты другим цветным изображением. Переведенные в цифровую форму изображения состоят из пикселей[1] (элемент картинки), в котором обычно каждый пиксель использует 24 бита (три байта). Каждый байт представляет один из первичных цветов (красный, зеленый или синий). Мы можем поэтому иметь 2^8 различных оттенков каждого цвета. В методе, названном LSB (Last Significant Bit), самый младший бит каждого байта установлен на нуль. От этого изображение становится немного светлее в некоторых областях, но это обычно не замечается. Теперь мы можем скрыть двоичные данные в изображении, сохраняя или изменяя самый младший бит. Если наша двоичная цифра — 0, мы сохраняем бит; если это — 1, мы изменяем бит на 1. Этим способом мы можем скрыть символ (восемь битов ASCII) в трех пикселях. Последний бит последнего пикселя не учитывается. Например, следующие три пикселя могут представить латинскую букву M ($4D_{16}$ или, в двоичной системе, 0100 1101):

0101001**0**	1011110**0**	0101010**0**
0101111**1**	1011110**1**	0110010**1**
0111111**0**	0100101**1**	0001010**0**

Другие методы скрытия. Возможны также другие методы скрытия. Секретное сообщение, например, может быть закрыто аудио- (звук и музыка) и видеоинформацией. И аудио, и видео сегодня подвергаются сжатию. Секретные данные могут быть внесены в информацию в процессе или перед сжатием.

Теперь мы прекращаем обсуждение этих методов и рекомендуем интересующимся более специализированную литературу по стеганографии.

[1] Пиксел (pixel – picture element) – наименьший адресуемый элемент изображения или точка на экране, которая может иметь различную яркость и цвет.

1.5. Остальные части книги

Остальной текст этой книги разделен на четыре части.

Первая часть: Шифрование симметричными ключами

Первая часть рассматривает два типа шифрования: и классическое, и современное, с использованием шифрования симметричными ключами. Лекции этой части показывают, как с помощью этой методики может быть реализована первая цель безопасности.

Вторая часть: Шифрование асимметричными ключами

Лекции второй части обсуждают шифрование асимметричными ключами. Эти лекции также показывают, как первая цель безопасности может быть достигнута с помощью этой методики.

Третья часть: целостность, установление подлинности и ключевое управление

Лекции третьей части рассказывают о третьем методе шифрования — хэшировании — и показывают, как это может быть объединено с материалами, о которых шла речь в частях I и II при осуществлении второй цели безопасности.

Четвертая часть: Сетевая безопасность

Лекции четвертой части показывают, как методы, рассмотренные в первых трех частях книги, могут быть объединены для создания сетевой безопасности, используя модель Интернет.

1.6. Рекомендованное чтение

Для более детального ознакомления с предметами, о которых шла речь в этой лекции, начинающим можно рекомендовать нижеследующие книги и сайты. Символы, заключенные в скобки, рассматриваются как ссылки к списку литературы в конце книги.

Книги

Несколько книг рассматривают цели безопасности, нападения и механизмы. Мы рекомендуем [Bis05]и [Sta06]. |

Сайты

Больше информации о темах, обсужденных в этой лекции, дают следующие сайты:

http://www.faqs.org/rfcs/rfc2828.html

http://www. fag.grm.hia.no/IKT7000/litteratur/paper/x800.pdf

1.7. Итоги

- Для информационной безопасности были определены три главные цели: конфиденциальность, целостность и готовность.
- Конфиденциальности информации угрожают два типа атак: вмешательство — и наблюдение за трафиком и его анализ. Четыре типа атак могут угрожать целостности информации: модификация, имитация источника, повторная передача информации и отказ от сообщения. Атака «прекращение обслуживания запроса» угрожает готовности информации.
- Организации, которые занимаются передачей данных и созданием сетей, такие как ITU-T или Internet Forum, определили несколько служб безопасности, предназначенных для целей информационной безопасности и защиты от атак. В этой лекции рассматривались пять общих служб безопасности: конфиденциальность данных, целостность данных, установление подлинности, исключение отказа от сообщений и управление доступом.
- ITU-T также рекомендует некоторые механизмы обеспечения безопасности. В лекции рассмотрены восемь из этих механизмов: шифрование, целостность данных, цифровая подпись, установление правомочности изменений, заполнение трафика, управление маршрутизацией, доверенность и управление доступом.
- Есть два метода — криптография и стеганография, которые могут реализовать некоторые или все механизмы. Криптография или «тайное письмо» включает скремблирование сообщения или создание дайджеста сообщения. Стеганография, или «закрытая запись», означает, что сообщение скрывается и закрывается другой информацией.

1.8. Набор для практики

Обзорные вопросы

1. Определите три цели безопасности.
2. Укажите различие между пассивными и активными атаками на секретную информацию. Назовите некоторые пассивные атаки. Назовите некоторые активные атаки.
3. Перечислите и определите пять служб безопасности, рассмотренные в этой лекции.
4. Определите восемь механизмов безопасности, рассмотренные в этой лекции.
5. Укажите различие между шифрованием и стеганографией.

Упражнения

1. Какая служба(ы) безопасности гарантируется при использовании каждого из следующих методов пересылки по почте в почтовом отделении?
   a. Обычная почта
   b. Обычная почта с подтверждением доставки

c. Обычная почта с доставкой и подписью получателя
d. Заказное письмо
e. Почта с объявленной ценностью
f. Зарегистрированная корреспонденция

2. Определить тип атаки на секретную информацию в каждом из следующих случаев:
   a. Студент проникает в офис профессора, чтобы получить копию теста, который будет проведен на следующий день.
   b. Студент дает чек на получение денег на 10$, чтобы купить уже подержанную книгу. Потом он узнает, что по чеку было получено 100$.
   c. Студент передает сотни запросов в день, используя фальшивый обратный адрес телефона другого студента.
3. Какие механизм(ы) безопасности реализованы в каждом из следующих случаев?
   a. Университет требует идентификатор студента и пароль, чтобы позволить студенту получить доступ в школьный сервер.
   b. Университетский сервер разъединяет студента, если он получил доступ в систему более чем два часа назад.
   c. Профессор отказывается передать оценки электронной почтой студентам, если они не соответствуют студенческой идентификации, заранее назначенной профессором.
   d. Банк требует подписи клиента для изъятия клиентом денег.
4. Какая методика (криптография или стеганография) используется в каждом из следующих случаев для защиты конфиденциальности?
   a. Студент пишет ответы на билеты на маленьком листочке бумаги, бумага свертывается и вставляется в шариковую ручку, а ручка передается другому студенту.
   b. Чтобы передать сообщение, шпион заменяет каждый символ в сообщении символом, который был согласован заранее как замена другого символа.
   c. Компания использует специальные чернила на своих чеках, чтобы предотвратить подделки.
   d. Аспирантка использует водяные знаки, чтобы защитить свою работу, которая вывешена на ее сайте.
5. Какой механизм(ы) безопасности реализуется, когда человек подписывает форму при заполнении заявления на кредитную карту?

Часть 1. Шифрование симметричными ключами

В **лекции 1** (Введение) мы видели, что криптография реализуется с помощью трех методов: шифры с симметричным ключом, шифры с асимметричными ключами, и хэширование. Первая часть посвящена шифрам с симметричным ключом. Лекции 2 и 4 рассматривают математические основы, необходимые для того, чтобы понять остальное содержание этой части. Лекция 3 исследует традиционные шифры, использовавшиеся в прошлом. Лекции 5, 6, и 7 показывают блочные шифры, которые применяются сегодня. Лекция 8 показывает, как современные блочные шифры и шифры потоков могут применяться для шифрования длинных сообщений.

Лекция 2: Математика криптографии: Часть I

Лекция 2 рассматривает некоторые математические понятия, необходимые для понимания следующих нескольких лекций, как то: арифметику целых чисел и сравнения по модулю m, матрицы и конгруэнтные отношения.

Лекция 3: Традиционные шифры с симметричным ключом

Лекция 3 вводит традиционные шифры с симметричным ключом. Хотя эти шифры не используются сегодня, они – основы современных шифров с симметричным ключом. Эта лекция подчеркивает две категории традиционных шифров: шифры замены и шифры перемещения. Она также вводит понятия шифров потока и блочных шифров.

Лекция 4: Математика криптографии: Часть II

Лекция 4 – это обзор математических основ, которые необходимы, чтобы понять содержание последующих лекций. Она рассматривает некоторые алгебраические структуры, такие, как группы, кольца и конечные поля, применяемые в модемных блочных шифрах.

Лекция 5: Введение в современные шифры с симметричным ключом

Лекция 5 — введение в модемные шифры с симметричным ключом. Понимание отдельных элементов, используемых в модемных шифрах с симметричным ключом, прокладывает путь к лучшему пониманию и анализу современных шифров. Эта лекция знакомит с компонентами блочных шифров, таких, как P-блоки и S-блоки. Также здесь вводится разделение между двумя классами шифров: шифры Файстеля и не-Файстелевские шифры.

Лекция 6: Стандарт кодирования данных (DES)

Лекция 6 работает с элементами, определенными в лекции 5, чтобы обсудить и проанализировать один из общих шифров с симметричным ключом, применяемых сегодня, — Стандарт Кодирования Данных **(DES — DATA ENCRIPTION STANDARD)**. Особое внимание уделяется тому, как DES использует 16 циклов шифров Файстеля.

Лекция 7: Усовершенствованный стандарт шифрования (AES)

Лекция 7 показывает, как алгебраические структуры, рассмотренные в лекции 4, и элементы, рассмотренные в лекции 5, могут создать очень мощный шифр — Усовершенствованный Стандарт Шифрования **(AES — ADVANCED ENCRYPTION STANDARD)**. Особое внимание уделяется тому, как алгебраические структуры, приведенные в лекции 4, работают для достижения целей безопасности.

Лекция 8: Шифрование, использующее современные шифры с симметричным ключом

Лекция 8 показывает, как современные блочные шифры и шифры потока могут фактически применяться, чтобы зашифровать длинные сообщения. Она объясняет, как современные блочные шифры используют пять режимов работы. Она также вводит два шифра потока для обработки данных в реальном масштабе времени.

Лекция 2. Математика криптографии. Часть I. Модульная арифметика, сравнения и матрицы

Цели и содержание

Эта лекция необходима, чтобы подготовить читателя к дальнейшему разговору о криптографии. Лекция имеет несколько целей.

- Рассмотреть арифметику целых чисел, которая базируется на теории делимости и нахождении наибольшего общего делителя, используя алгоритм Евклида.
- Показать, как расширенный алгоритм Евклида может применяться для решения линейных диофантовых уравнений, для решения конгруэнтных линейных уравнений и нахождения мультипликативной инверсии.
- Обратить внимание на важность модульной арифметики (арифметики над вычетами по модулю n) и операций в ней, потому что они широко используются в криптографии.
- Обратить внимание и рассмотреть матрицы и операции с матрицами вычетов, которые широко применяются в криптографии.
- Решить набор уравнений сравнения, используя матрицы вычетов.

Криптография базируется на некоторых специфических областях математики, включая теорию чисел, линейную алгебру и алгебраические структуры. В этой лекции мы обсуждаем только те темы в вышеупомянутых областях, которые необходимы для понимания содержания следующих нескольких лекций. Читатели, которые знакомы с этими темами, могут пропустить эту лекцию полностью или частично. Подобные теоретические лекции встречаются всюду в этой книге, когда это необходимо. Доказательства теорем и алгоритмов пропущены, но заинтересованный читатель может найти их в приложении Q.

2.1. Арифметика целых чисел

В арифметике целых чисел мы используем множество целых чисел и несколько операций. Вы знакомы с этим множеством и соответствующими операциями, но они рассмотрены здесь, чтобы объяснить потом основы действий со сравнениями по модулю m.

Множество целых чисел

Множество целых чисел, обозначенных **Z**, содержит все числа (без дробей) от минус бесконечности до плюс бесконечности (рис. 2.1).

$$\mathbf{Z = \{.....,-2,-1,0,1,2,.....\}}$$

Рис. 2.1. Множество целых чисел

Бинарные операции

В криптографии нас интересуют три бинарных операции в приложении к множеству целых чисел. **Бинарные операции** имеют два входа и один выход. Для целых чисел определены три общих бинарных операции — *сложение*, *вычитание* и *умножение*. Каждая из этих операций имеет два входа (*a* и *b*) и выход (*c*), как это показано на рис. 2.2. Два входа принимают числа из множества целых чисел; выход выводит результат операции — число из множества целых чисел.

Обращаем внимание, что *деление* не относится к этой категории операций, потому что мы скоро убедимся, что этой операции нужны два выхода вместо одного.

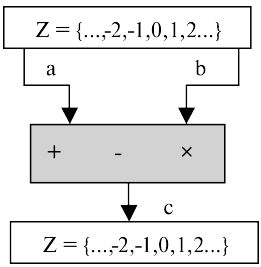


Рис. 2.2. Три бинарных операции для множества целых чисел

Пример 2.1

Следующие примеры показывают результаты трех двоичных операций на множестве двух целых чисел. Поскольку каждый вход может быть или положителен или отрицателен, мы имеем четыре случая для каждой операции.

Сложение	5 + 9=14	(-5) + 9=4	5 + (-9)=-4	(-5) + (-9)=-14
Вычитание	5 - 9=4	(-5) - 9=-14	5 - (-9)=14	(-5) - (-9)=+4
Умножение	5 × 9=45	(-5) × 9=-45	5 × (-9)=-45	(-5) × (-9)=45

Деление целых чисел

В арифметике целых чисел, если мы *a* делим на *n,* мы можем получить *q* и *r.* Отношения между этими четырьмя целыми числами можно показать как

$$\boldsymbol{a = q \times n + r}$$

В этом равенстве *a* называется *делимое*; *q* — *частное*; *n* — *делитель* и *r* — *остаток*. Обратите внимание, что это — не операция, поскольку результат деления *a* на *n* — это два целых числа, *q* и *r.* Мы будем называть это *уравнением деления*.

Пример 2.2

Предположим, что $a = 255$, а $n = 11$. Мы можем найти $q = 23$ и $r = 2$, используя алгоритм деления, мы знаем из элементарной арифметики — оно определяется, как показано на рис. 2.3.

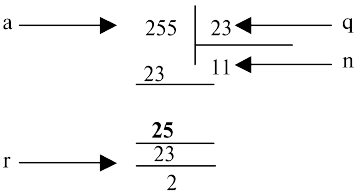


Рис. 2.3. Пример 2.2, нахождение частного и остатка

Большинство компьютерных языков может найти частное и остаток, используя заданные языком операторы. Например, на языке C оператор «/» может найти частное, а оператор «%» — остаток.

Два ограничения

Когда мы используем вышеупомянутое уравнение деления в криптографии, мы налагаем два ограничения. Первое требование: чтобы делитель был положительным целым числом $(n > 0)$. Второе требование: чтобы остаток был неотрицательным целым числом $(r > 0)$. Рисунок 2.4 показывает эти требования с двумя указанными ограничениями.

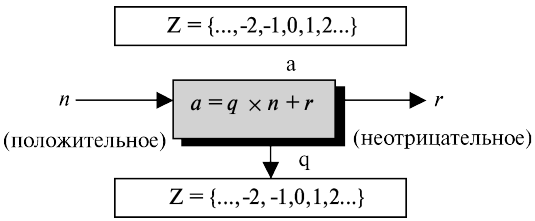


Рис. 2.4. Алгоритм деления целых чисел

Пример 2.3

Предположим, мы используем компьютер или калькулятор, а r и q отрицательны, при отрицательном a. Как можно сделать, чтобы выполнялось ограничение, что число r должно быть положительным? Решение простое: мы уменьшаем значение q на 1 и добавляем значение n к r, чтобы r стало положительным.

$$-255 = (-\mathbf{23} \times 11) + (-2) \quad \longleftrightarrow \quad -255 = (-\mathbf{24} \times 11) + \mathbf{9}$$

Мы уменьшили (–23), получили (–24) и добавили 11 к (–2), чтобы получить + 9. Полученное равенство эквивалентно исходному.

Граф уравнения деления

Мы можем изобразить рассмотренные выше уравнения с двумя ограничениями по *n* и *r* на рис. 2.5 с помощью двух графов. Первый показывает случай, когда число *a* положительно; второй — когда отрицательно.

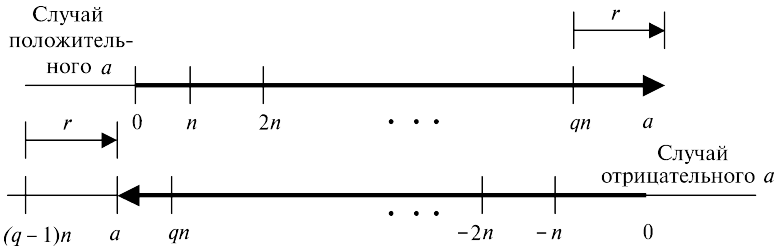


Рис. 2.5. Граф алгоритма деления

Граф начинается с нуля и показывает, как мы можем достигнуть точки, представляющей целое число *a* на линии. В случае положительного *a* мы должны перемещаться на величину $q \times n$ направо и затем добавить дополнительную *величину r* в том же направлении. В случае отрицательного *a* мы должны двигаться на величину $(q - 1) \times n$ налево (число *q* в этом случае отрицательно) и затем дополнять число r в противоположном для указанного выше движения направлении. В обоих случаях значение *r* положительно.

Теория делимости

Теперь кратко обсудим **теорию делимости** — тема, с которой мы часто сталкиваемся в криптографии. Если *a* не равно нулю, а $r = 0$, в равенстве деления мы имеем

$$a = q \times n$$

Мы тогда говорим, что *a* делится на *n* (или n — делитель *a*). Мы можем также сказать, что *a* делится без остатка на *n*. Когда мы не интересуемся значением *q*, мы можем записать вышеупомянутые отношения как $a|n$. Если остаток не является нулевым, то *n* не делится, и мы можем записать отношения как $a \nmid n$.

Пример 2.4

a. Целое число 4 делит целое число 32, потому что $32 = 8 \times 4$. Это можно отобразить как 4|32. Число 8 не делит число 42, потому что $42 = 5 \times 8 + 2$. В этом уравнении число 2 — остаток. Это можно отобразить как $8 \nmid 42$.

Пример 2.5

а. Отображение делимости 13|78, 7|98, –6|24, 4|44, и 11 | (–33).

б. Отображение неделимости 13†27, 7†50, – 6†T23, 4†41, и 11†(–32).

Свойства

Следующие несколько свойств теории делимости. Доказательства заинтересованный читатель может проверить в приложении Q.

Свойство 1: если $a \mid 1$, то a = ±1.

Свойство 2: если $a \mid b$ и $b \mid a$, то $a = \pm b$

Свойство 3: если $a \mid b$ и $b \mid c$, то $a \mid c$

Свойство 4: если $a \mid b$ и $a \mid c$, то $a \mid (m \times b + n \times c)$, где m и n — произвольные целые числа.

Пример 2.6

а. Если 3|15 и 15|45 то, согласно третьему свойству, 3|45.

б. Если 3|15 и 3|9, то, согласно четвертому свойству, 3|(15 x 2 + 9 x 4), что означает 3|66.

Все делители

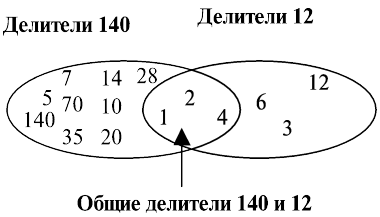
Положительное целое число может иметь больше чем один делитель. Например, целое число 32 имеет шесть делителей: 1, 2, 4, 8, 16 и 32. Мы можем упомянуть два интересных свойства делителей положительных целых чисел.

Свойство 1: целое число 1 имеет только один делитель — само себя.

Свойство 2: любое положительное целое число имеет по крайней мере два делителя — 1 и само себя (но может иметь больше).

Наибольший общий делитель

Одно целое число, часто необходимое в криптографии, — **наибольший общий делитель** двух положительных целых чисел. Два положительных целых числа могут иметь много общих делителей, но только один наибольший общий делитель.



Например, общие делители чисел 12 и 140 есть 1, 2 и 4. Однако наибольший общий делитель — 4 (см. рис. 2.6).

Рис. 2.6. Общие делители двух целых чисел

Наибольший общий делитель двух положительных целых чисел — наибольшее целое число, которое делит оба целых числа.

Алгоритм Евклида

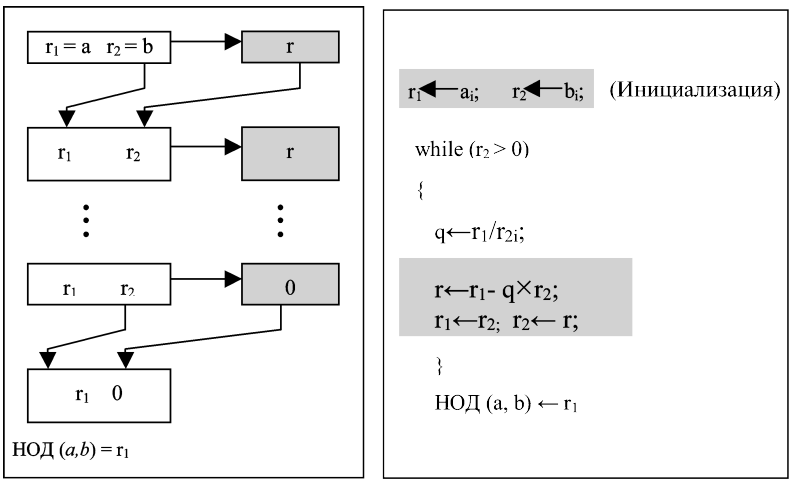
Нахождение наибольшего общего делителя (НОД) двух положительных целых чисел путем составления списка всех общих делителей непригодно для достаточно больших чисел.

К счастью, больше чем 2000 лет назад математик по имени Евклид разработал алгоритм, который может найти наибольший общий делитель двух положительных целых чисел. **Алгоритм Евклида** основан на следующих двух фактах (доказательство см. в приложении Q):

Факт 1: НОД (a, 0) = a
Факт 2: НОД *(a, b)* = НОД *(b,* r), где r — остаток от деления a на b

Первый факт говорит, что если второе целое число — 0, наибольший общий делитель равен первому числу. Второй факт позволяет нам изменять значение a на b, пока b не станет 0. Например, вычисляя НОД (36, 10), мы можем использовать второй факт несколько раз и один раз первый факт, как показано ниже.

НОД(36) = НОД(10,6) = НОД(6,4) = НОД(4,2) = НОД(2,0)



а) Процесс б) Алгоритм

Другими словами, НОД (36, 10) = 2, НОД (10, 6) = 2, и так далее. Это означает, что вместо вычисления НОД (36, 10) мы можем найти НОД (2, 0). Рисунок 2.7 показывает, как мы используем вышеупомянутые два факта, чтобы вычислить НОД (*a, b*).

Рис. 2.7. Алгоритм Евклида

Для определения НОД мы используем две переменные, r_1 и r_2, чтобы запоминать изменяющиеся значения в течение всего процесса. Они имеют начальное значение *a* и *b*. На каждом шаге мы вычисляем остаток от деления r_1 на r_2 и храним результат в виде переменной *r*. Потом заменяем r_1, на r_2 и r_2 на *r* и продолжа-

q	r_1	r_2	r
1	2740	1760	980
1	1760	980	780
1	980	780	200
3	780	200	180
1	200	180	20
9	180	20	0
	20	0	

ем шаги, пока *r* не станет равным 0. В этот момент процесс останавливается и НОД (*a, b*) равен r_1.

Пример 2.7

Нужно найти наибольший общий делитель 2740 и 1760.

Решение

q	r_1	r_2	*r*
0	25	60	25
2	60	25	10
2	25	10	5
2	10	5	0
	5	0	

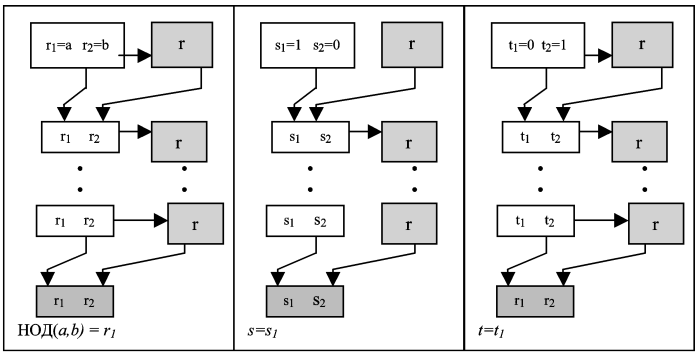
Применим вышеупомянутую процедуру, используя таблицу. Мы присваиваем начальное значение r_1 2740 и r_2 значение 1760. В таблице также показаны значения *q* на каждом шаге. Мы имеем НОД (2740, 1760) = 20.

Пример 2.8

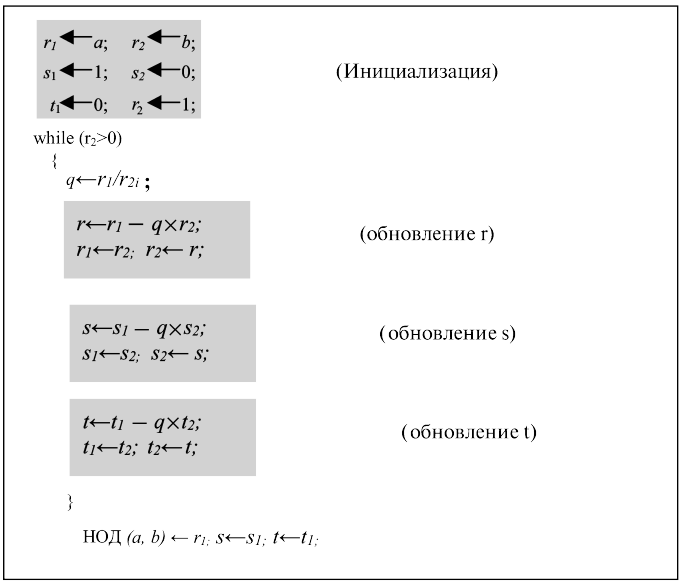
Найти наибольший общий делитель 25 и 60.

Решение

Мы выбрали этот конкретный пример, чтобы показать: для алгоритма Евклида безразлично, что первое число может быть меньше, чем второе. Все равно мы получаем правильный ответ НОД (25, 65) = 5.



а) Процесс



б) Алгоритм

Расширенный алгоритм Евклида

Даны два целых числа *a* и *b*. Нам зачастую надо найти другие два целых числа, *s* и r, такие, которые

$$s \times a + t \times b = \text{НОД}(a,b)$$

Расширенный алгоритм Евклида может вычислить НОД (*a, b*) и в то же самое время вычислить значения *s* и *t*. Алгоритм и процесс такого вычисления показан на рис. 2.8.

Здесь расширенный алгоритм Евклида использует те же самые шаги, что и простой алгоритм Евклида. Однако в каждом шаге мы применяем три группы вычислений вместо одной. Алгоритм использует три набора переменных: *r*, *s* и *t*.

Рис. 2.8. Алгоритм Евклида

На каждом шаге переменные r_1, r_2 и r используются так же, как в алгоритме Евклида. Переменным r_1 и r_2 присваиваются начальные значения *a* и *b* соответственно. Переменным s_1 и s_2 присваиваются начальные значения 1 и 0 соответственно. Переменным t_1 и t_2 присваиваются начальные значения 0 и 1 соответственно.

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
5	161	28	21	1	0	1	0	1	−5
1	28	21	7	0	1	−1	1	−5	6
3	21	7	0	1	−1	4	−5	6	−23
	7	0		**−1**	4		**6**	−23	

Вычисления r, *s* и t одинаковы, но с одним отличием. Хотя *r* — остаток от деления r_1 на r_2, такого соответствия в других двух группах вычислений нет. Есть только одно частное, *q*, которое вычисляется как r_1/r_2 и используется для других двух вычислений.

Пример 2.9

Дано $a = 161$ и $b = 28$, надо найти НОД (*a, b*) и значения *s* и *t*.

Решение

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
	17	0		1	0		0	1	

$$r = r_1 - q \times r_2 \qquad s = s_2 - qs_2 \qquad t = t_1 - q \times t_2$$

Для отображения алгоритма мы используем следующую таблицу:

Мы получаем НОД (161, 28) = 7, $s = -1$ и $t = 6$. Ответы могут быть проверены, как это показано ниже.

$$(-1) \times 161 + 6 \times 28 = 7$$

Пример 2.10

Дано $a = 17$ и $b = 0$, найти НОД (*a, b*) и значения *s* и *t*.

q	r_1	r_2	r	s_1	s_2	S	t_1	t_2	t
0	0	45	0	1	0	1	0	1	0
	45	0		0	1		1	0	

Решение

Для отображения алгоритма мы используем таблицу:

Обратите внимание, что нам не надо вычислять *q, r* и *s.* Первое значение r_2 соответствует условию завершения алгоритма. Мы получаем НОД (17, 0) = 17, $s = 1$ и $t = 0$. Это показывает, почему мы должны придавать начальные значения s_1 — 1 и t_1 — 0. Ответы могут быть проверены так, как это показано ниже:

$$(1 \times 17) + (0 \times 0) = 17$$

Пример 2.11

Даны $a = 0$ и $b = 45$, найти НОД (a, b) и значения s и t.

Решение

Для отображения алгоритма мы используем следующую таблицу:

Мы получаем НОД (0,45) = 45, $s = 0$ и $t = 1$. Отсюда ясно, что мы должны инициализировать s_2 равным 0, а t_2 – равным 1. Ответ может быть проверен, как это показано ниже:

$$(0 \times 0) + (1 \times 45) = 45$$

Линейные диофантовы уравнения

Хотя очень важное приложение расширенного алгоритма Евклида будет рассмотрено далее, здесь мы остановимся на другом приложении — «нахождение решения **линейных диофантовых уравнений** двух переменных», а именно, уравнения $ax + by = c$. Мы должны найти значения целых чисел для x и y, которые удовлетворяют этому уравнению. Этот тип уравнения либо не имеет решений, либо имеет бесконечное число решений. Пусть d = НОД (a, b). Если $d \dagger c$, то уравнение не имеет решения. Если $d|c$, то мы имеем бесконечное число решений. Одно из них называется частным, остальные — общими.

Линейное диофантово уравнение — это уравнение двух переменных:
$$ax + by = c.$$

Частное решение

Если $d|c$, то можно найти частное решение вышеупомянутого уравнения, используя следующие шаги.

1. Преобразуем уравнение к $a_1 x + b_1 y = c_1$, разделив обе части уравнения на d. Это возможно, потому, что d делит $a, b,$ и c в соответствии с предположением.
2. Найти s и t в равенстве $a_1 s + b_1 t = 1$, используя расширенный алгоритм Евклида.
3. Частное решение может быть найдено:

Частное решение: $x_0 = (c/d)\, s$ и $y_0 = (c/d)\, t$

Общие решения

После нахождения частного решения общие решения могут быть найдены:

Общие решения: $x = x_0 + k\,(b/d)$ и $y = y_0 - k\,(a/d)$,
где k — **целое число**

Пример 2.12

Найти частные и общие решения уравнения $21x + 14y = 35$.

Решение

Мы имеем d = НОД (21, 14) = 7. При 7|35 уравнение имеет бесконечное число решений. Мы можем разделить обе стороны уравнения на 7 и получим уравнение $3x + 2y = 5$. Используя расширенный алгоритм Евклида, мы находим s и t, такие, что $3s + 2t = 1$. Мы имеем S = 1 и $t = -1$. Решения будут следующие:

Частное решение : $x_0 = 5 \times 1 = 5$ и $y_0 = 5 \times (-1) = -5$ тогда 35/7 =5
Общие: $x = 5 + k \times 2$ $y = -5 - k \times 3$ где k — целое

Поэтому решения будут следующие (5, –5), (7, –8), (9, –11)...

Мы можем легко проверить, что каждое из этих решений удовлетворяет первоначальному уравнению.

Пример 2.13

Рассмотрим очень интересное приложение решения диофантовых уравнений в реальной жизни. Мы хотим найти различные комбинации объектов, имеющих различные значения. Например, мы хотим обменять денежный чек 100$ на некоторое число банкнот 20$ и несколько банкнот по 5$. Имеется много вариантов, которые мы можем найти, решая соответствующее диофантово уравнение $20x + 5y = 100$. Обозначим d = НОД (20, 5) = 5 и 5 | 100. Уравнение имеет бесконечное число решений, но в этом случае приемлемы только несколько из них (только те ответы, в которых и x и y являются неотрицательными целыми числами). Мы делим обе части уравнения на 5, чтобы получить $4x + y = 20$, и решаем уравнение $4s + t = 1$. Мы можем найти $s = 0$ и $t = 1$, используя расширенный алгоритм Эвклида. Частное решение: $x_2 = 0 \times 20 = 0$ и $y_0 = 1 \times 20 = 20$. Общие решения с неотрицательными x и y — (0, 20), (1, 16), (2, 12), (3, 8), (4, 4), (5, 0). Остальная часть решений неприемлема, потому что y становится отрицательным. Кассир в банке должен спросить, какую из вышеупомянутых комбинаций мы хотим получить. Первое число в скобках обозначает число банкнот по 20$; второе число обозначает число банкнот по 5$.

2.2. Модульная арифметика

Уравнение деления ($a = q \times n + r$), рассмотренное в предыдущей секции, имеет два входа (a и n) и два выхода (q и r). В модульной арифметике мы интересуемся только одним из выходов — остатком r. Мы не заботимся о частном q. Другими словами, когда мы делим a на n, мы интересуемся только тем, что *значение остатка равно r.* Это подразумевает, что мы можем представить изображение вышеупомянутого уравнения как бинарный оператор с двумя входами a и n и одним выходом r.

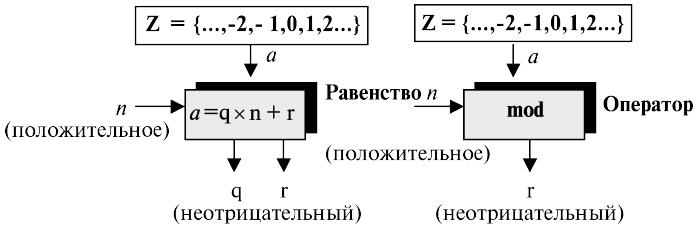


Рис. 2.9. Соотношение уравнения деления и оператора по модулю

Операции по модулю

Вышеупомянутый бинарный оператор назван **оператором по модулю** и обозначается как *mod.* Второй вход *(n)* назван **модулем.** Вывод *r* назван **вычетом.** Рисунок 2.9 показывает отношение деления по сравнению с оператором по модулю.

Как показано на рис. 2.9, оператор по модулю **(mod)** выбирает целое число (*a*) из множества Z и положительный модуль (*n*). Оператор определяет неотрицательный остаток (r).

Мы можем сказать, что

$$a \bmod n = r$$

Пример 2.14

Найти результат следующих операций:

a. 27 mod 5
b. 36 mod 12
c. −18 mod 14
d. −7 mod 10

Решение

Мы ищем вычет *r.* Мы можем разделить *a* на *n* и найти *q* и *r.* Далее можно игнорировать *q* и сохранить *r.*

а. Разделим 27 на 5 — результат: $r = 2$. Это означает, что 27 mod 5 = 2.

б. Разделим 36 на 12 — результат: $r = 0$. Это означает, что 36 mod 12 = 0.

в. Разделим (−18) на 14 — результат: $r = -4$. Однако мы должны прибавить модуль (14), чтобы сделать остаток неотрицательным. Мы имеем r = −4 + + 14 = 10. Это означает, что −18 mod 14 = 10.

г. Разделим (−7) на 10 — результат: r = −7. После добавления модуля −7 мы имеем $r = 3$. Это означает, что −7 mod 10 = 3.

$$\mathbf{Z_n = \{0,1,2,3,\ldots..,(n-1)\}}$$

$$\mathbf{Z_2 = \{0,1\}} \qquad \mathbf{Z_6 = \{0,1,2,3,4,5\}} \qquad \mathbf{Z_{11} = \{0,1,2,3,4,5,6,7,8,9,10\}}$$

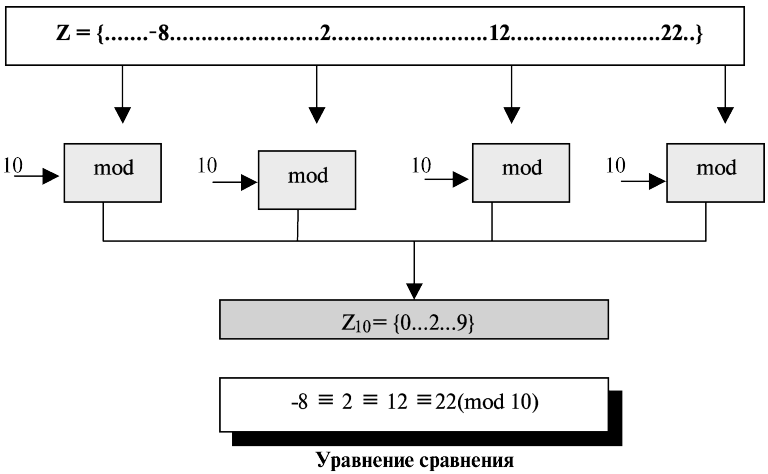
Система вычетов: Z_n

Результат операции по модулю *n* — всегда целое число между 0 и *n* - 1. Другими словами, результат *a* mod *n* — всегда неотрицательное целое число, меньшее, чем n. Мы можем сказать, что операция по модулю создает набор, который в модульной арифметике можно понимать как **систему наименьших вычетов по модулю *n*,** или Z_n. Однако мы должны помнить, что хотя существует только одно множество целых чисел (Z), мы имеем бесконечное число множеств вычетов (Z_n), но лишь одно для каждого значения *n*. Рисунок 2.10 показывает множество Z_n и три множества Z_2, Z_6 и Z_{11}.

Рис. 2.10. Некоторые наборы Z_n

Сравнения

В криптографии мы часто используем понятие **сравнения**[1] вместо равенства. Отображение Z в Z_n не отображаются «один в один». Бесконечные элементы множества Z могут быть отображены одним элементом Z_n. Например, результат 2 mod 10 = 2, 12 mod 10 = 2, 22 mod 10 = 2, и так далее. В модульной арифметике целые числа, подобные 2, 12, и 22, называются сравнимыми по модулю 10. Для того чтобы указать, что два целых числа сравнимы, используем **оператор сравнения** (≡). Мы добавляем mod n к правой стороне сравнения, чтобы определить значение модуля и сделать равенство правильным. Например, пишем:



$2 \equiv 12 \pmod{10}$ $13 \equiv 23 \pmod{10}$ $34 \equiv 24 \pmod{10}$ $-8 \equiv 12 \pmod{10}$
$3 \equiv 8 \pmod{5}$ $8 \equiv 13 \pmod{5}$ $23 \equiv 33 \pmod{5}$ $-8 \equiv 2 \pmod{5}$

[1] Сравнение – термин, принятый в математической литературе России; в оригинале принят термин конгруэнтность (congruence)*(прим. переводчика)*.

Рисунок 2.11 показывает принцип сравнения. Мы должны объяснить несколько положений.

а. Оператор сравнения напоминает оператор равенства, но между ними есть различия. Первое: оператор *равенства* отображает элемент Z самого на себя; оператор *сравнения* отображает элемент Z на элемент Z_n. Второе: оператор равенства показывает, что наборы слева и справа соответствуют друг другу «один в один», оператор сравнения — «многие — одному».

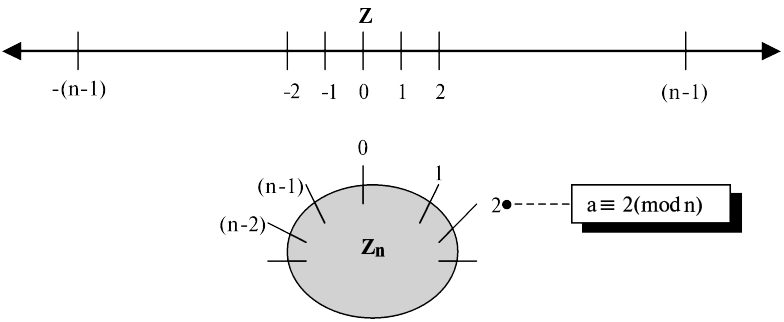
Рис. 2.11. Принцип сравнения

б. Обозначение (mod *n*), которое мы вставляем с правой стороны оператора сравнения, обозначает признак множества (Z_n). Мы должны добавить это обозначение, чтобы показать, какой модуль используется в отображении. Символ, используемый здесь, не имеет того же самого значения, как бинарный оператор в уравнении деления. Другими словами, символ mod в выражении 12 mod 10 — оператор; а сочетание (mod 10) в сравнении $2 \equiv 12$ (mod10) означает, что набор — $\mathbf{Z_{10}}$.

Класс вычетов

Класс вычетов $[a]$, или $[a]_n$, — множество целых чисел, сравнимых по модулю *n*. Другими словами, это набор всех целых чисел, таких, что $x = a \pmod{n}$. Например, если $n = 5$, мы имеем множество из пяти элементов [0], [1], [2], [3] и [4], таких, как это показано ниже:

[0] = {....., –15, 10, –5, 0, 5, 10, 15, ...}
[1] = {...., –14, –9, –4, 1, 6 , 11, 16,.....}
[2] = {..., –13, –8, –3, 2, 7, 12, 17,...}
[3] = {...., –12, –7, –2, 3, 8, 13, 18,...}
[4] = {....., 11, –6, –1, 4, 9, 14, 19,...}



Целые числа в наборе [0] все дают остаток 0 при делении на 5 (сравнимы по модулю 5). Целые числа в наборе [1] все дают остаток 1 при делении на 5 (сравнимы по модулю 5), и так далее. В каждом наборе есть один элемент, называемый наименьшим (неотрицательным) вычетом. В наборе [0] это элемент 0; в наборе [1] — 1, и так

далее. Набор, который показывает все наименьшие вычеты: $Z_5 = \{0, 1, 2, 3, 4\}$. Другими словами, набор Z_n — набор всех **наименьших вычетов по модулю n.**

Круговая система обозначений

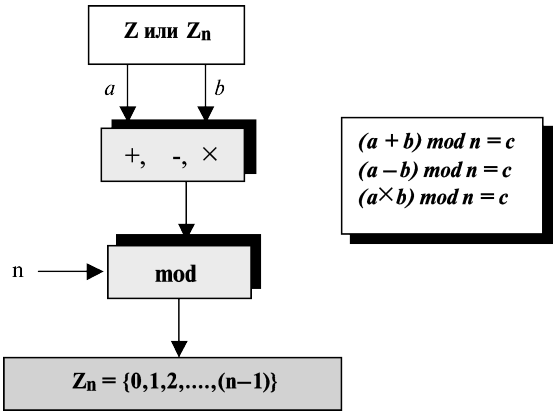
Понятие «сравнение» может быть лучше раскрыто при использовании круга в качестве модели. Так же, как мы применяем линию, чтобы показать распределение целых чисел в **Z**, мы можем использовать круг, чтобы показать распределение целых чисел в Z_n.

Рис. 2.12. Сравнение использования диаграмм для Z и Z_n

Рисунок 2.12 позволяет сравнить два этих подхода. Целые числа от 0 до $n-1$ расположены равномерно вокруг круга. Все целые числа, *сравнимые по модулю n*, занимают одни и те же точки в круге. Положительные и отрицательные целые числа от Z отображаются в круге одним и тем же способом, соблюдая симметрию между ними.

Пример 2.15

Мы пользуемся сравнением по модулю в нашей ежедневной жизни; например, мы применяем часы, чтобы измерить время. Наша система часов использует



арифметику по модулю 12. Однако вместо 0 мы берем отсечку 12, так что наша система часов начинается с 0 (или 12) и идет до 11. Поскольку наши сутки длятся 24 часа, мы считаем по кругу два раза и обозначаем первое вращение как утро до полудня, а второе — как вечер после полудня.

Операции в Z_n

Три бинарных операции (*сложение, вычитание* и *умножение*), которые мы обсуждали для Z, могут также быть определены для набора Z_n. Результат, может быть отображен в Z_n с использованием операции по модулю, как это показано на рис. 2.13.

Рис. 2.13. Бинарные операции в Z_n

Фактически применяются два набора операторов: первый набор — один из бинарных операторов (+, –, ×); второй — операторы по модулю. Мы должны использовать круглые скобки, чтобы подчеркнуть порядок работ. Как показано на рис. 2.13, входы (*a* и *b*) могут быть членами *Z* или *Zn*.

Пример 2.16

Выполните следующие операторы (поступающие от Z_n):

а. Сложение 7 и 14 в Z_{15}

б. Вычитание 11 из 7 в Z_{13}

в. Умножение 11 на 7 в Z_{20}

Решение

Ниже показаны два шага для каждой операции:

$(14 + 7) \bmod 15 \rightarrow (21) \bmod 15 = 6$

$(7 - 11) \bmod 13 \rightarrow (-4) \bmod 13 = 9$

$(7 \times 11) \bmod 20 \rightarrow (77) \bmod 20 = 17$

Пример 2.17

Выполните следующие операции (поступающие от Z_n).

a. Сложение 17 и 27 в Z_{14}

b. Вычитание 43 из 12 в Z_{13}

c. Умножение 123 на –10 в Z_{19}

Решение

Ниже показаны два шага для каждой операции:

$(17 + 27) \bmod 14 \rightarrow (44) \bmod 14 = 2$

$(12 - 43) \bmod 13 \rightarrow (-31) \bmod 13 = 8$

$(123) \times (-10)) \bmod 19 \rightarrow (-1230) \bmod 19 = 5$

Свойства

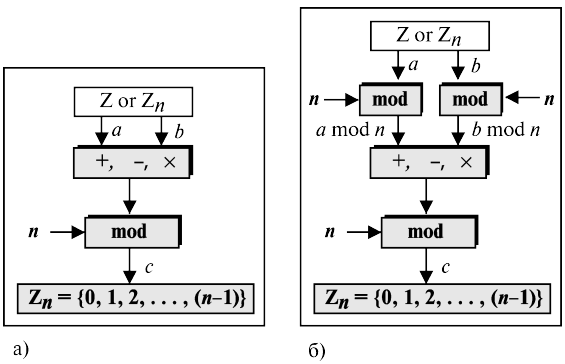


Рис. 2.14. Свойства оператора mod

Мы уже упоминали, что два входа для трех бинарных операторов в сравнении по модулю могут использовать данные из Z или Z_n. Следующие свойства позволяют нам сначала отображать два входа к Z_n (если они прибывают от Z) перед выполнением этих трех бинарных операторов

(+,–,×). Заинтересованные читатели могут найти доказательства для этих свойств в приложении Q.

Первое свойство: $(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$
Второе свойство: $(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$
Третье свойство: $(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$

Рисунок 2.14 показывает процесс до и после применения указанных выше свойств. Хотя по рисунку видно, что процесс с применением этих свойств более длительный, мы должны помнить, что в криптографии мы имеем дело с очень большими целыми числами. Например, если мы умножаем очень большое целое число на другое очень большое целое число, которое настолько большое, что не может быть записано в компьютере, то применение вышеупомянутых свойств позволяет уменьшить первые два операнда прежде, чем начать умножение. Другими словами, перечисленные свойства позволяют нам работать с меньшими числами. Этот факт станет понятнее при обсуждении экспоненциальных операций в последующих лекциях.

Пример 2.18

Следующие примеры показывают приложение вышеупомянутых свойств.

1. $(1\,723\,345 + 2\,124\,945) \bmod 11 = (8 + 9) \bmod 11 = 6$
2. $(1\,723\,345 - 2124\,945) \bmod 11 = (8 - 9) \bmod 11 = 10$
3. $(1\,723\,345 \times 2\,124\,945) \bmod 11 = (8 \times 9) \bmod 11 = 6$

Пример 2.19

В арифметике мы часто должны находить остаток от степеней числа 10 при делении на целое число. Например, мы должны найти $10 \bmod 3$, $10^2 \bmod 3$, $10^3 \bmod 3$, и так далее. Мы также должны найти $10 \bmod 7$, $10^2 \bmod 7$, $10^3 \bmod 7$, и так далее. Третье свойство модульных операторов, упомянутое выше, делает жизнь намного проще.

$10^n \bmod x = (10 \bmod x)^n$ Применение третьего свойства n раз.

Мы имеем

$10 \bmod 3 = 1 \rightarrow 10^n \bmod 3 = (10 \bmod 3)^n = 1$
$10 \bmod 9 = 1 \rightarrow 10^n \bmod 9 = (10 \bmod 9)^n = 1$
$(10 \bmod 7) = 3 \rightarrow 10^n \bmod 7 = (10 \bmod 7)^n = 3^n \bmod 7$

Пример 2.20

Мы уже говорили, что в арифметике остаток от целого числа, разделенного на 3, такой же, как остаток от суммы деления его десятичных цифр. Другими словами, остаток от деления 6371 равен числу 17, разделенному на 3, потому что 6 +

3 + 7 + 1 = 17. Мы можем доказать, что это утверждение использует свойства модульного оператора. Запишем целое число как сумму его цифр, умноженных на степени 10.

$a = a_n\ 10^n + \ldots\ldots\ldots + a_1 10^1 + a_0 10^0$
Например: $6371 = 6 \times 10^3 + 3 \times 10^2 + 7 \times 10^1 + 1 \times 10^0$

Теперь мы можем применить модульную операцию к двум сторонам равенства и использовать результат предыдущего примера, где остаток $10^n \bmod 3$ равен 1.

$$\begin{aligned} a \bmod 3 &= (a_n \times 10^n + \ldots\ldots\ldots + a_1 \times 10^1 + a_0 \times 10^0) \bmod 3 \\ &= (a_n \times 10^n) \bmod 3 + \ldots\ldots + (a_1 \times 10^1) \bmod 3 + (a_0 \times 10^0 \bmod 3) \bmod 3 \\ &= (a_n \bmod 3) \times (10^n \bmod 3) + \ldots\ldots\ldots + (a_1 \bmod 3) \times (10^1 \bmod 3) + \\ &\quad (a_0 \bmod 3) \times (10^0 \bmod 3) \bmod 3 \\ &= (a_n \bmod 3) + \ldots\ldots\ldots + (a_1 \bmod 3) + (a_0 \bmod 3) \bmod 3 \\ &= (a_n + \ldots\ldots\ldots + a_1 + a_0) \bmod 3 \end{aligned}$$

Инверсии

Когда мы работаем в модульной арифметике, нам часто нужно найти операцию, которая позволяет вычислить величину, обратную заданному числу. Мы обычно ищем **аддитивную инверсию** (оператор, обратный сложению) или **мультипликативную инверсию** (оператор, обратный умножению).

Аддитивная инверсия

В Z_2 два числа a и b **аддитивно инверсны** друг другу, если $b = n - a$. Например,

$$a + b \equiv 0 \pmod{n}$$

В Z_n аддитивная инверсия числу a может быть вычислена как $b = n - a$. Например, аддитивная инверсия 4 в Z_{10} равна $10 - 4 = 6$.

В модульной арифметике каждое целое число имеет аддитивную инверсию. Сумма целого числа и его аддитивной инверсии сравнима с 0 по модулю *n*.

Обратите внимание, что в модульной арифметике каждое число имеет аддитивную инверсию, и эта инверсия уникальна; каждое число имеет одну и только одну аддитивную инверсию. Однако инверсия числа может быть тем же самым числом.

Пример 2.21

Найдите все взаимно обратные пары по сложению в Z_{10}.

Решение

Даны шесть пар аддитивных инверсий — (0, 0), (1, 9), (2, 8), (3, 7), (4, 6) и (5, 5). В этом списке 0 — инверсия самому себе; также и 5. Обратите внимание: аддитив-

ные инверсии сложения обратны друг другу; если 4 — аддитивная инверсия 6, тогда 6 — также аддитивная инверсия числу 4.

Мультипликативная инверсия

В Z_n два числа a и b мультипликативно инверсны друг другу, если

$$a \times b \equiv 1 (\text{mod } n)$$

Например, если модуль равен 10, то мультипликативная инверсия 3 есть 7. Другими словами, мы имеем $(3 \times 7) \bmod 10 \times 1$.

В модульной арифметике целое число может или не может иметь мультипликативную инверсию. Целое число и его мультипликативная инверсия сравнимы с 1 по модулю n.

Может быть доказано, что a имеет мультипликативную инверсию в Z_n, если только НОД(n, a) = 1. В этом случае говорят, что a и n **взаимно простые.**

Пример 2.22

Найти мультипликативную инверсию 8 в Z_{10}.

Решение

Мультипликативная инверсия не существует, потому что НОД (10, 8) = 2 ≠ 1. Другими словами, мы не можем найти число между 0 и 9, такое, что при умножении на 8 результат сравним с 1 по mod 10.

Пример 2.23

Найти все мультипликативные инверсии в Z_{10}.

Решение

Есть только три пары, удовлетворяющие условиям существования мультипликативной инверсии: (1, 1), (3, 7) и (9, 9). Числа 0, 2, 4, 5, 6 и 8 не имеют мультипликативной инверсии.

Мы можем проверить, что

$(1 \times 1) \bmod 10 = 1 \qquad (3 \times 7) \bmod 10 = 1 \qquad (9 \times 9) \bmod 10 = 1$

Пример 2.24

Найти все мультипликативные обратные пары в Z_{11}.

Решение

Мы имеем семь пар: (1, 1), (2, 6), (3, 4), (5, 9), (7, 8), (9, 9) и (10, 10). При переходе от Z_{10} к Z_{11} число пар увеличивается. При Z_{11} НОД (11, a) = 1 (взаимно простые) для всех значений a, кроме 0. Это означает, что все целые числа от 1 до 10 имеют мультипликативные инверсии.

Целое число a в Z_n имеет мультипликативную инверсию тогда и только тогда, если НОД (n, a) ≡ 1(mod n)

Расширенный алгоритм Евклида, который мы обсуждали ранее в этой лекции, может найти мультипликативную инверсию *b* в Z_n, когда даны *n* и *b* и инверсия существует. Для этого нам надо заменить первое целое число a на n (модуль). Далее мы можем утверждать, что алгоритм может найти *s* и *t*, такие, что $s \times n + b \times t$ = НОД (*n, b*). Однако если мультипликативная инверсия b существует, НОД (*n, b*) должен быть 1. Так что уравнение будет иметь вид

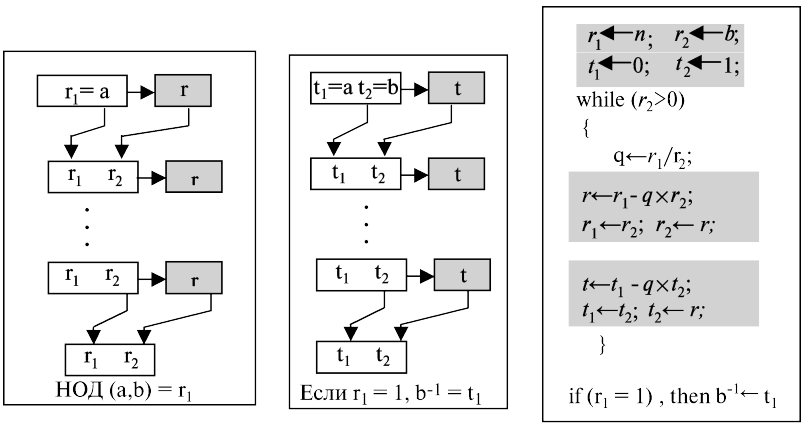
$$(s \times n) + (b \times t) = 1$$

Теперь мы применяем операции по модулю к обеим сторонам уравнения. Другими словами, мы отображаем каждую сторону к Z*n*. Тогда мы получим

$(s \times n + b \times t) \bmod n = 1 \bmod n$

$[(s \times n) \bmod n] + [(b \times t) \bmod n] = 1 \bmod n$

$0 + [(b \times t) \bmod n] = 1$



а) Процесс б) Алгоритм

$(b \times t) \bmod n = 1$ → Это означает, что t – это мультипликативная инверсия b в Z_n

Обратите внимание, что $[(s \times n) \bmod n]$ на третьей строке — 0, потому что, если мы делим $(s \times n)$ на n, частное — *s*, а остаток — 0.

q	r_1	r_2	r	t_1	t_2	t
2	26	11	4	0	1	−2
2	11	4	3	1	−2	5
1	4	3	1	−2	5	−7
3	3	1	0	5	−7	26
	1	0		−7	26	

Расширенный алгоритм Евклида находит мультипликативные инверсии *b* в Z_n, когда даны *n* и *b* и НОД *(n, b) = 1.*
Мультипликативная инверсия *b* — это значение *t*, отображенное в Z_n.

Рисунок 2.15 показывает, как мы находим мультипликативную инверсию числа, используя расширенный алгоритм Евклида.

Рис. 2.15. Применение расширенного алгоритма Евклида для поиска мультипликативной инверсии

Пример 2.25

q	r_1	r_2	r	t_1	t_2	t
4	100	23	8	0	1	4
2	23	8	7	1	−4	9
1	8	7	1	−4	9	−13
7	7	1	0	9	−13	100
	1	0		13	100	

Найти мультипликативную инверсию 11 в Z_{26}.

Решение

Мы используем таблицу, аналогичную одной из тех, которые мы уже применяли прежде при данных $r_1 = 26$ и $r_2 = 11$. Нас интересует только значение t.

НОД (26, 11) = 1, что означает, что мультипликативная инверсия 11 существует. Расширенный алгоритм Евклида дает $t_1 = (-7)$. Мультипликативная инверсия равна $(-7) \bmod 26 = 19$. Другими словами, 11 и 19 — мультипликативная инверсия в Z_{19}. Мы можем видеть, что $(11 \times 19) \bmod 26 = 209 \bmod 26 = 1$.

Пример 2.26

q	r_1	r_2	r	t_1	t_2	t
2	26	12	2	0	1	−2
6	12	2	0	1	−2	13
	2	0		−2	13	

Найти мультипликативную инверсию 23 в Z_{100}.

Решение

Мы используем таблицу, подобную той, которую применяли до этого при $r_1 = 100$ и $r_2 = 23$. Нас интересует только значение *t*.

НОД (100, 23) — 1, что означает, что инверсия 23 существует. Расширенный Евклидов алгоритм дает $t_1 = -13$. Инверсия — $(-13) \bmod 100 = 87$. Другими словами, 13 и 87 — мультипликативные инверсии в Z_{100}. Мы можем видеть, что $(23 \times 87) \bmod 100 = 2001 \bmod 100 = 1$.

Пример 2.27

Найти инверсию 12 в Z_{26}.

Решение

Мы используем таблицу, подобную той, которую мы применяли раньше при $r_1 = 26$ и $r_2 = 12$.

НОД(26,12)=2 ≠ 1, что означает отсутствие для числа 12 мультипликативной инверсии в Z_{26}

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Таблица сложения в Z_{10}

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Таблица умножения в Z_{10}

Сложение и умножение таблиц

Рисунок 2.16 показывает две таблицы для сложения и умножения. При сложении таблиц каждое целое число имеет аддитивную инверсию. Обратные пары могут быть найдены, если результат их сложения — ноль. Мы имеем (0, 0), (1, 9), (2, 8), (3, 7), (4, 6) и (5, 5). При умножении таблиц мы получаем только три мультипликативных пары (1, 1), (3, 7) и (9, 9). Пары могут быть найдены, когда результат умножения равен 1. Обе таблицы симметричны по диагонали, от левой вершины к нижней вершине справа. При этом можно обнаружить свойства коммутативности для сложения и умножения ($a+b = b+a$ и $a \times b = b \times a$). Таблица сложения также показывает, что каждый ряд или колонка может поменяться с другим рядом или колонкой. Для таблицы умножения это неверно.

Рис. 2.16. Таблицы сложения и умножения для Z_{10}

Различные множества для сложения и умножения

В криптографии мы часто работаем с инверсиями. Если отправитель посылает целое число (например, ключ для шифрования слова), приемник применяет инверсию этого целого числа (например, ключ декодирования). Если это действие (алгоритм шифрования/декодирования) является сложением, множество Z_n может быть использовано как множество возможных ключей, потому что каждое целое число в этом множестве имеет аддитивную инверсию. С другой стороны, если действие (алгоритм шифрования/декодирования) — умножение, Z_n не может быть множеством возможных ключей, потому что только некоторые члены этого множества имеют

$Z_6 = \{0,1, 2, 3,4,5\}$ $Z_{6*} = \{l, 5\}$

$Z_7 = = \{0,1, 2, 3,4, 5, 6\}$ $Z_{7*} = \{l, 2,3,4,5,6\}$

$Z_{10} = \{0, l, 2,3,4,5,6,7,8,9\}$ $Z_{10*} = \{1,3,7,9\}$

мультипликативную инверсию. Нам нужно другое множество, которое является подмножеством Z_n и включает в себя только целые числа, и при этом в Z_n они имеют уникальную мультипликативную инверсию. Это множество обозначается Zn*. Рисунок 2.1 показывает некоторые случаи двух множеств. Обратите внимание, что множество Zn* может быть получено из таблицы умножения типа показанной на рис. 2.16.

Каждый член Z_n имеет аддитивную инверсию, но только некоторые члены имеют мультипликативную инверсию. Каждый член Zn* имеет мультипликативную инверсию, но только члены множества имеют аддитивную инверсию.

Мы должны использовать Z_n, когда необходимы аддитивные инверсии; мы должны использовать Zn*, когда необходимы мультипликативные инверсии.

Рис. 2.17. Некоторые множества Zn и Z_{n*}

Еще два множества

Криптография часто использует еще два множества: Z_p, и Z_{p*}. Модули в этих двух множествах — простые числа. Простые числа будут обсуждаться в следующих лекциях; пока можно сказать, что простое число имеет только два делителя: целое число 1 и само себя.

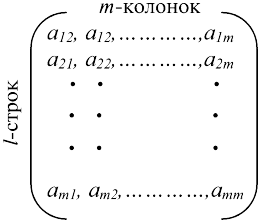
Множество Z_p — то же самое, что и Z_n, за исключением того, что *n — простое число*. Z_p содержит все целые числа от 0 до $p - 1$. Каждый элемент в Zp имеет аддитивную инверсию; каждый элемент кроме 0 имеет мультипликативную инверсию.

Множество Z_{p*} — то же самое, что Z_{n*}, за исключением того, что Z_{p*} содержит все целые числа от 0 до $p - 1$. Каждый элемент в Z_p имеет аддитивную и мультипликативную инверсии. Z_{p*} — очень хороший кандидат, когда мы нуждаемся во множестве, которое поддерживает аддитивную и мультипликативную инверсии.

Ниже показаны два множества, когда $p = 13$.

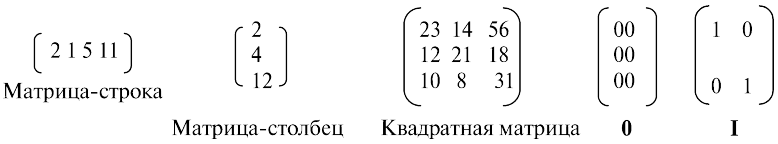
$Z_{13} = \{0,1,2, 3,4, 5, 6, 7, 8, 9, 11,12\}$,
$Z_{13*} = \{0,1,2, 3,4, 5, 6, 7, 8, 9, 11,12\}$,

Матрица A:



2.3. Матрицы

В криптографии мы должны обрабатывать матрицы. Хотя эта тема принадлежит специальному разделу алгебры, который называется линейной алгеброй, необходим краткий обзор матриц для подготовки к изучению криптографии. Читатели, знакомые с этими вопросами, могут пропустить часть или весь



этот раздел. Раздел начинается с некоторых определений и примеров использования матрицы в модульной арифметике.

Определения

Матрица — прямоугольный массив, содержащий $l \times m$ элементов, в которых l — число строк, m — число столбцов. Матрица обычно обозначается заглавной буквой, такой, как **A**. Элемент a_{ij} расположен в i-той строке и j-том столбце. Хотя элементы матрицы могут быть любым множеством чисел, мы обсуждаем только матрицы с элементами в **Z**.

Рисунок 2.18 показывает матрицу.

Рис. 2.18. Матрица размера $l \times m$

Если матрица имеет только одну строку ($l = 1$), она называется **матрицей-строкой**; если она имеет только один столбец ($m = 1$), то называется **матрицей-столбцом.** Матрица называется квадратной, если число строк равно числу столбцов ($l = m$) и содержит элементы главной диагонали a_{11}, a_{22}, ...,a_{mm}. Матрица обозначается **0**, если все строки и все столбцы содержат нули. **Единичная матрица** обозначается **I**, если она квадратная и содержит все единицы на главной диагонали и все нули на других местах. Рисунок 2.19 показывает некоторые примеры матриц с элементами из Z.

Рис. 2.19. Примеры матриц

Операции и уравнения

В линейной алгебре для матриц определены одно уравнение (равенство) и четыре операции (сложение, вычитание, умножение и скалярное умножение).

Равенство

Две матрицы равны, если они имеют одинаковое число строк и столбцов и соответствующие элементы равны. Другими словами, **A = B, если** мы имеем $a_{ij} = b_{ij}$ для всех i и j.

$$\begin{pmatrix} 12 & 4 & 4 \\ 11 & 12 & 30 \end{pmatrix} = \begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 10 \end{pmatrix} + \begin{pmatrix} 7 & 2 & 3 \\ 8 & 10 & 20 \end{pmatrix}$$

C = A + B

$$\begin{pmatrix} -2 & 0 & -2 \\ -5 & -8 & 10 \end{pmatrix} = \begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 10 \end{pmatrix} - \begin{pmatrix} 7 & 2 & 3 \\ 8 & 10 & 20 \end{pmatrix}$$

C = A - B

Рис. 2.20. Сложение и вычитание матриц

Сложение и вычитание

Операция сложения двух матриц может применяться, если матрицы имеют одинаковое число столбцов и строк. Сложение записывают как **C = A + B.** В этом случае полученная в результате матрица **C** имеет тот же номер строк и столбцов, как **A** или **B**. Каждый элемент **C** — сумма двух соответствующих элементов **A** и **B**: $a_{ij} + b_{ij}$.

$$\begin{matrix} C \\ (53) \end{matrix} = \begin{matrix} A \\ (5\ 2\ 1) \end{matrix} \times \begin{matrix} B \\ \begin{pmatrix} 7 \\ 8 \\ 2 \end{pmatrix} \end{matrix}$$

В которой $53 = 5 \times 7 + 2 \times 8 + 1 \times 2$

Операция вычитания производится аналогично сложению, за исключением того, что каждый элемент **B** вычитается из соответствующего элемента **A**: $d_{ij} = a_{ij} - b_{ij}$.

Пример 2.28

Рисунок 2.20 показывает пример сложения и вычитания.

$$\begin{matrix} \mathbf{C} \\ \begin{pmatrix} 52 & 18 & 14 & 9 \\ 41 & 21 & 22 & 7 \end{pmatrix} \end{matrix} = \begin{matrix} \mathbf{A} \\ \begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix} \end{matrix} \times \begin{matrix} \mathbf{B} \\ \begin{pmatrix} 7 & 3 & 2 & 1 \\ 8 & 0 & 0 & 2 \\ 1 & 3 & 4 & 0 \end{pmatrix} \end{matrix}$$

Умножение

Две матрицы различного размера могут быть перемножены, если число столбцов первой матрицы совпадает с числом строк второй матрицы. Если **A** — матрица размера $l \times m$, а матрица **B** размера $m \times p$, то произведением будет матри-

$$\begin{matrix} \mathbf{B} \\ \begin{pmatrix} 15 & 6 & 3 \\ 9 & 6 & 12 \end{pmatrix} \end{matrix} = 3 \times \begin{matrix} \mathbf{A} \\ \begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix} \end{matrix}$$

ца **C** размером $l \times p$. Если элемент матрицы A обозначить a_{ij}, а каждый элемент матрицы **B** обозначить b_{jk}, то элемент матрицы **C** — c_{ik} — вычисляется следующим

образом:

$$c_{ik} = \sum a_{ij} \times b_{jk} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \ldots + a_{im} \times b_{mj}$$

Пример 2.29

Рисунок 2.21 показывает произведение матрицы-строки (1 × 3) на матрицу-столбец (3 × 1). В результате получаем матрицу размером 1 × 1.

Рис. 2.21. Умножение матрицы-строки на матрицу-столбец

Пример 2.30

Рисунок 2.22 показывает произведение матрицы 2 × 3 на матрицу 3 × 4. В результате получаем матрицу 2 × 4.

Рис. 2.22. Умножение матрицы 2 × 3 на матрицу 3 × 4

Скалярное умножение

Мы можем также умножить матрицу на число (называемое ***скаляр***). Если **A** — матрица $l \times m$ и x — скаляр, то $\mathbf{C} = x\mathbf{A}$ — матрица $l \times m$, в которой $c_{ij} = x \times a_{ij}$.

Рис. 2.23. Скалярное умножение

Пример 2.31

$$\det\begin{bmatrix} 5 & 2 \\ 3 & 4 \end{bmatrix} = (-1)^{1+1} \times 5 \times \det[4] + (-1)^{1+2} \times 2 \times \det[3] \longrightarrow 5 \times 4 - 2 \times 3 = 14$$

или

$$\det\begin{bmatrix} a_{11} & a_{22} \\ a_{21} & a_{22} \end{bmatrix} = a_{11} \times a_{22} - a_{12} \times a_{21}$$

Рисунок 2.23 показывает пример скалярного умножения.

$$\det\begin{bmatrix} 5 & 2 & 1 \\ 3 & 0 & -4 \\ 2 & 1 & 6 \end{bmatrix} = (-1)^{1+1} \times 5 \times \det\begin{bmatrix} 0 & -4 \\ 1 & 6 \end{bmatrix} + (-1)^{1+2} \times 2 \times \det\begin{bmatrix} 3 & -4 \\ 2 & 6 \end{bmatrix} +$$

$$+(-1)^{1+3} \times 1 \times \det\begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} = (1) \times 5 \times (4) + (-1) \times 2 \times (26) + (1) \times 1 \times 1 \times (3) = -29$$

Детерминант

Детерминант — квадратная матрица **A** размера m × m, обозначаемая как det (**A**) — скалярное вычисление рекурсивно, как это показано ниже:

1. If $m = 1$, det (**A**) = a_{11}
2. If $m > 1$, det (**A**) = $\sum_{i=1...m} (-1)^{i+j} \times a_{ij} \times \det(A_{ij})$

где A_{ij} получается из **A** удалением i-той строки j-того столбца.

Детерминант определяется только для квадратной матрицы.

Пример 2.32

Рисунок 2.24 показывает, как можно вычислить детерминант матрицы 2 × 2, базируясь на детерминанте матрицы 1 × 1 и используя приведенное выше рекурсивное определение. Пример доказывает, что когда m 1 или 2, это позволяет найти детерминант матрицы достаточно просто.

Рис. 2.24 Вычисление детерминанта матрицы 2 × 2

Пример 2.33

Рисунок 2.25 показывает вычисление детерминанта матрицы 3 × 3.

Рис.2.25. Вычисление детераминаната матрицы 3 × 3

Инверсии

Матрицы имеют аддитивные и мультипликативные инверсии.

Аддитивная инверсия

Аддитивная инверсия матрицы — это другая матрица B, такая, что **A + B = 0.** Другими словами, мы имеем элементы $b_{ij} = -a_{ij}$ для всех значений i и j. Обычно аддитивная инверсия A обозначается как (-A).

Мультипликативная инверсия

Мультипликативная инверсия определена только для квадратных матриц. Мультипликативная инверсия квадратной матрицы **A** — квадратная матрица **B**, такая, что **A × B = B × A = I.** Обычно мультипликативная инверсия обозначается как A^{-1}. Мультипликативная инверсия существует только, если det (**A**) имеет мультипликативную инверсию в соответствующем инверсном множестве. Если целое число не имеет мультипликативной инверсии в **Z,** то не существует мультиплика-

$$A = \begin{bmatrix} 3 & 5 & 7 & 2 \\ 1 & 4 & 7 & 2 \\ 6 & 3 & 9 & 17 \\ 13 & 5 & 4 & 16 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 15 & 21 & 0 & 15 \\ 23 & 9 & 0 & 22 \\ 15 & 16 & 18 & 3 \\ 24 & 7 & 15 & 3 \end{bmatrix}$$

det (A) = 21 det (A^{-1}) = 5

тивной инверсии матрицы в Z. Однако матрицы с реальными элементами имеют инверсии, только если det (A) ≠ 0.

Мультипликативные инверсии определены только для квадратных матриц.

Матрицы вычетов

Криптография использует матрицы вычетов: матрицы могут содержать все элементы из Z_n. Все операции на матрицах вычетов выполняются так же, как и на матрицах целых чисел, за исключением того, что операции производятся в модульной арифметике. Есть одно интересное свойство: матрица вычетов имеет мультипликативную инверсию, если детерминант матрицы имеет мультипликативную инверсию в Z_n. Другими словами, матрица вычета имеет мультипликативную инверсию, если НОД (det (A), n) = 1.

Пример 2.34

Рисунок 2.26 показывает матрицу вычетов в Z_n и его мультипликативной инверсии $\mathbf{A^{-1}}$. Возьмем детерминант det (A) = 21, который имеет мультипликативную инверсию 5 в Z_{26}. Обратите внимание, что когда мы умножаем эти две матрицы, то результат — единичная матрица мультипликативная матрица, в Z_{26}.

Рис. 2.26. Матрица вычетов и мультипликативная инверсия

Сравнение

Две матрицы, сравнимые по модулю n, записываются как $\mathbf{A} \equiv \mathbf{B} \pmod{n}$, если они имеют одинаковое число строк и столбцов и все соответствующие элементы — сравнимые по модулю *n*. Другими словами, $\mathbf{A} \equiv \mathbf{B} \pmod{n}$, если $a_{ij} \equiv b_{ij} \pmod{n}$ для всех *i* и *j*.

2.4. Линейное сравнение

Криптография часто включает в себя решение уравнения или множества уравнений одной или более переменных с коэффициентом в Z_n. Этот раздел показывает, как решать уравнения с одним неизвестным, когда степень переменной равна 1 (линейное уравнение).

Линейные уравнения с одним неизвестным, содержащие сравнения

Давайте посмотрим, как решаются уравнения с одним неизвестным, содержащие уравнения, то есть уравнения $ax \equiv b \pmod{n}$. Уравнение этого типа может не иметь ни одного решения или иметь ограниченное число решений. Предположим, что НОД $(a, n) = d$. Если $d \mid b$, решение не существует. Если $d|b$, то имеется d решений.

Если $d|b$, то для того, чтобы найти решения, мы используем следующую стратегию.

1. Сократить уравнение, разделив обе стороны уравнения (включая модуль) на *d*.
2. Умножить обе стороны сокращенного уравнения на мультипликативную инверсию, чтобы найти конкретное решение x_0.
3. Общие решения будут $x = x_0 + k\,(n/d)$ для $k = 0, 1..., (d - 1)$.

Пример 2.35

Решить уравнение $10x \equiv 2 (\text{mod } 15)$.

Решение

Сначала мы найдем НОД(10,15) = 5. Полученное число 5 не делится на 2, решение отсутствует.

Пример 2.36

Решить уравнение $14x \equiv 12 (\text{mod } 18)$.

Решение

Заметим, что НОД (14, 18) = 2. Поскольку 2 делит 12, мы имеем точно два решения, но сначала сократим уравнение:

$14x \equiv 12 (\text{mod } 18) \rightarrow 7x \equiv 6 (\text{mod } 9) \rightarrow x \equiv 6(7^{-1}) (\text{mod } 9)$

$x_0 \times 6(7^{-1}) (\text{mod } 9) = (6 \times 4) (\text{mod } 9) = 6$

$x_1 = x_0 + 1 \times (18/2) = 15$

$$\begin{matrix} a_{11}x_1 + a_{12}x_2 + \ldots\ldots + a_{1n} = b_1 \\ a_{21}x_2 + a_{22}x_2 + \ldots\ldots + a_{2n} = b_2 \\ \cdot \quad \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \quad \cdot \\ a_{n1}x_1 + a_{n2}x_2 + \ldots\ldots + a_{nn}x_n = \end{matrix}$$

а) Система уравнений

$$\begin{pmatrix} a_{11} & a_{12} & \ldots\ldots & a_{1n} \\ a_{21} & a_{22} & \ldots\ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots\ldots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \equiv \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \equiv \begin{pmatrix} a_{11} & a_{12} & \ldots\ldots & a_{1n} \\ a_{21} & a_{22} & \ldots\ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots\ldots & a_{nn} \end{pmatrix}^{-1} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

б) интерпретация системы уравнений в) решение

Оба решения, 6 и 15, удовлетворяют уравнению сравнения, потому что (14 × 6) mod 18 = 12, а также (14 × 15) mod 18 = 12.

Пример 2.37

Решить уравнение $3x + 4 \equiv 6 (\text{mod } 13)$.

Решение

Сначала мы приводим уравнение к форме $ax \equiv b (\text{mod } n)$. Мы прибавляем (–4) к обеим сторонам (4 аддитивная инверсия). Получим $3x \equiv 2 (\text{mod } 13)$. Поскольку НОД (3, 13) = 1, уравнение имеет только одно решение, $x_0 = (2 \times 3^{-1}) \text{ mod } 13 = 18 \text{ mod } 13 = 5$. Мы можем видеть, что ответ удовлетворяет первоначальному уравнению: $3 \times 5 + 4 = 6 (\text{mod } 13)$.

Система линейных уравнений, содержащих сравнения

Мы можем решить систему линейных уравнений с одним и тем же модулем, если матрица, сформированная из коэффициентов системы уравнений, имеет обратную матрицу. Для решения уравнения составляются три матрицы. Первая — квадратная матрица — формируется из коэффициентов уравнения. Вторая — матрица-столбец — составляется из переменных. Третья — матрица-столбец в правой стороне оператора сравнения — состоит из значения b_n. Мы можем это уравнение представить как произведение матриц. Если обе стороны сравнения умножить на мультипликативную инверсию первой матрицы, в результате мы получим решение системы уравнений, как это показано на рис. 2.27.

Рис. 2.27. Система линейных уравнений

Пример 2.38

Решить систему следующих трех уравнений:

$3x + 5y + 7z \equiv 3 \pmod{16}$
$x + 4y + 13z \equiv 5 \pmod{16}$
$2x + 7y + 3z \equiv 4 \pmod{16}$

Решение

Здесь x, y и z играют роли x_1, x_2, и x_3. Матрица, сформированная из коэффициентов уравнений, — обратима. Мы находим мультипликативную инверсию матрицы и умножаем ее на матрицу столбца, сформированную из 3, 5 и 4. Результат — $x \equiv 15 \pmod{16}$, $y \equiv 4 \pmod{16}$ и $z \equiv 14 \pmod{16}$. Мы можем проверить ответ, подставляя эти значения в уравнения.

2.5. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Несколько книг дают простой, но полный охват теории чисел: [Ros06], [Sch99], [Cou99] и [BW00]. Матрицы обсуждаются в любой книге по линейной алгебре; [LEF04], [DF04] и [Dur05] — это хорошие книги для начинающих.

Сайты

Нижеследующие сайты дают больше информации о темах, рассмотренных в этой лекции.

http:en.wikipedia.org/wiki/Euclidean_algorithm
http:en.wikipedia.org/wiki/Multiplicative_inverse
http:en.wikipedia.org/wiki/Additive-inverse

2.6. Итоги

- Множество целых чисел, обозначаемое **Z**, содержит все целые числа от отрицательной бесконечности до положительной бесконечности. Для целых чисел определены три общих бинарных операции — сложение, вычитание и умножение. Деление не удовлетворяет определению бинарности, потому что требует два выхода вместо одного.
- В арифметике целых чисел, если мы делим a на n, мы можем получить q и r. Отношение между этими четырьмя целыми числами можно показать как $q \times n + r$. Мы говорим $a|b$, если $a = q \times n$. В этой лекции мы рассмотрели четыре свойства теории делимости.
- Два положительных целых числа могут иметь больше чем один общий делитель. Но мы обычно интересуемся наибольшим общим делителем. Алгоритм Евклида дает эффективный и систематический алгоритм вычисления наибольшего общего делителя двух целых чисел.
- Расширенный алгоритм Евклида может вычислить НОД (a, b) и вычислить значение s и t, которые удовлетворяют уравнению $as + bt$ = НОД (a, b).
- Линейное диофантово уравнение двух переменных: ax + by = c. Оно имеет частное и общие решения.
- В модульной арифметике мы интересуемся только остатками; мы хотим знать значение r, когда мы делим a на n. Мы используем новый оператор, названный модулем (mod), такой, что $a \bmod n = r$. Здесь n называется модулем, а r называется *вычетом*.
- Результат операции по модулю n — всегда целое число между 0 и n. Мы можем сказать, что операция по модулю n создает набор, который в модульной арифметике называется множеством наименьших вычетов по модулю n, или Zn.
- Отображение из Z в Zn не совпадают «один в один». Определенные элементы Z могут быть отображены в элемент Zn. В модульной арифметике все целые числа в Z, отображаемые в Z_n, называются сравнениями по модулю. Для обозначения этой операции применяется оператор сравнения (≡).
- Система вычетов [a] — множество целых чисел, сравнимых по модулю n. Это множество всех целых чисел $x = a$ (mod n).
- Три бинарных операции (сложение, вычитание и умножение), определенные для множества Z, могут быть также определены для множества Z_n. При необходимости результат может быть отражен в Z_n при помощи операции mod.
- В этой лекции для модульных операторов были определены несколько свойств.
- В Z_n два числа a и b — аддитивные инверсии по отношению друг к другу, если $a + b \equiv 0$ (mod n). Они — мультипликативные инверсии по отношению друг к другу, если $a \times b \equiv 1$ (mod n). Целое число a имеет мультипликативную инверсию в Z_n тогда и только тогда, когда НОД (n, a) = 1(a и n — взаимно простые числа).

- Расширенный алгоритм Евклида находит мультипликативные инверсии b в Z_n , когда даны n и b и НОД $(n, b) = 1$. Мультипликативная инверсия b — это значение t при соответствующем отображении в Zn.
- Матрица — прямоугольный массив $l \times m$. элементы, где l является номером строки, а m — номер столбца. Мы обозначаем матрицу заглавной буквой и жирным шрифтом, например, **A.** Элемент a_{ij} расположен в i-той строке и j-том столбце.
- Две матрицы равны, если они имеют одинаковое число строк и столбцов и соответствующие элементы равны.
- Сложение и вычитание можно делать только для матриц равного размера. Мы можем умножить друг на друга две матрицы различных размеров, если число столбцов первой матрицы совпадает с числом строк второй матрицы.
- В матрицах вычетов все элементы берутся из Z_n .
- Все операции на матрицах вычетов проводятся в модульной арифметике. Матрица вычета имеет инверсию, если детерминант матрицы имеет инверсию.
- Уравнение $ax \equiv b \pmod{n}$ не может иметь решения или ограниченное число решений. Если НОД (a, n) | b, то имеется ограниченное число решений.
- Система линейных уравнений с тем же самым модулем может быть решена, если матрица, сформированная из коэффициентов уравнений, имеет инверсию.

2.7. Набор для практики

Обзорные вопросы

1. Покажите различие между Z и Z_n. Какое из этих множеств может содержать отрицательные целые числа? Как мы можем отобразить целое число в Z в целое число в Z_n?
2. Перечислите четыре свойства теории делимости, обсужденной в этой лекции. Приведите пример целого числа с единственным делителем. Приведите пример целого числа только с двумя делителями. Приведите пример целого числа с более чем двумя делителями.
3. Определите наибольший общий делитель двух целых чисел. Какой алгоритм может эффективно найти наибольший общий делитель?

$5 \mid 26 \quad 3 \mid 123 \quad 27 \nmid 127 \quad 15 \nmid 21 \quad 23 \mid 96 \quad 8 \mid 5$

4. Что такое линейное диофантово уравнение двух переменных? Сколько решений может иметь такое уравнение? Как может быть найдено решение(я)?
5. Что такое оператор по модулю и какие у него имеются приложения? Перечислите все свойства, которые мы упоминали в этой лекции для операций по модулю.

6. Определите сравнение и сопоставьте его свойства со свойствами равенства.
7. Определите систему вычетов и наименьший вычет.
8. Какова разница между множеством Z_n и множеством Z_n^*? В каком множестве каждый элемент имеет аддитивную инверсию? В каком множестве каждый элемент имеет мультипликативную инверсию? Какой алгоритм используется, чтобы найти мультипликативную инверсию целого числа в Z_n?
9. Дайте определение матрицы. Что такое матрица-строка? Что такое матрица-столбец? Что такое квадратная матрица? Какая матрица имеет детерминант? Какая матрица может иметь инверсию?
10. Определите линейное сравнение. Какой алгоритм может использоваться, чтобы решить уравнение $ax \equiv b \pmod n$? Как мы можем решить набор линейных уравнений?

Упражнения

11. Какие из следующих отношений являются истинными, а какие — ложными?

12. Используя алгоритм Эвклида, найдите наибольший общий делитель следующих пар целых чисел:
    а. 88 и 220
    б. 300 и 42
    в. 24 и 320
    г. 401 и 700
13. Решите следующие примеры:
    а. Дано НОД (a, b) = 24, найдите НОД (a, b, 16)
    б. Дано НОД (a, b, c) = 12, найдите НОД (a, b, c, 16)
    в. Найдите НОД (200, 180, и 450)
    г. Найдите НОД (200, 180 450 610)
14. Предположим, что n — неотрицательное целое число.
    а. Найдите НОД ($2n + 1$, n)
    б. Используя результат части а, найдите НОД (201, 100), НОД (81, 40) и НОД (501, 250)
15. Предположим, что n — неотрицательное целое число.
    а. Найдите НОД ($3n + 1$,$2n + 1$).
    б. Используя результат части а, найдите НОД (301, 201) и НОД (121, 81)
16. Используя расширенный алгоритм Евклида, найдите наибольший общий делитель следующих пар и значения s и t:
    а. 4 и 7
    б. 291 и 42
    в. 84 и 320
    г. 400 и 60
17. Найдите результаты следующих операций:
    а. 22 mod 7

б. 140 mod 10
в. –78 mod 13
г. 0 mod 15

18. Выполните следующие операции, сначала используя сокращение :
    а. (273 + 147) mod 10
    б. (4223 + 17323) mod 10
    в. (148 + 14432) mod 12
    г. (2467+461) mod 12
19. Выполните следующие операции, сначала используя сокращение :
    а. (125 × 45) mod 10
    б. (424 × 32) mod 10
    в. (144 × 34) mod 12
    г. (221 × 23) mod 22
20. Используя свойства оператора mod, докажите следующее:
    а. Остаток от любого целого числа, когда оно делится на 10, — самая правая цифра
    б. Остаток от любого целого числа, когда оно делится на 100, — целое число, составленное из двух самых правых цифр
    в. Остаток от любого целого числа, когда оно делится на 1000, — целое число, составленное из трех самых правых цифр
21. Из арифметики известно, что остаток от целого числа при делении на 5 — такой же, что и остаток от деления самой правой цифры на 5. Используйте свойства оператора mod, чтобы доказать это утверждение.
22. Из арифметики известно, что остаток от целого числа при делении на 2 — такой же, что и остаток от деления самой правой цифры на 2. Используйте свойства оператора mod, чтобы доказать это утверждение.
23. Из арифметики известно, что остаток от целого числа при делении на 4 — такой же, что и остаток от деления двух самых правых цифр на 4. Используйте свойства оператора mod, чтобы доказать это утверждение.
24. Из арифметики известно, что остаток от целого числа при делении на 8 — такой же, что и остаток от деления самых правых трех цифр на 8. Используйте свойства оператора mod, чтобы доказать это утверждение.
25. Из арифметики известно, что остаток от целого числа при делении на 9 — такой же, как и остаток от деления суммы его десятичных цифр на 9. Другими словами, остаток от деления 6371 на 9 — такой же, как при делении 17 на 9, потому что 6 + 3 + 7 + 1 = 17. Используйте свойства оператора mod, чтобы доказать это утверждение.
26. Следующие упражнения показывают остатки от степени 10 при делении на 7. Мы можем доказать, что эти значения будут повторяться для более высоких степеней.

$$10^0 \bmod 7 = 1 \quad 10^1 \bmod 7 = 3 \quad 10^2 \bmod 7 = 2$$
$$10^3 \bmod 7 = 1 \quad 10^4 \bmod 7 = -3 \quad 10^5 \bmod 7 = -2$$

Используя вышеупомянутую информацию, найдите остаток от деления целого числа на 7. Проверьте ваш метод с числом 631453672.

27. Следующие упражнения показывают остатки от деления степеней 10 на 11. Мы можем доказать, что эти значения будут повторяться для более высоких степеней.

$$10^0 \bmod 11 = 1 \quad 10^1 \bmod 11 = -1 \quad 10^2 \bmod 11 = 1 \quad 10^3 \bmod 11 = -1$$

Используя вышеупомянутую информацию, найдите остаток от деления целого числа на 11. Проверьте ваш метод с числом 631453672.

28. Следующие упражнения показывают остатки от деления степеней 10 на 13. Мы можем доказать, что эти значения будут повторяться для более высоких степеней.

$$10^0 \bmod 13 = 1 \quad 10^1 \bmod 13 = -3 \quad 10^2 \bmod 13 = -4$$
$$10^3 \bmod 13 = -1 \quad 10^4 \bmod 13 = 3 \quad 10^5 \bmod 13 = 4$$

Используя вышеупомянутую информацию, найдите остаток от целого числа при делении на 13. Проверьте ваш метод с числом 631453672.

29. Назначим числовые значения для заглавных букв латинского алфавита (A = 0, B = 1... Z = 25). Мы можем создать модульную арифметику, используя модуль 26.
    а. Что является (A + N) mod 26 в этой системе?
    б. Чему равно (A + 6) mod 26 в этой системе?
    в. Чему равно (Y – 5) mod 26 в этой системе?
    г. Чему равно (C – 10) mod 26 в этой системе?
30. Перечислите все пары аддитивной инверсии по модулю 20.
31. Перечислите все мультипликативные обратные пары по модулю 20.
32. Найдите мультипликативную инверсию каждого из следующих целых чисел в Z_{180}, используя расширенный алгоритм Евклида.
    а. 38
    б. 7
    в. 132

$$\underset{\mathbf{A}}{\begin{bmatrix}3 & 7 & 10\end{bmatrix}} \times \underset{\mathbf{B}}{\begin{bmatrix}2\\4\\12\end{bmatrix}} \qquad \underset{\mathbf{A}}{\begin{bmatrix}3 & 4 & 6\\1 & 1 & 8\\5 & 8 & 3\end{bmatrix}} \times \underset{\mathbf{B}}{\begin{bmatrix}2 & 0 & 1\\1 & 1 & 0\\5 & 2 & 4\end{bmatrix}}$$

    г. 24
33. Найдите частное и общие решения следующих линейных диофантовых уравнений:
    а. $25x + 10y = 15$

$$\underset{\mathbf{A}}{\begin{bmatrix}3 & 0\\1 & 1\end{bmatrix}} \qquad \underset{\mathbf{B}}{\begin{bmatrix}4 & 2\\1 & 1\end{bmatrix}} \qquad \underset{\mathbf{C}}{\begin{bmatrix}3 & 4 & 6\\1 & 1 & 8\\5 & 5 & 3\end{bmatrix}}$$

б. $19x + 13y = 20$
в. $14x + 21y = 77$
г. $40x + 16y = 88$

34. Покажите, что нет ни одного решения следующих линейных диофантовых уравнений:
а. $15x + 12y = 13$
б. $18x + 30y = 20$
в. $15x + 25y = 69$
г. $40x + 30y = 98$
35. Почтовое отделение продает марки только за 15 центов и за 39 центов. Найдите число марок, которые должен купить клиент, чтобы оплатить пересылку пакета стоимостью 2,70$. Найдите несколько решений.
36. Найдите все решения каждого из следующих линейных уравнений:
а. $3x \equiv 4 \pmod 5$
б. $4x \equiv 4 \pmod 6$
в. $9x \equiv 12 \pmod 7$
г. $256x \equiv 442 \pmod{60}$
37. Найдите все решения каждого из следующих линейных уравнений:
а. $3x + 5 \equiv 4 \pmod 5$
б. $4x + 6 \equiv 4 \pmod 6$
в. $9x + 4 \equiv 12 \pmod 7$
г. $232x + 42 \equiv 248 \pmod{50}$
38. Найдите **(A × B)** mod 16, используя матрицы на рис. 2.28.

Рис. 2.28. Матрицы для упражнения 38

39. На рисунке 2.29 найдите детерминант и мультипликативную инверсию для каждой матрицы вычетов в множестве Z_{10}.

Рис. 2.29. Матрицы для упражнения 39

40. Найдите все решения для следующих систем линейных уравнений:
а. $3x + 5y \equiv 4 \pmod 5$
$2x + y \equiv 3 \pmod 5$
б. $3x + 2y \equiv 5 \pmod 7$
$4x + 6y \equiv 4 \pmod 7$
в. $7x + 3y \equiv 3 \pmod 7$
$4x + 2y \equiv 5 \pmod 7$
г. $2x + 5y \equiv 5 \pmod 8$
$x + 6y \equiv 3 \pmod 8$

Лекция 3. Традиционные шифры с симметричным ключом

Цели и содержание

Эта лекция представляет собой обзор традиционных шифров с симметричным ключом, которые использовались в прошлом. Изучение принципов таких шифров готовит читателя к следующим лекциям, которые рассматривают современные симметричные шифры. Эта лекция имеет несколько целей.

- Определить терминологию и концепции шифров с симметричным ключом.
- Обратить внимание на две категории традиционных шифров: шифры подстановки и шифры перестановки.
- Описать категории криптоанализа, которые использовались обычно для взламывания симметричных шифров.
- Ввести концепцию шифров потока и шифров блокировки.
- Обсудить некоторые наиболее доминирующие шифры, которые применялись в прошлом, такие, например, как машина «Энигма», рассмотренная в этой лекции.

Общая идея шифров с симметричным ключом будет представлена с использованием примеров из криптографии. Вводимые термины и определения используются во всех более поздних лекциях, где речь пойдет о шифрах с симметричным ключом. Затем мы обсуждаем традиционные шифры с симметричным ключом. Эти шифры не применяются сегодня, но мы изучаем их по нескольким причинам. Во-первых, они проще, чем современные шифры, и их легче понять. Во-вторых, они демонстрируют основы криптографии и шифрования. Эти основы могут использоваться для понимания современных шифров. В-третьих, рассказ о них подводит нас к изложению принципов построения и необходимости современных шифров, потому что традиционные шифры могут быть легко атакованы любыми пользователями компьютеров, и шифры, которые были безопасны в прежнее время, не обеспечивают безопасности при современном развитии компьютеров.

3.1. Введение

Рисунок 3.1 иллюстрирует общую идею шифра с симметричным ключом.

На рисунке 3.1 объект, Алиса, может передать сообщение другому объекту, Бобу, по несекретному каналу, учитывая, что противник (назовем его Ева), не может понять содержание сообщения, просто подслушивая его по каналу.

Первоначальное сообщение от Алисы Бобу названо **исходным текстом**; сообщение, передаваемое через канал, названо **зашифрованным текстом**. Чтобы создать зашифрованный текст из исходного текста, Алиса использует **алгоритм шифрования** и **совместный ключ засекречивания**. Для того чтобы создать обычный текст из зашифрованного текста, Боб использует **алгоритм дешифрования** и тот же секретный ключ. Мы будем называть совместное действие алгоритмов шифрования и дешифрования **шифровкой**. **Ключ** — набор значений (чисел), которыми оперирует алгоритм шифровки.

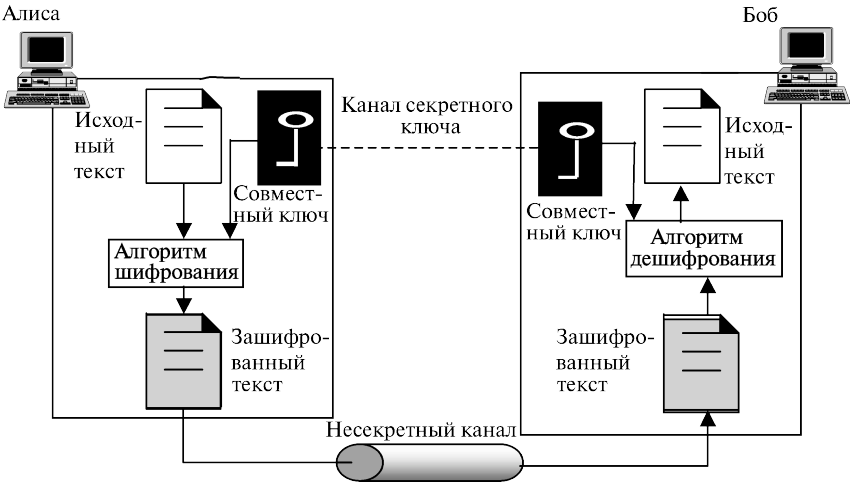


Рис. 3.1. Общая идея шифрования с симметричным ключом

Обратите внимание, что шифрование симметричными ключами использует единственный ключ (ключ, содержащий непосредственно набор кодируемых значений) и для кодирования, и для дешифрования. Кроме того, алгоритмы шифрования и дешифрования — инверсии друг друга. Если P — обычный текст, C — зашифрованный текст, а K — ключ, алгоритм кодирования $E_k(x)$ создает зашифрованный текст из исходного текста.

Алгоритм же дешифрования $D_k(x)$ создает исходный текст из зашифрованного текста. Мы предполагаем, что $E_k(x)$ и $D_k(x)$ инверсны по отношению друг к другу. Они применяются, последовательно преобразуя информацию из одного вида в другой и обратно. Мы имеем:

$$\text{Шифрование: } C = E_k(P), \qquad \text{Расшифровка: } P = D_k(C)$$
$$\text{где } D_k(E_k(x)) = E_k(D_k(x)) = x$$

Мы можем доказать, что исходный текст, созданный Бобом, тот же самый, что и исходный, переданный Алисой. Мы предполагаем, что Боб создает P_1; мы докажем, что $P_1 = P$:

$$\textbf{Алиса: } C = E_k(P) \qquad \textbf{Боб}: P_1 = D_k(C) = D_k(E_k(P)) = P$$

Мы должны подчеркнуть, что согласно принципу Керкхоффа[1] (приведенному далее) лучше делать алгоритм шифрования и дешифрования открытым, но сохранять в тайне совместный ключ.

[1] Огюст Керкхофф (August Kerckoff) — голландский криптограф, сформулировавший в 1883 году принципы создания и распространения алгоритмов шифрования в книге «Военная криптография».

Это означает, что Алиса и Боб нуждаются в другом защищенном канале для обмена ключом засекречивания. Алиса и Боб могут однажды встретиться и обменяться ключом лично. Защищенный канал аналогично представляет собой «встречу лицом к лицу» для обмена ключом. Они могут также довериться третьему лицу, чтобы он дал им одинаковые ключи. Они могут создать временный ключ засекречивания, используя другой вид асимметрично-ключевых шифров, который будет подробно рассмотрен в более поздних лекциях. В этой лекции мы просто принимаем, что существует утвержденный ключ засекречивания между Алисой и Бобом.

Применяя шифрование симметричными ключами, Алиса и Боб могут использовать тот же самый ключ для связи на другом направлении, от Боба к Алисе. Именно поэтому метод назван симметричным.

Другой элемент в шифровании симметричными ключами — число ключей. Алиса нуждается в другом ключе засекречивания, чтобы связаться с другим человеком, скажем, Дэвидом. Если есть m группа людей, в которой каждый должен иметь связь друг с другом, сколько ключей необходимо? Ответ — $(m \times (m. - 1))/2$, потому что каждому человеку надо $m - 1$ ключ, чтобы связаться с остальной частью группы, но ключ между A и B может использоваться в обоих направлениях. В более поздних лекциях мы увидим, как решается эта проблема.

Шифрование можно представлять себе как замок, который запирает ящик, содержащий сообщение; дешифрование можно представлять себе как открытие замка такого ящика. В шифровании симметричными ключами один и тот же ключ замыкает и размыкает замок, как это показано на рис. 3.2. В более поздних лекциях будет рассказано, что шифрование асимметричными ключами нуждается в двух ключах: одном для замыкания, втором — для размыкания замка.

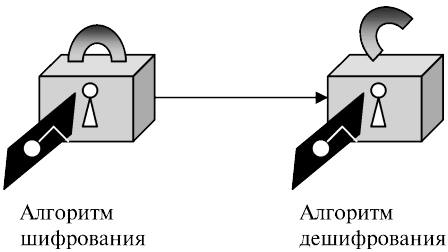


Рис. 3.2. Шифрования симметричными ключами, как замыкание и размыкание замка с тем же самым ключом

Принципы Керкхоффа

Хотя можно предположить, что шифр был бы более безопасен, если бы мы скрывали и алгоритм шифрования/дешифрования, и ключ засекречивания, это не рекомендуется. Согласно принципу Керкхоффа, нужно всегда предполагать, что противник — Ева — знает алгоритм кодирования/дешифрования. Противодействие шифра атаке должно базироваться только на тайне ключа. Другими словами, предполагается, что ключ должен быть настолько труден, что не надо скрывать алгоритм кодирования/дешифрования. Эти принципиальные положения

станут более ясны, когда мы будем изучать современные шифры. Для современных шифров сегодня существует немного алгоритмов. **Множество ключей (Ключевой домен)** для каждого алгоритма, однако, очень большое число, что мешает противнику найти ключ.

Криптоанализ

Криптография — наука и искусство создания секретных кодов, криптоанализ — наука и искусство взламывания этих кодов. В дополнение к изучению методов криптографии мы также должны изучить методы криптоанализа.

Это необходимо не для того, чтобы взламывать коды других людей, а чтобы оценить уязвимые места наших криптографических систем. Изучение криптоанализа помогает нам создавать лучшие секретные коды. Есть четыре общих типа атак криптоанализа, показанные на рис. 3.3. В этой и следующих лекциях мы будем разбирать некоторые из этих атак на конкретные шифры.

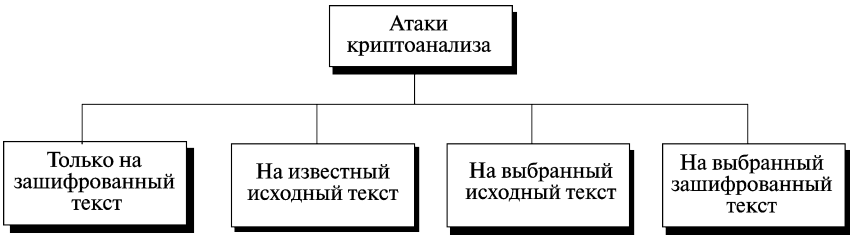


Рис. 3.3. Атаки криптоанализа

Атака только на зашифрованный текст

В **атаке только на зашифрованный текст** Ева имеет доступ лишь к некоторому зашифрованному тексту. Она пробует найти соответствующий ключ и исходный текст. При этом, согласно предположению, Ева знает алгоритм и может перехватить зашифрованный текст. Атака только зашифрованного текста — самая вероятная, потому что Еве для нее нужен только сам текст. Шифр должен серьезно препятствовать этому типу атаки и не позволить дешифрование сообщения противником. Рисунок 3.4 иллюстрирует процесс атаки.

В атаке только на зашифрованный текст могут использоваться различные методы. Мы рассмотрим здесь только некоторые из них.

Атака грубой силы

При методе грубой силы, или **методе исчерпывающего ключевого поиска**, Ева пробует применить все возможные ключи. Мы предполагаем, что Ева знает алгоритм и знает множество ключей (список возможных ключей). При использовании этого метода перехватывается исходный текст и задействуются все возможные ключи, пока не получится исходный текст. Создание атаки грубой силы было в прошлом трудной задачей; сегодня с помощью компьютера это стало проще. Чтобы предотвратить этот тип атаки, число возможных ключей должно быть очень большим.

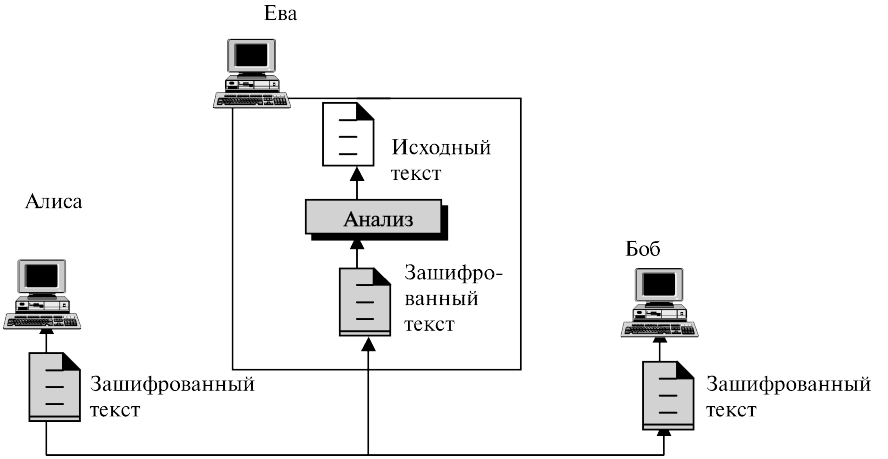


Рис. 3.4. Атака только на зашифрованный текст

Статистическая атака

Криптоаналитик может извлечь выгоду из некоторых свойственных языку исходного текста характеристик, чтобы начать **статистическую атаку**. Например, мы знаем, что буква E — наиболее часто используемая буква в английском тексте. Криптоаналитик находит наиболее часто используемый символ в зашифрованном тексте и принимает, что это соответствующий символ исходного текста — E. После определения нескольких пар аналитик может найти ключ и расшифровать сообщение. Чтобы предотвратить этот тип атаки, шифр должен скрывать характеристики языка.

Атака по образцу

Некоторые шифры скрывают характеристики языка, но создают некоторые образцы в зашифрованном тексте. Криптоаналитик может использовать **атаку по образцу**, чтобы взломать шифр. Поэтому важно использовать шифры, которые сделали бы просматриваемый зашифрованный текст насколько возможно неопределенным, абстрактным.

Атака знания исходного текста

При атаке знания исходного текста Ева имеет доступ к некоторым парам «исходный/зашифрованный текст» в дополнение к перехваченному зашифрованному тексту, который она хочет взломать, как показано на рис. 3.5.

Пары исходного/зашифрованного текста были собраны ранее. Например, Алиса передала секретное сообщение Бобу, но позже открыла содержание сообщения посторонним. Ева хранила и зашифрованный текст, и исходный текст, чтобы использовать их, когда понадобится взломать следующее секретное сообщение от Алисы Бобу, предполагая, что Алиса не изменит свой ключ. Ева собирает знания об отношениях между предыдущей парой, чтобы анализировать текущий зашифрованный текст. Те же самые методы, используемые в атаке только для за-

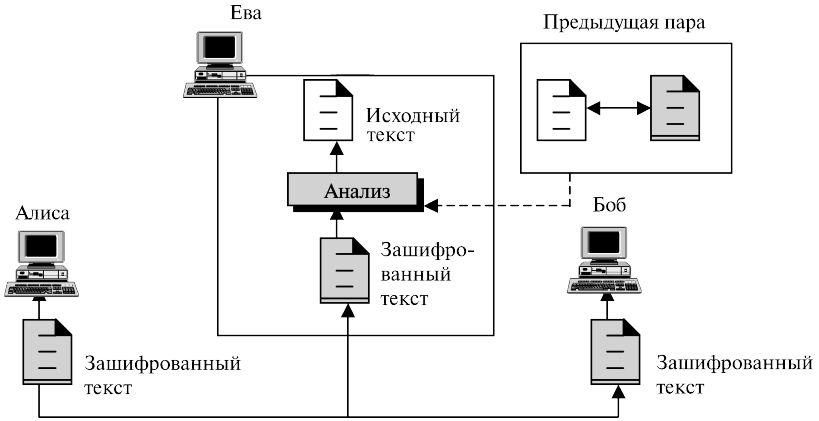


Рис. 3.5. Атака знания исходного текста

шифрованного текста, могут быть применены здесь. Но эту атаку осуществить проще, потому что Ева имеет больше информации для анализа. Однако может случиться, что Алиса изменила свой ключ или не раскрывала содержания любых предыдущих сообщений, — тогда подобная атака станет невозможной.

Атака с выборкой исходного текста

Атака с выборкой исходного текста подобна атаке знания исходного текста, но пары «исходный/зашифрованный текст» были выбраны и изготовлены самим нападавшим. Рисунок 3.6 иллюстрирует этот процесс.

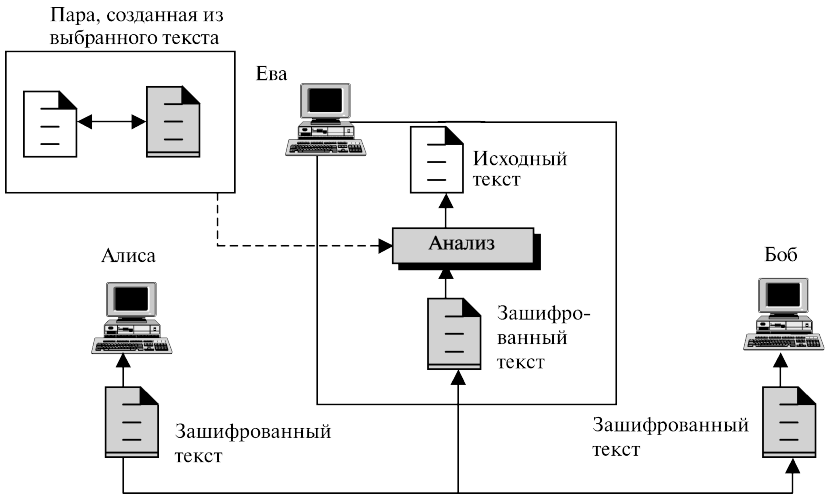


Рис. 3.6. Атака с выборкой исходного текста

Это может случиться, например, если Ева имеет доступ к компьютеру Алисы. Она выбирает некоторый исходный текст и создает с помощью компьютера зашифрованный текст. Конечно, она не имеет ключа, потому что ключ обычно размещается в программном обеспечении, используемом передатчиком. Этот тип атаки намного проще осуществить, но он наименее вероятен, поскольку подразумевает слишком много «если».

Атака с выбором зашифрованного текста

Атака с выбором зашифрованного текста подобна атаке с выборкой исходного текста, за исключением того, что выбирает некоторый зашифрованный текст и расшифровывает его, чтобы сформировать пару «зашифрованный/исходный текст» (это случается, когда Ева имеет доступ к компьютеру Боба). Рисунок 3.7 показывает этот процесс.

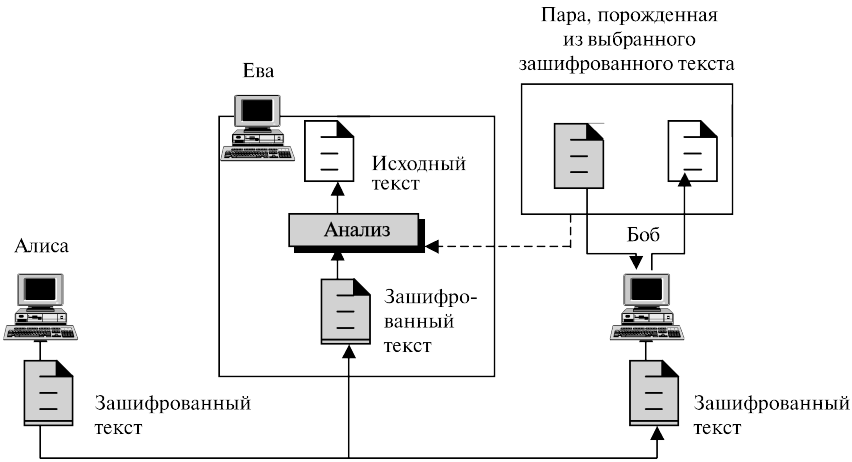


Рис. 3.7. Атака с выбором зашифрованного текста

Категории традиционных шифров

Мы можем разделить традиционные шифры с симметричным ключом на две обширные категории: шифры подстановки и шифры перестановки. В шифре подстановки мы заменяем один символ в зашифрованном тексте на другой символ; в шифре перестановки — меняем местами позиции символов в исходном тексте.

3.2. Шифры подстановки

Шифр подстановки заменяет один символ другим. Если символы в исходном тексте — символы алфавита, то мы заменяем одну букву другой. Например, мы можем заменить букву A буквой D, а букву T — буквой Z. Если символы — циф-

ры (от 0 до 9), мы можем заменить 3 на 7 и 2 на 6. Шифры подстановки могут быть разбиты на две категории: моноалфавитные или многоалфавитные шифры.

Шифр подстановки заменяет один символ другим.

Моноалфавитные шифры

Сначала обсудим шифры подстановки, называемые **моноалфавитными шифрами.** В такой подстановке буква (или символ) в исходном тексте всегда изменяется на одну и ту же букву (или символ) в зашифрованном тексте независимо от его позиции в тексте. Например, если алгоритм определяет, что буква A в исходном тексте меняется на букву D, то при этом каждая буква A изменяется на букву D. Другими словами, буквы в исходном тексте и зашифрованном тексте находятся в отношении «один к одному».

В моноалфавитной подстановке отношения между буквой в исходном тексте и буквой в зашифрованном тексте — «один к одному».

Пример 3.1

Приведенный ниже пример показывает исходный текст и соответствующий ему зашифрованный текст. Мы используем строчные символы, чтобы показать исходный текст, и заглавные буквы (символы верхнего регистра), чтобы получить зашифрованный текст. Шифр моноалфавитный, потому что оба *l* зашифрованы как O.

Исходный текст: hello **Зашифрованный текст: KHOOR**

Пример 3.2

Приведенный ниже пример показывает исходный текст и соответствующий ему зашифрованный текст. Шифр не является моноалфавитным, потому что каждая буква *l* (эль) зашифрована различными символами. Первая буква *l* (эль) зашифрована как N; вторая — как Z.

Исходный текст: hello **Зашифрованный текст: ABNZF**

Аддитивный шифр

Самый простой моноалфавитный шифр — **аддитивный шифр**, его иногда называют **шифром сдвига**, а иногда — **шифром Цезаря,** но термин *аддитивный шифр* лучше показывает его математический смысл. Предположим, что исходный текст состоит из маленьких букв (от a до z) и зашифрованный текст состоит из заглавных букв (от A до Z). Чтобы обеспечить применение математических операций к исходному и зашифрованному текстам, мы присвоим каждой букве числовое значение (для нижнего и верхнего регистра), как это показано на рис. 3.8.

На рисунке 3.8 каждому символу (нижний регистр или верхний регистр) сопоставлено целое число из Z_{26}. Ключ засекречивания между Алисой и Бобом —

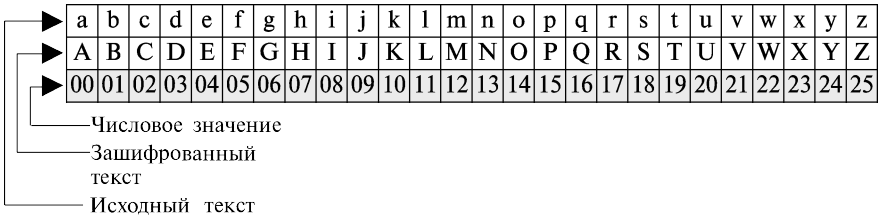


Рис. 3.8. Представление букв исходного текста и зашифрованного текста в Z_{26}

также целое число в Z_n. Алгоритм кодирования прибавляет ключ к символу исходного текста; алгоритм дешифрования вычитает ключ из символа зашифрованного текста. Все операции проводятся в Z_n. Рисунок 3.9 показывает процесс шифрования и дешифрования.

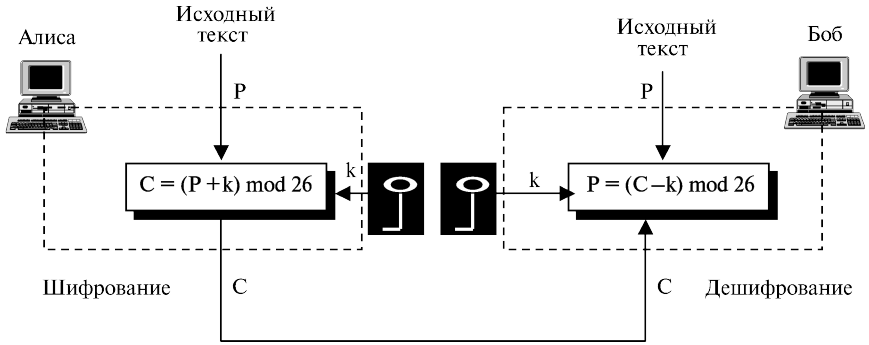


Рис. 3.9. Аддитивный шифр

Мы можем легко показать, что шифрование и дешифрование являются инверсными друг другу, потому что исходный текст, созданный Бобом (P_1), тот же самый, что и тот, который передан Алисой (P).

$$P_1 = (C - k) \bmod 26 = (P + k - k) \bmod 26 = P$$

Когда применяется аддитивный шифр, исходный текст, зашифрованный текст и ключ — целые числа в Z_{26}.

Пример 3.3

Используйте аддитивный шифр с ключом = 15, чтобы зашифровать сообщение «hello».

Решение

Мы применяем алгоритм кодирования к исходному тексту, буква за буквой:

Исходный текст h → 07	Шифрование (07 + 15) mod 26	Шифр. Текст 22 → W
Исходный текст e → 04	Шифрование (04+ 15) mod 26	Шифр. Текст 19 → T
Исходный текст l → 11	Шифрование (11 + 15) mod 26	Шифр. Текст 00 → A
Исходный текст l → 11	Шифрование (11 + 15) mod 26	Шифр. Текст 00 → A
Исходный текст o → 14	Шифрование (14 + 15) mod 26	Шифр. Текст 03 → D

Результат — «WTAAD». Обратите внимание, что шифр моноалфавитный, потому что два отображения одной и той же буквы исходного текста (l) зашифрованы как один и тот же символ (A).

Пример 3.4

Используйте шифр сложения с ключом = 15, чтобы расшифровать сообщение «WTAAD».

Решение

Мы применяем алгоритм дешифрования к исходному тексту буква за буквой:

Шифр. Текст W → 22	Шифрование (22 – 15) mod 26	Исходный текст 07 → h
Шифр. Текст T → 19	Шифрование (19 – 15) mod 26	Исходный текст 04 → e
Шифр. Текст A → 00	Шифрование (00 – 15) mod 26	Исходный текст 11 → *l*
Шифр. Текст A → 00	Шифрование (00 – 15) mod 26	Исходный текст 11 → *l*
Шифр. Текст D → 03	Шифрование (03 – 15) mod 26	Исходный текст 14 → 0

Результат — «hello». Обратите внимание, что операции проводятся по модулю 26 (см. лекцию 2), отрицательный результат должен быть отображен в Z_{26} (например, –15 становится 11).

Шифр сдвига

Исторически аддитивные шифры назывались **шифрами сдвига** — по той причине, что алгоритм шифрования может интерпретироваться как «*клавиша сдвига буквы вниз*», а алгоритм дешифрования может интерпретироваться как «*клавиши сдвига буквы вверх*». Например, если ключ = 15, алгоритм кодирования сдвигает букву на 15 букв вниз (к концу алфавита). Алгоритм дешифрования сдвигает букву на 15 букв вверх (к началу алфавита). Конечно, когда мы достигаем конца или начала алфавита, мы двигаемся по кольцу к началу (объявленные свойства операции по модулю 26).

Шифр Цезаря

Юлий Цезарь использовал аддитивный шифр, чтобы связаться со своими чиновниками. По этой причине аддитивные шифры упоминаются иногда как **шифры Цезаря.** Цезарь для своей связи брал цифру 3.

Аддитивные шифры упоминаются иногда как шифры сдвига или шифры Цезаря.

Криптоанализ

Аддитивные шифры уязвимы к атакам только зашифрованного текста, когда используется исчерпывающий перебор ключей (атака грубой силы). Множество ключей аддитивного шифра очень мало — их только 26. Один из ключей, нулевой, является бесполезным (зашифрованный текст будет просто соответствовать исходному тексту). Следовательно, остается только 25 возможных ключей. Ева может легко начать атаку грубой силы зашифрованного текста.

Пример 3.5

Ева перехватила зашифрованный текст «UVACLYFZLJBYL». Покажите, как она может взломать шифр, используя атаку грубой силы.

Решение

Ева пробует раскрыть текст и последовательно перебирает ключи начиная с первого. С помощью ключа номер 7 она получает осмысленный текст «not very secure» (не очень безопасный).

Зашифрованный текст: UVACLYFZLJBYL

K=1 **Исходный текст**: tubkxeykiaxk

K= 2 **Исходный текст**: styajwdxjhzwj

K= 3 **Исходный текст:** rsxzivewigyvi

K= 4 **Исходный текст:** qrwyhubvhfhuh

K= 5 **Исходный текст:** pqvxgtaugewtg

K= 6 **Исходный текст:** opuwfsztfdvst

K= 7 **Исходный текст:** notverysecure

Буква	Частота	Буква	Частота	Буква	Частота	Буква	Частота
E	12,7	H	6,1	W	2,3	K	0,08
T	9,1	R	6,0	F	2,2	J	0,02
A	8,2	D	4,3	G	2,0	Q	0,01
O	7,5	L	4,0	Y	1,9	X	0,01
I	7,0	C	2,8	P	1,5	Z	0,01
N	6,7	U	2,8	B	1,0		
S	6,3	M	2,4	V			

Статистические атаки

Аддитивные шифры также могут быть объектами статистических атак. Это особенно реально, если противник перехватил длинный зашифрованный текст. Противник может воспользоваться знаниями о частоте употребления символов в

конкретном языке. Таблица 3.1 показывает частоту появления определенных букв для английского текста длиной в 100 символов.

Таблица 3.1 Частота появления букв в английском тексте

Однако информации о частоте единственного символа недостаточно, и это затрудняет анализ шифрованного текста, основанный на анализе частоты появления букв. Весьма желательно знать частоту появления комбинаций символов. Мы должны знать частоту появления в зашифрованном тексте комбинаций с двумя

Диаграмма	TH, HE,IN,ER,AN,RE,ED,ON,ES,ST,EN,AT,TO,NT,HA, ND,OU,EA,NG,AS,OR,TI,IS,ET,IT,AR,TE,SE,HI,OF
Триграмма	THE,ING,AND,HER,ERE,ENT,THA,NTH,WAS,ETH,FOR,DTH

или с тремя символами и сравнивать ее с таковой частотой в языке, на котором написан исходный документ.

Наиболее употребляемые группы с двумя символами (диаграмма (diagrams)) и группы с тремя символами (триграмма (trigrams)) для английского текста показаны в таблице 3.2.

Таблица 3.2 Группы диаграмм и триграмм , основанные на их частоте появления в английском языке

Пример 3.6

Ева перехватила следующий зашифрованный текст. Используя статистическую атаку, найдите исходный текст.

XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX-
LQSVILY-VVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW

Решение

Когда Ева составит таблицу частоты букв в этом зашифрованном тексте, она получит: I = 14, V = 13, S = 12, и так далее. Самый частый символ – I — имеет 14 появлений. Это показывает, что символ I в зашифрованном тексте, вероятно, соответствует символу e в исходном тексте. Тем самым, ключ = 4. Ева расшифровывает текст и получает

the house is now for sale for four million dollars it is worth more hurry before the seller receives more offers
дом теперь продается за четыре миллиона долларов, стоит поспешить, пока продавец не получил больше предложений

Мультипликативные шифры

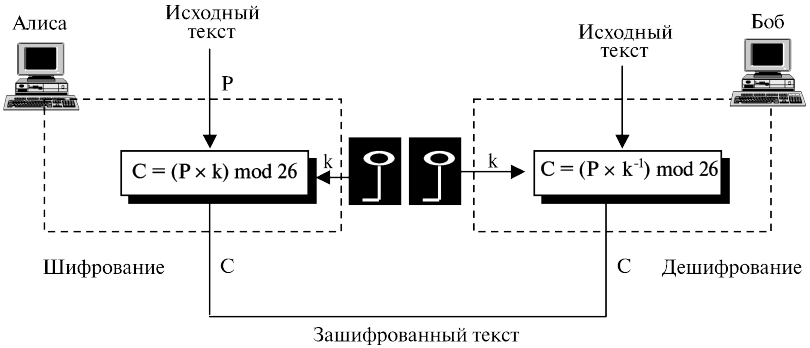


Рис. 3.10. Мультипликативный шифр

В **мультипликативном шифре** алгоритм шифрования применяет умножение исходного текста ключом, а алгоритм дешифрования применяет деление зашифрованного текста ключом, как показано на рис. 3.10. Однако поскольку операции проводятся в Z_{26}, дешифрование здесь означает умножение на мультипликативную инверсию ключа. Обратите внимание, что ключ должен принадлежать набору Z_{n^*} — это гарантирует, что шифрование и дешифрование инверсны друг другу.

В мультипликативном шифре исходный текст и зашифрованный текст — целые числа в Zn; ключ — целое число в Zn*.

Пример 3.7

Каково множество ключей для любого мультипликативного шифра?

Решение

Ключ должен быть в Z_{26^*}. Это множество имеет только 12 элементов: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.

Пример 3.8

Мы используем мультипликативный шифр, чтобы зашифровать сообщение «hello» с ключом 7. Зашифрованный текст «XCZZU».

Исходный текст h → 07	Шифрование (07 × 07) mod 26	Шифр. Текст 23 → X
Исходный текст e → 04	Шифрование (04 × 07) mod 26	Шифр. Текст 02 → C
Исходный текст l → 11	Шифрование (11 × 07) mod 26	Шифр. Текст 25 → Z
Исходный текст l → 11	Шифрование (11 × 07) mod 26	Шифр. Текст 25 → Z
Исходный текст o → 14	Шифрование (14 × 07) mod 26	Шифр. Текст 20 → U

Афинный шифр

Мы можем комбинировать аддитивные и мультипликативные шифры, чтобы получить то, что названо **аффинным шифром** — комбинацией обоих шифров с парой

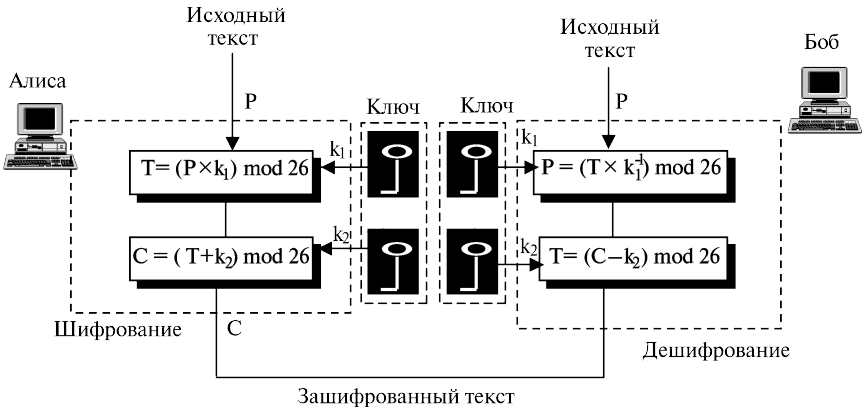


Рис. 3.11. Афинный шифр

ключей. Первый ключ применяется мультипликативным шифром, второй — аддитивным шифром. Рисунок 3.11 доказывает, что афинный шифр — фактически два шифра, применяемые один за другим. Мы могли бы показать только одну комплексную операцию для шифрования или дешифрования, такую, как C = (P × k_1 + k_2) mod 26 и P = ((C – k_2) × k_1^{-1}) mod 26. Однако мы использовали временный результат (T) и указали две отдельных операции, показав тем самым, что всякий раз, когда мы используем комбинацию шифров, нужно убедиться, что каждый из них имеет инверсию на другой стороне линии и что они используются в обратном порядке в шифровании и дешифровании. Если сложение — последняя работа в шифровании, то вычитание должно быть первым в дешифровании.

При аффинном шифре отношение между исходным текстом P и шифрованным текстом C определяется, как это показано ниже.

$$C = (P \times k_1 + k_2) \bmod 26 \qquad P = ((C - k_2) \times k_1^{-1}) \bmod 26$$

где k_1^{-1} мультипликативная инверсия k_1, а $(-k_2)$ – аддитивная инверсия k_2

Пример 3.9

Аффинный шифр применяет пару ключей, в которой первый ключ из Z_{26}^*, а второй — из Z_{26}. Область существования ключей равна 26 × 12 = 312.

Пример 3.10

Используйте аффинный шифр, чтобы зашифровать сообщение «hello» с ключевой парой (7, 2).

Решение

Мы используем 7 для мультипликативного ключа и 2 для аддитивного ключа. Получаем «ZEBBW».

P: h → 07	Шифрование (07 × 07 +2) mod 26	C: 25 → Z
P: e → 04	Шифрование (04 × 07 +2) mod 26	C: 04 → E
P: l → 11	Шифрование (11 × 07+2) mod 26	C: 01 → B
P: l → 11	Шифрование (11 × 07+2) mod 26	C: 01 → B
P: o → 14	Шифрование (14 × 07 +2) mod 26	C: 22 → W

Пример 3.11

Используйте аффинный шифр, чтобы расшифровать сообщение «ZEBBW» с ключевой парой (7, 2) в модуле 26.

Решение

Чтобы найти символы исходного текста, прибавим аддитивную инверсию от $(-2) \equiv 24 \pmod{26}$ к полученному зашифрованному тексту. Потом умножим результат на мультипликативную инверсию от $7^{-1} \equiv 15 \pmod{26}$. Поскольку 2 имеет аддитивную инверсию в Z_{26} и 7 имеет мультипликативную инверсию в Z_{26}^*, исходный текст — точно тот, что мы использовали в примере 3.10.

C: 25 → Z	Дешифрование (07 × 07 − 2) mod 26	P: 07 → h
C: 04 → E	Дешифрование (04 × 07 − 2) mod 26	P: 04 → e
C: 01 → B	Дешифрование (11 × 07 − 2) mod 26	P: 11 → l
C: 01 → B	Дешифрование (11 × 07 − 2) mod 26	P: 11 → l
C: 22 → W	Дешифрование (14 × 07 − 2) mod 26	P: 14 → o

Пример 3.12

Аддитивный шифр — частный случай аффинного шифра, при котором $k_1 = 1$. Мультипликативный шифр — частный случай аффинного шифра, в котором $k_2 = 0$.

Криптоанализ аффинного шифра

Хотя методы грубой силы и статистический метод атаки могут применяться только для зашифрованного текста, попробуем атаку с выборкой исходного текста. Предположим, что Ева перехватывает следующий зашифрованный текст:

PWUFFOGWCHFDWIWEJOUUNJORSMDWRHVCMWJUPVCCG

Ева также очень ненадолго получает доступ к компьютеру Алисы и имеет время, достаточное лишь для того, чтобы напечатать исходный текст с двумя символами: «et». Тогда она пробует зашифровать короткий исходный текст, используя два различных алгоритма, потому что не уверена, какой из них является аффинным шифром.

Для того чтобы найти ключ, Ева задействует следующую стратегию.

$$\begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{pmatrix} 4 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \begin{bmatrix} 22 \\ 2 \end{bmatrix} = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \begin{bmatrix} 22 \\ 2 \end{bmatrix} = \begin{bmatrix} 161 \\ 10 \end{bmatrix} \longrightarrow k_1 = 16 \quad k_2 = 10$$

a. Ева знает, что если первый алгоритм является аффинным, она может составить следующие уравнения, основанные на первом наборе данных:

e → W	04 → 22	$(04 \times k_1 + k_2) = 22 \pmod{26}$
t → C	19 → 02	$(19 \times k_1 + k_2) = 02 \pmod{26}$

Как мы узнали в лекции 2, эти два уравнения сравнения могут быть решены (могут быть найдены значения k_1, и k_2). Однако этот ответ неприемлем, потому что $k_1 = 16$ не может быть первой частью ключа. Его значение, 16, не имеет мультипликативной инверсии в Z_{26}*.

b. Ева теперь пробует использовать результат второго набора данных:

e → W	04 → 22	$(04 \times k_1 + k_2) = 22 \pmod{26}$
t → C	19 → 05	$(19 \times k_1 + k_2) = 05 \pmod{26}$

Квадратная матрица и ее инверсия — те же самые, что и в предыдущем примере. Теперь Ева получает $k_1 = 11$ и $k_2 = 4$, эта пара является приемлемой, потому что k_1 имеет мультипликативную инверсию в Z_{26}*. Она пробует пару ключей (19, 22), которые являются инверсией пары (11, 4), и расшифровывает сообщение. Исходный текст

Best time of the year is spring when flower bloom
Самое лучшее время года — весна, когда цветут цветы

Моноалфавитный шифр подстановки

Исходный текст

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
N	O	A	T	R	B	E	C	F	U	X	D	Q	G	Y	L	K	H	V	I	J	M	P	Z	S	W

Зашифрованный текст

Поскольку аддитивные, мультипликативные и аффинные шифры имеют малое множество ключей, они очень уязвимы к атаке грубой силы. Алиса и Боб согласовали единственный ключ, который они используют, чтобы зашифровать каждую букву в исходном тексте или расшифровать каждую букву в зашифрованном тексте. Другими словами, ключ независим от передаваемых букв.

Лучшее решение состоит в том, чтобы создать отображение каждой буквы исходного текста на соответствующий символ зашифрованного текста. Алиса и Боб могут договориться об отображении для каждой буквы и записать его в виде таблицы. Рисунок 3.12 показывает пример такого отображения.

Рис. 3.12. Пример ключа для моноалфавитного шифра подстановки

Пример 3.13

Мы можем использовать ключ, показанный на рисунке 3.12, чтобы зашифровать сообщение

This message is easy to encrypt but hard to find the key
(это сообщение просто зашифровать, но трудно найти ключ)

Зашифрованное сообщение имеет вид

ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS

Криптоанализ

Размер ключевого пространства для моноалфавитного шифра подстановки — число перестановок из 26, т.е. 26! (почти 4×10^{26}). Это делает атаку грубой силы чрезвычайно трудной для Евы, даже если она использует мощный компьютер. Однако она может применить статистическую атаку, основанную на частоте символов. Шифр не изменяет частоту употребления символов.

Моноалфавитные шифры не изменяют частоту появления символов в зашифрованном тексте, что делает шифры уязвимыми к статистической атаке.

Многоалфавитные шифры

В многоалфавитной подстановке каждое появление символа может иметь различную замену. Отношения между символом в исходном тексте и символом в зашифрованном тексте — «один ко многим». Например, «a» может быть зашифровано как «D» в начале текста, но как «N» — в середине. Многоалфавитные шифры имеют преимущество: они скрывают частоту появления символа основного языка. Ева не может использовать статистическую частоту отдельного символа, чтобы взломать зашифрованный текст.

Чтобы создать многоалфавитный шифр, мы должны сделать каждый символ зашифрованного текста зависящим от соответствующего символа исходного текста и позиции символа исходного текста в сообщении. Это подразумевает, что наш ключ должен быть потоком подключей, в которых каждый подключ так или иначе зависит от позиции символа исходного текста, который используется для выбора подключа шифрования. Другими словами, мы должны иметь ключевой поток $k = (k_1, k_2, k_3.....)$, в котором k_i применяется, чтобы зашифровать i-тый символ в исходном тексте и создать i-тый символ в зашифрованном тексте.

Автоключевой шифр

Чтобы понять зависимость ключа от позиции, обсудим простой многоалфавитный шифр, названный «автоключевым». В этом шифре ключ — поток подключей, в котором каждый подключ используется, чтобы зашифровать соответствующий символ в исходном тексте. Первый подключ — определенное заранее значение, тайно согласованное Алисой и Бобом. Второй подключ — значение первого

символа исходного текста (между 0 и 25). Третий — значение второго символа исходного текста. И так далее.

$P = P_1\ P_2\ P_{3.....}$ $C = C_1\ C_2\ C_{3.....}$ $K = (k_1 P_1, P_2, P_{3,.....})$

Шифрование $C_i = (P_i + k_i) \bmod 26$ Дешифрование $P_i = (C_i - k_i) \bmod 26$

Название шифра, *автоключевой*, подразумевает, что подключи создаются автоматически в зависимости от символов шифра исходного текста в процессе шифрования.

Пример 3.14

Предположим, что Алиса и Боб согласились использовать автоключевой шифр с начальным ключевым значением $k_1 = 12$. Теперь Алиса хочет передать Бобу сообщение «Attack is today» («Атака — сегодня»). Шифрование проводится символ за символом. Каждый символ в исходном тексте сначала заменяется его значением целого числа, как показано на рис. 3.8, первый подключ прибавляется, чтобы создать первый символ зашифрованного текста. Остальная часть ключа создается по мере чтения символов исходного текста. Обратите внимание, что шифр является многоалфавитным, потому что эти три появления «a» в исходном тексте зашифрованы различно. Три возникновения «t» также зашифрованы различно.

Исходный текст:	a	t	t	a	c	k	i	s	t	o	d	a	y
Значения P:	00	19	19	00	02	10	08	18	19	14	03	00	24
Поток ключей	*12*	*00*	*19*	*19*	*00*	*02*	*10*	08	*18*	*19*	*14*	*03*	*00*
Значения C:	12	19	12	19	02	12	18	00	11	7	17	03	24
Зашифрованный Текст:	**M**	**T**	**M**	**T**	**C**	**M**	**S**	**A**	**L**	**H**	**R**	**D**	**Y**

Криптоанализ

Автоключевой шифр действительно скрывает статистику частоты отдельного символа. Однако он так же уязвим при атаке с помощью грубой силы, как и аддитивный шифр. Первый подключ может быть только одним из 25 значений (1–25).

Секретный ключ =

L	G	D	B	A
Q	M	H	E	C
U	R	N	I/J	F
X	V	S	O	K
Z	Y	W	T	P

Мы нуждаемся в многоалфавитных шифрах, которые не только скрывают характеристики языка, но и имеют большие множества ключей.

Шифр плейфеера

[1] плейфеер (playfair) – играющий по правилам.

Другой пример многоалфавитного шифра — Шифр плейфеера[1], использовавшийся британской армией в течение Первой мировой войны. Ключ засекречивания в этом шифре сделан из 25 букв алфавита, размещенных в матрице 5 × 5 (буквы I и J рассматриваются при шифровании как одинаковые). С помощью различных соглашений о размещении букв в матрице можно создать много различных ключей засекречивания. Одно из возможных соглашений показано на рисунке 3.13.

Рис. 3.13. Пример секретного ключа плейфеера

Перед шифрованием исходный текст разбивается на пары; если два буквы пары одинаковые, то, чтобы отделить их, вставляется фиктивная буква. После вставки фиктивных букв, если число символов в исходном тексте нечетно, в конце добавляется один дополнительный фиктивный символ, чтобы сделать число символов четным.

Шифр использует три правила для шифрования:

a. если эти две буквы-пары расположены в одной и той же строке таблицы ключа засекречивания, соответствующий зашифрованный символ для каждой буквы — следующий символ справа в той же самой строке (с возвращением к началу строки; если символ исходного текста — последний символ в строке);
b. если эти две буквы-пары расположены в одном и том же столбце таблицы ключа засекречивания, соответствующий зашифрованный символ для каждой буквы — символ ниже этого в том же самом столбце (с возвращением к началу столбца; если символ исходного текста — последний символ в столбце);
c. если эти две буквы-пары не находятся в одной строке или столбце таблицы засекречивания, соответствующий зашифрованный символ для каждой буквы — символ, который находится в его собственной строке, но в том же самом столбце, что и другой символ.

Шифр плейфеера соответствует нашим критериям для многоалфавитного шифра. Ключ — поток подключей, в котором они создаются по два одновременно. В шифре плейфеера поток ключей и поток шифра — одни и те же. Это означает, что вышеупомянутые правила можно представить как правила для создания потока ключей. Алгоритм кодирования берет пару символов из исходного текста и создает пару подключей, следуя указанным правилам. Мы можем сказать, что поток ключей зависит от позиции символа в исходном тексте. Зависимость от позиции имеет здесь различную интерпретацию: подключ для каждого символа исходного текста зависит от следующего или предыдущего «соседа». Рассматривая шифр плейфеера, таким образом, можно сказать, что зашифрованный текст — это фактически поток ключей.

$$P = P_1\ P_2\ P_{3.....} \qquad C = C_1\ C_2\ C_{3.....} \qquad k = [(k_1,k_2), (k_3,k_4),.....]$$

Шифрование $C_i = k_i$ Дешифрование $P_i = k_i$

Пример 3.15

Пусть нам надо зашифровать исходный текст «hello», использующий ключи на рис. 3.13. Когда мы группируем буквы по парам, мы получаем «he», *ll*,o». Мы должны вставить *x* между двумя *l* (эль), после чего получим «he, *lx*, *l*o». Мы имеем

he → EC lx → QZ lo → BX

Исходный текст: hello Зашифрованный текст: ECQZBX

Мы можем видеть из этого примера, что наш шифр — фактически многоалфавитный шифр: два появления l (эль) зашифрованы как «Q» и «B».

Криптоанализ шифра плейфеера

Очевидно, атака грубой силы шифра плейфеера очень трудна. Размер домена — 25! (факториал 25). Кроме того, шифровка скрывает частоту отдельных букв. Однако частоты двухбуквенных комбинаций (диаграмм) сохранены (до некоторой степени из-за вставки наполнителя), так что криптоаналитик может использовать атаку только для зашифрованного текста, основанную на испытании частоты диаграмм, чтобы найти ключ.

Шифр Виженера

Один интересный вид многоалфавитного шифра был создан Блезом де Виженером, французским математиком шестнадцатого столетия. Шифр Виженера применяет различную стратегию создания потока ключей. Поток ключей — повторение начального потока ключа засекречивания длины m, где мы имеем $1 < m < 26$. Шифр может быть описан следующим образом: (k_1, k_2, ..., k_m) — первоначальный ключ засекречивания, согласованный Алисой и Бобом.

Исходный текст	s	h	e	i	s	l	i	s	t	e	n	i	n	g
Значения P	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Поток ключей	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***
Значения C	**07**	**07**	**22**	**10**	**18**	**22**	**23**	**18**	**11**	**6**	**13**	**19**	**02**	**06**
Шифрованный текст	**H**	**H**	**W**	**K**	**S**	**W**	**X**	**S**	**L**	**G**	**N**	**T**	**C**	**G**

$P = P_1\ P_2\ P_{3.....}$ $C = C_1\ C_2\ C_{3.....}$ $k = [(k_1,k_2), (k_3,k_4),.....]$

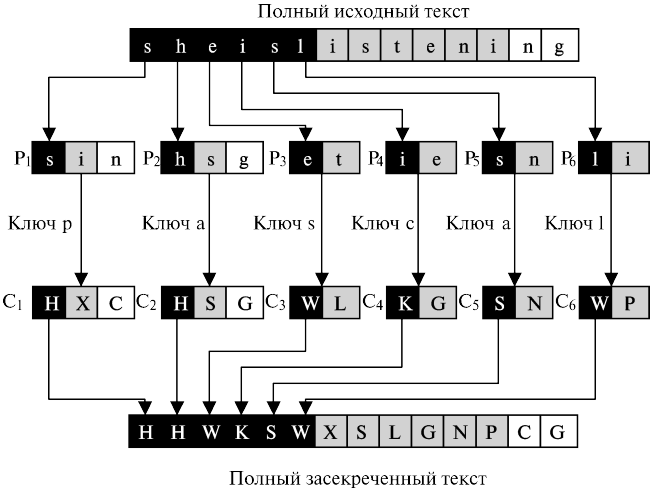
Шифрование $C_i = k_i$ Дешифрование $P_i = k_i$

Одно важное отличие между шифром Виженера и другими двумя многоалфавитными шифрами, которые мы рассмотрели: поток ключей Виженера не зависит от символов исходного текста; он зависит только от позиции символа в исходном тексте. Другими словами, поток ключей может быть создан без знания сути исходного текста.

Пример 3.16

Посмотрим, как мы можем зашифровать сообщение «She is listening (Она слушает)», используя ключевое слово на 6 символов «PASCAL». Начальный поток ключей — это (15, 0, 18, 2, 0, 11). Поток ключей — повторение этого начального потока ключей (столько раз, сколько необходимо).

Пример 3.17



Шифр Виженера может восприниматься как комбинации аддитивных шифров. Рисунок 3.14 показывает, что исходный текст предыдущего примера можно рассматривать как состоящий из нескольких частей по шесть элементов в каждом

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

(хотя в одном не хватило букв исходного текста), где каждый из элементов зашифрован отдельно. Рисунок поможет нам позже понять криптоанализ шифров Виженера. Имеется m частей исходного текста, каждый зашифрован различным ключом, чтобы разделить зашифрованный текст *на m*. частей.

Пример 3.18

Разобрав пример 3.18, мы убедимся, что аддитивный шифр — частный случай шифра Виженера, в котором m = 1.

Список Виженера

Другой способ рассмотрения шифров Виженера — с помощью того, что названо **списком Виженера (Vigenere tableau)** и показано в таблице 3.3.

Рис. 3.14. Шифр Виженера как комбинация аддитивных шифров

Таблица 3.3. Список Виженера

Первая строка показывает символы исходного текста, который будет зашифрован. Первая колонка содержит столбец символов, которые используются ключом. Остальная часть таблицы показывает символы зашифрованного текста. Чтобы найти зашифрованный текст для исходного текста «she listening», применяя слово «PASCAL» как ключ, мы можем найти «s» в первой строке, «P» в первом столбце, на пересечении строки и столбца — символ из зашифрованного текста «H». Находим «h» в первой строке и «A» во втором столбце, на пересечении строки и столбца — символ «H» из зашифрованного текста. И повторяем те же действия, пока все символы зашифрованного текста не будут найдены.

Криптоанализ шифра Виженера

Шифры Виженера, подобно всем многоалфавитным шифрам, не сохраняют частоту символов. Однако Ева может использовать некоторые методы для того, чтобы расшифровать перехваченный зашифрованный текст. Криптоанализ в данном случае состоит из двух частей: находят длину ключа и потом непосредственно находят ключ.

1. Были изобретены несколько методов, чтобы найти длину ключа. Один метод рассмотрим ниже. В так называемом **тесте Казиского**[1] **(Kasiski)** криптоаналитик в зашифрованном тексте ищет повторные сегменты по крайней мере из трех символов. Предположим, что найдены два сегмента, и расстояние между ними – d. Криптоаналитик предполагает, что d|m, где *m* — длина ключа. Если можно найти больше повторных сегментов с расстоянием d_1, d_2, ..., d_n, тогда НОД*(*d_1, d_2, ..., d_n....)/ *m*. Это предположение логично, потому что если два символа одинаковы и $k \times m$. *(*k = 1, 2...) — символы, выделенные в исходном тексте, то одинаковы и $k \times m$. символы, выделенные в зашифрованном тексте. Криптоаналитик использует сегменты по крайней мере из трех символов, чтобы избежать случаев, где символы имеют один и тот же ключ. Пример 3.20 может помочь нам понять эти рассуждения.

[1] Ф. Казиский – немецкий криптолог, который в 1860 году предложил тест для раскрытия криптограмм.

2. После того как длина ключа была найдена, криптоаналитик использует идею, показанную в примере 3.18. Здесь зашифрованный текст делится на *m* различных частей и применяется метод, используемый в криптоанализе аддитивного шифра, включая атаку частоты. Каждая часть зашифрованного текста может быть расшифрована и соединена с другими, чтобы создать целый исходный текст, другие слова. Весь зашифрованный текст не сохраняет частоту отдельной буквы исходного текста, но каждая часть делает это.

Комбинация	Первое расстояние	Второе расстояние	Разность
JSU	68	168	100
SUM	69	117	48
VWV	72	132	60
MPH	119	127	8

Пример 3.19

C_1	LWGW	CRAO	KTEP	GTQC	TJVU	EGVG	UQGE	CVPR	PVJG	TJEU	GCJG
P_1	*jueu*	*apym*	*ircn*	*eroa*	*rhts*	*thih*	*ytra*	*hcie*	*ixst*	*hcar*	*rehe*
C_2	IGGG	QHGW	GKVC	TSOS	QSWV	WFVY	SHSV	FSHZ	HWWF	SOHC	OQSL
P_2	*usss*	*ctsl*	*swho*	*feae*	*ceih*	*cete*	*soec*	*atnp*	*nkhe*	*rhck*	*ecex*
C_3	OFDH	URWQ	ZKLZ	HGVV	LUVL	SZWH	WKHF	DUKD	HVIW	HUHF	WLUW
P_3	*lcae*	*rotn*	*whiw*	*edss*	*irsi*	*irh*	*eteh*	*retl*	*tiid*	*eatr*	*airt*
C_4	MEVH	CWIL	EMWV	VXGE	TMEX	LMLC	XVEL	GMIM	BWXL	GEVV	ITX
P_4	*iard*	*yseh*	*aisr*	*rtca*	*piaf*	*pwte*	*thec*	*arha*	*esft*	*erec*	*tpt*

Предположим, что мы перехватили следующий зашифрованный текст:

LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETKKMEV-
LWPCZVGTHVTSGXQOVGCSVETQLTJSUMVWVEUVLXEWSLGFZ-
MVVWLGYHCUSWXQHKVGSHEEVFLCFDGVSUMPHKIRZDMPH-
HBVWVWJWIXGFWLTSHGJOUEEHHVUCFVGOWICQLIJSUXGLW

Тест Казиского на повторение сегментов из трех символов приводит к результатам, показанным в таблице 3.14.

Таблица 3.4. Тест Казиского для примера 3.19

Наибольший делитель — 4, что означает длину ключа, пропорциональную 4. Сначала пробуем $m = 4$. Делим зашифрованный текст на четыре части. Часть C_1 состоит из символов 1, 5, 9...; часть C_2 состоит из символов 2, 6, 10..., и так далее. Используем статистическую атаку каждой части отдельно. Перебираем расшифровывающиеся части по одному символу одновременно, чтобы получить целый исходный текст.

Если исходный текст не имеет смысла, попробуем с другим *m*.

В рассматриваемом случае исходный текст имеет смысл (читаем по столбцам).

[1] Лестер С. Хилл — американский криптолог, предложивший в 1929 году многоалфавитный шифр.

Julius Caesar used a cryptosystem in his wars, which is now referred to as Caesar chipper. It is an additive chipper with the key set to three. Each character in the plaintext is shift three characters to create ciphertext.

Перевод этого текста приведен ниже.

Юлий Цезарь использовал в своих войнах криптографическую систему, которая упоминается теперь как шифр Цезаря. Это аддитивный шифр с ключом,

$$\begin{pmatrix} k_{11} & k_{12} & \ldots\ldots & k_{1m} \\ k_{21} & k_{22} & \ldots\ldots & k_{2m} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ k_{m1} & k_{m2} & \ldots\ldots & k_{mm} \end{pmatrix}$$

установленным на три. Каждый символ в исходном тексте сдвинут на три символа, чтобы создать зашифрованный текст.

Шифр Хилла

Другой интересный пример многоалфавитного шифра — **шифр Хилла**, изобретенный Лестером С. Хиллом[1]. В отличие от других многоалфавитных шифров, которые мы уже рассмотрели, здесь исходный текст разделен на блоки равного размера. Блоки зашифрованы по одному таким способом, что каждый символ в блоке вносит вклад в шифрование других символов в блоке. По этой причине шифр Хилла принадлежит к категории шифров, названных *блочными шифрами*. Другие шифры, которые мы изучали до сих пор, принадлежат к категории, называемой *шифры потока*. Отличие между шифрами блока и шифрами потока обсуждаются в конце этой лекции.

В шифре Хилла ключ — квадратная матрица размера $m \times m$., в котором m. является размером блока. Если мы вызываем ключевую матрицу **K**, то каждый элемент $k_{i,j}$ определяется матрицей, как показано на рис. 3.15.

Рис. 3.15. Ключ в шифре Хилла

Покажем, как получается один блок зашифрованного текста. Обозначим m символов блоков исходного текста P_1, P_2 ..., P_m, соответствующие символы в блоках зашифрованного текста будут C_1, C_2 , ..., C_m. Тогда мы имеем

$$C_1 = P_1k_{11} + P_2k_{21} + ...+ P_mk_{m1}$$
$$C_2 = P_1k_{12} + P_2k_{22} + ...+ P_mk_{m2}$$
$$.....$$
$$C_m = P_1k_{1m} + P_2k_{2m} + ...+ P_mk_{mm}$$

$$\underset{C}{\begin{pmatrix} 14 & 07 & 10 & 13 \\ 08 & 07 & 06 & 11 \\ 11 & 08 & 18 & 18 \end{pmatrix}} = \underset{P}{\begin{pmatrix} 02 & 14 & 03 & 04 \\ 08 & 18 & 17 & 04 \\ 00 & 03 & 24 & 25 \end{pmatrix}} \underset{K}{\begin{pmatrix} 09 & 07 & 11 & 13 \\ 04 & 07 & 05 & 06 \\ 02 & 21 & 14 & 09 \\ 03 & 23 & 21 & 08 \end{pmatrix}}$$

а) Шифрование

$$\underset{P}{\begin{pmatrix} 02 & 14 & 03 & 04 \\ 08 & 18 & 17 & 04 \\ 00 & 03 & 24 & 25 \end{pmatrix}} = \underset{C}{\begin{pmatrix} 14 & 07 & 10 & 13 \\ 08 & 07 & 06 & 11 \\ 11 & 08 & 18 & 18 \end{pmatrix}} \underset{K^{-1}}{\begin{pmatrix} 02 & 15 & 22 & 03 \\ 15 & 00 & 19 & 03 \\ 09 & 09 & 03 & 11 \\ 17 & 00 & 04 & 07 \end{pmatrix}}$$

б) Дешифрование

Уравнения показывают, что каждый символ зашифрованного текста, такой, как C_1, зависит от символов всего исходного текста в блоке (P_1, P_2,..., P_m). Однако мы должны знать, что не все квадратные матрицы имеют мультипликативные инверсии в Z_{26}, так что Алиса и Боб должны быть осторожны в выборе ключа. Боб не сможет расшифровать зашифрованный текст, передаваемый Алисой, если матрица не имеет мультипликативной инверсии.

Ключевая матрица в шифре Хилла должна иметь мультипликативную инверсию.

Пример 3.20

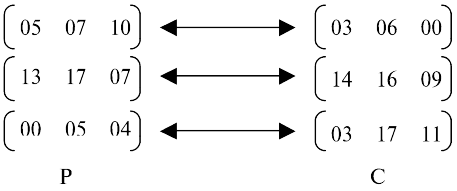
Использование матриц позволяет Алисе зашифровать весь исходный текст. В этом случае исходный текст – $l \times m$ — матрица, в которой l является номером блоков. Например, исходный текст «code is ready» («код готов»), может быть представлен как матрица 3 при добавлении дополнительного фиктивного символа «z» к последнему блоку и удалении пробелов; зашифрованный текст выглядит как «OHKNIHGKLISS». Боб может расшифровать сообщение, применяя инверсную матрицу-ключ. Шифрование и дешифрование показано на рис. 3.16.

Рис.3.16. Пример 3.20

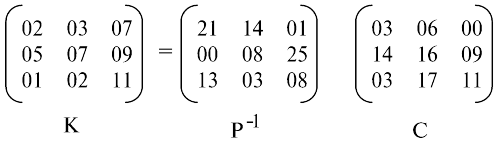
Криптоанализ шифров Хилла

Криптоанализ только для зашифрованного шифрами Хилла текста труден. Во-первых, атака грубой силы при шифре Хилла чрезвычайно сложна, потому что матрица-ключ — $m \times m$. Каждый вход может иметь одно из 26 значений. Во-первых, это означает размер ключа $26^{m \times m}$. Однако, не все матрицы имеют мультипликативную инверсию. Поэтому область существования ключей все же не такая огромная.

Во-вторых, шифры Хилла не сохраняют статистику обычного текста. Ева не может провести анализ частоты употребления отдельной буквы, двух или трех букв. Анализ частоты слов размера m мог бы сработать, но очень редко исходный текст имеет много одинаковых строк размера m. Ева, однако, может



провести атаку на шифр, используя метод знания исходного текста, если она знает значение *m* и знает пары «исходный текст/зашифрованный текст», по крайней мере *m* блоков. Блоки могут принадлежать тому же самому сообщению или различным сообщениям, но должны быть различны. Ева может создать две *m* × *m*. матрицы, P (обычный текст) и C (зашифрованный текст), в котором соответствующие строки представляют известные пары обычного/за-



шифрованного текста. Поскольку **C** = **PK**, Ева может использовать отношения $\mathbf{K} = \mathbf{CP}^{-1}$, чтобы найти ключ, если P является обратимым. Если P не является обратимым, то Ева должна задействовать различные наборы *m* пар обычного/зашифрованного текста.

Если Ева не знает значение *m*, она может попробовать различные значения при условии, что *m* не является очень большим.

Пример 3.21

Предположим, что Ева знает, что *m* = 3. Она перехватила три пары блока исходного/зашифрованного текста (не обязательно из того же самого сообщения), как показано на рис. 3.17.

Рис. 3.17. Пример 3.22, формирования шифра зашифрованного текста

Она составляет матрицы P и C из этих пар. Поскольку в данном случае матрица P обратима, она инвертирует эту матрицу и умножает ее на C, что дает матрицу ключей K, как это показано на рис. 3.18.

Рис. 3.18. Пример 3.22, поиска ключа

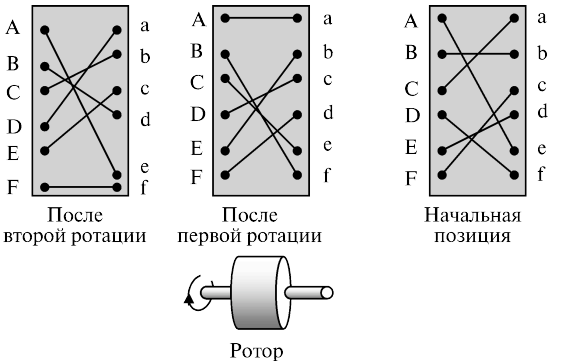
Теперь она имеет ключ и может взломать любой шифрованный текст, где применен этот ключ.

Одноразовый блокнот

Одна из целей криптографии — идеальная секретность. Исследования Шеннона показали, что идеальная секретность может быть достигнута, если символы исходного текста зашифрованы с помощью ключа, выбранного случайно из некоторой области ключей. Например, аддитивный шифр может быть легко взломан,

потому что используется один и тот же ключ. Но даже и этот шифр может быть идеальным, когда ключ, который применяется для шифрования каждого символа, выбран случайно из множества ключей (00, 01, 02.... 25): если первый символ зашифрован с помощью ключа 04, второй символ — с помощью ключа 02, третий — с помощью ключа 21, и так далее. Атака «только для зашифрованного текста» становится невозможна. Если передатчик изменяет ключ, используя каждый раз иную случайную последовательность целых чисел, другие типы атак также будут невозможны.

Эта идея реализована в шифре, который называется **одноразовым блокнотом**. Его изобрел американский инженер Вернам. В этом шифре ключ имеет ту же самую длину, что и исходный текст, и выбран совершенно случайно.



Одноразовый блокнот — идеальный шифр, но его почти невозможно реализовать коммерчески. Если ключ каждый раз генерируется заново, как Алиса может сообщать Бобу новый ключ? Для этого каждый раз нужно передавать сообщение. Однако есть некоторые случаи, когда возможно использование одноразового блокнота. Например, если президент страны должен передать полностью секретное сообщение президенту другой страны, он может перед посылкой сообщения передать с помощью доверенного посланника случайный ключ.

Некоторые вопросы изменения шифра одноразового блокнота обсуждаются в дальнейших лекциях, когда будет рассматриваться введение в современную криптографию.

Роторный шифр

Хотя шифры одноразового блокнота не применяются на практике, один шаг от него к более защищенному шифру — роторный шифр. Он возвращается к идее моноалфавитной подстановки, но меняет принцип отображения исходного текс-

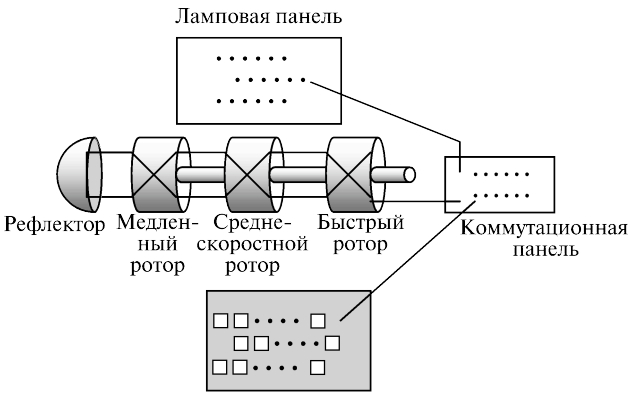
1 Щетки – специальный вид контакта, который обеспечивает соединение подвижной и неподвижной частей прибора и представляет собой металлическую щетку, скользящую по поверхности подвижной части прибора *(прим. ред.)*.

та в символы зашифрованного текста для каждого символа исходного текста. Рисунок 3.19 показывает упрощенный пример роторного шифра.

Рис. 3.19. Роторный шифр

Ротор, показанный на рис. 3.19, применен только для 6 букв, но реальные роторы используют 26 букв. Ротор постоянно связывает символы исходного и зашифрованного текстов, но подключение обеспечивается щетками[1]. Обратите внимание, что соединение символов исходного и зашифрованного текстов показано так, как если бы ротор был прозрачен и можно было видеть внутреннюю часть.

Начальная установка (позиция) ротора — ключ засекречивания между Алисой и Бобом — это зашифрованный первый символ исходного текста. Используется начальная установка и второй символ зашифрован после того, как проведе-



но первое вращение (на рис. 3.19 — это поворот на 1/6 круга, на реальной установке — поворот на 1/26), и так далее.

Слово с тремя буквами, такими, как «bee», зашифровано как «BAA», если ротор неподвижен (моноалфавитный шифр подстановки), но оно будет зашифровано как «BCA», если он вращается (роторный шифр). Это показывает, что роторный шифр — многоалфавитный шифр, потому что два появления того же самого символа исходного текста зашифрованы как различные символы.

Роторный шифр является стойким к атаке грубой силы, как моноалфавитный шифр подстановки, потому что Ева должна найти первое множество отображений среди возможных 26! (факториал). Роторный шифр является намного более стойким к статистической атаке, чем моноалфавитный шифр подстановки, потому что в нем не сохраняется частота употребления буквы.

Машина «Энигма»

Машина «Энигма» была первоначально изобретена в Сербии, но была изменена специалистами немецкой армии и интенсивно использовалась в течение

Второй Мировой Войны. Машина базировалась на принципе шифров ротора. Рисунок 3.20 показывает упрощенную схему построения машины.

Рис. 3.20. Примерное построение Машины Энгима

Ниже перечислены главные компоненты машины.

1. Клавиатура с 26-ю ключами, используемыми для того, чтобы вводить исходный текст при шифровании, и для того, чтобы вводить зашифрованный текст при расшифровке.
2. Ламповая панель с 26-ю лампами, которая показывает символы зашифрованного текста при шифровании и символы исходного текста при дешифровании.
3. Коммутационная панель с 26-ю штепселями, вручную подключенными 13-ю проводами. Конфигурация изменяется каждый день, чтобы обеспечить различное скрэмблирование.
4. Три замонтированных ротора, такие же, как рассмотренные в предыдущей секции. Эти три ротора выбираются ежедневно из пяти доступных роторов. Быстрый ротор вращается на 1/26 поворота при каждом символе, введенном с помощью клавиатуры. Средний ротор делает 1/26 поворота при каждом полном повороте быстрого ротора. Медленный ротор делает 1/26 поворота для каждого законченного поворота среднего ротора.
5. Отражатель, который является постоянным и предварительно замонтированным.

Кодовая книга — справочник шифров

Чтобы использовать «Энигму», была издана кодовая книга, которая в течение каждого дня дает несколько параметров настройки, включая:

a. три ротора, которые должны быть выбраны из пяти доступных;
b. порядок, в котором эти роторы должны быть установлены;
c. параметры установок для коммутационной панели;
d. код с тремя буквами дня.

Процедура шифровавшая Сообщения

Чтобы зашифровать сообщение, оператор должен последовательно сделать шаги, перечисленные ниже:

1. установить стартовую позицию роторов согласно коду дня. Например, если код был «HUA», роторы должны быть инициализированы на “H”, “U”и “A” соответственно;
2. выбрать случайный код с тремя буквами, например ACF. Зашифровать текст «ACFACF» (повторный код), используя начальную установку роторов шага 1. Например, предположим, что зашифрованный код — «OPNABT»;
3. установить стартовые позиции роторов к OPN (половина зашифрованного кода);
4. добавить зашифрованные шесть букв, полученных на шаге 2 («OPNABT’), в конец к начальному сообщению;

5. зашифровать сообщение, включая код с 6-ю буквами. Передать зашифрованное сообщение.

Процедура для расшифровки сообщения

Чтобы расшифровывать сообщение, оператор должен сделать следующие шаги:

1. получить сообщение и отделить первые шесть букв;
2. установить стартовую позицию роторов согласно коду дня;
3. расшифровать первые шесть букв писем, используя начальную установку шага 2;
4. установить позиции роторов на первую половину расшифрованного кода;
5. Расшифровать сообщение (без первых шести букв).

Криптоанализ

Мы знаем, что «Энигма» во время войны была взломана, хотя германская армия и остальная часть мира не знала об этом факте еще несколько десятилетий после того. Вопрос: как такой сложный шифр был атакован? Немецкий язык весьма сложен. Союзники, так или иначе, получили некоторые копии машин. Следующим шагом был поиск параметров установки в течение каждого дня и кода, передаваемого для инициализации роторов для каждого сообщения. Изобретение первого компьютера помогло союзникам преодолеть эти трудности. Подробное изображение машины «Энигма» и ее криптоанализ может быть найден на сайтах, посвященных этой машине.

3.3. Шифры перестановки

Шифр перестановки не заменяет одним символом другой, вместо этого он изменяет местоположение символов. Символ в первой позиции исходного текста может появиться в десятой позиции зашифрованного текста. Символ, который находится в восьмой позиции исходного текста, может появиться в первой позиции зашифрованного текста. Другими словами, шифр перестановки ставит в другом порядке (перемещает) символы.

Шифр перестановки меняет порядок следования символов.

Шифры перестановки без использования ключа

Простые шифры перестановки, которые применялись в прошлом, не использовали ключ. Есть два метода для перестановки символов. В первом методе текст записывается в таблице столбец за столбцом и затем передаётся строка за строкой. Во втором методе текст записан в таблице строка за строкой и затем передаётся столбец за столбцом.

Пример 3.22

Хороший пример шифра без использования ключа — шифр изгороди (rail fence cipher). В этом шифре исходный текст размещен на двух линиях как зигзагообразный шаблон (что может рассматриваться как столбец за столбцом таблицы, которая имеет две строки); зашифрованный текст составляется при чтении шаблона строка за строкой. Например, чтобы передать сообщение «Meet me at the park» («Встречай меня в парке»), Алиса пишет Бобу:

Алиса создает зашифрованный текст «MEMATEAKETETHPR», посылая первую строку, сопровождаемую второй строкой. Боб получает зашифрованный текст и разделяет его пополам (в этом случае вторая половина имеет на один символ меньше). Первая половина формы — первая строка; вторая половина — вторая строка. Боб читает результат по зигзагу. Поскольку нет никакого ключа и номер строк установлен (2), криптоанализ зашифрованного текста был бы очень прост для Евы. Все, что она должна знать, — это тот факт, что применен шифр изгороди.

Пример 3.23

Алиса и Боб могут договориться о числе столбцов и использовать второй метод. Алиса пишет тот же самый исходный текст, строка за строкой, в таблице из четырех столбцов.

m	**e**	**e**	**t**
m	**e**	**a**	**t**
t	**h**	**e**	**p**
a	**r**	**k**	

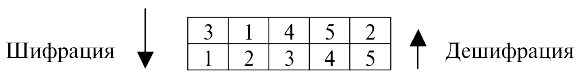
Алиса создает зашифрованный текст «MMTAEEHREAEK TTP», передавая символы столбец за столбцом. Боб получает зашифрованный текст и применяет обратный процесс. Он пишет полученное сообщение столбец за столбцом и читает его строка за строкой как исходный текст. Ева может легко расшифровать сообщение, если она знает число столбцов.

Пример 3.24

Шифр в примере 3.23 — реальный шифр перестановки. Далее покажем перестановку каждой буквы исходного текста и зашифрованный текст, базируясь на номерах их позиций.

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
01	05	09	13	02	06	10	14	03	07	11	15	04	08	12

Второй символ в исходном тексте передвинулся на пятую позицию в зашифрованном тексте; третий символ передвинулся на девятую позицию; и так далее. Хотя символы переставлены, они сами являются шаблонами: (01, 05, 09, 13), (02,



06, 10, 14), (03, 07, 11, 15) и (04, 08, 12). В каждой секции разность между двумя смежными номерами — 4.

Ключевые шифры перестановки

Бесключевые шифры переставляют символы, используя запись исходного текста одним способом (например, строка за строкой) и передачу этого текста в другом порядке (например, столбец за столбцом). Перестановка делается во всём исходном тексте, чтобы создать весь зашифрованный текст. Другой метод состоит в том, чтобы разделить исходный текст на группы заранее определенного размера, называемые блоками, а затем использовать ключ, чтобы переставить символы в каждом блоке отдельно.

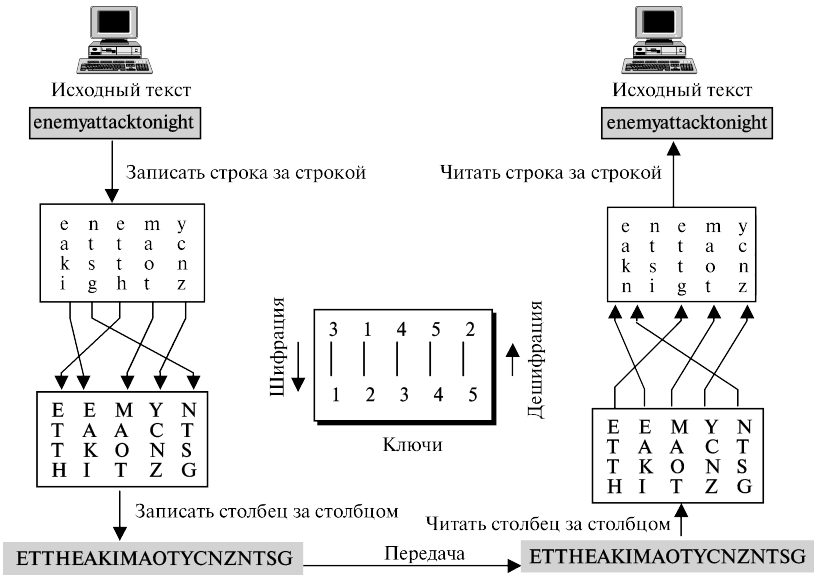
Пример 3.25

Алиса должна передать Бобу сообщение «Enemy attacks tonight» («Вражеские атаки сегодня вечером»). Алиса и Боб согласились разделить текст на группы по пять символов и затем переставить символы в каждой группе. Ниже показана группировка после добавления фиктивного символа в конце, чтобы сделать последнюю группу одинаковой по размеру с другими.

Enemy atttac kston ightz

Ключ, используемый для шифрования и дешифрования, — ключ перестановки, который показывает, как переставлять символы. Для этого сообщения примем, что Алиса и Боб использовали следующий ключ:

Третий символ в блоке исходного текста становится первым символом в зашифрованном тексте в блоке, первый символ в блоке исходного текста становит-



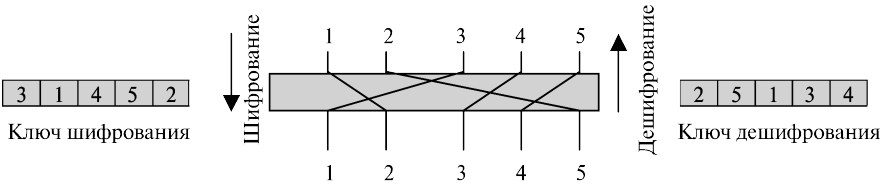
ся вторым символом в блоке зашифрованного текста и так далее. Результаты перестановки:

EEMYN TAACT TKONS HITZG

Алиса передает зашифрованный текст «EEMYNTAACTTKONSHITZG» Бобу. Боб делит зашифрованный текст на группы по 5 символов и, используя ключ в обратном порядке, находит исходный текст.

Объединение двух подходов

Современные шифры перестановки, чтобы достигнуть лучшего скремблирования, объединяют два подхода. Шифрование и дешифрование делается в три шага. Первый: текст пишется таблицей строка за строкой. Второй: делается перестановка, изменяя порядок следования столбцов. Третий: столбец за столбцом читается новая таблица. Первые и третьи шаги обеспечивают бесключевое глобальное изменение



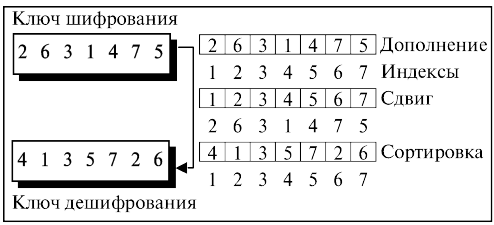
порядка следования; второй шаг обеспечивает блочную ключевую перестановку. Эти типы шифров упоминаются часто как ключевые шифры перестановки столбцов.

Пример 3.26

Предположим, что Алиса снова зашифровывает сообщение в примере 3.25, на сей раз используя объединенный подход. Шифрование и дешифрование показано на рис. 3.21.

Рис. 3.21. Пример 3.27

Первая таблица, созданная Алисой, содержит исходный текст, записанный строка за строкой. Столбцы переставлены с применением того же самого ключа, что и в предыдущем примере. Зашифрованный текст создан с помощью чтения второй



а. Ручной процесс

```
Given: EncKey {index}
index ← 1
while {index ⩽ Column}
 {
   DecKey [EncKey [index]←index
   Index←index +1
 }
Return: Dec Key [index]
```

б. Алгоритм

таблицы столбец за столбцом. Боб делает те же самые три шага в обратном порядке. Он считывает таблицу зашифрованного текста столбец за столбцом в первую таблицу, переставляет столбцы, а затем читает вторую таблицу строку за строкой.

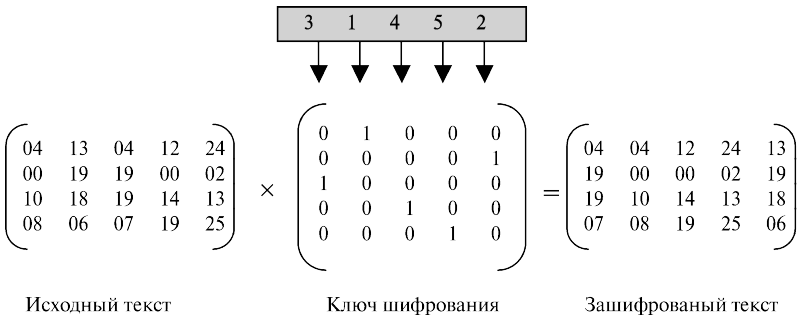
Ключи

В примере 3.27 единственный ключ использовался в двух направлениях для изменения порядка следования столбцов — вниз для шифрования, вверх для дешифрования. Обычно принято создавать два ключа для этого графического представления: один для шифрования и один для дешифрования. Ключи накапливаются в таблицах, имеющих один адрес (вход) для каждого столбца. Вход содержит исходный номер столбца — номер столбца пункта назначения, указывающий его положение от номера входа. Рисунок 3.22 показывает, как эти две таблицы могут быть созданы с помощью графического представления ключа.

Рис. 3.22. Шифрование / дешифрование в шифре перестановки

Ключ шифрования — (3 1 4 5 2). Первый вход показывает, что столбец 3 (содержание) в источнике становится столбцом 1 (положение или индекс входа) в пункте назначения. Ключ дешифрования — (2 5 1 3 4). Первый вход показывает, что столбец 2 в источнике становится столбцом 1 в пункте назначения.

Как найти ключ дешифрования, если дан ключ шифрования или, наоборот, дан ключ дешифрации? Процесс может быть выполнен вручную за несколько шагов, как это показано на рис. 3.23. Сначала добавим индексы к таблице ключей,



потом сделаем сдвиг в соответствии с полученным ключом, и, наконец, сортируем пару согласно индексу.

Рис. 3.23. Инверсия ключа в шифре перестановки

Использование матриц

Мы можем использовать матрицы, чтобы показать процесс шифрования/дешифрования для шифра перестановки. Исходный текст и зашифрованный текст — матрица $l \times m$., представляющая числовые значения символов; ключи — квадратные матрицы размера $m \times m$. В матрице перестановки каждая строка или столбец имеют строго одну единицу (1), и остальная часть значений — нули (0). Шифрование выполняется умножением матрицы исходного текста на ключевую матрицу, чтобы получить матрицу зашифрованного текста; дешифрование

производим умножением зашифрованного текста на инверсию ключевой матрицы, после чего получаем исходный текст.

Очень интересно, что матрица дешифрования в этом случае, как и всегда, — инверсия матрицы шифрования. Однако нет никакой необходимости инвертировать матрицу — ключевая матрица шифрования может просто быть переставлена (сдвиг строк и столбцов), чтобы получить ключевую матрицу дешифрования.

Пример 3.27

Рисунок 3.24 показывает процесс шифрования. Умножение матрицы 4 × 5 исходного текста на ключевую матрицу шифрования 5 × 5 дает матрицу зашифрованного текста 4 × 5. Матричная манипуляция требует изменения символов в примере 3.27 к их числовым значениям (от 00 до 25). Обратите внимание, что матричное умножение обеспечивает только перестановку столбцов; чтение и запись в матрицу должны быть обеспечены остальной частью алгоритма.

Рис. 3.24. Представление ключа в виде матрицы в шифре перестановок

Криптоанализ шифров перестановки

Шифры перестановки уязвимы к нескольким видам атак только для зашифрованного текста.

Статистическая атака

Шифр перестановки не изменяет частоту букв в зашифрованном тексте; он лишь переставляет буквы. Так что первая атака, которая может быть применена, – анализ частоты отдельной буквы. Этот метод может быть полезен, если длина зашифрованного текста достаточно большая. Мы такую атаку рассматривали раньше. Однако шифры перестановки не сохраняют частоту пар и триграмм. Это означает, что Ева не может использовать такие инструментальные средства. Фактически, если шифр не сохраняет частоту пар и триграмм, но сохраняет частоту отдельных букв, то вероятнее всего, что это шифр перестановки.

Атака грубой силы

Ева, чтобы расшифровать сообщение, может попробовать все возможные ключи. Однако число ключей может быть огромно: 1! + 2! + 3! + • • • + L!, где *L* — длина зашифрованного текста. Лучший подход состоит в том, чтобы попробовать отгадать число столбцов. Ева знает, что число столбцов делится на *L*. Например, если длина шифра — 20 символов, то 20 =1 x 2 x 2 x 5.

Это означает, что номером столбцов может быть комбинация этих коэффициентов (1, 2, 4, 5, 10, 20). Однако только один столбец и только одна строка — маловероятные варианты.

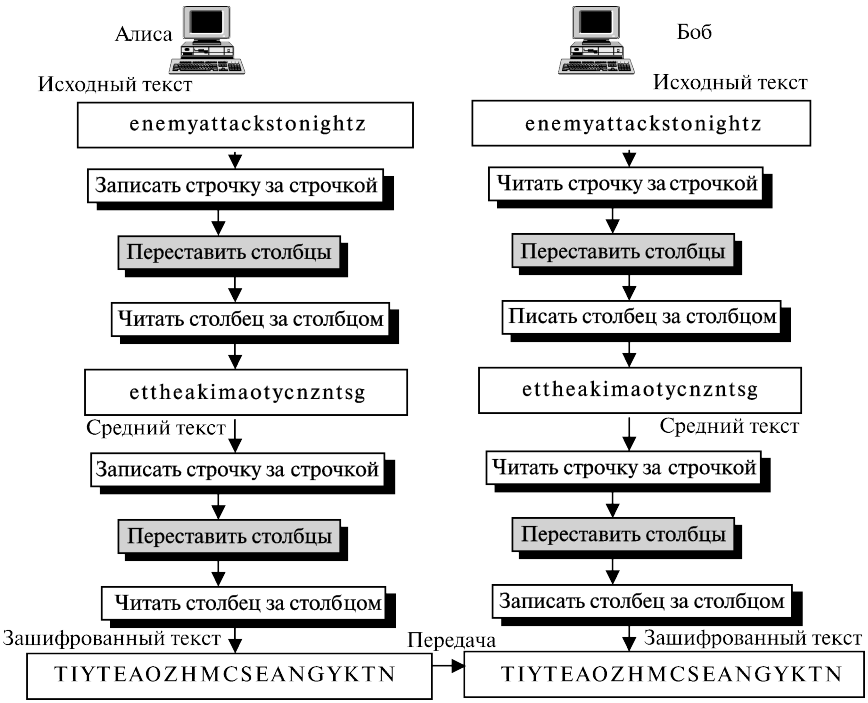
Пример 3.28

Предположим, что Ева перехватила сообщение зашифрованного текста «EEMYNTAACTTKONSHITZG». Длина сообщения L = 20, число столбцов может быть 1, 2, 4, 5, 10 или 20. Ева игнорирует первое значение, потому что это означает «только один столбец» и маловероятно.

a. Если число столбцов — 2, единственные две перестановки — (1,2) и (2, 1). Первое означает, что перестановки не было. Ева пробует вторую комбинацию. Она делит зашифрованный текст на модули по два символа “EE MY NT AA CT TK ON SH IT ZG». Затем она пробует переставлять каждый модуль из них, получая текст «ee ym nt aa tc kt no hs ti gz», который не имеет смысла.
b. Если номер столбцов — 4, тогда имеется 4! = 24 перестановки. Первая перестановка (1 2 3 4) означает, что не было никакой перестановки. Ева должна попробовать остальные. После испытания всех 23 возможностей Ева находит, что никакой исходный текст при таких перестановках не имеет смысла.
c. Если число столбцов — 5, тогда есть 5! = 120 перестановок. Первая (1 2 3 4 5) означает отсутствие перестановки. Ева должна попробовать остальные. Перестановка (2 5 13 4) приносит плоды — исходный текст «enemyattackstonightz», который имеет смысл после удаления фиктивной буквы z и добавления пробелов.

Атака по образцу

Другая атака шифра перестановки может быть названа атакой по образцу. Зашифрованный текст, созданный с помощью ключевого шифра перестановки,



имеет некоторые повторяющиеся образцы. Следующий пример показывает зашифрованный текст, относительно которого известно, что каждый символ в зашифрованном тексте в примере 3.28 получается из исходного текста по следующему правилу:

03 08 13 18 *01 06 11 16* **04 09 14 19** *05 10 15 20* **02 07 12 17**

1-й символ в зашифрованном тексте получается из 3-го символа исходного текста. 2-й символ в зашифрованном тексте получается из 8-го символа исходного текста. 20-й символ в зашифрованном тексте получается из 17-го символа исходного текста, и так далее. У нас имеются образцы в вышеупомянутом списке. Мы имеем пять групп: (3, 8, 13, 18), (1, 6, 11, 16), (4, 9, 14, 19), (5, 10, 15, 20) и (2, 7, 12, 17). Во всех группах разность между двумя смежными номерами — 5. Эта регулярность может использоваться криптоаналитиком, чтобы взломать шифр. Если Ева знает или может предположить число столбцов (в этом случае оно равняется 5), она может преобразовать зашифрованный текст в группы по четыре символа. Перестановка групп может обеспечить ключ к нахождению исходного текста.

Шифры с двойной перестановкой

Шифры с двойной перестановкой могут затруднить работу криптоаналитика. Примером такого шифра является повторение дважды алгоритма, используемого для шифрования и дешифрования в примере 3.26. На каждом шаге может применяться различный ключ, но обычно ключ используется один и тот же.

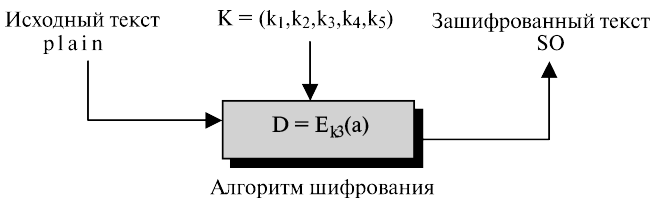
Пример 3.29

Повторим пример 3.26, где использована двойная перестановка. Рисунок 3.25 показывает процесс.

Рис. 3.25. Двойной шифр перестановки

Хотя криптоаналитик может еще использовать частоту появления отдельного символа для статистической атаки на зашифрованный текст, атака по образцу теперь затруднена. Образец анализа текста выглядить так:

13 16 05 07 03 06 10 20 18 04 10 12 01 09 15 17 08 11 19 02



Сравнив приведенный текст и результат примера 3.28, мы видим, что теперь нет повторяющихся образцов. Двойная перестановка удалила ту регулярность, что существовала раньше.

3.4. Шифры потока и блочные шифры

В литературе симметричные шифры разделяют на две категории: шифры потока и блочные шифры. Хотя эти категории применяются к современным шифрам, но они могут также работать в традиционных шифрах.

Шифры потока

В шифрах потока шифрование делается в один момент времени над одним символом (таким, как буква или бит). Мы имеем поток исходного текста, поток зашифрованного текста и поток ключей. Обозначим исходный поток P, поток зашифрованного текста — C и поток ключей — K.

$$P = P_1P_2P_3,..... \quad C = C_1C_2C_3,.... \quad K = (k_1, k_2, k_3,...)$$
$$C_1 = E_{k1}(P_1) \quad C_2 = E_{k2}(P_2) \quad C3 = E_{k3}(P_3)...$$

Фигура 3.26 показывает идею указанного ранее шифра потока. Символы обычного текста принимаются алгоритмом шифрования по одному. Символы зашифрованного текста также создаются по одному в один и тот же момент времени. Ключевой поток может быть создан многими способами. Это может быть поток с заранее определенными значениями; это может быть только одно значение, используемое алгоритмом. Значения могут зависеть от исходного текста или символов зашифрованного текста. Значения могут также зависеть от предыдущих ключевых значений.

Рис. 3.26. Шифр потока

Рисунок 3.26 показывает момент, когда третий символ в потоке исходного текста был зашифрован с использованием третьего значения в ключевом потоке. Результат — это третий символ в потоке зашифрованного текста.

Пример 3.30

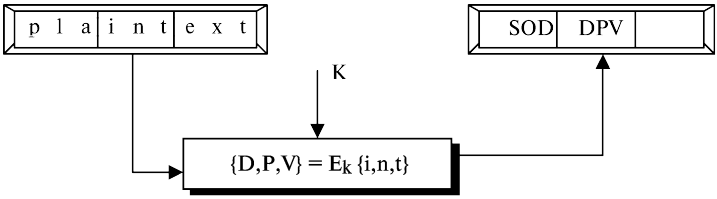
Аддитивные шифры могут быть отнесены к категории шифров потока, в которых ключевой поток является повторным значением ключа. Другими словами, ключевой поток рассматривают как заранее определенный поток ключей или K = (*k,k,...k*). В этом шифре, однако, каждый символ в зашифрованном тексте зависит только от соответствующего символа в исходном тексте, потому что ключевой поток генерировался независимо.

Пример 3.31

Моноалфавитные шифры подстановки, которые мы рассмотрели в этой лекции, — также шифры потока. Однако каждое значение ключевого потока в этом случае — отображение текущих исходных букв в соответствующие символы зашифрованного текста по таблице отображения.

Пример 3.32

Шифры Виженера — также шифры потока согласно определению. В этом случае ключ потока — повторение m значений, где m. — размер ключевого слова. Другими словами,



Алгоритм шифрования

$$K = (k_1, k_2, ...k_m, k_1, k_2, ...k_m....)$$

Пример 3.33

Мы можем установить критерий для разделения шифров потока, основанных на ключевых потоках. Мы можем сказать, что шифр потока — моноалфавитный шифр, если значение k_i не зависит от исходного символа исходного текста в потоке исходного текста; в противном случае шифр является многоалфавитным.

- Аддитивные шифры являются моноалфавитными, потому что k_i в ключевом потоке — зафиксированный (постоянный), он не зависит от позиции символа в исходном тексте.
- Моноалфавитные шифры подстановки являются явно *моноалфавитными шифрами потока*, потому что k_i не зависит от позиции соответствующего символа в потоке исходного текста, а зависит лишь от значения символа в исходном тексте.
- Шифры Виженера — многоалфавитные шифры, потому что k_i явно зависит от позиции символа исходного текста. Однако зависимость является циклической. Одинаковый ключ для двух символов разделен m позициями.

Блочные шифры

В **блочном шифре** группа символов исходного текста размера m ($m > 1$) зашифровывается, создавая вместе группу зашифрованного текста одного и того же размера. Основанный на этом определении блочный шифр использует единственный ключ, чтобы зашифровать целый блок, даже если ключ делает перемножение значений. Рисунок 3.27 показывает принципы блочного шифра.

Рис. 3.27. Блочный шифр

В блочном шифре блок зашифрованного текста зависит от целого блока исходного текста.

Пример 3.34

Шифры плейфеера — блочные шифры. Размер блока — $m = 2$. Два символа зашифрованы вместе.

Пример 3.35

Шифры Хилла — блочные шифры. Блок исходного текста размера 2 или больше зашифрован с совместным использованием единственного ключа (матрицы). В этих шифрах значение каждого символа в зашифрованном тексте зависит от всех значений символов в исходном тексте. Хотя ключ может быть получен *из* $m \times m$. значений, он рассматривается как единственный ключ.

Пример. 3.36

Из определения блочного шифра ясно, что каждый блочный шифр — это многоалфавитный шифр, потому что каждая буква шифрованного текста в случае блочного шифра зависит от всех букв исходного текста.

Комбинация

На практике блоки исходного текста шифруются индивидуально, но они используют ключи потока для того, чтобы зашифровать все сообщение блок за блоком. Другими словами, шифр — блочный, когда применяется к индивидуальным блокам, но он же и шифр потока, когда применяется ко всему сообщению, рассматривая каждый блок как единицу. Каждый блок применяет различный ключ, который был сгенерирован заранее или в течение процесса шифрования. Примеры этого будут рассмотрены позднее.

3.5. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, приведены в списке ссылок в конце книги.

Книги

Несколько книг рассматривают классические шифры с симметричным ключом. [Kah961 и [Sin99] дают полную историю этих шифров. [Sti06L [Bar02], ITW06], [Cou99], [Sta06], [SchOl], [Mao03] и [Gar0l] приводят хороший анализ технических деталей.

Сайты

Нижеследующие сайты дают больше информации о темах, рассмотренных в этой лекции.

http://www.cryptogram.org
http://www.cdt.org/crypto/
http://www.cacr.math.uwaterloo.ca/
http://www.acc.stevens.edu/crypto.php
http://www.crypto .com/
http: // theory.lcs.mit.edu / ~ rivest/crypto-security.html

http://www.trincoll.edu/depts/cpsc/cryptography/substitution.html
http://hem.passagen.se/tan01/transpo.html
http://www.strangehorizons.comy2001/20011008/steganography.shtml

3.6. Итоги

- Шифрование симметричными ключами использует единственный ключ для шифрования и для дешифрования. Алгоритмы шифрования и дешифрования являются обратными друг другу.
- Первоначальное сообщение называется исходным текстом. Сообщение, которое передаётся через канал, называется зашифрованным текстом. Чтобы создавать зашифрованный текст из исходного текста, алгоритм шифрования пользуется общедоступным ключом засекречивания. Чтобы создавать исходный текст из зашифрованного текста, применяется алгоритм дешифрования тот же самый ключ засекречивания.
- На основании принципа Керкхоффа, нужно всегда принимать, что противник знает алгоритм шифрования/дешифрования. Устойчивость шифра к атаке должна быть основана только на тайне ключа.
- Криптоанализ — наука и искусство «взлома» шифров. Есть четыре общих типа атак криптоанализа: атака только на зашифрованный текст, атака по образцу, атака с выборкой исходного текста, атака с выборкой зашифрованного текста.
- Традиционные шифры с симметричным ключом могут быть разделены на две обширных категории: шифры подстановки и шифры перестановки. Шифр подстановки заменяет один символ другим символом. Шифр перестановки переупорядочивает символы.
- Шифры подстановки могут быть разделены на две обширных категории: моноалфавитные шифры и многоалфавитные шифры. В моноалфавитной подстановке отношения между символом в исходном тексте и символом в зашифрованном тексте являются непосредственными. В многоалфавитной подстановке отношения между символами в исходном тексте и символами в зашифрованном тексте — «один ко многим».
- Моноалфавитные шифры включают в себя аддитивные, мультипликативные, аффинные и моноалфавитные шифры подстановки.
- Многоалфавитные шифры включают в себя шифры: автоключевой, плейфера, Виженера, Хилла, одноразового блокнота, ротора, и шифры «Энигмы».
- Шифры перестановки включают в себя: бесключевой, ключевой шифры и шифры с двойной перестановкой.
- Симметричные шифры могут также быть разделены на две обширных категории: шифры потока и блочные шифры. В шифре потока шифрование и дешифрование одного символа производятся в один момент времени. В блочном шифре символы в блоке зашифрованы вместе. Практически, блоки исходного текста зашифрованы индивидуально, но они используют поток ключей, чтобы зашифровать все сообщение блок за блоком.

3.7. Набор для практики

Обзорные вопросы

1. Определите шифр с симметричным ключом.
2. Поясните отличия между шифром подстановки и шифром перестановки.
3. Поясните отличия между моноалфавитным и многоалфавитным шифрами.
4. Поясните отличия между шифром потока и блочным шифром.
5. Все ли шифры потока являются моноалфавитными? Поясните.
6. Все ли блочные шифры являются многоалфавитными? Поясните.
7. Перечислите три моноалфавитных шифра.
8. Перечислите три многоалфавитных шифра.
9. Перечислите два шифра перестановки.
10. Перечислите четыре вида атак криптоанализа.

Упражнения

1. Маленький частный клуб имеет только 100 членов. Ответьте на следующие вопросы:
   a. Сколько ключей засекречивания необходимо иметь, если все члены клуба хотят передавать секретные сообщения друг другу?
   b. Сколько ключей засекречивания необходимо, если каждый доверяет президенту клуба? Если один член клуба должен передать сообщение другому, он сначала передает это президенту; президент тогда передает сообщение другому члену клуба.
   c. Сколько ключей засекречивания необходимо, если президент решает, что два члена клуба, которые должны связаться друг с другом, должны сначала войти в контакт с ним? Президент тогда создает временный ключ, который используется между этими двумя членами клуба. Временный ключ зашифровывается и посылается обоим членам клуба.
2. Археологи нашли новый манускрипт, написанный на неизвестном языке. Позже они нашли маленькую табличку, которая содержит предложение, написанное на том же самом языке с переводом на греческий язык. Используя табличку, они смогли прочитать первоначальную рукопись. Какую атаку применили археологи?
3. Алиса может использовать только аддитивный шифр на своем компьютере, чтобы передать сообщение другу. Она думает, что сообщение будет более безопасно, если она зашифрует его два раза, каждый раз с различным ключом. Действительно ли она права? Обоснуйте ваш ответ.
4. Алиса хочет передать длинное сообщение. Она использует моноалфавитный шифр подстановки. Она думает, что если она сожмет сообщение, это может защитить текст от атаки Евы по частоте отдельных букв. Помогает ли сжатие? Должна ли она сжать сообщение, прежде чем зашифрует его или после этого? Обоснуйте ваш ответ.
5. Алиса часто должна зашифровывать исходный текст, использующий вместе буквы (от a до z) и цифры (от 0 до 9).

a. Если она применяет аддитивный шифр, что является множеством ключей? Какие будут модули?
b. Если она применяет мультипликативный шифр, что является множеством ключей? Какие будут модули?
c. Если она применяет аффинный шифр, что является множеством ключей? Какие будут модули?

6. Предположим, что к исходному тексту добавляются пробелы, точки и знаки вопроса, чтобы увеличить множество ключей элементарных шифров.
   a. Каково множество ключей, если используется аддитивный шифр?
   b. Каково множество ключей, если используется мультипликативный шифр?
   c. Каково множество ключей, если используется аффинный шифр?
7. Алиса и Боб решили игнорировать принципы Керкхоффа и скрывают тип шифра, который они используют.
   a. Как может Ева понять, использовался ли шифр подстановки или шифр перестановки?
   b. Если Ева знает, что использованный шифр — шифр подстановки, как может она определить, был ли он аддитивным, мультипликативным или аффинным шифром?
   c. Если Ева знает, что использованный шифр — шифр перестановки, как она может определить размер секции (m)?
8. В каждом из следующих шифров — какое максимальное число символов может быть изменено в зашифрованном тексте, если в исходном тексте изменен только единственный символ?
   a. Аддитивный
   b. Мультипликативный
   c. Аффинный
   d. Виженера
   e. Автоключевой
   f. Одноразовый блокнот
   g. Роторный
   h. «Энигма»
9. В каждом из следующих шифров — какое максимальное число символов будет изменено в зашифрованном тексте, если в исходном тексте изменен только один символ?
   a. Одиночная перестановка
   b. Двойная перестановка
   c. Плейфеер

$$K = \begin{pmatrix} 03 & 02 \\ 05 & 07 \end{pmatrix}$$

10. Для каждого из следующих шифров определите, является ли он шифром потока или блочным шифром. Обоснуйте ваши ответы.
    a. Плейфеер

b. Автоключ
c. Одноразовый блокнот
d. Ротор
e. «Энигма»

11. Зашифруйте сообщение «this is exercise» («это — упражнение»), используя один из следующих шифров. Игнорируйте пробелы между словами. Расшифруйте сообщение, чтобы получить первоначальный исходный текст.
    a. Аддитивный шифр с ключом = 20
    b. Мультипликативный шифр с ключом = 15
    c. Аффинный шифр с ключом = (15, 20)
12. Зашифруйте сообщение «the house is being sold tonight» («дом продан сегодня вечером»), используя один из следующих шифров. Игнорируйте пробелы между словами. Расшифруйте сообщение, чтобы получить исходный текст.
    a. Шифр Виженера с ключом: «dollars»
    b. Шифр с автоматическим ключом = 7
    c. Шифр плейфеера с ключом, созданным в тексте (см. рис. 3.13)
13. Используйте шифр Виженера с ключевым словом «HEALTH», чтобы зашифровать сообщение «Life is full surprises» («Жизнь полна сюрпризов»).
14. Используйте шифр плейфеера, чтобы зашифровать сообщение «The key hidden under the door pad» («ключ спрятан под ковриком у двери»). Ключ засекречивания можно составить, заполняя первую и вторую часть строки со словом «GUIDANCE» и заполняя остальную часть матрицы с остальной частью алфавита.
15. Используйте шифр Хилла, чтобы зашифровать сообщение «We live in an insecure world» («Мы живем в опасном мире»). Примените следующий ключ:
16. Джон читает тайную книгу введения в криптографию. В одной части книги автор дает зашифрованный текст «CIW» и двумя параграфами позже говорит читателю, что это — ключ сдвига, и исходный текст — «YES» («да»). В следующей лекции герой нашел табличку с выгравированным на ней текстом «XV1EWYW1». Джон немедленно разгадал фактическое значение зашифрованного текста. Какой тип атаки предпринял Джон? Каков исходный текст?
17. Ева тайно получает доступ к компьютеру Алисы и, используя ее шифр, печатает «abcdefghij»; на экране появилось «CABDEHEGIJ». Предположим, Ева знает, что Алиса использует ключевой шифр перестановки. Ответьте на следующие вопросы:
    a. Какую атаку предпринимает Ева?
    b. Каков размер ключа перестановки?
18. Используйте атаку грубой силы, чтобы расшифровать следующее сообщение, зашифрованное Алисой, применяя аддитивный шифр. Предположим, что Алиса всегда использует ключ, связанный с ее днем рождения, который приходится на 13-е число месяца.

NCJAEZRCLASJLYODEPRLYZRCLASJLCPEHZDTOPDZQLNZTY J

19. Используйте атаку грубой силы, чтобы расшифровать следующее сообщение. Предположите, что Вам известно: шифр — аффинный и исходный текст «ab» зашифрован «GL».

XPALASXYFGFUKPXUSOGEUTKCDGFXANMGNVS

20. Используйте атаку частоты отдельных букв, чтобы расшифровать следующее сообщение. Предположите, что Вам известно, что оно зашифровано с применением моноалфавитного шифра подстановки.

ONHOVEJHWOBEVGWOCBWHNUGBLHGBGR

21. Предположим, что знаки препинания (точки, вопросительные знаки и пробелы) складываются с алфавитом шифрования шифра Хилла, потом для шифрования и дешифрования используются ключевые матрицы 2 × 2 в Z_{29}.
    a. Найдите общее количество возможных матриц.
    b. Доказано, что общее количество обратимых матриц — $(N^2 - 1)(N^2 - N)$, где N — число размера алфавита. Найдите множество ключей шифра Хилла, используя этот алфавит.
22. Используйте атаку частоты отдельных букв, чтобы взломать следующий зашифрованный текст. Предположите, что Вам известно, что он был создан с использованием аддитивного шифра.

OTWEWNGWCBPQABIZVQAPMLJGZWTTQVOBQUMAPMIDGZCAB
EQVBMZLZIXMLAXZQVOQVLMMXAVWEIVLLIZSNZWAB
JQZLWNLMTQOPBVIUMLGWCBPAEQNBTGTMNBBPMVMAB
ITIAKWCTLVBBQUMQBEPQTMQBEIAQVUGBZCAB

23. Используйте тест Казиского и атаку частоты отдельных букв, чтобы нарушить следующий зашифрованный текст. Предположите, что Вам известно, что он был создан шифром Виженера.

MPYIGOBSRMIDBSYRDIKATXAILFDFKXTPPSNTTJIGTHDELT
TXAIREIHSVOBSMLUCFIOEPZIWACRFXICUVXVTOPXDLWPENDHPRSI
DDBXWWTZPHNSOCLOUMSNRCCVUUXZHHNWSVXAUHIK
LXTIMOICHTYPBHMHXGXHOLWPEWWWWDALOCTSQAELT

24. Ключ шифрования в шифре перестановки — (3, 2, 6, 1, 5, 4). Найдите ключ дешифрования.
25. Покажите матричное представление ключа шифрования перестановки с ключом (3, 2, 6, 1, 5, 4). Найдите матричное представление ключа дешифрования.

26. Даны исходный текст «letusmeetnow» и соответствующий зашифрованный текст «HBCDFNOPIKLB». Известно, что алгоритм — шифр Хилла, но вы не знаете размер ключа. Найдите ключевую матрицу.
27. Шифры Хилла и мультипликативные шифры очень похожи. Шифры Хилла — блочные шифры, использующие умножение матриц; мультипликативные шифры — шифры потока, использующие скалярное умножение.

	1	2	3	4	5
1	z	q	p	f	e
2	y	r	o	g	d
3	x	s	n	h	c
4	w	t	m	i/j	b
5	v	u	l	k	a

    a. Определите блочный шифр, который, подобно аддитивному шифру, использует сложение матриц.
    b. Определите блочный шифр, который, подобно аффинному шифру, использует умножение и сложение матриц.
28. Определите новый шифр потока. Шифр является аффинным, но ключи зависят от позиции символа в исходном тексте. Если символ исходного текста будет зашифрован в позиции *i,* мы можем найти ключи:
    a. Мультипликативный ключ — (*i* mod 12) элемент в Z_{26}^*.
    b. Аддитивный ключ — (*i* mod 26) элемент в Z_{26}.

    Зашифруйте сообщение «cryptography is fun» («криптография — забавно»), используя этот новый шифр.
29. Предположим, что для шифра Хилла исходный текст является мультипликативной единичной матрицей (I). Найдите отношения между ключом и зашифрованным текстом. Используйте результат вашего исследования и попытайтесь атаковать выборку исходного текста, применяющего шифр Хилла.
30. Atbash был популярным шифром среди Библейских авторов (VI век до нашей эры). В Atbash «A» — шифровалось буквой «Z», «B» был зашифрован буквой «Y», и так далее. Аналогично «Z» был зашифрован как «A», «Y» зашифрован как «B», и так далее. Предположим, что алфавит разделен на две половины и буквы в первой половине зашифрованы как буквы во второй и наоборот. Найдите тип шифра и ключа. Зашифруйте сообщение «упражнение», используя Atbash-шифр.
31. В шифре Полибиуса (Polybius — римский историк, живший в IV веке до нашей эры) каждая буква зашифрована как два целых числа. Ключ — матрица символов 5 × 5, как в шифре плейфеера. Исходный текст — матрица символов, зашифрованный текст — эти два целых числа (каждое между 1 и 5), представляющие столбцы и строки. Зашифруйте сообщение «An exercise» («упражнение»), используя шифр Полибиуса (Polybius) со следующим ключом:

Лекция 4. Математика криптографии. Часть II. Алгебраические структуры

Цели и содержание

Мы постараемся здесь подготовить читателя к следующим нескольким лекциям, в которых рассматриваются современные шифры с симметричным ключом, основанные на алгебраических структурах. Эта лекция имеет несколько целей:

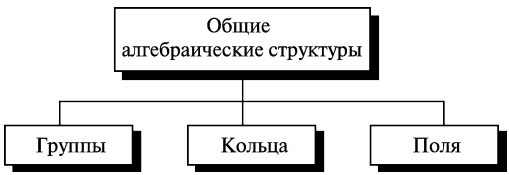
- рассмотреть понятие алгебраических структур;
- определить и привести некоторые примеры алгебраических групп;
- определить и привести некоторые примеры алгебраических колец;
- определить и привести некоторые примеры алгебраических полей;
- поговорить о таких операциях, как сложение, вычитание, умножение и деление с *n*-битовыми словами в современных блочных шифрах.

Следующие лекции посвящены обсуждению современных симметрично-ключевых блочных шифров, которые выполняют операции с n-битовыми словами. Понимание и анализ этих шифров требуют некоторого знания разделов современной алгебры, называемых алгебраическими структурами. В начале этой лекции делается обзор алгебраических структур, а затем показано, как выполнить сложение или умножение с n-битовыми словами.

4.1. Алгебраические структуры

В лекции 2 мы обсуждали некоторые множества чисел, таких, как Z, Z_n, Z_{n^*}, Z_p и Z_{p^*}. Криптография требует, чтобы были заданы множества целых чисел, и операции, определенные для них. Комбинация множеств и операций, которые могут быть применены к элементам множества, называется **алгебраической структурой.** В этой лекции мы определим три общих алгебраических структуры: *группы, кольца* и *поля* (рис. 4.1).

Рис. 4.1. Общие алгебраические структуры



Группы

Группа (G) — набор элементов с бинарной операцией «•» — обладает четырьмя свойствами (или удовлетворяет аксиомам), которые будут перечислены ниже.

- **Коммутативная группа**, также называемая **абелевой**, — группа, в которой оператор обладает теми же четырьмя свойствами для групп плюс дополнительным — коммутативностью. Эти пять свойств определены далее.

- **Замкнутость.** Если *a* и *b* — элементы G, то c = *a* • *b* — также элемент G.

Это означает, что результат применения операции к любым двум элементам множества есть элемент этого множества.

- **Ассоциативность.** Если a, *b* и *c* — элементы **G**, то верно

(a • *b)* • *c* = *a* • *(b* • c)

Другими словами, не имеет значения, в каком порядке мы применяем операцию более чем к двум элементам.

- **Коммутативность.** Для всех *a* и *b* в **G** мы имеем *a* • *b* = *b* • *a*. Обратите внимание, что это свойство должно быть верно только для коммутативной группы.
- **Существование нейтрального элемента.** Для все элементов в **G** существует элемент *e*, который называется нейтральным элементом, такой, что *e* • *a* = *a* • *e* = *a*.
- **Существование инверсии.** Для каждого *a* в **G** существует элемент *a',* называемый обратным к *a*, такой, что *a* • *a'* = *a'* • *a* = *e*.

Рисунок 4.2 иллюстрирует понятие группы.

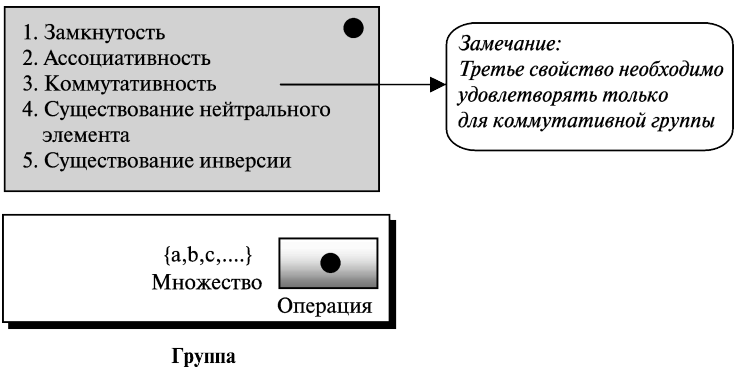


Рис. 4.2. Группа

Приложение

Хотя группа включает единственный оператор, свойства, присущие каждой операции, позволяют использование пары операций, если они инверсны друг другу. Например, если определенный выше оператор — сложение, то группа поддерживает и сложение, и вычитание, ибо вычитание и сложение — аддитивно инверсные операции. Это также верно для умножения и деления. Однако группа может поддержать только сложение/вычитание или умножение/деление, но не оба сочетания операторов одновременно.

Пример 4.1

Множество вычетов целых чисел с операцией сложения,

G = <Z_n, +>, является коммутативной группой. Мы можем выполнить сложение и вычитание на элементах этого множества, не выходя за его пределы.

Проверим эти свойства.

1. Замкнутость удовлетворяется. Результат сложения двух целых чисел в Z_n — другое целое число в Z_n.
2. Ассоциативность удовлетворяется. Результат 4 + (3 + 2) тот же самый, что в случае (4 + 3) + 2.
3. Коммутативность удовлетворяется. Мы имеем 3 + 5 = 5 + 3.
4. Нейтральный элемент — 0. Мы имеем 3 + 0 = 0 + 3 = 3.
5. Каждый элемент имеет аддитивную инверсию. Инверсия элемента — его дополнение. Например, инверсия 3 — это −3 *(n* − 3 в Z_n), и инверсия −3 — это 3. Инверсия позволяет нам выполнять вычитание на множестве.

Пример 4.2

Множество Z_{n*} с оператором умножения $G = <Z_{n*}, \times >$ является также абелевой группой. Мы можем выполнить умножение и деление на элементах этого множества, не выходя за его пределы. Это облегчает проверку первых трех свойств. Нейтральный элемент равен 1. Каждый элемент имеет инверсию, которая может быть найдена согласно расширенному алгоритму Евклида.

Пример 4.3

Хотя мы обычно представляем группу как множество чисел с обычными операторами, такими, как сложение или вычитание, определения группы позволяют нам определять любое множество объектов и операций, которые удовлетворяют вышеупомянутым свойствам. Определим множество G = <{a, *b, c, d,},* •> и операцию, показанную с помощью таблицы 4.1.

Таблица 4.1. Таблица операции для примера 4.3

•	a	b	c	d
a	a	b	c	d
b	b	c	d	a
c	c	d	a	b
d	d	a	b	c

Это — абелева группа. Все пять свойств удовлетворены.

1. Замкнутость удовлетворена. Применение оператора на любой паре элементов дает в результате другой элемент этого множества.
2. Ассоциативность также удовлетворена. Чтобы доказать это, мы должны проверить свойство для любой комбинации из трех элементов. Например, (a+ b) + c = a+ (b + c) = d.
3. Операция коммутативна. Мы имеем a + b = b + a.
4. Группа имеет нейтральный элемент, которым является *a*.
5. Каждый элемент имеет инверсию. Обратные пары могут быть найдены. В таблице они указаны теневыми элементами в каждой строке. Пары — (a, a), (b, d), (c, c).

Пример 4.4

Элементы в группе не обязательно должны быть числами или объектами; они могут быть правилами, отображениями, функциями или действиями. Очень

интересная группа — **группа перестановок.** Множество всех перестановок и оператор является композицией: применения одной перестановки за другой. Рисунок 4.3 показывает композиции двух перестановок, которые перемещают три входных сигнала, чтобы создать три выходных сигнала.

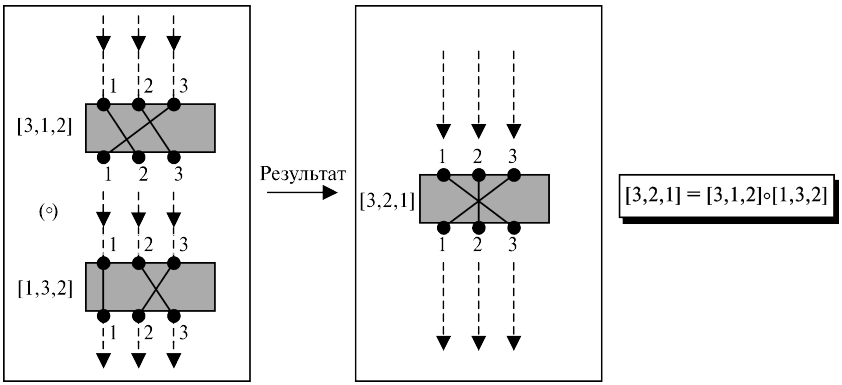


Рис. 4.3. Композиции перестановок (Пример 4.4)

Входные сигналы и выходные сигналы могут быть символами (лекция 2) или битами (лекция 5). Мы изобразили каждую перестановку прямоугольником, внутри которого показано, как исходящий входной сигнал и индекс (1,2,3) определяет выходной сигнал. Композиция состоит из двух перестановок в порядке одна за другой. При трех входных сигналах и трех выходных сигналах может быть 3! или 6 различных перестановок. Таблица 4.2 дает определение этого оператора. Первая строка — первая перестановка; первый столбец — вторая перестановка. Результат содержится на пересечении.

В этом случае удовлетворены только четыре свойства; поэтому группа — не абелева.

1. Замкнутость удовлетворена.
2. Ассоциативность также удовлетворена. Чтобы доказать это, мы должны проверить свойство для любой комбинации из трех элементов.
3. Свойство коммутативности не удовлетворено. Это может быть легко проверено, но мы оставим проверку для упражнения.

Таблица 4.2. Таблица операции для группы перестановок

	[1 2 3]	[1 3 2]	[2 1 3]	[2 3 1]	[3 1 2]	[3 2 1]
[1 2 3]	[1 2 3]	[1 3 2]	[2 1 3]	[2 3 1]	[3 1 2]	[3 2 1]
[1 3 2]	[1 3 2]	[1 23]	[2 3 1]	[2 1 3]	[3 2 1]	[3 1 2]
[2 1 3]	[2 1 3]	[3 1 2]	[1 2 3]	[3 2 1]	[1 3 2]	[2 3 1]
[2 3 1]	[2 3 1]	[3 2 1]	[1 3 2]	[3 1 2]	[1 2 3]	[2 1 3]
[3 1 2]	[3 1 2]	[2 1 3]	[3 2 1]	[1 2 3]	[2 3 1]	[1 3 2]
[3 2 1]	[3 2 1]	[2 3 1]	[3 1 2]	[1 3 2]	[2 1 3]	[1 2 3]

4. Множество имеет нейтральный элемент [1 2 3] (перестановка отсутствует). Эти элементы показаны другим цветом.
5. Каждый элемент имеет инверсию. Обратные пары могут быть найдены, если использовать нейтральные элементы.

Пример 4.5

В предыдущем примере мы показали, что множество перестановок с композицией операций — группа. Поэтому применение двух перестановок (одна за другой) не может усилить безопасность шифра. Мы сможем всегда найти перестановку, которая сделает ту же самую операцию, используя свойства замкнутости.

Конечная группа

Группа называется **конечной группой**, если множество имеет конечное число элементов; иначе это — **бесконечная группа**.

Порядок группы

Порядок группы, **G** , — это число элементов в группе. Если группа не конечна, ее порядок бесконечен; если конечна, порядок конечен.

Подгруппы

Подмножество **H** группы **G** — **подгруппа G**, если само **H** — группа относительно операции на **G**. Другими словами, если **G** = **<S,** • > — группа, то **H** = **<T,** • **>** — группа для той же самой операции, и если T — непустое подмножество S, то **H —** подгруппа **G.** Вышеупомянутое определение подразумевает, что:

1. если a и *b* — члены обеих групп, то $c = a \cdot b$ — также элемент обеих групп;
2. для группы и подгруппы имеется один и тот же нейтральный элемент;
3. если этот элемент принадлежит обеим группам, инверсия *a* — также элемент обеих групп;
4. группа, полученная с помощью нейтрального элемента **G**, H = <{e}, • >, является подгруппой **G**;
5. каждая группа — подгруппа самой себя.

Пример 4.6

Является ли группа H = $<Z_{10}$, +> подгруппой группы G = $<Z_{12}$, +>?

Решение

Ответ — нет. Хотя H — подмножество G, операции, определенные для этих двух групп, различны. Операция H — сложение по модулю 10; операция в G — сложение по модулю 12.

Циклические подгруппы

Если подгруппа группы может быть сгенерирована, используя возведение в степень элемента, то такая подгруппа называется **циклической подгруппой.** Термин *возведение в степень* здесь означает многократное применение к элементу групповой операции:

$$a^n \rightarrow a \cdot a \cdot \ldots \cdot a \ (n \text{ раз})$$

Множество, полученное в результате этого процесса, обозначается в тексте как $<a>$. Обратите внимание также, что $a^0 = e$.

Пример 4.7

Из группы $G = < Z_6, +>$ могут быть получены четыре циклических подгруппы. Это $\mathbf{H_1} = <\{0\},+>$, $\mathbf{H_2} =<\{0, 2, 4\}, +>$, $\mathbf{H_3} = <\{0, 3\}, +>$ и $\mathbf{H_4 = G}$. Заметим, что когда операция — сложение, a^n означает умножение n на a. Заметим также, что во всех этих группах операция — это сложение по модулю 6. Ниже показано, как мы находим элементы этих циклических подгрупп.

а. Циклическая подгруппа, сгенерированная из 0, — это $\mathbf{H_1}$, которая имеет только один элемент (нейтральный элемент).

$0^0 \bmod 6 = 0$ (остановка, далее процесс повторяется)

б. Циклическая подгруппа, сгенерированная на основе 1, — **это H4**, которая есть сама группа G.

$1^0 \bmod 6 = 0$
$1^1 \bmod 6 = 1$
$1^2 \bmod 6 = (1 + 1) \bmod 6 = 2$
$1^3 \bmod 6 = (1 + 1 + 1) \bmod 6 = 3$
$1^4 \bmod 6 = (1 + 1 + 1 + 1) \bmod 6 = 4$
$1^5 \bmod 6 = (1 + 1 + 1 + 1 + 1) \bmod 6 = 5$ (остановка, далее процесс повторяется)

в. Циклическая подгруппа, сгенерированная на основе 2, — это $\mathbf{H_2}$, которая имеет три элемента: 0, 2, и 4.

$2^0 \bmod 6 = 0$
$2^1 \bmod 6 = 2$
$2^2 \bmod 6 = (2 + 2) \bmod 6 = 4$ (остановка, далее процесс повторяется)

г. Циклическая подгруппа, сгенерированная на основе 3, — это $\mathbf{H_3}$, которая имеет два элемента: 0 и 3.

$3^0 \bmod 6 = 0$
$3^1 \bmod 6 = 3$ (остановка, далее процесс повторяется)

д. Циклическая подгруппа, сгенерированная на основе 4, — $\mathbf{H_2}$; это — не новая подгруппа.

$4^0 \bmod 6 = 0$
$4^1 \bmod 6 = 4$
$4^2 \bmod 6 = (4 + 4) \bmod 6 = 2$ (остановка, далее процесс повторяется)

е. Циклическая подгруппа, сгенерированная на основе 5, — **это** $\mathbf{H_4}$, она есть сама группа G.

$5^0 \bmod 6 = 0$
$5^1 \bmod 6 = 5$
$5^2 \bmod 6 = 4$
$5^3 \bmod 6 = 3$
$5^4 \bmod 6 = 2$
$5^5 \bmod 6 = 1$ (остановка, далее процесс повторяется)

Пример 4.8

Из группы $G = \langle Z_{10^*}, \times \rangle$ можно получить три циклических подгруппы. G имеет только четыре элемента: 1, 3, 7 и 9. Циклические подгруппы — $H_1 = \langle \{1\}, \times \rangle$, $H_2 = \langle \{1, 9\}, \times \rangle$ и $\mathbf{H_3 = G}$. Ниже показано, как мы находим элементы этих подгрупп.

а. Циклическая подгруппа, сгенерированная на основе 1, — это $\mathbf{H_1}$. Подгруппа имеет только один элемент, а именно — нейтральный.

$1^0 \bmod 10 = 1$ (остановка, далее процесс повторяется)

б. Циклическая подгруппа, сгенерированная на основе 3, — это $\mathbf{H_3}$, которая есть группа G.

$3^0 \bmod 10 = 1$
$3^1 \bmod 10 = 3$
$3^2 \bmod 10 = 9$
$3^3 \bmod 10 = 7$ (остановка, далее процесс повторяется)

в. Циклическая подгруппа, сгенерированная на основе 7, — это $\mathbf{H_3}$, которая есть группа G.

$7^0 \bmod 10 = 1$
$7^1 \bmod 10 = 7$
$7^2 \bmod 10 = 9$
$7^3 \bmod 10 = 3$ (остановка, далее процесс повторяется)

г. Циклическая подгруппа, сгенерированная на основе 9, — это $\mathbf{H_2}$. Подгруппа имеет только два элемента.

$9^0 \bmod 10 = 1$
$9^1 \bmod 10 = 9$ (остановка, далее процесс повторяется)

Циклические группы

Циклическая группа — группа, которая является собственной циклической подгруппой. В примере 4.7 группа G имеет циклическую подгруппу $\mathbf{H_5 = G}$. Это означает, что группа G — циклическая группа. В этом случае элемент, который генерирует циклическую подгруппу, может также генерировать саму группу. Этот элемент далее именуется «генератор». Если g — генератор, элементы в конечной циклической группе могут быть записаны как

$\{e, g, g^2, ..., g^{n-1}\}$, где $g^n = e$.

Заметим, что циклическая группа может иметь много генераторов.

Пример 4.9

а. Группа $G = <Z_6, +>$ — циклическая группа с двумя генераторами, $g = 1$ и $g = 5$.

б. Группа $G = <Z_{10}^*, \times >$ — циклическая группа с двумя генераторами, $g = 3$ и $g = 7$.

Теорема Лагранжа

Теорема Лагранжа показывает отношение порядка группы к порядку ее подгруппы. Предположим, что G — группа и H — подгруппа G. Если порядок G и H — |G| и |H|, соответственно, то согласно этой теореме |H| делит |G|. В примере 4.7 |G| = 6. Порядок подгруппы — $|H_1| = 1$, $|H_2| = 3$, $|H_3| = 2$ и $|H_4| = 6$. Очевидно, все эти порядки есть делители 6.

Теорема Лагранжа имеет очень интересное приложение. Когда дана группа G и ее порядок |G|, могут быть легко определены порядки потенциальных подгрупп, если могут быть найдены делители. Например, порядок группы $G = <Z_{17}, +>$ — это |17|. Делители 17 есть 1 и 17. Это означает, что эта группа может иметь только две подгруппы — нейтральный элемент и $H_2 = G$.

Порядок элемента

Порядок элемента в группе ord (a) (порядок (a)) является наименьшим целым числом *n*, таким, что $a^n = e$. Иными словами: порядок элемента — порядок группы, которую он генерирует.

Пример 4.10

a. В группе $G = <Z_6, +>$, порядки элементов: порядок ord(0) = 1, порядок ord (1) = 6, порядок ord (2) = 3, порядок ord (3) = 2, порядок ord (4) = 3, порядок ord (5) = 6.

b. В группе $G = <Z_{10}^*, \times >$, порядки элементов: порядок ord (1) = 1, порядок ord (3) = 4, порядок ord (7) =4, порядок (9) = 2.

Кольцо

Кольцо, обозначенное как $R = <\{...\}, \bullet, \square>$, является алгебраической структурой с двумя операциями. Первая операция должна удовлетворять всем пяти свойствам, требуемым для абелевой группы. Вторая операция должна удовлетворять только первым двум свойствам абелевой группы. Кроме того, вторая операция должна быть распределена с помощью первой. **Дистрибутивность** означает, что для всех *a, b* и *c* элементов из **R** мы имеем $a \square (b \bullet c) = (a \square b) \bullet (a \square c)$ и $(a \bullet b) \square c = (a \square c) \bullet (b \square c)$. Коммутативное кольцо — кольцо, в котором коммутативное свойство удовлетворено и для второй операции. Рисунок 4.4 показывает кольцо и коммутативное кольцо.

Дополнительное замечание

Кольцо включает две операции. Однако вторая операция может не соответствовать третьему и четвертому свойствам. Другими словами, первая операция — фактически операция пары операций, таких, как сложение и вычитание; вторая операция может содержать единственную операцию, например, умножение, но может не содержать деление.

Пример 4.11

Множество Z с двумя операциями — сложением и умножением — является коммутативным кольцом, которое обозначается **R = <Z,** +, × >. Сложение удовлетворяет всем пяти свойствам; умножение удовлетворяет только трем свойствам.

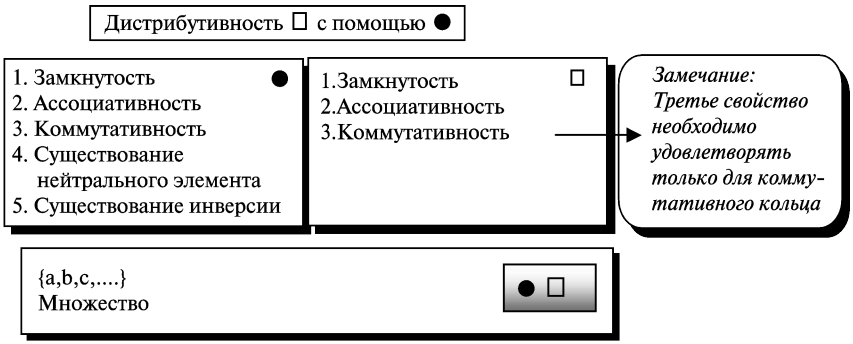


Рис. 4.4. Кольцо

Умножение дистрибутивно с помощью сложения. Например, 5 × (3 + 2) = (5 × 3) + (5 × 2) = 25. Но, мы можем выполнить на этом множестве сложение и вычитание и умножение, но не деление. Деление не может применяться в этой структуре, потому что оно приводит к элементу из другого множества. Результат деления 12 на 5 есть 2,4, и он не находится в заданном множестве.

Поле

Поле, обозначенное F = <{...}, •, □> — коммутативное кольцо, в котором вторая операция удовлетворяет всем пяти свойствам, определенным для первой операции, за исключением того, что нейтральный элемент первой операции (иногда называемый нулевой элемент) не имеет инверсии. Рисунок 4.5 показывает поле.

Дополнительное замечание

Поле — структура, которая поддерживает две пары операций, используемые в математике: сложение/вычитание и умножение/деление. Есть одно исключение: не разрешено деление на нуль.

Рис. 4.5. Поле

Конечные поля

Хотя общее определение касается полей бесконечного порядка, в криптографии используются экстенсивно только конечные поля. **Конечное поле** — поле с конечным числом элементов — является очень важной структурой в криптографии. Галуа показал, что поля, чтобы быть конечными, должны иметь число элементов p^n, где p — простое, а n — положительное целое число. Конечные поля обычно называют **полями Галуа** и обозначают как $GF(p^n)$.

Поле Галуа, $GF(p^n)$, — конечное поле с p^n элементами.

Поля GF (p)

Когда $n = 1$, мы получаем поле GF (p). Это поле может быть множеством *Zp,* (0, 1, ...p–1) с двумя арифметическими операциями (сложение и умножение). Любой элемент в этом множестве имеет аддитивную инверсию, и элементы, отличные от нуля, имеют мультипликативную инверсию (мультипликативная инверсия для 0 отсутствует).

Пример 4.12

Очень общее поле в этой категории — GF (2) с множеством {0,1}, двумя операциями, и умножением, как показано на рисунке 4.6.

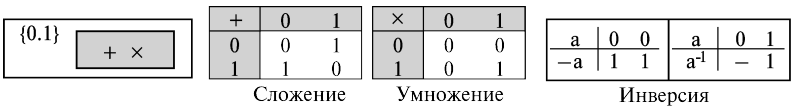


Рис. 4.6. *Поле GF (2)*

Есть несколько моментов, которые следует отметить в определении этого поля. Первый: множество имеет только два элемента, которые являются двоичными цифрами или битами (0 и 1). Второй: операция сложения — фактически ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR), операция, которую мы используем с двумя дво-

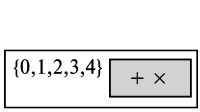
ичными цифрами. Третий: операция умножения — AND, операция, которую мы используем с двумя двоичными цифрами. Четвертый: сложение и операции вычитания — те же самые (операция XOR). Пятый: умножение и операции деления — те же самые (ОПЕРАЦИЯ AND).

Сложение/вычитание в GF(2) — операция ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR); умножение/деление — ОПЕРАЦИЯ И (AND).

Пример 4.13

Мы можем определить GF(5) на множестве Z_5 (5 простое) с операторами сложения и умножения, показанными на рис. 4.7.

Хотя есть возможность использовать расширенный алгоритм Евклида, чтобы найти мультипликативные инверсии элементов в GF(5), проще составить таблицу умножения и находить каждую пару, произведение которой равняется 1. Это (1, 1), (2, 3), (3, 2) и (4, 4). Заметим, что мы можем на этом множестве применить вычитание и умножение/деление (за исключением запрещенного деления на 0).



+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	3	4	0	1	3

Сложение

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Умножение

Аддитивная инверсия

a	0	1	2	3	4
-a	0	4	3	2	1

a	0	1	2	3	4
a^{-1}	-	1	3	2	4

Мультипликативная инверсия

Рис. 4.7. Поле GF (5)

Поля GF(p^n)

В дополнение к полям *GF(p)* в криптографии мы также интересуемся полями GF(p^n). Однако множества **Z,** Z_n, Z_{n^*} и Z_p, которые мы использовали до сих пор с операциями сложения и умножения, не могут удовлетворить требованиям поля. Поэтому должны быть определены некоторые новые множества и некоторые новые операции на этих множествах. В следующей секции мы рассматриваем очень полезное в криптографии поле GF(2^n).

Итоги рассмотренных структур

Изучение трех алгебраических структур позволяет нам использовать множества, в которых могут применяться операции, подобные сложению/вычитанию и

Таблица 4.3 Итоги определения алгебраических структур

Алгебраическая структура	Используемые операции	Используемые наборы целых чисел
Группа	(+ -) или (× ÷)	Z_n или Z_n*
Кольцо	(+ -) и (×)	Z
Поле	(+ -) и (× ÷)	Z_p

умножению/делению. Мы должны различать эти три структуры. Первая структура — группа, поддерживает одну пару связанных операций. Вторая структура — кольцо, поддерживает одну пару связанных операций и одну одиночную операцию. Третья структура — поле, поддерживает две пары операций. Таблица 4.3 поможет нам увидеть эту разницу.

4.2. Поля GF(2^n)

В криптографии мы часто должны использовать четыре операции (сложение, вычитание, умножение и деление). Другими словами, мы применяем поля. Однако когда мы работаем с компьютерами, положительные целые числа сохраняются в компьютере как n-битовые слова, в которых n является обычно 8, 16, 32, 64, и так далее. Это означает, что диапазон целых чисел — 0 до $2^n - 1$. Модуль — 2^n. Так что возможны два варианта, если мы хотим использовать поле.

1. Мы можем задействовать GF(p) с множеством Z_p, где p — наибольшее простое число, меньшее, чем 2^n. Но эта схема неэффективна, потому что мы не можем использовать целые числа от p до $2^n - 1$. Например, если $n = 4$, то наибольшее простое число, меньшее, чем 2^4, — это 13. Это означает, что мы не можем использовать целые числа 13, 14 и 15. Если $n = 8$, наибольшее простое число, меньшее, чем 2^8, — это 251, так что мы не можем взять 251, 252, 253, 254 и 255.
2. Мы можем работать в GF(2^n) и использовать множество 2^n элементов. Элементы в этом множестве — n-битовые слова. Например, если $n = 3$, множество равно:

{000, 001, 010,100,101,110,111}

Однако мы не можем интерпретировать каждый элемент как целое число от 0 до 7, потому что не могут быть применены обычные четыре операции (модуль 2^n — не простое число). Мы должны определить множество слов по 2 бита и две новых операции, которые удовлетворяют свойствам, определенным для поля.

Пример 4.14

Определим GF(2^2) поле, в котором множество имеет четыре слова по 2 бита: {00, 01, 10, 11}. Мы можем переопределить сложение и умножение для этого поля таким образом, чтобы все свойства этих операций были удовлетворены, как это показано на рис. 4.8.

⊕	00	01	10	11
00	**00**	01	10	11
01	01	**00**	11	10
10	10	11	**00**	01
11	11	10	01	**00**

Нейтральный элемент 00

⊗	00	01	10	11
00	**00**	00	00	00
01	00	**01**	10	11
10	00	10	**11**	**01**
11	00	11	**01**	**10**

Нейтральный элемент 01

Рис. 4.8. Пример поля GF(2^2)

Каждое слово — аддитивная инверсия себя. Каждое слово (кроме 00) имеет мультипликативную инверсию. Мультипликативные обратные пары — (01,01) и (10, 11). Сложение и умножение определено в терминах полиномов.

Полиномы

Хотя мы можем непосредственно определить правила для операций сложения и умножения слов из 2-х бит, которые удовлетворяют свойства в $GF(2^n)$, проще работать с полиномами степени $n - 1$ побитным представлением слов. **Полином** степени $n - 1$ имеет форму

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ...+ a_1x^1 + a_0x^0$$

где x^i назван термином «i-тый элемент», а a_i называется коэффициентом i- того элемента. Хотя мы знаем полиномы в алгебре, но при представлении n-битовых слов полиномами необходимо следовать некоторым особым правилам:

a. степень x определяет позицию бита в n-битовых слов. Это означает, что крайний левый бит находится в нулевой позиции (связан с x^0), самый правый бит находится в позиции $n-1$ (связан с x^{n-1});
b. коэффициенты сомножителей определяют значение битов. Каждый бит принимает только значение 0 или 1, поэтому наши полиномиальные коэффициенты могут иметь значение 0 или 1.

Пример 4.15

Использование полиномов для предоставления слова из 8 бит (10011001) показано на рис. 4.9.

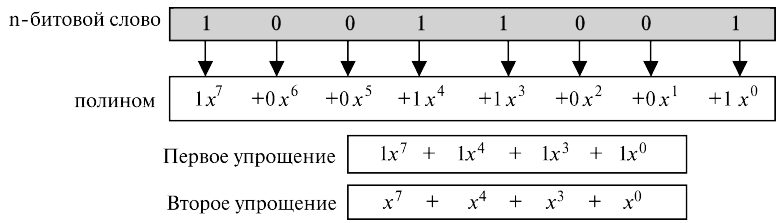


Рис. 4.9. Представление 8-ми битового слова полиномом

Заметим, что элемент полностью пропущен, если его коэффициент равен 0, и пропущен только коэффициент, если это 1. Также заметим, что элемент x^0 равен 1.

Пример 4.16

Чтобы найти слово на 8 битов, связанное с полиномом $x^5 + x^2 + x$, мы сначала восстановим пропущенные сомножители. Мы имеем $n = 8$, это означает полином степени 7. Расширенный полином имеет вид

$$0x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 1x^1 + 0x^0$$

Он связан со словом на 8 битов 00100110.

Операции

Обратите внимание, что любая операция на полиномах фактически включает две операции: операции над коэффициентами и операциии над двумя полиномами. Другими словами, мы должны определить два поля: одно для коэффициентов и одно для полиномов. Коэффициенты равны 0 или 1; для этой цели мы можем использовать GF(2)-поле. Мы уже говорили о таком поле (см. пример 4.14). Для полиномов нам нужно поле $GF(2^n)$, которое мы коротко обсудим ниже.

Полиномы, представляющие *n*-битовые слова, используют два поля: GF(2) и $GF(2^n)$.

Модуль

Перед определением операций на полиномах мы должны поговорить о полиномах-модулях. Сложение двух полиномов никогда не создает полином, выходящий из множества. Однако умножение двух полиномов может создать полином со степенью большей, чем $n - 1$. Это означает, что мы должны делить результат на модуль и сохранять только остаток, как мы делали в модульной арифметике. Для множеств полиномов в $\mathbf{GF}(2^n)$ группа полиномов степени n определена как модуль. Модуль в этом случае действует как *полиномиальное простое число*. Это означает, что никакие полиномы множества не могут делить этот полином. Простое полиномиальное число не может быть разложено в полиномы со степенью меньшей, чем n. Такие полиномы называются ***неприводимые полиномы***. Таблица 4.4 показывает примеры полиномов 1-5 степеней.

Для каждого значения степени часто есть более чем один неразлагаемый полином, — это означает, что когда мы определяем наш $GF(2^n)$, мы должны объявить, какой неприводимый полином мы используем как модуль.

Таблица 4.4. Список неприводимых полиномов

Степень	Неприводимый полином
1	$(x+1), x$
2	(x^2+x+1)
3	$(x^3+x^2+1), (x^3+x+1)$
4	$(x^4+x^3+x^2+x+1), (x^4+x^3+1), (x^4+x+1)$
5	$(x^5+x^2+1), (x^5+x^3+x^2+x+1), (x^5+x^4+x^3+x+1)$ $(x^5+x^4+x^3+x^2+1), (x^5+x^4+x^2+x+1)$

Сложение

Теперь определим операцию сложения для полиномов с коэффициентом в GF(2). Операция сложения очень простая: мы складываем коэффициенты соответствующих элементов полинома в поле GF(2). Обратите внимание, что сложение двух полиномов степени $n - 1$ всегда дает полином со степенью $n - 1$ — это означает, что мы не должны использовать вычитание модуля из результата.

Пример 4.17

Произведем сложение $(x^5 + x^2 + x) \oplus (x^3 + x^2 + 1)$ в GF(2^8). Мы используем символ $\oplus$ для обозначения полиномиального сложения. Ниже показана процедура:

$$0x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 1x^1 + 0x^0 \quad \oplus$$
$$0x^7 + 0x^6 + 0x^5 + 0x^4 + 1x^3 + 1x^2 + 0x^1 + 1x^0$$
$$0x^7 + 0x^6 + 1x^5 + 0x^4 + 1x^3 + 0x^2 + 1x^1 + 1x^0 \rightarrow x^5 + x^3 + x + 1$$

В упрощенном полиноме (показан справа) сохранены элементы с коэффициентом 1 и удалены элементы с коэффициентом 0. Кроме того, удалены совпадающие элементы обеих полиномов, а несовпадающие сохраняются. Другими словами, x^5, x^3 и x^1 сохраняются, а x^2, который является совпадающим в этих двух полиномах, удален.

Пример 4.18

Поскольку сложение в GF(2) означает операцию ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR), мы можем получить результат ИСКЛЮЧАЮЩЕГО ИЛИ для этих двух слов бит за битом. В предыдущем примере $x^5 + x^2 + x$ есть 00100110, или полином, и $x^3 + x^2 + 1$ есть 00001101. Результат — 00101011 или, в полиномиальном обозначении, $x^5 + x^3 + x + 1$.

Аддитивный нейтральный элемент — тождество. Аддитивный нейтральный элемент полинома — нулевой полином (полином со всеми коэффициентами, равными нулю), потому что, прибавляя этот полином к самому себе, в результате получаем нулевой полином.

Аддитивная инверсия полинома с коэффициентами в GF(2) — сам полином. Это означает, что операция вычитания та же самая, что и операция сложения.

Сложение и операции вычитания на полиномах — та же самая операция.

Умножение

Умножение в полиномах — сумма умножения каждого элемента одного полинома с каждым элементом второго полинома. Однако необходимо отметить три особенности.

Первая: умножение коэффициента проводится в поле GF(2).

Вторая: умножение x^i на x^j дает результат x^{i+j}.

Третья: умножение может создать элементы со степенью большей, чем n–1, и это означает, что результат должен быть уменьшен с применением полинома-модуля.

Сначала покажем, как умножить два полинома согласно вышеупомянутому определению. Позже будет дан более эффективный алгоритм, который может использоваться компьютерной программой.

Пример 4.19

Найдите результат $(x^5 + x^2 + x) \otimes (x^7 + x^4 + x^3 + x^2 + x)$ в GF(2^8) с неразлагаемым полиномом ($x^8 + x^4 + x^3 + x + 1$). Обратите внимание, что для обозначения умножения двух полиномов применен символ ⊗.

Решение

Сначала умножаем эти два полинома так, как мы это делали в обычной алгебре. Обратите внимание, что в этом процессе пара элементов с равной степенью удаляется. Например, результат $x^9 + x^9$ полностью удален, потому что он нулевой полином, по причине, которую мы обсуждали раньше при рассмотрении операции сложения.

$x^{12}+x^7+x^2$	$x^8+x^4+x^3+x+1$
$x^{12}+x^8+x^7+x^5+x^4$	x^4+1
$x^8+x^5+x^4+x^2$	
$x^8+x^4+x^3+x+1$	
$x^5+x^3+x^2+x+1$	Остаток

$P_1 \otimes P_2 = x^5 (x^7 + x^4 + x^3 + x^2 + x) + x^2 (x^7 + x^4 + x^3 + x^2 + x) + x (x^7 + x^4 + x^3 + x^2 + x)$

$P_1 \otimes P_2 = x^{12} + x^9 + x^8 + x^7 + x^6 + x^9 + x^6 + x^5 + x^4 + x^3 + x^8 + x^5 + x^4 + x^3 + x^2$

$P_1 \otimes P_2 = (x^{12} + -x^7 + x^2) \bmod (x^8 + x^4 + x^3 + x + 1) = x^5 + x^3 + x^2 + x + 1$

Чтобы найти конечный результат, разделим полином степени 12 на полином степени 8 (модуль) и сохраним только остаток. Процесс деления такой же, что и в обычной алгебре, но мы должны помнить, что здесь вычитание то же самое, что и сложение. Рисунок 4.10 показывает процесс деления.

Рис. 4.10. Полиномиальное деление с коэффициентами в поле GF (2)

Мультипликативное тождество — всегда равно 1. Например, в GF(2^8) мультипликативная инверсия — в побитном изображении 00000001.

Мультипликативная инверсия. Поиск мультипликативной инверсии требует привлечения расширенного алгоритма Евклида. Алгоритм Еклида должен быть применен к модулю и полиному, выполнение алгоритма является таким же, как и для целых чисел.

q	r_j	r_2	r	t_j	t_2	t
x^2+1	(x^4+x+1)	(x^2+1)	(x)	(0)	(1)	(x^2+1)
(x)	(x^2+1)	(x)	(1)	(1)	(x^2+1)	(x^3+x+1)
(x)	(x)	(1)	(0)	(x^2+1)	(x^3+x+1)	(0)
	(1)	(0)		(x^3+x+1)	(0)	

Пример 4.20

В GF(2^4) найдите инверсию $(x^2 + 1) \bmod (x^4 + x + 1)$.

Решение

Мы используем расширенный евклидов алгоритм, как это показано в таблице 4.5:

Таблица 4.5. Алгоритм Евклида для упражнения 4.20

Это означает, что $(x^2 + 1)^{-1} \bmod (x^4 + x + 1)$ есть $(x^3 + x + 1)$. Ответ может быть проверен просто: надо перемножить эти два полинома и найти остаток. В этом случае результат деления на модуль равен

$$[(x^2 + 1) \otimes (x^3 + x + 1)] \bmod (x^4 + x + 1) = 1$$

q	r_1	r_2	r	t_1	t_2	t
(x^3)	$(x^8+x^4+x^3+x+1)$	(x^5)	$(x^4+x^3+x+x+1)$	(0)	(1)	(x^3)
(x+1)	(x^5)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)	(x^4+x^3+1)
(x)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)	(x^4+x^3+1)	$(x^5+x^4+x^2+x)$
	(x^3+x^2+1)	(1)	(0)	(x^4+x^3+1)	$(x^5+x^4+x^2+x)$	(0)
	(1)	(0)		$(x^5+x^4+x^2+x)$	(0)	

Пример 4.21

В GF(2^8) найдите инверсию $(x^5) \bmod (x^8 + x^4 + x^3 + x + 1)$.

Решение

Будем использовать расширенный евклидов алгоритм, как это показано в Таблице 4.6:

Таблица 4.6. Евклидов алгоритм для примера 4.21

Это означает, что $(x^5)^{-1} \bmod (x^8 + x^4 + x^3 + x + 1)$ есть $(x^5 + x^4 + x^2 + x)$. Результат может быть легко проверен умножением этих двух полиномов и определением остатка деления по модулю.

$$[(x^5) \otimes (x^5 + x^4 + x^3 + x)] \bmod (x^8 + x^4 + x^3 + x + 1) = 1$$

Умножение, использующее компьютер

Операция деления порождает проблему написания эффективной программы умножения двух полиномов. Лучший алгоритм для компьютерной реализации использует неоднократное умножение уменьшенного полинома на *x*. Например, вместо того чтобы находить результат $(x^2 \otimes P_2)$, программа находит результат $(x \otimes (x \otimes P_2))$. Преимущества этой стратегии будет обсуждаться далее, но сначала рассмотрим пример, чтобы проиллюстрировать алгоритм.

Пример 4.22

Найдите результат умножения $P_1 = (x^5 + x^2 + x)$ на $P_2 = (x^7 + x^4 + x^3 + x^2 + x)$ в поле GF(2^8) с неприводимым полиномом $(x^8 + x^4 + x^3 + x + 1)$, используя алгоритм, изложенный выше.

Решение

Процесс показан в таблице 4.7. Мы сначала находим промежуточный результат умножения x^0, x^1, x^2, x^3, x^4 и x^5. Заметим, что необходимы только три со-

Таблица 4.7. Эффективный алгоритм умножения, использующий полиномы (пример 4.22)

Степень	Операция	Новый результат	Вычитание
$x^0 \otimes P_2$		$x^7+x^4+x^3+x^2+x$	Нет
$x^1 \otimes P_2$	$x \otimes (x^7+x^4+x^3+x^2+x)$	x^5+x^2+x+1	**ДА**
$x^2 \otimes P_2$	$x \otimes (x^5+x^2+x+1)$	$x^6+x^3+x^2+x$	Нет

ставляющие произведения $x^m \otimes P_2$ для m от 0 до 5, каждое вычисление зависит от предыдущего результата.

Рассмотренный выше алгоритм имеет два преимущества. Первое — умножение полинома на x может быть выполнено простым сдвигом одного бита в n-битовом слове; операция может быть реализована на любом языке программирования. Второе — результат может быть использован, если максимальная степень полинома n–1. В этом случае сокращение может быть сделано просто с помощью применения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с заданным модулем. В нашем примере самая высокая степень — только 8. Мы можем разработать простой алгоритм для нахождения промежуточных результатов.

1. Если старший разряд предыдущего результата равен 0, надо сдвинуть предыдущий результат на один бит влево.
2. Если старший бит предыдущего результата равен 1:

Степень	Операция сдвига влево	ИСКЛЮЧАЮЩЕЕ ИЛИ
$x^0 \otimes P_2$		10011110
$x^1 \otimes P_2$	00111100	(00111100) ⊕ (00011010) = **00100111**
$x^2 \otimes P_2$	01001110	**01001110**
$x^3 \otimes P_2$	10011100	10011100
$x^4 \otimes P_2$	00111000	(00111000) ⊕ (00011010) = 00100011
$x^5 \otimes P_2$	01000110	**01000110**
$P_1 \otimes P_2$ = (000100110) ⊕ (01001110) ⊕ (01000110) = 00101111		

   а. надо сдвинуть на один бит влево, и
   б. применить к нему операцию ИСКЛЮЧАЮЩЕЕ ИЛИ с модулем, исключив из этой операции старший разряд.

Повторим пример 4.22 для двоичной последовательности размером 8 бит.

Пусть P_1 = 000100110, P_2 = 10011110, модуль = 100011010 (девять битов). Обозначим операцию ИСКЛЮЧАЮЩЕЕ ИЛИ как ⊕. Пример приведен в таблице 4.8.

Таблица 4.8. Эффективное умножение с применением *n*-битового слова

В этом случае для умножения этих двух полиномов нам требуется только пять операций левого сдвига и четыре ИСКЛЮЧАЮЩЕЕ ИЛИ. Вообще, для ум-

ножения двух полиномов степени n-1 необходимо максимально (n–1) операций левого сдвига и 2n операций ИСКЛЮЧАЮЩЕЕ ИЛИ.

Умножение полиномов в GF(2^n) может быть выполнено с помощью операций левого сдвига и ИСКЛЮЧАЮЩЕЕ ИЛИ.

Пример 4.24

Поле GF(2^3) состоит из 8 элементов. Покажем умножение и сложение таблиц для этого поля, используя неприводимый полином x^3+x^2+1. Мы будем оперировать с трехбитовым словом и полиномом. Заметим, что имеется два полинома третьей степени (см. таблицу 4.4). Другой полином ($x^3 + x + 1$) для умножения имеет таблицу, полностью отличающуюся от первой. Таблица 4.9 показывает сложение. Затемненные клетки (нулевые) дают обратные пары для сложения.

Таблица 4.10 показывает умножение. Затемненные клетки (единичные) дают обратные пары при умножении.

⊕	000 **(0)**	001 **(1)**	010 **(x)**	011 **(x+1)**	100 **(x^2)**	101 **(x^2+1)**	110 **(x^2+ x)**	111 **(x^2+x+1)**
000 **(0)**	000 **(0)**	001 **(1)**	010 **(x)**	011 **(x+1)**	100 **(x^2)**	101 **(x^2+1)**	110 **(x^2+ x)**	111 **(x^2+x+1)**
001 **(1)**	001 **(1)**	000 **(0)**	011 **(x+1)**	010 **(x)**	101 **(x^2+1)**	100 **(x^2)**	111 **(x^2+ x+1)**	110 **(x^2+ x)**
010 **(x)**	010 **(x)**	011 **(x+1)**	000 **(0)**	001 **(1)**	110 **(x^2+ x)**	111 **(x^2+x+1)**	100 **(x^2)**	101 **(x^2+1)**
011 **(x+1)**	011 **(x+1)**	010 **(x)**	001 **(1)**	000 **(0)**	111 **(x^2+x+1)**	110 **(x^2+ x)**	101 **(x^2+1)**	100 **(x^2)**
100 **(x^2)**	100 **(x^2)**	101 **(x^2+1)**	110 **(x^2+ x)**	111 **(x^2+x+1)**	000 **(0)**	001 **(1)**	010 **(x)**	011 **(x^2+1)**
101 **(x^2+1)**	101 **(x^2+1)**	100 **(x^2)**	111 **(x^2+x+1)**	100 **(x^2+ x)**	001 **(1)**	000 **(0)**	011 **(x + 1)**	010 **(x)**
110 **(x^2+1)**	110 **(x^2+1)**	111 **(x^2+ x+1)**	100 **(x^2)**	101 **(x^2+1)**	010 **(x)**	011 **(x^2+1)**	000 **(0)**	001 **(1)**
111 **(x^2+x+1)**	111 **(x^2+x+1)**	110 **(x^2+ x)**	101 **(x^2+1)**	100 **(x^2)**	011 **(x^2+1)**	010 **(x)**	001 **(1)**	000 **(0)**

Использование генератора

Иногда проще определить элементы поля GF(2^n), используя генератор. В этом поле с неприводимым полиномом *f(x)* и элементом поля *a* нужно удовлетворить отношение *f(a)* = 0. В частности, если g — генератор поля, то *f(g)=0.* Тогда можно доказать, что элементы поля могут быть сгенерированы как

$$\{0, g, g, g^2, \ldots. g^n\}, \text{ где } N = 2^n - 2$$

Таблица 4.9. Сложение в поле GF(2^3)

Пример 4.25

Таблица 4.10. Умножение в поле GF(2^3)

⊗	000 **(0)**	001 **(1)**	010 **(*x*)**	011 **(*x*+1)**	100 **(*x*²)**	101 **(*x*²+1)**	110 **(*x*²+ *x*)**	111 **(*x*²+*x*+1)**
000 **(0)**	000 **(0)**	000 **(0)**	000 **(0)**	000 **(0)**	000 **(0)**	000 **(0)**	000 **(0)**	000 **(0)**
001 **(1)**	000 **(0)**	001 **(1)**	010 **(*x*)**	011 **(*x*+1)**	100 **(*x*²)**	101 **(*x*²+1)**	110 **(*x*²+ *x*)**	111 **(*x*²+*x*+1)**
010 **(*x*)**	000 **(0)**	010 **(*x*)**	100 **(*x*²)**	110 **(*x*²+ *x*)**	101 **(*x*²+1)**	111 **(*x*²+*x*+1)**	001 **(1)**	011 **(*x*+1)**
011 **(*x*+1)**	011 **(*x*+1)**	011 **(*x*+1)**	110 **(*x*²+ *x*)**	101 **(*x*²+1)**	001 **(1)**	010 **(*x*)**	111 **(*x*²+*x*+1)**	100 **(*x*²)**
100 **(*x*²)**	000 **(0)**	100 **(*x*²)**	101 **(*x*²+1)**	001 **(1)**	111 **(*x*²+*x*+1)**	011 **(*x*+1)**	010 **(*x*)**	110 **(*x*²+ *x*)**
101 **(*x*²+1)**	000 **(0)**	101 **(*x*²+1)**	111 **(*x*²+*x*+1)**	010 **(*x*)**	011 **(*x*+1)**	110 **(*x*²+ *x*)**	100 **(*x*²)**	001 **(1)**
110 **(*x*²+*x*)**	000 **(0)**	110 **(*x*²+*x*)**	001 **(1)**	111 **(*x*²+*x*+1)**	010 **(*x*)**	100 **(*x*²)**	011 **(*x*+1)**	101 **(*x*²+1)**
111 **(*x*²+*x*+1)**	000 **(0)**	111 **(*x*²+*x*+1)**	011 **(*x*+1)**	100 **(*x*²)**	110 **(*x*²+ *x*)**	001 **(1)**	101 **(*x*²+1)**	010 **(*x*)**

Для генерирования элементов поля GF(2^4) используйте полином $f(x) = x^4 + x + 1$.

Решение

Элементы 0, g^0, g^1, g^2 и g^3 могут быть сгенерированы достаточно просто, потому что в 4-битовом поле они представлены 0, x^0, x^1, x^2 и x^3 (не требуется деления на полином). Элементы от g^4 до g^{14}, которые представляют от x^4 до x^{14}, нужно разделить на неприводимый полином. Для такого деления можно применить полином $f(g) = g^4 + g + 1 = 0$. Применив это отношение, мы имеем $g^4 = -g - 1$. поскольку сложение полей и вычитание полей — та же самая операция, $g^4 = g + 1$. Мы используем это отношение, чтобы найти значение всех элементов в виде 4-битовых слов:

0	= 0	= 0	= 0	→	0 = (0000)
g^0	= g^0	= g^0	= g^0	→	g^0 = (0001)
g^1	= g^1	= g^1	= g^1	→	g^1 = (0010)
g^2	= g^2	= g^2	= g^2	→	g^2 = (0100)
g^3	= g^3	= g^3	= g^3	→	g^3 = (1000)
g^4	= g^4	= g^4	= g +1	→	g^4 = (0011)
g^5	= g (g^4)	= g (g +1)	= g^2 + g	→	g^5 = (0110)
g^6	= g (g^5)	= g (g^2 + g)	= g^3+ g^2	→	g^6 = (1100)
g^7	= g (g^6)	= g (g^3+ g)	= g^3+ g + 1	→	g^7 = (1011)
g^8	= g (g^7)	= g (g^3+ g + 1)	= g^2+1	→	g^8 = (0101)
g^9	= g (g^8)	= g (g^2+1)	= g^3+ g	→	g^9 = (1010)
g^{10}	= g (g^9)	= g (g^3+ g)	= g^2+ g + 1	→	g^{10} = (0111)
g^{11}	= g (g^{10})	= g (g^2+ g + 1)	= g^3+ g^2 + g	→	g^{11} = (1110)
g^{12}	= g (g^{11})	= g (g^3+ g^2 + g)	= g^3+ g^2 + g + 1	→	g^{12} = (1111)
g^{13}	= g (g^{12})	= g (g^3+ g^2 + g + 1)	= g^3+ g^2 + 1	→	g^{13} = (1101)

$g^{14} = g(g^{13}) = g(g^3 + g^2 + 1) = g^3 + 1 \rightarrow g^{14} = (1001)$

Главная идея состоит в том, что вычисление элементов поля от g^4 до g^{14} сводится к использованию соотношения $g^4 = g + 1$ и предыдущих вычислений. Например,

$$g^{12} = g(g^{11}) = g(g^3 + g^2 + g) = g^4 + g^3 + g^2 = g^3 + g^2 + g + 1$$

После сокращения можно просто преобразовать степени в n-битовое слово. Скажем, $g^3 + 1$ эквивалентно 1001, потому что присутствуют элементы со степенью 0 и 3. Заметим, что элементы с одинаковой степенью при таком процессе вычисления отменяют друг друга. Например, $g^2 + g^2 = 0$.

Инверсии

Нахождение инверсий при использовании приведенного выше метода представления достаточно просто.

Аддитивные инверсии

Аддитивная инверсия каждого элемента — элемент непосредственно, потому что сложение и вычитание в этом поле — одна и та же операция, $g^3 = g^3$.

Мультипликативные инверсии

Найти мультипликативную инверсию каждого элемента также очень просто. Например, может найти мультипликативную инверсию элемента g^3, как показано ниже:

$$(g^3)^{-1} = g^{-3} = g^{12} = g^3 + g^2 + g + 1 \rightarrow (1111)$$

Заметим, что в этом случае степень рассчитывается по модулю $2^n - 1$, $2^4 - 1 = 15$.

Поэтому $-3 \bmod 15 = 12 \bmod 15$.

Можно легко доказать, что g^3 и g^{12} есть инверсные (обратные числа), потому что $g^3 \times g^{12} = g^{15} = g^0 = 1$.

Сложение и вычитание

Сложение и вычитание — это одинаковые операции. Промежуточные результаты могут быть упрощены, как проиллюстрировано в следующем примере.

Пример 4.26

Этот пример показывает результаты операций сложения и вычитания:

a. $(g^3 + g^{12} + g^7) = g^3 + (g^3 + g^2 + g + 1) + (g^3 + g + 1) = g^3 + g^2 \rightarrow (1100)$

b. $g^3 - g^6 = g^3 + g^6 = g^3 + (g^3 + g^2) = g^2 \rightarrow (0100)$

Умножение и деление

Умножение есть сложение степени по модулю $2^n - 1$. Деление — это умножение, которое использует мультипликативную инверсию.

Пример 4.27
Ниже показаны операции умножения и деления:
а. $g^9 \times g^{11} = g^{20} = g^{20 \bmod 15} = g^5 = g^2 + g \rightarrow (0110)$
б. $g^3 / g^8 = g^3 \times g^7 = g^{10} = g^2 + g + 1 \rightarrow (0111)$

Итоги раздела конечные поля

Конечное поле GP (2^n) может использоваться для того, чтобы определить четыре операции — сложение, вычитание, умножение и деление n-битных слов. Только деление на нуль не определено. Каждое n-битовое слово может быть представлено как полином степени $n - 1$ с коэффициентами в GF(2), — это означает, что операции на n-битовых словах могут быть представлены как операции на этом полиноме. При умножении двух полиномов необходимо сделать эти операции операциями по модулю. Для этого мы должны определить неприводимый полином степени n. Чтобы найти мультипликативные инверсии к полиномам, может быть применен расширенный алгоритм Евклида.

4.3. Рекомендованная литература

Нижеследующие книги и сайты обеспечивают более детальное рассмотрение понятий, обсужденных в этой лекции.

Книги

[Dur05], [Ros06J], [Bla03], [BW00] и [DF04] основательно рассматривают алгебраические структуры.

Сайты

Нижеследующие сайты дают больше информации о темах, обсуждаемых в этой лекции.

http://en.wikipedia.org/wiki/Algebraic_structure
http: // en.wikipedia.org/wiki/Ring _ % 28mathematics%29
http://en.wikipedia.org/wiki/Polynomials
http: // www.math.niu.edu / ~ rusin/known-math/index/20-XX.html
http: // www.math.niu.edu / ~ rusin/known-math/index/13-XX.html
http://www.hypermaths.org/quadibloc/math/abaint.htm
http://en.wikipedia.org/wiki/Finite_field

4.4. Итоги

- Криптография требует заданных множеств и операций, определенных на этих множествах. Комбинации множеств и операций, приложенных к элементам этих множеств, есть алгебраическая структура. Были введены три алгебраических структуры: группы, кольца и поля.

- Группа — алгебраическая структура с бинарной операцией, удовлетворяющая четырем свойствам: замкнутость, ассоциативность, существование тождества (единичного элемента) и существование инверсии. Коммутативная группа, также называемая абелевой группой, — группа, оператор которой удовлетворяет дополнительному свойству: коммутативности.
- Подмножество H группы G — подгруппа G, если само H является группой с соответствующими операциями на G. Если подгруппа группы может быть сгенерирована, используя степень элемента, то она называется циклической подгруппой. Циклическая группа — это собственная циклическая подгруппа группы.
- Теорема Лагранжа связывает порядок группы и порядок ее подгруппы. Если порядок групп G и H соответственно |G| и |H|, тогда |H| делит |G|.
- Порядок элемента a в группе — наименьшее положительное целое число n, такое, что $a^n = e$ (единичному элементу).
- Кольцо — алгебраическая структура с двумя операциями. Первая операция должна удовлетворять всем пяти свойствам, требуемым для абелевой группы. Вторая операция должна удовлетворять только первым двум. Кроме того, вторая операция должна быть относительно первой дистрибутивной. Коммутативное кольцо — это кольцо, в котором вторая операция удовлетворяет свойству коммутативности.
- Поле — коммутативное кольцо, в котором вторая операция удовлетворяет пяти свойствам, определенным для первой операции, за одним исключением: единичный элемент первой операции не имеет инверсии. Конечное поле, также называемое полем Галуа, — поле с элементами p^n, где p — простое число, а n — положительное целое число. $GF(p^n)$ поля используется в операциях на n-битовых словах в криптографии.
- Для того чтобы представить n-битовые слова, используются полиномы с коэффициентами в GF(2). Сложение и умножение n-битовых слов могут быть определены как сложение и умножение полиномов.

Иногда проще определить элементы $GF(2^n)$-поля, используя генератор. Если g — генератор поля, то $f(g) = 0$. Нахождение инверсий и выполнение операций на элементах поля становятся более простыми, когда элементы представлены как степени генератора поля.

4.5. Вопросы и упражнения

Обзорные вопросы

1. Определите алгебраическую структуру и назовите три алгебраических структуры, обсужденные в этой лекции.
2. Определите группу и приведите различия между группой и коммутативной группой.
3. Определите кольцо и приведите различия между кольцом и коммутативным кольцом.
4. Определите поле и приведите различия между бесконечным полем и конечным полем.

5. Покажите число элементов в поле Галуа для простого числа.
6. Дайте один пример группы, использующей множество вычетов (операций по модулю).
7. Дайте один пример кольца, использующего множество вычетов (операций по модулю).
8. Дайте один пример поля, использующего множество вычетов (операций по модулю).
9. Покажите, как полином может представить *n*-битовое слово.
10. Определите неприводимый полином.

Упражнения

1. Для группы $G = <Z_4, +>$:
   а. Докажите, что это — абелева группа.
   б. Покажите результат операций 3 + 2 и 3–2 в этой группе.
2. Для группы $G = <Z_6 *, \times >$:
   а. Докажите, что это — абелева группа.
   б. Покажите результат операций 5×1 и $1 \div 5$.
   с. Объясните, почему мы не должны беспокоиться о делении на нуль в этой группе.
3. В таблице 4.1 для группы была определена только одна операция. Предположим, что эта операция — сложение. Покажите таблицу для операции вычитания (обратная операция).
4. Докажите, что перестановка в группе, показанной в таблице 4.2, не является коммутативной.
5. Докажите, что перестановка в группе, показанной в таблице 4.2, частично, в нескольких случаях, удовлетворяет свойству ассоциативности.
6. Создайте таблицу перестановки для двух входов и двух выходов, подобных показанным в таблице 4.2.
7. Алиса применяет три последовательных перестановки — [1 3 2], [3 2 1] и [2 1 3]. Покажите, как Боб может использовать только одну перестановку, чтобы изменить процесс на противоположный. Пользуйтесь таблицей 4.2.
8. Найдите все подгруппы следующих групп:
   а. $\mathbf{G} = <Z_{16}, +>$
   б. $\mathbf{G} = <Z_{23}, +>$
   в. $\mathbf{G} = <Z_{16} *, \times >$
   г. $\mathbf{G} = <Z_{17} *, \times >$
9. Используя теорему Лагранжа, найдите порядок всех потенциальных подгрупп для следующих групп:
   а. $\mathbf{G} = <Z_{18}, +>$
   б. $\mathbf{G} = <Z_{29}, +>$
   в. $\mathbf{G} = <Z_{12} *, \times >$
   г. $\mathbf{G} = <Z_{19} *, \times >$
10. Найдите порядок всех элементов в следующих группах:
    а. $\mathbf{G} = <Z_8, +>$

б. $\mathbf{G} = \langle Z_7, + \rangle$
в. $\mathbf{G} = \langle Z_9 *, \times \rangle$
г. $\mathbf{G} = \langle Z_7 *, \times \rangle$

11. Повторите пример 4.25, используя неприводимый полином $f(x) = x^4 + x^3 + 1$.
12. Повторите пример 4.26, используя неприводимый полином $f(x) = x^4 + x^3 + 1$.
13. Повторите пример 4.25, используя неприводимый полином $f(x) = x^4 + x^3 + 1$.
14. Какое выражение из нижеследующих является правильным полем Галуа?
    а. GF(12)
    б. GF(13)
    в. GF(16)
    г. GF(17)
15. Для каждого из следующих n-битовых слов найдите полиномы, которые представляют эти слова.
    а.10010
    б. 10
    в. 100001
    г. 00011
16. Найдите n-битовое слово, которое представлено каждым из следующих полиномов:
    а. $x^2 + 1$ в GF(2^4)
    б. $x^2 + 1$ в GF(2^5)
    в. $x + 1$ в GF(2^3)
    г. x^7 в GF(2^8)
17. В поле GF(7) найдите результат следующих операций:
    а. 5+3
    б. 5–4
    в. 5 × 3
    г. 5 ÷ 3
18. Докажите, что (x) и $(x + 1)$ — неприводимые полиномы степени 1.
19. Докажите, что $(x^2 + x + 1)$ — неприводимый полином степени 2.
20. Докажите, что $(x^3 + x^2 + 1)$ — неприводимый полином степени 3.
21. Умножьте следующие 2-битовые слова, используя полиномы:
    а. (11) × (10)
    б. (1010) × (1000)
    в. (11100) × (10000)
22. Найдите мультипликативную инверсию следующих полиномов в GF(2^2). (Заметим, что для этого поля есть только один модуль.)
    а. 1
    б. x
    в. x + 1
23. Используйте расширенный евклидов алгоритм, чтобы найти инверсию ($x^4 + x^3 + 1$) в GF(2^5), применяя модуль ($x^5 + x^2 + 1$).
24. Создайте таблицу сложения и умножения для GF(2^4), используя $(x^4 + x^3 + 1)$ как модуль.
25. Используя таблицу 4.10, выполните следующие операции:
    а. (100) ÷ (010)

Лекция 5. Введение в основы современных шифров с симметричным ключом

Цели и содержание

В этой лекции поставлено несколько целей.

- Показать различие между традиционными и современными шифрами с симметричным ключом.
- Привести современные блочные шифры и обсудить их характеристики.
- Объяснить, почему современные блочные шифры должны быть спроектированы как шифры подстановки.
- Ввести компоненты блочных шифров, таких, как P-блоки и S-блоки.
- Обсудить и показать различие между двумя классами шифров: шифры Файстеля и шифры не-Файстеля.
- Обсудить два вида атак, особо направленных на раскрытие современных блочных шифров: дифференциальный и линейный криптоанализ.
- Ввести понятие «шифры для потока» и показать различие между синхронными и несинхронными шифрами.
- Обсудить линейную и нелинейную обратную связь регистров сдвига для реализации поточных шифров.

Традиционные шифры с симметричным ключом, которые мы изучали до сих пор, ориентируются на символы. С появлением компьютера стали необходимы шифры, ориентированные на бит. Потому что информация, которую надо зашифровать, — не всегда только текст; она может также состоять из чисел, графики, аудио- и видеоданных. Удобно преобразовать эти типы данных в поток битов, чтобы зашифровать этот поток и затем передать зашифрованный поток. Кроме того, когда текст обработан на разрядном уровне, каждый символ заменен на 8 (или 16) бит, а это означает, что число символов становится в 8 (или 16) раз больше. Смешивание большего числа символов увеличивает безопасность.

Эта лекция обеспечивает необходимую основу для изучения современных блочных и поточных шифров, которые рассматриваются в следующих трех лекциях. Большая часть этой лекции посвящена обсуждению общих идей современных блочных шифров, и только малая часть — принципам современных поточных шифров.

5.1. Современные блочные шифры

Современный блочный шифр с симметричными ключами шифрует n-битовый блок исходного текста или расшифровывает n-битовый блок зашифрованного текста. Алгоритм шифрования или дешифрования используют k-битовый ключ. Алгоритм дешифрования должен быть инверсией алгоритма шифрования, и оба в работе используют один и тот же ключ засекречивания так, чтобы Боб мог восстановить сообщение, передаваемое Алисой. Рисунок 5.1 показывает общую идею шифрования и дешифрования в современном блочном шифре.

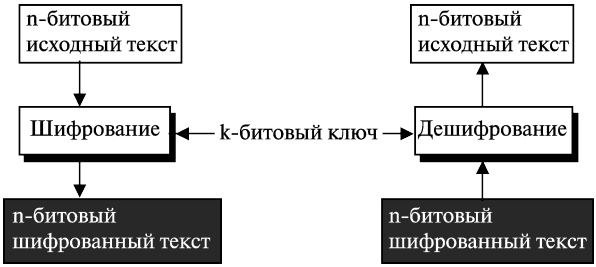


Рис. 5.1. Современный блочный шифр

Если сообщение имеет размер меньше, чем *n* бит, нужно добавить заполнение, чтобы создать этот n-разрядный блок; если сообщение имеет больше, чем *n* бит, оно должно быть разделено на *n*-разрядные блоки, и в случае необходимости нужно добавить к последнему блоку соответствующее заполнение. Общие значения для *n* обычно 64, 128, 256 или 512 битов.

Пример 5.1

Сколько дополнительных битв нужно добавить к сообщению 100 символов, если для кодирования используется ASCII по 8 битов и блочный шифр принимает блоки 64 бита?

Решение

Закодировать 100 символов, используя ASCII по 8 битов. Это сообщение содержит 800 бит. Исходный текст должен делиться без остатка на 64. Если | M | и | Pad | — длина сообщения и длина заполнения, то

$$| m | + | Pad | = 0 \bmod 64 \rightarrow | Pad | = -800 \bmod 64 \rightarrow 32 \bmod 64$$

Это означает, что нужно к сообщению нужно добавить 32 бита заполнения (например, нулей). Текст тогда будет состоять из 832 битов или тринадцати 64-разрядных блоков. Заметим, что только последний блок содержит заполнение. Шифратор использует алгоритм шифрования тринадцать раз, чтобы создать тринадцать блоков зашифрованного текста.

Подстановка, или транспозиция

Современный блочный шифр может быть спроектирован так, чтобы действовать как шифр подстановки или как шифр транспозиции[1]. Это — та же самая идея, которая используется и в традиционных шифрах, за исключением того, что символы, которые будут заменены или перемещены, содержат биты вместо символов.

Если шифр спроектирован как *шифр подстановки*, значения бита 1 или 0 в исходном тексте могут быть заменены либо на 0, либо на 1. Это означает, что ис-

[1] Транспозиция – перестановка элементов множества X, которая меняет местами только два элемента *(прим. пер.)*

ходный текст и зашифрованный текст могут иметь различное число единиц. Блок исходного текста на 64 бита, который содержит 12 нулей и 52 единицы, может быть представлен в зашифрованном тексте 34 нулями и 30 единицами. Если шифр спроектирован как *шифр перестановки* (*транспозиции*), биты только меняют порядок следования (перемещаются), сохраняя то же самое число символов в исходном и зашифрованном текстах. В любом случае, число возможных *n*-битовых исходных текстов или зашифрованных текстов равно 2^n, потому что каждый из *n* битов, использованных в блоке, может иметь одно из двух значений — 0 или 1.

Современные блочные шифры спроектированы как шифры подстановки, потому что свойства транспозиции (сохранение числа единиц или нулей) делают шифр уязвимым к атакам исчерпывающего поиска, как это показывают нижеследующие примеры.

Пример 5.2

Предположим, что мы имеем блочный шифр, где $n = 64$. Если есть 10 единиц в зашифрованном тексте, сколько испытаний типа «проб и ошибок» должна сделать Ева, чтобы получить исходный текст перехваченного зашифрованного текста в каждом из следующих случаев?

a. Шифр спроектирован как шифр подстановки.

b. Шифр спроектирован как шифр транспозиции.

Решение

a. В первом случае (подстановка) Ева понятия не имеет, сколько единиц находится в исходном тексте. Ева должна попробовать все возможные 2^{64} блока по 64 бита, чтобы найти один, который имеет смысл. Если бы Ева могла пробовать 1 миллиард блоков в секунду, то и тогда ей потребовалось бы сотни лет, прежде чем эта работа могла бы принести успех.

b. Во втором случае (перестановка) Ева знает, что в исходном тексте есть точно 10 единиц, потому что транспозиция не изменяет числа единиц (или нулей) в зашифрованном тексте. Ева может начать атаку исчерпывающего поиска, используя только те 64-битовые блоки, которые имеют точно 10 единиц. Есть только (64!) / [(10!) (54!)] = 151 473 214 816 из 2^{64} слов по 64 бита, которые имеют точно 10 единиц. Ева может проверить всех их меньше чем за 3 минуты, если она может провести 1 миллиард испытаний в секунду.

Стойкий к атаке исчерпывающего поиска современный блочный шифр должен быть спроектирован как шифр подстановки.

Блочные шифры как групповые математические перестановки[1]

Как мы увидим в следующих лекциях, нам надо знать, является ли современный блочный шифр математической группой (см. лекцию 4). Чтобы ответить

[1] Перестановка – это упорядоченный набор чисел 1,2,3,.....,n. При этом n называется порядком перестановки. Число перестановок порядка n равно $n! = 1 \times 2 \times 3 \times ... \times n$.

на этот вопрос, сначала предположим, что ключ достаточно длинный, чтобы создать отображение любой возможной входной информации в выходную. Он называется *полноразмерным ключевым шифром*. На практике, ключ бывает меньше; длинный ключ можно применять только для некоторых отображений входной информации в выходную. Хотя блочный шифр должен иметь ключ, который является секретным при обмене между передатчиком и приемником, в шифре используются также компоненты, которые не зависят от ключа.

Полноразмерные ключевые шифры

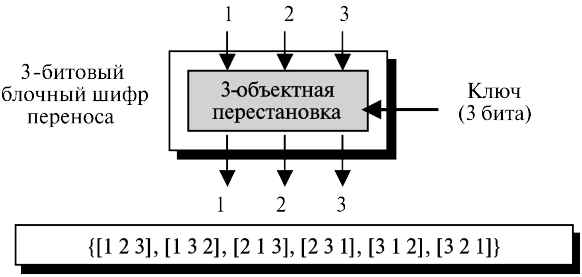
Хотя полноразмерные ключевые шифры сейчас практически не применяются, мы сначала поговорим о них, чтобы сделать более понятным обсуждение шифров с ключом частичного размера.

Полноразмерные ключевые блочные шифры транспозиции. Такой ключевой шифр перемещает биты, не изменяя их значения, так что может быть смоделирован как перестановка *n*-мерного объекта с множеством n! таблиц перестановки, в которых ключ определяет, какая таблица используется Алисой и Бобом. Мы должны иметь $n!$ возможных ключей, и такой ключ должен иметь длину $[\log_2 n!]$ бит.

Пример 5.3

Покажите модели и множество таблиц перестановки для блочного шифра транспозиции на 3 бита, где размер блока — 3 бита.

Решение



Множество таблиц перестановки с 3! = 6 элементов

Множество таблиц перестановки имеет 3! = 6 элементов, как показано на рис. 5.2. Ключ должен быть длиной $[\log_2 n!] = 3$ бита. Заметим, что хотя ключ на 3 бита может выбрать $2^3 = 8$ различных отображений, мы используем только 6 из них.

Рис. 5.2. Блочный шифр транспозиции в виде перестановки

Полноразмерные ключевые блочные шифры подстановки. Такие шифры не перемещают биты — они заменяют биты. На первый взгляд кажется, что полноразмерный ключевой шифр подстановки не может быть смоделирован как перестановка. Однако мы можем применить модель перестановки для шифра подстановки, если сможем декодировать входную информацию и кодировать выходную.

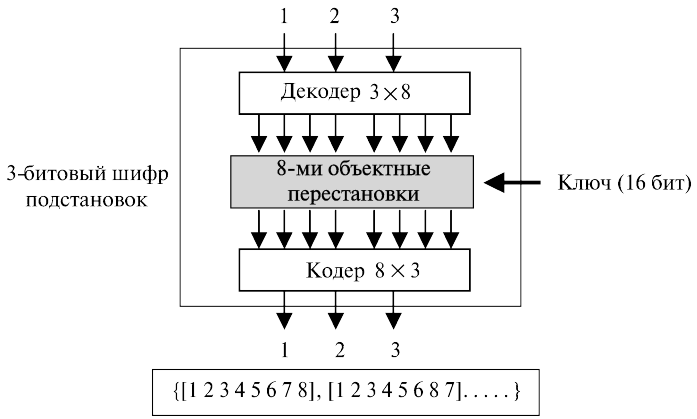
Декодирование здесь означает преобразование n-разрядного целого числа в строку 2^n-бит с единственной единицей 1 и 2^n–1 нулями. Позиция единственной единицы указывает значение целого числа в упорядоченной последовательности позиций строки от 0 до $2^n - 1$. Поскольку новая входная информация имеет всегда единственную единицу, шифр может быть смоделирован как перестановка $2^n!$ объектов.

Пример 5.4

Покажите модель и множество таблиц перестановки для шифра подстановки блока на 3 бита.

Решение

Три входных исходных текста могут быть обозначены целыми числами от 0 до 7. Это может быть закодировано как строка, содержащая 8 битов с единственной единицей. Например, комбинация 000 может быть закодирована как 00000001 (первая единица справа); комбинация 101 может быть закодирована как 00100000 (шестая единица справа). Рисунок 5.3 показывает модель и множество таблиц перестановки. Заметим, что число элементов в закодированном множестве намного



Множество таблиц перестановки с 8! = 40 320 элементов

больше, чем число элементов в шифре транспозиции (8! = 40 320). Ключ — также намного более длинный $[\log_2 40320] = 16$ бит. Хотя ключ на 16 битов может определить 65 536 различных отображений, используются только 40 320.

Рис. 5.3. Блочный шифр подстановки моделируется как шифр перестановки

Полноразмерный ключ — это n-разрядный шифр транспозиции или блочный шифр подстановки. Они могут быть смоделированы как шифры перестановки, но размеры их ключа различны:
для шифра транспозиции ключ длиной — $[\log_2 n!]$,
для шифра подстановки ключ длиной — $[\log_2 (2^n)!]$.

Групповая перестановка. Факт, что полноразмерная ключевая транспозиция или шифр подстановки/перестановки показывает, что если шифрование (или дешифрование) использует больше чем одну любую комбинацию из этих шифров, результат эквивалентен операции групповой перестановки. Как уже обсуждалось в лекции 4, две или больше каскадных перестановки могут всегда быть заменены единственной перестановкой. Это означает, что бесполезно иметь больше чем один каскад полноразмерных ключевых шифров, потому что эффект тот же самый, как и при наличии единственного шага.

Шифры ключа частичного размера

Фактические шифры не могут использовать полноразмерные ключи, потому что размер ключа становится несуразно большим, особенно для блочного шифра подстановки. Например, общий шифр подстановки – DES (см. лекцию 6) — применяет 64-разрядный блочный шифр. Если бы проектировщики DES применяли полноразмерный ключ, он был бы $\log_2 (2^{64}!) = 2^{70}$ битов. На практике ключ для DES — только 56 битов, что является очень маленьким фрагментом полноразмерного ключа. Это означает, что DES использует только 2^{56} отображений из приблизительно $2^{2^{70}}$ возможных отображений.

Группа перестановки. Зададим себе вопрос: можно ли установить, что многоступенчатая транспозиция с частичным ключом или подстановка — это группа перестановки с композицией операций? Ответ на этот вопрос чрезвычайно важен, потому что он говорит нам о том, является ли многоступенчатая версия с частичным шифром таким же средством шифрования, как и сам шифр. Этот факт позволяет достигнуть большей степени безопасности (см. обсуждение многократной DES в лекции 6).

Частичный ключевой шифр – это группа, если это — подгруппа соответствующего размера ключа шифра. Другими словами, если полноразмерный ключевой шифр — это группа G = <M, o>, где M. — множество отображений и (o) — композиция операций, то шифр с ключом частичного размера должен представлять подгруппу H = <N, o>, где N — подмножество M с теми же самыми операциями.

Например, было доказано, что многоступенчатый DES с 56-битовым ключом не является группой, потому что подгруппа с 2^{56} отображениями не может быть создана из группы с 2^{64}! отображениями.

Частичный ключевой шифр есть группа с набором операций, если он является подгруппой соответствующего полноразмерного ключевого шифра.

Шифры без ключа

Хотя использование отдельно шифра без ключа фактически бесполезно, возможно их применение в качестве компонентов ключевых шифров.

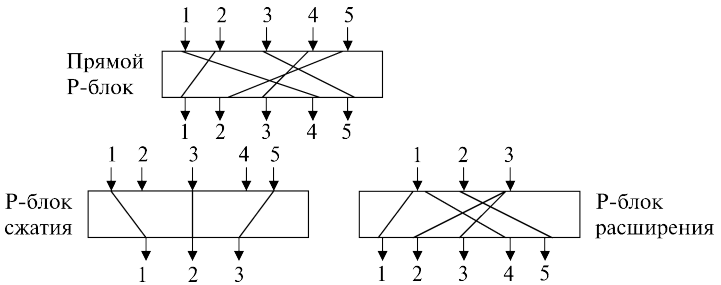
Шифр транспозиции без ключа. Шифры без ключа (или с фиксированным ключом) можно рассматривать как шифр транспозиции, реализованный в аппаратных средствах. Фиксированный ключ (единственное правило перестановки)

может быть представлен как таблица в случае реализации шифра в программном обеспечении. Следующая часть этой лекции обсуждает шифры транспозиции без ключа, названные P-блоками, которые используются как стандартные блоки современных блочных шифров.

Шифры подстановки без ключа. Такой шифр без ключа (или с фиксированным ключом) можно представить себе как заранее определенное отображение входной информации к выходной. Отображение может быть представлено как таблица, как математическая функция, а также другими способами. В следующей части этой лекции рассматриваются шифры подстановки без ключей, названные S-блоками, которые применяются как стандартные блоки современных блочных шифров.

Компоненты современного блочного шифра

Современные блочные шифры обычно являются ключевыми шифрами подстановки, в которых ключ позволяет только частичные отображения возможных входов информации в возможные выходы. Однако эти шифры обычно не проектируются как единый модуль. Чтобы обеспечивать требуемые свойства современного блочного шифра, такие, как рассеяние и перемешивание информации (обсуждается кратко), этот шифр формируется как комбинация модулей транспозиции (называемых P-блоками), модулей подстановки (называемых S-блоками) и некоторыми другими модулями (обсуждается кратко).



P-блок (блок перестановки) подобен традиционному шифру транспозиции символов. Он перемещает биты. В современных блочных шифрах мы можем найти три типа P-блоков: прямые P-блоки, P-блоки расширения и P-блоки сжатия, что и показано на рис. 5.4.

Рис. 5.4. Три типа P-блоков

Рисунок 5.4 показывает прямой P-блок 5 × 5, P-блок сжатия 5 × 3 и P-блок расширения 3 × 5. Рассмотрим каждый из них более подробно.

Прямые P-блоки. Прямой P-блок с n входами и n выходами – это перестановка с $n!$ возможными отображениями.

Пример 5.5

Рисунок 5.5 показывает все 6 возможных отображений P-блока 3 × 3.

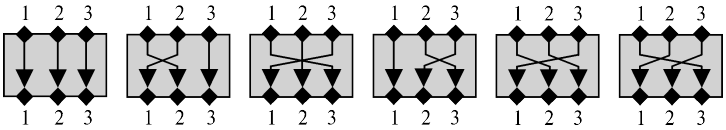


Рис. 5.5. Возможные отображения P-блока 3 × 3

Хотя P-блок может использовать ключ, чтобы определить одно из n! отображений, обычно P-блоки – без применения ключа, то есть отображение задано заранее. Если P-блок задан заранее и вмонтирован в аппаратные средства или если он реализован в программном обеспечении, таблицы перестановок задают правило отображения. Во втором случае входы в таблице указывают в позиции, в которых указаны позиции выходов. Таблица 5.1 дает пример таблицы перестановок, когда n равно 64.

Таблица 5.1. Пример таблицы перестановки для прямого P-блока

58	50	42	34	26	18	10	02	60	52	44	36	28	20	12 04
62	54	46	38	30	22	14	06	64	56	48	40	32	24	16 *08*
57	49	41	33	25	17	09	01	59	51	43	35	27	19	11 03
6153	45	37	29	21	13	05	63	55	47	39	31	23	15	07

Таблица 5.1 имеет 64 табличных входа, которые фиксируют соответствие 64 информационным входам. Позиция (индекс) входа соответствует выходу. Например, первый табличный вход содержит номер 58. Это означает, что первый выход будет соответствовать 58-му входу. Поскольку последний табличный вход — 7, это означает, что 64-й выход будет соответствовать 7-му информационному входу, и так далее.

Пример 5.6

Составьте таблицу перестановки для прямого P-блока 8 × 8, которая перемещает два средних бита (биты 4 и 5) во входном слове к двум крайним битам (биты 1 и 8) выходного слова. Относительные позиции других битов не изменяются.

Решение

Нам надо создать прямой P-блок с таблицей [4 1 2 3 6 7 8 5]. Относительные позиции бит 1, 2, 3, 6, 7 и 8 не меняются, но первый информационный выход связан с четвертым информационным входом, восьмой информационный выход — с пятым информационным входом.

P-блоки сжатия. P-блок сжатия – это P-блок с n входами и m выходами, где $m < n$. Некоторые из информационных входов блокированы и не связаны с выходом (см. рисунок 5.4). P-блоки сжатия, используемые в современных блочных шифрах, обычно являются бесключевыми с таблицей перестановки, которая указывает правила перестановки бит. Нам надо учитывать, что таблица перестановок для P-блока сжатия имеет m табличных входов, но в содержании каждого табличного входа – от 1 до n, и некоторые из них могут отсутствовать (те информационные входы, которые блокированы). Таблица 5.2 показывает пример таблицы пере-

Таблица 5.2. Пример таблицы перестановки 32 × 24

01	02	03	21	22	26	27	28	29	13	14	17
18	19	20	04	05	06	10	11	12	30	31	32

становки для P-блока сжатия 32 × 24. Обратите внимание, что входы 7, 8, 9, 16, 23, 24 и 25 блокированы.

P-блоки сжатия используются, когда мы должны переставить биты и в то же время уменьшить число битов для следующей ступени.

P-блок расширения — P-блок с n входами и m выходами, где $m > n$. Некоторые из входов связаны больше чем с одним выходом (см. рис. 5.4). P-блоки расширения, используемые в современных блочных шифрах, обычно без ключа. Правила перестановки бит указываются в таблице. Таблица перестановки для P-блока расширения имеет m табличных входов, но $m - n$ входов (те входы, которые связаны больше чем с одним информационным выходом). Таблица 5.3 показывает пример таблицы перестановки для P-блока расширения 12 × 16. Обратите внимание, что каждый из 1, 3, 9 и 12 соединен с двумя выходами.

Таблица 5.3. Пример таблиц перестановки 12 × 16

01	09	10	11	12	01	02	03	03	04	05	06	07	08	09	12

P-блоки расширения применяются, когда мы должны переставить биты и в то же время увеличить число битов для следующего каскада шифрования.

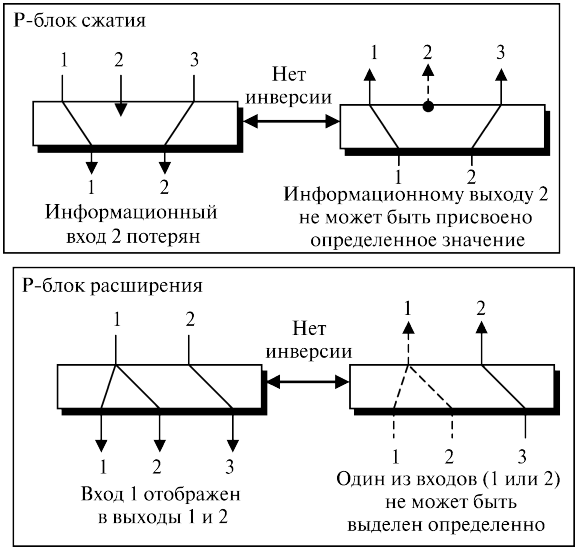
Обратимость. Прямой P-блок является обратимым. Это означает, что мы можем использовать прямой P-битовый шифр и дешифровать его. Таблицы перестановки, однако, должны быть обратимыми по отношению друг к другу. В лекции 3 мы видели, как можно получить обратную таблицу перестановки.

Пример 5.7

Рисунок 5.6 показывает, как изменить таблицу перестановки в случае одномерной таблицы.

Рис. 5.6. Изменение таблицы перестановки

P-блоки сжатия и расширения необратимы. В P-блоках сжатия вход может быть отброшен в процессе шифрования; алгоритм дешифрования не имеет клю-



ча, чтобы восстановить отброшенный бит. В P-блоке расширения вход в процессе шифрования может быть отображен более чем в один выход; алгоритм дешифрования не имеет ключа и не определяет, тем самым, какие из нескольких входов отображены в данном выходе. Рисунок 5.7 демонстрирует оба случая.

Рис. 5.7. P-блоки сжатия и расширения как необратимые компоненты

Рисунок 5.7 также показывает, что P-блок сжатия не является обратным шифром P-блока расширения и наоборот. Это означает, что если мы используем P-блок сжатия для шифрования, мы не сможем использовать P-блок расширения для дешифрования и наоборот. Однако, как будет показано позже в этой лекции, есть шифры, которые применяют P-блоки сжатия или расширения для шифрования; но их эффективность хуже, чем у некоторых других способов.

Прямой P-блок является обратимым, а P-блоки сжатия и расширения — нет.

S-блоки

S-блок (блок подстановки) можно представить себе как миниатюрный шифр подстановки. Этот блок может иметь различное число входов и выходов. Другими словами, вход к S-блоку может быть *n*-битовым словом, а выход может быть *m* разрядным словом, где *m* и *n* — не обязательно одинаковые числа. Хотя S-блок может быть ключевым или без ключа, современные блочные шифры обычно используют S-блоки без ключей, где отображение от информационных входов к информационным выходам заранее определено.

S-блок — m × *n* модуль подстановки, где m и *n* не обязательно равны.

Линейный и нелинейный S-блоки. В S-блоке с n входами и m выходами мы обозначим входы x_0, x_1,...., x_n и выходы y_1 ,..., y_m. Соотношения между входами и выходами могут быть представлены как система уравнений

$$y_1 = f_1(x_1,x_2,...,x_n)$$
$$y_2 = f_2(x_1,x_2,...,x_n)$$
$$....$$
$$y_m = f_m(x_1,x_2,...,x_n)$$

В **линейном S-блоке** вышеупомянутые соотношения могут быть выражены как

$$\boldsymbol{y_1 = a_{1,1}\,x_1 \oplus a_{1,2}\,x_2 \oplus \ldots \oplus a_{1,n}x_n}$$
$$\boldsymbol{y_2 = a_{2,1}\,x_1 \oplus a_{2,2}\,x_2 \oplus \ldots \oplus a_{2,n}x_n}$$
$$\boldsymbol{...}$$
$$\boldsymbol{y_m = a_{m,1}\,x_1 \oplus a_{m,2}\,x_2 \oplus \ldots \oplus a_{m,n}x_n}$$

В **нелинейном S-блоке** мы не можем всегда задать для каждого выхода указанные выше соотношения.

Пример 5.8

В S-блоке с тремя входами и двумя выходами мы имеем

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 111 \\ 100 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$y_1 = x_1 \oplus x_2 \oplus x_2 \oplus x_3 \qquad y_2 = x_1$$

S-блок линеен, потому что $a_{1,1} = a_{1,2} = a_{1,3} = a_{2,1}=1$ и $a_{2,2} = a_{2,3} = 0$. Эти соотношения могут быть представлены матрицами, как показано ниже:

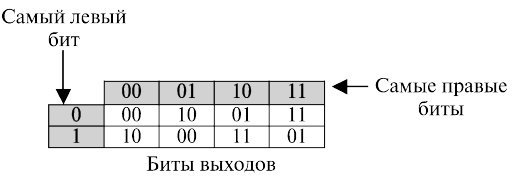
Пример 5.9

В S-блоке с тремя входами и двумя выходами мы имеем

$$y_1 = (x_1)^3 + x_2 \quad y_2 = (x_1) + x_1x_2 + x_3$$

где умножение и сложение проводится в GF(2). S-блок нелинеен, потому что нет линейных соотношений между входами и выходами.

Пример 5.10



Следующая таблица определяет отношения между входами/выходами для S-блока размера 3 × 2. Крайний левый бит входа определяет строку; два самых правых бита входа определяют столбец. Два бита выхода – это значение на пересечении секции выбранной строки и столбца.

Основываясь на таблице, вход 010 порождает выход 01. Вход 101 порождает выход 00.

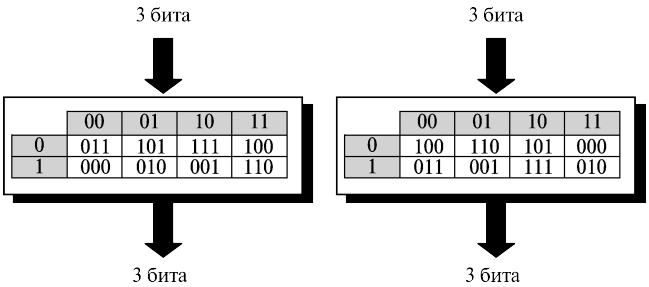
Обратимость. S-блоки — шифры подстановки, в которых отношения между входом и выходом определены таблицей или математическим соотношением. S-блок может быть или может не быть обратимым. В обратимом S-блоке число входных битов должно быть равным числу бит выхода.

Пример 5.11

Рисунок 5.8 показывает пример обратимого S-блока. Одна из таблиц используется в алгоритме шифрования; другая таблица — в алгоритме дешифрования. В каждой таблице крайний левый бит входа определяет строку; следующие два бита определяют столбец. Выход — это значение на пересечении строки и столбца.

Например, если вход к левому блоку — 001, выход — 101. Вход 101 в правой таблице дает выход 001. Это показывает, что эти две таблицы позволяют получить обратный результат по отношению друг к другу.

ИСКЛЮЧАЮЩЕЕ ИЛИ



Важный компонент в большинстве блоков шифрования — операция ИСКЛЮЧАЮЩЕЕ ИЛИ: Как мы уже обсуждали в лекции 4, операции сложения и вычитания в $GF(2^n)$ выполняется с помощью одной и той же операции, называемой ИСКЛЮЧАЮЩЕЕ ИЛИ или (XOR):

Рис. 5.8. Таблицы S-блока для примера 5.11

Свойства. Пять свойств операции ИСКЛЮЧАЮЩЕЕ ИЛИ в поле $GF(2^n)$ делают эту операцию очень удобной для использования в блочном шифре.

1. *Замкнутость.* Это свойство гарантирует, что в результате этой операции два *n*-битовых слова дают другое *n*-битовое слово.

2. *Ассоциативность.* Это свойство позволяет нам использовать больше чем одно ИСКЛЮЧАЮЩЕЕ ИЛИ, которые можно вычислять в любом порядке.

$$x \oplus (y \oplus z) \leftrightarrow (x \oplus y) \oplus z$$

3. *Коммутативность.* Это свойство позволяет нам менять местами операторы (входную информацию), не изменяя результат (выходную информацию).

$$x \oplus (y\ z) \leftrightarrow (x \oplus y) \oplus z$$

4. *Существование нулевого (тождественного) элемента.* Нулевой элемент для операции ИСКЛЮЧАЮЩЕЕ ИЛИ – слово, которое состоит из всех нулей, или (00... 0). Это подразумевает, что существует слово с нейтральными элементами, которое при проведении операции не изменяет слово.

$$x \oplus (00....00) = (00\ ...\ 0)$$

Мы используем это свойство в шифре Файстеля, который рассмотрим позже в этой лекции.

5. *Существование инверсии.* В поле GF(2^n) каждое слово есть аддитивная инверсия самого себя. Это подразумевает, что проведение операции ИСКЛЮЧАЮЩЕЕ ИЛИ слова с самим собой приводит к нулевому элементу:

$$x \oplus x = (00...\ 0)$$

Мы также используем это свойство в шифре Файстеля, который рассмотрим позже в этой лекции.

Дополнение. Операция дополнения — одноместная операция (один информационный вход и один информационный выход), которая инвертирует каждый бит в слове. 0-вой бит меняет на 1 (единичный) бит; 1 (единичный) бит меняет на 0-вой бит. Нас интересует операции с дополнением относительно операции ИСКЛЮЧАЮЩЕЕ ИЛИ. Если $\overline{x}$ — дополнение «*x*», тогда верны следующие два соотношения:

$$x \oplus \overline{x} = (11....1) \text{ и } x \oplus (11...\ 1) = \overline{x}$$

Мы также используем эти свойства позже в этой лекции, когда будем обсуждать безопасность некоторых шифров.

Инверсия. Инверсия компонента в шифре имеет смысл, если компонент представляет одноместную операцию (один вход и один выход). Например, P-блок без ключа или S-блок без ключа могут быть обратимыми, потому что они имеют один вход и один выход. Операция ИСКЛЮЧАЮЩЕЕ ИЛИ — бинарная операция. Инверсия операции ИСКЛЮЧАЮЩЕЕ ИЛИ может иметь смысл, только если один из входов зафиксирован (один и тот же при шифровании и дешифровании). Например, если один из входов — ключ, который обычно являет-

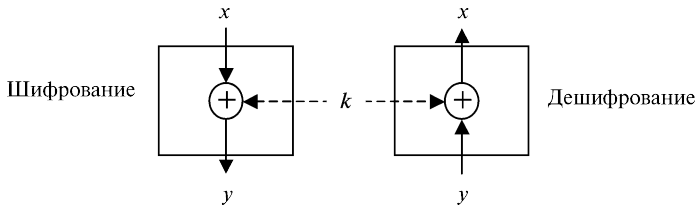


Рис. 5.9. Обратимость операции ИСКЛЮЧАЮЩЕЕ ИЛИ

ся одним и тем же в шифровании и дешифровании, тогда операция ИСКЛЮЧАЮЩЕЕ ИЛИ является обратимой, как показано на рис. 5.9.

На рисунке 5.9 свойство аддитивной инверсии подразумевает, что

$$y = x \oplus k \qquad x = k \oplus y$$

Мы используем это свойство, когда позже в этой лекции будем обсуждать структуру блочных шифров.

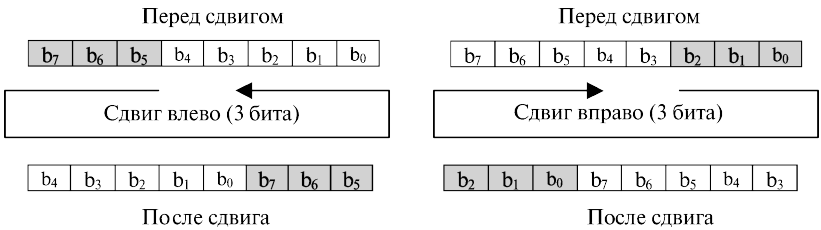
Циклический сдвиг

Другой компонент, применяемый в некоторых современных блочных шифрах, – **операция циклического сдвига.** Смещение может быть влево или вправо. Круговая операция левого сдвига сдвигает каждый бит в n-битовом слове на k позиции влево; крайние левые k-биты удаляются слева и становятся самыми правыми битами. Круговая операция правого сдвига сдвигает каждый бит в n-битовом слове на k позиций вправо; самые правые k-биты справа удаляются и становятся крайними левыми битами. Рисунок 5.10 показывает и левые и правые операции в случае, где $n = 8$ и $k = 3$.

Циклическая операция сдвига смешивает биты в слове и помогает скрыть образцы в первоначальном слове. Хотя число позиций, на которые биты будут сдвинуты, может использоваться как ключ, циклическая операция сдвига обычно – без ключа; значение k устанавливается и задается заранее.

Обратимость. Циклическая операция левого сдвига – инверсия операции правого сдвига. Если одна из них используется для шифрования, другая может применяться для дешифрования.

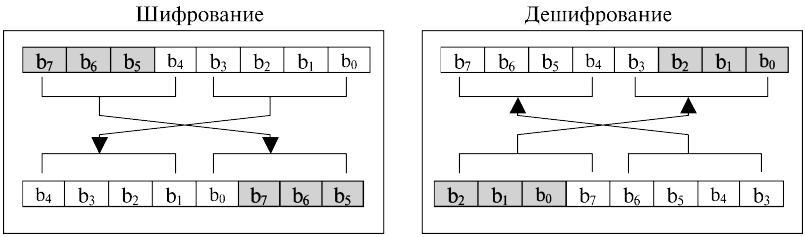
Свойства. Операция циклического сдвига имеет два свойства, которые нам надо знать. Первая — это смещение по модулю n. Другими словами, если $k = 0$ *или*



$k = n$, никакого смещения не происходит. Если k является большим, чем n, тогда входная информация сдвинута на $k \bmod n$ бит. Второе свойство, операция циклического сдвига над соединением операций — есть группа группа. Это означает, что если смещение деляется неоднократно, то одно и то же значение может появиться несколько раз..

Рис. 5.10. Циклический сдвига 8 битового слова налево или направо

Замена

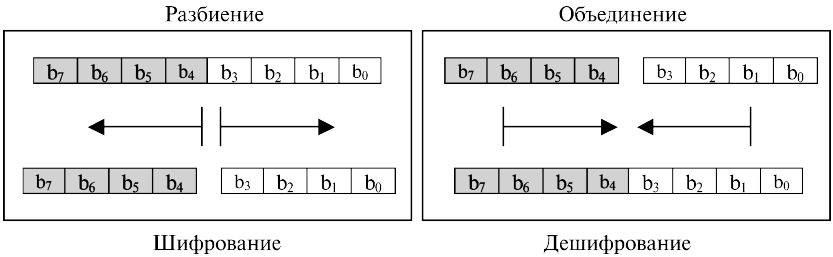


Операция замены — специальный случай операции циклического сдвига, где $k = n/2$ означает, что эта операция возможна, только если n — четный номер. Поскольку сдвиг влево n/2 — то же самое, что сдвиг n/2 вправо, эта операция является обратимой. Операция замены для шифрования может быть полностью раскрыта операцией замены для дешифрации. Рисунок 5.11 иллюстрируетт операцию замены для слова на 8 битов.

Рис. 5.11. Операция замена в 8-битовом а слове

Разбиение и объединение

Две других операции, применяемые в некоторых блочных шифрах, — разбиение и объединение. **Разбиение** обычно разделяет n-битовое слово в середине, создавая два слова равной длины. **Объединение** связывает два слова равной длины, чтобы создать n-битовое слово. Эти две операции инверсны друг другу и могут использоваться как пара, чтобы уравновесить друг друга. Если одна применяется для



шифрования, то другая — для дешифрования. Рисунок 5.12 показывает эти две операции для случая $n = 8$.

Составные шифры

Шеннон ввел понятие **составные шифры.** Составной шифр – комплекс, который объединяет подстановку, перестановку и другие компоненты, рассмотренные в предыдущих разделах.

Рис. 5.12. Операции разбиения и объединения с 8-битовым словом

Рассеивание и перемешивание

Идея Шеннона в представлении составного шифра должна была дать возможность блочным шифрам иметь две важных свойства: рассеяние и перемешивание. **Рассеивание** должно скрыть отношения между зашифрованным текстом и исходным текстом. Это собьет с толку противника, который использует статистику зашифрованного текста, чтобы найти исходный текст. Рассеивание подразумевает, что каждый символ (символ или бит) в зашифрованном тексте зависит от одного или всех символов в исходном тексте. Другими словами, если единственный символ в исходном тексте изменен, несколько или все символы в зашифрованном тексте будут также изменены.

Рассеивание скрывает отношения между зашифрованным текстом и исходным текстом.

Идея относительно **перемешивания** — в том, что оно должно скрыть отношения между зашифрованным текстом и ключом. Это собьет с толку противника, который стремится использовать зашифрованный текст, чтобы найти ключ. Другими словами, если единственный бит в ключе изменен, все биты в зашифрованном тексте будут также изменены.

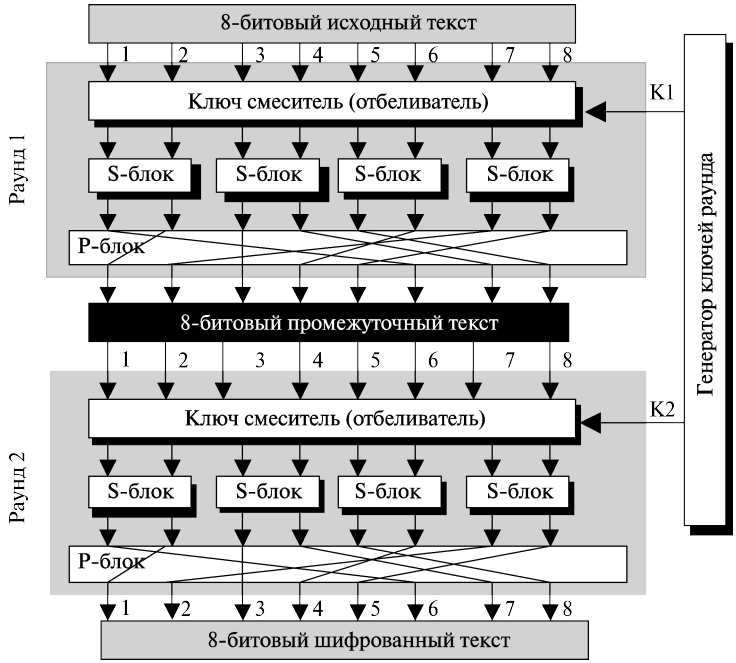
Перемешивание скрывает отношения между зашифрованным текстом и ключом.

Раунды

Рассеивание и перемешивание могут быть достигнуты с помощью применения повторения составных шифров, где каждая итерация — комбинация S-блоков, P-блоков и других компонентов. Каждая итерация называется **раундом.** Блочный шифр использует **список ключей**, или **генератор ключей,** который создает различные ключи для каждого раунда от ключа шифра. В N-раундном шифре, чтобы создать зашифрованный текст, исходный текст шифруется N раз; соответственно, зашифрованный текст расшифровывается N раз. Текст, созданный на промежуточных уровнях (между двумя раундами), называется **средним текстом**. Рисунок 5.13 показывает простой составной шифр с двумя раундами. На практике составные шифры имеют больше чем два раунда. На рис. 5.13 в каждом раунде проводятся три преобразования:

[1] Отбеливание (whiting) – процесс проведения операции, которая приближает текст к равновероятному появлению всех символов. Термин образован в соответствии с процессом приведения к «белому шуму» – сигналу, имеющему равномерный спектр. (*прим. перев.*)

a. 8-битовый текст смешивается с ключом, чтобы сделать символы текста равновероятными (скрыть биты, используя ключ) — «отбелить» текст (whiting)[1].



Это обычно делается с помощью операции ИСКЛЮЧАЮЩЕЕ ИЛИ слова на 8 битов с ключом на 8 битов;

b. выходы «отбеливателя» разбиты на четыре группы по 2 бита и подаются в четыре S-блока. Значения битов изменяются в соответствии с построением S-блоков в этом преобразовании;

c. выходы S-блоков поступают в P-блок, при этом биты переставлены так, чтобы в следующем раунде результат каждого блока поступил на различные входы.

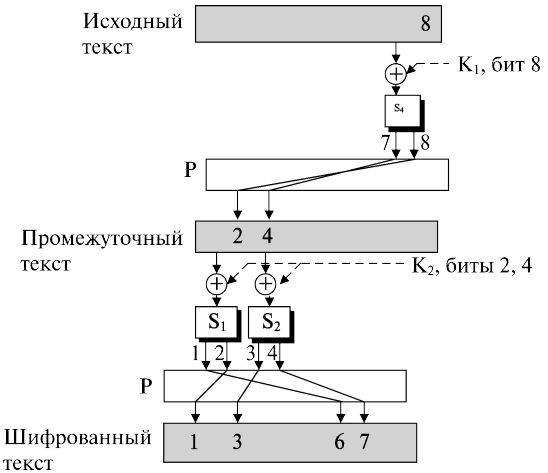
Рис. 5.13. Составнной шифр, состоящий из двух раундов

Рассеивание, которое показано на упрощенном рис. 5.13 как составной шифр, используя комбинацию S-блоков и P-блоков, может гарантировать результат.

a. В первом раунде бит 8, после проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с соответствующими битами ключа K_1, изменяет два бита (биты 7 и 8) через S-блок 4. Бит 7 переставлен и становится битом 2; бит 8 переставлен и становится битом 4. После первого раунда бит 8 изменяет биты 2 и 4. Во втором раунде бит 2 после проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с соответствующими битами ключа K_2 изменяет два бита (биты 1 и

2) через S-блок 1. Бит 1 – переставлен и становится битом 6; бит 2 переставлен и становится битом 1. Бит 4 после проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с соответствующим битом в K_2 изменяет биты 3 и 4. Бит 3 остается, бит 4 переставлен и становится битом 7. После второго раунда из 8 бит изменены биты 1, 3, 6 и 7.

b. Прохождение этих шагов в другом направлении (от зашифрованного текста до исходного текста) показывает, что каждый бит в зашифрованном тексте изменяет исходный текст на несколько битов.



Перемешивание. На рисунке 5.14 показано, как изменение единственного бита в исходном тексте вызывает изменение многих битов в зашифрованном тексте. Рисунок 5.14 также доказывает нам, что свойство перемешивания может быть получено с помощью составного шифра. Четыре бита зашифрованного текста, биты 1, 3, 6 и 7 преобразованы с помощью трех битов в ключах (бит 8 в K_1 и битах 2 и 4 в K_2). Прохождение в обратном направлении показывает, что каждый бит ключа в каждом раунде затрагивает несколько битов в зашифрованном тексте. Отношения между битами зашифрованного текста и ключевыми битами показаны в затененных прямоугольниках.

Рис. 5.14. Рассеивание и перемешивание в блочном шифре

Практические шифры. Чтобы улучшить рассеивание и перемешивание, практические шифры используют крупные блоки данных, больше S-блоков и больше раундов. Очевидно, что некоторое увеличение числа раундов при применении большого числа S-блоков может создать лучший шифр, в котором зашифрованный текст выглядит все более как случайное *n*-битовое слово. Таким образом, отношения между зашифрованным текстом и исходным текстом будут полностью скрыты (рассеяны). Увеличение числа раундов увеличивает число ключей раундов, что лучше скрывает отношения между зашифрованным текстом и ключом.

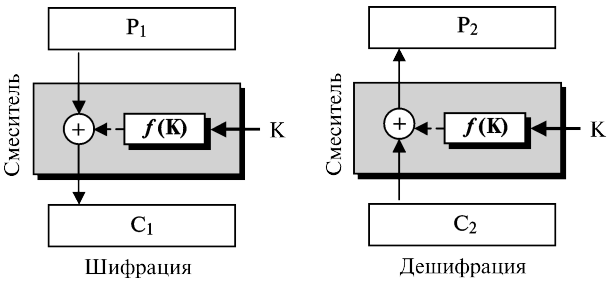
Два класса составных шифров

Все современные блочные шифры — составные, но они разделены на два класса. Шифры в первом классе используют и обратимые, и необратимые компоненты. Эти шифры упоминаются обычно как шифры Файстеля. Блочный шифр DES (DATA ENCRYPTION STANDARD), обсуждаемый в лекции 6, — хороший пример шифра Файстеля. Шифры во втором классе применяют только обратимые компоненты. Обращаем ваше внимание на шифры в этом классе как шифры не-Файстеля[1] (из-за отсутствия другого названия). Блочный шифр AES (ADVANCED ENCRYPTION STANDARD), обсуждаемый в лекции 7, — хороший пример шифра не-Файстеля.

Шифры Файстеля

Файстель проектировал очень интеллектуальный и интересный шифр, который использовался в течение многих десятилетий. **Шифр Файстеля** может иметь три типа компонентов: самообратимый, обратимый и необратимый.

Шифр Файстеля содержит в блоках все необратимые элементы и использует один и тот же модуль в алгоритмах дешифрования и шифрования. Вопрос в том, как алгоритмы шифрования и дешифрования позволяют инвертировать от-



крытый и закрытый тексты друг в друга, если каждый содержит необратимый модуль. Файстель показал, что они могут быть сбалансированы.

Первая идея. Чтобы лучше понять шифр Файстеля, давайте посмотрим, как мы можем использовать один и тот же необратимый компонент в алгоритмах дешифрования и шифрования. Эффекты необратимого компонента в алгоритме шифрования могут быть отменены в алгоритме дешифрования, если мы применяем операцию ИСКЛЮЧАЮЩЕЕ ИЛИ, как показано на рис. 5.15.

Рис. 5.15. Первая идея в разработке шифра Файстеля

[1] Хорст Файстель – сотрудник фирмы IBM, разработавший в 1977 году криптоалгоритм, который лег в основу стандарта шифрования в США, широко известного под названием DES.

В шифровании ключ поступает на вход необратимой функции $f(K)$, которая является одним из слагаемых оператора ИСКЛЮЧАЮЩЕГО ИЛИ с исходным текстом. Результат становится зашифрованным текстом. Мы будем называть комбинацию функции и операции ИСКЛЮЧАЮЩЕЕ ИЛИ **смесителем** (из-за отсутствия другого названия). Смеситель играет важную роль в более поздних вариантах шифра Файстеля.

Поскольку ключ один и тот же в шифровании и дешифровании, мы можем доказать, что два алгоритма инверсны друг другу. Другими словами, если $C_2 = C_1$ (любое изменение в зашифрованном тексте в течение передачи), то $P_2 = P_1$.

Шифрование: $C_1 = P_1 \oplus f(K)$
Дешифрование: $\mathbf{P_2} = \mathbf{C_2} \oplus f(K) = C_1 \oplus f(K) = \mathbf{P_1} \oplus f(K) \oplus f(K) = P_1 \oplus (00...0) = P_1$

Обратите внимание, что использовались два свойства операции ИСКЛЮЧАЮЩЕЕ ИЛИ (существование инверсии и существование нулевого кода).

Уравнения, показанные выше, доказывают, что хотя смеситель имеет неконвертируемый элемент, сам смеситель является самоконвертируемым.

Пример 5.12

Это тривиальный пример. Имеется исходный текст и зашифрованный текст, каждый 4 бита длиной, и ключ 3 бита длиной. Предположим, что функция извлекает первый и третий биты ключа, интерпретирует биты как десятичный номер, находит квадрат этого числа и интерпретирует результат как 4-битовую двоичную последовательность.

Покажите результаты шифрования и дешифрования, если первоначальный исходный текст — 0111, и ключ — 101.

Решение

Функция извлекает первые и вторые биты ключа и получается в результате 11 в двоичном виде или 3 в десятичном отображении. Результат возведения во вторую степень (квадрат) — 9, в двоичном отображении 1001.

Шифрование: $C = P \oplus f(K) = 0111 \oplus 1001 = 1110$
Дешифрование: $P = \mathbf{C} \oplus f(K) = 1110 \oplus 1001 = 0111$. Совпадет с исходным текстом P

Функция $f(101) = 1001$ является неконвертируемой, но операция ИСКЛЮЧАЮЩЕЕ ИЛИ позволяет нам использовать функцию и в алгоритмах дешифрования, и в шифровании. Другими словами, функция является неконвертируемой, но смеситель будет самоконвертируемым.

Усовершенствование. Попробуем улучшить нашу первую идею, чтобы приблизиться к шифру Файстеля. Мы знаем, что должны применить вход к неконвертируемому элементу (функции), но мы не будем использовать только ключ. Мы задействуем также вход к функции, чтобы применить ее для шифрования части исходного текста и дешифрования части зашифрованного текста. Ключ может использоваться как второй вход к функции. Этим способом наша функция становится сложным элементом с некоторыми неключевыми элементами и некоторыми ключевыми элементами. Чтобы достичь цели, разделим исходный текст и зашифрованный текст на

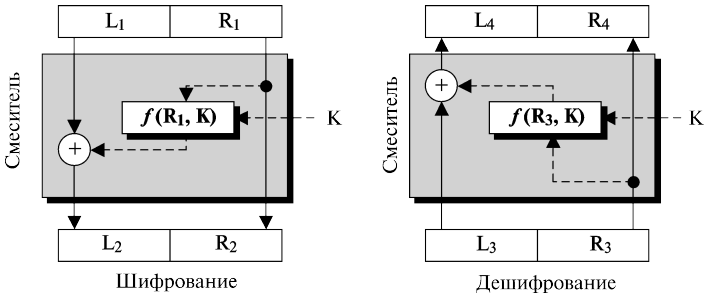


Рис. 5.16. Усовершенствование предыдущей схемы Файстеля

два блока равной длины – левый (L) и правый (R). Правый блок вводится в функцию, а левый блок складывается с помощью операции ИСКЛЮЧАЮЩЕЕ ИЛИ с выходом функции. Мы должны запомнить, что входы к функции должны точно совпадать в шифровании и дешифровании. Это означает, что правая секция исходного текста до шифрования и правая секция зашифрованного текста после дешифрования будут совпадать. Другими словами, секция должна войти в шифрование и выйти из дешифрования неизмененной. Рисунок 5.16 иллюстрирует идею.

Алгоритмы шифрования и дешифрования инверсны друг другу. Предположим, что $L_3 = L_2$ и $R_3 = R_2$ (в зашифрованном тексте в течение передачи не произошло изменений).

$$R_4 = R_3 = R_2 = R_1$$
$$L_4 = L_3 \oplus f(R_3,K) = L_2 \oplus f(R_2,K) = L_1 \oplus f(R_1,K) \oplus f(R_1,K) = L_1$$

Исходный текст, используемый в алгоритме шифрования, — это текст, правильно восстановленный алгоритмом дешифрования.

Окончательный вариант. Предыдущее усовершенствование имеет один недостаток: правая половина исходного текста никогда не изменяется. Ева может немедленно найти правую половину исходного текста, разбивая на части зашифрованный текст и распаковывая его правую половину. Проект нуждается в дальнейших шагах усовершенствования.

Первое: увеличим число раундов. Второе: добавим новый элемент в каждый раунд — **устройство замены.** Эффект устройства замены в раунде шифрования компенсируется эффектом устройства замены в раунде дешифрования. Однако это позволяет нам менять левые и правые половины в каждом раунде. Рисунок 5.17 иллюстрирует новый вариант шифра Файстеля с двумя раундами.

Обратите внимание, что есть два ключа раундов: K_1 и K_2. Ключи применяются в обратном порядке в шифровании и дешифровании.

Поскольку два смесителя инверсны друг другу и устройства замены инверсны друг другу, очевидно, что шифрование и дешифрование также инверсны друг другу. Однако мы можем доказать этот факт, используя отношения между левыми и правыми секциями в каждом шифре. Другими словами, если $L_6 = L_1$ и $R_6 = R_1$,

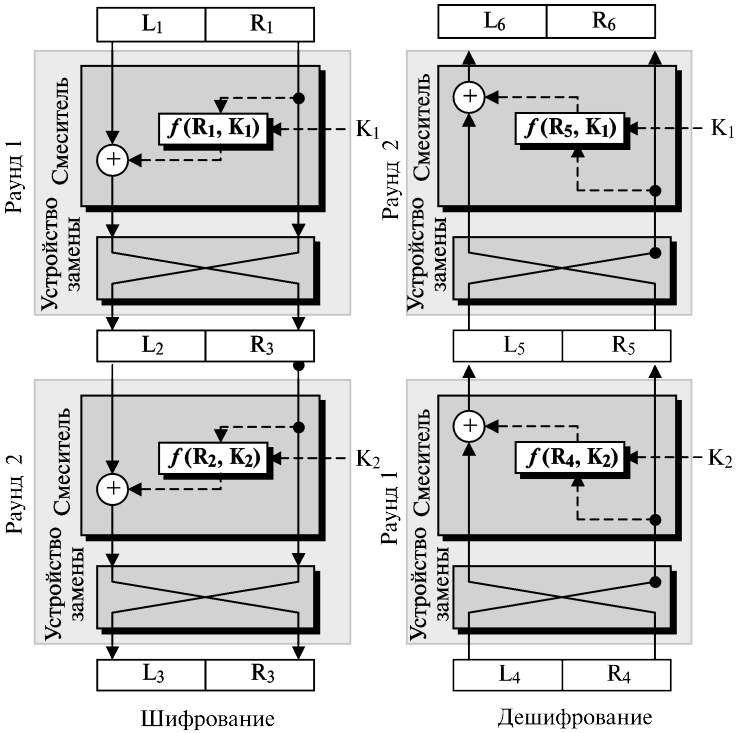


Рис. 5.17. Окончательный вариант шифра Файстеля с двумя раундами

предположим, что $L_4 = L_3$ и $R_4 = R_3$ (шифрованный текст не изменился при передаче). Вначале докажем это для промежуточного текста:

$$L_5 = R_4 \oplus f(L_4, K_2) = R_3 \oplus f(R_2, K_2) = L_2 \oplus f(R_2, K_2) \oplus f(R_2, K_2) = L_2$$
$$R_5 = L_4 = L_3 = R_2$$

Тогда просто доказать равенство для двух блоков исходного текста.

$$L_6 = R_5 \oplus f(L_5, K_1) = R_2 \oplus f(L_2, K_1) = L_1 \oplus f(R_1, K_1) \oplus f(R_1, K_1) = L_1$$
$$R_6 = L_5 = L_2 = R_1$$

Шифры не-Файстеля

Шифр не-Файстеля использует только обратимые компоненты. Компонент в исходном тексте имеет соответствующий компонент в шифре. Например, S-блоки должны иметь равное число входов и выходов, чтобы быть совместимыми. Не позволяется никакое сжатие или расширение P-блоков, потому что они станут необратимыми. В шифре не-Файстеля нет потребности делить исходный текст на две половины, как мы видели в шифрах Файстеля.

Рисунок 5.13 можно рассматривать как графическую иллюстрацию принципа шифра не-Файстеля, потому что единственные компоненты в каждом раунде — самообратимые операции ИСКЛЮЧАЮЩЕЕ ИЛИ, S-блоки 2 × 2, которые могут быть спроектированы так, чтобы быть обратимыми, и прямые P-блоки, которые обратимы, если использована соответствующая таблица перестановки. Поскольку каждый компонент является обратимым, можно показать, что и каждый раунд является обратимым. Мы только должны применять ключи раундов в обратном порядке. Шифрование использует ключи раундов K_1 и K_2. Алгоритм дешифрования должен пользоваться ключами раундов K_2 и K_1.

Атаки на блочные шифры

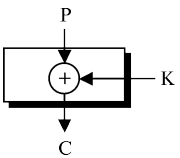
Атаки традиционных шифров могут также использоваться для современных блочных шифров, но сегодняшние блочные шифры успешно противостоят большинству атак, обсужденных в лекции 3. Например, грубая силовая атака ключа, как правило, неосуществима, потому что ключи обычно имеют очень большую длину. Однако недавно были изобретены некоторые новые виды атак блочных шифров, которые основаны на структуре современных блочных шифров. Эти атаки используют методы и дифференциального, и линейного анализа.

Дифференциальный криптоанализ

Идею относительно **дифференциального криптоанализа** предложили Эли Бихам и Ади Шамир[1]. Это — *атака с выборкой исходного текста.* Ева может каким-либо образом получить доступ к компьютеру Алисы и завладеть выборочно частью исходного текста и соответствующего зашифрованного текста. Цель состоит в том, чтобы найти ключ шифра Алисы.

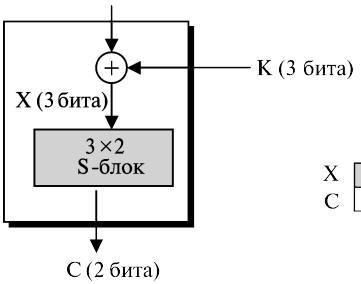
Алгоритм анализа. Перед тем как Ева предпримет атаку с выборкой исходного текста, она должна проанализировать алгоритм шифрования, чтобы собрать некоторую информацию об отношениях зашифрованного и исходного текстов. Очевидно, Ева не знает ключ шифра. Однако некоторые шифры имеют слабости в структурах, которые могут позволить Еве найти различия исходного текста и различия зашифрованного текста, не зная ключ.

Пример 5.13



[1] Эли Бихам (Eli Biham) и Эди Шамир (Adi Shamir) — сотрудники Израильского Научно-Исследовательского Института им. Вейцмана — (Департамент математики) предложили в 1990 году дифференциальный метод криптоанализа.

Предположим, что шифр состоит только из одной операции ИСКЛЮЧАЮЩЕЕ ИЛИ, как показано на рис. 5.18. Не зная значения ключа, Ева может легко



X	000	001	010	011	100	101	110	111
C	11	00	10	10	01	00	11	00

Таблица S-блока

найти отношения между разностями исходного текста и разностями зашифрованного текста. Если разность исходного текста мы обозначим $P_1 \oplus P_2$ и разность зашифрованного текста мы обозначим $C_1 \oplus C_2$, приведенные следующие преобразования доказывают, что $C_1 \oplus C_2 = P_1 \oplus P_2$:

$$C_1 = P_1 \oplus K \; C_2 = P_2 \oplus K \rightarrow C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

Однако этот пример нереалистичен; модемные блочные шифры не настолько просты.

Рис. 5.18. Диаграмма для примера 5.13

Пример 5.14

В примере 5.13 мы добавляем один S-блок, как показано на рис. 5.19.

Рис. 5.19. Диаграмма для примера 5.14

$P_1 \oplus P_2$ \ $C_1 \oplus C_2$	00	01	10	11
000	**8**			
001	**2**	**2**		**4**
010	**2**	**2**	**4**	
011		**4**	**2**	**2**
100	**2**	**2**	**4**	
101		**4**	**2**	**2**
110	**4**		**2**	**2**
111			**2**	**6**

Хотя эффект шифрования ключом не действует, когда мы используем разности между двумя X и двумя P ($X_1 \oplus X_2 = P_1 \oplus P_2$), существование S-блока мешает Еве найти и определенные отношения между разностями исходного текста и разностями зашифрованного текста. Однако возможно установить вероятностные отношения. Ева может составить таблицу 5.4, которая показывает для разности исходного текста, сколько можно создать разностей зашифрованного текста — шифр. Обращаем внимание, что таблица сделана по информации, которая произведена с учетом таблицы входа-выхода S-блока по рис. 5.19, потому что $P_1 \oplus P_2 = X_1 \oplus X_2$.

Таблица 5.4. Дифференциальная таблица для входов и выходов для шифра в примере 5.14

Поскольку размер ключей — 3 бита, может быть восемь случаев для каждой разности во вводе. Таблица показывает, что если входная разность — $(000)_2$, раз-

	00	01	10	11
000	1	0	0	0
001	0,25	0,25	0	0,50
010	0,25	0,25	0,50	0
011	0	0,50	0,25	0,25
100	0,25	0,25	0,50	0
101	0	0,50	0,25	0,25
110	0,50	0	0,25	0,25
111	0	0	0,25	0,75

ность выхода — всегда $(00)_2$. С другой стороны, таблица показывает, что если входная разность — $(100)_2$, то имеется два случая разностей выхода $(00)_2$, два случая разностей выхода $(01)_2$ и четыре случая разностей выхода $(01)_2$.

Пример 5.15

Эвристический результат примера 5.14 может создать вероятностную информацию для Евы, как показано в таблице 5.5. Входы в таблице соответствуют вероятностям появления. Разности с нулевой вероятностью никогда не будут возникать.

Таблица 5.5. Дифференциальная таблица входов и выходов для шифра в примере 5.15

Как мы увидим позже, Ева теперь располагает достаточным количеством информации, чтобы начать атаку. Таблица показывает, что вероятности распределены неоднородно из-за слабости в структуре S-блока. Таблица 5.5 упоминается иногда как **дифференциальная таблица распределения** или **профайл ИСКЛЮЧАЮЩЕЕ ИЛИ.**

Запуск атаки выборки исходного текста. После того как анализ однажды сделан, он может быть сохранен для будущего использования, пока структура шифра не изменится. Ева может выбрать для атак исходные тексты. Дифференциальная таблица распределения вероятности (таблица 5.5) поможет Еве их выбирать — она возьмет те, которые имеют самую высокую вероятность в таблице.

Предположительное значение ключа. После запуска некоторых атак с соответствующей выборкой исходного текста Ева может найти некоторую пару «исходный текст / зашифрованный текст», которая позволяет ей предположить некоторое значение ключа. Процесс начинается от C и продвигается к P.

Пример 5.16

Рассматривая таблицу 5.5, Ева знает, что если $P_1 \oplus P_2 = 001$, то $C1 \oplus C2 = 11$ с вероятностью 0,50 (50 процентов). Она пробует взять $C_1 = 00$ и получает $P_1 = 010$ (атака с выборкой зашифрованного текста). Она еще пробует $C_2 = 11$ и получает $P_2 = 011$ (другая атака с выборкой зашифрованного текста). Теперь она пробует вернуться к анализу, основанному на первой паре, P_1 и C_1:

$C_1 = 00 \rightarrow X_1 = 001$ или $X_1 = 111$
Если $X_1 = 001 \rightarrow K = X_1 \oplus P_1 = 011. \rightarrow$ Если $X_1 = 111 \rightarrow K = X_1 \oplus P_1 = 101$

Используя пару P2 и C2, получим

$C_2 = 11 \rightarrow X_2 = 000$ или $X_1 = 110$
Если $X_2 = 000 \rightarrow K = X2 \oplus P2 = 011 \rightarrow$ Если $X_{12} = 110 \rightarrow K = X2 \oplus P2 = 101$

Два испытания показывают, что K = 011 или K =101. Хотя Ева не уверена, какое из них точное значение ключа, она знает, что самый правый бит — 1 (общий бит между двумя значениями).

Продолжая атаку, можно учитывать, что самый правый бит в ключе – 1. Таким образом можно определить другие биты в этом ключе.

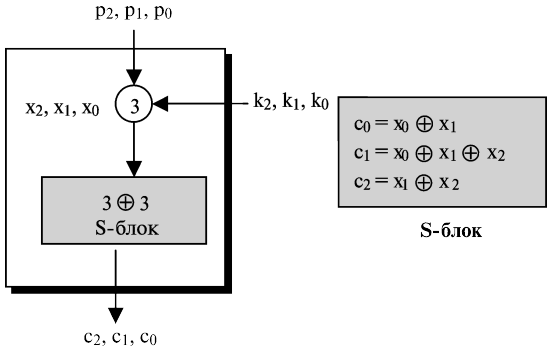
Общая процедура. Современные блочные шифры имеют большую сложность, чем, та, которую мы обсуждали в этом разделе. Кроме того, они могут содержать различное количество раундов. Ева может использовать следующую стратегию.

1. Поскольку каждый раунд содержит одни и те же операции, Ева может создать таблицу дифференциальных распределений (профайл ИСКЛЮЧАЮЩЕГО ИЛИ) для каждого S-блока и комбинировать их, чтобы создать распределение для каждого раунда.
2. Предположим, что каждый раунд независим (справедливое предположение). Ева может создать таблицу распределения для всего шифра, умножая соответствующие вероятности.
3. Ева может теперь сделать список исходных текстов для атак, основанных на таблице распределений на втором шаге. Заметим, что таблица в шаге 2 только помогает Еве выбирать меньшее количество пар «исходный текст / зашифрованный текст»
4. Ева выбирает зашифрованный текст и находит соответствующий исходный текст. Затем она анализирует результат, чтобы найти некоторые биты в ключе.
5. Ева повторяет шаг 4, чтобы найти больше битов в ключе.
6. После нахождения достаточного количества битов в ключе Ева может использовать атаку грубой силы, чтобы найти весь ключ.

Дифференциальный криптоанализ базируется на таблице неоднородных дифференциальных распределений, S-блоков в блочном шифре.

[1] Митцуру Мацуи (Mitsuru Matsui) — криптолог, сотрудник фирмы Mitsubishi Electric Corporation (Япония).

Более детально дифференциальный криптоанализ приводится в Приложении N.



Линейный криптоанализ

Линейный криптоанализ был представлен Митцури Мацуи (Mitsuru Matsui[1]) в 1993 году. Анализ использует *атаки знания исходного текста* (в отличии от атак с выборкой исходного текста в дифференциальном криптоанализе). Полное обсуждение этой атаки базируется на некоторых понятиях теории вероятностей, которые находятся за рамками этой книги. Чтобы рассмотреть главную идею этой атаки, предположим, что шифр состоит из одного раунда, как показано на рис. 5.20, где c_0,c_1 и c_2 представляют три бита на выходе и x_0, x_1 и x_2 представляют три бита на входе S-блока.

S-блок — линейное преобразование, в котором каждый вывод является линейной функцией ввода, как мы обсуждали ранее в этой лекции. С этим линейным компонентом мы можем создать три линейных уравнения между исходным текстом и битами зашифрованного текста, как показано ниже:

Рис. 5.20. Простой шифр с линейным S-блоком

$$c_0 = p_0 \oplus k_0 \oplus p_1 \oplus k_1$$
$$c_1 = p_0 \oplus k_0 \oplus p_1 \oplus k_1 \oplus p_2 \oplus k_2$$
$$c_2 = p_1 \oplus k_1 \oplus p_2 \oplus k_2$$

Решая систему уравнений для трех неизвестных, мы получаем

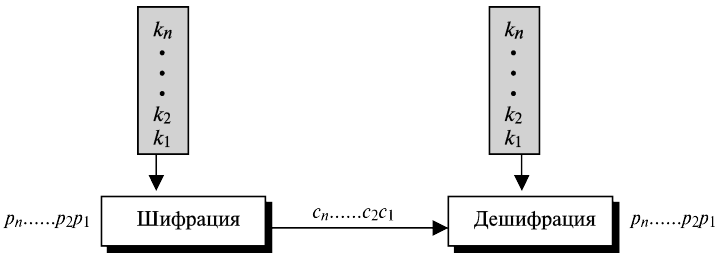
$$k_1 = (p_1) \oplus (c_0 \oplus c_1 \oplus c_2)$$
$$k_2 = (p_2) \oplus (c_0 \oplus c_1)$$
$$k_0 = (p_0) \oplus (c_1 \oplus c_2)$$

Это означает, что три атаки типа «знания исходного текста» могут найти значения k_1, и k_2 . Однако реальные блочные шифры не так просты, как этот; они имеют больше компонентов, и S-блоки не линейны.

Линейная аппроксимация. В некоторых современных блочных шифрах может случиться, что некоторые S-блоки не полностью нелинейные; тогда они могут быть в вероятностном смысле аппроксимированы некоторыми линейными функциями. Вообще, задавая исходный и зашифрованный текст в *n* бит и ключ *m* бит, мы ищем некоторые уравнения, имеющие вид

$$(k_0 \oplus k_1 \oplus \ldots \oplus k_x) = (p_0 \oplus p_1 \oplus \ldots \oplus p_y) \oplus (c_0 \oplus c_1 \oplus \ldots \oplus c_z)$$

где $1 \le x \le m$, $1 \le y \le n$ и $1 \le z \le n$. Биты в перехваченных исходном тексте и зашифрованном тексте могут быть использованы для того чтобы найти биты ключа.



Для того, чтобы достигнуть эффективности, каждое уравнение должно быть связано с вероятностью$1/2 + \varepsilon$, где ε назвается *погрешность*. Уравнение, где ε больше, эффективнее чем уравнение с малым значением ε.

Более детально линейный криптоанализ дан в приложении N.

5.2. Современные шифры потока

В лекции 3 мы кратко обсуждали разницу между традиционными шифрами потока и традиционными блочными шифрами. Подобные отличия существуют также между современными шифрами потока и современными блочными шифрами. В **современном шифре потока** шифрование и дешифрование *r* бит проводятся одновременно. Мы имеем поток бит исходного текста $P = p_n \ldots p_2 p_1$, поток бит зашифрованного текста $C = c_n \ldots c_2 c_1$, и ключевой поток бит $K = k_n \ldots k_2 k_1$, в которых p_i c_i, и k_i — это *r*-битные слова. Шифрование — $c_i = E(k_i, p_i)$, и дешифрование — $pi = D(k_i, c_i)$, как показывает рис. 5.21.

Рис. 5.21. Шифр потока

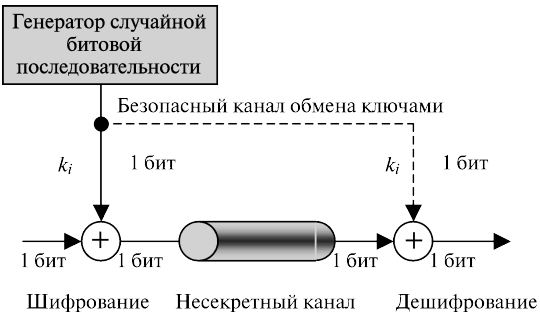
Шифры потока быстрее, чем блочные шифры. Аппаратная реализация шифра потока также более проста. Когда мы должны зашифровать двоичные потоки и передать их на постоянной скорости, лучший выбор — использовать шифр потока. Шифры потока обладают большей защитой против искажения битов в течение передачи.

В современном шифре потока каждое *r*-битовое слово в потоке исходного текста зашифровано, используя *r*-битовое слово в ключевом потоке, чтобы создать соответствующее *r*-битовое слово в потоке зашифрованного текста.

Рассматривая рис. 5.21, можно предположить, что главная проблема в современных шифрах потока — как генерировать ключевой поток $K = k_n ..... k_2 k_1$. Современный шифр потока можно разделить на две обширные категории: синхронный и несинхронный.

Синхронные шифры потока

В **синхронном шифре потока** ключевой поток независим от потока зашифрованного текста или исходного текста. Генерируемый ключевой поток не имеет



никакой связи между ключевыми битами и исходным текстом или битами зашифрованного текста.

В синхронном шифре потока ключ независим от исходного текста или зашифрованного текста.

Одноразовый блокнот

Наиболее простой и самый безопасный тип синхронного шифра потока назван **шифром одноразового блокнота**, или, по имени изобретателя, «шифром Вернама[1]». Шифр одноразового блокнота использует ключевой поток, который беспорядочно выбран для каждой шифровки. Алгоритмы шифрования и дешифрования применяют единственную операцию — ИСКЛЮЧАЮЩЕЕ ИЛИ. Шифры, которые базируются на свойствах операции ИСКЛЮЧАЮЩЕЕ ИЛИ, обсуждались ранее. В них алгоритмы шифрования и дешифрования инверсны друг другу. Важно, что в этом шифре операция ИСКЛЮЧАЮЩЕЕ ИЛИ используется только для одного бита одновременно. Другими словами, операция производится над словом не более чем из одного бита и полем GF (2). Заметим, что также нужно иметь безопасный канал для того, чтобы Алиса могла передать ключевую последовательность потока Бобу (рис. 5.22).

[1] Система симметричного шифрования, изобретенная в 1917 году сотрудниками AT&T Гильбертом Вернамом и Меджером Джозефом Моборном.

Рис. 5.22. Одноразовый блокнот

Одноразовый блокнот — идеальный шифр. Он совершенен. Нет метода, который дал бы противнику возможность распознать ключ или статистику зашифрованного текста и исходного текста. Нет никаких соотношений между исходным и зашифрованным текстами. Другими словами, зашифрованный текст – настоящий случайный поток битов, даже если получить некоторые образцы исходного текста. Ева не может нарушить шифр, если она не попробует все возможные случайные ключевые потоки, которые были бы 2^n, если размер исходного текста — n-битовый. Однако при этом есть проблема. Передатчик и приемник для того, чтобы совместно использовать одноразовый блокнот ключей, должны устанавливать соединение каждый раз, когда они хотят обменяться информацией. Они должны, так или иначе, договориться о случайном ключе. Так что этот совершенный и идеальный шифр очень трудно реализовать.

Пример 5.17

Какой вид имеет зашифрованный текст при использовании шифра одноразового блокнота в каждом из следующих случаев?

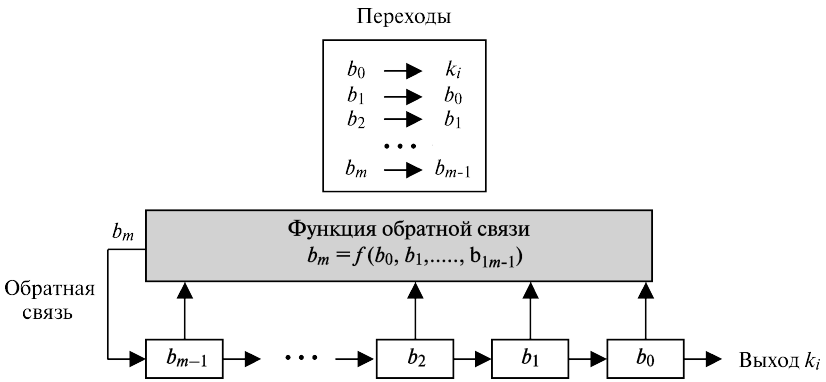
а. Исходный текст состоит из n нулей.

б. Исходный текст состоит из n единиц.

в. Исходный текст состоит из чередующихся нулей и единиц.

г. Исходный текст — случайная строчка бит.

Решение



a. Поскольку $0 \oplus k_i = k_i$, поток зашифрованного текста совпадет с ключевым потоком. Если ключ случайный, зашифрованный текст также случайный. Отрывки исходного текста в зашифрованном тексте не сохраняются.

b. Поскольку $1 \oplus k_i = \overline{k_i}$, где $\overline{k_i}$ является дополнением, поток зашифрованного текста — дополнение ключевого потока. Если ключевой поток случай-

ный, то зашифрованный текст также случайный, отрывки исходного текста не сохраняются в зашифрованном тексте.

c. В этом случае каждый бит в потоке зашифрованного текста является или тем же самым, что и в ключевом потоке, или его дополнением. Поэтому результат — также случайный, если ключевой поток случайный.

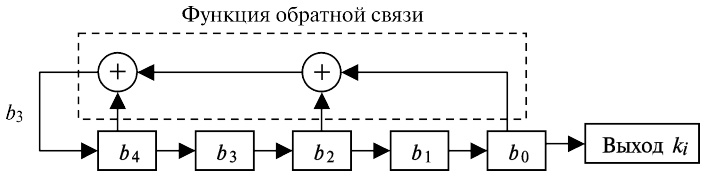
d. В данном случае зашифрованный текст явно случайный, потому что проведение операции ИСКЛЮЧАЮЩЕЕ ИЛИ двух случайных битов в результате дает случайный поток бит.

Регистр сдвига с обратной связью

Имеется одно усовершенствование к одноразовому блокноту — **Регистр сдвига с обратной связью** (FSR — Feedback Shift Register). FSR может быть реализован или в программном обеспечении, или в аппаратных средствах, но для простоты мы рассмотрим аппаратную реализацию. **Регистр сдвига с обратной связью** состоит из **регистра сдвига** и **функции обратной связи,** как показано на рис. 5.23.

Рис. 5.23. Регистр сдвига с обратной свчзью (FSR)

Регистр сдвига – последовательность из m ячеек от b_0 до b_{m-1}, где каждая ячейка предназначена для сохранения единственного бита. Ячейки рассматриваются как n-битовое слово, называемое в начале «начальное значение» или **источник.** Всякий раз, когда необходимо получить бит на выходе (например, по сигналу в определенное время), каждый бит сдвигается на одну ячейку вправо. Это означает, что значение каждой ячейки присваивается правой соседней ячейке и принимает значение левой ячейки. Самая правая ячейка b_0 считается выходом и дает выходное значение (k_i). Крайняя левая ячейка, b_{m-1}, получает свое значение согласно значению информации функции обратной связи. Обозначаем выход функции с информацией обратной связи b_m. Функция информации обратной связи определяет, какие значения имеют ячейки, чтобы вычислить b_m. Регистр сдвига информации обратной связи может быть линейный или нелинейный.

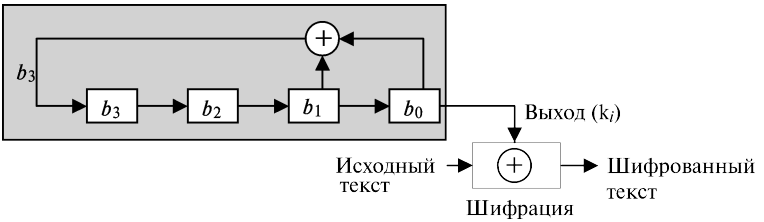


Регистр сдвига с линейной обратной связью (LFSR). Примем, что b_m — это линейная функция $b_0, b_1, \ldots\ldots, b_{m-1}$, для которой

$$b_m = c_{m-1}\, b_{m-1} + \ldots\ldots + c_2 b_2 = c_1 b_1 = c_0 b_0 \quad (c_0 \bullet 0)$$

Линейный регистр сдвига с обратной связью работает с двоичными цифрами, поэтому умножение и сложение находятся в поле GF(2), так что значение c_i является или 1, или 0, но c_0 должно быть 1, чтобы получить информацию обрат-

ной связи на выходе. Операция сложения – это операция ИСКЛЮЧАЮЩЕЕ ИЛИ. Другими словами,



$$bm = c_{m-1}\, b_{m-1} \oplus \ldots\ldots \oplus c_2 b_2 \oplus c_1 b_1 \oplus c_0 b_0 \quad (c_0 \neq 0)$$

Пример 5.18

Построим линейный регистр сдвига с обратной связью с 5-ю ячейками, в которых $b_5 = b_4 \oplus b_2 \oplus b_0$.

Решение

Если $c_i = 0$, b_i не играет роли в вычислении b_m, то это означает, что b_i не связан с функцией информации обратной связи. Если $c_i = 1$, b_i включается в вычис-

Текущее значение	b_4	b_3	b_2	b_1	b_0	k_i
Начальное значение	1	0	0	0	1	
1	0	1	0	0	0	1
2	0	0	1	0	0	0
3	1	0	0	1	0	0
4	1	1	0	0	1	0
5	0	1	1	0	0	1
6	1	0	1	1	0	0
7	0	1	0	1	1	0
8	1	0	1	0	1	1
9	1	1	0	1	0	1
10	1	1	1	0	1	0
11	1	1	1	1	0	1
12	0	1	1	1	1	0
13	0	0	1	1	1	1
14	0	0	0	1	1	1
15	1	0	0	0	1	1
16	0	1	0	0	0	1
17	0	0	1	0	0	0
18	1	0	0	1	0	0
19	1	1	0	0	1	0
20	1	1	1	0	0	1

ление b_m. В этом примере c_1 и c_3 — нули, это означает, что мы имеем только три подключения. Рисунок 5.24 показывает схему линейного регистра сдвига с обрат-

ной связью.

Рис. 5.24. Линейный регистр сдвига с обратной связью

Пример 5.19

Построим линейный регистр сдвига с обратной связью с 4-мя ячейками, в которых $b_4 = b_1 \oplus b_0$. Покажите значение регистра после 20 операций (сдвигов), если исходное значение — $(0001)_2$.

Решение

Рисунок 5.25 показывает схему и использование линейного регистра сдвига с обратной связью для шифрования.

Рис. 5.25. Линейный регистр сдвига с обратной связью для примера 5.19

Таблица 5.6. показывает значение потока ключей. Для каждого перехода сначала вычисляется значение b4, а затем каждый бит сдвигается на одну ячейку вправо.

Таблица 5.6. Значение и последовательность ключей для примера 5.18.

Заметим, что поток ключей — 1000100110101111 1001..... . Он выглядит, на первый взгляд, как случайная последовательность, но если просмотреть большое число транзакций (сдвигов), мы можем увидеть, что последовательности периодичны. Это повторение по 15 бит показано ниже.

1000100110101111 **000100110101111** 000100110101111 **000100110101111**

Ключ потока генерирует с помощью линейного регистра сдвига с обратной связью псевдослучайную последовательность, в которой повторяются последовательности длиною N. Поток периодичен. Он зависит от схемы генератора и начальной информации и может быть не свыше $2^m - 1$. Каждая схема порождает m-битовые последовательности от содержащих все нули до содержащих все единицы. Однако если начальная последовательность состоит только из нулей, результат бесполезен — исходный текст был бы потоком из одних нулей. Поэтому такая начальная последовательность исключена.

Максимальный период последовательностей, которые генерируются с помощью линейного регистра сдвига с обратной связью, — $2^m - 1$.

В предыдущем примере максимальный период — $(2^4 - 1 = 15)$. Чтобы достичь этой максимальной периодичности (наилучшей рандомизации), мы должны в первую очередь представить функцию обратной связи как **характеристический полином** с коэффициентами в поле GF(2).

$$b_m = c_{m-1}b_{m-1} + \ldots\ldots + c_1b_1 + c_0b_0 \rightarrow x^m = c_{m-1}x^{m-1} + \ldots\ldots + c_1x^1 + c_0x^0$$

Поскольку сложение и вычитание в этом поле одни и те же, все элементы могут быть перенесены в одну сторону, что дает полином степени m. (называемый **характеристическим полиномом**).

$$x^m + c_{m-1}x^{m-1} + \ldots\ldots + c_1x^1 + c_0x^0 = 0$$

Линейный регистр сдвига с обратной связью имеет максимальный период $2^m - 1$, если он имеет четное число ячеек, и характеристический полином — **примитивный полином.** Примитивный полином — неприводимый полином, который является делителем $x^e - 1$, где e — наименьшее целое число в форме $e = 2^k - 1$ и $k \geqslant 2$. Примитивный полином получить нелегко. Полином выбирается случайно, а затем проверяется на примитивность. Однако существуют таблицы проверенных примитивных полиномов (см. приложение G).

Пример 5.20

Характеристический полином для линейного регистра сдвига с обратной связью в примере 5.19 — $(x^4 + x + 1)$ — является примитивным полиномом. Таблица 4.4 (лекция 4) показывает, что это — неприводимый полином. Этот полином также делит $(x^7 + 1) = (x^4 + x + 1)(x^3 + 1)$, что означает $e = 2^3 - 1 = 7$.

Атаки шифров, полученных с помощью линейных регистров сдвига с обратной связью. Линейный регистр сдвига с обратной связью имеет очень простую структуру, но эта простота делает шифр уязвимым к атакам. Два общих типа атаки приведены ниже.

1. Если структура линейного регистра сдвига с обратной связью известна, то после перехвата и анализа одного n-битового куска зашифрованного текста Ева может предсказать все будущие зашифрованные тексты.
2. Если структура линейного регистра сдвига с обратной связью неизвестна, Ева может использовать атаку знания исходного текста *длиной 2n* бит, чтобы вскрыть шифр.

Регистр сдвига с нелинейной обратной связью (NLFSR). Линейный регистр сдвига с обратной связью уязвим главным образом из-за его линейности. Более устойчивый шифр потока может быть получен при использовании **нелинейного регистра сдвига с обратной связью.** Он имеет ту же структуру, что и линейный регистр сдвига с обратной связью, за исключением того, что b_m — нелинейная функция $b_0, b_1, \ldots, b_m$. Например, в 4-битном **нелинейном регистре сдвига с обратной связью** структура определяется соотношением, показанным ниже, где операция AND означает поразрядную операцию И, а OR означает поразрядную операцию ИЛИ. Черточка над переменной означает инверсию.

$$b_4 = (b_3 \text{ AND } b_2) \text{ or } (b_1 \text{ AND } \overline{b}_0)$$

Однако нелинейный регистр сдвига с обратной связью не имеет общего характера, поскольку нет математического обоснования, как получить такой регистр с максимальным периодом.

Можно применить линейный регистр сдвига с обратной связью с максимальным периодом и затем скомбинировать его обратную связь с помощью нелинейной функции.

Несинхронные шифры потока

В несинхронном шифре потока каждый ключ в ключевом потоке зависит от предыдущего исходного текста или зашифрованного текста.

В несинхронном шифре потока ключ зависит либо от исходного текста, либо от зашифрованного текста.

Два метода, которые используются, чтобы создать различные режимы работы для блочных шифров (*режим обратной связи по выходу* и *режим счета сцеплений блоков шифра*), фактически создают шифры потока (см. лекцию 8).

5.3. Рекомендованная литература

Нижеследующие книги и сайты дают более детальные сведения по обсуждаемым вопросам, которые рассмотрены в этой лекции. Ссылки, помещенные в скобки, приведены в списке в конце книги.

Книги

[Sti06] и [PHS03] содержат полные сведения о P-блоках и S-блоках. Поточные шифры тщательно рассмотрены в [Sch99] и [Sal03]. [Sti06], [PHS03] и [Vau06] — полный и интересный анализ дифференциального и линейного криптоанализа.

Сайты

Нижеследующие сайты дают более подробную информацию о темах, обсужденных в этой лекции.

http://en.wikipedia.org/wiki/FeisteL_cipher
http://www.quadibloc.com/crypto/co040906.htm
tigger.uic.edu/~jleon/mcs425-s05/handouts/feistal-diagram.pdf

5.4. Итоги

- Традиционные шифры с симметричным ключом — шифры, ориентированные на символ. С появлением компьютера стали нужны шифры, ориентированные на биты.
- Современный симметричный ключевой блочный шифр зашифровывает n-битный блок исходного текста или расшифровывает n-битовый блок зашифрованного текста. Алгоритмы шифрования или дешифрования используют k-битные ключи.

- Современный блочный шифр может быть спроектирован так, чтобы действовать как шифр подстановки или шифр транспозиции. Однако чтобы быть стойким к атаке исчерпывающего поиска, современный блочный шифр должен быть спроектирован как шифр подстановки.
- Современные блочные шифры — обычно ключевые шифры подстановки, в которых ключ практически позволяет отображение всех возможных входов во все возможные выходы.
- Современный блочный шифр состоит из комбинации P-блоков, модулей подстановки, S-блоков и некоторых других модулей.
- P-блок (блок перестановки) подобен традиционному шифру транспозиции для символов. Есть три типа P-блоков: прямые P-блоки, P-блоки расширения и P-блоки сжатия.
- S-блок (блок подстановки) можно представить себе как маленький блок шифра подстановки.
  Однако в S-блоке может быть различное число входов и выходов.
- Операция ИСКЛЮЧАЮЩЕЕ ИЛИ — важный компонент в большинстве блочных шифров: она представляет операции сложения или вычитания в поле GF (2).
- В современных блочных шифрах часто применяется операция циклического сдвига, в которой смещение может быть влево или вправо. Операция перестановки — специальный случай операции циклического сдвига, где $k = n/2$. Две других операции, применяемые в некоторых блочных шифрах, — разбиение и комбинирование.
- Шеннон ввел понятие составного шифра. Составной шифр — сложный шифр, объединяющий S-блоки, P-блоки и другие компоненты, чтобы достигнуть рассеивания и перемешивания. Рассеивание скрывает отношения между исходным текстом и зашифрованным текстом, перемешивание скрывает отношения между ключом шифра и зашифрованным текстом.
- Современные блочные шифры — все составные шифры, но они разделены на два класса: шифры не-Файстля и шифры Файстеля. Шифры Файстеля используют и обратимые, и необратимые компоненты. Шифры не-Файстля используют только обратимые компоненты.
- Некоторые новые атаки блочных шифров базируются на структуре современных шифров. Эти атаки используют дифференциальные и линейные методы криптоанализа
- В современном шифре потока каждое слово r-бита в потоке исходного текста зашифровано, для чего используется r-битовое слово в потоке ключей, чтобы создать соответствующее r-битовое слово в потоке зашифрованного текста. Современные шифры потока могут быть разделены на две обширные категории: синхронные шифры потока и несинхронный шифр потока в синхронном шифре потока. В первом случае ключевой поток независим от потока зашифрованного текста или исходного текста. В несинхронном шифре потока ключевой поток зависит от исходного текста или потока зашифрованного текста.

- Самый простой и самый безопасный тип синхронного шифра потока назван одноразовым блокнотом. Шифр одноразового блокнота использует ключевой поток ключей, который выбран беспорядочно для каждого шифрования. Алгоритмы шифрования и дешифрования применяют операцию ИСКЛЮЧАЮЩЕЕ ИЛИ. Шифр одноразового блокнота неудобен на практике, потому что ключ должен быть индивидуальным для каждого сеанса связи. Один из компромиссных вариантов одноразового блокнота — регистр сдвига с обратной связью (FSR), который может быть реализован в аппаратных средствах или программном обеспечении.

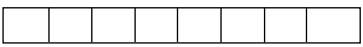
5.5. Вопросы и упражнения

Обзорные вопросы

1. Укажите различия между современным и традиционным шифрами с симметричным ключом.
2. Объясните, почему современные блочные шифры спроектированы как шифры подстановки вместо того, чтобы применять шифры транспозиции.
3. Объясните, почему шифр подстановки можно представить себе как шифр транспозиции.
4. Перечислите некоторые компоненты современного блочного шифра.
5. Определите P-блок и перечислите его три варианта. Какой вариант является обратимым?
6. Определите S-блок и покажите необходимое условие обратимости S-блока.
7. Определите составной шифр и перечислите два класса составных шифров.
8. Укажите различие между рассеиванием и перемешиванием
9. Укажите различие между блочным шифром Файстеля и не-Файстеля.
10. Укажите различие между дифференциальным и линейным криптоанализом. Какой из них использует атаку выборки исходного текста? Какой из них использует также атаку знания исходного текста?
11. Укажите различие между синхронным и несинхронным шифрами потока.
12. Определите регистр сдвига с обратной связью и перечислите два варианта, используемые в шифре потока.

Упражнения

1. Блок транспозиции имеет 10 входов и 10 выходов. Каков порядок группы перестановки? Каков размер ключевой последовательности?
2. Блок подстановки имеет 10 входов и 10 выходов. Каков порядок группы перестановки? Каков размер ключевой последовательности?
3.
   a. Покажите результат циркулярного левого сдвига на 3 бита на слове $(10011011)_2$.



б. Покажите результат циркулярного правого сдвига на 3 бита на слове, полученном в пункте а.

в. Сравните результат пункта б с первоначальным словом пункта а.

4.

а. Измените слово $(10011011)_2$ с помощью перестановки.

б. Измените слово, полученное по пункту а, с помощью перестановки.

в. Сравните результаты пункта а и пункта b, чтобы показать, что перестановка — самообратимая операция.

5. Найдите результат следующих операций:

а. (01001101) ⊕ (01001101)

б. (01001101) ⊕ (10110010)

в. (01001101) ⊕ (00000000)

г. (01001101) ⊕ (11111111)

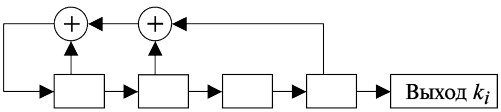
6.

а. Расшифруйте слово 010, используя декодер 3 × 8.

б. Зашифруйте слово 00100000, используя кодирующее устройство 8 × 3.

Вход: левый бит \ Вход: правый бит	0	1
0	01	11
1	10	00

7. Сообщение имеет 2000 символов. Оно будет зашифровано с использованием блочного шифра 64 битов. Найдите размер дополнения и номера блоков.
8. Покажите таблицу перестановки для прямого P-блока на рис. 5.4
9. Покажите таблицу перестановки для P-блока сжатия на рис. 5.4.



10. Покажите таблицу перестановки для P-блока расширения на рис. 5.4.
11. Покажите P-блок, определенный следующей таблицей:

8 1 2 3 4 5 6 7

12. Определите, является ли P-блок со следующей таблицей перестановки прямым P-блоком, P-блоком сжатия или P-блоком расширения.

1 1 2 3 4 4

13. Определите, является ли P-блок со следующей таблицей перестановки прямым P-блоком, P-блоком сжатия или P-блоком расширения.

1 3 5 6 7

14. Определите, является ли P-блок со следующей таблицей перестановки прямым P-блоком, P-блоком сжатия или P-блоком расширения.

1 2 3 4 5 6

15. Отношение вход-выход в 2 × 2 S-блока показаны в следующей таблице для S-блока. Покажите таблицу для инверсного блока.

16. Покажите LFSR с характеристическим полиномом $x^5 + x^2 + 1$. Каков период получаемой последовательности?
17. Каков характеристический полином следующего LFSR? Каков максимальный период?

18. Покажите ключевой поток на 20 битов, сгенерированный от LFSR на рис. 5.25, если начальное значение — 1110.
19. Максимальная длина периода LFSR — 32. Сколько битов имеет регистр сдвига?
20. 6 × 2 S-блок производит операцию ИСКЛЮЧАЮЩЕЕ ИЛИ с нечетными битами, чтобы получить левый бит выхода, и ИСКЛЮЧАЮЩЕЕ ИЛИ с четными битами, чтобы получить правый бит выхода. Если вход — 110010, что является выходом? Если вход — 101101, что является выходом?
21. Крайний левый бит 4 × 3 S-блока определяет смещение других трех бит. Если крайний левый бит равен 0, то три других бита перемещаются вправо на один бит. Если крайний левый бит — 1, три других бита перемещаются влево на один бит. Если вход — 1011, какой результат будет на выходе? Если вход — 0110, какой результат будет на выходе?
22. Напишите процедуру в псевдокоде для разбиения ***n-битового*** слова на два слова, каждое из которых состоит из n/2.
23. Напишите процедуру в псевдокоде для объединения двух n/2-битовыхt слов в n-битовое слово.
24. Напишите процедуру в псевдокоде, которая переставляет левые и правые половины n-битового слова.
25. Напишите процедуру в псевдокоде, которая циклически сдвигает в *n*-разрядном слове на k бит влево или вправо, в соответствии с процедурой по п. 24.
26. Напишите процедуру в псевдокоде для P-блока, в котором перестановка определена таблицей.
27. Напишите процедуру в псевдокоде для S-блока, в котором вход-выход определен таблицей.
28. Напишите процедуру в псевдокоде, которая моделирует каждый раунд не-Файстеля, показанный на рис. 5.13.
29. Напишите процедуру в псевдокоде, которая моделирует каждый раунд шифра, показанный на рис. 5.17.
30. Напишите процедуру в псевдокоде, которая моделирует *n*-битовый LFSR.

Лекция 6. Стандарт шифрования данных (DES)

Цели и содержание

В этой лекции мы обсуждаем Стандарт шифрования данных (DES — DATA ENCRIPTION STANDARD) — современный блочный шифр с симметричными ключами. Наши основные цели для этой лекции:

- рассмотреть короткую историю DES;
- определить основную структуру DES;
- описать детали основных элементов DES;
- описать процесс генерации ключей для раундов;
- провести анализ DES.

Особое внимание уделяется тому, как DES использует шифр Файстеля, чтобы достигнуть перемешивания и рассеивания на выходе из битов исходного текста к битам зашифрованного текста.

6.1. Введение

Стандарт шифрования данных (DES) — блочный шифр с симметричными ключами, разработан Национальным Институтом Стандартов и Технологии (NIST – National Institute of Standards and Technology).

История

В 1973 году NIST издал запрос для разработки предложения национальной криптографической системы с симметричными ключами.

Предложенная IBM модификация проекта, названная Lucifer, была принята как DES. DES были изданы в эскизном виде в *Федеральном Регистре* в марте 1975 года как **Федеральный Стандарт Обработки Информации (FIPS – Federal Information Processing Standard).**

После публикации эскиз строго критиковался по двум причинам.

Первая: критиковалась сомнительно маленькая длина ключа (только 56 битов), что могло сделать шифр уязвимым к атаке «грубой силой».

Вторая причина: критики были обеспокоены некоторым скрытым построением внутренней структуры DES.

Они подозревали, что некоторая часть структуры (S-блоки) может иметь лазейку, которая позволит расшифровывать сообщения без ключа. Впоследствии проектировщики IBM сообщили, что внутренняя структура была доработана, чтобы предотвратить криптоанализ.

DES был наконец издан как FIPS 46 в Федеральном Регистре в январе 1977 года. Однако FIPS объявил DES как стандарт для использования в неофициальных приложениях. DES был наиболее широко используемым блочным шифром с симметричными ключами, начиная с его публикации. Позже NIST предложил новый стандарт (FIPS 46-3), который рекомендует использование тройного DES (трехкратно повторенный шифр DES) для будущих приложений. Как мы увидим далее, в лекции 7, предполагается, что более новый стандарт AES заменит DES.

Общие положения

Как показано на рис. 6.1., DES — блочный шифр.

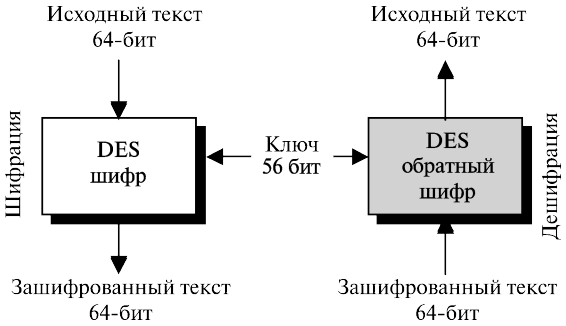


Рис. 6.1. Шифрование и дешифрование в DES

На стороне шифрования DES принимает 64-битовый исходный текст и порождает 64-битовый зашифрованный текст; на стороне дешифрования DES принимает 64-битовый зашифрованный текст и порождает 64-битовый исходный текст. На обеих сторонах для шифрования и дешифрования применяется один и тот же 56-битовый ключ.

6.2. Структура DES

Рассмотрим сначала шифрование, а потом дешифрование. Процесс шифрования состоит из двух перестановок (P-блоки) — они называются начальные и конечные перестановки, — и шестнадцати раундов Файстеля. Каждый раунд использует различные сгенерированные 48-битовые ключи. Алгоритм генерации будет рассмотрен в этой лекции позднее. Рисунок 6.2 показывает элементы шифра DES на стороне шифрования.

Начальные и конечные перестановки

Рисунок 6.3 показывает начальные и конечные перестановки (P-блоки).

Каждая из перестановок принимает 64-битовый вход и переставляет его элементы по заданному правилу. Мы показали только небольшое число входных портов и соответствующих выходных портов. Эти перестановки — прямые перестановки без ключей, которые инверсны друг другу. Например, в начальной перестановке 58-й бит на входе переходит в первый бит на выходе. Аналогично, в конечной перестановке первый входной бит переходит в 58-й бит на выходе. Другими словами, если между этими двумя перестановками не существует раунда, 58-й бит, поступивший на вход устройства начальной перестановки, будет доставлен на 58-й выход финальной перестановкой.

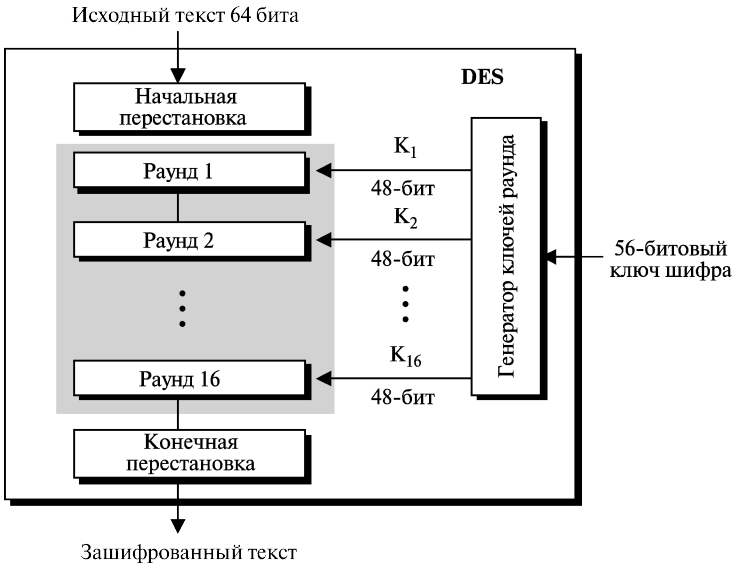


Рис. 6.2. Общая структура DES

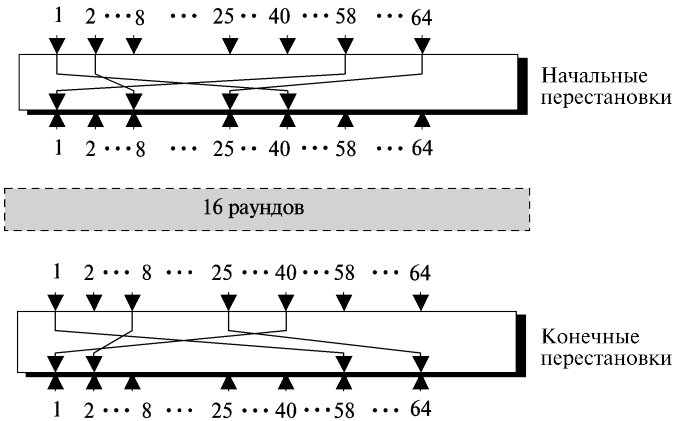


Рис. 6.3. Начальные и конечные шаги перестановки DES

Правила перестановки для этого P-блока показаны в таблице 6.1. Таблицу можно представить как 64-элементный массив. Заметим, что работу с таблицей мы обсуждали: значение каждого элемента определяет номер входного порта, а порядковый номер (индекс) элемента определяет номер выходного порта.

Таблица 6.1. Таблица начальных и конечных перестановок

Начальные перестановки								Конечные перестановки							
58	50	42	34	26	18	10	02	40	08	48	16	56	24	64	32
60	52	44	36	28	20	12	04	39	07	47	15	55	23	63	31
62	54	46	38	30	22	14	06	38	06	46	14	54	22	62	30
64	56	48	40	32	24	16	08	37	05	45	13	53	21	61	29
57	49	41	33	25	17	09	01	36	04	44	12	52	20	60	28
59	51	43	35	27	19	11	03	35	03	43	11	51	19	59	27
61	53	45	37	29	21	13	05	34	02	42	10	50	18	58	26
63	55	47	39	31	23	15	07	33	01	41	09	49	17	57	25

Эти две перестановки не имеют никакого значения для криптографии в DES. Обе перестановки – без ключей и предопределенны. Причина, почему они включены в DES, не ясна и не была указана проектировщиками DES. Можно предположить, что DES был проектом, который предполагалось реализовать в аппаратных средствах (на чипах), и что эти две сложные перестановки должны были затруднить программное моделирование механизма шифрования.

Пример 6.1

Найдите выход начального блока перестановки, когда на вход поступает шестнадцатеричная последовательность, такая, как

0x0002 0000 0000 0001

Решение

Вход имеет только две единицы — (бит 15 и бит 64); выход должен также иметь только две единицы (прямая перестановка). Используя таблицу 6.1, мы можем найти выход, связанный с этими двумя битами. Бит 15 на входе становится битом 63 в выходе. Бит 64 во входе становится битом 25 в выходе. На выходе будем иметь только две единицы — бит 25 и бит 63. Результат в шестнадцатеричном исчислении:

0x0000 0080 0000 0002

Пример 6.2

Докажем, что начальные и финальные перестановки инверсны друг другу. Преобразуем полученную выходную последовательность во входную.

0x0000 0080 0000 0002

Решение

Единичные биты — 25 и 63, другие биты равны нулю. В конечной перестановке 25-й бит переходит в 64-й, а 63-й — в 15-й. Результат:

0x0002000000000001

Начальные и конечные перестановки – это прямые P-блоки, которые инверсны друг другу. Они не имеют значения для криптографии DES.

Раунды

DES использует 16 раундов. Каждый раунд DES применяет шифр Файстеля, как это показано на рис. 6.4.

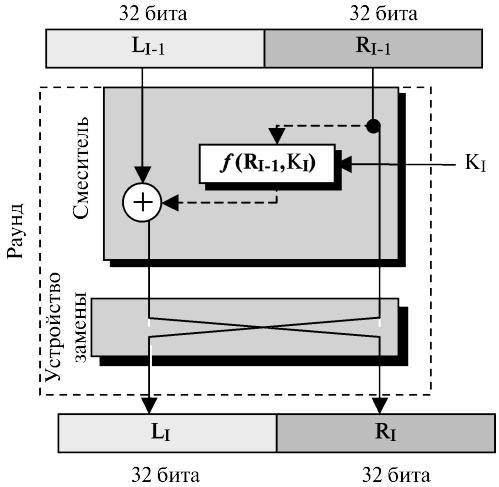


Рис. 6.4. Раунд в DES (сторона шифрования)

Раунд принимает L_{I-1} R_{I-1} от предыдущего раунда (или начального блока перестановки) и создает для следующего раунда L_I и R_I, которые поступают на следующий раунд (или конечный блок перестановки). Как мы указали в лекции 5, можно принять, что каждый раунд имеет два элемента шифра (смеситель и устройство замены). Каждый из этих элементов является обратимым. Устройство замены — очевидно обратимо, оно меняет местами левую половину текста с правой половиной. Смеситель является обратимым, потому что операция ИСКЛЮЧАЮЩЕЕ ИЛИ обратима. Все необратимые элементы сосредоточены в функции $f(R_{I-1},K_I)$.

Функция DES

Основной блок DES — функция DES. Функция DES с помощью 48-битового ключа зашифровывает 32 самых правых бит R_{I-1}, чтобы получить на выходе 32-битовое слово. Эта функция содержит, как это показано на рис. 6.5, четыре секции: отбеливатель (whitener), P-блок расширения, группу S-блоков и прямой P-блок.

P-блок расширения. Так как вход R_{I-1} имеет длину 32 бита, а ключ K_I — длину 48 битов, мы сначала должны расширить R_{I-1} до 48 бит. R_{I-1} разделяется на 8 секций по 4 бита. Каждая секция на 4 бита расширяется до 6 бит. Эта перестанов-

ка расширения происходит по заранее определенным правилам. Для секции значения входных бит 1, 2, 3 и 4 присваиваются битам 2, 3, 4 и 5 соответственно на выходе. Выходной бит 1 формируется на основе входного бита 4 из предыдущей секции; бит выхода 6 формируется из бита 1 в следующей секции. Если секции 1 и 8 рассматривать как соседние секции, то те же самые правила применяются к битам 1 и 32. Рисунок 6.6 показывает входы и выходы в перестановке расширения.

Хотя отношения между входом и выходом могут быть определены математически, но чтобы определить этот P-блок, DES использует таблицу 6.2. Обратите внимание, что число выходов 48, но диапазон значений — только от 1 до 32. Некоторые из входов идут больше к чем одному выходу. Например, значение входного бита 5 становится значением битов выхода 6 и 8.

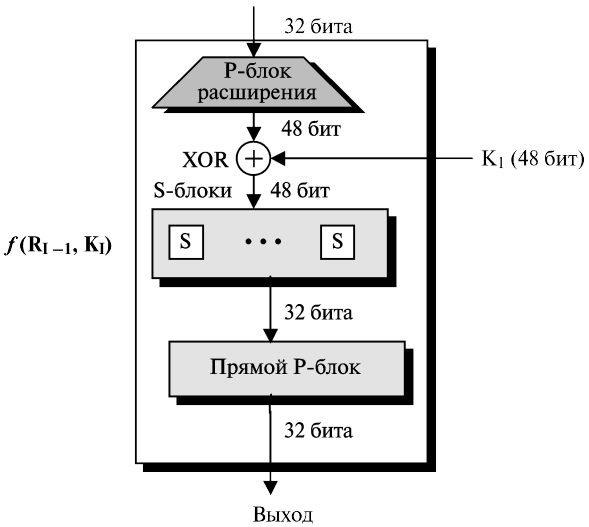


Рис. 6.5. Функция DES

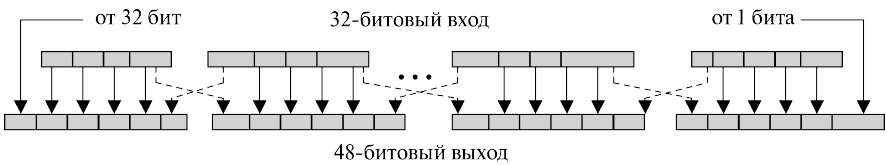


Рис. 6.6. Перестановка расширения

Отбеливатель (whitener). После расширения DES использует операцию XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ) над расширенной частью правой секции и ключом раунда. Заметим, что правая секция и ключ имеют длину 48 бит. Также заметим, что ключ раунда использует только эту операцию.

Таблица 6.2. Таблица P-блока расширения

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

S-блоки. S-блоки смешивают информацию (операция перемешивания). DES использует S-блоки, каждый с 6-ю входными битами и 4-мя выходными (см. рис. 6.7).

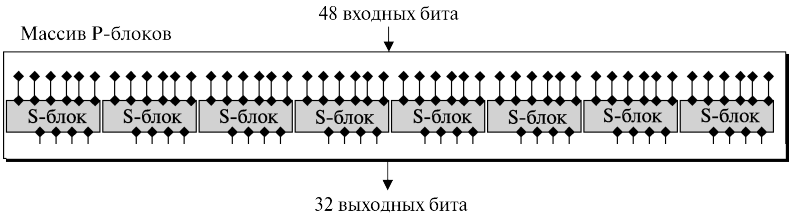


Рис.6.7. S - блоки

Данные из 48 битов от второй операции DES разделены на восемь кусков по 6 битов, и каждый кусок поступает в блок. Результат каждого блока — кусок на 4 бита; когда они объединены, результат выражается в 32-битовом тексте. Подстановка в каждом блоке происходит по заранее определенным правилам, основанным на таблице из 4-х строк и 16-ти столбцов. Комбинация битов 1 и 6 на входе определяет одну из четырех строк; комбинация битов от 2-го до 5-го определяет один из шестнадцати столбцов, как показано на рис. 6.8. Далее мы поясним это примерами.

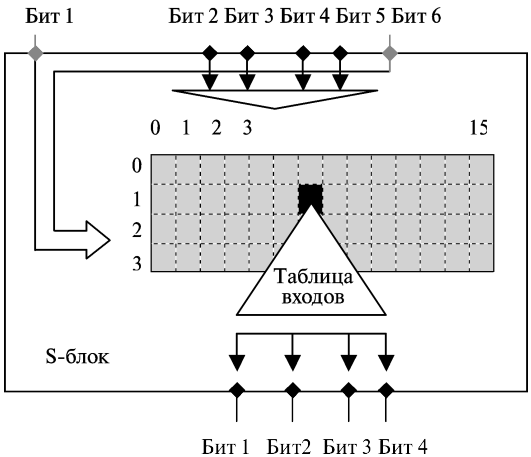


Рис. 6.8. Правила для S- блока

Поскольку для каждого S-блока есть собственная таблица, необходимо иметь восемь таблиц, например таких, как это показано в таблицах 6.3-6.10. Значение входа (номер строки и номер столбца) и значения выхода даются как десятичные номера, чтобы сэкономить место на странице. В реальности они могут быть заменены двоичными числами.

Таблица 6.3. S-блок 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	07	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Таблица 6.4. S-блок 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

Таблица 6.5. S-блок 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	O8
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

Таблица 6.6. S-блок 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	6	09	10	1	02	08	05	1 1	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	1 2	1 1	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

Таблица 6.7. S-блок 5

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

Таблица 6.8. S-блок 6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	10	00	08	13

Таблица 6.9. S-блок 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	00	08	13	03	12	09	07	05	10	06	0l
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

Таблица 6.10. S-блок 8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	1 1	01	10	09	03	14	05	00	1 2	07
1	01	15	13	08	10	03	07	04	12	05	06	1 1	10	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	10	15	03	05	08
3	02	01	14	07	04	10	S	13	15	1 9	09	09	03	05	06	11

Пример 6.3

Входная последовательность S блока 1 — **1**0001**1**. Какая последовательность будет на выходе?

Решение

Если мы запишем первый и шестой биты вместе, мы получим в двоичном исчислении 11, который выражается как число 3 при десятичном исчислении. Остающаяся часть битов 0001 в двоичном исчислении является 1 в десятичном исчислении. Мы ищем значение строки 3 и столбца 1 в Таблице 6.3 (S-блок 1). Результат — 12 в десятичном исчислении или 1100 в двоичном исчислении. Тогда вход 1100011 дает выход 1100.

Пример 6.4

Входная последовательность S-блока 8 — **0**0000**0**. Какая последовательность будет на выходе?

Решение

Если мы запишем первый и шестые биты вместе, то получим в двоичном исчислении число 00, которое выражается числом 0 при десятичном исчислении. Остающаяся часть битов — 0000 в двоичном исчислении, то есть 0 в десятичном исчислении. Мы ищем значение строки 0 и столбца 0 в таблице 6.10 (S-блок 8). Результат — 13 в десятичном исчислении или 1101 в двоичном исчислении. Тогда вход 000000 дает выход 1101.

Прямая перестановка — последняя операция в функции DES — перестановка с 32 битами на входе и 32 битами на выходе. Отношения «вход-выход» для этой операции показаны в Таблице 6.11. Они следуют тем же общим правилам, как и предыдущие таблицы перестановки. Например, седьмой бит входа становится вторым битом выхода.

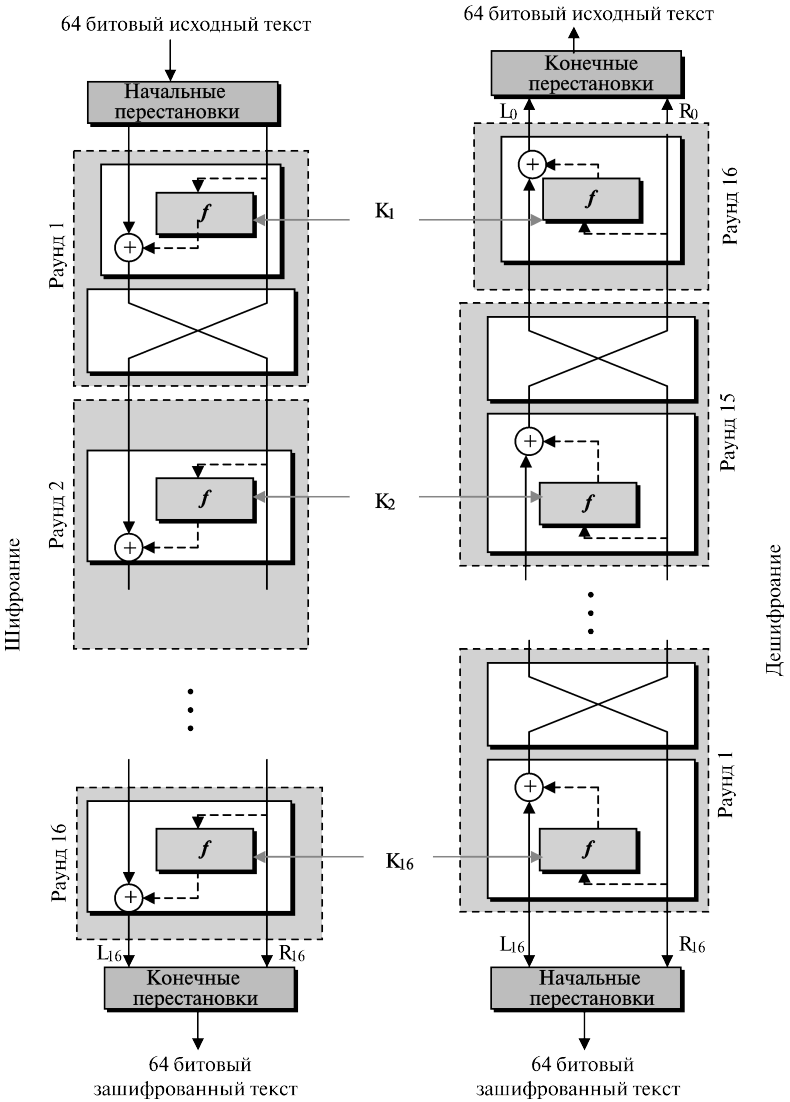


Рис. 6.9. DES-шифр и обратный шифр для первого способа

Таблица 6.11. Таблица прямой перестановки

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Шифр и обратный шифр

Используя смеситель и устройство замены, мы можем создать шифр и обратный шифр для каждого из 16-ти раундов. Шифр применяется на стороне шифрования; обратный шифр — на стороне дешифрования. Алгоритмы создания шифра и обратного шифра аналогичны.

Первый способ

Один из методов, чтобы достигнуть поставленной цели (шифрование и обратное шифрование), состоит в том, чтобы сделать последний раунд отличающимся от других; он будет содержать только смеситель и не будет содержать устройства замены, как это показано на рис. 6.9.

Мы доказали в лекции 5, что смеситель и устройство замены самоинверсны. Конечные и начальные перестановки также инверсны друг другу. Левая секция исходного текста на стороне шифрования шифруется L_0 как L_{16}, и L_{16} дешифруется на стороне дешифратора как L_0. Аналогичная ситуация с R_0 и R_{16}.

Нужно запомнить очень важное положение, которое касается шифров: ключи раундов (K_1 и K_{16}) применяются при шифровании и дешифровании в обратном порядке. На стороне шифрования первый раунд применяет ключ K_1, а раунд 16 — ключ K_{16}; при дешифровании раунд 1 использует ключ K_{16}, а раунд16 — ключ K_1

В первом методе последний раунд не имеет устройства замены.

Алгоритм

Алгоритм 6.1. приведен в псевдокодах для шифрования и соответствует четырем шагам первого метода. Коды для остальных могут быть сделаны как упражнение.

Альтернативный способ

При первом способе раунд 16 отличается от других раундов тем, что там не применяется устройство замены. Это необходимо, чтобы сделать последний и первый смесители в шифре одинаковыми. Мы можем делать все 16 раундов одинаковыми, добавляя к 16-му раунду дополнительное устройство замены (два устройства замены позволяют нейтрализовать друг друга). Разработку этой схемы мы оставляем для упражнений.

Генерация ключей

Генератор ключей создает шестнадцать ключей по 48 битов из ключа шифра на 56 битов. Однако ключ шифра обычно дается как ключ из 64-х битов, в кото-

Алгоритм 6.1. Псевдокоды для DES-шифра

```
Cipher (plainBlock[64], Round Keys[16,48])
{
    permute (64,64, plainBlock, inBlock, IntialPermutationTable)
    spilt (64, 32, inBlock, leftBlock, right Block)
        for (round = 1 to 16)
        {
            mixer (leftBlock, right Block, RoundKeys[round])
            if (round!=16) swapper(leftBlock, right Block)
        }
        combine (32, 64, leftBlock, right Block, outBlock)
        permute (64,64, outBlock, cipherBlock, FinalPermutationTable)
}
mixer (leftBlock[48], right Block[48], RoundKey[48])
{
    copy (32, rightBlock, T1)
    function (T1, RoundKey,T2)
    exclusiveOr (32, leftBlock, T2,T3)
    copy (32, rightBlock, T1)
}
swapper { leftBlock[32], right Block[32])
{
    copy (32, leftBlock, T)
    copy (32, rightBlock, , leftBlock)
    copy (32, T, rightBlock)
}
Function (inBlock[32], RoundKey[48],outBlock[32])
{
    permute (32,48, inBlock,T1, ExpansionPermutationTable)
    exclusiveOr (48,T1, RoundKey, T2)
    substitute (T2,T3, SubstituteTable)
    permute (32,32,T3, outBlock, StraightPermutationTable)
}
Substitute (in block [32], outblock [48], SubstituteTables[8,4,16])
{
    for(I =1 to8)
    {
        row ← 2 × inBlock[i × 6+1] + inBlock[i × 6+6]
    col ← 8 × inBlock[i × 6+2] + 4 × inBlock[i × 6+3]+
            2 × inBlock[i × 6+4] + inBlock[i × 6+5]

    value = SubstituteTables[i][row][col]

    outBlock[i × 4+1] ← value/ 8    value ← value mod 8
    outBlock[i × 4+2] ← value/ 4    value ← value mod 4
    outBlock[i × 4+3] ← value/ 2    value ← value mod 8
    outBlock[i × 4+4] ← value
    }
}
```

ром 8 дополнительных битов являются битами проверки. Они отбрасываются перед фактическим процессом генерации ключей, который показан на рис. 6. I0.

Удаление битов проверки

Предварительный процесс перед расширением ключей — перестановка сжатия, которую мы называем **удалением битов проверки.** Она удаляет биты четности (биты 8, 16, 24, 32..., 64) из 64-битового ключа и переставляет остальную часть битов согласно таблице 6.12. Остающееся значение на 56 битов — фактический ключ шифра, который используется, чтобы генерировать ключи раунда. Биты удаляются с помощью перестановки (Р-блока сжатия), как это показано в таблице 6.12.

Таблица 6.12. Таблица удаления проверочных битов

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	37
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Сдвиг влево

После прямой перестановки ключ разделен на две части по 28 битов. Каждая часть сдвигается влево (циклический сдвиг) на один или два бита. В раундах 1, 2, 9 и 16 смещение – на один бит, в других раундах — на два бита. Затем эти две части объединяются, чтобы создать часть в 56 бит Таблица 6.13 показывает число сдвигов манипуляций для каждого раунда.

Таблица 6.13. Число сдвигаемых бит

Раунд	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Число бит	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Перестановка сжатия

Перестановка сжатия (Р-блок) изменяет 56 битов на 48 битов, которые используются для формирования ключа раунда. Перестановка сжатия показана в таблице 6.14.

Таблица 6.14. Таблица сжатия ключа

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

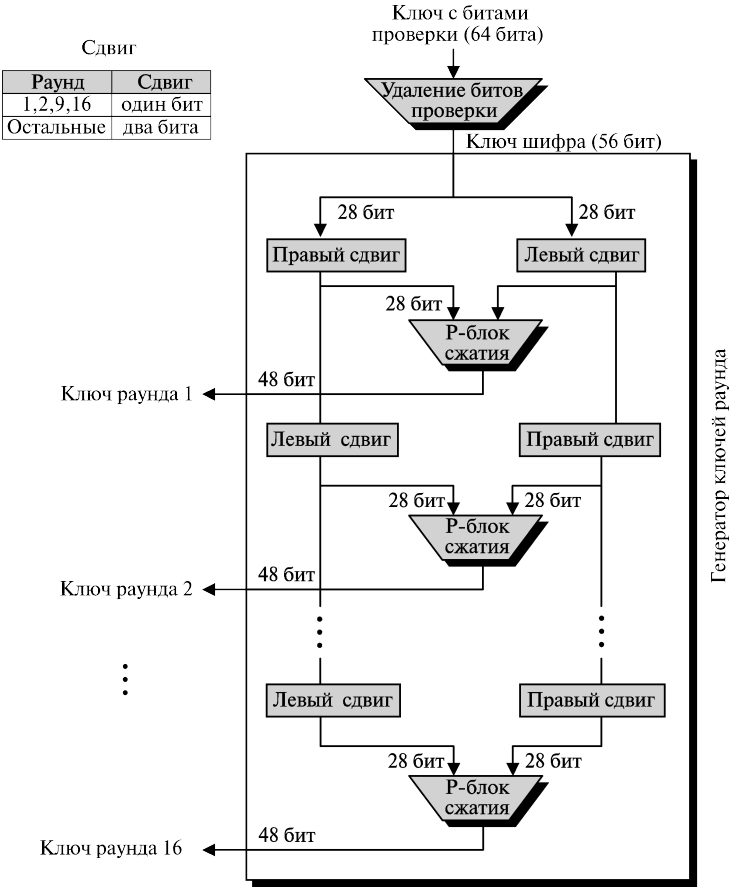


Рис. 6.10. Генерация ключей

Алгоритм

Теперь напишем простой алгоритм для создания ключа из ключа с проверочными битами. Алгоритм 6.2 использует несколько процедур алгоритма 6.1. Имеется и одна новая процедура — левый сдвиг (ShiftLeft). Обратите внимание, что T — временный блок.

Алгоритм 6.2. Алгоритм для генерации ключей раунда

```
Key_Generator (keyWithParities[64], RoundKeys[16. 48]. SinfTable[16])
{
   permute (64, 56, keyWithParities, cipherKey, ParityDropTable)
   spilt (56, 28, cipherKey, leftKey, rightKey)
   for (round = 1 to 16)
```

```
        {
            shiftLeft (leftKey, ShiftTable[round1])
            shiftLeft (rightKey, ShiftTable [round])
            combine (28, 56, leftKey, rightKey, preRoundKey)
            permute (56, 48, preRoundKey, RoundKeys[ round],
              KevCompressionTable)
        }
    }

    shiftLeft (block[28], NumOfShifts)
    {
        for (i = 1 to numShifts)
        {
            T <-   block[ 1]
            for (j = 2 to 28)
            {
               block [j-1] <- block [j]
            }
            block [28] <- T
    }
```

Примеры

Перед анализом DES рассмотрим несколько примеров, чтобы понять, как шифрование и дешифрование меняют значение битов в каждом раунде.

Пример 6.5

Мы выбираем случайный блок исходного текста и случайный ключ и определяем, каким должен быть блок зашифрованного текста (все цифры даны в шестнадцатеричном исчислении).

Plaintext: 123456ABCD 132536 Key: AABB09182736CCDD
Cipher Text: COB7ASD05F3AS29C

Покажем результат каждого раунда и текста, созданного до и после раундов. Таблица 6.15 показывает результаты первых шагов перед началом раунда. Исходный текст — прошедший через начальную перестановку для получения различных 64 бит (16 шестнадцатеричных цифр).

Таблица показывает результат 16 раундов, которые включают смешивание и замену (исключая последний раунд). Результаты последних раундов (L_{16} и R_{16}) объединены. Наконец, текст проходит конечную перестановку, для того чтобы получить зашифрованный текст.

Следует отметить некоторые положения. Правая секция каждого раунда совпадает с левой секцией следующего раунда. Причина в том, что правая секция проходит через смеситель без изменения, а устройство замены переносит ее в левую секцию. Например, R_1 передается через смеситель второго раунда без изме-

Таблица 6.15. Трассировка данных в примере 6.5

Исходный текст: 123456ABCD 132536			
После первоначальной перестановки: 14A7D67818CA18D После разбиения: L_0= 14A7D678 R_0 = 18CA18D			
Раунд	Левая	Правая	Ключ раунда
Раунд 1	18CA18AD	5A78E394	194CD072DE8C
Раунд 2	5A78E394	4A1210F6	4568581ABCCE
Раунд 3	4A1210F6	B8089591	06EDA4ACF5B5
Раунд 4	B8089591	236779C2	DA2D032B6EE3
Раунд 5	236779C2	A15A4B87	69A629FEC913
Раунд 6	A15A4B87	2E8F9C65	C1948E87475E
Раунд 7	2E8F9C65	A9FC20A3	708AD2DDB3C0
Раунд 8	A9FC20A3	308BEE97	34F822F0C66D
Раунд 9	308BEE97	10AF9D37	84BB4473DCCC
Раунд 10	10AF9D37	6CA6CB20	02765708B5BF
Раунд 11	6CA6CB20	FF3C485F	6D5560AF7CA5
Раунд 12	FF3C485F	22A5963B	C2C1E96A4BF3
Раунд 13	22A5963B	387CCDAA	99C31397C91F
Раунд 14	387CCDAA	BD2DD2AB	251B8BC717D0
Раунд 15	BD2DD2AB	CF26B472	3330C5D9A36D
Раунд 16	19BA9212	CF26B472	181C5D75C66D
После объединения: 19BA9212 CF26B472			
Зашифрованный текст: C0B7A8D05F3A829C (после конечной перестановки)			

нения, но затем, пройдя устройство замены, он становится L_2. Второе интересное положение: на последнем раунде мы не имеем устройства замены. Именно поэтому R_{15} становится R_{16} вместо того чтобы стать L_{16}.

Таблица 6.16. Трассировка данных в примере 6.6

Зашифрованный текст: C0B7A8D05F3A829C			
После первоначальной перестановки: 19BA9212 CF26B472 После разбиения: L_0= 19BA9212 R_0 = CF26B472			
Раунд	Левая	Правая	Ключ раунда
Раунд 1	CF26B472	BD2DD2AB	181C5D75C66D
Раунд 2	BD2DD2AB	387CCDAA	3330C5D9A36D
.....	.....	.....	.....
Раунд 15	5A78E394	18CA182AD	4568581ABCCE
Раунд 16	14A7D678	18CA18AD	194CD072DE8C
После объединения: 14A7D67818CA18D			
Исходный текст: 123456ABCD 132536 (после конечной перестановки)			

Пример 6.6

Давайте рассмотрим, как Боб в пункте назначения может расшифровать зашифрованный текст, полученный от Алисы, с помощью совпадающего ключа. Для экономии времени мы разберем только несколько раундов. Таблица 6.16 показывает интересующие нас точки. Первое правило: ключи раунда должны использоваться в обратном порядке. Сравните таблицу 6.15 и таблицу 6.16. Ключ раунда 1 — такой же как ключ для раунда 16. Значения L_0 и R_0 при дешифрации те же самые, что и значения L_{16} и R_{16} при шифровании. Аналогичные совпадения будут получены и в других раундах. Это доказывает не только, что шифр и обратный шифр инверсны друг другу, но также то, что каждый раунд при шифрации имеет соответствующий раунд при дешифрации в обратном шифре. Результат свидетельствует, что начальные и конечные перестановки также являются инверсиями друг друга.

6.3. Анализ DES

DES был подвергнут тщательному анализу. Были проведены испытания, чтобы измерить интенсивность некоторых желательных свойств в блочном шифре. Элементы DES прошли исследования на соответствие некоторым критериям. Ниже мы обсудим некоторые из них.

Свойства

Два желательных свойства блочного шифра — эффект лавины и законченность.

Лавинный эффект

Лавинный эффект означает, что небольшие изменения в исходном тексте (или ключе) могут вызвать значительные изменения в зашифрованном тексте. Было доказано, что DES имеет все признаки этого свойства.

Пример 6.7

Чтобы проверить эффект лавины в DES, попробуем зашифровать два блока исходного текста, которые отличаются только одним битом текста, с помощью одного и того же ключа и определим разницу в числе бит в каждом раунде.

Исходный текст: 0000000000000000 Ключ: 22234512987ABB23
Зашифрованный текст: 4789FD476E82A5F1

Исходный текст: 0000000000000001 Ключ: 22234512987ABB23
Зашифрованный текст: OA4ED5C15A63FEA3

Хотя два блока исходного текста отличаются только самым правым битом, блоки зашифрованного текста отличаются на 29 бит. Это означает, что изменение приблизительно в 1,5 процентах исходного текста создают изменение приблизительно 45 процентов зашифрованного текста. Таблица 6.17 показывает изменение

в каждом раунде. Можно увидеть, что существенные изменения возникают уже в третьем раунде.

Таблица 6.17. Число различных бит в примере 6.7

Раунд	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Разница в битах	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

Эффект полноты

Эффект полноты заключается в том, что каждый бит зашифрованного текста должен зависеть от многих битов исходного текста. Рассеивание и перемешивание, произведенное P-блоками и S-блоками в DES, указывает на очень сильный эффект полноты.

Критерии разработок DES

Проект DES был предъявлен IBM в 1994 году. Многочисленные испытания DES показали, что он удовлетворяет некоторым из заявленных критериев. Ниже кратко обсуждаются некоторые проблемы разработок DES.

S-блоки

Мы рассмотрели общие критерии построения S-блоков в лекции 5. Здесь мы только обсуждаем критерии, выбранные для DES. Структура блоков обеспечивает перемешивание и рассеивание от каждого раунда до следующего. Согласно этому положению и некоторому анализу, мы можем упомянуть несколько свойств S-блоков.

1. Входы каждой строки есть перестановки значений между 0 и 15.
2. S-блоки — нелинейные. Другими словами, выход — не аффинное преобразование. (См. лекции 3 (аффинное преобразование) и 5, где рассматривалась линейность и S-блоков.)
3. Если мы изменяем единственный бит на входе, на выходе будут изменены два или больше бита.
4. Если два входа S-блока отличаются только двумя средними битами (битами 3 и 4), выходная информация должна отличаться, по крайней мере, двумя битами. Другими словами, S (x) и S ($x \oplus 001100$) должны отличаться по крайней мере двумя битами, где x — вход и $S(x)$ — выход.
5. Если два входа в S-блок отличаются первыми двумя битами (биты 1 и 2) и последними двумя битами (5 и 6), два выхода должны быть различны. Другими словами, мы должны иметь следующее отношение: $S(x) \neq S(x \oplus$ 11bc00), в котором b и c — произвольные биты.
6. Есть только 32 шестибитовые пары «вход-выход» (x_i и x_j), в которых $x_i \oplus x_j \neq (000000)_2$. Эти 32 входных пары создают 32 пары слова выхода по 4 бита. Если мы создаем какие-то различия между 32 выходами пар, $d = y_i \oplus y_j$, то из этих d должны быть одинаковыми не больше чем 8.
7. Такой же критерий, как в пункте 6, применяется к трем S-блокам.

8. В любом S-блоке, если единственный входной бит сохраняется как константа (0 или 1), то другие биты изменяются случайно так, чтобы разности между числом нулей и единиц были минимизированы.

P-блоки

Между двумя рядами S-блоков (в двух последующих раундах) есть один прямой P-блок (32 на 32) и один P-блок расширения (32 на 48). Эти два P-блока вместе обеспечивают рассеивание битов. Мы уже говорили об общем принципе построения P-блока в лекции 5. Здесь мы обсудим только прикладные P-блоки, используемые в DES. В структуре P-блоков были реализованы следующие критерии

1. Каждый вход S-блока подключается к выходу другого S-блока (в предыдущем раунде).
2. Ни один вход к данному S-блоку не соединяется с выходом от того же самого блока (в предыдущем раунде).
3. Четыре бита от каждого S-блока идут в шесть различных S-блоков (в следующем раунде).
4. Ни один из двух битов выхода от S-блока не идет в тот же самый S-блок (в следующем раунде).
5. Если число блоков S-блоков 8, то S_1, $S_{2,}$..., S_8.
   а. Выход S_{j-2} переходит в один из первых двух битов S_j (в следующем раунде).
   б. Бит выхода от S_{i-1} переходит в один из последних двух битов $_{Sj}$ (в следующем раунде).
   в. Выход S_{j+1} переходит в один из двух средних битов S_j (в следующем раунде).
6. Для каждого S-блока два бита выхода идут в первые или последние два бита S-блока в следующем раунде. Другие два бита выхода идут в средние биты S-блока в следующем раунде.
7. Если выход от S_j переходит в один из средних битов в S_k (в следующем раунде), то бит выхода от S_k не может идти в средний бит S_j. Если мы допускаем $j = k$, то подразумеваем, что ни один средний бит S-блока не может идти в один из средних битов того же самого S-блока в следующем раунде.

Число раундов

DES используют шестнадцать раундов шифра Файстеля. Доказано, что после того как каждый текст зашифрован за восемь раундов, каждый бит зашифрованного текста — функция каждого бита исходного текста и каждого ключевого бита. Зашифрованный текст — полностью случайная функция исходного текста и зашифрованного текста. Отсюда вроде бы следует, что восьми раундов должно быть достаточно для хорошего шифрования. Однако эксперименты показывают, что некоторые версии DES с менее чем шестнадцатью раундами более уязвимы к атакам знания исходного текста, чем к атаке грубой силы, которая требует использования шестнадцати раундов DES.

Слабости DES

В течение прошлых нескольких лет критики нашли некоторые слабости в DES.

Мы кратко укажем на недостатки, которые были обнаружены в структуре шифра.

S-блоки. В литературе указываются, по крайней мере, три проблемы S-блоков.

1. В S-блоке 4 три бита выхода могут быть получены тем же самым способом, что и первый бит выхода: дополнением некоторых из входных битов.
2. Два специально выбранных входа к массиву S-блока могут создать тот же самый выход.
3. Можно получить тот же самый выход в одном единственном раунде, изменяя биты только в трех соседних S-блоках.

P-блоки. В структуре P-блока были найдены одна загадка и одна слабость.

1. Не ясно, почему проектировщики DES использовали начальную и конечную перестановки. Эти перестановки не вносят никаких новых свойств с точки зрения безопасности.
2. В перестановке расширения (в функции) первые и четвертые биты последовательностей на 4 бита повторяются.

Слабость в ключе шифра

Размер ключа. Критики утверждают, что самая серьезная слабость DES — это размер ключа (56 битов). Чтобы предпринять атаку грубой силы данного блока зашифрованного текста, злоумышленники должны проверить 2^{56} ключей.

a. Применяя доступную сегодня технологию, можно проверить один миллион ключей в секунду. Это означает, что потребуется более чем две тысячи лет, чтобы выполнить атаку грубой силы на DES, используя компьютер только с одним процессором.

b. Если мы сделаем компьютер с одним миллионом чипов процессоров (параллельная обработка), то сможем проверить все множество ключей приблизительно за 20 часов. Когда был введен DES, стоимость такого компьютера была более чем несколько миллионов долларов, но она быстро снизилась. Специальный компьютер был создан в 1998 году — и нашел ключ за 112 часов.

c. Компьютерные сети могут моделировать параллельную обработку. В 1977 году команда исследователей использовала 3500 компьютеров, подключенных к Internet, чтобы найти ключ RSA за 120 дней. Множество ключей было разделено среди всех этих компьютеров, и каждый компьютер был ответственен за проверку части домена DES. Если 3500 связанных в сеть компьютеров могут найти ключ через 120 дней, то секретное общество из 42 000 членов может найти ключ через 10 дней.

Приведенное выше показывает, что DES с размером ключа шифра 56 битов не обеспечивает достаточной безопасности. Позже в этой лекции мы увидим, что есть одно решение этой проблемы – это использование тройного DES(3DES) с двумя ключами (112 битов) или тройного DES с тремя ключами (биты 16).

Слабые ключи. Четыре ключа из 2^{56} возможных ключей называются **слабыми ключами.** Слабые ключи — это одни из тех, которые после операции удаления проверочных бит (используя таблицу 6.12) состоят из всех нулей или всех единиц или половины нулей и половины единиц. Такие ключи показаны в таблице 6.18.

Таблица 6.18. Слабые ключи

Ключи до удаления проверочных бит (64 бита)	Действующие ключи (56 бит)
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 1F1F 1F1F	0000000 FFFFFFF
E0E0 E0E0 E0E0 E0E0	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

Ключи раунда, созданные от любого из этих слабых ключей, — те же самые и имеют тот же самый тип, что и ключ шифра. Например, эти шестнадцать ключей раунда создают первый ключ, который состоит из всех нулей или всех единиц или наполовину из нулей и единиц.

Это происходит по той причине, что алгоритм генерирования ключей сначала делит ключ шифра на две половины. Смещение или перестановка блока не изменяют блок, если он состоит из всех нулей, или всех единиц, или наполовину из нулей и единиц.

В чем опасность использования слабых ключей? Если мы зашифровали блок слабым ключом и впоследствии расшифровали результат тем же самым слабым ключом, мы получаем первоначальный блок. Процесс создает один и тот же первоначальный блок, если мы расшифровываем блок дважды. Другими словами, каждый слабый ключ есть инверсия самого себя: E_k. (E_k (P)) = P, как это показано на рис. 6.11.

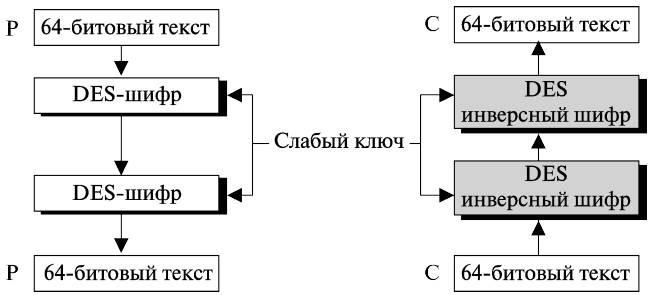


Рис. 6.11. Двойное шифрование и дешифрование со слабым ключом

Слабых ключей надо избегать, потому что противник может легко распознать их на перехваченном шифре. Если после двух этапов дешифрации результат тот же самый, противник определяет, что он нашел ключ.

Пример 6.8

Давайте попробуем применить первый слабый ключ в таблице 6.18, чтобы два раза зашифровать блок. После того как проведено два шифрования с тем же самым ключом, в результате получим исходный текст. Обратите внимание, что мы ни разу не использовали алгоритм дешифрования, а только провели два раза шифрование.

Ключ: 0x0101 0101 0101 0101 Зашифрованный текст: 0x814FE938589154F7
Исходный текст: *0xl23456887654321*

Ключ: 0x0101 0101 0101 0101
Исходный текст: 0x814FE938589154F7 Зашифрованный текст: 0*xl234.56887654321*

Полуслабые ключи. Имеются шесть ключевых пар, которые названы **полуслабыми ключами.** Этим шесть пар показаны в Таблице 6.19 (формат на 64 бита перед удалением проверочных бит). Полуслабые ключи создают только два различных ключа раунда и затем повторяют их восемь раз. Кроме того, ключи раунда, созданные от каждой пары, — одни и те же в различном порядке.

Таблица 6.19. Полуслабые ключи

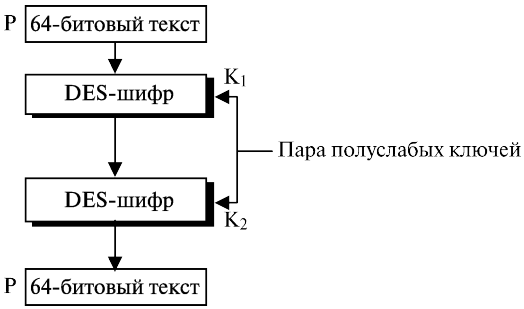
Первый ключ в паре	**Второй ключ в паре**
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FEO 1FEO OEF1 OEF1	E01F E01F F10E F10E
01EO 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE OEFE OEFE	FE1F FE1F FEOE FEOE
011F 011F 010E 010E	1F01 1F01 OE01 OE01
EOFE EOFE FIFE FIFE	FEEO FEEO FEF1 FEFl

Чтобы проиллюстрировать идею, мы создали ключи раунда от первой пары, как показано ниже:

Ключ раунда 1	9153E54319BD	6EAC1ABCE642
Ключ раунда 2	6EAC1ABCE642	9153E54319BD
Ключ раунда 3	6EAC1ABCE642	9153E54319BD
Ключ раунда 4	6EAC1ABCE642	9153E54319BD
Ключ раунда 5	6EAC1ABCE642	9153E54319BD
Ключ раунда 6	6EAC1ABCE642	9153E54319BD
Ключ раунда 7	6EAC1ABCE642	9153E54319BD
Ключ раунда 8	6EAC1ABCE642	9153E54319BD
Ключ раунда 9	9153E54319BD	6EAC1ABCE642
Ключ раунда10	9153E54319BD	6EAC1ABCE642
Ключ раунда 11	9153E54319BD	6EAC1ABCE642
Ключ раунда 12	9153E54319BD	6EAC1ABCE642
Ключ раунда 13	9153E54319BD	6EAC1ABCE642
Ключ раунда 14	9153E54319BD	6EAC1ABCE642
Ключ раунда 15	9153E54319BD	6EAC1ABCE642

Ключ раунда 16 6EAC1ABCE642 9153E54319BD

Как показывает список, имеется восемь одинаковых ключей раунда в каждом полуслабом ключе. Кроме того, ключи раунда 1 в первом множестве — те же, что и ключи раунда 16 во втором; ключи раунда 2 в первом — те же самые, что и



ключи раунда 15 во втором, и так далее. Это означает, что ключи инверсны друг другу: Ek_2. (Ek_1E (P)) = P, как показано на рис. 6.12.

Рис. 6.12. Пара полуслабых ключей при шифровании и дешифровании

Возможно слабые ключи. Также имеется 48 ключей, которые называются **возможно слабыми ключами.** Возможно слабый ключ создает только четыре различных ключа раунда; другими словами, шестнадцать ключей раундов разделены на четыре группы, и каждая группа состоит из четырех одинаковых ключей раунда.

Пример 6.9

Какова вероятность случайного выбора слабого, полуслабого или возможно слабого ключа?

Решение

Множество ключей DES равно 2^{56}. Общее количество вышеупомянутых ключей — 64 (4 + 12 + 48). Вероятность выбора одного из этих ключей равна $8{,}8 \times 10^{-16}$, т.е. исключительно мала.

Ключевое дополнение. Среди множества ключей (2^{56}) некоторые ключи могут быть получены инверсией (изменение из 0 в 1 или 1 в 0) каждого бита в ключе. Ключевое дополнение упрощает процесс криптоанализа. Ева может использовать только половину возможных ключей (2^{55}), чтобы выполнить атаку грубой силы, потому что

$$C = E(K, P) \rightarrow \overline{C} = E(\overline{K}, \overline{P})$$

Другими словами, если мы зашифровали дополнение исходного текста дополнением ключа, мы получаем дополнение зашифрованного текста. Ева не

должна проверять все 2^{56} возможных ключей, она может проверить только половину из них и затем дополнить результат.

Пример 6.10

Давайте проверим эти сведения о ключах дополнения. Мы используем произвольный ключ и исходный текст, для того чтобы найти соответствующий зашифрованный текст. Если мы имеем ключевое дополнение и исходный текст, то получим дополнение предыдущего зашифрованного текста (таблица 6.20).

Таблица 6.20. Результаты примера 6.10

	Оригинал	Дополнение
Ключ	1234123412341234	EDCBEDCBEDCBEDCB
Исходный текст	12345678ABCDEF12	EDCBA98754321.0ED
Зашифрованный текст	E112BE1DEFC7A367	1EED41E210385C98

Кластерный ключ. Кластерный ключ рассматривает ситуации, в которых два или более различных ключа создают один и тот же зашифрованный текст из одного и того же исходного текста. Очевидно, каждая пара полуслабых ключей — ключевой кластер. Однако больше кластеров не было найдено. Будущие исследования, возможно, могут открыть некоторые другие.

6.4. Многократное применение DES

Как мы уже видели, основная критика DES направлена на длину ключа. Возможные технологии и возможности параллельных процессоров делают реальной атаку грубой силы. Одно из решений для улучшения безопасности — это отказ от DES и разработка нового шифра. Это решение мы рассмотрим в лекции 7 при применении AES. Второе решение — многократное (каскадное) применение множества ключей. Это решение, которое использовалось некоторое время, не требует инвестиций в новое программное обеспечение и аппаратные средства. Ниже рассмотрен такой подход.

Как мы узнали в лекции 5, подстановка, которая размещает все возможные входы во все возможные выходы, является группой с отображениями элементов множества и набором операций. В этом случае применение двух последовательных отображений бесполезно, потому что мы можем всегда найти третье отображение, которое эквивалентно композиции этих двух (свойство замкнутости). Это означает, что если DES — группа, то однократный DES с ключом k_3 делает то же самое (рисунок 6.13).

К счастью DES — не группа. Она базируется на следующих двух параметрах.

а. Номер возможных входов или выходов в DES — $N = 2^{64}$. Это означает, что $N! = (2^{64})! = 10^{347\ 380\ 000\ 000\ 000\ 000\ 000}$ отображений. Один из способов способ сделать DES группой — это поддержать все эти отображения с размером ключа $\log_2 (2^{64}!) — 2^{70}$ битов. Но мы знаем, что длина ключа в DES — 56 бит (только маленькая часть этого требуемого огромного ключа).

б. Другой способ сделать DES группой – сделать, чтобы множество отображений было подмножеством множества в смысле зависимости от первого

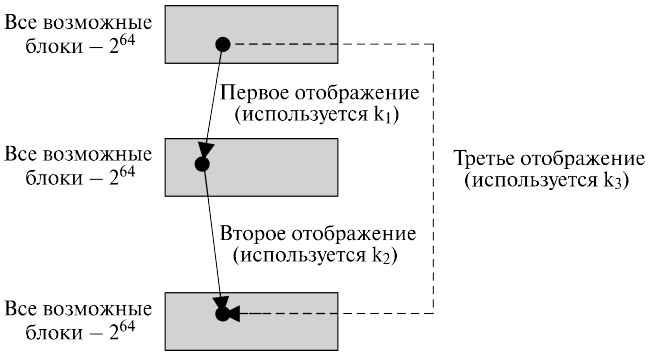


Рис. 6.13. Композиция отображений

параметра; но было доказано, что группы, созданные из группы с помощью первого параметра, имеют ключевой размер 56 битов.

Если DES не является группой, то очень маловероятно, что мы можем найти ключ k_3, такой, что

$$E_{k2}(E_{k1}(P)) = E_{k3}(P)$$

Это означает, что мы можем применить двукратные или трехкратные DES, чтобы увеличить размер ключа.

Двукратный DES

Первый подход состоит в том, чтобы использовать **двукратный DES (2DES).** При этом подходе мы применяем два типа шифров DES для шифрования и два типа обратных шифров для дешифрования. Каждый тип использует различный ключ, что означает, что размер ключа теперь удвоился (112 битов). Однако двукратный DES уязвим к атаке знания открытого текста, как это обсуждается в следующем разделе.

На первый взгляд двукратные DES увеличивают число испытаний при поиске ключа от 2^{56} (в однократном DES) к 2^{112} (в двукратном DES). Однако при использовании атаки знания исходного текста, называемой **атакой сведения к середине**, можно доказать, что двукратный DES улучшает эту устойчивость (до 2^{57} по испытаниям), но не чрезвычайно (к 2^{112}). Рисунок 6.14 показывает диаграмму для двукратного DES. Алиса использует два ключа, k_1 и k_2, чтобы зашифровывать исходный текст P в зашифрованный текст C; Боб использует зашифрованный текст C и два ключа, k_2 и k_1, для восстановления P.

В средней точке M — текст, созданный первым шифрованием или первым дешифрованием. Для обеспечения правильной работы он должен быть одинаковым для шифрования и дешифрования. Другими словами, мы имеем два отношения:

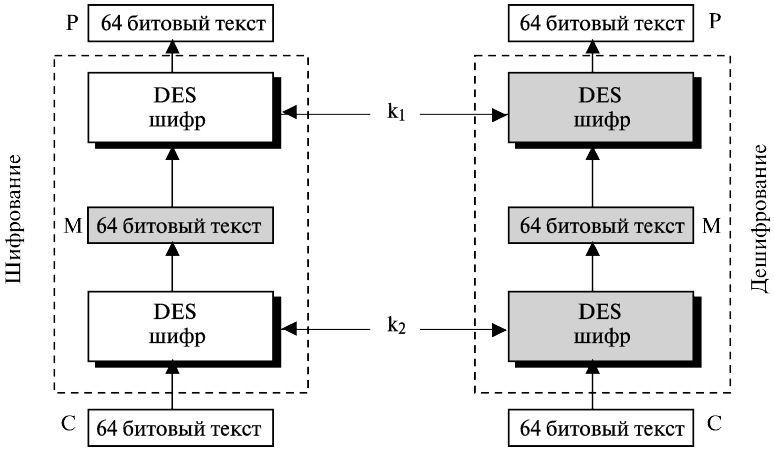


Рис. 6.14. Атака сведения к середине в двукратном DES

$$M = E_{K1}(P) \text{ и } M = E_{K2}(C)$$

Предположим, что Ева перехватила предыдущую пару P и C (атака знания исходного текста). Базируясь на первом отношении из упомянутых выше, Ева зашифровывает P, используя все возможные значения (2^{56}) k_1, и записывает все значения, полученные для M. Базируясь на вторых отношениях, упомянутых выше, Ева расшифровывает C, используя все возможные значения (2^{56}) k_2. Она записывает все значения, полученные для M. Далее Ева создает две таблицы, отсортированные согласно значениям M. Она сравнивает значения для M, пока не находит те пары k_1 и k_2, для которых значение M является одним и тем же в обеих таблицах (как показано на рис. 6.15). Обратите внимание, что должна быть по крайней мере одна пара, потому что она делает исчерпывающий поиск комбинации двух ключей.

1. Если есть только одно соответствие. Ева нашла два ключа (k_1 и k_2). Если есть больше чем один кандидат, Ева перемещается в следующий шаг.
2. Она берет другую перехваченную пару зашифрованного текста и исходного текста и использует каждого кандидата для получения пары ключей, чтобы установить, может ли она получить зашифрованный текст из исходного текста. Если она находит больше чем одного кандидата в виде пары ключей, она повторяет шаг 2, пока, наконец, не находит уникальную пару.

Было доказано, что после применения второго шага к нескольким перехваченным парам «зашифрованный текст — исходный текст» ключи были найдены. Это означает, что вместо того чтобы использовать поиск ключей с помощью 2^{112} испытаний, Ева проводит 2^{56} испытаний поиска ключа и проверяет два раза (несколько больше испытаний требуется, если найден на первом шаге единственный кандидат). Другими словами, двигаясь от однократного DES до двукратного DES,

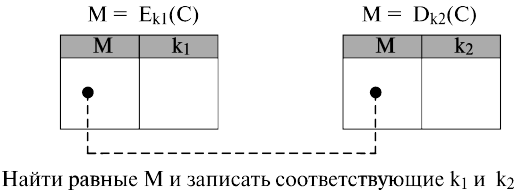


Рис. 6.15 Таблицы для атаки «сведения к середине»

мы увеличили объем испытаний от 2^{56} до 2^{57} (а не до 2^{112}, как это кажется при поверхностном подходе).

Трехкратный DES

Для того чтобы улучшить безопасность DES, был предложен **трехкратный DES (3DES).** Он применяет три каскада DES для шифрования и дешифрования. Сегодня используются две версии трехкратных DES: трехкратный DES с двумя ключами и трехкратный DES с тремя ключами.

Трехкратный DES с двумя ключами

В трехкратном DES с двумя ключами есть только два ключа: k_1 и k_2. Первый и третий каскады используют k_1; второй каскад использует k_2. Чтобы сделать трехкратный DES совместимым с DES, средний каскад применяет дешифрование (обратный шифр) на стороне шифрования и шифрование (шифр) на стороне де-

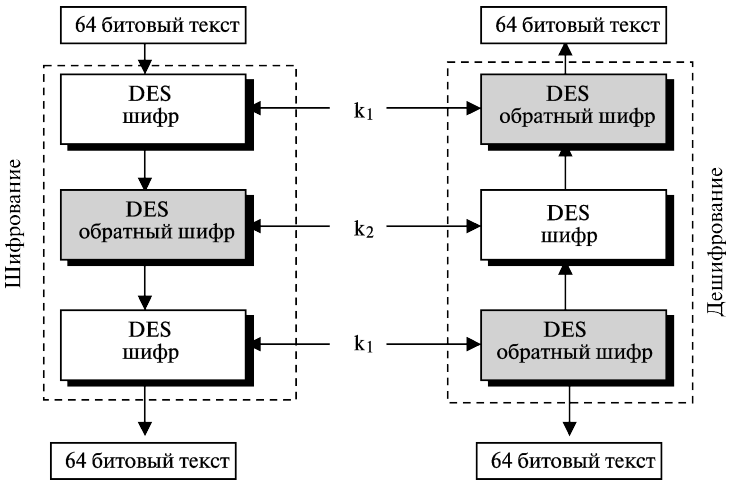


Рис. 6.16. Трехкратный DES с двумя ключами

шифрования. Таким способом сообщение, зашифрованное DES-ключом k, может быть расшифровано трехкратным DES, если $k_1 = k_2 = k$. Хотя трехкратный DES с двумя ключами также уязвим при атаке «знания исходного текста», он гораздо устойчивее, чем двукратный DES. Он был принят для банков. Рисунок 6.16 показывает трехкратный DES с двумя ключами.

Трехкратный DES с тремя ключами

Возможность атак «знания исходного текста» при трехкратном DES с двумя ключами *заставила* некоторые приложения использовать **трехкратный DES с тремя ключами.** Алгоритм может применять три каскада шифра DES на стороне шифрования и три каскада обратных шифров на стороне дешифрования. Для совместимости с однократным DES сторона шифрования использует EDE, а сторона дешифрования — DED. E (encryption) — каскад шифрования, D (decryption) — каскад дешифрования. Совместимость с однократным DES обеспечивается при $k_1 = k$ и установкой k_2 и k_3 к одному и тому же произвольному ключу, выбранному приемником. Трехкратный DES с тремя ключами используется многими приложениями, такими, как PGP (Pretty Good Privacy), поскольку гарантирует очень хорошую конфиденциальность (см. лекцию 16).

6.5. Безопасность DES

DES, как первый блочный шифр, имеющий важное значение, прошел через много испытаний на безопасность. Среди предпринятых атак лишь три представляют интерес: грубая сила, дифференциальный криптоанализ и линейный криптоанализ.

Атака грубой силы

Мы уже обсуждали слабость шифра с коротким ключом. Слабость ключа совместно с другими рассмотренными недостатками приводит к тому, что DES может быть взломан с числом испытаний 2^{55}. Однако сегодня большинство приложений использует либо 3DES с двумя ключами (размер ключа 2^{112}), либо 3DES с тремя ключами (размер ключа 2^{168}). Эти две многократные версии DES позволяют ему показывать существенную стойкость к атакам грубой силы.

Дифференциальный криптоанализ

Мы в лекции 5 уже обсуждали методику дифференциального криптоанализа для современных блочных шифров. DES не является устойчивым к такому виду атаки. Однако многое указывает, что разработчики DES уже знали о такой опасности и проектировали S-блоки и специально выбрали число раундов, чтобы сделать DES стойким к этому типу атаки. Сегодня показано, что DES может быть взломан, используя дифференциальный криптоанализ, если мы имеем 2^{47} выборок исходного текста или 2^{55} известных исходных текстов. Хотя это выглядит более эффективно, чем в атаке грубой силы, предположить, что кто-то знает 2^{47} выборок исходного текста или 2^{55} выборок исходного текста, практически невозможно. Поэтому мы можем сказать, что DES является стойким к дифференциальному крип-

тоанализу. Также показано, что увеличение числа раундов до 20 увеличивает число требуемых выборок исходного текста для атаки более чем до 2^{64}. Такое увеличение невозможно, потому что число блоков исходного текста в DES только 2^{64}.

Мы покажем пример дифференциального криптоанализа в приложении N.

Линейный криптоанализ

Мы обсуждали методику линейного криптоанализа для современных блочных шифров в лекции 5. Линейный криптоанализ — более новая методика, чем дифференциальный криптоанализ. DES более уязвим к применению линейного криптоанализа, чем к дифференциальному криптоанализу, — вероятно, потому, что этот тип атак не был известен проектировщикам DES и S-блоки не являются очень стойкими к линейному криптоанализу. Показано, что DES может быть взломан с использованием 2^{43} пары известных исходных текстов. Однако с практической точки зрения перехват такого количества пар очень маловероятен.

Мы покажем пример линейного криптоанализа DES в приложении N.

6.6. Рекомендованная литература

Нижеследующие книги и сайты обеспечивают более детальную информацию о предметах, которые мы обсуждали в этой лекции.

Книги

[Sta06], [Sti06J, [Rhe03], [Sal03J, [Mao04J] и [TW06] — это книги, которые рассматривают DES.

Сайты

Нижеследующие сайты содержат боле подробную информацию о темах, обсужденных в этой лекции,

http://www.itl.nist.gov/fipspubs/np46-2.htm
www.nist.gov/director/prog-ofc/report01-2.pdf
www.engr.mun.ca/-how ard/PAPERS/ldc_tutorial.ps
islab.oregonstate.edu/koc/ece575/notes/dc 1.pdf
homes.esat.kuleuven.be / ~ abiryuko/Cryptan/matsui_des
http://nsfsecurity.pr.erau.edu/crypto/lincrypt.html

6.7. Итоги

- Стандарт шифрования данных (DES) — блочный шифр с симметричными ключами, изданный национальным Институтом Стандартов и Технологии (NIST) как FIPS 46 в Федеральном Регистре.
- На стороне шифрования DES принимает исходный текст на 64 бита и создает зашифрованный текст на 64 бита. На стороне дешифрования DES принимает зашифрованный текст на 64 бита и создает блок на 64 бита исходного текста. Ключ шифра на 56 битов одного типа используется и для шифрования, и для дешифрования.
- Процесс шифрования состоит из двух перестановок (P-блоки), которые называются начальными и конечными перестановками, и шестнадцати раундов Файстеля. Каждый раунд DES — шифр Файстеля с двумя элементами (смеситель и устройство замены). Каждый из этих элементов является обратимым.
- Основой DES является функция DES. Функция DES применяет ключ на 48 битов к самым правым 32 битам, чтобы получить на выходе 32 бита. Эта функция составлена из четырех операций: перестановки расширения, отбеливателя (который добавляет ключ), группы S-блоков и прямой перестановки.
- Генератор ключей раунда создает из ключа шифра на 56 битов шестнадцать ключей по 48 битов. Однако ключ шифра обычно представляется как ключ на 64 бита, в котором 8 дополнительных битов являются проверочными битами — они отбрасываются перед фактическим процессом генерирования ключей.
- DES показывает хорошие рабочие характеристики по отношению к эффектам полноты (законченности) и лавины. К числу слабостей DES относятся: построение шифра (S-блоки и P-блоки) и ключ шифра (длина), слабые ключи, полуслабые ключи, возможно слабые ключи и дополнение ключей.
- Так как DES — не группа, одно из решений по улучшению безопасности DES состоит в том, чтобы пользоваться многократными DES, которые используют кратное число применения ключей (двукратный или трехкратный DES). Двукратный DES уязвим к атаке сведения к середине, поэтому в обычных приложениях используется трехкратный DES с двумя ключами или с тремя ключами.
- Разработка S-блоков и число раундов сделали DES почти обладающим иммунитетом от дифференциального криптоанализа. Однако DES уязвим при применении линейного криптоанализа, если злоумышленники смогут собрать достаточно много исходных текстов.

6.8. Набор для практики

Обзорные вопросы

1. Каков размер блока в DES? Каков размер ключа шифра в DES? Каков размер ключей раунда в DES?
2. Каково число раундов в DES?
3. Сколько смесителей и устройств замены используется в первом способе шифрования и обратного дешифрования? Сколько их используется при втором способе?
4. Сколько перестановок используется в алгоритме шифра DES?
5. Сколько операций ИСКЛЮЧАЮЩЕЕ ИЛИ используется в DES-шифре?
6. Почему DES-функции необходима расширяющая перестановка?
7. Почему генератор ключей раунда нуждается в удалении проверочных бит?
8. Какова разность между слабым ключом, полуслабым ключом и возможно слабым ключом?
9. Что такое двукратный DES? Какая атака делает двукратный DES бесполезным?
10. Что такое трехкратный DES? Что такое трехкратный DES с двумя ключами? Что такое трехкратный DES с тремя ключами?

Упражнения

1. Ответьте на следующие вопросы об S-блоках в DES:
   а. покажите результат прохождения 110111 через S-блок 3.
   б. покажите результат прохождения 001100 через S-блок 4.
   в. покажите результат прохождения 000000 через S-блок 7.
   г. покажите результат прохождения 111111 через S-блок 2.
2. Нарисуйте таблицу, которая показывает результат прохождения 000000 через все 8 S-блоков. Рассмотрите результат на выходе.
3. Нарисуйте таблицу, которая показывает результат прохождения 111111 через все 8 S-блоков. Рассмотрите результат на выходе.
4. Проверьте третий критерий для S-блока 3, используя следующие пары входов:
   a. 000000 и 000001
   b. 111111 и 111011
5. Проверьте четвертый критерий построения S-блока 2, используя следующие пары входа:
   a. 001100 и 110000
   b. 110011 и 001 111
6. Проверьте пятый критерий построения S-блока 4, используя следующие пары входов:
   a. 001100 и 110000
   b. 110011 и 001 111

7. Создайте 32 6-битовые входные пары, чтобы проверить шестой критерий построения S-блока 5.
8. Покажите, как выполнены восемь критериев построения S-блока 7.
9. Докажите первый критерий построения P-блоков, проверив входы к S-блоку 2 раунда 2.
10. Докажите второй критерий построения для P-блоков, проверяя входы к S-блоку раунда 4.
11. Докажите третий критерий построения P-блоков, проверяя выход S-блока раунда 3.
12. Докажите четвертый критерий построения P-блоков, проверяя выход S-блока 6 раунда 12.
13. Докажите пятый критерий проекта для P-блоков, проверяя отношение между S-блоками 3, 4 и 5 в раундах 10 и 11.
14. Докажите шестой критерий построения P-блоков, проверяя отношение S-блока конечного пункта произвольного блока.
15. Докажите седьмой критерий проекта для P-блоков, проверяя отношения между S-блоком 5 в раунде 4 и S-блоком 7 в раунде 5.

16. Измените рисунок 6.9, используя альтернативный подход.
17. Докажите, что обратный шифр на рис. 6.9 — фактически инверсия шифра DES с тремя раундами. Стартуя с исходного текста и начального шифра, докажите, что вы можете получить тот же самый исходный текст в конце процесса применения обратного шифра.
18. Тщательно изучите ключ при перестановке сжатия таблицы 6.14.
    a. Какие входные порты отсутствуют на выходе?
    b. Все ли левые 24 бита выхода идут от всех левых 28 входных бит?
    c. Все ли правые 24 бита выхода идут от всех 28 входных битов?
19. Покажите результат следующих шестнадцатеричных данных
    0110 10234110 1023
    после прохождения их через начальный блок перестановки.
20. Покажите результат следующих шестнадцатеричных данных
    AAAA BBBB CCCC DDDD
    после прохождения их через конечный блок перестановки.
21. Если ключ с проверочными битами (64 бита) — 0123 ABCD 2562 1456, найдите ключ первого раунда.
22. Используя блок исходного текста всех нулей и ключ на 56 битов из всех нулей, докажите слабость ключевого дополнения, предполагающую, что DES состоит только из одного раунда.
23. Вы можете изобрести атаку «сведение к середине» для трехкратного DES?
24. Напишите псевдокод для *переставляющей* процедуры, используемой в алгоритме 6.1:

permute (n,m, inBlock [n], outBlock [m], *permutationTable [m])*

25. Напишите псевдокод для процедуры *разбиения*, используемой в алгоритме 6.1:

Лекция 7. Усовершенствованный стандарт шифрования (AES — Advanced encryption standard)

Цели и содержание

В этой лекции мы обсуждаем Усовершенствованный стандарт шифрования (AES — ADVANCED ENCRYPTION STANDARD) — современный блочный шифр с симметричными ключами, который может заменить DES. Эта лекция имеет несколько целей:

- рассмотреть короткую историю AES;
- определить основную структуру AES;
- определить преобразования, используемые AES;
- определить процесс расширения состава ключей;
- обсудить различные реализации.

Особое внимание уделяется алгебраическим структурам, которые мы обсуждали в лекции 4, — они обеспечивают безопасность AES.

7.1. Введение

Усовершенствованный стандарт шифрования (ADVANCED ENCRYPTION STANDARD) — стандарт на блочный шифр с симметричными ключами, изданный Национальным Институтом Стандартов и Технологии (NIST) в декабре 2001 года.

История

В 1997 году NIST начал искать замену для DES, которая была названа *Усовершенствованным стандартом шифрования (ADVANCED ENCRYPTION STANDARD или AES).* В NIST-спецификациях были заложены требования размера блока из 128 битов и трех различных размеров ключей: 128, 192 и 256 битов. Спецификации также требовали, чтобы AES был открытым алгоритмом, публично доступным во всем мире.

Спецификации стандартов были объявлены во многих странах, чтобы запросить ответы у специалистов и заинтересованных лиц со всех континентов.

После *Первой Конференции по выбору Кандидатов AES* NIST объявила, что 15 из 21 полученных алгоритмов отвечают поставленным требованиям и выбраны как первые кандидаты (август 1998 г.). Алгоритмы были представлены от многих стран; разнообразие этих предложений демонстрировало открытость процесса и участие всего мира.

После Второй Конференции по выбору Кандидата AES, которая была проведена в Риме, NIST объявила 5 из 15 из кандидатов. Алгоритмы MARS, RC6, Rijndael, Serpent и Twofish были выбраны как финалисты (август 1999).

После *Третей Конференции по выбору Кандидата AES* NIST объявила, что выбран алгоритм *Усовершенствованного Стандарта Шифрования — Rijndael,* спроектированный бельгийскими исследователями Джоном Даеменом и Винсентом Риджменом (октябрь 2000 г.).

В феврале 2001 г. NIST объявил, что эскиз **Федерального Стандарта обработки Информации (FIPS)** доступен для общественного рассмотрения и комментариев.

Наконец, AES был издан как FIPS 197 в *Федеральном Регистре* в декабре 2001 г.

Критерии

Критерии, определенные NIST для выбора AES, относятся к трем областям: безопасность, стоимость и реализация. В конце концов *Rijndael* был оценен в совокупности как лучший, отвечающий этим критериям.

Безопасность

Особое внимание уделялось безопасности. Поскольку NIST ясно потребовал 128-битовый ключ, этот критерий определял то, что внимание обращалось на устойчивость шифра к другим атакам криптоанализа, нежели атака грубой силы.

Стоимость

Вторым критерием была стоимость, которая задает требуемую вычислительную эффективность и требования для различных реализаций, таких, как аппаратные средства, программное обеспечение или интеллектуальные карты доступа.

Реализация

Этот критерий включал требование, что алгоритм должен иметь гибкость (возможность быть реализованным на любой платформе) и простоту.

Раунды

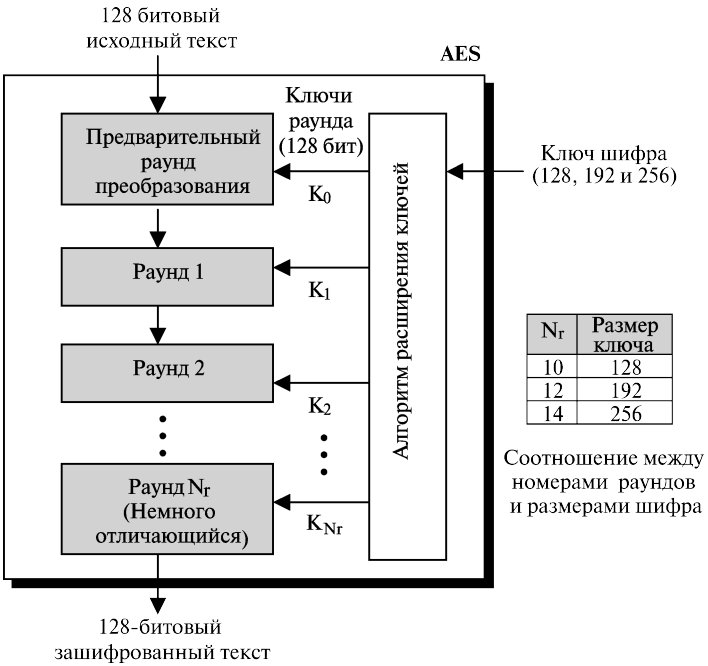
AES — шифр не-Файстеля, который зашифровывает и расшифровывает блок данных 128 битов, используя 10, 12 или 14 раундов. Размер ключа может быть 128, 192 или 256 битов и зависит от числа раундов. Рисунок 7.1 показывает общую схему: алгоритм шифрования (называемого шифром); алгоритм дешифрования (называемый обратным шифром), для которого применяются те же ключи, но в обратном порядке.

На рис. 7.1 N_r определяет число раундов. Рисунок также показывает отношение между числом раундов и размером ключа — это означает, что мы имеем три различных версии AES; они обозначаются как AES-128, AES-192 и AES-256. Однако ключи раунда, которые созданы алгоритмом расширения ключей всегда 128 бит, имеют тот же самый размер, что и блоки зашифрованного или исходного текста.

AES определил три версии, с 10, 12 и 14 раундами. Каждая версия использует различный размер ключа шифра (128, 192 или 256), но ключ раунда — всегда 128 бит.

Число ключей раунда, сгенерированных алгоритмом расширения ключей, всегда на один больше, чем число раундов. Другими словами, мы имеем

$$\text{число ключей раунда} = N_r + 1$$



Nr	Размер ключа
10	128
12	192
14	256

Соотношение между номерами раундов и размерами шифра

Рис. 7.1. Общее построение шифрования AES-шифра

Мы обозначаем ключи раунда как K_0, K_1, K_2..., K_N.

Единицы данных

AES использует пять единиц для представления данных: биты, байты, слова, блоки и массивы состояний. Бит — наименьшая и элементарная единица; другие единицы могут быть выражены в терминах меньших единиц Рисунок 7.2 показывает единицы неэлементарных данных: байт, слово, блок, массив состояний (state).

Бит

В AES **бит** — двоичная цифра со значением 0 или 1. Мы используем строчные буквы для обозначения бит.

Байт

Байт — группа из восьми битов, которая может быть обработана как единый объект: матрица из одной строки (1 x 8) восьми битов или столбец матрицы (8x1) из восьми битов. Когда информация байта обрабатываются как матрица строки, то биты вставляются в матрице слева направо. Когда байт обрабатывается как матрица столбца, биты вставляются в матрице сверху вниз. Мы будем использовать строчную «жирную» букву (Bold) для обозначения байта.

Слово

Слово — группа из 32 битов, которая может быть обработана как единый объект. Это матрица из строки в четыре байта или столбец матрицы из четырех байтов. Когда слово обрабатывается как матрица-строка, байты вставляются слева направо. Когда слово представляется матрицей-колонкой, байты вставляются сверху вниз. Мы будем использовать строчную «жирную» букву **W** для обозначения слова.

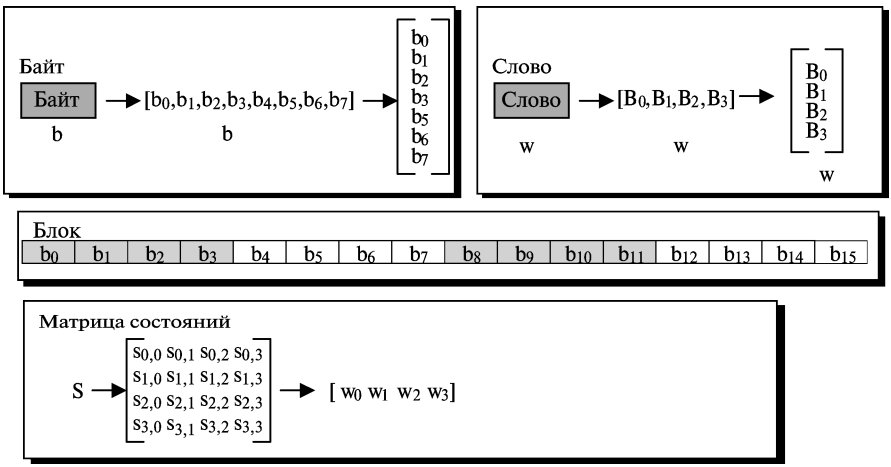


Рис.7.2. Единицы данных, используемых в AES

Блок

AES зашифровывает и расшифровывает блоки данных. **Блок** в AES — группа 128 битов. Однако блок может быть представлен как матрица-строка из 16-ти байтов.

Матрица состояний

AES использует несколько раундов, каждый раунд состоит из несколько каскадов. Блок данных преобразовывается от одного каскада к другому. В начале и в конце шифра AES применяется термин *блок данных;* до и после каждого каскада блок данных называется **матрицей состояний.** Мы используем «жирную» заглавную букву, чтобы обозначить эту матрицу. Хотя матрица состояний на различных каскадах обычно обозначается **S**, мы иногда применяем букву **T**, чтобы обозначить временную матрицу состояний. Матрицы состояний, подобно блокам, состоят из 16 байтов, но обычно обрабатываются как матрицы 4 × 4 байтов. В этом случае каждый элемент матрицы состояний обозначается как $S_{r,c}$, где *r* (от 0 до 3) определяет строку и *c* (от 0 до 3) определяет столбец. Иногда матрица состояний обрабатывается как матрица-строка слов (1 × 4). Это имеет смысл, если мы представляем слово как матрицу-столбец. В начале шифра байты в блоке данных вставляются в матрицу состояний столбец за столбцом, в каждом столбце — свер-

ху вниз. В конце шифра байты в матрице состояний извлекаются, как это показано на рис. 7.3.

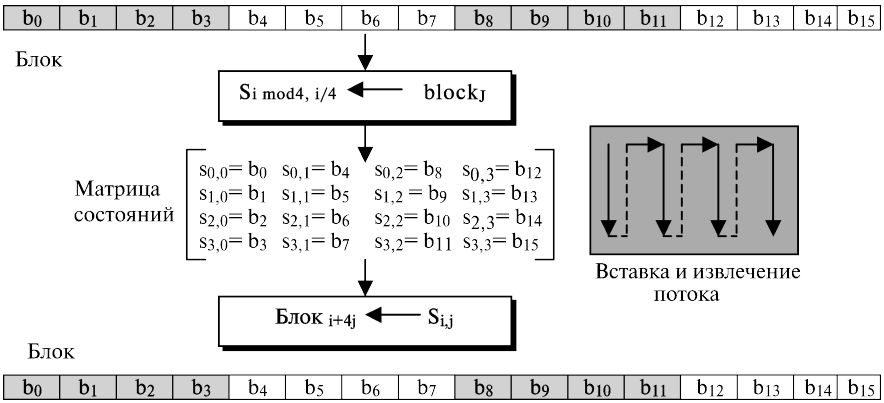


Рис. 7.3. Преобразование блок – матрица состояний и матрица состояний — в блок

Пример 7.1

Рассмотрим, как можно изобразить блок с 16 символами в виде матрицы 4 × 4. Предположим, что текстовый блок — «AES uses a matrix». Добавим два фиктивных символа в конце и получим «AESUSESAMATRIXZZ». Теперь мы заменим каждый символ целым числом между 00 и 25. Представим каждый байт как целое число с двумя шестнадцатеричными цифрами. Например, символ «S» сначала поменяем на 18, а затем запишем в шестнадцатеричном изображении как 12. Матрица состояний тогда заполняется столбец за столбцом, как это показано на рис. 7.4.

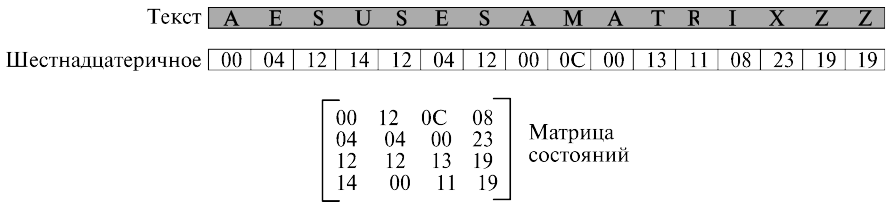


Рис. 7.4. Переход шифрованного текста в матрицу состояний

Структура каждого раунда

Рисунок 7.5 показывает структуру каждого раунда на стороне шифрования. Каждый раунд, кроме последнего, использует четыре преобразования, которые являются обратимыми. Последний раунд имеет только три преобразования.

Как показывает рис. 7.5, каждое преобразование принимает матрицу состояний и создает другую матрицу состояний, которая применяется для следующего преобразования или следующего раунда. Секция, предваряющая раунд, использует только одно преобразование (AddRoundKey); последний раунд использует только три преобразования (MixColumns — преобразование отсутствует).

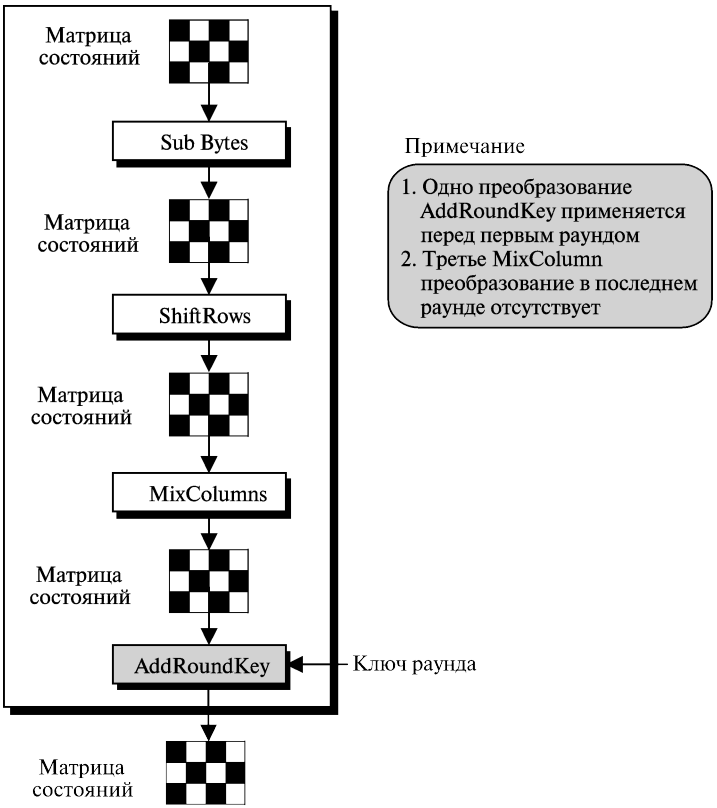


Рис. 7.5. Структура каждого раунда на стороне шифрования

7.2. Преобразования

Чтобы обеспечить безопасность, AES использует четыре типа преобразований: подстановка, перестановка, смешивание и добавление ключа. Ниже мы обсудим каждое.

Подстановка

AES подобно DES применяет подстановку. Однако этот механизм имеет различия. Первое: подстановка делается для каждого байта. Второе: для преобразова-

ния каждого байта используется только одна таблица — это означает, что если два байта одинаковы, то и результат преобразования одинаков. Третье: преобразование определяется или процессом поиска в таблице, или математическим вычислением в GF(2^8) поле. AES работает с двумя обратимыми преобразованиями.

SubBytes

Первое преобразование, **SubBytes**, применяется на стороне шифрования. Чтобы применить подстановку к байту, мы интерпретируем байт как две шестнадцатеричные цифры. Левая цифра определяет строку, а правая — колонку в таблице перестановки. На пересечении строки и колонки, обозначенных этими шестнадцатеричными цифрами, находится новый байт. Рисунок 7.6 иллюстрирует идею.

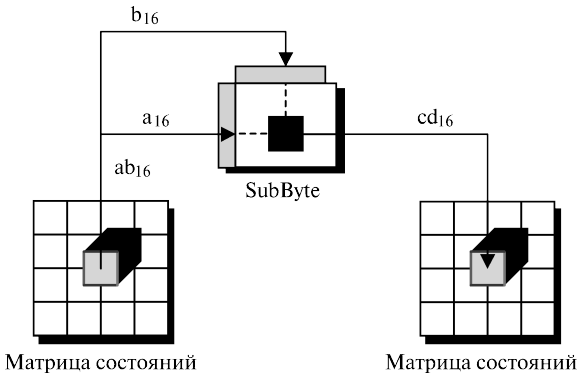


Рис. 7.6. Преобразование SubByte

В преобразовании SubBytes состояние обрабатывается как матрица байтов 4 × 4. В один момент проводится преобразование одного байта. Содержание каждого байта изменяется, но расположение байтов в матрице остается тем же самым. В процессе преобразования каждый байт преобразуется независимо от других — это шестнадцать личных преобразований байт в байт.

Операция SubByte включает 16 независимых преобразований байта в байт.

Таблица 7.1 показывает таблицу подстановки (S-блок) для преобразования SubBytes. Преобразование обеспечивает эффект перемешивания. Например, два байта, $5A_{16}$ и $5B_{16}$, которые отличаются только одним битом (самый правый бит), преобразованы в BE_{16} и 39_{16}, которые отличаются четырьмя битами.

InvSubBytes

InvSubBytes — инверсия **SubBytes**. Преобразование сделано с использованием таблицы 7.2, и мы можем легко проверить, что эти два преобразования являются обратными друг другу.

Таблица 7.1. Таблица преобразования SubBytes

	0	*I*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	FO	AD	D4	A2	AF	9C	A4	72	CO
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	1 8	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	AO	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	DO	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DE
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	IF	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	OE	61	35	57	B9	86	C1	1D	9E:
E	El	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	Al	89	OD	BF	E6	42	68	41	99	2D	OF	BO	54	BB	16

Таблица 7.2. Таблица преобразования InvSubBytes

	0	*I*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	52	099	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00C	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DE	6E
A	47	E1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7E	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF:
E	A0	E0	3B	4D	AE	2A	F5	B0	CB	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Пример 7.2

Рисунок 7.7 показывает, как матрица состояний преобразуется с использованием SubBytes. Рисунок также показывает, что InvSubBytes однозначно воспроиз-

водит оригинал. Заметим, что если два байта имеют одинаковое значение, то они преобразуются одинаково. Например, два байта 04_{16} и 04_{16} в левой матрице состояний преобразуются в $F2_{16}$ и $F2_{16}$ в правой матрице состояний и наоборот. Причина в том, что каждый байт использует одну и ту же таблицу преобразований.

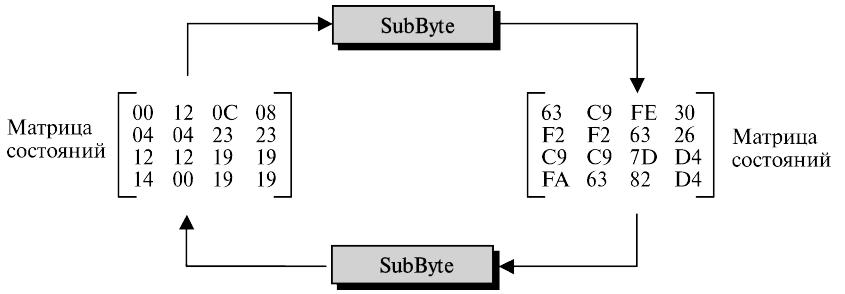


Рис. 7.7. Преобразование SubByte по примеру 7.2

Преобразование с использованием поля GF(2^8)

Хотя мы можем использовать таблицы 7.1 и.7.2 для перестановки каждого байта, AES дает определение алгебраическим преобразованиям на основе поля GF(2^8) с помощью неприводимых полиномов ($x^8 + x^4 + x^3 + x + 1$), как это показано на рис. 7.8.

Преобразование SubByte повторяет процесс, называемый *subbyte*, шестнадцать раз. Inv SubByte повторяет процесс, называемый *invsubbyte*. Каждый шаг преобразования обрабатывает один байт.

В процедуре *subbyte* байт мультипликативная инверсия байта (двоичной строки на 8 битов) находится в GF(2^8) с помощью неприводимого полинома по модулю ($x^8 + x^4 + x^3 + x + 1$). Обратите внимание, что байт — 00_{16} сам является собственной инверсией. Инвертированный байт затем интерпретируется как матрица-столбец с самым младшим битом наверху и самым старшим битом внизу. Эта матрица-столбец умножается на постоянную квадратную матрицу, **X**, и результат, который является матрицей-столбцом, складывается с постоянной матрицей столбца **y**, что дает новый байт. Обратите внимание, что умножение и сложение битов происходит в GF(2). *invsubbyte* делает те же действия в обратном порядке.

После нахождения байта инверсного сомножителя процесс похож на аффинное шифрование, которое мы обсуждали в лекции 3. При шифровании умножение является первой операцией, сложение — второй. При дешифровании вычитание (сложение инверсией) является первым, а деление (умножение с инверсией) — вторым. Мы можем легко доказать, что эти два преобразования инверсны друг другу, потому что сложение или вычитание в GF(2) — фактически операция ИСКЛЮЧАЮЩЕЕ ИЛИ.

subbyte: $d = \mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y}$

invsubbyte: $[\mathbf{X}^{-1}\,(d \oplus y)^{-1} = [\mathbf{X^{-1}}\,(\mathbf{X}\,((s_{r,c})^{-1}\,\mathbf{y} \oplus \mathbf{y}]^{\mathbf{-1}} = [(s_{r,c})^{-1}]^{-1} = s_{r,c}$

Преобразования SubBytes и InvSubBytes инверсны друг другу.

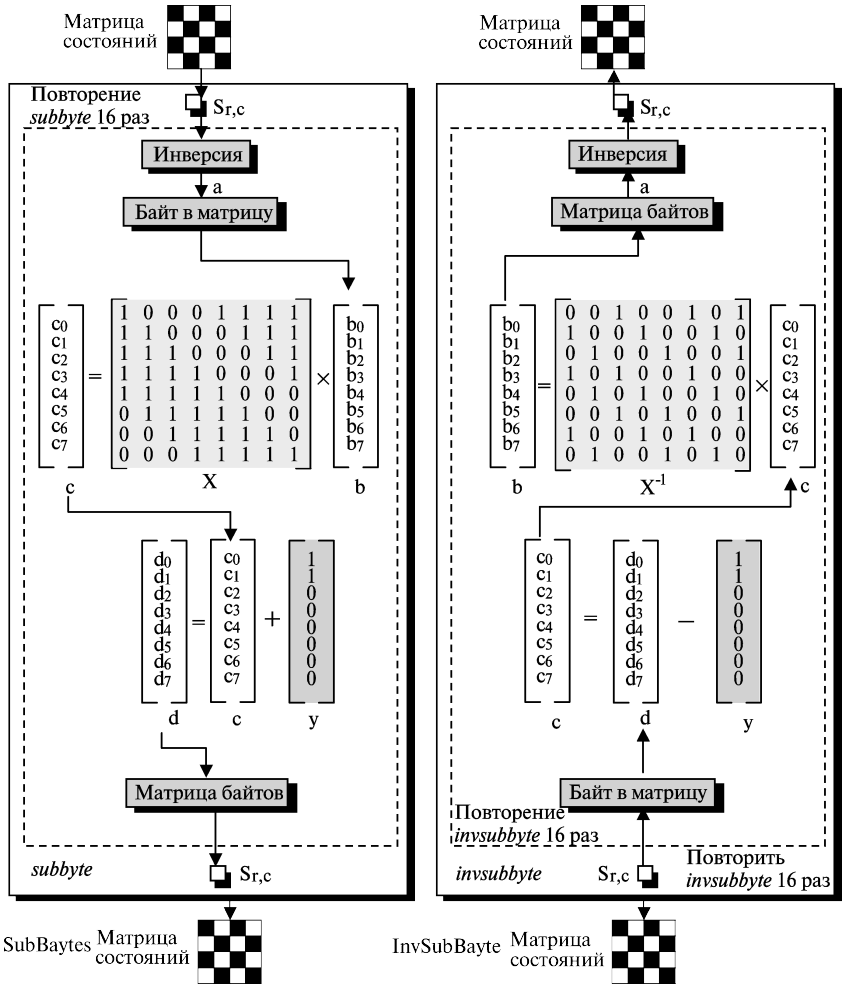


Рис. 7.8. Процессы *SubBytes и InvBytes*

Пример 7.3

Покажем, как байт 0C преобразуется в FE с помощью процедуры *subbyte* и преобразуется обратно в 0C с помощью процесса *invsubbyte.*

1. *subbyte*
   a. Инверсный сомножитель в поле GF(2^8) есть B0 или в двоичном отображении **b** (1011 0000).
   b. Умножая на матрицу **X** эту матрицу, имеем в результате c = (10011101).
   c. Результат применения операции XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ) – **d** = (11111110) FE в шестнадцатеричном представлении.

2. *invsubbyte*
   a. Результат применения операции XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ) – **c** = (10011101)
   b. Результат умножения на матрицу $\mathbf{X}^{-1}$ – (11010000) или B0.
   c. Инверсия по умножению B0 – это 0C.

Алгоритм

Хотя на рисунке мы показали матрицы, чтобы подчеркнуть характер подстановки (аффинное преобразование), алгоритм не обязательно использует умножение и сложение матриц, потому что большинство элементов в постоянной квадратной матрице — только 0 или 1. Значение постоянной матрицы столбца — 0 × 63. Мы можем написать простой алгоритм, чтобы выполнить SubByte. Алгоритм 7.1 вызывает процедуру *subbyte* 16 раз — один раз для каждого байта в матрице состояний.

Процедура ByteToMatrix преобразовывает байт к матрицу-столбец 8 × 1. Процедура MatrixToByte преобразовывает матрицу-столбец 8 × 1 к байту. Расширение этого алгоритма для InvSubBytes оставляем как упражнение.

Нелинейность

Хотя умножение и сложение матриц в процедуре *subbyte* — преобразование аффинного типа и линейно, замена байта его мультипликативной инверсией в $GF(2^8)$ — нелинейная. Этот шаг делает все преобразование нелинейным.

Перестановка

Другое преобразование производит сдвиг в раунде. Этот сдвиг переставляет байты. В отличие от DES, в котором делается поразрядная перестановка, преобразование сдвига делается на уровне байта; порядок битов в байте не меняется.

Алгоритм 7.1. Программа на псевдокоде для преобразования *SubBytes*

```
SubBytes (S)
{
   for (r = 0 to 3)
      (c = 0 to 3)
         S[r,c]=subbyte(S[r,c])
}

subbyte (byte)
{
   a ← byte^-1          //Multiplicative inverse in GF(2^8 ) with inverse
   ByteToMatrix (a,b)   //of 00 to be 00
   For (i= 0 to7)
   {
      c_i ← b_i ⊕ b_(i+4)mod8 ⊕ b_(i+5)mod8 ⊕ b_(i+6)mod8 ⊕ b_(i+7)mod8
      d_i ← c_i ⊕ ByteToMatrix (0 × 63)
   }
   MatrixToByte (d,d)
   byte ← d
}
```

ShiftRows

При шифровании применяется преобразование, называемое **ShiftRows**, со смещением влево. Число сдвигов зависит от номера строки (0, 1, 2 или 3) матрицы состояний. Это означает, что строка 0 не сдвигается и последняя строка сдвигается на три байта. Рисунок 7.9 показывает преобразование смещения.

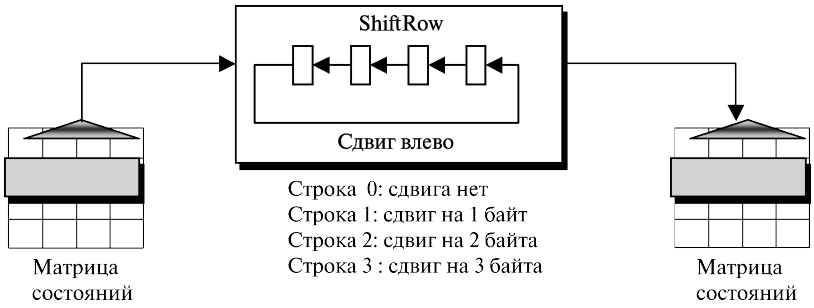


Рис. 7.9. Преобразование *ShiftRows*

Обратите внимание, что преобразование ShiftRows работает одновременно только с одной строкой.

InvShiftRows

При дешифровании применяется преобразование, называемое **InvShiftRows**, со смещением вправо. Число сдвигов равно номеру строки (0, 1, 2 и 3) в матрицы состояний.

ShiftRows- и InvShiftRows-преобразования инверсны друг другу.

Алгоритм

Алгоритм 7.2 для преобразования ShiftRows очень прост. Однако чтобы подчеркнуть, что преобразование делается одновременно только с одной стро-

Алгоритм 7.2. Программа на псевдокоде для преобразования ShiftRows

```
{
   for (r = 1 to 3)
      shiftrow (Sr, r)                    // Sr r-тая строка}

shiftrow (row, n)   // n - число байтов, на которое должен быть сделан сдвиг

{
   CopyRow (row, t) for (c = 0 to 3)   //t - временная строка
   for (c = 0 to 3)
      row (c-n) mod4 ← tc
}
```

кой, мы используем процедуру, называемую *shiftrow*, которая сдвигает байт в единственной строке. Мы вызываем эту процедуру три раза. Процедура *shiftrow* сначала копирует строку во временную матрицу строки (матрица t), а потом сдвигает строку.

Пример 7.4

Рисунок 7.10 показывает, как, используя преобразование ShiftRows, преобразуется матрица состояний. Рисунок также иллюстрирует, как преобразование InvShiftRows создает первоначальную матрицу состояний.

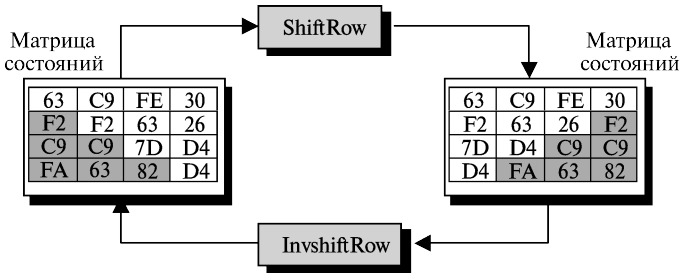


Рис. 7.10 Пример преобразования *ShiftRow* в примере 7.4

Смешивание

Подстановка, которая делается преобразованием SubByte, изменяет значение байта, основанное только на первоначальном значении и входе в таблице; процесс не включает соседние байты. Мы можем сказать, что SubByte — *внутрибайтовое* преобразование. Перестановка, которая делается ShiftRows-преобразованием, обменивает местами байты, не переставляя биты в байтах. Мы можем сказать, что ShiftRows — преобразование *обмена байтами*. Теперь нам нужно внутрибайтовое преобразование, изменяющее биты в байте и основанное на битах в соседних байтах. Мы должны смешать байты, чтобы обеспечить рассеивание на разрядном уровне.

Преобразование смешивания изменяет содержание каждого байта, преобразовывая четыре байта одновременно и объединяя их, чтобы получить четыре новых байта. Чтобы гарантировать, что каждый новый байт будет отличаться от другого (даже если все четыре байта те же самые), процесс сначала умножает каждый байт на различный набор констант и затем смешивает их. Смешивание может быть обеспечено матричным умножением. Как мы обсуждали в лекции 2, когда мы умножаем квадратную матрицу на матрицу-столбец, результат — новая матрица-столбец. После того как матрица умножена на значения строки в матрице констант, каждый элемент в новой матрице зависит от всех четырех элементов старой матрицы. Рисунок 7.11 иллюстрирует эту идею.

AES определяет преобразование, называемое MixColumns. Для применения такого преобразования вводится также обратное преобразование, называемое InvMixColumns. Рисунок 7.12 показывает матрицу констант, используемую для

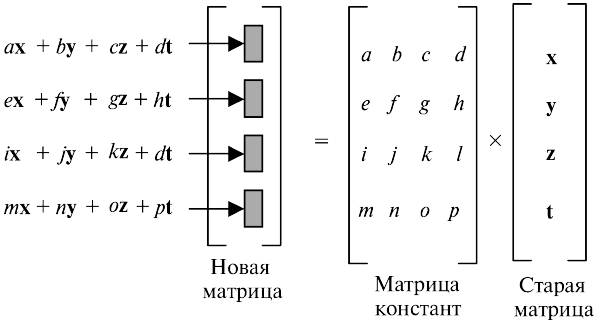


Рис. 7.11. Смешивание байтов с использованием смешивающей матрицы

этих преобразований. Эти две матрицы инверсны друг другу, когда элементы интерпретируются как слова из 8-ми битов (или полиномы) с коэффициентами в GF (2^8). Доказательство мы оставляем как упражнение.

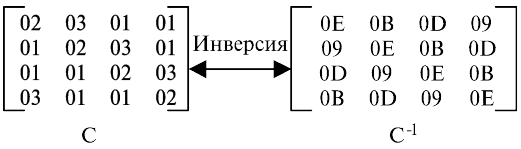


Рис. 7.12. Матрица констант, используемая MixColumns и InvMixColumns

MixColumns

Преобразование **MixColumns** работает на уровне столбца; оно преобразовывает каждый столбец матрицы состояний в новый столбец. Это преобразование — фактически матричное умножение столбца матрицы состояний и квадратной матрицы констант. Байты в столбце матрицы состояний и в матрице констант интерпретируются как слова по 8 битов (или полиномы) с коэффициентами в GF (2). Умножение байтов выполняется в GF(2^8) по модулю (10001101) или ($x^8 + x^4 + x^3 + x + 1$). Сложение — это применение операции ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) к словам по 8 бит. Рисунок 7.13 показывает преобразование MixColumns.

InvMixColumns

Преобразование **InvMixColumns** похоже на MixColumns-преобразование. Если две матрицы констант инверсны друг другу, то легко доказать, что эти два преобразования также инверсны друг другу.

Преобразования MixColumns и InvMixColumns инверсны друг другу.

Алгоритм

Алгоритм 7.3. — программа для преобразования MixColumns.

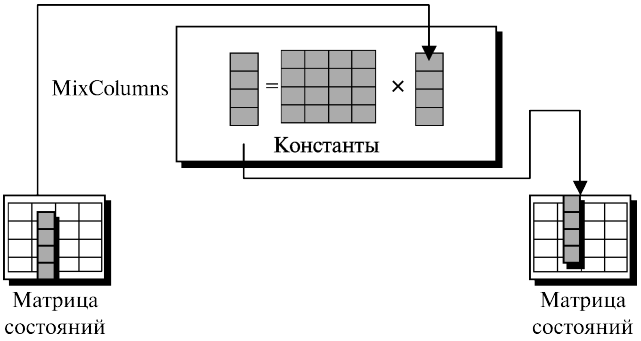


Рис. 7.13. Преобразование MixColumns

Алгоритм 7.3. Программа на псевдокоде для преобразования MixColumns

```
MixColumns (S)
{
   for (c = 0 to 3)
      mixcolumn (sc)
}

mixcolumn (col)
{
   CopyColumn (col, t)        //t is a temporary column

   col0 ← (0x02)• t0 ⊕ (0x03)• t1 ⊕ t2 ⊕ t3
   col1 ← t0 ⊕ (0x02)• t1 ⊕ (0x03)• t2 ⊕ t3
   col2 ← t0 ⊕ t1 ⊕ (0 × 02)• t2 ⊕ (0x03)• t3
   col3 ← (0x03)• t0) ⊕ t1 ⊕ t2 ⊕ (0x02) • t3
}
```

Алгоритмы MixColumns и InvMixColumns включают умножение и сложение в поле GF(2^8). Как мы видели в лекции 4, есть простой и эффективный алгоритм для умножения и сложения в этом поле. Однако чтобы показать характер алгоритма (преобразование одного столбца одновременно), мы используем процедуру, называемую *mixcolumn*. Она может быть вызвана алгоритмом четыре раза. Процедура *mixcolumn* просто умножает строки матрицы констант на столбец в матрицы состояний. В вышеупомянутом алгоритме оператор (•), используемый в процедуре *mixcolumn*, — умножение в поле GF(2^8). Оно может быть заменено простой процедурой, как это уже рассматривалось в лекции 4. Программу для InvMixColumns оставляем как упражнение.

Пример 7.5

Рисунок 7.14 показывает, как матрица состояний преобразуется, используя преобразование MixColumns. Рисунок также показывает, что преобразование InvMixColumns создает первоначальный текст.

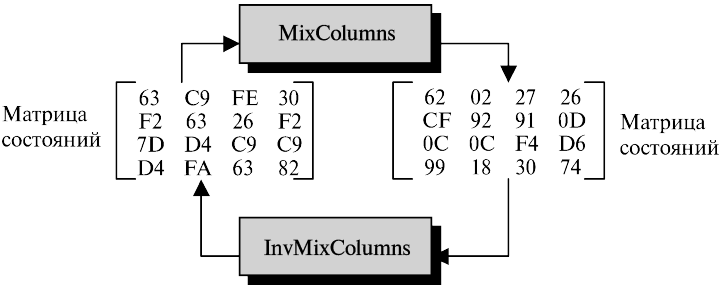


Рис. 7.14. Преобразование *MixColumns [в примере 7.5.]*

Обратите внимание, что байты, которые равны между собой в старой матрице состояний, больше не равны в новой матрице состояний. Например, два байта F2 во второй строке изменены на CF и 0D.

Добавление ключей

Вероятно, самое важное преобразование — это преобразование, которое включает ключ шифра, Все предыдущие преобразования используют известные алгоритмы, которые являются обратимыми. Если не добавлять ключевой шифр в каждом раунде, противник очень просто найдет исходный текст по данному ему зашифрованному тексту. В этом случае хранитель тайны Алисы и Боба — один-единственный ключ шифра.

AES использует процесс, названный ключевым расширением (мы его обсудим позже), который из ключа шифра создает N_r+1 ключей раунда. Каждые ключи раунда — 128-битовой длины, и обрабатываются они как четыре слова по 32 бита. Для добавления ключа к матрице состояний каждое слово рассматривается как матрица-столбец.

AddRoundKey

AddRoundKey обрабатывает в один момент времени один столбец. Он подобен MixColumns. MixColumns умножает квадратную матрицу констант на каждый столбец матрицы состояний. AddRoundKey складывает ключевое слово раунда с каждым столбцом матрицы состояний. В MixColumns применяется матричное умножение; в AddRoundKey — операции сложения и вычитания. Так как сложение и вычитание в этом поле одни и те же, AddRoundKey инверсен сам себе. Рисунок 7.15 показывает преобразование AddRoundKey.

Преобразование AddRoundKey инверсно само себе.

Алгоритм

Преобразование AddRoundKey может быть представлено как операция ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) каждой колонки матрицы состояний с соответствующим ключевым словом. Мы еще будем обсуждать, как расширяется ключ шифрования во множестве ключевых слов, но в данном случае мы определим преобразование, как это показано в алгоритме 7.4. Заметим, что s_c и $w_{round\ +4c}$ матрицы — колонки 4 × 1.

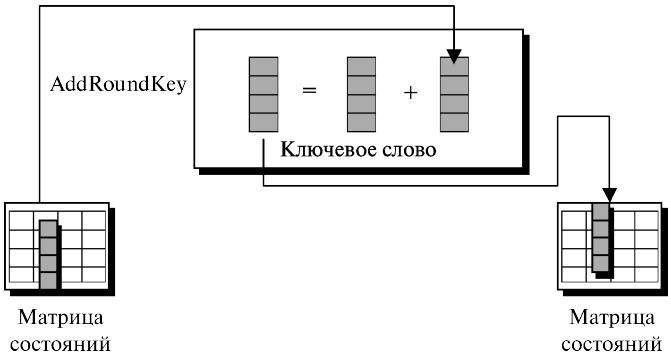


Рис. 7.15. Преобразование *AddRoundKey*

Алгоритм 7.4. Преобразование AddRoundKey

```
AddRoundKey (S)
{
   for (c = 0 to 3)
      sc ← sc ⊕ wround +4c
}
```

Напомним, что оператор ⊕ здесь означает операцию ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) для двух колонок матрицы, каждая из 4-х байт. Подробный разбор этой простой процедуры оставим для упражнений.

7.3. Расширение ключей

Для того чтобы создать ключ для каждого раунда, AES использует процесс ключевого расширения. Если номер раунда — N_r, процедура расширения ключей создает N_r+1 ключ раунда на 128 бит от единственного 128-битового ключа шифра. Первые ключи раунда используются для преобразования перед раундом (AddRoundKey); остающиеся ключи раунда применяются для последнего преобразования (AddRoundKey) в конце каждого раунда.

Процедура расширения ключей создает слово за словом ключи раунда, где слово — массив из четырех байтов. Результатом будут 4 × (N_r +1) слова, которые обозначаются

$$w_0, w_1, w_2 \ldots, w_4 (Nr + 1) - 1$$

Другими словами, в версии AES-128 (10 раундов) имеются 44 слова; в AES-192 версии (12 раундов), есть 52 слова; и в AES 256 версий (с 14 раундами) есть 60 слов. Каждый ключ раунда состоит из четырех слов. Таблица 7.3 показывает отношения между раундами и словами.

Таблица 7.3. Слова для каждого раунда

Раунд	Слова			
1	w_0	w_1	w_2	w_3
2	w_4	w_5	w_6	w_7
.....	.....			
N_r	w_{4Nr}	w_{4Nr+1}	w_{4Nr+2}	w_{4Nr+3}

Расширение ключей в AES-128

Посмотрим, как создаются ключи в версии AES-128; процессы для других двух версий за исключением небольших изменений — те же самые. Рисунок 7.16 показывает, как из исходного ключа получить 44 слова.

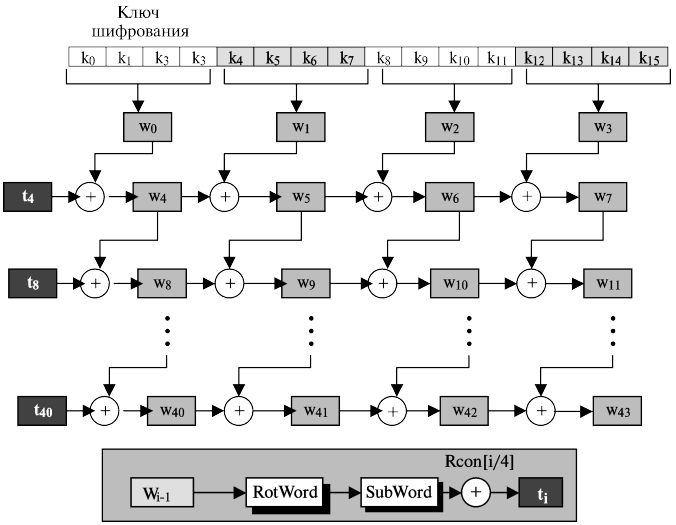


Рис. 7.16. Расширение ключей в AES

Процесс — следующий.

1. Первые четыре слова (w_0,w_1,w_2,w_3) получены из ключа шифра. Ключ шифра представлен как массив из 16 байтов (k_0 до k_{15}). Первые четыре байта (k_0 до k_3) становятся $\mathbf{w}_0$; следующие четыре байта (k4 до k7) становятся $\mathbf{w}_1$; и так далее. Другими словами, последовательное соединение (конкатенация) слов в этой группе копирует ключ шифра.

2. Остальная часть слов (w_i) от $i = 4 - 43$ получается следующим образом:

a. если (i mod 4) $\neq 0$, $w_i = w_{i-1} \oplus w_{i-4}$, то согласно рисунку 7.16 это означает, что каждое слово получено из одного левого и одного верхнего;

b. если (i mod 4) = 0, $w_i = t \oplus w_{i-4}$. Здесь t — временное слово, результат применения двух процессов, *subword* и *rotword*, со словом w_{i-1} и применения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с константой раунда Rcon. Другими словами, мы имеем

$$\mathbf{t = SubWord(Rotword\ (w_{i-1}))} \oplus \text{RCon}_{i/4}.$$

RotWord

RotWord (rotate word) — процедура, подобная преобразованию ShiftRows, но применяется только к одной строке. Процедура принимает слово как массив из четырех байт и сдвигает каждый байт влево с конвертированием.

SubWord

SubWord (substitute word) — процедура, подобная преобразованию SubBytes, но применяется только к одной строке. Процедура принимает каждый байт в слове и заменяет его другим.

RoundConstants

Каждая константа раунда RCon — это 4-байтовое значение, в котором самые правые три байта являются всегда нулевыми. Таблица 7.4 показывает значения для версии AES-128 (с 10 раундами).

Таблица 7.4. Константы RCon

Раунд	Константа (RCon)	Round	Константа (RCon)
1	(**01** 00 00 00)$_{16}$	6	(**20** 00 00 00) $_{16}$
2	(**02** 00 00 00) $_{16}$	7	(**40** 00 00 00) $_{16}$
3	(**04** 00 00 00) $_{16}$	8	(**80** 00 00 00) $_{16}$
4	(**08** 00 00 00) $_{16}$	9	(**1B** 00 00 00)$_{16}$
5	(**10** 00 00 00) $_{16}$	10	(**36** 00 00 00) $_{16}$

Процедура расширения ключей может использовать либо приведенную выше таблицу, когда вычисляет слова, либо поле GF(2^8), когда вычисляет крайние левые биты динамически, как это показано ниже.

$RC_1 \rightarrow x^{1-1} = x^0$ mod *prime* $= 1 \rightarrow 00000001 \rightarrow 01_{16}$

$RC_2 \rightarrow x^{2-1} = x^1$ mod *prime* $= x \rightarrow 00000010 \rightarrow 02_{16}$

RC_3	$\rightarrow x^{3-1} = x^2$	mod *prime*	$= x^2$	$\rightarrow$ 00000100	$\rightarrow 04_{16}$
RC_4	$\rightarrow x^{4-1} = x^3$	mod *prime*	$= x^3$	$\rightarrow$ 00001000	$\rightarrow 08_{16}$
RC_5	$\rightarrow x^{5-1} = x^4$	mod *prime*	$= x^4$	$\rightarrow$ 00010000	$\rightarrow 10_{16}$
RC_6	$\rightarrow x^{6-1} = x^5$	mod *prime*	$= x^5$	$\rightarrow$ 0100000	$\rightarrow 20_{16}$
RC_7	$\rightarrow x^{7-1} = x^6$	mod *prime*	$= x^6$	$\rightarrow$ 01000000	$\rightarrow 40_{16}$,
RC_8	$\rightarrow x^{8-1} = x^7$	mod *prime*	$= x^7$	$\rightarrow$ 10000000	$\rightarrow 80_{16}$
RC_9	$\rightarrow x^{9-1} = x^8$	mod *prime*	$= x^4 + x^3 + x + 1$	$\rightarrow$ 00011011	$\rightarrow 1B_{16}$
RC_{10}	$\rightarrow x^{10-1} = x^9$	mod *prime*	$= x^5 + x^4 + x^2 + x$	$\rightarrow$ 00110110	$\rightarrow 36_{16}$

Крайний левый байт, который обозначен RC_i — это x^{i-1}, где i — номер раунда. AES использует неприводимый полином ($x^8 + x^4 + x^3 + x + 1$).

Алгоритм

Алгоритм 7.5 — простой алгоритм для процедуры расширения ключа (версия AES-128).

Алгоритм 7.5. Программа на псевдокоде для расширения ключей в AES128

```
KeyExpansion ([key0 to key15], [w 0 to w43]
{
    for (i = 0 to 3)
        wi ← key4i, + key4i+1 + key4i+2 + key4i+3
    for (i = 4 to 43)
    {
        if (i mod 4 ≠ 0) wi ← wi-1 + w i-4
        else
        {
            t ← SubWord (RotWord (wi-1)) ⊕ RCon i/4   // t is a temporary word
            wi ← t + wi-4
        }
    }
}
```

Пример 7.6

Таблица 7.5 показывает, как вычисляются ключи для каждого раунда. Предполагается, что между Алисой и Бобом согласован ключ шифра на 128 битов — $(24\ 75\ A2\ B3\ 34\ 75\ 56\ 88\ 31\ E2\ 12\ 00\ 13\ AA\ 54\ 87)_{16}$.

Таблица 7.5. Пример расширения ключей

Раунд	Значения t	Первое слово в раунде	Второе слово в раунде	Третье слово в раунде	Четвертое слово в раунде
–		w_{00}=2475A2B3	w_{01}=34755688	w_{02}=31E21200	w_{03}=13AA5487
1	AD20177D	w_{04}=8955B5CE	w_{05}=BD20E346	w_{06}=8CC2F146	w_{07}=9F68A5C1

2	470678DB	w_{08}=CE53CD15	w_{09}=73732E53	w_{10}=FFB1DF15	w_{11}=60D97AD4
3	31DA48DO	w_{12}= FF8985C5	w_{13}=8CFAAB96	w_{14}=734B7483	w_{15}=2475A2B3
4	47AB5B7D	w_{16}=B822DEB8	w_{17}=34D8752E	W_{18}=479301AD	w_{19}=54010FFA
5	6C762D20	w_{20}= D454F398	w_{21}= E08C86B6	w_{22}= A71F871B	w_{23}= F31E88E1
6	52C4F80D	w_{24}=86900B95	w_{25}=661C8D23	w_{26}= C1030A38	w_{27}=321D82D9
7	E4133523	w_{28}=62833EB6	w_{29}=049FB395	w_{30} C59CB9AD	w_{31}= F7813B74
8	8CE29268	w_{32}=EE61ACDE	w_{33}=EAFE1F4B	w_{34}=2F62A6E6	w_{35}=D8E39D92
9	OA5E4F61	w_{36}= E43FE3BF	w_{37}=OEC1FCF4	w_{38}=21A35A12	w_{39}= F940C780
10	3PC6CD99	w_{40}= DBF92E26	w_{41}=D538D2D2	w_{42}=F49B88CO	w_{43}=ODDB4F40

В каждом раунде вычисление последних трех слов очень просто: мы должны сначала вычислить значение временного слова **(t).** Например, первое t (для раунда 1) вычислено как

RotWord (13AA5487) = AA548713 → **SubWord** (AA548713)= AC20177D
t=AC20177D ⊕ $Rcon_1$=AC20 17 7D ⊕ 01000000_{16} = AD20177D

Пример 7.7

Каждый ключ раунда в AES зависит от предыдущих ключей раунда. Зависимость, однако, нелинейная из-за преобразования SubWord. Сложение констант раунда также гарантирует, что каждый ключ раунда будет отличаться от предыдущего.

Пример 7.8

R.	Ключи для множества 1	Ключи для множества 2	B.D.
–	1245A2A1 2331A4A3 B2CCAA34 C2BB7723	1245A2A1 2331A4A3 B2CCAB34 C2BB7723	01
1	F9B08484 DA812027 684D8A13 AAF6FD30	F9B08484 DA812027 684D8B13 AAF6FC30	02
2	B9E48028 6365AOOF OB282A1C A1DED72C	B9008028 6381AOOF OBCC2B1C A13AD72C	17
3	AOEAF11A C38F5115 C8A77B09 6979AC25	3DOEF11A 5E8F5115 55437A09 F479AD25	30
4	1E7BCEE3 DDF49FF6 1553E4FF 7C2A48DA	839BCEA5 DD149FBO 8857E5B9 7C2E489C	31
5	EB2999F3 36DD0605 238EE2FA 5FA4AA20	A2C910B5 7FDD8F05 F78A6ABC 8BA42220	34
6	82852E3C B4582839 97D6CAC3 C87260E3	CB5AA788 B487288D 430D4231 C8A96011	56
7	82553FD4 360D17ED A1DBDD2E 69A9BDCD	588A2560 ECODODED AF004FDC 67A92FCD	50
8	D12F822D E72295CO 46F948EE 2F50F523	OB9F98E5 E7929508 4892DAD4 2F3BF519	44
9	99C9A438 7EEB31F8 38127916 17428C35	F2794CFO 15EBD9F8 5D79032C 7242F635	51
10	83AD32C8 FD460330 C5547A26 D216F613	E83BDABO FDD00348 AOA90064 D2EBF651	52

Два множества ключей раунда могут быть созданы от двух шифроключей, которые различаются только в одном бите.

Cipher Key 1: 12 45 A2 A1 23 31 A4 A3 B2 CC A**A**34 C2BB 77 23
Cipher Key 2: 12 45 A2 A1 23 31 A4 A3 B2 CC A**B**34 C2 BB 77 23

Как показывает таблица 7.6, имеются существенные разности между двумя соответствующими ключами раунда. R означает «раунд», B и D означают разность битов.

Таблица 7.6. Сравнение двух множеств ключей раунда

Пример 7.9

Понятие слабых ключей, которое мы обсуждали для DES в лекции 6, не применимо к AES. Предположим, что все биты в ключе шифра — нули. Ниже для этого случая показаны слова для некоторых раундов:

Перед раундом:	00000000	00000000	00000000	00000000
Round 01:	62636363	62636363	62636363	62636363
Round 02:	9B9898C9	F9FBFBAA	9B9898C9	F9FBFBAA
Round 03:	90973450	696CCFFA	F2F45733	OBOFAC99
.....	.....	.....	.....	.....
Round 10:	B4EF5BCB	3E92E211	23E951CF	6F8F188E

Все слова перед раундом и в первом раунде равны между собой. Во втором раунде первое слово соответствует третьему; второе слово соответствует четвертому. Однако после второго раунда совпадений нет; все слова различны.

Расширение ключа в AES-192 и AES-256

Алгоритмы расширения ключей в AES-192 и AES-256 очень похожи на алгоритм расширения ключа в AES-128, со следующими отличиями.

1. В AES-192 слова сгенерированы в группы по шесть вместо четырех.
   a. Ключ шифра создает первые шесть слов (w_0 к w_5).
   b. Если $i \bmod 6 \neq 0$, то $w_i \leftarrow w_{i-1} + w_{i-6}$; иначе $w_i \leftarrow t + w_{i-6}$.
2. В AES-256 слова сгенерированы в группы по восемь вместо четырех.
   a. Ключ шифра создает первые восемь слов (w_0 до w_7).
   b. Если $i \bmod 8 \neq 0$, то $w_i \leftarrow w_{i-1} + w_{i-8}$; иначе, $w_i \leftarrow t + w_{i-8}$.
   c. Если $i \bmod 4 = 0$, но $i \bmod 8 \neq 0$, то $w_i = \text{Subword}(w_{i-1}) + w_{i-8}$.

Анализ расширения ключа

Механизм расширения ключа в AES был разработан так, чтобы обеспечить несколько свойств, которые срывают действия криптоаналитика по раскрытию сообщения.

1. Даже если Ева знает только часть ключа шифра или значения слов в некотором ключевом раунде, этого недостаточно: она еще должна найти остальную часть ключа шифра, прежде чем сможет найти ключи всех раундов. Это обеспечивается нелинейностью процесса расширения ключа, полученной с помощью преобразования SubWord.
2. Два различных ключа шифра, независимо от того, как они соотносятся друг с другом, производят два расширения, которые отличаются по крайней мере в нескольких раундах.
3. Каждый бит ключа шифра изменяется в нескольких раундов. Например, изменение единственного бита в ключе шифра изменяет некоторые биты в нескольких раундах.
4. Использование констант, RCons, удаляет любую симметрию, которая может быть создана другими преобразованиями.
5. В отличие от DES в AES отсутствуют любые слабые ключи.

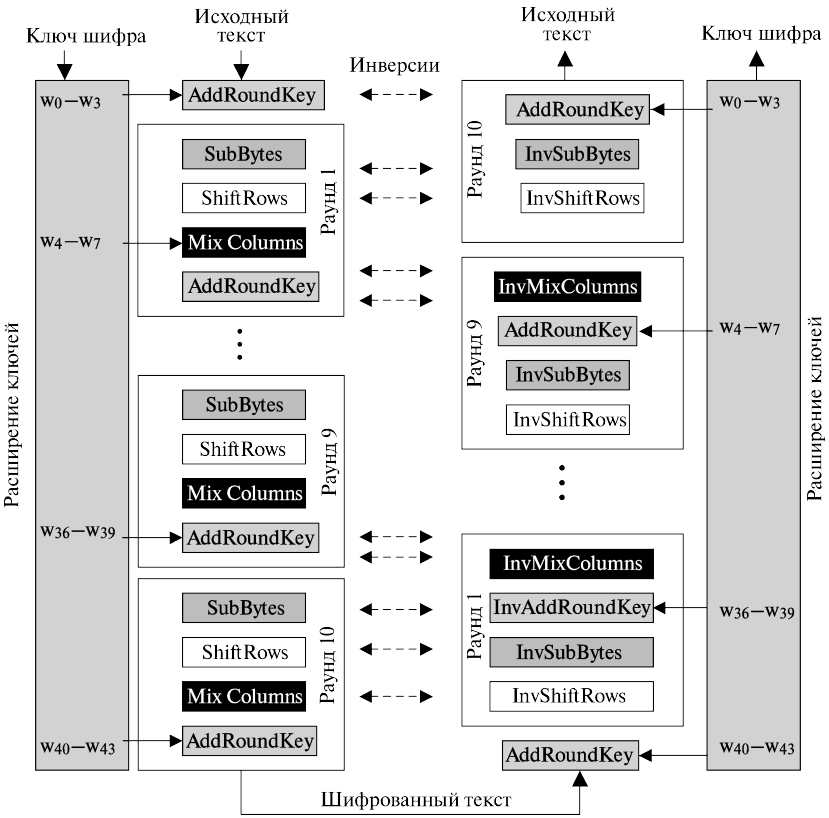


Рис. 7.17. Шифр и обратный шифр начального проекта

6. Процесс расширения ключа может быть легко реализован на всех платформах.
7. Процедура расширения ключа может быть реализована без применения отдельных таблиц; вычисления могут быть сделаны с использованием полей GF(2^8) и FG(2).

7.4. Шифры

Теперь посмотрим, как AES использует четыре типа преобразований для шифрования и дешифрования. В стандарте алгоритм шифрования упоминается как **шифр** и алгоритм дешифрования — как **обратный шифр**.

Как мы упоминали прежде, ADVANCED ENCRYPTION STANDARD — шифр не-Файстеля, а это означает, что каждое преобразование или группа преобразований должны быть обратимыми. Кроме того, шифр и обратный шифр должны использовать эти операции таким способом, чтобы они отменяли друг друга. Ключи раунда должны применяться в обратном порядке. Ниже приведены два различных проекта, которые могут быть использованы для различных реализаций. Мы обсудим оба проекта для AES-128; для других версий применяются те же самые проекты.

Первоначальный проект

В первоначальном проекте порядок преобразований в каждом раунде в шифре и обратном шифре не совпадает. Рисунок 7.17 показывает эту версию.

Во-первых, в обратном шифре изменяется порядок следования SubBytes (InvSubBytes) и ShiftRows (InvShiftRows). Во-вторых, в обратном шифре изменен порядок выполнения MixColumns и AddRoundKey. Эти изменения в порядке необходимы, чтобы в обратном шифре сделать порядок работы обратных преобразований инверсным по отношению к прямому шифру.

Следовательно, алгоритм дешифрования в целом — инверсия алгоритма шифрования. Мы показали только три раунда, но остальные имеют аналогичный вид. Обратите внимание, что ключи раунда используются в измененном порядке. Обратите также внимание, что алгоритмы шифрования и дешифрования в первоначальном проекте не совпадают.

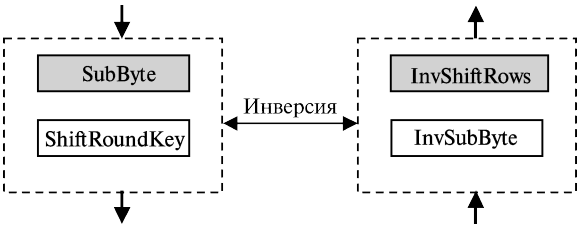
Алгоритм

Код для версии AES-128 этого проекта показан в алгоритме 7.6. Код для обратного шифра оставляем как упражнение.

Алгоритм 7.6. Программа на псевдокоде для шифра первоначального проекта

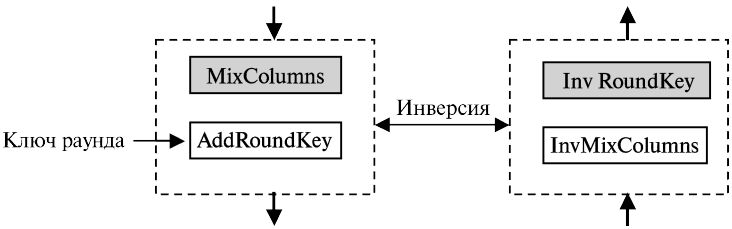
```
Cipher( InBlock [16], OutBlock[16], w[0…43]
{
    BlockToState (InBlock, S)

    S ← SubBytes (S)
    for (round = 1 to10)
```

```
    {
        S ← SubByres (S)
        S ← ShiftRows (S)
        If (round ≠ 10) S ← MixColumns (S)
        S ← AddRoundKey (S, w[4 × round, 4 × round +3])
    }
    StateToBlock (S, OutBlock);
}
```



Альтернативный проект

Для тех приложений, которые предпочитают совпадающие алгоритмы для шифрования и дешифрования, был разработан обратный шифр. В такой версии преобразования в обратном шифре перестроены так, чтобы сделать порядок преобразований тем же самым в прямом шифре и обратном шифре. В этом проекте обеспечена обратимость для пары преобразований, а не для каждого одиночного преобразования.



Пары SubBytes/ShiftRows

SubBytes изменяет содержание каждого байта, не изменяя порядок байтов матрицы состояний; ShiftRows изменяет порядок байтов в матрице состояний, не изменяя содержание байтов. Это подразумевает, что мы можем изменить порядок этих двух преобразований в обратном шифре, не затрагивая обратимость целого алгоритма; рисунок 7.18 иллюстрирует идею. Обратите внимание, что комбинации двух преобразований — в шифре и обратном шифре — инверсны друг другу.

Рис. 7.18. Обратимость совокупности SubByte и ShiftRows

Пара MixColumns/AddRoundKey

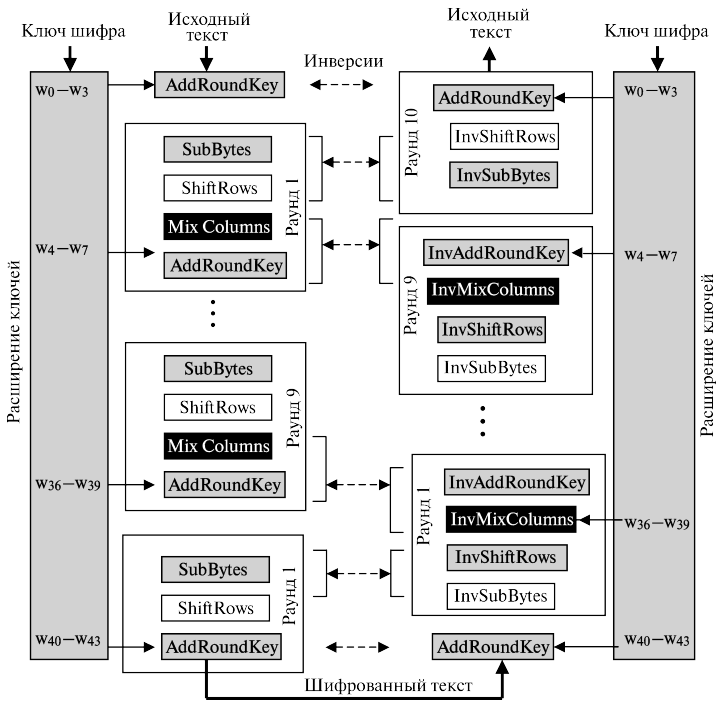
Здесь применяются два преобразования, которые имеют различные свойства. Однако эти пары могут стать инверсиями друг друга, если мы умножим матрицу ключей на инверсию матрицы констант, используемой в преобразовании MixColumns. Мы называем новое преобразование **InvAddRoundKey.** Рисунок 7.19 показывает новую конфигурацию.

Рис. 7.19. Обратимость совокупности MixColumns и AddRoundKey

Можно доказать, что эти две комбинации теперь инверсны друг другу. В шифре мы обозначим входную матрицу состояний — S и выходную матрицу — T. В обратном шифре входная матрица — T. Ниже показано, что выходная матрица состояний — также S. Обратите внимание, что преобразование MixColumns — фактически произведение матрицы C (матрицы констант на матрицу состояний).

Cipher: T = CS ⊕ K

Inverse Cipher: $C^{-1} \oplus C^{-1}K = C^{-1}(CS \oplus K) \oplus C^{-1}K = C^{-1}CS \oplus C^{-1}K \oplus C^{-1}K = S$



Теперь мы можем показать шифр и обратный шифр для альтернативного проекта. Обратите внимание, что мы все еще должны использовать два преобразования AddRoundKey в дешифровании. Другими словами, мы имеем девять InvAddRoundKey и два AddRoundKey преобразования, как это показано на рис. 7.20.

Исходный текст	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19
Ключ шифрования	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
Зашифрованный текст	BC	02	8B	D3	E0	E3	B1	95	55	0D	6D	FB	E6	F1	82	41

Изменение алгоритма расширения ключей

Раунд	Входная матрица состояний	Выходная матрица состояний	Ключ раунда
Предварительный раунд	00 12 OC 08 04 04 00 23 12 12 13 19 14 00 11 19	24 26 3D 1B 71 71 E2 89 BO 44 01 4D A7 88 11 9E	24 34 31 13 75 75 E2 AA A2 56 12 54 B3 88 00 87
1	24 26 3D 1B 71 71 E2 89 BO 44 01 4D A7 88 11 9E	6C 44 13 BD Bl 9E 46 35 C5 B5 F3 02 5D 87 FC 8C	89 BD 8C 9F 55 20 C2 68 B5 E3 F1 A5 CE 46 46 C1
2	6C 44 13 BD Bl 9E 46 35 C5 B5 F3 02 5D 87 PC 8C	1A 90 15 B2 66 09 ID FC 20 55 5A B2 2B CB 8C 3C	CE 73 FF 60 53 73 Bl D9 CD 2E DF 7A 15 53 15 D4
3	1A 90 15 B2 66 09 ID FC 20 55 5A B2 2B CB 8C 3C	F6 7D A2 BO 1B 61 B4 B8 67 09 C9 45 4A 5C 51 09	FF 8C 73 13 89 FA 4B 92 85 AB 74 OE C5 96 83 57
4	F6 7D A2 BO 1B 61 B4 B8 67 09 C9 45 4A 5C 51 09	CA E5 48 BB D8 42 AF 71 Dl BA 98 2D 4E 60 9E DF	B8 34 47 54 22 D8 93 01 DE 75 01 OF B8 2E AD FA
5	CA E5 48 BB D8 42 AF 71 Dl BA 98 2D 4E 60 9E DF	90 35 13 60 2C FB 82 3A 9E FC 61 ED 49 39 CB 47	D4 EO A7 F3 54 8C IF IE F3 86 87 88 98 B6 1B El
6	90 35 13 60 2C FB 82 3A 9E FC 61 ED 49 39 CB 47	18 OA B9 B5 64 68 6A FB 5A EF D7 79 8E B2 10 4D	86 66 C1 32 90 1C 03 ID OB 8D OA 82 95 23 38 D9
7	18 OA B9 B5 64 68 6A FB 5A EF D7 79 8E B2 10 4D	01 63 F1 96 55 24 3A 62 F4 8A DE 4D CC BA 88 03	62 04 C5 F7 83 9F 9C 81 3E B3 B9 3B B6 95 AD 74

8	01 63 F1 96 55 24 3A 62 F4 8A DE 4D CC BA 88 03	2A 34 D8 46 2D 6B A2 D6 51 64 CF 5A 87 A8 F8 28	EE EA 2F D8 61 FE 62 E3 AC IF A6 9D DE 4B E6 92
9	2A 34 D8 46 2D 6B A2 D6 51 64 CF 5A 87 A8 F8 28	OA D9 Fl 3C 95 63 9F 35 2A 80 29 00 16 76 09 77	E4 OE 21 F9 3P Cl A3 40 E3 FC 5A C7 BF F4 12 80
10	OA D9 Fl 3C 95 63 9F 35 2A 80 29 00 16 76 09 77	BC EO 55 E6 02 E3 OD Fl 8B Bl 6D 82 D3 95 F8 41	DB D5 F4 OD F9 38 9B DB 2E D2 88 4F 26 D2 CO 40

Вместо того чтобы использовать преобразование InvRoundKey в обратном шифре, можно изменить алгоритм расширения ключей так, чтобы создавать различное множество ключей раунда для обратного шифра.

18 0A B9 B5 64 68 6A PB SA BF D7 79 8E 82 10 D4	7C FB Al 90 36 80 AA FC ID E3 BF 7E 7B 4B F4 C4	C4 DE F7 9E 4C 95 CO 35 FD 7B 69 C7 59 E3 IE BA	2A 34 D8 46 2D 6B A2 D6 51 64 CF 5A 87 A8 F8 28	01 03 F1 96 55 24 3A 62 F4 8A DE 4D CC BA 88 03
Входная матрица состояний	После SubBytes	После ShiftRows	После MixColumns	Выходная матрица состояний

Однако отметим, что ключ раунда для операций предварительного раунда и последнего раунда должен быть изменен. Ключи раунда от 1 до 9 необходимо умножить на матрицу констант. Этот алгоритм оставим для упражнений.

Исходный текст	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Ключ шифрования	24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87
Зашифрованный текст	63 2C D4 5E 5D 56 ED B5 62 04 01 AO AA 9C 2D 8D

7.5. Примеры

В этом разделе приводятся некоторые примеры шифрования, дешифрования и генерации ключей, которые обсуждались в предыдущих разделах.

Исходный текст 1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Исходный текст 2	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
Зашифрованный текст 1	63 2C D4 5E 5D 56 ED B5 62 04 01 AO AA 9C 2D 8D
Зашифрованный текст 2	26 F3 9B BC A1 9C 0F B7 C7 2E 7E 30 63 92 73 13

Рис. 7.20. Шифр и обратный шифр альтернативного проекта

Пример 7.10

Ниже показан блок зашифрованного текста, полученный из исходного текста с помощью случайной выборки ключей.

Исходный текст	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19
Ключ шифрования	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Зашифрованный текст	5A	GF	4B	67	57	B7	A5	D2	C4	30	91	ED	64	9A	42	72

Таблица 7.7. показывает значения матрицы состояний и ключей раунда для этого примера.

Таблица 7.7. Пример шифрования

Пример 7.21

Пример показывает матрицы состояний, раунд 7 в примере 7.10.

Рис. 7.21. Матрицы состояний одного раунда

Пример 7.12

Один из курьезных случаев при рассмотрении шифрования — когда исходный текст состоит из одних нулей. Используем ключ шифра из примера 7.10 и получаем зашифрованный текст:

Пример 7.13

Давайте проверим лавинный эффект, который мы обсуждали в лекции 6. Изменим только один бит в исходном тексте и сравним результаты. Мы изменили только один бит в последнем байте. Результат показывает эффект рассеивания и перемешивания. Изменение единственного бита в исходном тексте затронуло много бит в зашифрованном тексте.

Пример 7.14

Ниже показан эффект использования ключа шифрования «все нули».

7.6. Анализ AES

Далее дан краткий обзор трех характеристик AES.

Безопасность

AES был разработан после DES. Большинство известных атак на DES было проверено на AES; ни одна из них до сих пор не нарушила безопасность AES.

Атака грубой силы

AES явно более безопасен, чем DES, из-за большего размера ключа (128, 192 и 256 битов). Давайте сравним DES с ключом шифра на 56 битов и AES с ключом шифра на 128 битов. Для DES, чтобы найти ключ, мы нуждаемся в 2^{56} испытаний (игнорируя проблему дополнения ключа); для AES — в 2^{128} испытаниях. Это означает, что если мы можем нарушить DES в t секунд, нам нужно ($2^{72} \times t$) секунд, чтобы нарушить AES. Это почти невозможно. Кроме того, AES обеспечивает две другие версии более длинными ключами шифра. Отсутствие слабых ключей — еще одно преимущество AES перед DES.

Статистические атаки

Сильное рассеивание и перемешивание, обеспеченное комбинацией преобразований SubByte, ShiftRows и MixColumns, удаляют любую частотную закономерность в исходном тексте. Многочисленные попытки выполнить статистический анализ зашифрованного текста не увенчались успехом.

Дифференциальные и линейные атаки

AES был разработан после DES. Дифференциальные и линейные атаки криптоанализа были, без сомнения, учтены. Подобные атаки на AES пока не обнаружены.

Реализация

AES может быть реализован в программном обеспечении, аппаратных средствах и программируемом оборудовании. Реализация может применять процесс поиска в таблице или процедуры, в которых существует четкая алгебраическая структура. Преобразование может быть ориентировано или на байт, или на слово. В ориентированной на байт версии алгоритм может использовать процессор на 8 битов; в ориентированной на слово версии — процессор на 32 бита. В любом случае, обработка делается очень быстро.

Простота и стоимость

Алгоритмы, применяемые в AES, настолько просты, что могут быть легко реализованы с помощью дешевых процессоров и минимального объема памяти.

7.7. Рекомендованная литература

Нижеследующие книги и сайты дают более детальную информацию о предметах, рассмотренных в этой лекции. Пункты, приведенные в квадратных скобках, содержатся в списке в конце книги.

Книги

[Sta06L [Sti06L [Rhe0S], [Sal03], [Mao04L] и [TW06] рассматривают AES.

Сайты

Нижеследующие сайты дают больше информации о темах, рассмотренных в этой лекции.

csrc.nist.gov/publications/fips/ripsl97/fips-197.pdf
http://www.quadibloc.com/crypto/co040401 .htm
http://www. ietef.org/rfc/rfc 3394. txt

7.8. Итоги

- Усовершенствованный стандарт шифрования (AES — ADVANCED ENCRYPTION STANDARD) — стандарт блочного шифра с симметричными ключами, опубликованный Национальным Институтом стандартов NIST (National Institute of Standard and Technology) как Федеральный Стандарт Обработки Информации 197 (FIPST-197 — FEDERAL INFORMATION PROCESSING STANDARD 197). AES базируется на Rijndael-алгоритме.
- AES — шифр не-Файстеля, который зашифровывает и расшифровывает блок данных длиной 128 битов. Оно использует 10, 12 или 14 раундов. Размер ключа, 128, 192 или 256 битов, зависит от числа раундов.
- AES ориентирован на работу с байтами. Исходный текст на 128 битов или зашифрованный текст рассматривается как шестнадцать байтов по 8 битов. Чтобы обеспечить выполнение некоторых математических преобразований для байтов, в AES определено понятие матрицы состояний. Матрица состояний — это матрица 4×4, в которой каждый вход является байтом.
- Чтобы обеспечить безопасность, AES использует четыре типа преобразований: подстановка, перестановка, смешивание и добавление ключа. Каждый раунд AES, кроме последнего, применяет эти четыре преобразования. Последний раунд использует только три из четырех преобразований.
- Подстановка определяется либо процессом поиска в таблице, либо математическим вычислением в поле $GF(2^8)$. AES использует два обратимых преобразования — SubBytes и InvSubBytes, которые являются инверсиями друг друга.
- Второе преобразование в раунде — сдвиг, которое переставляет байты. В шифровании преобразование названо ShiftRows, в дешифровании — InvShiftRows. Преобразования ShiftRows и InvShiftRows инверсны друг другу.
- Преобразование смешивания изменяет содержание каждого байта, обрабатывая одновременно четыре байта и объединяя их, чтобы получить четыре новых байта. AES определяет два преобразования, MixColumns и InvMixColumns, применяемые в шифровании и дешифровании. MixColumns умножает матрицу состояний на квадратную матрицу констант; InvMixColumns делает то же самое, используя обратную матрицу констант. Преобразования MixColumns и InvMixColumns инверсны друг другу.
- Преобразование, которое выполняет отбеливание, называется AddRoundKey. Предыдущая матрица состояний складывается (матрич-

ное сложение) с матричным ключом раунда, чтобы создать новую матрицу. Сложение отдельных элементов в этих двух матрицах выполняется в GFE(2^8) — это означает, что над словами по 8 битов проводится операция ИСКЛЮЧАЮЩЕЕ ИЛИ (XORed). Преобразование AddRoundKey инверсно само себе.

- В первой конфигурации (10 раундов с ключами на 128 битов) генератор ключей создает одиннадцать ключей раунда на 128 битов из ключа шифра на 128 битов. AES использует понятие слова для генерации ключей. Слова состоят из четырех байт. Ключи раунда генерируются слово за словом. AES нумерует слова от w_2 до w_{43}. Процесс называется «расширение ключа».
- Шифр AES для дешифрования использует два алгоритма. В первоначальном проекте порядок преобразований в каждом раунде — различен при шифровании и дешифровании. В альтернативном проекте преобразования в алгоритмах дешифрования перестроены так, чтобы порядок в шифровании и дешифровании был один и тот же. Во второй версии обратимость обеспечена для пары преобразований.

7.9. Набор для практики

Обзорные вопросы

1. Перечислите критерии, определенные NIST для AES.
2. Перечислите параметры (размер блока, размер ключа и число раундов) для трех версий AES.
3. Сколько преобразований имеется в каждой версии AES? Сколько ключей необходимо для каждой версии?
4. Сравните DES и AES. Какой из них ориентирован на работу с битом, а какой — на работу с байтом?
5. Определите матрицу состояний в AES. Сколько матриц состояний имеется в каждой версии AES?
6. Какие из четырех преобразований, определенных для AES, изменяют содержание байтов, а какие — не изменяют?
7. Сравните подстановку в DES и AES. Почему мы имеем только одну таблицу перестановки (S-блок) в AES и несколько — в DES?
8. Сравните перестановки в DES и AES. Почему надо иметь расширение и сжатие перестановки в DES и не надо — в AES?
9. Сравните ключи раунда в DES и AES. В каком шифре размер ключа раунда равен размеру блока?
10. Почему смешивающее преобразование (MixColumns) нужно в DES, но не нужно в AES?

Упражнения

1. При шифровании S-блоки могут быть или *статическими,* или *динамическими.* Параметры в статическом S-блоке не зависят от ключа.

a. Указать преимущества и недостатки статического и динамического S-блоков.
b. S-блоки в AES (таблицы подстановки) — статические или динамические?

2. AES имеет больший размер блока, чем DES (128 — в AES и 64 — в DES). Объясните, преимущество это или недостаток.
3. AES определяет различные реализации с различным числом раундов (10, 12 и 14); DES определяет только одну реализацию с 16 раундами. Объясните, преимущество это или недостаток AES и DES и в чем отличие.
4. AES определяет три различных размера ключа к шифру (128, 192 и 256); DES определяет только один размер ключа к шифру (56). Каковы преимущества и недостатки такого отличия?
5. В AES размер блока равен размеру ключей раунда (128 бит). В DES размер блока 64 бита, но размер ключей раунда — только 48 бит. Является ли эта разница преимуществом или недостатком AES по сравнению с DES?
6. Докажите, что преобразования ShiftRows и InvShiftRows — инверсны.
   a. Покажите таблицу перестановки для ShiftRows. Таблица должна иметь 128 входов, но так как содержание байта не изменяется, таблица может иметь только 16 выходов; каждый выход представляет байт.
   b. Повторите чсть *a* для InvShiftRows преобразования.
   c. Используя результаты частей *a* и *b*, докажите, что преобразования ShiftRows и InvShiftRows инверсны друг другу.
7. Используйте один и тот же ключ шифра, применив его для каждого из следующих преобразований на двух исходных текстах, которые отличаются только по первому биту. Найдите число изменившихся битов после каждого преобразования.
   a. SubBytes
   b. ShiftRows
   c. Mix Columns
   d. AddRoundKey (с теми же самыми ключами раунда по вашему выбору)
8. Для того чтобы увидеть нелинейность преобразования SubBytes, покажите, что если *a* и *b* — два байта, то мы имеем

$$\text{SubBytes}\ (a \oplus b) \neq \text{SubBytes}\ (a) \oplus \text{SubBytes}\ (b)$$

Как пример используйте $a = 0x57$ и $b = 0xA2$.

9. Дайте общую формулу для вычисления числа в каждом из видов преобразования SubBytes, ShiftRows, MixColumns и AddRoundKey и числа полных преобразований для каждой версии AES. Формула должна быть функцией числа раундов.
10. Измените рисунок 7.16 для AES-192 и AES-256.
11. Создайте две новые таблицы, которые показывают RCons константы для реализаций AES-192 и AES-256 (см. таблицу 7.4).

12. В AES-128 для предварительного раунда используются такие же ключи, как ключи шифрования. Справедливо ли это для AES-192? Справедливо ли это для AES-256?
13. На рисунке 7.8 перемножьте X и X^{-1} матрицы, чтобы доказать, что они инверсны друг другу.
14. Используя рисунок 7.12, перепишите квадратные матрицы C и C^{-1}, применяя полиномы с коэффициентами в GF(2). Перемножьте эти две матрицы и докажите, что они являются обратными друг другу.
15. Докажите, что код в алгоритме 7.1 (преобразование SubByte) соответствует процессу, показанному на рис. 7.8.
16. Используя алгоритм 7.1 (преобразование SubByte), сделайте следующее:
    a. Напишите код для процедуры, которая вычисляет инверсию байта в $GF(2^8)$.
    b. Напишите код для ByteToMatrix.
    c. Напишите код для MatrixToByte.
17. Напишите алгоритм для преобразования InvSubBytes.
18. Докажите, что код в алгоритме 7.2 (преобразование ShiftRows) соответствует процессу, показанному на рис. 7.9.
19. Используя алгоритм 7.2 (преобразование ShiftRows), напишите код для процедуры *copyrow*.
20. Напишите алгоритм для преобразования InvShiftRows.
21. Докажите, что код в алгоритме 7.3 (преобразование MixColumns) соответствует процессу, показанному на рис. 7.13.
22. Используя алгоритм 7.3 (преобразование MixColumns), напишите код для процедуры *copycolumns*.
23. Перепишите алгоритм 7.3 (преобразование MixColumns), заменив операторы (.) процедурой, называемой *multfield*, чтобы вычислить произведение двух байтов в поле $GF(2^8)$.
24. Напишите алгоритм для преобразования InvMixColumn.
25. Докажите, что код в алгоритме 7.4 (преобразование AddRoundKey), соответствует процессу, показанному на рис. 7.15.
26. В алгоритме 7.5 (расширение ключа):
    a. Напишите кодовую программу для процедуры Subbyte.
    b. Напишите код программу для процедуры Rotword-.
27. Дайте два новых алгоритма для расширения ключа в AES-192 и AES-256 (см. алгоритм 7.5).
28. Напишите алгоритм расширения ключей для обратного шифра в альтернативном проекте.
29. Напишите алгоритм для обратного шифра в первоначальном проекте.
30. Напишите алгоритм для обратного шифра в альтернативном проекте.

Лекция 8. Шифрование, использующее современные шифры с симметричным ключом

Цели и содержание

Эта лекция имеет несколько целей.

- Показать, как современные стандартные шифры, такие, как DATA ENCRIPTION STANDARD или ADVANCED ENCRYPTION STANDARD, используются для того, чтобы зашифровать длинные сообщения.
- Обсудить пять режимов работы, разработанных для применения с современными блочными шифрами.
- Определить, какой режим работы создает шифр потока из основных блочных шифров.
- Обсудить аспекты безопасности и распространение ошибок при различных режимах работы.
- Обсудить два шифра потока, применяемые для обработки данных в реальном масштабе времени.

Эта лекция показывает, как концепции двух современных блочных шифров, рассмотренные в лекциях 5, 6 и 7, могут использоваться, чтобы зашифровать длинные сообщения. Она также вводит два новых понятия шифра потока.

8.1. Применение современных блочных шифров

Шифрование симметричными ключами может быть выполнено средствами современных блочных шифров. Два современных блочных шифра, обсужденные в лекциях 6 и 7, а именно DES и AES, разработаны для того, чтобы зашифровать и расшифровать блок текста фиксированного размера. DES зашифровывает и расшифровывает блок 64 битов; AES — блок 128 битов. В реальной жизни текст, который будет зашифрован, имеет переменный размер и обычно намного больший, чем 64 или 128 битов. **Режимы работы** были изобретены, чтобы зашифровать текст любого размера, используя либо DES, либо AES. Рисунок 8.1 показывает эти пять режимов работы, которые мы обсудим далее.

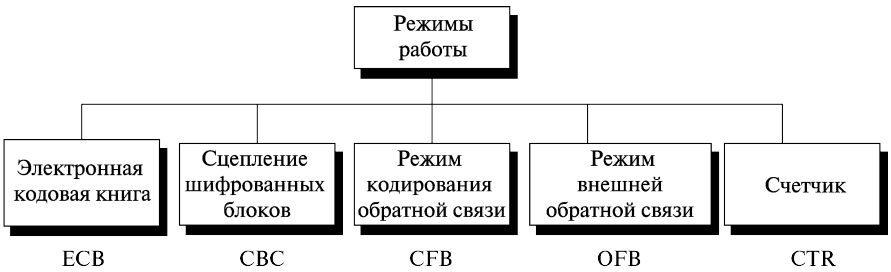
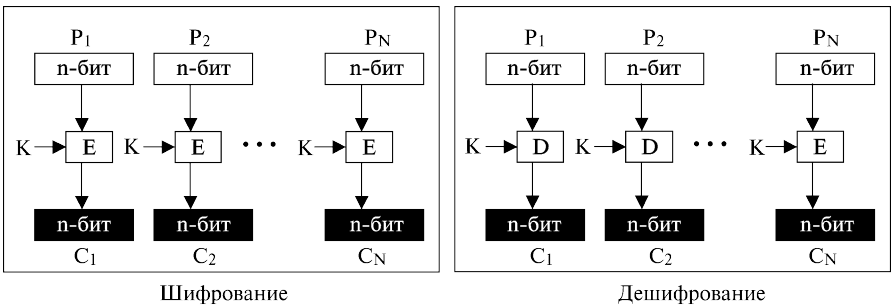


Рис. 8.1. Режимы работы

Режим электронной кодовой книги

Самый простой режим работы назван режимом **электронной кодовой книги (ECB — ELECTRONIC CODEBOOK)**. Исходный текст разделен на *N* блоков. Размер блока — *n* бит. Этот размер исходного текста не является кратным числом размера блока, текст дополняется, чтобы сделать последний блок по размеру таким же, как другие блоки. Один и тот же ключ используется, чтобы зашифровать и расшифровывать каждый блок. Рисунок 8.2 показывает шифрование и дешифрование в этом режиме.



E Шифрование D Дешифрование
P_i Блок исходного текста *i* C_i Блок зашифрованного текста i
K Секретный ключ

Рис. 8.2. Режим электронной кодовой книги (ECB)

Соотношение между исходным и зашифрованным текстами показано ниже:

$$\text{Шифрование: } C_i = E_K(P_i) \qquad \text{Дешифрование: } P_i = D_K(C_i)$$

Пример 8.1

Этот пример показывает, как можно обеспечить, что каждый блок, посланный Алисой, может быть точно восстановлен на стороне Боба. Здесь используется то, что шифрование и дешифрование инверсны друг другу.

$$P_i = D_K(C_i) = P_i = D_K(E_K(P_i))$$

Пример 8.2

Такой режим называется режимом электронной кодовой книги, потому что он может быть представлен 2^K кодовыми книгами (одной на каждый кодовый ключ). Каждая кодовая книга имеет 2^n входов и две колонки. Каждый вход сопоставляет исходному тексту соответствующий зашифрованный текст. Однако если K и n очень велики, кодовая книга будет слишком велика и ее компоновка и эксплуатация — неудобны.

Проблемы безопасности

В режиме EBC имеются следующие проблемы безопасности.

1. Образцы на уровне блока сохраняются. Например, одинаковые блоки в исходном тексте имеют одинаковый вид в соответствующих блоках зашифрованного текста. Если Ева узнает, что в зашифрованном тексте блоки 1, 5 и 10 одинаковы, она поймет, что блоки исходного текста 1, 5 и 10 — тоже одинаковые. Это — «дырочка» в безопасности. Например, Ева может выполнить исчерпывающий поиск и расшифровать только один из этих блоков, чтобы найти содержание всех их.
2. Независимость блоков создает Еве возможность для замены некоторых блоков зашифрованного текста без знания ключа. Например, если она знает, что блок 8 всегда передает некоторую заданную информацию, она может заменить этот блок соответствующим блоком в предварительно перехваченном сообщении.

Пример 8.3

Предположим, что Ева работает в компании месяц, несколько часов в неделю (ее ежемесячная оплата очень низкая). Она знает, что компания использует несколько блоков информации для каждого служащего, в котором седьмой блок является суммой денег, которая будет депонирована в учетной записи служащего. Ева может прервать зашифрованный текст, передаваемый банку в конце месяца, заменить блок информацией о ее оплате с копией блока с информацией об оплате полной рабочей недели её коллеги. Каждый месяц Ева может получать больше денег, чем ей положено.

Распространение ошибки

Единственный бит ошибки в передаче может создать ошибки в нескольких битах (обычно половине битов или всех битах) в соответствующих блоках. Однако ошибка не имеет никакого воздействия на другие блоки.

Алгоритм

Для шифрования или дешифрования могут быть созданы простые алгоритмы. Алгоритм 8.1 содержит процедуру, написанную в псевдокоде для шифрования. Процедуру для дешифрования оставляем как упражнение. E_K зашифровывает только один-единственный блок и может быть одним из шифров, рассмотренных в главах 6 или 7 (DES или AES).

Алгоритм 8.1. Режим шифрования ECB

```
ECB_Encryption (K, Plaintext bloks)
{
    for(i = 1 to N)
    {
        Ci ← EK(Pi)
    }
    return Cliphertext blocks
}
```

Захват зашифрованного текста

В режиме ECB может потребоваться дополнение, которое добавляется к последнему блоку, если он содержит менее *n* бит. Такое дополнение не всегда возможно. Например, рассмотрим ситуацию, когда зашифрованный текст должен быть сохранен в буфере, где до этого был предварительно сохранен исходный текст. В этом случае исходный текст и зашифрованный текст должны быть одинаковой длины. Техника, которая называется **захват зашифрованного текста (CTS — CipherText Stealing)**, позволяет использовать режим ECB без указанного выше дополнения. В этой методике последние два блока исходного текста P_{N-1} и P_N зашифрованы раздельно другим способом и в другом порядке, как показано ниже. Предположим, что P_{N-1} имеет *n* бит, а P_N имеет m бит, где $m < n$.

$$X = E_K(P_{N-1}) \quad \rightarrow \quad C_N = head_m(X)$$
$$Y = P_N \mid tail_{n-m}(X) \quad \rightarrow \quad C_{N-1} = E_K(Y)$$

где $head_m$ — функция, отделяющая крайние левые m бит, $tail_{n-m}$ — функция, отделяющая крайние правые n-m бит.

Приложения

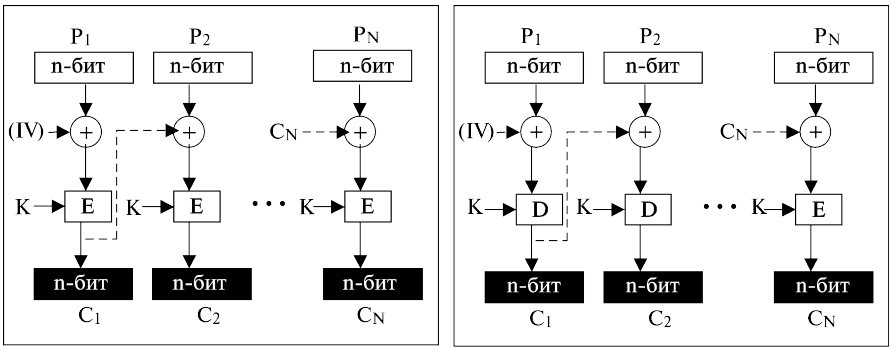
Режим работы ECB не рекомендуется для шифрования сообщений, содержащих больше чем один блок, который будет передан через несекретный канал. Если сообщение достаточно коротко, чтобы передать его в одном блоке, проблемы безопасности и распространения ошибок терпимы.

Есть одна область, где независимость между блоками зашифрованного текста полезна, — это там, где информация будет зашифрована прежде, чем она будет сохранена в базе данных, или расшифрована прежде, чем она будет извлечена из памяти. Поскольку порядок шифрования и дешифрования блоков не важен, в этом режиме доступа к базе данных он может быть случаен, если каждая запись — блок или множество блоков. Запись может быть извлечена из середины, расшифрована, и зашифрована, не затрагивая другие записи. Другое преимущество этого режима — это то, что мы можем использовать параллельную обработку, если нужно, например, создать огромную зашифрованную базу данных.

Режим сцепления блоков шифрованного текста (CBC)

Следующая эволюция в работе режимов — **режим сцепления блоков шифрованного текста (CBC — Cipher Block Chaining)**. В режиме CBC каждый блок исходного текста, прежде чем быть зашифрованным, обрабатывается с помощью проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с предыдущим блоком шифра. Когда блок зашифрован, блок передают, но копия сохраняется в памяти, которая используется в шифровании следующего блока. Читатель может задать вопрос о начальном блоке, поскольку перед первым блоком нет блока зашифрованного текста. В этом случае применяется фальшивый блок, называемый **вектор инициализации (IV)**. Передатчик и приемник согласуют заданный заранее IV. Другими словами, IV используется вместо несуществующего C_0. Рисунок 8.3 показывает режим CBC. На передающей стороне операция ИСКЛЮЧАЮЩЕЕ ИЛИ прово-

дится перед шифрованием; а на стороне приемника дешифрование проводится перед операцией ИСКЛЮЧАЮЩЕЕ ИЛИ.



E Шифрование D Дешифрование
P_i Блок исходного текста *i* C_i Блок зашифрованного текста i
K Секретный ключ

Рис. 8.3. Режим цепочки блоков шифротекста

Соотношение между исходным текстом и зашифрованным текстом показано ниже:

Шифрование:	**Дешифрование:**
$C_0 = IV$	$C_0 = IV$
$C_i = E_K (P_i \oplus C_{i-1})$	$P_i = D_K (C_i) \oplus C_{i-1}$

Пример 8.4

Можно доказать, что каждый блок исходного текста на стороне Алисы может быть точно восстановлен на стороне Боба — потому что шифрование и дешифрование инверсны друг другу.

$$P_i = D_K (C_i) \oplus C_{i-1} = D_K (E_K (P_i \oplus C_{i-1})) \oplus C_{i-1} = P_i \oplus C_{i-1} \oplus C_{i-1} = P_i$$

Вектор инициализации (IV)

Вектор инициализации (IV) должен быть известен передатчику и приемнику. Хотя сохранение этого вектора в тайне не требуется, целостность вектора играет важную роль в безопасности режима CBC; IV помогает сохранить безопасность изменения информации. Если Ева может изменить значения бит IV, это может изменить значения бит первого блока.

Для того чтобы использовать IV, рекомендовано несколько методов. Передатчик может выбрать псевдослучайное число и передать его через безопасный канал (например, использующий режим ECB). Фиксированное значение может

быть согласовано Алисой и Бобом как IV, когда ключ засекречивания установлен. Это может быть часть ключа засекречивания, и так далее.

Проблемы безопасности

В режиме **CBC** имеются следующие две проблемы безопасности.

1. Одинаковые блоки исходного текста, принадлежащие одному и тому же сообщению, зашифровываются в различные блоки зашифрованного текста. Другими словами, отдельные образцы в блоке не сохраняются. Однако когда два сообщения равны, их шифрованный текст одинаковый, если они используют те же самые IV. Фактически, если первые M блоков в двух различных сообщениях равны и IV совпадает, они будут зашифрованы в одинаковые блоки. По этой причине некоторые специалисты рекомендуют использование для IV метки времени.
2. Ева может добавить некоторые блоки зашифрованного текста в конце потока зашифрованного текста.

Распространение ошибки

В режиме CBC единственный бит ошибки в блоке C_j зашифрованного текста в процессе передачи может в процессе дешифрования создать ошибку в большинстве битов блока P_j исходного текста. Однако эта одиночная ошибка изменяет только один бит в исходном тексте блока P_{j+1} (бит в том же самом местоположении). Доказательство этого факта оставляем как упражнение. Исходный текст блоков от P_{j+2} до P_N не затрагивается этим единственным битом ошибки. Единственный бит ошибки в зашифрованном тексте — *самовосстанавливаемый.*

Алгоритм

Алгоритм 8.2 дает псевдокод для шифрования — процедура *encrypt.* Она зашифровывает единственный блок (например, DES или AES). Алгоритм дешифрования оставляем как упражнение.

Алгоритм 8.2. Алгоритм шифрования для режима ECB

```
CBC_Encryption (IV, K, Plaintext bloks)
{
   C0 ← IV
   for (i =1 to N)
   {
      Temp ← Pi ⊕ Ci-1
      Ci ← EK (Temp)
   }
return Ciphertext blocks
}
```

Захват зашифрованного текста

Методика захвата зашифрованного текста, рассмотренная для режима ECB, может также быть применена к режиму CBC, как это показано ниже.

$$U = P_{N-1} \oplus C_{N-2} \rightarrow X = E_K(U) \rightarrow C_N = head_m(X)$$
$$V = P_N | pad_{n-m}(0) \rightarrow Y = X \oplus V \rightarrow C_{N-1} = E_K(Y)$$

Функция *head* — та же самая, что описана в режиме ECB; функция *pad* вставляет нули.

Приложения

Режим работы CBC может использоваться, чтобы зашифровать сообщения. Однако из-за механизма формирования цепочки параллельная обработка невозможна. Режим CBC не используется при шифровании и дешифровании массивов файлов с произвольным доступом, потому что шифрование и дешифрование требуют доступа к предыдущим массивам. Как мы увидим в лекции 11, режим CBC применяется для установления подлинности сообщения.

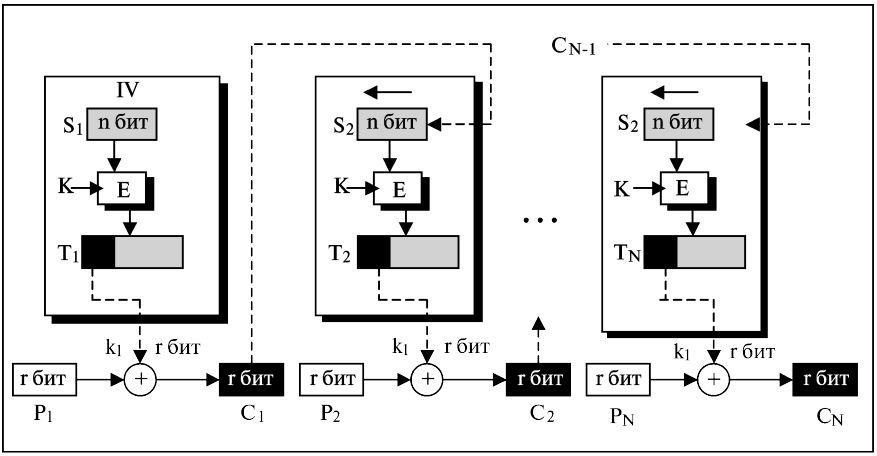
Режим кодированной обратной связи (CFB)

ECB и режимы CBC предназначены для шифрования и дешифрования блоков сообщений. Размер блока, *n*, определяется принятым шифром. Например, $n = 64$ для DES и $n = 128$ для AES. В некоторых ситуациях мы должны использовать DES или AES как секретные шифры, но исходный текст или размеры блока зашифрованного текста должны быть меньшими. Например, чтобы зашифровать и расшифровывать символы 8-битового ASCII, вы не захотели бы применить один из традиционных шифров, обсужденных в лекции 3, потому что они ненадежны. Решение состоит в том, чтобы применить DES или AES в **режиме кодированной обратной связи (CFB)**. В этом режиме размер блока, используемого в DES или AES, — n, но размер исходного текста или блока зашифрованного текста — r, где $r < n$.

Идея состоит в том, что DES или AES используются не для того, чтобы зашифровать исходный текст или расшифровывать зашифрованный текст, а для того, чтобы зашифровать или расшифровывать содержание регистра сдвига, S, размером *n*. Шифрование сделано с применением операции ИСКЛЮЧАЮЩЕЕ ИЛИ к *r*-битовому блоку исходного текста с *r*-битовым регистром сдвига. Дешифрование сделано с применением операции ИСКЛЮЧАЮЩЕЕ ИЛИ к *r*-битовому блоку зашифрованного текста с *r*-битовым регистром сдвига. Для каждого блока регистр сдвига S_i выполняет сдвиг регистра S_{i-1} (предыдущий регистр сдвига) на *r* бит влево, заполняя самые правые *r* битов с C_{i-1}. Тогда S_i зашифрован в T_i. Только самые правые *r* битов T*i* обрабатываются с помощью ИСКЛЮЧАЮЩЕГО ИЛИ с исходным текстом, из блока P_i получая C_i. Обратите внимание, что S_i, для первого блока — это IV, — не сдвигается.

Рисунок 8.4 показывает режим CFB для шифрования; дешифрование то же самое, но роли блоков исходного текста (P_i) и блоков зашифрованного текста (C*i*) меняются местами. Обратите внимание, что шифрование и дешифрование используют функцию шифрования основного блочного шифра (например, DES или AES).

В режиме CFB шифрование и дешифрование используют функцию шифрования основного блочного шифра.



E: Шифрование
D: Дешифрование
S_i: Регистр сдвига
P_i: Блок исходного текста
C_i: Блок зашифрованного текста
T_i: Временный регистр
K: Секретный ключ
IV: Начальный вектор (S_1)

Рис. 8.4. Шифрование в режиме кодированной обратной связи

Соотношение между исходным текстом и блоками зашифрованного текста показано ниже:

Шифрование: $C_i = P_i \oplus \text{SelectLeft}_r \{E_k [\text{ShiftLeft}_r (S_{i-1}) \mid \mathbf{C}_{i-1})]\}$
Дешифрование: $P_i = \mathbf{C}_i \oplus \text{SelectLeft}_r \{E_k [\text{ShiftLeft}_r (S_{i-1}) \mid C_{i-1})]\}$

где ShiftLeft — процедура, которая сдвигает содержание ее параметра на r бит влево (крайние левые *r*-биты отбрасываются). Оператор | показывает конкатенацию (последовательное соединение). SelectLeft-процедура выбирает только крайние левые *r*-битов параметра. Возможно доказать, что каждый блок исходного текста на стороне Алисы может быть точно восстановлен на стороне Боба. Это доказательство оставляем как упражнение.

Интересно, что в этом режиме не требуется дополнение блоков, потому что размер блоков, r, обычно выбирается так, чтобы удовлетворить размеру блока данных, который нужно зашифровать (например, символ). Интересное также другое — что система не должна ждать получения большого блока данных (64 бита или 128 битов) для того, чтобы начать шифрование. Процесс шифрования выполняется для маленького блока данных (таких как символ), Эти два преимущества приводят к двум недостаткам. CFB менее эффективен, чем CBC или ECB, потому что он применяет шифрование основным блочным шифром маленького блока размером *r*.

CFB как шифр потока

Хотя CFB — режим, предназначенный для того, чтобы использовать блочные шифры, такие как DES или AES, — он может быть шифром потока. Факти-

чески, это несинхронный шифр потока, в котором ключевой поток зависит от зашифрованного текста. Рисунок 8.5 показывает процесс дешифрования и шифрования и генератор ключей.

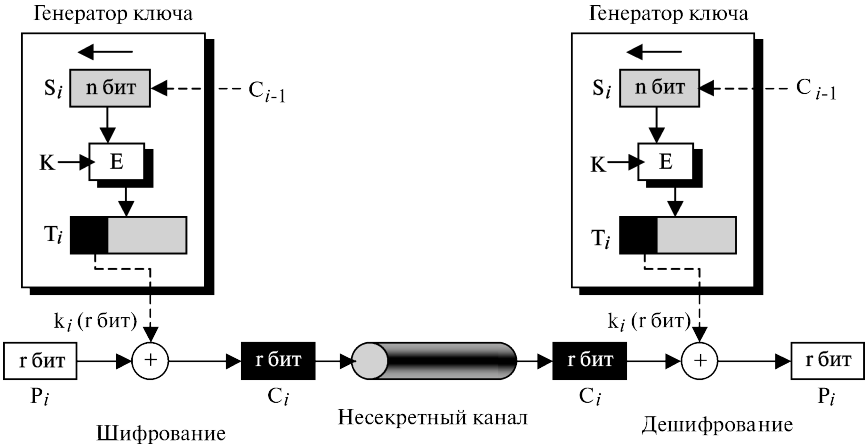


Рис. 8.5. Шифрование в режиме кодированной обратной связи как шифр потока

Рисунок 8.5 показывает, что основной шифр (DES или AES), ключ шифра (K) и предыдущий блок шифра (C_i) используются только для того, чтобы создать ключевые потоки (k_1, k_2,..., k_N).

Алгоритм

Алгоритм 8.3 дает процедуру для шифрования. Он вызывает несколько других процедур, детали которых оставляем как упражнения. Обратите внимание, что мы написали алгоритм так, чтобы показать характер режима потока (обработка в реальном масштабе времени). Алгоритм выполняется, пока блоки исходного текста не будут зашифрованы.

Алгоритм 8.3. Алгоритм шифрования кодированной обратной связи

```
CFB_Encryption (IV, K, r)
{
   i ← 1
   while (more blocks to encrypt)
   {
   input (Pi)
   (if i=1)
      S ← IV
   else
      {
      Temp ← shiftLeftr(S)
```

```
            S ← concatenate (Temp, Ci-1)
            }
        T ← Ek(s)
        k, ← selectLeftr (T)
        Ci ← Pi ⊕ ki
        output (Ci)
        i ← i + 1
        }
    }
```

Проблемы безопасности

В режиме CFB есть три первичных проблемы безопасности.

1. Точно так же как CBC, образцы на уровне блока не сохраняются. Одинаковые блоки исходного текста, принадлежащие одному и тому же сообщению, зашифровываются в различные блоки.
2. Одним и тем же ключом может быть зашифровано больше чем одно сообщение, но тогда значение IV должно быть изменено для каждого сообщения. Это означает, что Алиса должна использовать различные IV каждый раз, когда она передает сообщение.
3. Ева может добавить некоторый блок зашифрованного текста к концу потока зашифрованного текста.

Распространение ошибки

В CFB ошибка в единственном бите в зашифрованном тексте блока C_j в процессе передачи создает единственный бит ошибки (в той же самой позиции) в исходном тексте блока P_j. Однако большинство битов в следующих блоках исходного текста будут с ошибкой (с 50-процентной вероятностью), пока биты C_j все еще находятся в регистре сдвига. Вычисление числа затронутых блоков оставляем как упражнение. После того как регистр сдвига полностью регенерирован, система избавляется от ошибки.

Приложение

Режим работы CFB может использоваться, чтобы зашифровать блоки небольшого размера, такие как один символ или бит. Нет необходимости в дополнении, потому что размер блока исходного текста обычно устанавливается (8 для символа или 1 для бита).

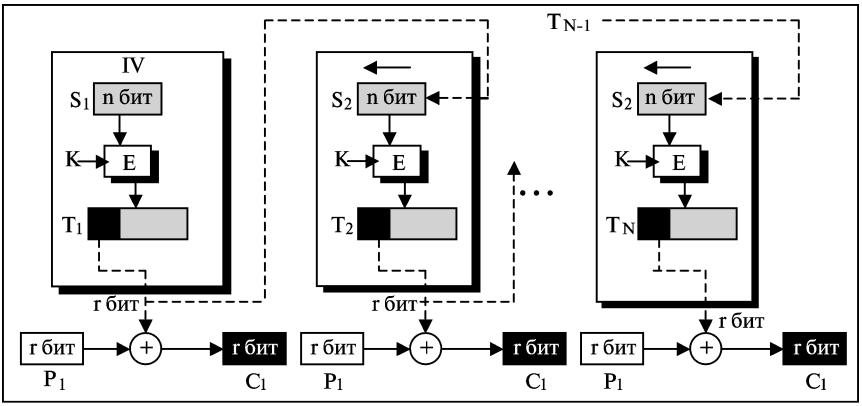
Специальный случай

Если блоки в тексте и в основном шифре — одного и того же размера (n = r), шифрование/дешифрование становится более простым, но построение диаграммы и алгоритм оставляем как упражнение.

Режим внешней обратной связи (OFB)

Режим внешней обратной связи (OFB — OUTPUT FEEDBACK) очень похож на режим CFB, с одной особенностью: каждый бит в зашифрованном тексте не-

зависим от предыдущего бита или битов. Это позволяет избежать распространения ошибок. Если при передаче возникает ошибка, она не затрагивает следующие биты. Подобно CFB, и передатчик и приемник используют алгоритм шифрования. Рисунок 8.6 показывает режим OFB.



Шифрование

E: Шифрование
D: Дешифрование
P_i: Блок i исходного текста
C_i: Блок i зашифрованного текста
K: Секретный ключ
IV: Начальный вектор (S1)
S_i: Регистр сдвига
T_i: Временный регистр

Рис. 8.6. Шифрование в режиме внешней обратной связи

OFB как шифр потока

OFB, так же как и CFB, может создать поточный шифр на базе основного шифра. Однако ключ потока не зависит от исходного текста или зашифрованного текста; значит, шифр потока — синхронный, как это обсуждалось в лекции 5. Рисунок 8.7 показывает шифрование и дешифрование, а также генератор ключей.

Алгоритм

Алгоритм 8.4 дает процедуру шифрования. Этот алгоритм последовательно вызывает другие процедуры, детали которых мы оставляем как упражнение. Заметим, что алгоритм написан так, чтобы показать режим потока (ситуация реального времени). Алгоритм работает, пока все блоки исходного текста не будут зашифрованы.

Проблемы безопасности

В режиме OFB имеются следующие две проблемы безопасности.

1. Точно так же как в режиме CFB, образцы на уровне блока не сохраняются. Одинаковые блоки исходного текста, принадлежащие тому же самому сообщению, зашифровываются в различные блоки.

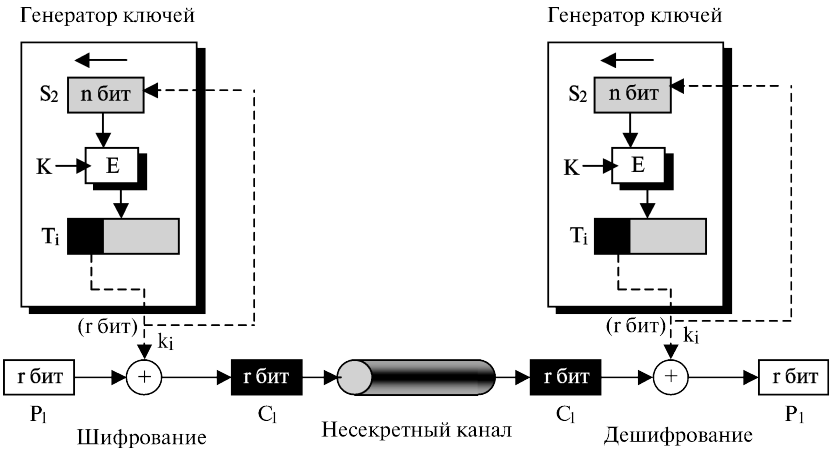


Рис. 8.7. Шифрование в режиме внешней обратной связи как шифрование потока

Алгоритм 8.4.

```
OFB_Encryption (IV, K, r)
{
    i ← 1
    while (more blocks to encrypt)
    {
    input (Pi)
    if (i=1)
        S ← IV
    else
        {
        Temp ← shiftLeftr(S)
        S ← concatenate (Temp, ki-1)
        }
    T ← Ek(s)
    k, ← selectLeftr (T)
    Ci ← Pi ⊕ ki
    output (Ci)
    i ← i + 1
    }
}
```

2. Любое изменение в зашифрованном тексте затрагивает исходный текст, зашифрованный в приемнике.

Распространение ошибки

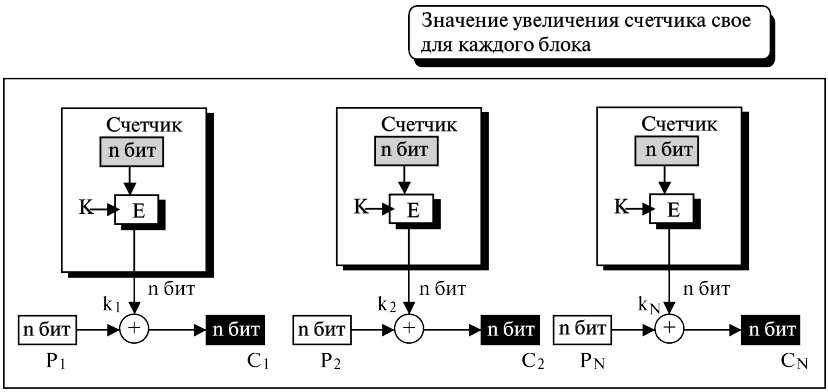
Единственная ошибка в зашифрованном тексте затрагивает только соответствующий бит в исходном тексте.

Специальный случай

Если блоки в тексте и основном шифре имеют один тот же размер *(n* = r), шифрование/дешифрование становится более простым, но мы оставляем диаграммы и алгоритм как упражнение.

Режим счетчика (CTR)

В режиме счетчика (CTR — Counter) нет информации обратной связи. Псевдослучайный ключевой поток достигается с помощью счетчика. Счетчик на n бит инициализируется в заранее определенное значение (IV) и увеличивается по основному и заранее определенному правилу (mod 2^n). Чтобы обеспечивать случайность, величина приращения может зависеть от номера блока. Исходный текст и блок зашифрованного текста имеют один и тот же размер блока, как и основной шифр (например, DES или AES). Блоки размера *n* исходного текста зашифрованы так, чтобы создать зашифрованный текст с блоком размера n. Рисунок 8.8 показывает шифрование в режиме счетчика.



Шифрование

E	Шифрование	D	Дешифрование
P_i	Блок *i* исходного текста	C_i	Блок *i* зашифрованного текста
K	Секретный ключ	IV	Начальный вектор
k_i	Ключ шифрования *i*		

Рис. 8.8. Шифрование в режиме счетчика

Отношение между исходным текстом и блоками зашифрованного текста показано ниже.

Шифрование: $C_i = P_i \oplus E_{ki}$ (Счетчик) **Дешифрование:** $P_i = C_i \oplus E_{ki}$ (Счетчик)

CTR использует функцию шифрования основного блочного шифра (E_K) и для шифрования, и для дешифрования. Достаточно легко доказать, что блок P*i* исходного текста может быть восстановлен из зашифрованного текста C*i*. Это мы оставляем как упражнение.

Мы можем сравнить режим CTR с режимами OFB и ECB. Подобно OFB, CTR создает ключевой поток, который независим от предыдущего блока зашифрованного текста, но CTR не использует информацию обратной связи. Так же как ECB, CTR создает *n*-битовый зашифрованный текст, блоки которого независимы друг от друга — они зависят только от значений счетчика. Отрицательной стороной этого свойства является то, что режим CTR, подобно режиму ECB, не может применяться для обработки в реальном масштабе времени. Алгоритм шифрования ждет перед шифрованием законченный n-разрядный блок данных. Положительная сторона этого свойства: режим, подобно режиму ECB, может использоваться, чтобы зашифровать и расшифровывать файлы произвольного доступа, и значение счетчика может быть связано номером записи в файле.

CTR как шифр потока

CFB, OFB и CTR — фактически шифры потока. Рисунок 8.9 показывает шифрование и дешифрование *i*-того блока данных.

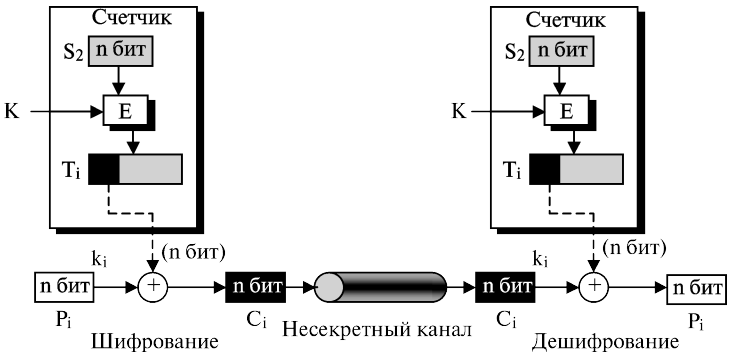


Рис. 8.9. Шифрование в режиме счетчика как шифр потока

Алгоритм

Алгоритм 8.5 содержит процедуру в псевдокоде для шифрования; алгоритм для дешифрования оставляем как упражнение. Здесь значение приращения зависит от номера блока. Другими словами, значения счетчика — IV, IV + 1, IV + 3, IV + 6, и так далее. Предполагается, что все *N*-блоки исходного текста готовы до начала шифрования, но алгоритм может быть переписан, чтобы избежать этого предположения.

Безопасность

Проблемы безопасности для режима CTR те же самые, что и для режима OFB.

Распространение ошибки

Единственная ошибка в зашифрованном тексте затрагивает только соответствующий бит в исходном тексте.

Алгоритм 8.5. Алгоритм шифрования CTR

```
CTR_Encryption (IV, K, Plaintext blocks)
{
       Counter <- IV
       for(i= 1 to N)
       {
          Counter <- (Counter + i -1)mod2^N
          k_i <- E_K (Counter)
          C_i <- P_i ⊕ k_i
       }
   return Ciphertext blocks
}
```

Сравнение различных режимов

Таблица 8.1 сравнивает пять различных режимов работы, рассмотренных в этой лекции.

Таблица 8.1. Итоги режимов работы

Режим работы	Описание	Тип результата	Размер блока
ECB	Каждый *n-битовый блок* шифруется независимо тем же самым ключом	Блочный шифр	*n*
CBC	То же самое, что и в ECB, но каждый блок сначала складывается (ИСКЛЮЧАЮЩЕЕ ИЛИ) с предыдущим зашифрованным текстом блочный шифр	Блочный шифр	*n*
CFB	Каждый r-битовый блок складывается (ИСКЛЮЧАЮЩЕЕ ИЛИ) с r-битовым ключом, который является частью предыдущего текста шифра	Шифр потока	$r \le n$
OFB	То же самое, что и в CFB, но регистр сдвига модифицирован с помощью предыдущего r-битового ключа	Шифр потока	$r \le n$
CTR	То же самое, как в OFB, но счетчик используется вместо регистра сдвига	Шифр потока	*n*

8.2. Использование шифров потока

Хотя эти пять режимов работы допускают использование блочных шифров для шифрования сообщений или файлов в больших модулях (ECB, CBC и CTR) и

маленьких модулях (OFB и OFB), иногда необходимо передать поток для того, чтобы зашифровать маленькие единицы информации — символы или биты. Шифры потока более эффективны для обработки в реальном масштабе времени. Некоторые шифры потока использовались в различных протоколах в течение прошлых нескольких десятилетий. Мы рассмотрим только два: RC4 и A5/1.

RC4

RC4 — потоковый шифр, который был разработан в 1984 г. Рональдом Ривестом. RC4 используется во многих системах передачи данных и протоколах организации сети, например, SSL/TLS (см. лекцию 17) и IEEE 802.11 (беспроводный стандарт LAN).

RC4 — байт-ориентированный шифр потока, в котором байт (8 битов) исходного текста складывается (ИСКЛЮЧАЮЩЕЕ ИЛИ) с байтом ключа, чтобы получить байт зашифрованного текста. Ключ засекречивания, из которого сгенерированы однобайтовые ключи в потоке ключей, может содержать от 1 до 256 байтов.

Матрица состояний

RC4 базируется на понятии матрицы состояний. В каждый момент матрица состояний 256 байтов активизируется, из нее случайным образом выбирается один байт, чтобы служить ключом для шифрования

Идея может быть показана в виде массива байтов:

```
S [0] S [l] S [2] ... S [255]
```

Заметим, что индексы диапазона элементов — между 0 и 255. Содержание каждого элемента — байт (8 битов), который может интерпретироваться как целое число от 0 до 255.

Идея

Рисунок 8.10 показывает идею RC4. Первые два блока выполняются только один раз (инициализация); перестановки для того, чтобы создавать ключ потока, повторяются, пока есть байты исходного текста, предназначенные для шифрования.

Инициализация

Инициализация делается в два шага.

1. На первом шаге матрица состояний инициализируется для значений 0, 1,..., 255. Создается также массив ключей K [0], K [1] ..., K [255]. Если ключ засекречивания имеет точно 256 байтов, байты копируются в массив K; иначе — байты повторяются, пока не заполнится массив K.

```
for (i = 0 to 255)
{
   S[i] <- i
   K[i] <- Key [i mod Key Length]
}
```

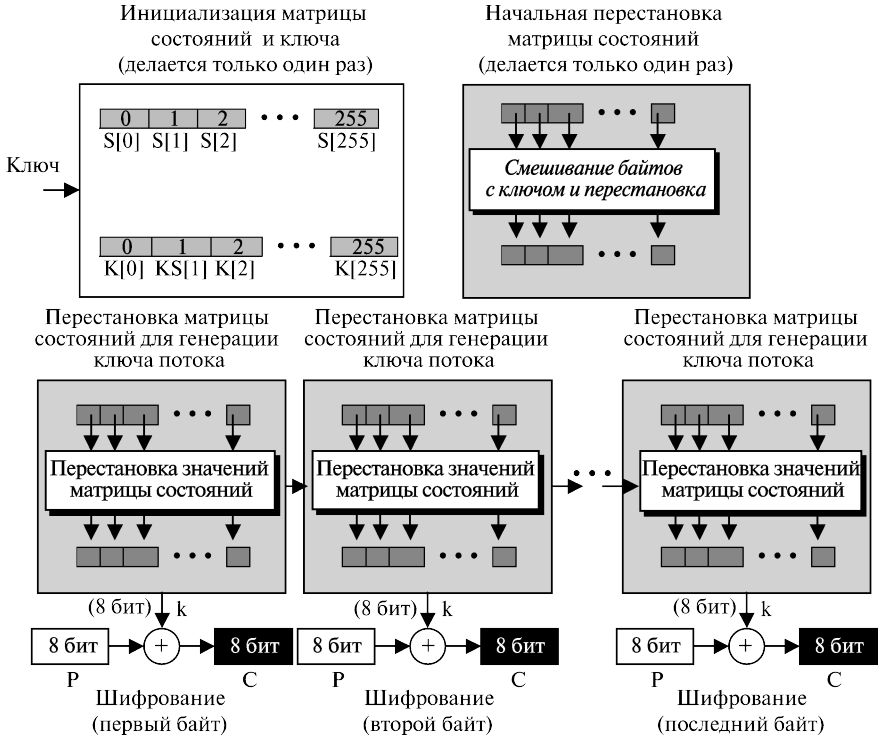


Рис. 8.10. Идея шифра потока RC4

2. На втором шаге инициализированная матрица проходит перестановку (скрэмблирование элементов), основанную на значении байтов в K[*i*]. Ключевой байт используется только на этом шаге, чтобы определить, какие элементы должны быть заменены. После этого шага байты матрицы полностью перетасованы.

```
j <- 0
for (i = 0 to 255)
{
   j <- (j + S[i] + K[i]) mod 256
   замена (S[i], S[j])
}
```

Генерация ключевого потока. Ключи *k* в ключевом потоке генерируются один другим. Сначала элементы матрицы состояний переставляются на основе значений своих элементов и значений двух индивидуальных переменных *i* и *j*. Затем значения двух элементов матрицы состояний в позициях *i* и *j* используются, чтобы определить индекс элемента матрицы состояний, который служит как

ключ *k*. Следующий код повторяется для каждого байта исходного текста, чтобы создать новый ключевой элемент в ключевом потоке. Переменные *i* и *j* инициализируются в 0 прежде, чем будет проведена первая итерация, но значение копируется от одной итерации к следующей.

```
i <- (i +1) mod 256
j <- (j +S[i]mod256
замена (S [i] , S[j]
k <- S [(S[I] + S[j]) mod 256]
```

Шифрование или дешифрование. После того как *k* были создан, байт исходного текста зашифровывается с помощью *k*, чтобы создать байт зашифрованного текста. Дешифрование представляет собой обратный процесс.

Алгоритм

Алгоритм 8.6 показывает процедуру, написанную на псевдокоде, для RC4.

Алгоритм 8.6. Алгоритм шифрования для RC4

```
RC4_Encryption (K)
{
   // Создание начальной матрицы состояний и ключевых байтов
   for (i = 0 to 255)
   {
      S[i] <- i
      K[i] <- Key [i mod Key Length]
   }
   // Перестановка байтов матрицы состояний на основе значений
   // байта ключа
   j <- 0
   for (i = 0 to 255)
   {
      j <- (j+ S[i] + K[i] mod 256
      замена (S[i] , S[j])
   //Непрерывная перестановка байтов, генерация ключей и шифрование
   i <- 0
   j <- 0
   while (пока есть байты для шифрования)
   {
      {i <- (i + 1) mod 256
      j <- (j +S[i])mod256
      замена (S [i] ,S[j])
      k <- S [(S[i] +S[j])mod256]
      // Ключ готов, шифрование
      input P
      C <- P ⊕ k
      output C
   }
}
```

Пример 8.5

Чтобы показать случайность ключа потока, мы используем ключ засекречивания со всеми нулевыми байтами. Ключевой поток для 20 значений A: (222, 24, 137, 65, 163, 55, 93, 58, 138, 6, 30, 103, 87, 110, 146, 109, 199, 26, 127, 163).

Пример 8.6

Повторим пример 8.5, но пусть ключ засекречивания будет пять байтов (15, 202, 33, 6, 8). Ключевой поток — (248, 184, 102, 54, 212, 237, 186, 133, 51, 238, 108, 106, 103, 214, 39, 242, 30, 34, 144, 49). Снова случайность в ключевом потоке очевидна.

Проблемы безопасности

Известно, что шифр безопасен, если размер ключа — по крайней мере, 128 битов (16 байтов). Это подтверждается сообщениями о некоторых атаках для малых размеров ключей (меньше, чем 5 байтов). Протоколы, которые сегодня использует RC4, устанавливают размеры ключей, которые делают RC4 безопасным. Однако, как и для многих других шифров, рекомендуется, чтобы для различных сеансов применялись различные ключи. Это препятствует Еве использовать дифференциальный криптоанализ шифра.

A5/1

В этом разделе мы вводим шифр потока, который применяет линейный регистр сдвига (см. лекцию 5, LFSR Linear Feed Back Shift Register), чтобы создать битовый поток: A5/1. A5/1 (член семейства шифров A5) используется в **Глобальной Системе Мобильной связи** (GSM). Телефонная связь в GSM осуществляется как последовательность кадров на 228 битов, при этом каждый кадр длится 4,6 миллисекунды. A5/1 создает поток бит, исходя из ключа на 64 бита. Разрядные потоки собраны в буфере по 228 битов, чтобы складывать их по модулю два с кадром на 228 битов, как показано на рис. 8.11.

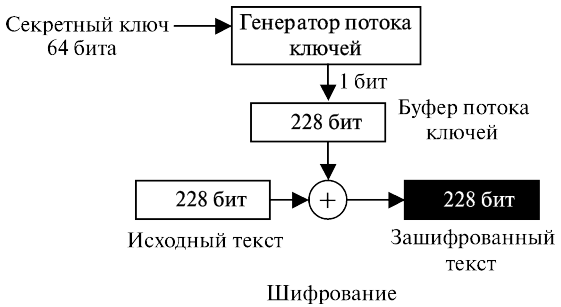


Рис. 8.11. Общий вид A5/1

Генератор ключей

A5/1 используются три LFSR на 19,22,23 бита. LFSR , содержащие биты символов и синхронизации, показаны на рис 8.12.

Заметим, что три «черных ящика» используются для *мажоритарной функции*

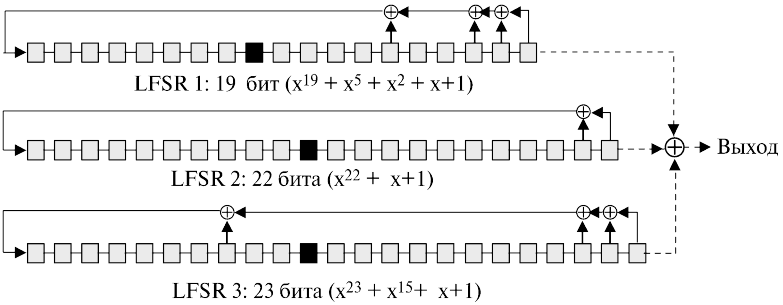


Рис. 8.12. Три линейных регистра сдвига для A5/1

Однобитовый выход обеспечивает тактовыми импульсами буфер на 228 битов, который используется для шифрования (или дешифрования).

Инициализация. Инициализация выполняется для каждого кадра шифрования (или дешифрования). Она применяет ключ засекречивания на 64 бита и 22 бита соответствующего номера кадра. Следующие шаги:

1. Сначала все биты в трех линейных регистрах сдвига устанавливаются в 0.
2. Второй: ключ на 64 бита смешивается со значением регистра согласно следующему коду. Каждый линейный регистр смещается на один шаг (*синхронизация*).

```
For (i = 0 to 63)
   {
      Сложение по модулю 2 K[i] с крайними левыми битами всех трех
      регистров.
      Синхронизация всех трех линейных регистров сдвига
   {
```

3. Повторить предыдущий процесс, но использовать 22-битовый кадр.

```
for (i = 0 to 22)
   {
      Сложение по модулю 2 номера кадра [i] с крайними левыми
      битами всех трех регистров.
      Синхронизация всех трех линейных регистров сдвига
   {
```

4. В течение 100 циклов синхронизируется весь генератор. При этом применена мажоритарная функция (см. следующий абзац), для того чтобы определить, какой линейный регистр сдвига должен быть синхронизирован. Обратите внимание: иногда синхронизация здесь означает, что два, а то и все три линейных регистра сдвига проходят процесс смещения.

```
for (i = 0 to 99)
   {
      Синхронизация всего генератора, на основе мажоритарной функции
   }
```

Мажоритарная функция. Значение мажоритарной функции (majority) с параметрами (b_1, b_2, b_3) равно 1, если значение большинства битов — 1; если это — 0, то ее значение — 0. Например, majority (1, 0, 1) = 1, но majority (0, 0, 1) = 0. Значение мажоритарной функции определяется перед поступлением тактового импульса; три входных бита названы *синхронизирующими битами*: если самый правый бит равен нулю, это — биты линейных регистров LFSR1 [10], LFSR2 [11] и LFSR3 [11]. Обратите внимание, что в литературе эти биты 8, 10 и 10 отсчитывают слева (как это показано на рис. 8.12). Мы будем рассматривать 10, 11 и 11, считая справа. Это соглашение соответствует месту бита в характеристическом полиноме.

Ключевые биты потока. Генератор ключей создает ключевой поток в один бит при каждом тактовом импульсе. Прежде чем ключ будет создан, вычисляется мажоритарная функция. Затем каждый линейный регистр сдвига синхронизируется, если его бит синхронизации соответствует результату мажоритарной функции; иначе — он не синхронизируется.

Пример 8.7

В некоторый момент времени биты синхронизации — 1, 0 и 1. Какой должен быть LFSR?

Решение

Результат Majority (1, 0, 1) = 1. LFSR1 и LAFS3 сдвигаются, а LFSR2 — нет.

Шифрование/дешифрование

Разрядные потоки, созданные генератором ключей, записываются в буфер, чтобы потом сформировать ключ на 228 битов, который затем складывает по модулю два с кадром исходного текста, чтобы создать кадр зашифрованного текста. В один момент времени делается шифрование/дешифрование одного кадра.

Проблемы безопасности

Хотя GSM продолжает использовать A5/1, уже были зарегистрированы несколько атак на GSM. Две из них были упомянуты. В 2000 году Алекс Бирюков, Дэвид Вагнер и Эди Шамир показали, что атака в реальном масштабе времени находит ключ за несколько минут на основе известных малых исходных текстов, но это требует этапа предварительной обработки с 2^{48} шагами. В 2003 Экдахи и Джонсон (Ekdahi и Johannson) опубликовали атаку, которая вскрывала A5/1 за несколько минут, применяя анализ исходного текста в течение 2-5 минут. Имея в виду некоторые новые атаки GSM, возможно, в будущем нужно будет сменить или укрепить A5/1.

8.3. Другие проблемы

Шифрование, которое использует блоки с симметричными ключами или шифры потока, требует обсуждения других проблем.

Управление ключами

Алиса и Боб должны совместно использовать секретный ключ, чтобы иметь надежную связь с использованием шифра с симметричным ключом. Если есть n объектов в сообществе, каждый из которых хочет связаться с $n - 1$ другим объектом, то тогда необходимы $n(n - 1)$ ключей засекречивания. Однако при шифровании симметричными ключами один ключ может применяться в обоих направлениях: от Алисы к Бобу и от Боба к Алисе. Это означает, что нужно только $n(n - 1)/2$ ключей. Если n — приблизительно миллион, то должны быть выданы почти пятьсот миллионов ключей. Поскольку это нереально, то были найдены несколько других решений. Первое: каждый раз, когда Алиса и Боб хотят связаться, они могут создать между собой сеансовый (временный) ключ. Второе: могут быть установлены один или более центров распределения ключей, чтобы распределять сеансовые ключи для объектов. Все эти проблемы — часть теории управления ключами, которая будет обсуждаться в лекции 15 после того, как будут рассмотрены необходимые инструментальные средства.

Управление ключами будет обсуждаться в лекции 15.

Генерирование ключей

Другая проблема в шифровании симметричными ключами — безопасная генерация ключа. Различные шифры с симметричным ключом нуждаются в ключах различных размеров. Выбор ключа должен базироваться на гарантии безопасности систематического метода для избежания утечки. Если Алиса и Боб генерируют сеансовые ключи между собой, они должны выбрать ключ случайным образом, так, чтобы Ева не могла предвидеть, каков будет следующий ключ. Если ключи должен распределять ключевой центр, они должны иметь случайный характер, чтобы Ева не могла получить ключ, назначенный Алисе и Бобу, из ключа, назначенного Джону и Еве. Это подразумевает, что нужен генератор случайных (или псевдослучайных) чисел. Поскольку обсуждение генератора случайных чисел включает некоторые темы, которые еще не были рассмотрены, изучение *генераторов случайных чисел* представлено в приложении К.

Генераторы случайных чисел будут обсуждаться в приложении К.

8.4. Рекомендованная литература

Нижеследующие книги и сайты дают более детальную информацию о предметах, рассмотренных в этой лекции. Пункты, приведенные в квадратных скобках, содержатся в списке в конце книги.

Книги

[Sch99], [Sta06], [PHS03], [Sti06], [MOV97] и [KPS02] рассматривают режимы работы. [Vau06] и [Sta06] дают полные сведения о шифрах потока.

Сайты

Нижеследующие сайты содержат больше информации о темах, обсужденных в этой лекции.

http: // en.wikipedia.org/wiki/Block_cipher_modes_of_operation
http://www.itl.nist.gov/fipspubs/fip81.htm
en.wikipedia.org/wiki/A5/1
en.wikipedia.org/wiki/RC4

8.5. Итоги

- В реальных приложениях зашифрованный текст имеет переменные размеры и обычно намного большие, чем размер блока, определенный для современных блочных шифров. Режимы работы были изобретены, чтобы зашифровать текст любого размера, который обслуживается современными блочными шифрами. В этой лекции были рассмотрены пять режимов работы.
- Самый простой режим работы называется режимом электронной кодовой книги (ECB — ELECTRONIC CODEBOOK). Исходный текст разделен на *N* блоков. Размер блока — *n* бит. Каждый блок использует для шифрования и дешифрования один и тот же ключ.
- В режиме сцепления блоков шифротекста (CBC — Cipher Block Chaining) каждый блок исходного текста, прежде чем зашифровывать, складывают по модулю два с предыдущим блоком зашифрованного текста. Когда блок зашифрован, его передают, но его копия сохраняется в памяти, чтобы ее можно было использовать для шифрования следующего блока. Передатчик и приемник согласуют заранее заданный вектор инициализации (IV), чтобы складывать его по модулю два с первым блоком зашифрованного текста.
- Чтобы шифровать маленькие модули данных в реальном масштабе времени, применяется режим кодированной обратной связи (CFB — CIPHER FEEDBACK), CFB применяет стандартные блочные шифры, такие как DES или AES, регистр сдвига, но использует операцию сложения по модулю два, чтобы зашифровать или расшифровывать модули данных. Режим CFB применяет блочные шифры, но в результате — это шифр потока, потому что каждый модуль данных зашифровывается своим ключом.
- Режим внешней обратной связи (OFB) очень похож на режим CFB, с одной разницей — каждый бит в зашифрованном тексте независим от предыдущего бита или битов. Это позволяет избежать распространения ошибки. Вместо того чтобы использовать предыдущий блок зашифрованного текста, OFB берет предыдущий ключ как информацию обратной связи.
- В режиме счетчика (CTR) нет информации обратной связи. Псевдослучайность в потоке достигается с помощью счетчика. Счетчик на *n битов* инициализируется установкой заранее заданного значения (IV) и увеличивается по заранее заданному правилу.

- Чтобы зашифровать маленькие единицы данных, такие как символы или биты, были разработаны и испытываются несколько шифров потока. Эти шифры потока более эффективны для обработки в реальном масштабе времени. В этой лекции рассматривались только два шифра потока — RC4 и A5/l.
- RC4 — шифр потока, ориентированный на байт, в котором байт (8 битов) исходного текста надо сложить по модулю два с байтом ключа, чтобы создать байт зашифрованного текста. Секретный ключ, из которого генерируются однобайтовые ключи в ключевом потоке, может содержать от 1 до 256 байтов. Ключевой генератор потока базируется на перестановке 256 байтов.
- A5/1 — шифр потока, используемый для мобильной телефонной связи. A5/1 создает поток бит из ключа на 64 бита, используя три линейных регистра сдвига.

8.6. Набор для практики

Обзорные вопросы

1. Объясните, почему необходимы режимы работы, если для шифровки используются современные блочные шифры.
2. Перечислите пять режимов работы, рассмотренных в этой лекции.
3. Определите ECB (ELECTRONIC CODEBOOK) и перечислите его преимущества и недостатки.
4. Определите CBC (CIPHER BLOCK CHAINING) и перечислите ее преимущества и недостатки.
5. Определите CFB (CIPHER FEEDBACK) и перечислите его преимущества и недостатки
6. Определите OFB (OUTPUT FEEDBACK) и перечислите его преимущества и недостатки.
7. Определите CTR и перечислите его преимущества и недостатки.
8. Разделите пять режимов работы на две группы: те, которые используют функции шифрования и дешифрования, — основные шифры (например, DES или AES), и те, которые используют только функцию шифрования.
9. Разделите пять режимов работы на две группы: те, которые требуют дополнение текста, и те, которые не требуют этого.
10. Разделите пять режимов работы на две группы: те, которые используют один и тот же ключ для шифрования всех блоков, и те, которые используют ключевой поток для шифровки блоков.
11. Объясните основные различия между RC4 и A5/1. Какой из них использует линейный регистр сдвига?
12. Каков размер модуля данных в RC4? Каков размер модуля данных в A5/1?
13. Перечислите режимы работы, которые могут быть ускорены параллельной обработкой.
14. Перечислите режимы работы, которые могут использоваться для шифровки файлов произвольного доступа.

Упражнения

1. Покажите, почему режим CFB создает несинхронный шифр потока, а режим OFB создает синхронный.
2. Сколько блоков затрагивает единственный бит ошибки в передаче в режиме CFB?
3. В режиме ECB бит 17 в зашифрованном тексте блока 8 разрушен в течение передачи. Найдите возможные разрушенные биты в исходном тексте.
4. В режиме CBC биты 17 и 18 в зашифрованном тексте блока 9 в процессе передачи были разрушены. Найдите возможные разрушенные биты в исходном тексте.
5. В режиме CFB биты 3-6 в зашифрованном тексте блока 11 разрушены *(r* = 8). Найдите возможные разрушенные биты в исходном тексте.
6. В режиме CTR блоки 3 и 4 полностью разрушены. Найдите возможные разрушенные биты в исходном тексте.
7. В режиме OFB полный зашифрованный текст блока 11 разрушен (r = 8). Найдите возможные разрушенные биты в исходном тексте.
8. Докажите, что исходный текст, используемый Алисой, может быть восстановлен Бобом в режиме CFB.
9. Докажите, что исходный текст, используемый Алисой, может быть восстановлен Бобом в режиме OFB.
10. Докажите, что исходный текст, используемый Алисой, может быть восстановлен Бобом в режиме CTR.
11. Покажите диаграмму для шифрования и дешифрования в режиме CFB, когда $r = n$.
12. Покажите диаграмму для шифрования и дешифрования в режиме OFB, когда r = n.
13. Покажите процесс, используемый для алгоритма дешифрования в режиме ECB, если применяется захват зашифрованного текста (CTS) .
14. Покажите диаграммы шифрования и дешифрования для режима ECB (только последние два блока), когда используется захват зашифрованного текста (CTS).
15. Покажите процесс, используемый для алгоритма дешифрования в режиме CBC, если применяется захват зашифрованного текста (CTS).
16. Покажите шифрование и диаграмму дешифрования для режима CBC (только последние два блока), когда используется захват зашифрованного текста (CTS).
17. Объясните, почему нет потребности в захвате зашифрованного текста в режимах CFB, OFB и CTR (CIPHER FEEDBACK, OUTPUT FEEDBACK).
18. Покажите эффект распространения ошибки, когда ECB (ELECTRONIC CODEBOOK) использует методику CTS.
19. Покажите эффект распространения ошибки, когда CBC использует методику CTS.
20. Режим *Формирование цепочки блоков* является вариантом, в котором все предыдущие блоки зашифрованного текста перед шифрованием склады-

ваются по модулю два с текущим исходным текстом. Создайте рисунок-диаграмму, которая показывает шифрование и дешифрование.

21. Режим размножения *Цепочка блоков шифротекста* (PCBC) является вариантом CBC, в котором перед шифрованием предыдущий блок исходного текста и предыдущий блок зашифрованного текста складывается по модулю два с текущим блоком исходного текста. Нарисуйте диаграмму, которая показывает шифрование и дешифрование.
22. *Р*ежим *Цепочка блоков шифротекста с контрольной суммой* (CBCC) является вариантом CBC, в котором все предыдущие блоки исходного текста перед шифрованием складываются по модулю два с текущим блоком исходного текста. Нарисуйте диаграмму, чтобы показать шифрование и дешифрование и проиллюстрировать процедуру.
23. В RC4 покажите первые 20 элементов ключевого потока, если ключ засекречивания — 7 байтов со значениями 1, 2, 3, 4, 5, 6 и 7. Вы можете при желании написать маленькую программу.
24. В RC4 найдите значение для ключа засекречивания, который не изменяет матрицу состояний после первого и второго шагов инициализации.
25. Алиса обменивается сообщениями с Бобом, используя в RC4 для засекречивания16-байтовый ключ засекречивания. Ключ засекречивания изменяется каждый раз, используя рекурсивное определение $K = (K_{i-1}+K_{i-1}) \bmod 2^{128}$. Покажите, сколькими сообщениями они могут обменяться перед тем, как текст начнет повторяться.
26. В A5/1 найдите максимальный период каждого линейного регистра сдвига.
27. В A5/1 найдите значение следующих функций. В каждом случае показать, сколько синхронизируется линейных регистров сдвига.
    a. Majority (1, 0, 0)
    b. Majority (0, 1, 1)
    c. Majority (0, 0, 0)
    d. Majority (1, 1, 1)
28. В A5/1 найдите выражение для мажоритарной функции.
29. Напишите алгоритм дешифрования в псевдокоде для режима ECB.
30. Напишите алгоритм дешифрования в псевдокоде для режима CBC.
31. Напишите псевдокод алгоритма дешифрования для режима CFB.
32. Напишите алгоритм дешифрования в псевдокоде для режима OFB.
33. Напишите алгоритм дешифрования в псевдокоде для режима CTR.
34. Напишите алгоритм для *shiftleft*-процедуры, используемой в алгоритме 8.4.
35. Напишите алгоритм для *selectleft*-процедуры, используемой в алгоритме 8.4.
36. Напишите алгоритм для процедуры *конкатенации*, используемой в алгоритме 8.4.

Часть 2. Шифрование с асимметричными ключами

В лекции 1 мы видели, что криптография обеспечивает три метода: шифрование с симметричными ключами, шифрование с асимметричными ключами и хэширование. Часть 2 посвящена шифрам с ассиметричными ключами. Лекция 9 рассматривает математические основы, необходимые для понимания лекций в этой части и остальной части книги. Лекция 10 исследует современные шифры с ассиметричными ключами.

Лекция 9: Математика криптографии. Часть III

Лекция 9 рассматривает некоторые математические понятия, необходимые, чтобы понять следующие лекции. Мы поговорим о простых числах и их приложении в криптографии. Будут введены алгоритмы для испытания простоты чисел и оценка их эффективности. Другие темы включают разложение на множители, китайскую теорему об остатках и квадратичные сравнения, возведение в степень и логарифмы по модулю, чтобы проложить путь к рассмотрению криптографических систем с открытым ключом в лекции 10.

Лекция 10: Асимметрично-ключевая криптография

Лекция 10 рассматривает шифрование с асимметричным ключом (открытый ключ доступа). Он вводит несколько криптографических систем, таких как алгоритмы RSA, Rabin, ElGamal и криптосистема на основе метода эллиптических кривых (ECC — Elliptic Curve Cryptosystem). Здесь упоминается большинство видов атак для каждой системы и представляются рекомендации как предотвратить эти атаки.

Лекция 9. Математика криптографии. Часть III. Простые числа и уравнения сравнения

Цели и содержание

Эта лекция имеет несколько целей.

- Ввести простые числа и их приложения в криптографии.
- Обсудить некоторые алгоритмы проверки простоты чисел и их эффективность.
- Обсудить алгоритмы разложения на множители и их приложения в криптографии.
- Описать китайскую теорему об остатках и ее приложения.
- Ввести квадратичное сравнение.
- Ввести возведение в степень по модулю и логарифмы.

Асимметрично-ключевая криптография, которую мы обсудим в лекции 10, базируется на некоторых положениях теории чисел, включая теории, связанные с простыми числами, разложением на множители составных объектов в простые числа, модульном возведение в степень и логарифмах, квадратичных вычетах и ки-

тайской теореме об остатках. Эти проблемы будут рассмотрены здесь, в лекции 9, чтобы упростить понимание лекции 10.

9.1. Простые числа

Асимметрично-ключевая криптография широко использует простые числа. Тема простых чисел — большая часть любой книги по теории чисел. Эта лекция обсуждает только несколько понятий и фактов, чтобы открыть путь к лекции 10.

Определение

Положительные целые числа могут быть разделены на три группы: число 1, простые числа и составные объекты, как это показано на рис. 9.1.

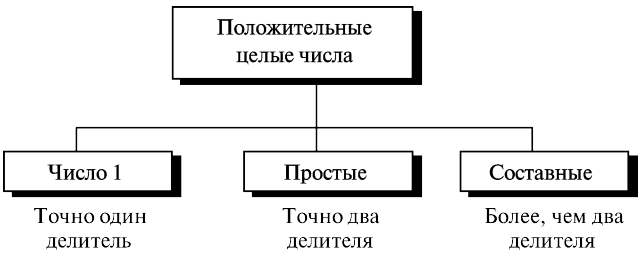


Рис. 9.1. Три группы положительных целых чисел

Положительное целое число — **простое число** *тогда и только тогда*, когда оно точно делимо без остатка на два целых числа — на 1 и на само себя. **Составной объект** — положительное целое число больше с чем двумя делителями.

Простое число делимо без остатка только на себя и 1.

Пример 9.1

Какое наименьшее простое число?

Решение

Наименьшее простое число — 2, оно делится без остатка на 2 (само на себя) и 1. Обратите внимание, что целое число 1 — не простое число согласно определению, потому что простое число должно быть делимо без остатка двумя различными целыми числами, не больше и не меньше. Целое число 1 делимо без остатка только на себя; поэтому 1 — это не простое число.

Пример 9.2

Перечислите простые числа, меньшие, чем 10.

Решение

Есть четыре простых числа меньше чем 10: 2, 3, 5 и 7. Интересно, что процент простых чисел в диапазоне 1-10 — 40%. С увеличением диапазона процент уменьшается.

Взаимно простые числа

Два положительных целых числа a и *b* являются **взаимно простыми (coprime)**, если НОД (a, *b*) = 1, потому что число 1 является взаимно простым с любым целым числом. Если p — простое число, тогда все числа от 1 до *p*–1 являются взаимно простыми к p. В лекции 2 мы обсуждали множество Z_n^*, чьи элементы — все числа, взаимно простые к *n*. Множество Z_p^* является тем же самым, за исключением того, что модуль *(p)* — простое число.

Количество простых чисел

После того как понятие простых чисел было определено, естественно возникает вопрос: число простых чисел конечно или бесконечно? Возьмем число n. Сколько есть простых чисел меньших, чем это число, или равных n?

Число простых чисел

Число простых чисел бесконечно. Приведем нестрогое доказательство: предположим, что множество простых чисел конечно (ограничено), и пусть *p* — наибольшее простое число. Перемножим все простые числа, входящие в это множество, и получим результат P = 2 × 3 × ... × *p*. Целое число (P + 1) не может иметь простого делителя $q \leqslant p$ (*p* — *наибольшее простое число*). Тогда этот делитель должен быть одним из множителей, входящих в P. Это значит, что *q* делит P. Если *q* также делит (P + 1), то *q* делит (P + 1) – P = 1. Единственное число, которое делит 1, — это сама 1, которая не является простым числом. Поэтому *q* должно быть большим, чем *p*, и ряд простых чисел не исчерпывается принятым конечным множеством.

Число простых чисел бесконечно.

Пример 9.3

Как тривиальный пример, предположим, что единственные простые числа находятся в множестве {2, 3, 5, 7, 11, 13, 17}. Здесь P = 510510 и P + 1 = 510511. Однако 510511 состоит из следующих простых чисел 510511 = 19 × 97 × 277; ни одно из этих простых чисел не было в первоначальном списке. Эти три простых числа больше, чем 17.

Число простых чисел, меньших n

Чтобы рассмотреть вторую возможность, введем функцию π *(n)*, которая определяет число простых чисел, меньших или равных *n*. Ниже показаны значения этой функции для различного π (n).

π (1).= 0 π (2).=1 π (3).= 2 π (n).= 4 π (n).= 8 π (50).=15 π (100).=25

Но если *n* является очень большим, как мы можем вычислить π (*n*)? Для ответа можно использовать только приближение, которое показано ниже:

$$[n / (\ln n)] < \pi(n). < [n / (\ln n - 1.08366)]$$

Гаусс обнаружил верхний предел; Лагранж обнаружил нижний предел.

Пример 9.4

Найдите количество простых чисел, меньших, чем 1 000 000.

Решение

Приближение дает диапазон от 72 383 до 78 543. Фактическое число простых чисел — 78 498.

Проверка на простое число

Следующий вопрос, который приходит на ум: как мы можем определить для данного числа n, является ли оно простым числом? Мы должны проверить, делимо ли без остатка это число всеми простыми числами, меньшими, чем $\sqrt{n}$. Мы знаем, что этот метод неэффективен, но он хорош для начала.

Пример 9.5

Действительно ли 97 — простое число?

Решение

Наибольшее ближайшее целое число — $\sqrt{97} = 9$. Простые числа меньше, чем 9 — 2, 3, 5 и 7. Проверим, делимо ли без остатка 97 любым из этих номеров. Ответ: не делимо, так что 97 — простое число.

Пример 9.6

Действительно ли 301 — простое число?

Решение

Наибольшее ближайшее целое число $\sqrt{301} = 17$. Мы должны проверить 2, 3, 5, 7, 11, 13 и 17. Числа 2, 3 и 5 не делят 301, но 7 — делит. Поэтому 301 — не простое число.

Решето Эратосфена

Греческий математик Эратосфен изобрел метод, как найти все простые числа, меньшие, чем n.

Метод назван **решетом Эратосфена.** Предположим, что мы хотим найти все числа, меньшие, чем 100. Мы записываем все числа между 2 и 100. Поскольку $\sqrt{100} = 10$, мы должны видеть, делим ли без остатка любое число меньше чем 100 на числа 2, 3, 5 и 7. Таблица 9.1 показывает результат.

Процесс состоит в следующем

1. Вычеркнуть все числа, делимые без остатка на 2 (кроме самого 2).
2. Вычеркнуть все числа, делимые без остатка на 3 (кроме самого 3).
3. Вычеркнуть все числа, делимые без остатка на 5 (кроме самого 5).
4. Вычеркнуть все числа, делимые без остатка на 7 (кроме самого 7).
5. Оставшиеся числа – простые.

Phi-функция Эйлера

Phi-функция Эйлера, $\phi(n)$, которую иногда называют **тотиентой Эйлера**, играет очень важную роль в криптографии. Функция $\phi(n)$ находит из ряда чисел 0,1...., n–1 числа, взаимно простые с n. Можно вспомнить из лекции 2, что множество Z_{n^*} —

Таблица 9.1. Решето Эратосфена

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

числа, которые больше чем n и взаимно простые с n. Функция $\phi(n)$ вычисляет число элементов этого множества. Ниже показано, как найти это значение

1. $\phi(1) = 0$.
2. $\phi(p) = p - 1$, если p — простое число.
3. $\phi(m \times n) = \phi(m) \times \phi(n)$ если m и n — взаимно простые.
4. $\phi(p^e) = p^e - p^{e-1}$, если p — простое.

Мы можем объединить эти четыре правила, предназначенные для нахождения $\varphi(n)$.

$$\phi(n) = (p_1^{e1} - p_1^{e1-1}) \times (p_2^{e2} - p_2^{e2-1}) \times \ldots \times (p_k^{ek} - p_k^{ek-1})$$

Очень важно заметить, что значение $\phi(n)$ для больших чисел может быть найдено, если может быть найдено число n и если n может быть представлено в виде разложения простых чисел. Другими словами, трудность нахождения $\phi(n)$ зависит от трудности нахождения разложения n. Это рассматривается в следующем разделе.

Трудность нахождения $\phi(n)$ зависит от трудности нахождения разложения n.

Пример 9.7
Какое значение имеет $\phi(13)$?
Решение
Поскольку 13 — простое число, $\phi(13) = (13 - 1) = 12$.

Пример 9.8
Какое значение имеет $\phi(10)$?
Решение
Мы можем использовать третье правило: $\phi(10) = \phi(2) \times \phi(5) = 1 \times 4 = 4$, поскольку 2 и 5 — простые числа.

Пример 9.9
Какое значение имеет $\phi(240)$?

Решение
Мы можем записать $240 = 2^4 \times 3^1 \times 5^1$.
Тогда

$$\phi(240) = (2^4 - 2^3) \times (3^1 - 3^0) \times (5^1 \times 5^0) = 64$$

Пример 9.10
Можно ли утверждать, что $\phi(49) = \phi(7) \times \phi(7) = 6 \times 6 = 36$?
Решение
Нет.

$$\phi(49) = 7^2 - 7^1 = 42$$

Пример 9.11
Какие числа являются элементами в Z_{14}^*?
Решение
$\phi(14) = \phi(14) = \phi(7) \times \phi(2) = 6 \times 1 = 6$. Элементы – это 1, 3, 5, 9, 11 и 13.

Интересный факт: если n >2, значение $\phi(n)$ — четное.

Малая теорема Ферма

Малая теорема Ферма играет очень важную роль в теории чисел и криптографии. Ниже мы приводим две версии теоремы.

Первая версия
Первая версия говорит, что если p — простое число и *a* — целое число, такое, что *p* не является делителем *a,* то $a^{p-1} \equiv 1 \bmod p$.

Вторая версия
Вторая версия вводит ограничивающие условие на *a*. Она утверждает, что если p — простое число и *a* — целое число, то $a^{p-1} \equiv a \bmod p$.

Приложения
Хотя мы будем рассматривать приложения этой теоремы позже в этой лекции, теорема очень полезна для того, чтобы решить некоторые проблемы.

Возведение в степень. Малая теорема Ферма иногда полезна для того, чтобы быстро найти решение при возведении в степень. Следующие примеры показывают это.

Пример 9.12
Найдите результат $6^{10} \bmod 11$.
Решение
Мы имеем $6^{10} \bmod 11 \equiv 1$. Это первая версия малой теоремы Ферма, где $p = 11$.

Пример 9.13

Найдите результат $3^{12} \bmod 11$.

Решение

Здесь степень (12) и модуль (11) не соответствуют условиям теоремы Ферма. Но, применяя преобразования, мы можем привести решение к использованию малой теоремы Ферма.

$$3^{12} \bmod 11 = (3^{11} \times 3) \bmod 11 = (3^{11} \bmod 11) \times (3 \bmod 11) = (3 \times 3) \bmod 11 = 9$$

Мультипликативные инверсии. Очень интересное приложение теорема Ферма имеет для некоторых мультипликативных инверсий, если модуль — простое число. Если p — простое число и a — целое число, такое, что p не является его делителем, тогда $a^{-1} \bmod p = a^{p-2} \bmod p$. Это может быть легко доказано, если мы умножим обе стороны равенства на a и используем первую версию малой теоремы Ферма:

$$a \times a^{-1} \bmod p = a \times a^{p-2} \bmod p = a^{p-1} \bmod p = 1 \bmod p$$

Это приложение позволяет не применять расширенный алгоритм Евклида для нахождения мультипликативных инверсий.

Пример 9.14

Инверсии по модулю простого числа могут быть найдены без использования расширенного Евклидова алгоритма:

a. $8^{-1} \bmod 17 = 8^{17-2} \bmod 17 = 8^{15} \bmod 17 = 15 \bmod 17$

b. $5^{-1} \bmod 23 = 5^{23-2} \bmod 23 = 5^{21} \bmod 23 = 14 \bmod 23$

c. $6^{-1} \bmod 101 = 60^{101-2} \bmod 101 = 60^{99} \bmod 101 = 32 \bmod 101$

d. $22^{-1} \bmod 211 = 22^{211-2}$ moda $211 = 22^{209} \bmod 211 = 48 \bmod 211$

Теорема Эйлера

Теорему Эйлера можно представить как обобщения малой теоремы Ферма. Модуль в теореме Ферма — простое число, модуль в теореме Эйлера — целое число. Мы вводим две версии этой теоремы.

Первая версия

Первая версия теоремы Эйлера подобна первой версии малой теоремы Ферма. Если a и n – взаимно простые, то $a^{\phi(n)} \equiv 1 \bmod n$.

Вторая версия

Вторая версия теоремы Эйлера подобна второй версии малой теоремы Ферма; она устраняет условие, что n должно быть взаимно простым с a. Если $n = p \times q$, $a < n$, а k — целое число, то $a^{k \times \phi(n) + 1} \equiv a \bmod n$.

Приведем нестрогое доказательство второй версии, основанной на первой версии. Поскольку $a < n$, то возможны три случая.

1. Если a не кратно ни числу p, ни числу q, то a и n – взаимно простые.

$$a^{k\times\phi(n)+1} \bmod n = (a^{\varphi(n)})^k \times a \bmod n = (1)^k \times a \bmod n = a \bmod n$$

2. Если a — кратное число p, $a = i \times p$, но не кратно числу q.

$$a^{\phi(n)} \bmod q = (a^{\phi(q)} \bmod q)^{\phi(p)} \bmod q = 1 \quad --> a^{\phi(n)} \bmod q = 1$$
$$a^{k\times\phi(n)} \bmod q = (a^{\phi(n)} \bmod q)^k \bmod q = 1 \quad --> a^{k\times\phi(n)} \bmod q = 1$$
$$a^{k\times\phi(n)} \bmod q = 1 \quad --> a^{k\times\phi(n)} = 1 + j\times q \text{ (интерпретация сравнения)}$$
$$a^{k\times\phi(n)+1} = a\times(1 + j\times q) = a + j\times q\times a = a + (i\times j)\times q\times p = a + (i\times j)\times n$$
$$a^{k\times\phi(n)+1} = a + (i\times j)\times n \quad --> a^{k\times\phi(n)+1} = a \bmod n \text{ (отношение конгруэнтности)}$$

3. Если a кратно q $(a = i \times q)$, но не кратно p, доказательство второго случая то же самое, но p и q меняются местами.

Вторая версия теоремы Эйлера используется в криптографической системе RSA (лекция 10).

Приложения

Хотя мы рассмотрим некоторые приложения теоремы Эйлера позже в этой лекции, теорема очень полезна для того, чтобы решать некоторые задачи.

Возведение в степень. Теорема Эйлера иногда полезна, чтобы быстро найти решение некоторых задач с возведением в степень. Следующие примеры показывают идею этого процесса.

Пример 9.15

Найдите результат $6^{24} \bmod 35$.

Решение

Мы имеем $6^{24} \bmod 35 = 6^{\phi(35)} \bmod 35 = 1$

Пример 9.16

Найдите результат $20^{62} \bmod 77$.

Решение

Если введем k = 1 согласно второй версии, мы имеем:

$20^{62} \bmod 77 = (20 \bmod 77) \bmod 77 = (20)(20) \bmod 77 = 15$

Мультипликативные инверсии. Теорема Эйлера может использоваться, чтобы найти мультипликативную инверсию по простому модулю. Теорема Эйлера может применяться, чтобы найти мультипликативные инверсии по составному модулю. Если n и a — взаимно простые, то $a^{-1} \bmod n = a^{\phi(n)-1} \bmod n$. Это может быть легко доказано умножением обеих сторон равенства на a.

$$a^{-1} \bmod n = a \times a^{\phi(n)-1} \bmod n = a^{\phi(n)} \bmod n = 1 \bmod n$$

Пример 9.17

Мультипликативная инверсия по составному модулю может быть найдена без использования расширенного евклидова алгоритма, если мы знаем разложение на множители составного объекта:

a. $8^{-1} \bmod 77 = 8^{\phi(77)-1} \bmod 77 = 8^{59} \bmod 77 = 29 \bmod 77$

b. $7^{-1} \bmod 15 = 7^{\phi(15)-1} \bmod 15 = 7^{7} \bmod 15 = 13 \bmod 15$

c. $6^{-1} \bmod 187 = 60^{\phi(187)-1} \bmod 187 = 60^{159} \bmod 187 = 53 \bmod 187$

d. $71^{-1} \bmod 100 = 71^{\phi(100)-1} \bmod 100 = 71^{39} \bmod 100 = 31 \bmod 100$

Генерация простых чисел

Два математика, Мерсенна и Ферма, попытались получить формулу, которая могла бы генерировать простые числа.

Простые числа Мерсенны

Мерсенна предложил следующую формулу, которую называют **числа Мерсенны**. Он предполагал, что формула перечисляет все простые числа.

$$\mathbf{M_p = 2^p - 1}$$

Если p в приведенной выше формуле — простое число, то, как предполагали, M_p должно быть простым числом. Годы спустя было доказано, что не все числа, полученные по формуле Мерсенны, — простые числа. Ниже приведен список некоторых номеров Мерсенны.

$M_2 = 2^2 - 1 = 3$	
$M_3 = 2^3 - 1 = 7$	
$M_5 = 2^5 - 1 = 31$	
$M_7 = 2^7 - 1 = 127$	
$M_{11} = 2^{11} - 1 = 2047$	**Непростое число** ($2047 = 23 \times 89$)
$M_{13} = 2^{13} - 1 = 8191$	
$M_{17} = 2^{17} - 1 = 131071$	

Оказалось, что M_{11} — не простое число. Однако было найдено, что 41 число по формуле Мерсенны — простые; одно из последних найденных чисел Мерсенны — $M_{124036583}$, наибольшее число содержит 7 253 733 цифр. Поиск продолжается.

Ферма пробовал найти формулу, которая генерирует простые числа. Следующая формула — для **чисел Ферма**.

Число в формуле $M_p = 2^p - 1$, называемое числом Мерсенны, может быть или не быть простым числом.

Простые числа Ферма

Ферма попытался найти формулу для генерации простых чисел. Он предложил следующую формулу, которая теперь называется формулой Ферма, и проверил номера от F_0 (n=0,1,...) до F_4, но оказалось, что уже F_4 — не простое число.

$$F_n = 2^{2^n} + 1$$

$F_0 = 3$
$F_1 = 17$
$F_2 = 257$
$F_3 = 65537$
$F_4 = 4294967297 = 641 \times 6700417$. **Не простое число**

Фактически было доказано, что многие номера до F_{24} — составные числа.

9.2. Испытание простоты чисел

Если формулы получения простых чисел, подобно формулам Ферма или Мерсенна, не гарантируют, что полученные числа — простые, то как мы можем генерировать большие простые числа для криптографии? Мы можем только выбрать случайно большое число и провести испытание, чтобы убедиться, что оно — простое.

Нахождение алгоритма, который правильно и эффективно проверяет очень большое целое число и устанавливает, *простое* это *число* или же *составной объект*, — всегда было проблемой в теории чисел и, следовательно, в криптографии. Однако, недавние исследования (одно из которых мы обсуждаем в этом разделе) выглядят очень перспективными.

Алгоритмы, которые решают эту проблему, могут быть разделены на две обширные категории — детерминированные алгоритмы и вероятностные алгоритмы. Ниже рассматриваются некоторые представители обеих категорий. Детерминированный алгоритм всегда дает правильный ответ. Вероятностный алгоритм дает правильный ответ в большинстве, но не во всех случаях. Хотя детерминированный алгоритм идеален, он обычно менее эффективен, чем соответствующий вероятностный.

Детерминированные алгоритмы

Детерминированный алгоритм, проверяющий простоту чисел, принимает целое число и выдает на выходе признак: это число — *простое число* или *составной объект*. До недавнего времени все детерминированные алгоритмы были неэффективны для нахождения больших простых чисел. Как мы коротко покажем, новые взгляды делают эти алгоритмы более перспективными.

Алгоритм теории делимости

Самое элементарное детерминированное испытание на простоту чисел — испытание на делимость. Мы используем в качестве делителей все числа, меньшие, чем $\sqrt{n}$. Если любое из этих чисел делит n, тогда n — *составное*. Алгоритм 9.1 показывает проверку на делимость в ее примитивной и очень неэффективной форме.

Алгоритм может быть улучшен, если проверять только нечетные номера. Он может быть улучшен далее, если пользоваться таблицей простых чисел между 2 и

$\sqrt{n}$. Число арифметических операций в алгоритме 9.1 — $\sqrt{n}$. Если мы принимаем, что каждая арифметическая операция использует только операцию на один бит (чисто условное соглашение), тогда сложность разрядной операции алгоритма 9.1 — $f(n_b) = \sqrt{2^{n_b}} = 2^{n_b/2}$, где n_b — число битов в n. В больших системах, обозначаемых O, сложность может быть оценена $O(2^{n_b})$: экспоненциально (см. приложение L). Другими словами, алгоритм делимости неэффективен, если n_b большое.

Сложность побитного испытания делимостью показательна.

Пример 9.18

Предположим, что n имеет 200 битов. Какое число разрядных операций должен был выполнить алгоритм делимости?

Решение

Сложность побитовых операций этого алгоритма — $2^{n_b/2}$. Это означает, что алгоритму необходимо провести 2^{100} битовых операций. Если алгоритм имеет скорость 2^{30} операций в секунду, то для проведения испытаний необходимо 2^{70} секунд.

Алгоритм 9.1.

```
Тест на делимость (n)      //n - число тестов на простоту
{
   r ← 2
   while (r< √n)
   {
   if (r | n) return "a composite" // составное
   r ← r+1
   }
   return "a prime" //простое
}
```

AKS-алгоритм

В 2002 г. индийские ученые Агравал, Каял и Сахсена (Agrawal, Kayal и Saxena) объявили, что они нашли алгоритм для испытания простоты чисел с полиномиальной сложностью времени разрядных операций 0 $((\log_2 n_b))$. Алгоритм использует тот факт, что $(x - a)^p \equiv (x^p - a) \bmod p$. Интересно отметить, что некоторые будущие разработки делают этот алгоритм стандартным тестом для определения простоты чисел в математике и информатике.

Пример 9.19

Предположим, что n имеет 200 битов. Какое число разрядных операций должен был выполнить алгоритм AKS?

Решение

Сложность разрядной операции этого алгоритма — $\mathbf{O}((\log_2 n_b)^{12})$. Это означает, что алгоритму надо только $(\log_2 200)^{12} = 39\,547\,615\,483$ битовых операций. На компьютере, способном выполнить 1 миллиард битов в секунду, алгоритму требуется только 40 секунд.

Вероятностные алгоритмы

До AKS-алгоритма все эффективные методы для испытания простоты чисел были вероятностные. Эти методы могут использоваться еще некоторое время, пока AKS формально не принят как стандарт. Вероятностный алгоритм не гарантирует правильность результата. Однако мы можем получить вероятность ошибки настолько маленькую, что это почти дает гарантию, что алгоритм вырабатывает правильный ответ. Сложность разрядной операции алгоритма может стать полиномиальной, при этом мы допускаем небольшой шанс для ошибок. Вероятностный алгоритм в этой категории возвращает результат либо *простое число,* либо *составной объект,* основываясь на следующих правилах:

a. если целое число, которое будет проверено, — фактически простое число, алгоритм явно возвратит *простое число;*
b. если целое число, которое будет проверено, — фактически составной объект, алгоритм возвращает *составной объект* с вероятностью 1−ε, но может возвратить *простое число* с ε вероятности. Вероятность ошибки может быть уменьшена, если мы выполняем алгоритм несколько раз с различными параметрами или с использованием различных методов. Если мы выполняем алгоритм m раз, вероятность ошибки может уменьшиться до ε^m.

Тест Ферма

Первый вероятностный метод, который мы обсуждаем, — **испытание простоты чисел тестом Ферма.**

Если n — простое число, то $a^{n-1} \equiv 1 \bmod n$.

Обратите внимание, что если n — простое число, то сравнение справедливо. *Это не означает, что если сравнение справедливо, то n — простое число.* Целое число может быть простым числом или составным объектом. Мы можем определить следующие положения как тест Ферма:

Если n — простое число, то $a^n - 1 \equiv 1 \bmod n$
Если n — составной объект, то возможно, что $a^n - 1 \equiv 1 \bmod n$

Простое число удовлетворяет тесту Ферма. Составной объект может пройти тест Ферма с вероятностью ε. Сложность разрядной операции испытания Ферма равна сложности алгоритма, который вычисляет возведение в степень. Позже в этой лекции мы рассмотрим алгоритм для быстрого возведения в степень со сложностью разрядной операции $\boldsymbol{O}(n_b)$, где $\boldsymbol{O}$ — номер битов в n. Вероятность может быть улучшена, если проверка делается с несколькими числами (a_1, a_2 и так далее). Каждое испытание увеличивает вероятность, что испытуемое число – это простое число.

Пример 9.20

Проведите испытание Ферма для числа 561.

Решение

Используем в качестве основания число 2.

$$2^{561-1} = 1 \bmod 561$$

Число прошло тест Ферма, но это — не простое число, потому что $561 = 33 \times 17$.

Испытание квадратным корнем

В модульной арифметике, если n — простое число, то квадратный корень равен только 1 (либо +1, либо –l). Если n — составной объект, то квадратный корень — +1 или (-1), но могут быть и другие корни. Это называют **испытанием простоты чисел квадратным корнем.** Обратите внимание, что в модульной арифметике -1 означает $n-1$.

Если n — простое число, $\sqrt{1} \bmod n = \pm 1$.
Если n — составной объект, $\sqrt{1} \bmod n = \pm 1$, и возможны другие значения.

Пример 9.21

Каковы квадратные корни 1 mod n, если n равно 7 (простое число)?

Решение

Единственные квадратные корни 1 mod n – это числа 1 и -1. Мы можем видеть, что

$1^2 = 1 \bmod 7$ $\quad$ $(-1)^2 = 1 \bmod 7$
$2^2 = 4 \bmod 7$ $\quad$ $(-2^2) = 4 \bmod 7$
$3^2 = 2 \bmod 7$ $\quad$ $(-3^2) = 2 \bmod 7$

Заметим, что тест не дает результатов для 4, 5 и 6, потому что $4 = -3 \bmod 7$, $5 = -2 \bmod 7$ и $6 = -1 \bmod 7$.

Пример 9.22

Каков квадратный корень из 1 mod n, если n равно 8 (составное)?

Решение

Имеется три решения: 1, 3, 5 и 7 (а также -1). Мы можем также видеть, что

$1^2 = 1 \bmod 8$ $\quad$ $(-1)^2 = 1 \bmod 8$
$3^2 = 1 \bmod 8$ $\quad$ $(-5^2) = 1 \bmod 8$

Пример 9.23

Каков квадратный корень из 1 mod n, если n равно 17 (простое)?

Решение

Имеются только два решения, соответствующие поставленной задаче: это 1 и (-1).

$1^2 = 1 \bmod 17$ $\quad$ $(-1)^2 = 1 \bmod 17$
$2^2 = 4 \bmod 17$ $\quad$ $(-2^2) = 4 \bmod 17$
$3^2 = 9 \bmod 17$ $\quad$ $(-3^2) = 9 \bmod 17$
$4^2 = 16 \bmod 17$ $\quad$ $(-4)^2 = 16 \bmod 17$

$5^2 = 8 \bmod 17$ $(-5^2) = 8 \bmod 17$
$6^2 = 2 \bmod 17$ $(-6^2) = 2 \bmod 17$
$7^2 = 15 \bmod 17$ $(-7)^2 = 15 \bmod 17$
$8^2 = 13 \bmod 17$ $(-8^2) = 13 \bmod 17$

Заметим, что не надо проверять целые числа, большие 8, потому что
$9 = -8 \bmod 17$

Пример 9.24
Каков квадратный корень из 1 mod n, если n равно 22 (составное)?
Решение
Сюрприз в том, что имеется только два решения: +1 и –1, хотя 22 — составное число.

$1^2 = 1 \bmod 22$
$(-1)^2 = 1 \bmod 22$

Хотя во многих случаях имеется испытание, которое показывает нам однозначно, что число составное, но это испытание провести трудно. Когда дано число n, то все числа, меньшие, чем *n* (кроме чисел 1 и n–1), должны быть возведены в квадрат, чтобы гарантировать, что ни одно из них не равно 1. Такое испытание может использоваться для чисел (не +1 или –1), которые в квадрате по модулю n дают значение 1. Этот факт помогает в испытании Миллера–Рабина, которое рассматривается в следующем разделе.

Тест Миллера-Рабина
Тест Миллера-Рабина определения простого числа есть комбинация тестов *Ферма* и *квадратного корня*. Он элегантным способом находит **сильное псевдопростое число** (простое число с очень высокой вероятностью). В этом тесте мы записываем *n*–1 как произведение нечетного числа m и степени числа 2.

$$n - 1 = m \times 2^k$$

В тесте Ферма при основании *a* можно записать так, как это показано на рис. 9.2.

$$a^{n-1} = a^{m \times 2^k} = [a^m]^{2^k} = [a^m]^{2^{2^{\cdot^{\cdot^{2}}}}} \text{ k-раз}$$

Рис. 9.2. Идея теста на простоту числа на основе Ферма

Другими словами, вместо того чтобы вычислять $a^{n-1} (mod\ n)$ в один шаг, мы можем сделать это в $k + 1$ шагов. Какое преимущество в таком применении? Преимущество заключается именно в том, что испытание квадратным корнем может быть выполнено на каждом шаге. Если квадратный корень показывает сомнительные результаты, мы останавливаемся и объявляем *n* составным номером. На каждом шаге

мы обеспечиваем, что тест Ферма и испытание квадратным корнем удовлетворено на всех парах смежных шагов, если оно удовлетворительно (если результат равен 1).

Инициализация

Выберите основу и вычислите T = a^m, в который $m = (n - 1) / 2^k$.

a. Если T равно +1 или −1, объявляют, что n — псевдопростое число, и процесс останавливается. Мы говорим, что n прошел два испытания: тест Ферма и испытание квадратным корнем. Почему? Потому что если T равно ±1, то T станет 1 на следующем шаге и остается 1 до прохождения теста *Ферма*. Кроме того, T прошел испытание *тестом квадратного корня*, потому что T был бы равен 1 на следующем шаге и квадратный корень был бы равен 1 (на следующем шаге) и равен ±1 (на этом шаге).

b. Если T равен другому значению, мы не уверены, является ли n простым числом или составным объектом, значит, процесс будет продолжаться на следующем шаге.

Шаг 1

Возводим T в квадрат.

a. Если результат равен +1, мы определенно знаем, что тест Ферма пройден, потому что T остается 1 для последующих испытаний. Испытание квадратным корнем, однако, не пройдено. Поскольку T равно 1 на этом шаге и имело на предыдущем шаге другое значение, чем 1 (причина, почему мы не остановились на предыдущем шаге), n объявляют составным объектом, и процесс останавливается.

b. Если результат равен (−1), мы знаем, что n в конечном счете пройдет тест Ферма. Мы знаем, что он пройдет испытание квадратным корнем, потому что T равно (−1) в этом шаге и станет 1 на следующем шаге. Мы объявляем n псевдослучайным простым числом и останавливаем процесс.

c. Если T имеет еще какое-либо значение, мы не уверены, имеем ли мы дело с простым числом, и процесс продолжается на следующем шаге.

Шаги 2 до k−1

Этот шаг и все остальные шаги до k−1 такие же, как и шаг 1.

Этот шаг не является необходимым. Если мы достигли его и не приняли решение, он не поможет нам. Если результат этого шага (−1), значит, тест Ферма пройден, но поскольку результат предыдущего шага — не ±1, испытание квадратное корня не пройдено. После шага $k - 1$, если процесс не остановлен, мы объявляем, что n — составное.

Тест Миллера-Рабина требует от 0 до k−1.

Алгоритм 9.2 показывает псевдокод для теста Миллера-Рабина.

Существует доказательство, что каждый раз, когда для числа проводится тест Миллера-Рабина, вероятность получить результат «не простое число» — 1/4. Если прошло m тестов (с m различными основаниями), вероятность, что тест выдаст не простое число — $(1/4)^m$.

Алгоритм 9.2. Псевдокод для теста Миллера-Рабина

```
Тест Миллера-Рабина (n, a)        // n – число; a – основание
{
   Find m and k such that n-1 = m x 2^k
   T ← a^m mod n
   if ( T = ±1) return "a prime"
   for (I ← 1 to k-1)             // k-1 – максимальное число шагов
   {
      T ← T^2 mod n
      if (T = +1) return " a composite"   // составное
      if (T = -1) return " a prime"       // простое
   }
   return " a composite"
}
```

Пример 9.25

Проведите тест Миллера-Рабина к числу 561.

Решение

Используя основание 2, получим $561 - 1 = 35 \times 2^4$, что означает, что $m = 35$, $k = 4$ и $a = 2$

Инициализация:	$T = 2^{35} \bmod 561 = 263 \bmod 561$
k = 1	$T = 263^2 \bmod 561 = 166 \bmod 561$
k = 2	$T = 166^2 \bmod 561 = 67 \bmod 561$
k = 3	$T = 67^2 \bmod 561 = +1 \bmod 561$ → **составное**

Пример 9.26

Мы уже знаем, что 27 — не простое число. Попробуем применить тест Миллера-Рабина.

Решение

Основание равно 2, тогда $27 - 1 = 13 \times 2^1$, что означает m = 13, $k = 1$ и $a = 2$. В этом случае $k - 1 = 0$, и мы должны сделать только шаг инициализации: $T = 2^{13} \bmod 27 = 11 \bmod 27$. Однако поскольку алгоритм не делает ни одного цикла, вырабатывается решение «*составной объект*».

Пример 9.27

Мы знаем, что 61 — простое число; давайте посмотрим, что даст тест Миллера-Рабина.

Решение

Мы используем основание 2.

$61 - 1 = 15 \times 2^2 \rightarrow m = 15 \quad k = 2 \quad a = 2$

Инициализация: $T = 2^{15} \bmod 61 = 11 \bmod 61$

$k = 1 \qquad T = 11^2 \bmod 61 = -1 \bmod 61$ → **простое число**

Обратите внимание, что последний результат — это 60 mod 61, но мы знаем, что $60 = -1 \bmod 61$.

Рекомендованные тесты простоты чисел

Сегодня один из самых популярных тестов простоты чисел — комбинация теории делимости и тест Миллера-Рабина. При этом рекомендуются следующее шаги.

1. Выбрать нечетное целое число, потому что все четные целые числа (кроме 2) — явно составные объекты.
2. Сделать некоторые тривиальные испытания теории делимости на некоторых известных простых числах, таких, как 3, 5, 7, 11, 13: так, чтобы убедиться, что вы не имеете дело с очевидным составным объектом. Если они не являются делителями при всех этих испытаниях, сделайте следующий шаг. Если выбранное число не прошло хотя бы один из этих тестов, вернитесь на один шаг и выберите другое нечетное число.
3. Выбрать набор оснований для теста. Большое множество оснований предпочтительно.
4. Сделать тест Миллера-Рабина на каждом из оснований. Если любой из них не проходит, вернитесь на один шаг и выберите другой нечетный номер. Если тесты прошли для всех оснований, объявите это число как сильное псевдопростое число.

Пример 9.28

Номер 4033 — составной объект (37×109). Это подтверждает рекомендованное испытание простоты чисел?

Решение

1. Выполним проверку согласно теории делимости. Проверим сначала числа 2, 3, 5, 7, 11, 17 и 23 – они не являются делителями числа 4033.
2. Выполним испытание Миллера-Рабина с основанием 2, тогда $4033 - 1 = 63 \times 2^6$, что означает $m = 63$ и $k = 6$.

Инициализация: $T \equiv 2^{63} \pmod{4033} \equiv 1 \pmod{4033}$
$k = 1$ $\quad T \equiv T^2 = 3521^2 \pmod{4033} \equiv -1 \pmod{4033}$ → **Тест прошел**

3. Но мы не удовлетворены. Мы продолжаем с другим основанием — 3.

Инициализация: $T \equiv 3^{63} \pmod{4033} \equiv 3551 \pmod{4033}$
$k = 1$ $\quad T \equiv T^2 \equiv 3551^2 \pmod{4033} \equiv 2443 \pmod{4033}$
$k = 2$ $\quad T \equiv T^2 \equiv 2443^2 \pmod{4033} \equiv 3442 \pmod{4033}$
$k = 3$ $\quad T \equiv T^2 \equiv 3442^2 \pmod{4033} \equiv 2443 \pmod{4033}$
$k = 4$ $\quad T \equiv T^2 \equiv 2443^2 \pmod{4033} \equiv 3442 \pmod{4033}$
$k = 5$ $\quad T \equiv T^2 \equiv 3442^2 \pmod{4033} \equiv 2443 \pmod{4033}$ → **Не соответствует (составное)**

9.3. Разложение на множители

Разложение на множители — предмет непрерывного исследования в прошлом; и такие же исследования, вероятно, продолжатся в будущем. Разложение на множители играет очень важную роль в безопасности некоторых криптосистем с открытым ключом (см. лекцию 10).

Основная теорема арифметики

Согласно *Основной теореме арифметики* любое положительное целое число больше единицы может быть уникально записано в следующей главной форме **разложения на множители**, где $p_1, p_2, \ldots, p_k$ — простые числа и $e_1, e_2, \ldots, e_k$ — положительные целые числа:

$$n = p_1^{e1} \times p_2^{e2} \times \ldots \times p_k^{ek}$$

Есть непосредственные приложения разложения на множители, такие как вычисление наибольшего общего делителя и наименьшего общего множителя.

Наибольший общий делитель

В лекции 2 мы уже обсуждали наибольший общий делитель двух номеров, НОД (a, *b*). Посмотрите, как евклидов алгоритм находит это значение, но это значение может также быть найдено, если мы знаем разложение на множители чисел a и *b*.

$$a = p_1^{a1} \times p_2^{a2} \times \ldots \times p_k^{ak} \qquad b = p_1^{b1} \times p_2^{b2} \times \ldots \times p_k^{bk}$$
$$\text{НОД (a,b)} = p_1^{min(a1,b1)} \times p_2^{\min(a2,b2)} \times \ldots \times p_k^{\min(ak,bk)}$$

Наименьшее общее кратное

Наименьшее общее кратное, *НОК* (*a, b*), — наименьшее целое число, кратное числам *a* и *b*. Используя разложение, мы также находим НОК (*a*, *b*).

$$a = p_1^{a1} \times p_2^{a2} \times \ldots \times p_k^{ak} \qquad b = p_1^{b1} \times p_2^{b2} \times \ldots \times p_k^{bk}$$
$$\text{НОК (a,b)} = p_1^{max(a1,b1)} \times p_2^{\max(a2,b2)} \times \ldots \times p_k^{\max(ak,bk)}$$

Может быть доказано, что НОД (a,b) и НОК (a,b) связаны с друг другом, как это показано ниже:

$$\text{НОК (a,b)} \times \text{НОД (a,b)} = \text{a} \times \text{b}$$

Методы разложения на множители

Поиск эффективных алгоритмов для разложения на множители больших составных чисел ведется давно. К сожалению, совершенный алгоритм для этого пока не найден. Хотя есть несколько алгоритмов, которые могут разложить число на множители, ни один не способен провести разложение достаточно больших

чисел в разумное время. Позже мы увидим, что это хорошо для криптографии, потому что современные криптографические системы полагаются на этот факт. В этой секции мы даем несколько простых алгоритмов, которые проводят разложение составного числа. Цель состоит в том, чтобы сделать процесс разложения на множители менее трудоёмким.

Метод проверки делением

Самый простой и наименее эффективный алгоритм — **метод разложения на множители проверкой делением.** Мы просто пробуем все положительные целые числа начиная с 2, для того чтобы найти одно, которое делит n. После обсуждения *решета Эратосфена* мы знаем, что если n составное, то делитель будет простым числом $p \le \sqrt{n}$. Алгоритм 9.3 реализует этот метод. Алгоритм имеет два цикла: один внешний и один внутренний, находит уникальные множители в разложении; внутренняя петля находит повторяющиеся множители разложения. Например, $24 = 2^3 \times 3$. Внешний цикл множители 2 и 3. Внутренний цикл находит, что число 2 — множитель .

Алгоритм 9.3.

```
разложение проверкой_делением (n)   // n раскладываемое число
{
   a ← 2
   while (a ≤ √n)
   {
      while (n mod a = 0)
      {
         output a      // элементы выхода «один за другим»
         n = n/a
      }
      a ← a +1
   }      // n не имеет больше множителей
   if (n>1) output n
}
```

Сложность. Метод проверки делением обычно хорош, если $n < 2^{10}$, но он неэффективен и неосуществим для разложения больших целых чисел. Сложность алгоритма (приложение L) *показательна*.

Пример 9.29

Используйте алгоритм проверки делением, чтобы найти сомножители числа 1233.

Решение

Мы выполняем программу, основанную на алгоритме, и получаем следующий результат:

$1233 = 3^2 \times 137$

Пример 9.30

Используйте алгоритм проверки делением, чтобы найти сомножители 1523357784.

Решение

Мы выполняем программу, основанную на алгоритме, и получаем следующий результат:

$1523357784 = 2^3 \times 3^2 \times 13 \times 37 \times 43987$

Метод Ферма

Метод Ферма разложения на множители (алгоритм 9.4) делит номер n на два положительных целых числа (a и b — не обязательно простые числа) так, чтобы $n = a \times b$.

Алгоритм 9.4. Псевдокод для разложения на множители по методу Ферма

```
Разложение_на_множители Ферма (n)   // n - раскладываемое число
{
   x ← √n  // наименьшее целое, большее, чем √n
   while (<n)
   {
   w ← x² - n
   if (w полный квадрат числа) y← √w; a ← x + y; b ← x-y;
   return a and b
   x ← x+1
   }
}
```

Метод Ферма основан на факте, что если мы можем найти x и y, такие, что $n = x^2 - y^2$, тогда мы имеем

$$n = x^2 - y^2 = a \times b \text{ при } a = (x + y) \text{ и } b = (x - y)$$

Метод сводится к попытке найти два целых числа a и b, близкие друг к другу ($a \approx b$). Начинаем с наименьшего целого числа, большего, чем $x = \sqrt{n}$. Потом пробуем найти другое целое число y, такое, чтобы выполнялось уравнение $y^2 = x^2 - n$. В каждой итерации мы должны рассмотреть, является ли результат $x^2 - n$ полным квадратом. Если мы находим такое значение для y, мы вычисляем a и b и выходим из цикла. Если мы не делаем этого, мы проводим другую итерацию.

Заметим, что метод не обязательно находит разложение на простые числа (каноническое разложение); *алгоритм* должен быть повторен рекурсивно для каждого из значений a и b, пока не будут найдены сомножители в виде простых чисел.

Сложность. Сложность метода Ферма является близкой к показательному закону (см. приложение L).

p – 1 метод Полларда

В 1974 г. Джон Поллард разработал метод, который находит разложение числа p на простые числа. Метод основан на условии, что $p - 1$ не имеет сомножителя, большего, чем заранее определенное значение B, называемое границей. Алгоритм Полларда показывает, что в этом случае

$$p = \text{НОД}\,(2^{B!} - 1,\ n)$$

Алгоритм 9.5 показывает псевдокод для ***p* – 1 метода Полларда разложения на множители**. Когда мы выходим из второго цикла, в a сохраняется $2^{B!}$.

Алгоритм 9.5. Псевдокод для $p - 1$ метода Полларда разложения на множители

```
Pollard_(p-1)_Factorization (n,B) // n - раскладываемое число
{
   a ← 2
   e ← 2
   while (e ≤ B)
   {
      a ← a^e mod n
      e ← e +1
   }
   p ← gsd (a-1, n)   // gsd - НОД (наибольший общий делитель)
   if 1 < p < n return p
   return failure
}
```

Сложность. Заметим, что этот метод требует сделать B – 1 операций возведения в степень *(a = a e* mod *n)*. Как мы увидим позже в этой лекции, есть быстрый алгоритм возведения в степень, который выполняет это за $2\ 1\log_2 B$ операций. Метод также использует вычисления НОД, который требует n^3 операций. Мы можем сказать, что сложность — так или иначе больше, чем **O(B)** или $O(2^{n_b})$, где n_b — число битов в B. Другая проблема – этот алгоритм может заканчиваться сигналом об ошибке. Вероятность успеха очень мала, если B имеет значение, не очень близкое к величине $\sqrt{n}$.

Пример 9.31

Используя $p - 1$ метод Полларда, найдите сомножители числа 57247159 с границей B = 8.

Решение

Мы выполняем программу, основанную на рассмотренном выше алгоритме, и находим, что $p = 421$. Фактически $57247159 = 421 \times 135979$. Обратите внимание, что 421 — простое число и $p - 1$ не имеет ни одного сомножителя, большего 8, т.е. $(421 - 1 = 2^2 \times 3 \times 5 \times 7)$.

PO (Rho) – метод Полларда

В 1975 г. Джон М. Поллард разработал второй метод для разложения на множители, который базируется на следующих положениях.

a. Предположим, что есть два целых числа, x_1 и x_2, таких, что p делит $x_1 - x_2$, но эта разность не делится на n.

b. Может быть доказано, что p = НОД $(x_1 - x_2, n)$. Поскольку p делит $x_1 - x_2$, можно записать, что $x_1 - x_2 = q \times p$. Но поскольку n не делит $x_1 - x_2$, очевидно, что q не делится n. Это означает, что НОД $(x_1 - x_2, n)$ является либо 1, либо сомножитель.

Следующий алгоритм повторно выбирает x_1 и x_2, пока не находит соответствующую пару.

1. Выберите x_1 — малое случайное целое число, называемое первоисточником.
2. Используйте функцию, чтобы вычислить x_2, такую, чтобы n не делило $x_1 - x_2$. Функция, которая может быть применена, — это $x_2 = f(x_1) = x_1^2 + a$ (a обычно выбирается как 1).
3. Вычислить НОД $(x_1 - x_2, n)$. Если это не 1, результат – сомножитель. Алгоритм останавливается. Если это 1, то происходит возвращение, чтобы повторить процесс с x_1. Теперь мы вычисляем x_3. Заметим, что в следующем раунде мы начинаем с x_3 и так далее. Если мы перечислим значения нескольких x, используя PO (rho) алгоритм Полларда, мы увидим, что дуга значений в конечном счете повторяется, создавая форму, подобную греческой букве Po (rho или в греческом алфавите ρ), как это показано на рис. 9.3.

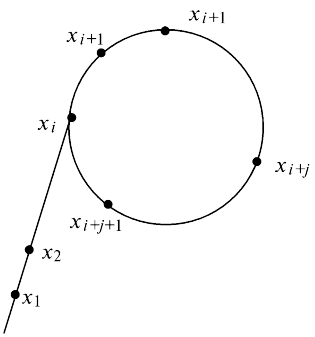


Рис. 9.3 Успешные числа в Po алгоритме Полларда

Чтобы уменьшить число итераций, алгоритм был немного изменен. Он начинается с пары (x_0, x_0), и итеративно вычисляет (x_1, x_2), (x_2, x_4), (x_3, x_6), ... (x_i, x_{2i}), используя равенство $x_{i+1} = f(x_i)$. В каждой итерации мы применяем функцию $f(x_i)$ (начиная с шага 2). При этом вычисление идут следующим образом: в паре вычисляется один раз первый элемент и дважды — второй элемент (см. алгоритм 9.6).

Алгоритм 9.6.

```
Pollard_rho_Factorization (n, B) // n – число, которое надо разложить
{
   x ← 2
   y ← 2
   p ← i
   while (p = 1)
   {
      x <- f(x) mod n
      y <- f(f(y) mod n) mod n  // gsd (a,b) – это НОД (a,b)
      p <- gcd (x - y, n)
   }
   return p      // если p = n, программа не выполнена
}
```

Комплексность. Метод требует $\sqrt{p}$ арифметических операций. Однако поскольку мы предполагаем, что p будет меньше или равняться $\sqrt{n}$, мы ожидаем около $n^{1/4}$ арифметические операций. Это означает, что сложность разрядной операции $O(2^{n_b/4})$ показательна.

Пример 9.32

Предположим, что есть компьютер, который, может выполнить 2^{30} (почти 1 миллиард) разрядных операций в секунду. Какое приблизительно время потребуется, чтобы разложить на множители целое число размера

a. 60 десятичных цифр

b. 100 десятичных цифр

Решение

a. Множество 60 десятичных цифр имеют почти 200 битов. Сложность — $2^{n_b/4}$ или 2^{50}. Со скоростью 2^{30} операций в секунду алгоритм может быть выполнен в 2^{20} секунды или почти за 12 дней

b. Множество 100 десятичных цифр — это почти 300 битов. Сложность — 2^{75}. Со скоростью 2^{30} операций в секунду алгоритм может быть выполнен в 2^{45} секунд или за много лет

Пример 9.33

Мы написали программу, чтобы вычислить разложение 434617. Результат — 709 ($434617 = 709 \times 613$) Таблица 9.2 показывает значения пар (x и y) и p в этом процессе.

Более эффективные методы

В течение прошлых десятилетий были предложены несколько методов разложения на множители, они кратко рассматриваются ниже.

Таблица 9.2. Значения *x*, *y* и *p* в примере 9.33

x	*y*	*p*
2	*2*	1
5	*26*	1
26	23713	1
677	142292	1
23713	157099	1
345589	52128	1
142292	41831	1
380320	68775	1
15 7099	427553	1
369457	2634	1
52128	63593	1
102901	161353	1
41831	64890	1
64520	21979	1
68775	16309	709

Квадратичное решето

Померанс изобрел метод разложения на множители, называемый **методом квадратичного решета.** Метод применяет процедуру просеивания, чтобы найти значение $x^2 mod\ n$. Метод используется, чтобы разложить на множители целые числа с более чем 100 цифрами. Его сложность — $0(e^c)$, где $C \approx 2(\ln n \ln\ln n)^{1/2}$. Обратите внимание, что это – субпотенциальная сложность[1].

Решето поля чисел

Эндрик Ленстра и Арджин Ленстра изобрели метод разложения на множители и назвали его **метод решета поля чисел.** Метод использует процедуру просеивания в алгебраической кольцевой структуре к $x^2 \equiv y^2 \bmod n$. Показано, что этот метод быстрее для разложения чисел с более чем 120 десятичными цифрами. Его сложность – $O(e^C)$ где $C \approx (\ln n)^{1/3}(\ln\ln n)^{2/3}$. Обратите внимание, что это — также субпоказательная сложность.

Пример 9.34

Предположим, что есть компьютер, который может выполнить 2^{30} (почти 1 миллиард) битовых операций в секунду. Какое приблизительно время требуется для этого компьютера, чтобы разложить на множители целое число из 100 десятичных цифр, используя один из следующих методов?

a. Метод квадратичного решета

b. Метод решета поля чисел

Решение

Номер с 100 десятичными цифрами имеет почти 300 битов ($n = 2^{300}$).

$\ln(2^{300}) = 207$ и $\ln\ln(2^{300}) = 5$.

a. Для метода квадратичного решета мы имеем $(207)^{1/2} \times (5)^{1/2} = 14 \times 2{,}23 = 32$.

[1] Субпотенциальная сложность — сложность меньше потенциальной.

Это означает, что нам надо e^{32} битовых операций, которые могут быть выполнены в $(e^{32})/(2^{30}) = 20$ часов.

b. При методе решета поля чисел мы имеем $(207) \times (5)^{2/2} = 6 \times 3 \approx 18$. Это означает, что нам надо e^{18} битовых операций, которые могут быть сделаны за $(e^{30}) / (2^{30}) \approx 6$ секунд. Однако эти результаты правильны, только если мы имеем компьютер, который может выполнить 1 миллиард битовых операций в секунду.

Другие проблемы

В лекции 10 мы обсудим прикладные вопросы задачи разложения на множители для вскрытия криптосистем с открытым ключом. Если будут изобретены более эффективные методы разложения на множители, то криптосистемы с открытым ключом вынуждены будут использовать большие целые числа, чтобы противостоять криптоанализу. Изобретатели RSA создали основу для конкуренции методов разложения на множители номеров до 2048 битов (больше чем 600 цифр).

9.4. Китайская теорема об остатках

Китайская теорема об остатках (CRT — Chinese Reminder Theorem) используется, чтобы решить множество уравнений с одной переменной, но различными взаимно простыми модулями, как это показано ниже:

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\ldots.$$
$$x \equiv a_k \pmod{m_k}$$

Китайская теорема об остатках утверждает, что вышеупомянутые уравнения имеют единственное решение, если модули являются взаимно простыми.

Пример 9.35

Следующий пример содержит систему уравнений с различными модулями:

$$x \equiv 2 \pmod{3}$$
$$x \equiv 3 \pmod{5}$$
$$x \equiv 2 \pmod{7}$$

Для этой системы уравнений $x = 23$. Это значение удовлетворяет все уравнения: $23 \times 2 \pmod 3$, $23 \times 3 \pmod 5$, $23 \times 2 \pmod 7$.

Решение

Решение системы уравнений выполняется в следующем порядке.

1. Найти $M = m_1 \times m_2 \times \ldots. \times m_k$. Это общий модуль.
2. Найти $M_1 = M/m_1$, $M_2 = M/m_2$,...., $M_k = M/m_k$.
3. Используя соответствующие модули m_1, m_2,...., m_k, найти мультипликативную инверсию M_1, M_2,..., M_k. Обозначим ее M_1^{-1}, M_2^{-1},..., M_k^{-1}.
4. Решение системы уравнений

$$x = (a_1 \times M_1 \times M_1^{-1} + a_2 \times M_2 \times M_2^{-1} + ... + a_k \times M_k \times M_k^{-1}) \bmod M$$

Обратите внимание, что система уравнений может иметь решение, даже если модули не взаимно простые. Однако в криптографии мы интересуемся только решением уравнений с взаимно простыми модулями.

Пример 9.36

Найдите решение системы уравнений

$$x \equiv 2 \pmod 3$$
$$x \equiv 3 \pmod 5$$
$$x \equiv 2 \pmod 7$$

Из предыдущего примера мы уже знаем, что ответ $x = 23$. Определим его в четыре шага.

Решение

1. $M = 3 \times 5 \times 7 = 105$
2. $M_1 = 105/3 = 35$, $M_2 = 105/5 = 21$, $M_3 = 105/7 = 15$
3. Инверсии $M_1^{-1} = 2$, $M_2^{-1} = 1$, $M_3^{-1} = 1$
4. $x = (2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1 = 23 \bmod 105$

Пример 9.37

Найти целое, которое дает в остатке 3, если его разделить на 7 и 13, но без остатка делится на 12.

Решение

Это проблема китайской теоремы об остатках. Мы можем составить три уравнения и найти значение x.

$$x \equiv 3 \pmod 7$$
$$x \equiv 3 \pmod{13}$$
$$x \equiv 0 \pmod{12}$$

Если проведем четыре шага, мы найдем $x = 276$. Можем проверить, что $276 = 3 \bmod 7$, $276 = 3 \bmod 13$ и 276 делится на 12 (частное 23 и остаток 0).

Приложения

Китайская теорема об остатках часто применяется в криптографии. Одно из таких применений – решение квадратных уравнений — будет обсуждаться в следующей секции. Другое приложение — представление очень большого числа в виде списка малых целых чисел.

Пример 9.38

Предположим, нам надо вычислить $z = x + y$, где $x = 123$ и $y = 334$, но система принимает только числа меньше 100. Эти числа можно представить следующими уравнениями:

$x \equiv 24 (\text{mod } 99)$		$y \equiv 37 (\text{mod } 99)$
$x \equiv 25 (\text{mod } 98)$		$y \equiv 40 (\text{mod } 98)$
$x \equiv 26 (\text{mod } 97)$		$y \equiv 43 (\text{mod } 97)$

Сложим каждое уравнение x с соответствующим уравнением y:

$x + y \equiv 61 (\text{mod } 99)$	$\rightarrow$	$z \equiv 61 (\text{mod } 99)$
$x + y \equiv 65 (\text{mod } 98)$	$\rightarrow$	$z \equiv 65 (\text{mod } 98)$
$x + y \equiv 69 (\text{mod } 97)$	$\rightarrow$	$z \equiv 69 (\text{mod } 97)$

Теперь эти три уравнения могут быть решены, с использованием китайской теоремы об остатках, чтобы найти z. Один из приемлемых ответов z = 457.

9.5. Квадратичное сравнение

Линейное сравнение уже рассматривалось в лекции 2, а китайская теорема об остатках была обсуждена в предыдущей секции. Для решения задач криптографии мы также должны уметь решать **квадратичное сравнение**, имеющее следующую форму: $a_2x^2 + a_1x + a_0 = 0 (\text{mod } n)$. Мы ограничим наше обсуждение только квадратичными уравнениями, в которых $a_2 = 1$ и $a_1 = 0$. Тогда рассмотрение будет касаться уравнений следующей формы:

$$x^2 \equiv a (\text{mod } n).$$

Квадратичное сравнение с модулем в виде простого числа

Мы сначала рассматриваем случай, в котором модуль является простым числом. Другими словами, мы хотим найти решения уравнения формы $x^2 \equiv a (\text{mod } p)$, в котором p является простым числом и a — целое число, такое, что p и a — взаимно простые. Может быть доказано, что этот тип уравнения либо не имеет никакого решения, либо имеет только два неконгруэнтных решения.

Пример 9.39

Уравнение $x^2 \equiv 3 (\text{mod } 11)$ имеет два решения: $x \equiv 5 (\text{mod } 11)$ и $x \equiv -5 (\text{mod } 11)$. Но заметим, что $-5 \equiv 6 (\text{mod } 11)$, так что фактически эти два решения 5 и 6. Также обратите внимание, что эти два решения неконгруэнтны (несравнимы).

Пример 9.40

Уравнение $x^2 \equiv 2 (\text{mod } 11)$ не имеет решения. Не может быть найдено ни одного целого числа x, такого, что квадрат равен 2 mod 11.

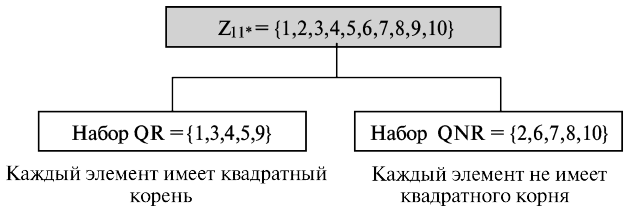
Квадратичные вычеты и невычет

В уравнении $x^2 \equiv a \pmod{p}$ a называется **квадратичным вычетом (QR)**, если уравнение имеет два решения; a называется **квадратичным невычетом (QNR)**, если уравнение не имеет решений. Может быть доказано, что в Z_{P^*} с $p - 1$ элементами $(p - 1)/2$ элементов — квадратичные вычеты и $(p - 1)/2$ являются квадратичными невычетами.

Пример 9.41

Есть 10 элементов в Z_{11^*}. Пять из них – квадратичные вычеты, и пять — невычеты. Другими словами, Z_{11^*} может быть разделен на два отдельных множества, QR и QNR, как это показано на рис. 9.4.

Критерий Эйлера



Как мы можем проверить, является ли целое число QR по модулю p? Критерий Эйлера дает признаки:

a. Если $a^{(p-1)/2} \equiv 1 \bmod p$ — квадратичный вычет по модулю p.

b. Если $a^{(p-1)/2} \equiv -1 \bmod p$ — квадратичный невычет по модулю p.

Рис. 9.4. Разделение Z_{11^*} на QR и QNR

Пример 9.42

Для того чтобы узнать, является ли 14 или 16 QR в Z_{11}*, сделаем следующие вычисления:

$14^{(23-1)./2} \bmod 23 \rightarrow 14^{11} \bmod 23 \rightarrow 22 \bmod 23 \rightarrow -1 \bmod 23$ **невычет**

$16^{(23-1)./2} \bmod 23 \rightarrow 16^{11} \bmod 23 \rightarrow 1 \bmod 23$ **вычет**

Решение квадратичного сравнения с модулем в виде простого числа

Хотя критерий Эйлера позволяет нам определить, является ли целое число a QR или QNR в Z_{p^*}, он не может найти решение $x^2 \equiv a \pmod{p}$. Чтобы найти решение этого квадратного уравнения, мы заметим, что простое число может быть представлено либо как $p = 4k + 1$, либо как $p = 4\kappa + 3$, в котором k является положительным целым числом. Решение квадратного уравнения — очень сложное в первом случае и более простое во втором. Мы обсудим только второй случай, который мы будем использовать в лекции 10, когда будем рассматривать криптографическую систему Рабина.

Специальный случай: $p = 4K + 3$, если p находится в форме $4K + 3$ (то есть $p = 3 \bmod 4$) и a есть QR в Z_p*, то

$$x \equiv a^{(p+1)/4} \pmod{p} \text{ и } x \equiv -a^{(p+1)/4} \pmod{p}$$

Пример 9.43

Решите следующие квадратные уравнения:

a. $x^2 \equiv 3 \pmod{23}$

b. $x^2 \equiv 2 \pmod{11}$

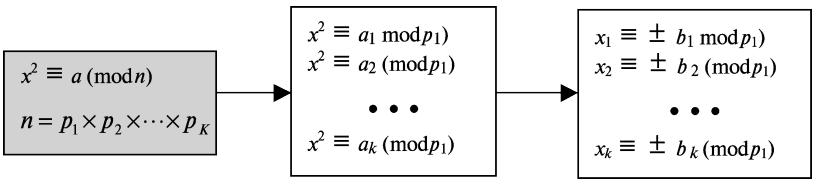
c. $x^2 \equiv 7 \pmod{19}$

Решения

a. В первом уравнении 3 – QR в Z_{23}, решение – $x \equiv \pm 16 \pmod{23}$. Другими словами, $\sqrt{3} \equiv \pm 16 \pmod{23}$.

b. Во втором уравнении 2 – QNR в Z_{11}. Нет решения для $\sqrt{2}$ в Z_{11}.

c. В третьем уравнении 7 – QR в Z_{19}, решение – $x \equiv \pm 11 \pmod{19}$. Другими словами $\sqrt{7} \equiv \pm 11 \pmod{19}$.



Квадратичное сравнение по составному модулю

Квадратичное сравнение по составному модулю может быть приведено к решению системы сравнений по модулю в виде простого числа. Другими словами, мы можем анализировать $x^2 \equiv a \pmod{n}$, если имеем разложение n на множители. Теперь мы можем решить каждое анализируемое уравнение (если оно разрешимо) и найти k пар ответов для x, как показано на рис. 9.5.

Рис. 9.5. Декомпозиция сравнения по составному модулю

Из k пар ответов мы можем составить 2 системы уравнений, которые могут быть решены с использованием китайской теоремы об остатках, чтобы найти 2 значения для x. В криптографии обычно n выбирают так, чтобы $n = p \times q$, — это означает $k = 2$, и мы имеем в целом только четыре ответа.

Пример 9.44

Предположим, что $x^2 \equiv 36 \pmod{77}$. Мы знаем, что $77 = 7 \times 11$. Мы можем написать

$x^2 = 36 \pmod{7} \equiv 1 \pmod{7}$ и $x^2 \equiv 36 \pmod{11} \equiv 3 \pmod{11}$

Обратите внимание, что мы выбрали 3 и 7, чтобы иметь форму $4k + 3$ — так, чтобы мы могли решить уравнения, основываясь на предыдущих рассуждениях. Из этих уравнений мы имеем квадратичные вычеты в собственном множестве. Ответы $x \equiv +1 \pmod{7}$, $x \equiv -1 \pmod{7}$, $x \equiv +5 \pmod{11}$ и $x \equiv -5 \pmod{11}$. Теперь мы можем из них составить четыре системы уравнений:

Система 1: $x \equiv +1 (\text{mod } 7)$ $x \equiv +5 (\text{mod } 11)$
Система 2: $x \equiv +1 (\text{mod } 7)$ $x \equiv -5 (\text{mod } 11)$
Система 3: $x \equiv -1 (\text{mod } 7)$ $x \equiv +5 (\text{mod } 11)$
Система 4: $x \equiv -1 (\text{mod } 7)$ $x \equiv -5 (\text{mod } 11)$

Ответы : $x = \pm 6$ и ± 27.

Сложность

Как сложно решить квадратичное сравнение по составному модулю? Главная задача — это разложение модуля на множители. Другими словами, сложность решения квадратичного сравнения по составному модулю — такая же, как и разложения на множители составного целого числа. Как мы видели раньше, если n очень большое, то разложение на множители неосуществимо.

Сложность решения квадратичного сравнения по составному модулю имеет ту же сложность, что и разложение модуля на множители.

9.6. Возведение в степень и логарифмы

Возведение в степень и логарифм инверсны друг другу. Следующие разделы показывают отношения между ними, в которых a называется основой возведения в степень или логарифма.

Возведение в степень: $y = a^x \rightarrow$ **логарифм:** $x = \log_a y$

Возведение в степень

В криптографии общая модульная операция — **возведение в степень.** Мы часто должны вычислять

$$y = a^x \bmod n$$

Криптографическая система RSA, которая будет обсуждаться в лекции 10, использует возведение в степень для шифрования и для дешифрования очень больших чисел. К сожалению, большинство компьютерных языков не имеет операторов, которые могут эффективно вычислять степень, особенно для очень больших чисел. Чтобы сделать эту операцию более эффективной при вычислении, мы нуждаемся в эффективных алгоритмах.

Быстрое возведение в степень

Быстрое возведение в степень возможно при использовании специальных **методов возведения в квадрат и умножения.** В традиционных алгоритмах, для возведения в степень применяется только *умножение*, но быстрый алгоритм возведения в степень использует и *возведение в квадрат*, и *умножение*. Главная идея этого

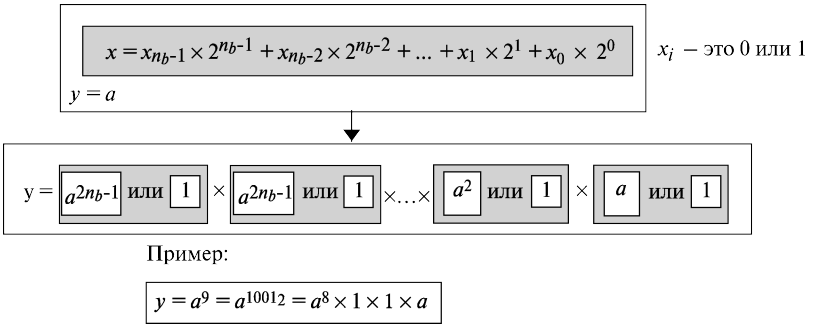


Рис. 9.6. Идея метода «возведения в квадрат и умножения»

метода — выполнение возведения в степень с помощью обработки двоичного числа с n_b битами (x_0 x_{nb-1}). Например, $x = 22 = (10110)_2$. Вообще, число x может быть записано как

$$x = x_{n_b-1} \times 2^{k-1} + x_{n_b-2} \times 2^{k-2} + ... + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$$

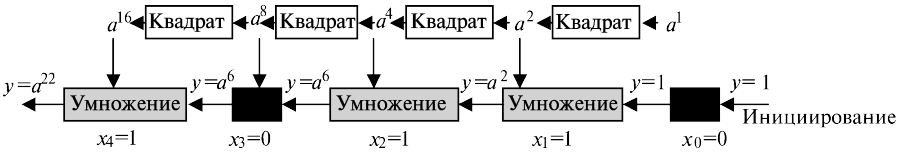
Теперь мы можем написать $y = a^x$, как это показано на рис. 9.6.

Обратите внимание, что y — произведение элементов числа n. Каждый элемент — либо 1 (если соответствующий бит — 0), либо a^{2i} (если соответствующий бит — 1). Другими словами, элемент участвует в умножении, если бит — 1, или не участвует, если бит — 0 (умножение на 1 не меняет числа). Рисунок 9.6 дает общую идею, как написать алгоритм. Мы можем непрерывно взводить в квадрат $a, a^2, a^4, ..., a^{2n_b-1}$. Если соответствующий бит — 0, элемент не участвует в процессе умножения; если бит — 1, то участвует. Алгоритм 9.7 отражает эти два свойства.

Алгоритм 9.7.

```
Возведение_в_ квадрат_и_умножение (a,x,n)
{
   y ← 1
   for (i ← 0 to nb - 1)     //nb - число бит в x
   {
      if (xi = 1) y ← a × y mod n     //умножение, только если бит 1
      a ← a2 mod n     //в последней итерации возведение в степень
                       //не нужно
   }
   return y
}
```

Алгоритм 9.7 использует n_b итерации. В каждой итерации он проверяет значение соответствующего бита. Если значение бита равно 1, он умножает текущее значение на предыдущее значение результата. Затем полученный результат явля-



ется базой для следующей итерации. Обратите внимание, что возведение в квадрат в последнем шаге не нужно (результат не используется).

Пример 9.45

Рисунок 9.7 показывает процесс для подсчета y = a^x с использованием алгоритма 9.7 (для простоты изображения модули не показаны). В этом случае x = 22 = $(10110)_2$. Это число имеет 5 бит.

Рис. 9.7. Демонстрация вычисления a^{22} с использованием метода «возведения в квадрат и умножения»

Возведение в квадрат делается на каждом шаге за исключением последнего. Умножение делается, если соответствующий бит равен 1. На рисунке 9.7 показано, как постепенно формируется значение *y* до значения $y = a^{22}$. Затемненный прямоугольник означает, что умножение не делается и предыдущее значение переносится на следующий шаг. Таблица 9.3 показывает, как вычисляется значение для y = 17^{22} mod 21. Результат — y = 4.

Таблица 9.3.

i	x_i	**Умножение (Инициализация *y* = 1)**		**Возведение в степень (Инициализация a = 17)**
0	0		→	a = 17^2 mod 21 =16
1	1	$y = 1 \times 16 \bmod 21 = 16$	→	a = 16^2 mod 21 = 4
2	1	$y = 16 \times 4 \bmod 21 = 1$	→	a = 4^2 mod 21 =16
3	0		→	a = 16^2 mod 21 = 4
4	1	$y = 1 \times 4 \bmod 21 = 4$	→	

Сложность. Алгоритм 9.7 использует максимально $2n_b$ арифметических операций, в которых n_b является длиной модуля в битах ($n_b = \log_2 n$). Сложность в битовых операциях алгоритма — $O(n_b)$ или полиномиальная сложность.

Сложность разрядной операции быстрого показательного алгоритма — полиномиальная.

Альтернативный алгоритм. Заметим, что алгоритм 9.7 проверяет значение битов в *x* справа налево (от самого младшего до самого старшего). Может быть написан другой алгоритм, чтобы использовать обратный порядок. Мы выбрали вышеупомянутый алгоритм, потому что операция возведения в квадрат полностью независима от операции умножения; они могут быть сделаны параллельно, чтобы

увеличить скорость обработки. Альтернативные алгоритмы оставляем как упражнение.

Логарифм

Мы также должны обсудить модульный логарифм, который используется в криптографии. Если мы применяем возведение в степень для того, чтобы зашифровать или расшифровывать сообщение, противник может использовать логарифм для раскрытия этого сообщения Мы должны знать, трудно ли получить операцию, противоположную возведению в степень.

Полный перебор

Первое решение, которое могло бы прийти на ум, для решения $x = \log_a y \pmod n$: мы можем написать алгоритм, который непрерывно вычисляет $y = a^x \bmod n$, пока не находит заданное значение y. Алгоритм 9.8 показывает этот подход.

Алгоритм 9.8. Алгоритм полного перебора для модульного логарифма

```
Modular_Logarithm (a, y, n)
    {
    for (x = 1 to n – 1)                    // k число бит в x
    {
        if (y = a^x mod n) return x
    }
    return failure
}
```

Алгоритм 9.8 явно очень неэффективен. Сложность разрядной операции — $\boldsymbol{O}\,(2^{n_b})$ или показательная.

Дискретный логарифм

Второй подход состоит в том, чтобы использовать понятие **дискретного логарифма**. Рассмотрение этого понятия требует изучения некоторых свойств мультипликативных групп.

Конечная мультипликативная группа. В криптографии мы часто используем мультипликативную конечную группу: $G = <Z_{n*}, \times>$, в которой применяемая операция является умножением. Множество Z_{n*} содержит целые числа от 1 до $n–1$, которые являются взаимно простыми с n, нейтральный элемент — $e = 1$. Обратите внимание, что когда модуль группы — простое число, мы имеем $G = <Z_{p*}, \times>$. Эта группа — специальный случай группы первого типа, так что мы концентрируемся на этой группе.

Порядок группы. В лекции 4 мы обсуждали порядок конечной группы $|G|$ для того, чтобы определить число элементов в группе G. Было доказано, что в группе

$G = <Z_n *, \times >$ порядок группы (число элементов) — число Эйлера $\phi(n)$. Мы показали, как вычислить $\phi(n)$, когда n может быть разложено на множители в виде простых чисел.

Пример 9.46

Найти порядок группы $G = <Z_{21} *, \times >$, $|G| = \phi(21) = \phi(3) \times \phi(7) = 2 \times 6 = 12$. В этой группе 12 элементов: 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19 и 20. Все — взаимно простые с 21.

Порядок элемента. В лекции 4 мы также обсуждали порядок элемента — ord (a). В $G = <Z_n *, \times >$, мы продолжаем определение. Порядок элемента a есть наименьшее целое число, такое, что $a^i \equiv e \pmod n$. В этом случае нейтральный элемент e равен 1.

Пример 9.47

Найдите порядок всех элементов в $G = <Z_{10}*, \times >$.

Решение

Эта группа имеет только $\phi(10) = 4$ элемента: 1, 3, 7, 9. Мы можем найти порядок каждого элемента методом «проб и ошибок». Однако, согласно результатам лекции 4, порядок элемента является делителем порядка группы (теорема Лагранжа). Целые числа, которые делят 4, — 1, 2 и 4. Это означает, что в каждом случае мы должны проверить только эти числа и найти порядок элемента,

a. $1^1 = 1 \bmod (10) \rightarrow \text{ord}(10) = 1$.

b. $3^1 = 3 \bmod (10)$; $3^2 \equiv 9 \bmod (10)$; $3^4 \equiv 1 \bmod (10) \rightarrow \text{ord}(3) = 4$.

c. $7^1 \equiv 7 \bmod (10)$; $7^2 \equiv 9 \bmod (10)$; $7^4 \equiv 1 \bmod (10) \rightarrow \text{ord}(7) = 4$.

d. $9^1 = 9 \bmod (10)$; $9^2 \equiv 1 \bmod (10) \rightarrow \text{ord}(9) = 2$.

Теорема Эйлера. Другая теорема, относящаяся к этому вопросу, — это теорема Эйлера, которая говорит, что если a является членом $G = <Z_n*, \times >$, то $a^{\phi(n)} = 1 \bmod n$;

Эта теорема очень полезна, потому что показывает, что равенство $a^i = 1 \bmod n$ сохраняется, если $i = \phi(n)$. Заметим: это не отрицает, что равенство может выполняться и если $i < \phi(n)$. Другими словами, отношение показывает, что равенство выполняется по меньшей мере однажды.

	i=1	*i=2*	*i=3*	*i=4*	*i=5*	*i=6*	*i=7*
a=1	*x:***1**	*x:*1	*x:*1	*x:*1	*x:*1	*x:*1	*x:*1
a=3	*x:*3	*x:***1**	*x:*3	*x:*1	*x:*3	*x:*1	*x:*3
a=5	*x:*5	***x:*1**	*x:*5	*x:*1	*x:*5	*x:*1	*x:*5
a=7	*x:*7	*x:***1**	*x:*7	*x:*1	*x:*7	*x:*1	*x:*7

Пример 9.48

Таблица 9.4 показывает результат $a^i = x \pmod 8$ для группы $G = <Z_8*, \times >$. Обратите внимание, что $\phi(8) = 4$. Это элементы — 1, 3, 5 и 7.

Таблица 9.4. Нахождение порядка элементов в примере 9.48

Таблица 9.4 демонстрирует некоторые моменты вычислений. Первый: затемненная область показывает результат применения теоремы Эйлера. Когда $i = \phi(8) = 4$, результат равен $x = 1$ для каждого a. Второй момент: таблица показывает, что значение 1 может быть получено для многих значений i — в первую очередь, значение i, равное порядку элемента (обведено в таблице жирной линией). Порядок элементов: ord (1) = 1, ord (3) = 2, ord (5) = 2 и ord (7) = 2.

Первообразные корни. Очень интересное понятие в мультипликативной группе — группы **первообразного корня**, которые используются в криптографической системе El Gamal (эль Гамаля) в лекции 10. В группе $G = <Z_{n^*}, \times>$, когда порядок *элемента* равен $\varphi(n)$, этот элемент называется первообразным корнем группы.

Пример 9.49

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
$a = 1$	x: 1	x: 1	x: 1	x: 1	x: 1	x: 1
$a = 2$	x: 2	x: 4	x: 1	x: 2	x: 4	x: 1
$a = 3$	x: 3	x: 2	x: 6	x: 4	x: 5	x: 1
$a = 4$	x: 4	x: 2	x: 1	x: 4	x: 2	x: 1
$a = 5$	x: 5	x: 4	x: 6	x: 2	x: 3	x: 1
$a = 6$	x: 6	x: 1	x: 6	x: 1	x: 6	x: 1

Таблица 9.4 показывает, что там нет первообразных корней $G = <Z_{8^*}, \times>$, потому что *ни один элемент* не имеет порядок, равный $\phi(8) = 4$. Порядок всех *элементов* — меньше, чем 4.

Пример 9.50

Таблица 9.5 показывает результат $a^i \equiv x \pmod 7$ для группы $G = <Z_7^*, \times>$. В этой группе $\phi(7) = 6$.

Таблица 9.5. Пример 9.50

Порядок элементов – ord (1) = 1, ord (2) = 3, ord (3) = **6**, ord (4) = 3, ord (5)= **6** и ord (6) = 1.

Таблица 9.5 показывает, что только два элемента, 3 и 5, имеют порядок в $i = \phi(n) = 6$. Поэтому эта группа имеет только два примитивных корня: 3 и 5.

Было доказано, что группа $G = <Z_{n^*}, \times>$ имеет примитивный корень, только если $n = 2$, 4, p^t, или $2p^t$, в которой p является нечетным простым числом (не 2) и t — целое число.

Группа $G = <Z_{n^*}, \times>$ имеет примитивные корни, только если n равно 2, 4, p^t или $2p^t$.

Пример 9.51

Для какого значения *n* группа $G = <Z_{n^*}, \times >$ имеет примитивные корни: 17, 20, 38 и 50?

Решение

a. $G = <Z_{17}^*, \times >$ имеет примитивные корни, потому что 17 — простое число (p^t, где t равно 1).

b. $G = <Z_{20}^*, \times >$ не имеет никаких примитивных корней.

c. $G = <Z_{38}^*, \times >$ имеет первообразные корни, потому что $38 = 2 \times 19$ и 19 — простое число.

d. $G = <Z_{50}^*, \times >$ имеет первообразные корни, потому что $50 = 2 \times 5^2$, а 5 — простое число.

Если группа имеет примитивный корень, то обычно таких корней несколько. Число примитивных корней может быть вычислено как — $\phi(\phi(n))$. Например, число примитивных корней $G = <Z_{17^*}, \times >$ — это $\phi(\phi(n)) = \phi(16) = 8$. Обращаем внимание, что нужно сначала проверить, имеет ли группа какой-либо примитивный корень, прежде чем находить число корней.

Если группа $G = <Z_{n^*}, \times >$ имеет хотя бы один примитивный корень, то число примитивных корней — $\phi(\phi(n))$.

Рассмотрим три вопроса.

1. Если дан элемент *a* и группа $G = <Z_{n^*}, \times >$, как можно определить, является ли *a* примитивным корнем G? Это не такая легкая задача.
   a. Мы должны найти $\phi(n)$, — эта задача по сложности подобна задаче разложения на множители числа *n*.
   б. Мы должны найти $\text{ord}(a) = \phi(n)$.
2. Если дана группа $G = <Z_{n^*}, \times >$, как найти все примитивные корни ϕ? Эта задача более трудная, чем первая задача, потому что мы должны повторить вычисления по п.1.*б* для всей группы.
3. Если дана группа $G = <Z_{n^*}, \times >$, то как выбирать примитивный корень G? В криптографии мы должны найти, по крайней мере, один примитивный корень в группе. Однако в этом случае значение n выбирается пользователем, и пользователь знает $\phi(n)$. Пользователь пробует последовательно несколько элементов, пока не находит первый из них.

Циклическая группа. Циклические группы уже обсуждались в лекции 4. Обратите внимание на то, что, если группа $G = <Z_{n^*}, \times >$ имеет примитивные корни, то они циклически повторяются. Каждый примитивный корень — генератор и может использоваться для создания целого набора. Другими словами, если g — примитивный корень в группе, мы можем генерировать набор Z_n^* как

$$Z_n^* = \{g^1, g^2, g^3 ..... g^{\phi(n)}\}$$

Пример 9.52

Группа $G = <Z_{10^*}, \times>$ имеет два примитивных корня, потому что $\phi(10) = 4$ и $\phi(\phi(10)) = 2$. Можно найти примитивные корни - это 3 и 7. Ниже показано, как можно создать целый набор Z_{10^*}, использующий каждый примитивный корень.

$g = 3 \rightarrow g^1 \bmod 10 = 3 \quad g^2 \bmod 10 = 9 \quad g^3 \bmod 10 = 7 \quad g^4 \bmod 10 = 1$
$g = 7 \rightarrow g^1 \bmod 10 = 7 \quad g^2 \bmod 10 = 9 \quad g^3 \bmod 10 = 3 \quad g^4 \bmod 10 = 1$

Обратите внимание, что группа $G(Z_{p^*}, \times)$ всегда циклическая, потому что p — простое.

Группа $G = <Z_n^*, \times>$ является циклической группой, если она имеет примитивные корни.

Группа $G = <Z_p^*, \times>$ всегда является циклической.

Идея дискретного логарифма. Группа $G = <Z_p^*, \times>$ имеет несколько интересных свойств.

1. Её элементы включают все целые числа от 1 до $p - 1$.
2. Она всегда имеет примитивные корни.
3. Она всегда является циклической. Элементы могут быть созданы с использованием g^x, где x — целое число от 1 до $\phi(n) = p-1$.
4. Примитивные корни можно представлять себе как основание логарифма. Если группа имеет k примитивных корней, то вычисления могут быть сделаны для k различных оснований. Данный $x = \log_g y$ для любого элемента y в данном множестве, но есть другой элемент x, который является логарифмом y по основанию g. Этот тип логарифма называют **дискретным логарифмом.** Дискретный логарифм в литературе определяется несколькими различными символами, но мы будем использовать обозначение L_g, чтобы показать, что основание — g (сравнение по модулю).

Решение модульного логарифма с использованием дискретных логарифмов

Теперь рассмотрим, как решаются задачи типа $y = a^x \ (mod\ n)$, т. е. дано y, а мы должны найти x.

Табулирование дискретных логарифмов. Один из способов решения вышеупомянутой проблемы — использовать таблицу для каждого Z_{p^*} и различных оснований. Этот тип таблицы может быть предварительно рассчитан и сохранен. Например, таблица 9.6 показывает значения дискретного логарифма для Z_{7^*}. Мы знаем, что есть два примитивных корня или основания в данном множестве.

Таблица 9.6. Дискретный логарифм для $G = <Z_p^*, \times>$

y	1	2	3	4	5	6
$x = L_3\, y$	6	2	1	4	5	3
$x = L_5\, y$	6	4	5	2	1	3

Составив таблицы для других дискретных логарифмов для всех групп и всех возможных оснований, мы можем решить любую дискретную логарифмическую

проблему. Этот подход подобен изучаемым в прошлом традиционным логарифмам. До появления калькуляторов и компьютеров таблицы использовались, чтобы вычислять логарифмы по основанию 10.

Пример 9.53

Найдите x в каждом из следующих случаев:

a. $4 \equiv 3^x \pmod 7$

b. $4 \equiv 3^x \pmod 7$

Решение

Мы можем легко применять таблицу 9.6 дискретного логарифма.

a. $4 \equiv 3^x \pmod 7 \quad \rightarrow \quad L_3\ 4 \pmod 7 = 4 \bmod 7$

b. $6 \equiv 5^x \pmod 7 \quad \rightarrow \quad L_5\ 6 \pmod 7 = 3 \bmod 7$

Использование свойств дискретных логарифмов. Чтобы показать, что дискретные логарифмы ведут себя точно так же, как традиционные логарифмы, в таблице 9.7 приводится несколько свойств обоих типов логарифмов. Обратите внимание, что основание модуля — $\phi(n)$ вместо n.

Таблица 9.7. Сравнение традиционных и дискретных логарифмов

Традиционные логарифмы	Дискретные логарифмы
$\log_a 1 = 0$	$L_g \equiv 0 \pmod{\phi(n)}$
$\log_a (x \times y) = \log_a x + \log_a y$	$L_g(x \times y) \equiv (L_g x + L_g y) \pmod{\phi(n)}$
$\log_a x^k = k \times \log_a x$	$L_g x \equiv k \times L_g x \pmod{\phi(n)}$

Использование алгоритмов, основанных на дискретных логарифмах. Таблицы и свойства дискретных логарифмов не могут применяться для решения уравнения $y = a^x \pmod n$, когда n является очень большим. Для решения этой проблемы были разработаны несколько алгоритмов, в которых используется основная идея дискретных логарифмов. Хотя все эти алгоритмы более эффективны, чем алгоритмы полного перебора, которые мы упоминали в начале этого раздела, но ни один из них не имеет полиномиальной сложности. Большинство этих алгоритмов имеет такой же уровень сложности, как проблема разложения на множители.

Проблема дискретного логарифма имеет такую же сложность, как проблема разложения на множители.

9.7. Рекомендованная литература

Нижеследующие книги и сайты дают более детальную информацию о предметах, рассмотренных в этой лекции. Пункты, приведенные в квадратных скобках, содержатся в списке в конце книги.

Книги

|Ros061 [Cou99], [BW00] и [Bla03] — для тем, которые обсуждаются в этой лекции.

Сайты

Нижеследующие сайты дают больше информации о темах, обсужденных в этой лекции.

http://en.wikipedia.org/wiki/Prime_number
http://primes.utm.edu/mersenne/
http://en.wikipedia.org/wiki/Primality__test
www..cl.cam.ac.uk/~jehl004/research/talks/miller-talk.pdf
http://mathworld.wolfram.com/TotientFunction.html
http: // en.wikipedia.org/wiki/Proofs_of_Fermat's_little_theorem
faculty.cs.tamu.edu/klappi/629/analytic.pdf

9.8. Итоги

- Положительные целые числа могут быть разделены на три группы: число 1, простые числа и составные объекты. Положительное целое число — простое число, такое и только такое, если оно точно делится без остатка на два различных целых числа, а именно на 1 и непосредственно само на себя. Составной объект — положительное целое число по крайней мере с двумя делителями.
- Эйлеровская phi-функция $\phi(n)$, которую иногда называют функцией-тотиентом Эйлера, играет очень важную роль в криптографии. Функция указывает число целых чисел, которые меньше чем n и являются взаимно простыми с n.
- В таблице 9.8 показаны малая теорема Ферма и теорема Эйлера, которые рассмотрены в этой лекции.

Таблица 9.8. Малая теорема Ферма и теорема Эйлера

Ферма	**Первая версия :** Если *НОД (a, p)* = 1, то $a^{p-1} \equiv 1 \pmod p$
	Вторая версия: $a^p \equiv a \pmod p$
Эйлер	**Первая версия:** Если *НОД (a,n)* = 1, то $a^{\phi(n)} = 1 \pmod n$
	Вторая версия: Если $n = p \times q$ и $a<n$, то $a^{k \times \phi(n)+1} \equiv a \pmod n$

- Чтобы получить большое простое число, мы выбираем большое случайное число и проверяем его — убеждаемся, что оно простое. Алгоритмы, которые решают эту проблему, могут быть разделены на две обширные

категории: детерминированные алгоритмы и вероятностные алгоритмы. Некоторые вероятностные алгоритмы для испытания простоты чисел — это испытание Ферма, испытание квадратного корня и испытание Миллера-Рабина. Некоторые детерминированные алгоритмы — испытание на делимости и AKS-алгоритм.

- Согласно *основной теореме арифметики*, любое положительное целое число, большее, чем 1, может быть разложено на множители в виде простых чисел. Мы рассмотрели несколько методов разложения на множители, включая проверку делением — Ферма, метод Полларда $p - 1$, метод PO (ρ) Полларда, квадратичное решето и решето поля чисел.
- Китайская теорема об остатках (Chinese Reminder Theorem — CRT) используется, чтобы решить систему уравнений для вычетов с одной переменной, но с различными взаимно простыми модулями.
- Мы рассмотрели решение квадратичного сравнения по модулю в виде простого числа и квадратичного сравнения по составному модулю. Однако, если модуль является большим, решение квадратичного сравнения по сложности одинаково с разложением модуля на множители.
- В криптографии применяется операция по модулю — возведение в степень. Для быстрого возведения в степень можно использовать метод «возведения в квадрат и умножения». Криптография также включает модульные логарифмы. Если возведение в степень применяется, чтобы зашифровать или расшифровывать информацию, то противник может использовать логарифмы для организации «атаки». Сложность операции, обратной возведению в степень, велика. Хотя возведение в степень может быть сделано с помощью быстрого алгоритма, применение модульного логарифма для больших значений модуля имеет ту же сложность, что и проблема разложения на множители.

9.9. Набор для практики

Обзорные вопросы

1. Объясните разницу между простым числом и составным целым числом.
2. Определите взаимно простые числа и их свойства.
3. Определите следующие функции и их приложения:
   a. $\pi(n)$ функция
   b. Функция (тотиент) Эйлера
4. Определите «решето Эратосфена» и объясните его приложения.
5. Определите малую теорему Ферма и объясните ее приложения.
6. Определите теорему Эйлера и объясните ее приложения.
7. Что такое простые числа Мерсенны? Что такое простые числа Ферма?
8. Объясните разницу между детерминированными и вероятностными алгоритмами для определения простых чисел.
9. Перечислите некоторые алгоритмы для разложения на множители простых чисел.
10. Определите китайскую теорему об остатках и ее приложения.

11. Определите квадратичное сравнение и важность вычетов (QRs) и невычетов (QNRs) в решении квадратных уравнений.
12. Определите дискретные логарифмы и объяснить их важность в решении логарифмических уравнений.

Упражнения

1. Используя аппроксимацию, найдите:
   a. число простых чисел между 100 000 и 200 000.
   b. число составных целых чисел между 100 000 и 200 000
   c. отношение простых чисел к составным в вышеупомянутом диапазоне и сравните это с тем же самым между 1 – 10.
2. Найдите наибольший простой сомножитель следующих составных целых чисел: 100, 1000, 10 000, 100 000 и 1 000 000. Также найти наибольший простой сомножитель 101, 1001, 10 001, 100 001 и 1 000 01.
3. Покажите, что каждое простое число может быть представлено в форме либо 4 k + 1, либо 4k + 3, где k — положительное целое число.
4. Найдите некоторые простые числа в форме 5 k + 1, 5 k + 2, 5 k + 3 и 5 k + 4, где k: является положительным целым числом.
5. Найдите значение. ϕ(29), ϕ (32), ϕ (80), ϕ (100) ϕ (101)
6. Покажите, что $2^{24} - 1$ и $2^{16} - 1$ – составные числа. Подсказка: используйте выражение $(a^2 - b^2)$.
7. Есть предположение, что каждое целое число, большее, чем 2, может быть представлено как сумма двух простых чисел. Проверьте это предположение для 10, 24, 28 и 100.
8. Есть предположение, что существует много простых чисел в форме n^2 + 1. Найдите некоторые из них.
9. Найдите результаты после использования малой теоремы Ферма:
   a. $5^{15} \bmod 13$
   b. $15^{18} \bmod 17$
   c. $456^{17} \bmod 17$
   d. 145 mod 101
10. Найдите, используя Малую теорему Ферма, результаты выражений, приведенных ниже:
    a. $5^{-1} \bmod 13$
    b. $15^{-1} \bmod 17$
    c. $27^{-1} \bmod 41$
    d. $70^{-1} \bmod 101$

    Обратите внимание, что все модули — простые числа.
11. Найдите, используя теорему Эйлера, результаты выражений, приведенных ниже:
    a. $12^{-1} \bmod 77$
    b. $16^{-1} \bmod 323$
    c. $20^{-1} \bmod 403$
    d. $44^{-1} \bmod 667$

Обратите внимание, что $77 = 7 \times 11$, $323 = 17 \times 19$, $403 = 31 \times 13$ и $667 = 23 \times 29$.

12. Определите, являются ли следующие числа Мерсенны простыми числами: M23, M29 и M31. Подсказка: любой делитель числа Мерсенны имеет форму $2kp + 1$.
13. Приведите некоторые примеры, чтобы показать, что если $2^n - 1$ — простое число, то n — простое число. Этот факт может использоваться для проверки на простоту? Объясните, как.
14. Определите, сколько из следующих целых чисел пройдут испытание Ферма на простоту чисел: 100, 110, 130, 150, 200, 250, 271, 341, 561. Используйте основание 2.
15. Определите, сколько из следующих целых чисел пройдут испытание Миллера-Рабина на простоту чисел: 100, 109, 201, 271, 341, 349. Используйте основание 2.
16. Используйте рекомендованное испытание, чтобы определить, является ли любое из следующих целых чисел простым числом: 271, 3149, 9673.
17. Используйте $a = 2$, $x = 3$ и несколько простых чисел, чтобы показать, что если p — простое число, то выполняется следующее сравнение: $(x - a)^p = (x^p - a) \pmod p$.
18. Говорят, что n-ное простое число может быть приближенно вычислено как $p_n = n \ln n$. Проверьте это на нескольких простых числах.
19. Найдите значение x для следующих наборов сравнений, используя китайскую теорему об остатках.
    a. $x \equiv 2 \bmod 7$, и $x \equiv 3 \bmod 9$
    b. $x \equiv 4 \bmod 5$, и $x \equiv 0 \bmod 11$
    c. $x \equiv 7 \bmod 13$, и $x \equiv 11 \bmod 12$
20. Найдите весь QRs и QNRs в Z_{13}^*, Z_{17}^* и Z_{23}^*.
21. Используя квадратичные вычеты, решите следующие сравнения:
    a. $x^2 \equiv 4 \bmod 7$
    b. $x^2 \equiv 5 \bmod 11$
    c. $x^2 \equiv 7 \bmod 13$
    d. $x^2 \equiv 12 \bmod 17$
22. Используя квадратичные вычеты, решите следующие сравнения:
    a. $x^2 \equiv 4 \bmod 14$
    b. $x^2 \equiv 5 \bmod 10$
    c. $x^2 \equiv 7 \bmod 33$
    d. $x^2 \equiv 12 \bmod 34$
23. Найдите результаты приведенных ниже выражений, используя метод «возведения в квадрат и умножения».
    a. $21^{24} \bmod 8$
    b. $320^{23} \bmod 461$
    c. $1736^{41} \bmod 2134$
    d. $2001^{35} \bmod 2000$
24. Для группы $G = <Z_{19}^*, \times>$:
    a. Найдите порядок группы
    b. Найдите порядок каждого элемента в группе

c. Найдите число первообразных корней в группе
d. Найдите первообразные корни в группе
e. Покажите, что группа является циклической
f. Составьте таблицу дискретных логарифмов

25. Используя свойства дискретных логарифмов, покажите, как решить сравнения:
a. $x^5 \equiv 11 \bmod 17$
b. $2x^{11} \equiv 22 \bmod 19$
c. $5x^{12} + 6x \equiv 8 \bmod 23$
26. Пусть мы имеем компьютер, выполняющий операции со скоростью 1 миллион бит в секунду. Вы хотите затратить только 1 час на испытание простоты чисел. Какое наибольшее число вы можете проверить, используя следующие методы, проверяющие простоту чисел?
a. теория делимости
b. AKS-алгоритм
c. Ферма
d. извлечением квадратного корня
e. Миллера-Рабина
27. Пусть мы имеем компьютер, выполняющий операции со скоростью 1 миллион бит в секунду. Вы хотите потратить только 1 час на разложение составного целого числа. Какое наибольшее число вы можете разложить на множители, используя следующие методы разложения на множители?
a. проверка делением
b. Ферма
c. Полларда (PO)
d. квадратичное решето
e. решето поля чисел
28. Метод «возведения в квадрат и умножения» — быстрый алгоритм возведения в степень — позволяет нам останавливать программу, если значение основания становится равным 1. Измените алгоритм 9.7, чтобы показать это.
29. Перепишите алгоритм 9.7, чтобы проверить биты в порядке от самого старшего к самому младшему.
30. Метод «возведения в квадрат и умножения» — быстрый алгоритм возведения в степень — может также быть спроектирован для проверки, является ли число четным или нечетным, вместо того чтобы проверять его разряды. Перепишите алгоритм 9.7, чтобы показать это.
31. Напишите алгоритм в псевдокоде для испытания простоты чисел по методу Ферма.
32. Напишите алгоритм в псевдокоде для испытания простоты чисел методом извлечения квадратного корня.
33. Напишите алгоритм в псевдокоде для китайской теоремы об остатках.
34. Напишите алгоритм в псевдокоде, чтобы найти вычет (QR) и невычет (QNR) для любого Z_p^*.
35. Напишите алгоритм в псевдокоде для нахождения первообразного корня для множества Z_p^*.

Лекция 10. Криптография с ассимметричным ключом

Цели и содержание

Эта лекция имеет несколько целей.

- Показать различия между симметричными и асимметрично-ключевыми криптографическими системами.
- Показать «лазейки» в односторонних функциях (trapdoor One Way Function) при их использовании в асимметрично-ключевых криптографических системах. «Лазейки» — это секретная информация, которая дает возможность простого обратного вычисления односторонней функции.
- Ввести ранцевую криптографическую систему как одну из первых идей в асимметрично-ключевой криптографии.
- Обсудить криптографическую систему RSA (RIVERST-SHAMIR-ADLEMAN).
- Обсудить криптографическую систему Рабина (Rabin).
- Обсудить криптографическую систему Эль-Гамаля (ElGamal).
- Обсудить криптосистему на основе метода эллиптических кривых.

В этой лекции рассматриваются несколько асимметрично-ключевых криптографических систем: RSA (RIVERST-SHAMIR-ADLEMAN), Рабина (Rabin), Эль-Гамаля (ElGamal), криптосистема на основе метода эллиптических кривых (ECC — Elliptic Curve Cryptosystem).

Обсуждение криптографической системы Диффи-Хеллмана (Diffie-Hellman) отложено до лекции 15, потому что это главным образом алгоритм защиты обмена ключами, а не алгоритм шифрования/дешифрования.

Криптографическая система Диффи-Хелмана обсуждается в лекции 15.

10.1. Введение

В лекциях 2-8 мы показали принципы **криптографии с симметричными ключами.** В этой лекции мы начинаем обсуждение **асимметрично-ключевой криптографии.** Симметричная и асимметрично-ключевая криптографии будут существовать параллельно и продолжать обслуживать общество. Мы верим, что они дополняют друг друга; преимущества одной компенсируют недостатки другой.

Концептуальные отличия между этими двумя системами базируются на том, как эти системы сохраняют секретность. В криптографии с симметричными ключами задача секретности должна быть разделена между двумя людьми. В асимметрично-ключевой криптографии секретность — персональная задача (неразделенная); человек создает и сохраняет свою собственную тайну.

В сообществе n людей при криптографии с симметричными ключами для сохранения секретности требуется $n\,(n-1)/2$ общедоступных ключей. В асимметрично-ключевой криптографии необходимы только n персональных ключей.

Сообщество с количеством участников (10^6) при криптографии с симметричными ключами требовало бы 5×10^{11} общедоступных ключей; асимметрично-ключевая криптография требовала бы 1 миллион персональных ключей.

Криптография с симметричными ключами базируется на совместном использовании ключей; асимметрично-ключевая криптография базируется на персональном ключе.

Есть некоторые другие аспекты безопасности помимо шифрования, которые присущи асимметрично-ключевой криптографии. Они включают установление подлинности и цифровые подписи. Всякий раз, когда приложение базируется на персональной тайне, мы должны использовать асимметрично-ключевую криптографию.

Обратим внимание на то, что криптография с симметричными ключами базируется на подстановке и перестановке символов (символов или бит), а асимметрично-ключевая криптография — на применении математических функций к числам. В криптографии с симметричными ключами исходный текст и зашифрованный текст представляют как комбинацию символов. Шифрование и дешифрование здесь — это перестановка этих символов или замена одного символа другим. В асимметрично-ключевой криптографии исходный текст и зашифрованный текст – числа; их шифрование и дешифрование — это математические функции, которые применяются к числам, чтобы создать другие числа.

В криптографии с симметричными ключами символы переставляются или заменяются другими; в асимметрично-ключевой криптографии числа преобразуются с помощью математических функций.

Ключи

Асимметричная ключевая криптография использует два отдельных ключа: один секретный (частный) и один открытый (общедоступный); шифрование и дешифрование представляют собой процесс запирания и отпирания замков ключами. В этом случае замок, запертый открытым ключом доступа, можно отпереть только с соответствующим секретным ключом. Рисунок 10.1 показывает, что если Алиса запирает замок открытым ключом доступа Боба, то только секретный ключ Боба может отпереть его.

Общая идея

Рисунок 10.2 показывает общую идею асимметрично-ключевой криптографии при использовании для шифрования. В будущих лекциях мы увидим другие приложения асимметрично-ключевой криптографии. Как показывают рисунки, в отличие от криптографии с симметричными ключами при асимметрично-ключевой криптографии ключи отличаются: существует **секретный ключ** и **открытый**

ключи доступа. Хотя в некоторых книгах используют термин *ключ засекречивания* вместо термина *секретный ключ*, мы будем пользоваться термином *ключ засекречивания* только для системы с симметричными ключами и термином *секретный ключ* и *открытый ключ доступа* — для асимметрично-ключевой криптографии. Мы даже применим различные символы, чтобы показать три различных типа ключей. Одна из причин разницы в терминах — характер *ключа засекречивания*, используемого в криптографии с симметричными ключами, отличается от характера *секретного ключа*, применяемого в асимметрично-ключевой криптографии.

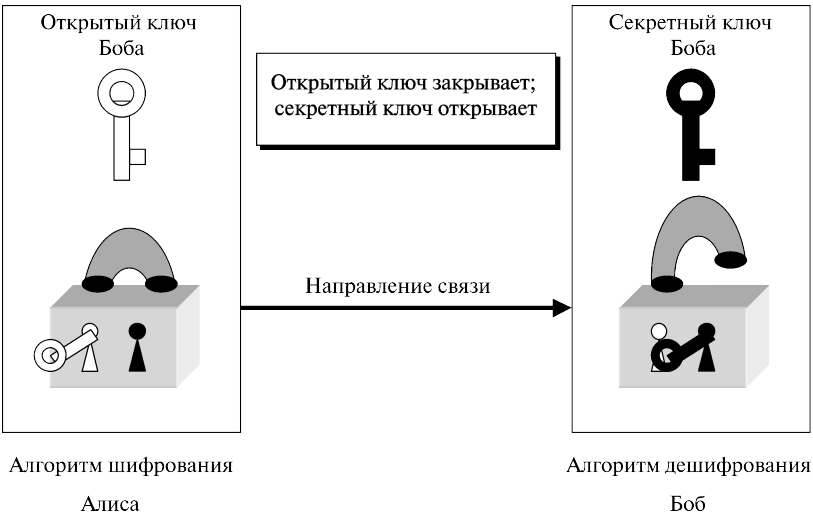


Рис.10.1. Закрытие и открытие в асимметрично-ключевой криптосистеме

Первый ключ работает обычно со строкой символов (например, биты), второй — с числами или множеством чисел. Другими словами, мы хотим показать, что *ключ засекречивания* не является взаимозаменяемым *с секретным ключом;* это два различных типа секретности.

Рисунок 10.2 иллюстрирует несколько важных фактов.

Первый: подчеркивает асимметричный характер криптографической системы. Ответственность за обеспечение безопасности находится, главным образом, на плечах приемника (в данном случае это Боб). Боб должен создать два ключа: один секретный (частный) и один открытый (общедоступный). Боб не несет ответственность за распределение открытого ключа доступа всему сообществу. Это может быть сделано через канал распределения открытого ключа доступа. Хотя этот канал не обязан обеспечивать секретность, он должен обеспечить установление подлинности и целостность информации о ключе. Ева не должна иметь возможности распространять свой открытый ключ сообществу, представляя его как открытый ключ доступа Боба. Проблемы распределения открытого ключа доступа обсуждаются в лекции 15. На данный момент мы принимаем, что такой канал существует.

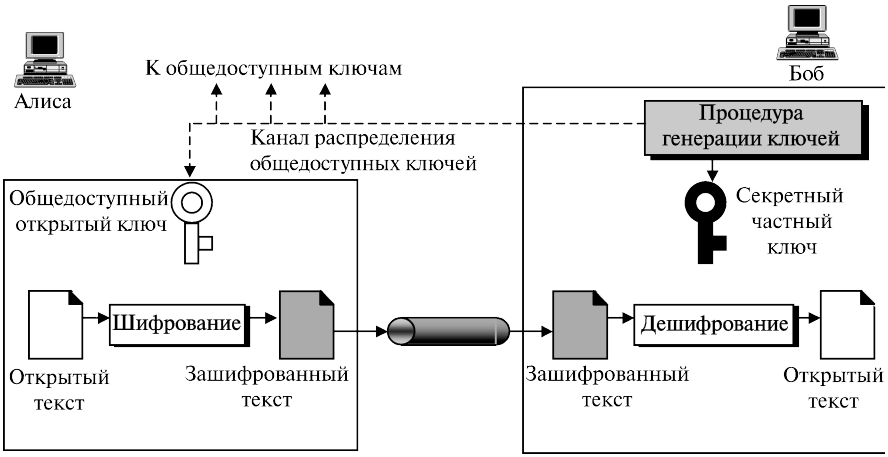


Рис. 10.2. Общая идея асимметрично-ключевой криптосистемы

Второй факт: асимметрично-ключевая криптография означает, что Боб и Алиса не могут использовать одно и то же множество ключей для двухсторонней связи. Каждый объект в сообществе создает свой собственный секретный и открытый ключи доступа. Рисунок 10.2 показывает, как Алиса может использовать открытый ключ доступа Боба, чтобы передать Бобу зашифрованные сообщения. Если Боб хочет ответить, Алиса устанавливает свои собственные секретный и открытый ключи доступа.

Третий: асимметрично-ключевая криптография означает, что Боб нуждается только в одном секретном ключе, чтобы получать всю корреспонденцию от любого участника сообщества. Алиса нуждается в ключах, чтобы связаться с *n* объектами в сообществе — один ключ доступа для каждого. Другими словами, Алиса нуждается в кольце ключей доступа.

Исходный текст / зашифрованный текст

В отличие от криптографии с симметричными ключами, в асимметрично-ключевой криптографии исходный текст и зашифрованный текст обрабатываются как целые числа. Сообщение должно перед шифрованием кодироваться как целое число (или множество целых чисел). После дешифрования оно должно быть расшифровано как целое число (или множество целых чисел). Асимметрично-ключевая криптография обычно зашифровывает или расшифровывает маленькие части информации, определяемые длиной ключа шифра. Другими словами, асимметрично-ключевая криптография обычно имеет вспомогательные цели помимо шифровки сообщения. Однако эти вспомогательные цели сегодня играют в криптографии очень важную роль.

Шифрование/дешифрование

Шифрование и дешифрование в асимметрично-ключевой криптографии — математические функции, которые применяются к числам, представляющим

исходный текст и зашифрованный текст. Зашифрованный текст можно представлять себе как C = *f* (K public, P). Исходный текст можно представлять себе как P = *g* (K private, C). Функция *f* шифрования используется только для шифрования; функция дешифрования *g* используется для дешифрования. Далее мы покажем, что функция *f* нуждается *в «лазейке» односторонней функции*, чтобы позволить Бобу расшифровывать сообщение, но препятствовать Еве делать то же самое.

Потребность в обеих криптосистемах

Есть очень важный факт, который иногда неправильно истолковывается. Появление асимметрично-ключевой криптографии (открытый ключ доступа) не устраняет потребность в криптографии с симметричными ключами (ключ засекречивания). Причина в том, что криптография с асимметричными ключами использует математические функции для шифрования и дешифрования намного медленнее, чем криптография с симметричными ключами. Для шифровки больших сообщений криптография с симметричными ключами необходима. С другой стороны, скорость криптографии с симметричными ключами не устраняет потребность в асимметрично-ключевой криптографии. Асимметрично-ключевая криптография необходима для установления подлинности цифровых подписей и работы станций по рассылке ключей засекречивания. Это означает способность системы использовать все аспекты безопасности. Сегодня мы нуждаемся в обеих системах криптографии. Одна криптосистема дополняет другую.

«Лазейка» в односторонней функции

Главная идея асимметрично-ключевой криптографии — понятие «лазейки» в односторонней функции.

Функции

Хотя понятие функции знакомо из математики, мы дадим неофициальное определение здесь. **Функция** — правило, по которому связывают (отображают) один элемент во множестве A, называемый доменом, и один элемент во множестве B, называемый диапазоном, как показано на рис. 10.3.

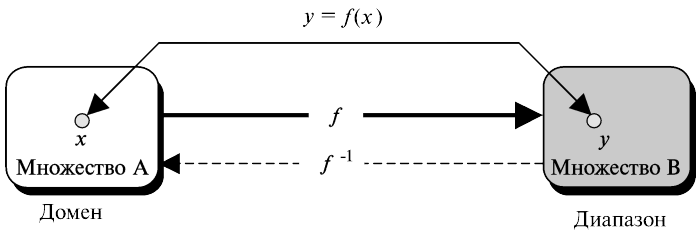


Рис. 10.3. Функция отображения домена в диапазон

Обратимая функция — функция, которая связывает каждый элемент в диапазоне с точно одним элементом в домене.

Односторонняя функция (OWF — One Way Function) — функция, которая обладает следующими двумя свойствами:

1. f вычисляется просто. Другими словами, при данном x может быть легко вычислен $y = f(x)$.
2. f^{-1} вычисляется трудно. Другими словами, при данном y, вычислить $x = f^{-1}(y)$ неосуществимо.

«Лазейка» в односторонней функции

«Лазейка» в односторонней функции (TOWF — Trapdoor One Way) — односторонняя функция с третьим свойством:

3. При данном y и ловушке (секретной) x может быть легко вычислен.

Пример 10.1

Когда n является большим, $n = p \times q$ — односторонняя функция. Обратите внимание, что в этой функции x — кортеж[1] (p, q) двух простых чисел, а y в данном случае — это n. При заданных p и q всегда просто вычислить n. При данном n очень трудно вычислить p и q. Это — *проблема разложения на множители,* которую мы рассматривали в Лекции 9. В этом случае для нахождения функции f^{-1} нет решения с полиномиальным временем.

Пример 10.2

Когда n является большим, функция $y = x^k \bmod n$ — «лазейка» в односторонней функции. При заданных x, k и n просто вычислить y, применяя алгоритм быстрого возведения в степень, который мы обсуждали в лекции 9. При заданных y, k и n очень трудно вычислить x. Это — *проблема дискретного логарифма*, которую мы обсуждали в лекции 9. В этом случае нет решения с полиномиальным временем для функции f^{-1}. Однако если мы знаем «лазейку» и k', такое, что $k \times k' = 1 \bmod \phi(n)$, мы можем использовать $x = y^k \bmod n$, чтобы найти x. Это — известный алгоритм (RSA — Riverst-Shamir-Adelman), который будет рассмотрен позже в этой лекции.

Ранцевая криптосистема,

Первая блестящая идея относительно криптографии открытого ключа доступа принадлежит Меркелю и Хеллману — она изложена в их **ранцевой криптосистеме.** Хотя эта система ненадежна в соответствии с сегодняшними стандартами, главная ее идея дает возможность понять современные криптосистемы с открытым ключом, которые будут рассматриваться позже в этой лекции.

Если нам говорят, какие элементы заранее определенного множества чисел мы имеем, то можно легко вычислить сумму чисел. Если нам говорят сумму, то трудно сказать, какие элементы «находятся в ранце».

[1] Кортеж — последовательность конечного числа элементов.

Определение

Предположим, что нам даны два k-кортежа, $a = [a_1, a_2, ..., a_k]$ и $x = [x_1, x_2, ..., x_k]$. Первый кортеж — заранее определенное множество; второй кортеж, в котором x равен только 0 или 1, определяет, какие элементы a должны быть отброшены в ранце. Сумма элементов в ранце равна

$$s = knapsackSum\ (a, x) = x_1 a_1 + x_2 a_2 + ... + x_k a_k$$

По данным a и x просто вычислить s. Однако по данному s трудно найти x. Другими словами, $s = knapsackSum\ (x, a)$ вычисляется просто, но $x = inv\_knapsackSum\ (s, a)$ труден. Функция *knapsackSum* — односторонняя функция, если a – общий k-кортеж.

Суперувеличение кортежа

Просто вычислить *knapsackSum* и *inv_knapsackSum*, если k-кортеж *суперувеличивается*. В **суперувеличивающемся кортеже** $a_i \geq a_1 + a_2 + ... + a_{i-1}$. Другими словами, каждый элемент (кроме a_1) больше или равен сумме всех предыдущих элементов. В этом случае мы вычисляем *knapsackSum* и *inv_knapsackSum*, как показано в Алгоритме 10.1. Алгоритм *inv_knapsackSum* запускается от наибольшего элемента и продолжает процесс к наименьшему. В каждой итерации он проверяет, находится ли элемент в рюкзаке.

Алгоритм 10.1. *knapsacksum* и *inv_knapsackSum* для суперувеличивающегося k-кортежа

```
knapsackSum (x[1, ..., k], a[1, ..., k] )
{
   s ← 0
   for (i=1 to k)
   {
      s ← s + a_i × x_i
   }
   return x
}
```

```
Inv_knapsackSum (s, a[1, ..., k])
{
    for (i=k down to 1)
   {
     if s ≥ a_j
     {
         x_i ← 1
         s ← s - a_i
      }
      else x_i ← 0
    }
    return x[1, ..., k]
}
```

Пример 10.3

Как очень тривиальный пример, предположим, что даны $a = [17, 25, 46, 94, 201, 400]$ и $s = 272$. Таблица 10.1 показывает, как найти кортеж, используя процедуру *inv_knapsackSum* в алгоритме 10.1.

Таблица 10.1. Значения *i, a* и *x* в примере 10.3

i	a_i	s	$s \geq a_i$	x_i	$s \leftarrow s - a_i \times x_i$
6	400	272	false	$x_6 = 0$	272
5	201	272	true	$x_5 = 1$	71
4	94	71	false	$x_4 = 0$	71
3	46	71	true	$x_3 = 1$	25
2	25	25	true	$x_2 = 1$	0
1	17	0	false	$x_1 = 0$	0

В этом случае $x = [0, 1, 1,0, 1,0]$, — это означает, что в рюкзаке находятся 25, 46 и 201.

Секретная связь с использованием ранца

Посмотрим, как Алиса может передать секретное сообщение Бобу, использующему ранцевую криптосистему. Идея показана на рис. 10.4.

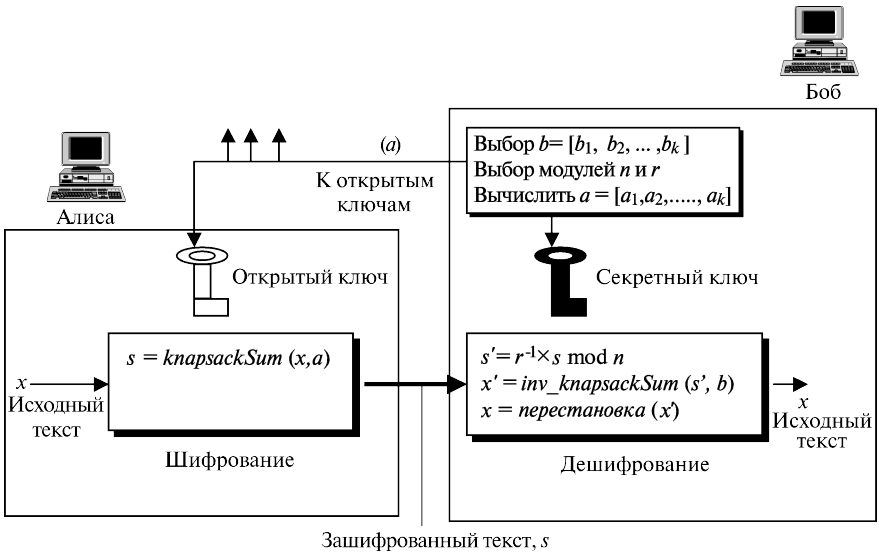


Рис. 10.4. Секретная связь с использованием ранцевой криптосистемы

Генерация ключей

Этот процесс:

а. Создает суперувеличивающийся k-кортеж $b = [b_1, b_2,..., b_k]$.

б. Выбирает модуль n, такой, что $n > b_1 + b_2 + ... + b_k$.

в. Выбирает случайное целое число r, которое является взаимно простым с n и $1 \leq r \leq n - 1$.

г. Создает временный k-кортеж $t = [t_1, t_2,..... t_k]$, в котором $t_i = r \times b_i \bmod n$.

д. Выбирает перестановку k-объектов и находит новый кортеж $a = permute(t)$.
е. Открытый ключ k-кортежа — a. Секретный ключ — n, r и k-кортеж b.

Шифрация

Предположим, что Алисе надо послать сообщение Бобу.

а. Алиса преобразует свое сообщение в k-кортеж $x=[x_1, x_2,.... x_k]$, в котором x_i — либо 0, либо 1. Кортеж представляет собой исходный текст.
б. Алиса использует *knapsackSum для* вычисления s в качестве исходного текста.

Дешифрация

а. Боб получает зашифрованный текст s.
б. Боб вычисляет $s' = r^{-1} \times s \bmod n$.
в. Боб переставляет x' для того, чтобы найти x. Кортеж x есть восстановленный исходный текст.

Пример 10.4

Это тривиальный (очень легко раскрываемый пример). Он приводится только для того, чтобы показать процедуру.

1. Генерация ключей:
   a. Боб создает суперувеличивающийся кортеж b = [7, 11, 19, 39, 79, 157, 313].
   b. Боб выбирает модуль n = 900 и r = 37, и [4 2 5 3 1 7 6] как таблицу перестановок.
   c. Боб теперь вычисляет кортеж t = [259, 407, 703, 543, 223, 409, 781].
   d. Боб теперь вычисляет кортеж a = *перестановка* (t)= [543, 407, 223, 703, 259, 781, 409].
   e. Боб объявляет a; он сохраняет в тайне n, r и b.
2. Предположим, что Алиса хочет передать единственный символ «g» Бобу.
   a. Она использует представление ASCII на 7 битов «g», $(1100111)_2$ и создает кортеж x = [1,1,0,0, 1, 1, 1]. Это — исходный текст.
   b. Алиса вычисляет s = *knupsackSum* (a, x) = 2165. Это — зашифрованный текст, передаваемый Бобу.
3. Боб может расшифровать зашифрованный текст, s = 2165.
   a. Боб вычисляет $s' = s \times r^{-1} \bmod n = 2165 \times 37^{-1} \bmod 900 = 527$.
   b. Боб вычисляет x' = *inv_knapsackSum* (s', b) = [1, 1, 0, 1, 0, 1, 1].
   c. Боб вычисляет x = *перестановка* (x') = [1, 1, 0, 0, 1, 1, 1]. Он интерпретирует строку $(1100111)_2$ как символ «g».

Лазейка

Вычисление суммы элементов в ранце Алисы — фактически умножение матрицы-строки x на матрицу-столбец a. Результат — матрица s 1 × 1. Матричное умножение: $s = \boldsymbol{x} \times \mathbf{a}$, в котором x является матрицей-строкой, а a — матрица-столбец — односторонняя функция. По данным $\boldsymbol{s}$ и $\boldsymbol{x}$ Ева не сможет легко найти a. Боб, однако, имеет лазейку. Боб использует его $s' = r^{-1} \times s$ и секретную, суперувеличивающуюся матрицу-столбец b, чтобы найти матрицу-строку x. При этом он

применяет процедуру *inv_knapsackSum*. Перестановка позволяет Бобу найти *x* по известному *x*'.

10.2. Криптографическая система RSA

Самый общий алгоритм открытого ключа доступа — **криптографическая система RSA,** названная по имени его изобретателей Ривеста, Шамира, Эделмана (Rivest, Shamir и Adelman).

Введение

RSA применяет два типа ключей — *e* и *d,* где *e* — открытый, а *d* — секретный. Предположим, что P — исходный текст и C — зашифрованный текст. Алиса использует $C = P^e \bmod n$, чтобы создать зашифрованный текст C из исходного текста P; Боб использует $P = C^d \bmod n$, чтобы извлечь исходный текст (файл), переданный Алисой. Модулей n создается очень большое количество с помощью процесса генерации ключей, который мы обсудим позже.

Для шифрования и дешифрования применяют возведение в степень по модулю. Как мы уже обсуждали в лекции 9, при использовании быстрого алгоритма возведение в степень по модулю выполнимо в полиномиальное время. Однако нахождение модульного логарифма так же сложно, как и разложение числа по модулю. Для него нет алгоритма с полиномиальным временем. Это означает, что Алиса может зашифровать сообщение общедоступным ключом (*e*) в полиномиальное время. Боб также может расшифровать его в полиномиальное время (потому что он знает *d*). Но Ева не может расшифровать это сообщение, потому что она должна была бы вычислить корень *e*-той степени из C с использованием модульной арифметики. Рисунок 10.5 показывает идею RSA.

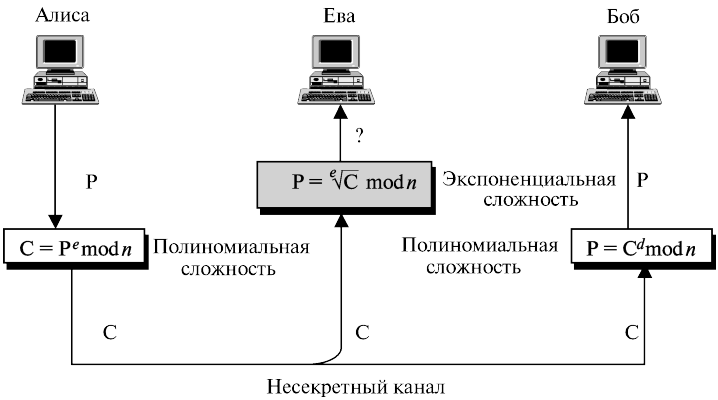


Рис. 10.5. Сложность операций в RSA

Другими словами, Алиса применяет одностороннюю функцию (возведение в степень по модулю) с лазейкой, известной только Бобу. Ева не знает лазейку, по-

этому не может расшифровать сообщение. Если когда-нибудь найдут полиномиальный алгоритм для модуля вычисления корня *e*-той степени из *n*, то возведение в степень по модулю *n* не будет больше односторонней функцией.

Процедура

Рисунок 10.6 показывает общую идею процедуры, используемой в RSA.

RSA использует возведение в степень по модулю для шифрования/дешифрования. Для того чтобы атаковать закрытый текст, Ева должна вычислить $\sqrt[e]{C} \bmod n$.

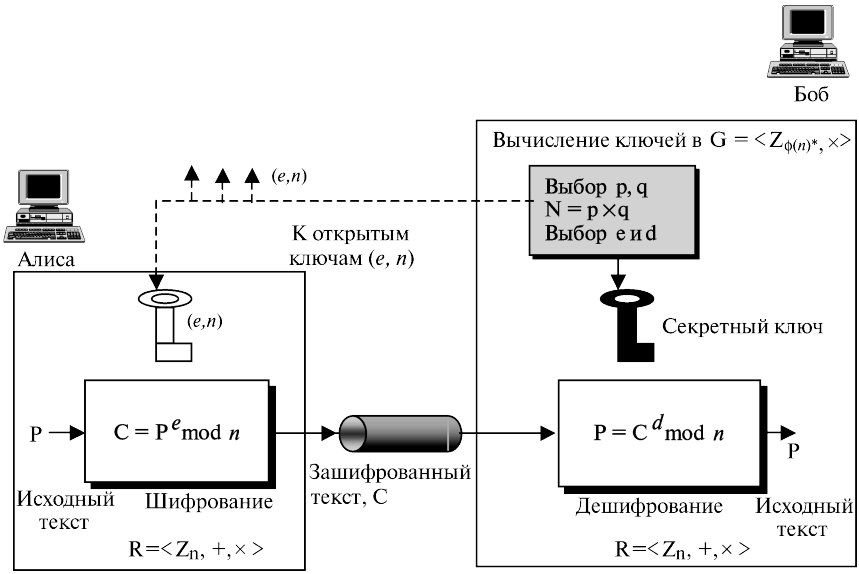


Рис. 10.6. Шифрование, дешифрование и генерация ключей в RSA

Две алгебраические структуры

RSA задействует две алгебраических структуры: кольцо и группу.

Кольца шифрования/дешифрования. Шифрование и дешифрование сделаны с использованием коммутативного кольца $R = <Z_n, +, \times >$ с двумя арифметическими операциями: сложение и умножение. В RSA это кольцо общедоступно, потому что модуль *n* общедоступен. Любой может послать сообщение Бобу, применяя это кольцо для шифрования.

Группы генерирования ключей. RSA использует мультипликативную группу $G = <Z_{\phi(n)}^*, \times >$ для генерации ключей. Группа поддерживает только умножение и деление (мультипликативную инверсию), которые необходимы для того, чтобы создать открытые и секретные ключи. Эту группу надо скрыть, потому что ее модуль $\phi(n)$ является секретным. Мы увидим, что если Ева найдет этот модуль, она сможет легко атаковать криптографическую систему.

RSA использует две алгебраических структуры:
открытое кольцо $R = <Z_n, +, \times>$ и секретную группу $G = <Z_{\phi(n)^*}, \times>$.

Генерация ключей

Боб использует шаги, показанные в алгоритме 10.2, чтобы создать свои открытый и секретный ключи. После генерации ключей Боб объявляет кортеж (*e, n*) как свой открытый ключ доступа: Боб сохраняет *d* как свой секретный ключ. Боб может отказаться от *p*, *q* и φ(*n*); они не могут изменить его секретный ключ, не изменяя модуль. Для безопасности рекомендуется размер для каждого простого p или q — 512 бит (почти 154 десятичные цифры). Это определяет размер модуля, n 1024 бита (309 цифр).

Алгоритм 10.2. RSA-генерация ключей

```
RSA Key_Generation (RSA- генерация ключа)
{
   Выбрать два больших простых p and q, таких, что p ≠ q.
   n ← p x q
   ϕ(n) ← (p-1) x (q -1)
   Выбрать e, такое, что 1 < e < ϕ(n) и e – взаимно простое с ϕ(n)
   d ← e^-1 mod φ (n)              // d - это инверсия e по модулю ϕ(n)
   Открытый ключ ← (e, n)          // Объявляется открытым
   Секретный ключ ← d              // Сохраняется в секрете
   return Public_key and Private_key    // Возврат открытого и секретного
                                        // ключей
}
```

В RSA кортеж *(e, n)* — открытый ключ доступа;
целое число d — секретный ключ.

Шифрование

Передать сообщение Бобу может любой, используя его открытый ключ доступа. Шифрование в RSA может быть выполнено с использованием алгоритма с полиномиальной сложностью по времени, как показано в алгоритме 10.3. Быстрый алгоритм возведения в степень был рассмотрен в лекции 9. Размер исходного текста должен быть меньше чем *n*; если размер исходного текста больше, то он должен быть разделен на блоки.

Алгоритм 10.3. Шифрование RSA

```
RSA_Encryption (P, e, n)        // P — исходный текст в Zn и P < n
{
C ← Fast_Exponentiation (P, e, n)        // Вычисление (P^e mod n}
return C
}
```

Дешифрование

Чтобы расшифровать сообщение зашифрованного текста, которое Боб получил в RSA, он может использовать алгоритм 10.4. Нам пригодится алгоритм с полиномиальной сложностью по времени, если размер зашифрованного текста меньше, чем *n*.

Алгоритм 10.4. Дешифрование RSA

```
RSA_Decryption (C, d, n)                //C – зашифрованный текст в Zn
{
   P <- Fast_Exponentiation (C, d, n)  // Вычисление (Cd mod n)
   return P
}
```

В RSA *p* и *q* должны быть по крайней мере 512 битов; *n* должны быть по крайней мере 1024 бит.

Доказательство RSA

Используя вторую версию теоремы Эйлера, которая обсуждалась в лекции 9, мы можем доказать, что шифрование и дешифрование инверсны друг другу.

Если n =: $p \times q$, <*n,* и k — целое число, тогда $a^{k \times \phi(n)+1} \equiv a \pmod{n}$.

Предположим, что исходный текст, восстановленный Бобом, есть P_1. Докажем, что он эквивалентен P.

$$P_1 = C^d \bmod n = (P^e \bmod n) \bmod n = P^{ed} \bmod n$$

$$ed = k\varphi(n) + 1 \qquad \text{// d и e инверсны по модулю } \varphi(n)$$

$$P_1 = P^{ed} \bmod n \rightarrow P_1 = P^{k\varphi(n)+1} \bmod n$$

$$P_1 = P^{k\varphi(n)+1} \bmod n = P \bmod n \qquad \text{// Теорема Эйлера (вторая версия)}$$

Некоторые тривиальные примеры

Рассмотрим некоторые тривиальные (ненадежные) примеры процедуры RSA. Критерии, которые делают систему RSA безопасной, будут обсуждены в более поздних разделах.

Пример 10.5

Боб выбирает 7 и 11 как *p* и *q* и вычисляет $n = 7 \times 11 = 77$. Значение $\phi(n) = (7-1)(11-1)$ или 60. Теперь он выбирает два ключа, *e* и *d*, из z_{60}*. Если он выбирает $e = 13$, то $d = 37$. Обратите внимание, что $e \times d \bmod 60 = 1$ (они инверсны друг другу). Теперь предположим, что Алиса хочет передать исходный текст 5 Бобу. Она использует общедоступный ключ 13, чтобы зашифровать 5.

Исходный текст: 5 $C = 5^{13} = 26 \bmod 77$ Зашифрованный текст: 26

Боб получает зашифрованный текст 26 и использует секретный ключ 37, чтобы расшифровать зашифрованный текст.

Зашифрованный текст: 26 $P =$ от 26^{37} до $5 \bmod 77$ Исходный текст 5

Переданный Алисой текст получен Бобом как исходный текст 5 .

Пример 10.6

Теперь предположим, что другой человек, Джон, хочет передать сообщение Бобу. Джон может применить открытый ключ доступа, объявленный Бобом (вероятно, на его сайте), — 13; исходный текст Джона — 63. Джон делает следующие вычисления:

Исходный текст: 63 $C = 63^{13} = 28 \bmod 77$ Зашифрованный текст: 28

Боб получает зашифрованный текст 28 и использует свой секретный ключ 37, чтобы расшифровать зашифрованный текст.

Зашифрованный текст: 28 $P = 28^{37} = 63 \bmod 77$ Исходный текст: 63

Пример 10.7

Дженнифер создает пару ключей для себя. Она выбирает $p = 397$ и $q = 401$. Она вычисляет $n = 397 \times 401 = 159197$. Затем она вычисляет $\phi(n) = 396 \times 400 = 158400$. Затем она выбирает $e = 343$ и $d = 12007$. Покажите, как Тэд может передать сообщение «No» Дженнифер, если он знает e и n.

Решение

Предположим, что Тэд хочет передать сообщение «No» Дженнифер. Он изменяет каждый символ на число (от 00 до 25), сопоставляет каждой букве число, содержащее две цифры. Затем он связывает два кодированных символа и получает четырехзначное число. Исходный текст — 1314. Затем Тэд использует e и n, чтобы

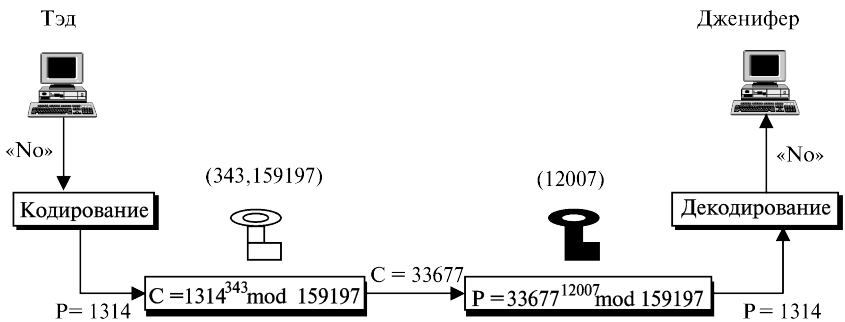


Рис. 10.7. Шифрование и дешифрование в примере 10.7

зашифровать сообщение. Зашифрованный текст $1314^{343} = 33677 \bmod 159197$. Дженнифер получает сообщение 33677 и использует d ключ дешифрования, чтобы расшифровать это сообщение: $33677^{12007} = 1314 \bmod 159197$. Затем Дженнифер расшифровывает 1314 как сообщение «No». Рисунок 10.7 показывает этот процесс.

Атаки RSA

До настоящего момента не было обнаружено никаких разрушительных атак RSA. Несколько атак были предсказаны. Они основаны на слабом исходном тексте, слабом выборе параметра или несоответствующей реализации. Рисунок 10.8 показывает категории потенциальных атак.

Рис. 10.8. Диаграмма возможных атак на RSA

Атака разложения на множители

Безопасность RSA базируется на следующей идее: модуль настолько большой, что разложение на множители в разумное время неосуществимо. Боб выбирает p и q и вычисляет $n = p \times q$. Число n общедоступно, p и q являются секретными. Если Ева сможет разложить на множители n и получить p и q, то она может вычислить $\phi(n) = (p - 1)(q - 1)$. Затем Ева тогда может вычислить $d = e^{-1} \bmod \phi(n)$, потому что e общедоступен. Секретный ключ d — лазейка, которую Ева может использовать, чтобы расшифровать зашифрованное сообщение.

Как мы узнали в лекции 9, есть много алгоритмов разложения на множители, но ни один из них не может найти сомножители большого целого числа с полиномиальной сложностью времени. Для того чтобы обеспечить безопасность, RSA требует, чтобы n был больше чем 300 десятичных цифр. Это означает, что модуль должен быть по крайней мере 1024 бита. Даже при использовании мощней-

шего и самого быстрого компьютера, доступного на сегодня, разложение на множители целого числа такого размера требует неосуществимо большого времени. Это означает, что RSA безопасен, пока не будет найден эффективный алгоритм разложения на множители.

Атака с выборкой зашифрованного текста

Потенциальная атака RSA базируется на мультипликативном свойстве RSA. Предположим, Алиса создает зашифрованный текст $C = P^e \bmod n$ и передает C Бобу. Также предположим, что Боб расшифрует произвольный зашифрованный текст для Евы – C_1, отличный от C. Ева перехватывает C и использует следующие шаги, чтобы найти P:

а. Ева выбирает случайное целое число X в $Z\ n*$.

б. Ева вычисляет $Y = C \times X^e \bmod n$.

в. Ева передает Y Бобу для дешифрования и получает $Z = Y^d \bmod n$; это шаг атаки выборкой зашифрованного текста.

г. Ева может легко найти P, потому что

$$Z = Y^d \bmod n = (C \times X^e)^d \bmod n = (C^d \times X^{ed}) \bmod n = (C^d \times X) \bmod n = (P \times X) \bmod n$$
$$Z = (P \times X) \bmod n \rightarrow P = Z \times X^{-1} \bmod n$$

Ева применяет расширенный евклидов алгоритм для того, чтобы найти мультипликативную инверсию X, и в конечном счете значение P.

Атаки на показатель степени шифрования

Чтобы уменьшить время шифрования, можно попытаться использовать короткий ключ шифрования — малое значение числа *e*, например, значение для *e*, такое как $e = 3$ (второе простое число). Однако есть некоторые потенциальные атаки на показатель при его малом значении степени шифрования, которые мы здесь кратко обсуждаем. Эти атаки вообще не кончаются вскрытием системы, но они все-таки должны быть предотвращены. Для того чтобы сорвать эти виды атак, рекомендуется использовать $e = 2^{16} + 1 = 65537$ (или простое число, близкое к этому значению).

Атака теоремы Куперсмита (Coppersmith) может быть главной для атаки малого показателя степени на ключ шифрования. Основное положение этой теоремы: для полинома $f(x)$ степени *e* по модулю *n,* чтобы найти корни, если один из корней является меньшим чем $n^{1/e}$, можно использовать алгоритм сложности, log *n*. Эта теорема может быть применена к RSA-криптосистеме $C = f(P) = P^e \bmod n$.

Если $e = 3$ и известны хотя бы две трети битов в исходном тексте P, алгоритм может найти все биты в исходном тексте.

Атака широковещательной передачи может быть начата, если один объект передает одно и то же сообщение группе получателей с тем же самым ключом шифрования. Например, предположим следующий сценарий: Алиса хочет передать одно и то же сообщение трем получателям с тем же самым общедоступным ключом $e = 3$ и модулями n_1, n_2 и n_3.

$C_1 = P^3 \bmod n_1 \qquad C_2 = P^3 \bmod n_2 \qquad C_3 = P^3 \bmod n_3$

Применяя китайскую теорему об остатках к этим трем уравнениям, Ева может найти уравнение формы $C' = P^3 \bmod n_1 n_2 n_3$. Это означает, что $P^3 < n_1 n_2 n_3$ и что $C' = P^3$ решается с помощью обычной арифметики (не модульной). Ева может найти значение $C' = P^{1/3}$.

Атака связанных между собой сообщений была обнаружена Франклином Рейтером (Franklin Reiter). Она может быть кратко описана следующим образом. Алиса зашифровала два исходных текста, P_1 и P_2, с помощью $e = 3$ и передает C_1 и C_2 Бобу. Если P_1 связан с P_2 линейной функцией, то Ева может восстановить P_1 и P_2 в выполнимое время вычисления.

Атака короткого списка, обнаруженная Куперсмитом, может быть кратко описана следующим образом. Алиса имеет сообщение M для передачи Бобу. Она записывает сообщение и зашифровывает его как сообщение r_1, а результат записывает как C_1 и передает C_1 (Бобу). Ева перехватывает C_1 и удаляет его. Боб сообщает Алисе, что он не получил сообщение, так что Алиса заполняет сообщение, снова зашифровывает как сообщение r_2 и передает это Бобу. Ева также перехватывает и это сообщение. Ева теперь имеет C_1 и C_2, и она знает, что оба зашифрованных текста принадлежат одному и тому же исходному тексту. Куперсмит доказал, что если r_1 и r_2 короткие, то Ева способна восстановить первоначальное сообщение M.

Атаки показателя степени дешифрации

Две формы атак могут быть проведены на показатель степени дешифрации: **атака раскрытой степени дешифрации** и **атака малого показателя степени дешифровации.** Они обсуждаются ниже.

Атака раскрытого показателя степени дешифрации. Очевидно, что если Ева может найти показатель степени дешифрации, *d*, она сможет расшифровать текущее зашифрованное сообщение. Однако на этом атака не останавливается. Если Ева знает значение *d*, она может использовать вероятностный алгоритм (не обсуждаемый здесь) к числу *n* и найти значения *p* и *q*. Следовательно, если Боб изменит только угрожающий безопасности показатель степени дешифрования, но сохранит тот же самый модуль *n*, Ева сможет расшифровать будущие сообщения, потому что она сможет разложить на множители *n*. Поэтому если Боб узнает, что показатель степени скомпрометирован, он должен выбрать новое значение для *p* и *q*, вычислить *n* и создать полностью новые секретный и открытый ключи доступа.

В RSA, если показатель степени *d* скомпрометирован, тогда *p*, *q*, *n*, *e* и *d* должны быть сгенерированы заново.

Атака малого значения показателя степени дешифрации. Боб может подумать, что использование малого значения степени секретного ключа *d* приводит к более быстрой работе алгоритма дешифрации. Винер показал, что в случае $d < 1/3n^{1/4}$

возможен специальный тип атаки, основанной на *цепной дроби*, — тема, которая рассматривается в теории чисел. Этот тип атаки может подвергнуть риску безопасность RSA. Для того чтобы это произошло, должно выполняться условие, что $q < p < 2q$; если эти два условия существуют, Ева может разложить n на сомножители в полиномиальное время.

В RSA рекомендовано, что d должно иметь величину $d \geq 1/3\, n^{1/4}$, чтобы предотвратить атаку малого значения ключа дешифрации.

Атаки исходного текста

Исходный текст и зашифрованный текст в RSA — это перестановки друг друга, потому что это целые числа в том же самом интервале (от 0 до $n - 1$). Другими словами, Ева уже знает кое-что об исходном тексте. Эти характеристики могут позволить некоторые атаки исходного текста. Три атаки были уже упомянуты в литературе: атака короткого сообщения, атака циклического повторения и явная атака.

Атака короткого сообщения. В **атаке короткого сообщения,** если Ева знает множество возможных исходных текстов, то ей известна еще одна информация и дополнительный факт, что зашифрованный текст — перестановка исходного текста. Ева может зашифровать все возможные сообщения, пока результат не будет совпадать с перехваченным зашифрованным текстом. Например, если известно, что Алиса посылает число с четырьмя цифрами Бобу, Ева может легко испытать числа исходного текста 0000 к 9999, чтобы найти исходный текст. По этой причине короткие сообщения должны быть дополнены случайными битами в начале и конце, чтобы сорвать этот тип атаки. Настоятельно рекомендуется заполнять исходный текст случайными битами прежде начала шифрования. Здесь используется метод, называемый OAEP, который будет позже обсужден в этой лекции.

Атака циклического повторения построена на факте, что если переставлять зашифрованный текст (перестановка исходного текста), то непрерывное шифрование зашифрованного текста в конечном счете кончится исходным текстом. Другими словами, если Ева непрерывно шифрует перехваченный зашифрованный текст C, она в итоге получит исходный текст. Однако сама Ева не знает, каков исходный текст, так что ей неизвестно, когда пора остановиться. Она должна пройти один шаг далее. Когда она получает зашифрованный текст C снова, она возвращается на один шаг, чтобы найти исходный текст.

Перехваченный зашифрованный текст C
$C_1 = C^e \bmod n$
$C_2 = C_1{}^e \bmod n$
.....
$C_k = C_{k-1}{}^e \bmod n$ ->, если $\mathbf{C_k = C}$, остановка: исходный текст – $\mathbf{P = C_{k-1}}$

Может ли это быть серьезной атакой на криптосистему RSA? Показано, что сложность алгоритма эквивалентна сложности разложения на множители n. Дру-

гими словами, нет никакого эффективного алгоритма, который может завершить эту атаку в полиномиальное время, если n является большим.

Явная атака сообщения. Другая атака, которая базируется на отношениях перестановки между исходным текстом и зашифрованным текстом, — **явная атака сообщения**. Явное сообщение — сообщение, которое зашифровано само в себя (не может быть скрыто). Было доказано, что есть всегда некоторые сообщения, которые шифруются сами в себя. Поскольку ключ шифрования обычно нечетен, имеются некоторые исходные тексты, которые зашифрованы сами в себя, такие как P = 0 и P = 1. Но если ключ шифровки выбран тщательно, число их незначительно. Программа шифровки может всегда проверить, является ли вычисленный зашифрованный текст таким же, как исходный текст, и отклонить исходный текст перед передачей зашифрованного текста.

Атаки модуля

Главной атакой RSA является атака разложения на множители. Ее можно рассматривать как атаку малого модуля. Однако поскольку мы уже обсудили эту атаку, сконцентрируемся на другом варианте: общей атаке модуля.

Общая атака модуля. Она может быть начата, если сообщество использует общий модуль, n. Например, люди в сообществе могли бы позволить третьей стороне, которой они доверяют, выбирать p и q, вычислять n и $\phi(n)$ и создать пару образцов (e_i, d_i) для каждого объекта. Теперь предположим, что Алиса должна передать сообщение Бобу. Зашифрованный текст Бобу — это $C = P^{e_B} \bmod n$ Боб использует свой секретный ключ, d_B, чтобы расшифровывать сообщение: $P = C^{e_B} \bmod n$. Проблема в том, что Ева может также расшифровать сообщение, если она — член сообщества и ей была назначена пара образцов (e_E и d_E), как мы узнали в разделе «атака малого значения ключа дешифрации». Используя свои собственные ключи (e_E и d_E), Ева может начать вероятностную атаку на сомножители n и найти d_B Боба. Чтобы сорвать этот тип атаки, модуль не должен быть в совместном пользовании. Каждый объект должен вычислить свой собственный модуль.

Атаки реализации

Предыдущие атаки базировались на основной структуре RSA. Как показал Дэн Бонех[1] (Dan Boneh), есть несколько атак реализации RSA. Мы приведем две из них: атака анализом времени и атака мощности.

Атака анализом времени (Timing attack). Пауль Кочер[2] (Paul Kocher) демонстрировал атаку только зашифрованного текста, называемую **атака анализом времени.** Атака основана на быстром алгоритме с показательным временем, который рассмотрен в лекции 9. Алгоритм использует только возведение во вторую степень, если соответствующий бит в секретном показателе степени d есть 0; он ис-

[1] Ученый, криптограф Стенфордского Университета.

[2] Один из самых известных и авторитетных специалистов в области криптографии, работает в Cryptography Research team.

пользуется и при возведении во вторую степень и умножении, если соответствующий бит — 1. Другими словами, синхронизация требует сделать каждую итерацию более длинной, если соответствующий бит — 1. Эта разность синхронизации позволяет Еве находить значение битов в d, один за другим.

Предположим, что Ева перехватила большое количество зашифрованных текстов от C_1 до C_m. Также предположим, что Ева наблюдала, какое количество времени требуется для Боба, чтобы расшифровать каждый зашифрованный текст, от T_1 до T_2 . Ева знает, сколько времени требуется для основных аппаратных средств, чтобы выполнить операцию умножения от t_1 до t_m. (здесь t_1 — время, требуемое для выполнения умножения). Результат операции умножения = Результат $\times C_i \bmod n$.

Ева может применить алгоритм 10.5, который является упрощенной версией алгоритма, используемого практически для вычисления всех бит в d (d_0 до d_{k-1}).

Алгоритм устанавливает начальное значение $d_0 = 1$ (потому что d должен быть нечетным) и вычисляет новые значения для T_i's (время дешифрования относится к d_1 до d_{k-1}). Алгоритм затем предполагает, что следующий бит — это 1, и находит несколько значений D_1 доD_2, основываясь на этом предположении.

Алгоритм 10.5. Атака синхронизации

```
RSA_Timing_Attack([T1……Tm])
{
   d0 ← 1                  // Потому что d – нечетное
   Вычислить [t1…. . tm]
   [T1……Tm] ← [T1……Tm] -[t1…. . tm]   // Обновление Ti для следующего бита
   for (j from1 to k-1)
   {
      Пересчитать [t1…. . tm]   // Пересчет ti в предположении,
                                // что следующий бит – это 1
      [D1…. Dm] ← [T1……Tm] - [t1…. . tm]
      var ← variance ([D1…. Dm]) - variance ([T1……Tm])
      if (var >0) dj ← 1 else dj ← 0
      [T1……Tm] ← [T1……Tm] - dj × [t1…. . tm]     // Обновление Ti
                                                   // для следующего бита
   }
}
```

Если принятое предположение верно, то каждый D_i является вероятно меньшим, чем соответствующее время передачи T_i. Однако алгоритм использует дисперсию (или другие критерии корреляции), чтобы рассмотреть все варианты Di и T_i. Если разность дисперсии положительная, алгоритм принимает предположение, что следующий бит равен 1 в противном случае предполагает, что следующий бит — 0. Алгоритм тогда вычисляет новые T_i , используя для этого оставшиеся биты.

Есть два метода сорвать атаку анализом времени:

1. добавить случайные задержки к возведению в степень, чтобы каждое возведение в степень занимало одно и то же время;
2. Ривест рекомендовал **«ослепление»**. По этой идее зашифрованный текст умножается на случайное число перед дешифрованием. Процедура содержит следующие шаги:
   a. Выбрать секретное случайное число r между 1 и ($n - 1$).
   b. Вычислить $C_1 = C \times r^e \bmod n$.
   d. Вычислить $P_1 = C_1^d \bmod n$.
   e. Вычислить $P = P_1 \times r^{-1} \bmod n$.

Атака анализом мощности подобна атаке анализом времени. Было показано, что если Ева может точно измерить мощность, использованную в течение дешифрования, она может начать атаку анализа мощности на основании принципов, рассмотренных для атаки анализом времени. Итеративное умножение и возведение в квадрат потребляют больше мощности, чем только итеративное возведение в квадрат. Та же самая группа методов, которая предотвращает атаки анализом времени, может сорвать атаки анализа мощности.

Рекомендации

Следующие рекомендации основаны на теоретических и экспериментальных результатах.

1. Число битов для n должно быть, по крайней мере, 1024. Это означает, что n должно быть приблизительно 2^{1024}, или 309 десятичных цифр.
2. Два простых числа p и q должны каждый быть по крайней мере 512 битов. Это означает, что p и q должны быть приблизительно 2^{512} или 154 десятичными цифрами.
3. Значения p и q не должен быть очень близки друг к другу.
4. $p - 1$ и $q - 1$ должны иметь по крайней мере один большой простой сомножитель.
5. Отношение p/q не должно быть близко к рациональному числу с маленьким числителем или знаменателем.
6. Модуль n не должен использоваться совместно.
7. Значение e должно быть $2^{16} + 1$ или целым числом, близким к этому значению.
8. Если произошла утечка частного ключа d, Боб должен немедленно изменить n так же, как e и d. Было доказано, что знание n и одной пары (e, d) может привести к открытию других пар того же самого модуля.
9. Сообщения должны быть дополнены, используя OAEP, который рассматривается далее.

Оптимальное асимметричное дополнение шифрования (OAEP — OPTIMAL ASSIMETRIC ENCRYPTION PADDING)

Как мы упоминали ранее, короткое сообщение в RSA делает зашифрованный текст уязвимым к *атакам короткого сообщения*. Там же показано, что простое

добавление фиктивных данных (дополнение) к сообщению затрудняет работу Евы, но, приложив дополнительные усилия, она может все еще атаковать зашифрованный текст. Решение, предложенное группой RSA и некоторыми другими разработчиками, состоит в том, чтобы применить процедуру, названную **оптимальным асимметричным дополнением шифрования (OAEP)**. Рисунок 10.9 показывает простую версию этой процедуры; реализация может использовать более сложную версию.

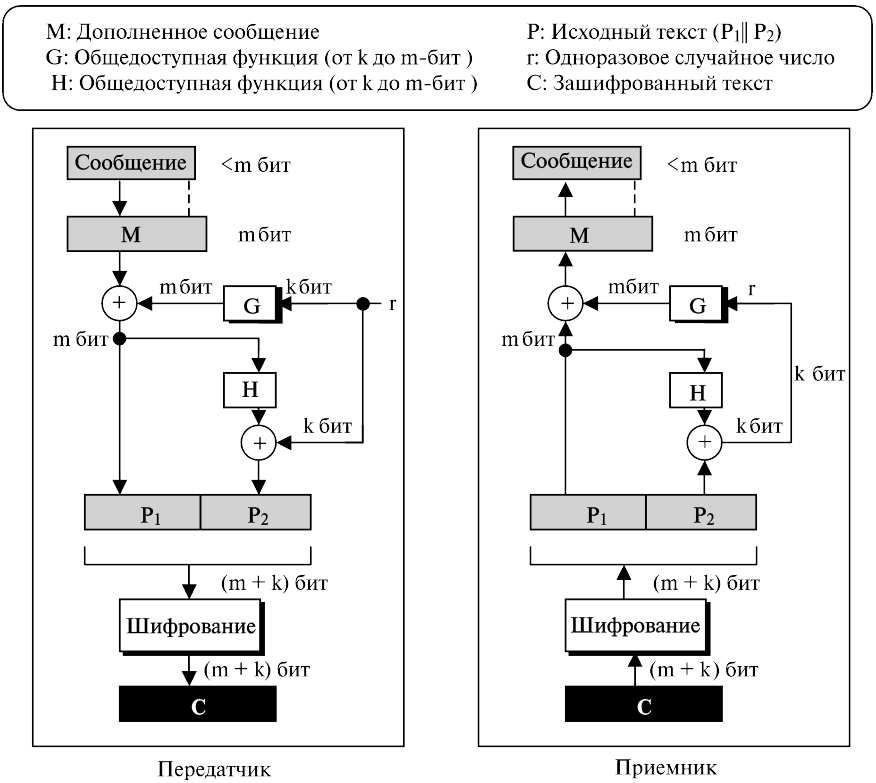


Рис. 10.9. Оптимальное асимметричное дополнение шифрования (OAEP)

Идея, показанная на рисунке 10.9, — это то, что $P = P_1 \| P_2$, где P_1 — замаскированная версия дополненного сообщения, M;

P_2 передается, чтобы позволить Бобу найти маску.

Шифрование. Ниже показаны шаги процесса шифрования.

1. Алиса дополняет сообщение, чтобы сделать его m-битовым. Мы обозначим его M.
2. Алиса выбирает случайное число *r* из *k* бит. Обратите внимание, что *r* применяется только однажды и затем уничтожается.

3. Алиса использует общедоступную одностороннюю функцию **G**, которая принимает целое r-битовое число, и создает m-разрядное целое число (*m* — размера M, и *r* <m). Это — маска.
4. Алиса применяет маску, G (*r*), чтобы создать первую часть исходного текста $P_1 = M \oplus G(r)$ является замаскированным сообщением.
5. Алиса создает вторую часть исходного текста $P_2 = H(P_1) \oplus r$. Функция H — другая общедоступная функция, которая принимает *m*-битовые входные сообщения и создает *k*-битовые выходные сообщения. Эта функция может быть *криптографической хэш-функцией* (см. лекцию 12). P_2 используется для того, чтобы дать возможность Бобу снова создать маску после дешифрации.
6. Алиса создает $C = P^e = (P_1 \parallel P_2)^e$ и передает C Бобу.

Дешифрование. Следующие шаги показывают процесс дешифрования:
1. Боб создает $P = C^d = (P_1 \parallel P_2)$.
2. Боб сначала обновляет значение *r*, используя $H(P_1) \oplus P_2 = H(P_1) \oplus H(P_2) \oplus r = r$.
3. Боб применяет $G(r) \oplus P = G(r) \oplus G(r) \oplus M. = M.$, чтобы обновить значение дополненного сообщения.
4. После удаления дополнения M, Боб находит первоначальное сообщение.

Ошибка в передаче

Если хотя бы один бит в течение передачи принят с ошибкой, текст, зашифрованный RSA, будет принят неправильно. Если полученный зашифрованный текст отличается от переданного, приемник не может определить первоначальный исходный текст. Исходный текст, вычисленный на стороне приемника, может очень отличаться от передаваемого передатчиком. Среда передачи должна быть освобожденной от ошибок за счет добавления избыточных бит или обнаружения и исправления ошибки в зашифрованном тексте.

Пример 10.8

Вот реальный пример. Мы выбираем 512-битовые *p* и *q*, вычисляем *n* и $\phi(n)$, затем выбираем *e* и испытываем, что оно взаимно простое с $\phi(n)$. Затем мы вычисляем *d*. Наконец, мы показываем результат шифрования и дешифрования. Целое число *p* — это число со 159 цифрами.

P = 96130345313583504574191581280615427909309845594996215822583150879647940455056470638491257160180347503120986666064924201918087806674210960633542199266612 09

Целое число *q* содержит 160 цифр.

q = 12060191957231446918276794204450896001555925054637033936061798321731482148483764659215389453209175225273226830107120695604602513887145524969000359660045617

Модуль $n = p \times q$. Это число имеет 309 цифр.

```
n =   1159350417396761496889250986461588752377145737545414477548552613
      7614788540S32635081727687881596832516846884930062548576411125016
      241455233918.29271625076567727274600970827141277304349605005563 4
      72745666280600999240371029914244722922157727985317270338393813 34
      692684137 32762200096667667183183108837342082344437095 3
```

$\phi(n) = (p - 1)(q - 1)$ имеет 309 цифр.

```
ϕ(n) =  1159350417396761496889250986461588752377145737545414477548555
        2613761478854083263508172768788159683251684688493006254857641111
        2501624145523391829271625076567510542336084929167520344826279 88
        1175547876570139234444057169895817281960982263610754672118646 12
        1713591073586406140088851702653772772644673410662438576641 28
```

Боб выбирает $e = 35535$ (идеально 65537), и испытание на простое число показывает, что это число и $\phi(n)$— взаимно простые числа. Затем Боб находит инверсию $e \bmod \phi(n)$— это обозначается d.

```
e =   35535
d =   58008302S60037763936093661289677917594669062089650962180422 86
      6111380593852S22358731706286910030021710859044338402170729869 08
      7600611530620252495988444804756824096624708148581713046324064 40
      7770483313401085094738529564507193677406119732655742423721761 76
      74620776371642 07600337085333288532144708859551366702948 31
```

Алиса хочет передать сообщение «THIS IS TEST», которое может быть представлено числовыми значениями, используя схему кодирования 00-26 (26 — *пробел*).

```
P =   19070818260818260026190418 19
```

Шифрованный текст, вычисленный Алисой, — это $\mathbf{C = P}^e$, числовое значение приведено ниже.

```
C =   4753091236462268272063655506105451809423717960704917165232392
      4305445296061319932856661784341835911415119741125200568297979 45
      7173603610127821884789274156609048002350719071527718591497518 84
      6588863210114835410336165789846796838676373376577465625079280 5
      2114814184404814184430812773059004692874248559166462108656
```

Боб может восстановить из зашифрованного текста исходный текст, используя $P = C^d$.

```
P =   19070818260818260026190418 19
```

После расшифровки восстановленный исходный текст — «THIS IS TEST».

Приложения

Хотя RSA может использоваться, чтобы зашифровать и расшифровывать реальные сообщения, это — очень длинные сообщения для RSA. Поэтому он является полезным для коротких сообщений. В частности мы увидим, что RSA применяется в цифровых подписях и других криптографических системах, которые нужны для шифрования маленьких сообщений без доступа к симметрическому ключу. Как мы увидим в последующих лекциях, RSA также используется для установления подлинности документа.

10.3. Криптосистема Рабина

Криптосистема Рабина (M. Rabin) является вариантом криптосистемы RSA. RSA базируется на возведении в степень сравнений. Криптосистема Рабина базируется на квадратичных сравнениях, и ее можно представить как криптографическую систему RSA, в которой значениям e и d присвоены значения $e = 2$ и $d = 1/2$. Другими словами, шифрование — $C \equiv p^2 \pmod{n}$ и дешифрование - $P \equiv C^{1/2} \pmod{n}$.

Открытый ключ доступа в криптосистеме Рабина — n, секретный ключ является кортежем (p, q). Каждый может зашифровать сообщение, используя n, но только Боб может расшифровать сообщение, используя p и q. Дешифрование сообщения неосуществимо для Евы, потому что она не знает значения p и q. Рисунок 10.10 показывает шифрование и дешифрование.

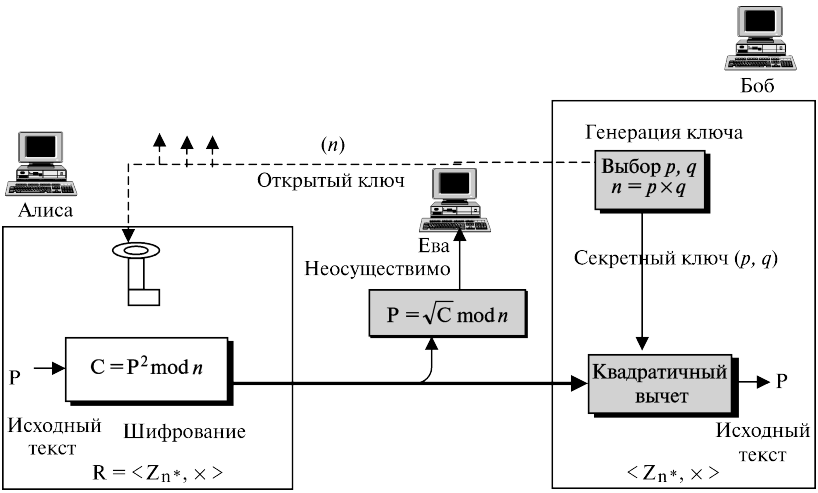


Рис. 10.10.Шифрование, дешифрование и генерация ключей в криптосистеме Рабина

Мы должны подчеркнуть, что если Боб применяет RSA, он может сохранить d и n и отказаться после генерации ключей от p, q и $\phi(n)$. Если Боб использует криптосистему Рабина, он должен сохранить p и q.

Процедура

Генерация ключей, шифрование и дешифрование показаны ниже.

Генерация ключей

Боб использует шаги, показанные в алгоритме 10.6, чтобы создать свой открытый ключ доступа и секретный ключ.

Алгоритм 10.6. Генерации ключей для криптосистемы Рабина

```
Rabin_Key_Generation
{
   Выберите два больших простых числа p и q в форме 4 k +3 и p ≠ q.
   n ← p × q
   Открытый_ключ ← n              // Может быть объявлен публично
   Секретный_ключ ← (q, n)        // Должен сохраняться в секрете
   return Открытый_ключ и Секретный_ключ
}
```

Хотя два простых числа, p и q, могут быть в форме $4k + 1$ или $4k + 3$, процесс дешифрования становится более трудным, если используется первая форма. Рекомендуют применять вторую форму, $4k + 3$, для того чтобы сделать дешифрование для Алисы намного проще.

Шифрование

Любой может передать сообщение Бобу, применив его открытый ключ доступа. Процесс шифрования показан алгоритмом 10.7.

Алгоритм 10.7. Шифрование в криптографической системе Рабина

```
Rabin_Encryption (n, P)    // n – открытый ключ доступа;
                           // P – зашифрованный текст Zn*
{
C← P 2mod n                // C – зашифрованный текст
return C
}
```

Хотя исходный текст P может быть выбран из множества Z_n, но чтобы сделать дешифрование более простым, мы определили множество, которое находится в Z_n^*.

Шифрование в криптосистеме Рабина очень простое. Операция нуждается только в одном умножении, что может быть сделано быстро. Это выгодно, когда ресурсы ограничены: например, при использовании карт с интегральной схемой, содержащей микропроцессор с ограниченной памятью, и при необходимости задействовать центральный процессор на короткое время.

Дешифрование

Боб может использовать алгоритм 10.8, чтобы расшифровать полученный зашифрованный текст.

Алгоритм 10.8. Дешифрование в криптосистеме Рабина

```
Rabin_Decryption (p, q, C)              // C – зашифрованный текст;
{                                       // p и q – секретные ключи
   a1 ← + (C^((p+1)/4)) mod p
   a2 ← - (C^((p+1)/4)) mod p
   b1 ← + (C^((q+1)/4)) mod q
   b2 ← - (C^((q+1)/4)) mod q
   // Алгоритм китайской теоремы об остатках вызывается четыре раза.
   P1 ← Китайский_остаток (a1, b1, p, q)
   P2 ← Китайский_остаток (a1, b2, p, q)
   P3 ← Китайский_остаток (a2, b1, p, q)
   P4 ← Китайский_остаток (a2, b2, p, q)
   return P1, P2, P3 и P4
}
```

Мы должны подчеркнуть здесь несколько моментов. Дешифрация базируется на решении квадратичного сравнения, которое рассмотрено в лекции 9. Поскольку полученный зашифрованный текст — квадрат исходного текста, это гарантирует, что C имеет корни (квадратичные вычеты) в Z_{n*}. Алгоритм китайской теоремы об остатках используется, чтобы найти четыре квадратных корня.

Самый важный пункт в криптосистеме Рабина — это то, что она недетерминирована. Дешифрование имеет четыре ответа. Задача получателя сообщения — точно выбрать один из четырех ответов как конечный ответ. Однако во многих ситуациях получатель может легко выбрать правильный ответ.

Криптосистема Рабина не детерминирована — дешифрование создает четыре одинаково вероятных исходных текста.

Пример 10.9

Вот очень тривиальный пример, чтобы проиллюстрировать идею.

1. Боб выбирает $p = 23$ и $q = 7$. Обратите внимание, что оба являются сравнениями 3 mod 4.
2. Боб вычисляет $n = p \times q = 161$.
3. Боб объявляет n открытым и сохраняет p и q в секрете.
4. Алиса хочет передать исходный текст P = 24. Обратите внимание, что 161 и 24 являются взаимно простыми; 24 находится в $Z_{161}*$. Она вычисляет C = от $24^2 = 93 \bmod 161$ и передает зашифрованный текст 93 Бобу.
5. Боб получает 93 и вычисляет четыре значения:
   a. $a_1 = + (93^{(23+1)/4}) \bmod 23 = 1 \bmod 23$
   b. $a_2 = - (93^{(23+1)/4}) \bmod 23 = 22 \bmod 23$
   c. $b_1 = + (93^{(7+1)/4}) \bmod 7 = 4 \bmod 7$

d. $b_2 = -(93^{(7+1)/4}) \bmod 7 = 3 \bmod 7$

6. Боб имеет четыре возможных ответа — (a_1, b_1), (a_1, b_2), (a_2, b_1), (a_2, b_2) и использует китайскую теорему об остатках, чтобы найти четыре возможных исходных текста: 116, 24, 137 и 45 (все из них взаимно простые к 161). Обратите внимание, что только второй ответ — исходный текст Алисы. Боб должен принять решение исходя из ситуации. Обратите внимание также, что все четыре ответа при возведении во вторую степень по модулю n дают зашифрованный текст 93, переданный Алисой.

$116^2 = 93 \bmod 161 \quad 24^2 = 93 \bmod 161 \quad 137^2 = 93 \bmod 161 \quad 45^2 = 93 \bmod 161$

Безопасность криптографической системы Рабина

Криптографическая система Рабина безопасна, пока p и q — большие числа. Сложность криптографической системы Рабина — такая же, как и у процедуры разложения на множители больших чисел n на два простых сомножителя p и q. Другими словами, криптографическая система Рабина так же безопасна, как и RSA.

10.4. Криптографическая система Эль-Гамаля

Помимо RSA и криптографической системы Рабина есть другая криптосистема с открытым ключом Эль-Гамаля (**ElGamal**), которая названа по имени ее изобретателя, Тахира Эль-Гамаля (Taher ElGamal). Криптосистема Эль-Гамаля базируется на свойствах дискретного логарифма, который обсуждался в лекции 9.

Криптографическая система Эль-Гамаля

На основании сведений лекции 9, если p — очень большое простое число, e_1 — первообразный корень в группе $G = <Z_{p^*}, \times >$ и r — целое число, тогда $e_2 = e_1^r \bmod p$ просто вычисляется с использованием быстрого показательного алгоритма (метод «возведения в квадрат и умножения»). Но по данным e_2, e_1 и p, невозможно вычислить $r = \log_{e1} e_2 \bmod p$ (проблема дискретного логарифма).

Процедура

Рисунок 10.11 показывает генерацию ключей, шифрование и дешифрование в криптосистеме Эль-Гамаля.

Генерация ключей

Боб использует шаги, показанные в алгоритме 10.9, чтобы создать свои общедоступный и частный ключи.

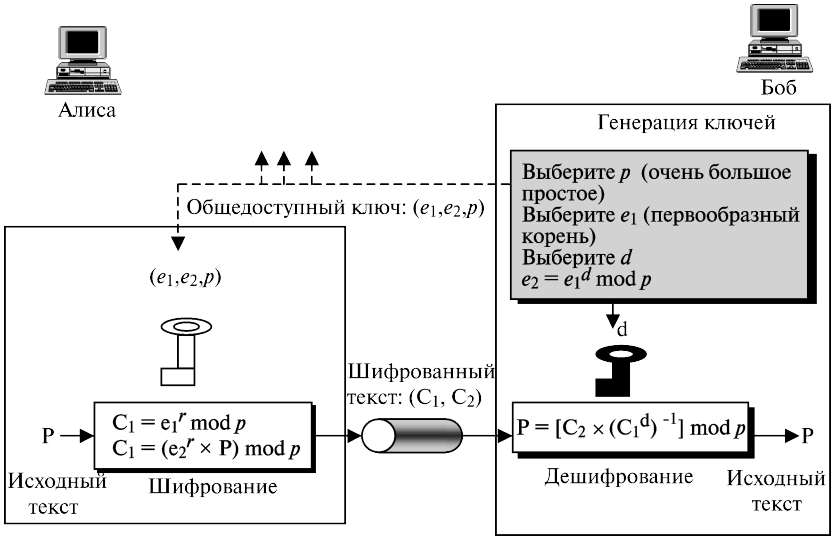


Рис. 10.11. Генерация ключей, шифрование, и дешифрование в криптосистеме Эль-Гамаля

Алгоритм 10.9. Генерация ключей в криптосистеме Эль-Гамаля

```
ElGamal_Key_Generation
{
    Выберите большое простое число p
    Выберите d, члена группы G = <Zp* × > , такое, что 1 <d <p - 2
    Выберите e1 – первообразный корнь в группе G = <Zp*, × >
    e2 ← e1^d mod p
    Общедоступный_ключ ←(e1, e2, p)      // Может быть объявлен публично
    Частный_ключ ←-d                     // Должен сохраняться в секрете
    return Общедоступный_ключ и Частный_ключ
}
```

Шифрование

Любой может передать сообщение Бобу, используя его открытый ключ доступа. Процесс шифрования показан в алгоритме 10.10. Если применяется быстрый показательный алгоритм (см. лекцию 9), шифрование в криптосистеме Эль-Гамаля может также быть выполнено по времени с полиномиальной сложностью.

Дешифрование

Боб может использовать алгоритм 10.11, чтобы расшифровать полученное сообщение зашифрованного текста.

Алгоритм 10.10. Шифрование в криптосистеме Эль-Гамаля

```
ElGamal_Encryption (e1, e2, p)        // P – исходный текст
{
     Выберите случайное целое число r в группе G = <Z.p*, × >
     C1 ← e1^r mod p
     C2 ← (P × e2^r) mod p            // C1 и C2 - зашифрованные тексты
                                      // return C1 и C2
}
```

Алгоритм 10.11. Дешифрование в криптосистеме Эль-Гамаля

```
ElGamal_Decryption {d, p, C1, C2)    // C1 и C 2 – зашифрованный текст
{
    P ← [C2 (C1^d)^-1] mod p         // P – исходный текст
    return P
}
```

Сложность разрядной операции шифрования или дешифрования в криптографической системе Эль-Гамаля — полиномиальная.

Доказательство

Криптосистема Эль-Гамаля проводит дешифрацию согласно выражению $C_2 \times (C_1^d)^{-1}$. Это выражение может быть проверено с помощью подстановки P:

$$[C_2 \times (C_1^d]^{-1} \bmod p = [(e_2^r \times P) \times (e_1^{rd}]^{-1} \bmod p = (e_1^{rd}) \times P \times (e_1^{rd})^{-1} = P$$

Пример 10.10

Рассмотрим тривиальный пример. Боб выбирает 11 в качестве p. Затем он выбирает $e_1 = 2$. Обратите внимание, что 2 — первообразный корень в Z_{11*} (см. приложение J). Затем Боб выбирает $d = 3$ и вычисляет $e_2 = e_1^d = 8$. Получены открытые ключи доступа — (2, 8, 11) и секретный ключ — 3. Алиса выбирает $r = 4$ и вычисляет C_1 и C_2 для исходного текста 7.

Исходный текст: 7
$C_1 = e_1^r \bmod 11 = 16 \bmod 11 = 5 \bmod 11$
$C2 = (P \times e_2^r) \bmod 11 = (7 \times 4096) \bmod 11 = 6 \bmod 11$l
Зашифрованный текст: (5, 6)

Боб получает зашифрованные тексты (5 и 6) и вычисляет исходный текст.

Зашифрованный текст: $[C_1 \times C_2{}^d)^{-1}] \bmod 11 = 6 \times (5^3)^{-1} \bmod 11 = 6 \times 3 \bmod 11 = 7 \bmod 11$
Исходный текст: 7

Пример 10.11

Вместо того чтобы использовать $P = [C_2 \times (C_1^d)^{-1}] \bmod p$ для дешифрования, мы можем избежать вычисления мультипликативной инверсии и применить $P = [C_2 \times C_1^{p-1-d}] \bmod p$ (см. малую теорему Ферма в лекции 9). В Примере 10.10 мы можем вычислить $P = [6 \times 5^{11-1-3}] \bmod 11 = 7 \bmod 11$.

Анализ

Очень интересная черта криптосистемы Эль-Гамаля — то, что Алиса создает *r* и сохраняет его в секрете; Боб создает *d* и сохраняет его в секрете. Это затруднение криптографической системы может быть решено следующим образом.

a. Алиса передает $C_2 = [e_2^r \times P] \bmod p = [(e_1^{rd}) \times P] \bmod p$. Выражение (e_1^{rd}) действует как маска, которая скрывает значение P. Чтобы найти значение P, Боб должен удалить эту маску.

b. Поскольку используется модульная арифметика, Боб должен создать точную копию маски и инвертировать ее (мультипликативная инверсия), чтобы снять воздействие маски.

c. Алиса передает Бобу $C_1 = e_1^r$, что является частью маски. Боб должен вычислить C_1^d, чтобы сделать точную копию маски, поскольку $C_1^d = (e_1^{r'})^d = (e_1^{rd})$. Другими словами, после получения точной копии маски Боб инвертирует ее и умножает результат на C_2, чтобы удалить маску.

d. Это можно представить так, что Боб помогает Алисе сделать маску (e_1^{rd}), не показывая значение d (*d* уже включено в $e_2 = e_1^{rd}$); Алиса помогает Бобу делать маску (e_1^{rd}), не раскрывая значение *r* (*r* уже включено в $C_1 = e_1^r$).

Безопасность криптосистемы Эль-Гамаля

Ранее были упомянуты две атаки на криптосистему Эль-Гамаля — атаки, основанные на малом значении модуля, и атаки знания исходного текста.

Атаки малого модуля

Если значение модуля *p* не является достаточно большим, Ева может использовать некоторые эффективные алгоритмы, чтобы решить проблему дискретного логарифма и найти *d* или *r*. Если *p* мало, Ева может просто найти $d = \log_{e1} e_2 \bmod p$ и сохранить его, чтобы расшифровать любое сообщение, передаваемое Бобу. Это может быть сделано единожды и работать, пока Боб применяет те же самые ключи. Ева может также использовать значение случайного числа *r*, применяемого Алисой в каждой передаче $r = \log_{e1} C_1 \bmod p$. Оба этих случая подчеркивают, что безопасность криптосистемы Эль-Гамаля зависит от решения проблемы дискретного логарифма с очень большим модулем. Поэтому рекомендовано, что *p* должны быть по крайней мере 1024 бита (300 десятичных цифр).

Атака знания исходного текста

Когда Алиса берет одно и то же значение случайного показателя степени r для того, чтобы зашифровать два исходных текста P и P', Ева обнаруживает P', если она знает P. Предположим, что $C_2 = P \times (e_2^r) \bmod p$ и $C'_2 = P' \times (e_2^r) \bmod p$. Ева находит P', используя следующие шаги:

1. $(e_2{}^r) = C_2 \times P^{-1} \bmod p$.
2. $P' = C'_2 \times (e_2{}^r)^{-1} \bmod p$.

Поэтому рекомендовано, чтобы Алиса брала при каждой передаче новое значение r, чтобы сорвать атаки.

Чтобы криптосистема Эль-Гамаля была безопасной, модуль *p* должен содержать по крайней мере 300 десятичных цифр, новых для каждой шифровки.

Пример 10.12

Вот более реальный пример. Боб использует случайное целое число длиной 512 битов (идеально — 1024) и целое число *p* длиной 155 цифр (идеал — 300 цифр). Боб выбирает e_1 и d, затем вычисляет e_2, как показано ниже; Боб объявляет (e_1, e_2, *p*) как свой открытый ключ и d как секретный ключ доступа.

p =	11534899272561676244925313717014331740490094532609834959814346921905689869862264593212975473787189514436889176526473093615929993728061165964347353440008577
e_1 =	2
d =	1007
e_2 =	9788641304300918950876685693809773904388006288733768761002206223325545070741561892123183177046101416733601508841329408572485377031582066010072558707455

Алиса имеет исходный текст P = 3200, чтобы передать Бобу. Она выбирает *r* = 545131, вычисляет C_1 и C_2 и передает их Бобу.

P =	3200
r =	545131
C_1 =	8872970693835284710225704714922756631202600672565621250181883514294172235997126811141053636617051730515815331891654009737363550802957367885690606191528881
C_2 =	7084543330489299445770160123807949995674360218361924469617745069212446961551658007794555930803458896144024085995259195792097216288796813505827795664302950

Боб вычисляет исходный текст $P = C_2 \times (C_1{}^d)^{-1} \bmod p = 3200 \bmod p$

P = 3200

Приложение

Криптосистема Эль-Гамаля может использоваться всякий раз, когда может использоваться RSA. Она применяется для замены ключей, установления подлинности, шифрования и дешифрования маленьких сообщений.

10.5. Криптосистемы на основе метода эллиптических кривых

Хотя RSA и Эль-Гамаль — безопасные асимметрично-ключевые криптографические системы, их безопасность обеспечивается ценой их больших ключей. Исследователи искали альтернативный метод, который дает такой же уровень безопасности с меньшими размерами ключей. Один из этих перспективных вариантов — **криптосистема на основе метода эллиптических кривых** (Elliptic Curve Cryptosystem — ECC). Система базируется на теории **эллиптических кривых**. Хотя глубокое рассмотрение этой теории находится вне задач и целей нашей книги, этот раздел сначала дает очень простое введение в три типа **эллиптических** кривых, а затем предлагает разновидности криптографических систем, которые используют некоторые из этих кривых.

Эллиптические кривые в вещественных числах

Эллиптические кривые, которые непосредственно не связаны с эллипсами, являются кубическими уравнениями двух переменных и обычно применяются для вычисления длины кривой в окружности эллипса. Общее уравнение для эллиптической кривой:

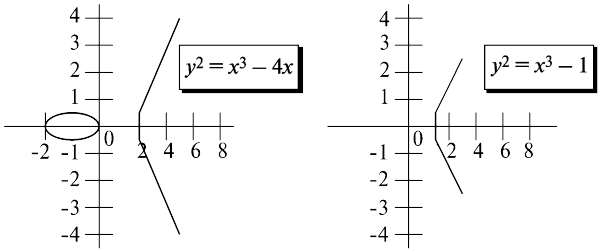
$$y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$$

Эллиптические кривые в поле вещественных чисел используют специальный класс формы эллиптических кривых:

$$y^2 = x^3 + ax + b$$

В этом случае, если $4a^3 + 27b^2 \neq 0$, уравнение представляет несингулярную **эллиптическую кривую**; в противоположном случае оно описывает **сингулярную эллиптическую кривую**. Для несингулярной эллиптической кривой уравнение $x^3 + ax + b = 0$ имеет три отличных корня (вещественных или комплексных); для сингулярной уравнение $x^3 + ax + b = 0$ не имеет трех отличных корней.

В уравнении, как мы можем видеть, левая сторона (y^2) имеет степень 2, в то время как правая сторона имеет степень 3 (x^3). Это означает, что горизонтальная линия может пересекать кривую в трех точках, если все корни вещественные. Однако вертикальная линия может пересечь кривую самое большее в двух точках.



a. Три вещественных корня

b. Один вещественный корень и два мнимых корня

Пример 10.13

Рисунок 10.12 показывает две эллиптические кривые с уравнениями $y^2 = x^3 - 4x$ и $y^2 = x^3 - 1$. Оба уравнения несингулярны. Однако первое имеет три вещественных корня ($x = -2$, $x = 0$, и $x = 2$), но второе — только один вещественный корень ($x = 1$) и два мнимых.

Рис. 10.12. Две эллиптические кривые в поле вещественных чисел

Абелева группа

Определим абелеву (коммутативную) группу (см. лекцию 4), использующую точки на эллиптической кривой. Кортеж P = (x_1, y_1) представляет точку на кривой, если x_1 и y_1 — координаты точки на кривой, которые удовлетворяют уравнению этой кривой. Например, точки P = (2,0; 0,0), Q = (0,0; 0,0), R = (–2,0; 0,0), S = (10,0;30,98), и T = (10,0; –30,98) – точки на кривой $y^2 = x^3 - 4x$. Обратите внимание, что каждая точка представлена двумя вещественными числами. По материалам лекции 4, для создания абелевой группы мы нуждаемся во множестве операций над множествами и пяти свойствах, которым удовлетворяют операции. В этом случае группа G = <**E**, +> — абелева.

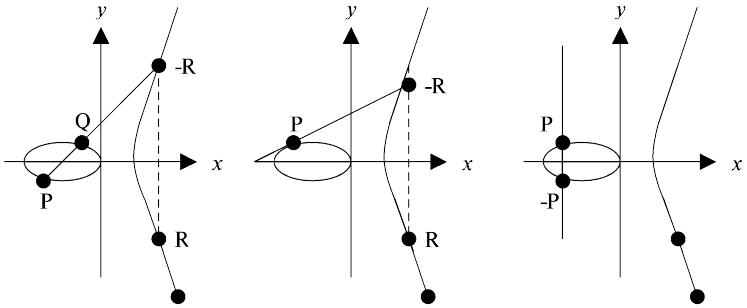
Множество. Мы определим множество как точки на кривой, где каждая точка — пара вещественных чисел. Например, множество E для эллиптической кривой $y^2 = -x^3 - 4x$ показано как

E = {(2,0; 0,0), (0,0; 0,0), (-2,0; 0,0), (10,0; 30,98), (10,0,-30,98)...}

Операция. Заданные свойства несингулярной эллиптической кривой позволяют нам определять операцию *сложение* точек на кривой. Однако мы должны помнить, что операция *сложения* здесь отличается от операции, которая была определена для целых чисел. Операция «сложение двух точек на кривой» проводится, чтобы получить другую точку на кривой.

R = P + Q, где P = (x_1, y_1), Q = (x_2, y_2), и R = (x_3, y_3)

Для того чтобы найти R на кривой, рассмотрим три случая, как это показано на рис. 10.13.



a. (R = P +Q) b. (R = P + P) c. (O = P + (-P))

Рис. 10.13. Три случая сложения на эллиптической кривой

1. В первом случае две точки P = (x_1, y_1) и Q = (x_2, y_2) имеют различные *x*-координаты и *y*-координаты ($x_1 \neq y_1$ и $x_2 \neq y_2$), как это показано на рис. 10.13a. Линия, соединяющая P и Q, пересекает кривую в точке, обозначенной R. R есть отражение (-R) относительно *y*-оси. Координаты точки R, x_3 и y_3 могут быть найдены по наклону линии, λ, и затем можно вычислить значений x_3 и y_3, как показано ниже:

$$\lambda = (y_2 - y_1)/(x_2 - x_1)$$
$$x_3 = \lambda^2 - x_2 - x_1 \qquad y_3 = \lambda (x_1 - x_3) - y_1$$

2. Во втором случае две точки совпадают (R = P + P) , как показано на рис. 10.13b. Наклон линии и координаты точки R могут быть найдены, как показано ниже.

$$\lambda = (3x_1^2 - a)/2y_1$$
$$x_3 = \lambda^2 - x_1 - x_2 \qquad y_3 = \lambda (x_1 - x_3) - y_1$$

3. В третьем случае две точки — аддитивные инверсии друг друга, как это показано на рис. 10.13c. Если первая точка равна P = (x_1, y_1), а вторая точка равна Q = (x_1, $-y_1$), линия, соединяющая эти две точки, не пересекает кривую в третьей точке. Математики говорят в этом случае, что точка пересечения находится в бесконечности. Они определяют точку ***O*** (см. рис. 10.13c) как *точку в бесконечности* или *нулевую точку*, которая *является аддитивным нейтральным элементом* группы.

Свойства операции. Краткие определения свойств операции, как они обсуждались в лекции 4:

1. *Замкнутость*. Может быть доказано, что сложение двух точек, с использованием операции сложения, определенное в предыдущем разделе, создает другую точку на кривой.

2. *Ассоциативность*. Может быть доказано, что (P + Q) + R = P + (Q + R).
3. *Коммутативность*. группа, состоящая из точек несингулярной эллиптической кривой, — абелева группа. Может быть доказано, что P + Q = Q + P.
4. *Существование нейтрального элемента*. Аддитивный нейтральный элемент в этом случае — нулевая точка. Другими словами, P + 0 = 0 + P.
5. *Существование инверсии*. Каждая точка на кривой имеет инверсию. Инверсия точки — это ее отражение относительно оси *x*. Другими словами, точки P = (x_1,y_1) и Q = (x_1, – y_1) — инверсии друг друга; это означает, что P + Q = 0. Заметим, что нейтральный элемент — это инверсия самого себя.

Группа и поле

Обратите внимание, что предыдущие рассуждения касаются двух алгебраических структур: группа и поле. Группа определяет множество точек на эллиптической кривой и операции сложения точек. Поле определяет сложение, вычитание, умножение и деление, применяющие операции над вещественными числами, которые необходимы, чтобы найти сложение точек в группе.

Эллиптические кривые в GF(p)

Наша предыдущая группа эллиптической кривой использовала вещественное поле для вычислений сложения точек. Криптография требует модульной арифметики. Мы определили группу эллиптической кривой с операцией сложения, но операция на координатах с точками в данном случае есть операция в **GF(*p*)** с $p > 3$. В модульной арифметике точки на кривой не представляют графы, как это можно было видеть на предыдущих рисунках, но сохраняются те же самые основные концепции. Мы используем ту же самую операцию сложения, но с вычислением по модулю *p*. В результате мы получаем эллиптическую кривую E_p (*a, b*), где *p* определяет модуль, и *b* — коэффициент уравнения $y^2 = x^3 + ax + b$. Обратите внимание, что хотя значение *x* в этом случае от 0 до *p*, обычно не все точки находятся на кривой.

Нахождение инверсии

Инверсия точки (*x, y*) равна (*x, – y*), где (*–y*) — аддитивная инверсия *y*. Например, если $p = 13$, инверсия (4, 2) равна (4, 11).

Нахождение точек на кривой

Алгоритм 10.12 показывает программу в псевдокоде для нахождения точек на кривой E_p (a, b).

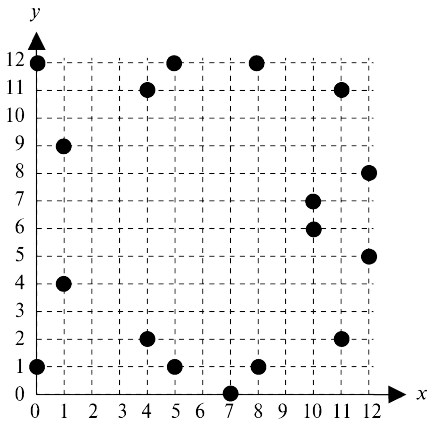
Алгоритм 10.12. Программа на псевдокоде для нахождения точек на эллиптической кривой

```
Elliptic_points (p, a, b)                          // p-модуль
{
   x ← 0
   while (x<p)
   {
      w ← ( x^3 + ax + b) mod p
```

```
        if (w – целое значение квадратного корня в Zp)      //w – это y²
        выход (x, √w)(x, – √w)
        x ← x + 1
    }
```

(0,1)	(0,12)
(1,4)	(1,9)
(4,2)	(4,11)
(5,1)	(5,12)
(7,0)	(7,0)
(8,1)	(8,12)
(10,6)	(10,7)
(11,2)	(11,11)

Точки



Граф

}

Пример 10.14

Определите эллиптическую кривую E_{13} (1, 1) по уравнению $y^2 = x^3 + x + 1$ и вычислите по модулю 13. Точки на кривой могут быть найдены, как показано на рис. 10.14.

Рис. 10.14. Точки на эллиптической кривой в поле GF (p)

Обратите внимание на следующее.

а. Некоторые значения y^2 не имеют квадратного корня по модулю 13. Они не являются точками на этой эллиптической кривой. Например, точки $x = 2$, $x = 3$, $x = 6$ и $x = 9$ не находятся на кривой.

б. Каждая точка, определенная на кривой, имеет инверсию. Инверсии перечислены как пары. Заметим, что (7, 0) — инверсия самой себя.

в. Обратите внимание, что для пары обратных точек значения y — аддитивные инверсии друг друга в Z_p. Например, 4 и 9 — аддитивные инверсии в Z_{13}. Так что мы можем сказать, что если 4 — это значение y, то значение $(-y)$ равно 9.

г. Инверсии находятся на тех же самых вертикальных линиях.

Сложение двух точек

Мы используем группу эллиптической кривой, определенную ранее, но вычисления сделаны в GF (p). Вместо вычитания и деления мы применяем аддитивные и мультипликативные инверсии.

Пример 10.15

Сложим две точки в примере 10.14, R = P + Q, где P = (4, 2) и Q = (10,6).

а. $X = (6 - 2) \times (10 - 4)^{-1} \bmod 13 = 4 \times 6^{-1} \bmod 13 = 5 \bmod 13$.

б. $x = (5^2 - 4 - 10) \bmod 13 = 11 \bmod 13$.

в. $y = [5 (4 - 11) - 2] \bmod 13 = 2 \bmod 13$.

г . R = (11, 2) является точкой на кривой в примере 10.14.

Умножение точки на константу

В арифметике умножение числа на константу *k* означает прибавление числа само к себе *k* раз. Здесь ситуация та же самая. Умножение точки P на эллиптической кривой *на* константу *k* означает прибавление точки P к себе *k* раз. Например, в E_{13} (1, 1), если точка (1, 4) умножается на 4, результат есть точка (5, 1). Если точка (8,1) умножается на 3, результат — точка (10, 7).

Эллиптические кривые в GF(2^n)

Вычисление в группе эллиптической кривой может быть определено в поле GF(2^n). В соответствии с лекцией 4, где мы говорили, что элементы множества в этом поле — *n*-битовые слова, которые можно интерпретировать как полиномы с коэффициентом в GF(2), сложение и умножение этих элементов такое же, как сложение и умножение полиномов. Для того чтобы определить эллиптическую кривую в GF(2^n), необходимо только изменить кубическое уравнение. Общее уравнение

$$y^2 + xy = x^3 + ax^2 + b$$

где $b \neq 0$. Обратите внимание, что значение *x*, *y*, *a* и *b* — полиномы, представляющие *n*-битовые слова.

Нахождение инверсии

Если P = (*x*, *y*), то (–P) = (*x*, *x* + *y*).

Нахождение точек на кривой

Мы можем написать алгоритм для нахождения точек на кривой, используя генераторы для полиномов, которые рассматривали в лекции 7. Но разработку этого алгоритма оставляем как упражнение. Далее следует очень тривиальный пример.

Пример 10.16

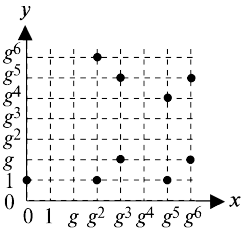
Мы выбираем GF (2^3) с элементами (0,1, *g*, g^2, g^3, g^4, g^5, g^6), использующими неприводимый полином $f(x) = x^3 + x + 1$. Этому соответствует полином $g^3 + g + 1 = 0$ или $g^3 = g + 1$. Другие степени *g* могут быть вычислены, как это показано ниже.

0	000	$g^3 = g + 1$	001
1	001	$g^4 = g^2 + g$	110
g	010	$g^5 = g^2 + g + 1$	111
g^2	100	$g^6 = g^2 + 1$	101

Используя эллиптическую кривую $y^2 + xy = x^3 + g^3x^2 + 1$, $a = g^3$ и $b = 1$, мы можем найти точки на этой кривой, как это показано на рисунке 10.15.

(0,1)	(0,1)
$(g^2,1)$	(g^2,g^6)
(g^3,g^2)	(g^3,g^5)
$(g^5,1)$	(g^5,g^4)
(g^6,g)	(g^6,g^5)

Точки



Граф

Рис. 10.15. Точки на эллиптической кривой в GF (2^n)

$$\lambda = (y_2 + y_1)/(x_2 + x_1)$$

$$x_1 = \lambda^2 + \lambda + x_1 + x_2 + a \qquad y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

$$\lambda = x_1 + y_1 / x_1$$

$$x_3 = \lambda^2 + \lambda + a \qquad y_3 = x_1^2 + (\lambda + 1)x_3$$

Сложение двух точек

Правила для сложения точек в GF(2^n) немного отличаются от правил GF(p).

1. Если P = (x_1, y_1), Q = (x_2, y_2), Q ≠ –P, Q ≠ P, то R = (x_3, y_3) = P + Q может быть найден как
2. Если Q = P, то R = P + P (или R = 2P) и может быть найден как

Пример 10.17

Пусть нам надо найти R = P + Q, где P = (0,1) и Q = (g^2,1). Мы имеем λ = 0 и R = (g^5, g^4).

Пример 10.18

Пусть нам надо найти R = 2P, где P = (g^2,1). Мы имеем λ = g^2 +1 /g^2 = g^2+ g^5= g +1 и R = (g^6, g^5).

Умножение точек на константу

Для того чтобы умножить точку на константу, точки должны складываться непрерывно согласно правилу R = 2P.

Криптография эллиптической кривой, моделирующая криптосистему Эль-Гамаля

Для шифрования и дешифрования текстов с помощью эллиптических кривых использовались несколько методов. Один из них состоит в том, чтобы моделировать криптосистему Эль-Гамаля, используя эллиптическую кривую в GF(p) или GF(2^n), как это показано на рис. 10.16.

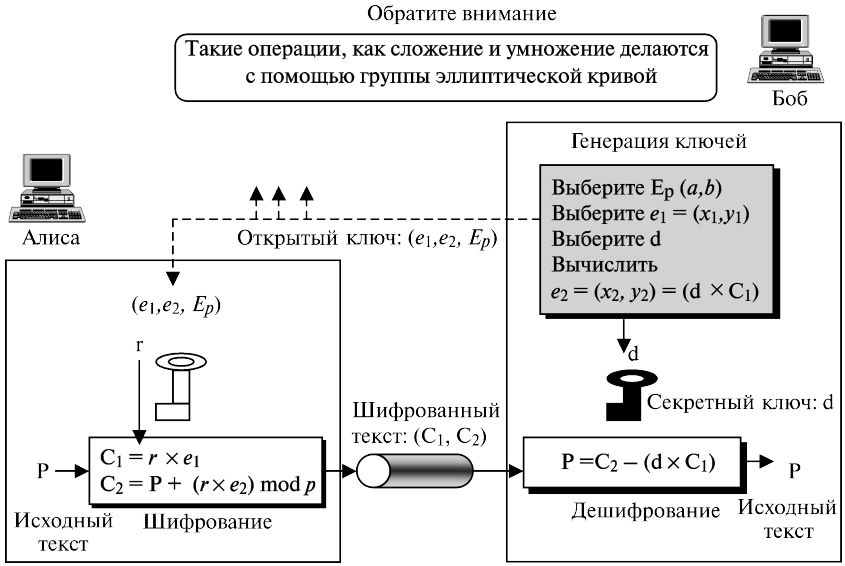


Рис. 10.16. Криптосистема Эль-Гамаля, использующая эллиптическую функцию

Генерация общедоступных и частных ключей

1. Боб выбирает E (a,b) с эллиптической кривой в GF(p) или GF(2^n).
2. Боб выбирает точку на кривой, e_1 (x_1, y_1).
3. Боб выбирает целое число d.
4. Боб вычисляет e_2 (x_2, y_2) = $d \times e_1$ (x_1, y_1). Обратите внимание: умножение здесь означает многократное сложение точек, которое было определено выше.
5. Боб объявляет E (a,b), e_1 (x_1, y_1) и e_2 (x_2, y_2) как свой открытый ключ доступа; он сохраняет d как секретный ключ.

Шифрование

Алиса выбирает P, точку на кривой, как ее исходный текст, P. Затем она вычисляет пару точек, направляет как зашифрованный текст:

$$\mathbf{C_1 = r \times e_1} \qquad \mathbf{C_2 = P + r \times e_2}$$

Читатель может задаться вопросом, как произвольным исходным текстом может быть точка на эллиптической кривой. Это одна из основных проблем в применении эллиптической кривой для моделирования. Алиса должна использовать алгоритм, чтобы найти непосредственное соответствие между символами (или блоками текста) и точками на кривой.

Дешифрование

Боб, после получения C_1 и C_2, вычисляет P, исходный текст, используя следующую формулу:

$P = C_2 - (d \times C_1)$ Знак «минус» здесь означает сложение с инверсией.

Мы можем доказать, что P, вычисленный Бобом, — тот же, что передан Алисой, как это показано ниже:

$$P + r \times e_2 - (d \times r \times e_1) = P + (r \times d \times e_1) - (r \times d \times e_1) = P + 0 = P$$

P, C_1, C_2 и e_2 — это точки на кривой. Обратите внимание, что результат сложения двух обратных точек на кривой — *нулевая точка*.

Пример 10.19

Вот очень тривиальный пример шифровки с использованием эллиптической кривой в GF (p).

1. Боб выбирает E_{67} (2, 3) как эллиптическую кривую в GF (p).
2. Боб выбирает $e_1 = (2, 22)$ и $d = 4$.
3. Боб вычисляет $e_2 = (13, 45)$, где $e_2 = d \times e_1$.
4. Боб публично объявляет кортеж (E, e_1, e_2).
5. Алиса хочет передать исходный текст P = (24, 26) Бобу. Она выбирает $r = 2$.
6. Алиса находит точку $C_1 = (35, 1)$, где $C_1 = r \times e_1$.
7. Алиса находит точку $C_2 = (21, 44)$, где $C_2 = P + r \times e_1$.
8. Боб получает C, и C_2. Он использует $2 \times C_1$, (35, 1) и получает (23, 25).
9. Боб инвертирует точку (23, 25) и получает точку (23, 42).
10. Боб складывает (23, 42) с $C_2 = (21, 44)$ и получает первоначальный исходный текст P = (24, 26).

Сравнение

Ниже приводится краткое сравнение алгоритма Эль-Гамаля с его вариантом, использующим эллиптическую кривую.

a. Алгоритм Эль-Гамаля использует мультипликативную группу; вариант — эллиптическую группу.
b. Эти два члена в алгоритме Эль-Гамаля — числа в мультипликативной группе; при применении варианта — точки на эллиптической кривой.
c. Секретный ключ в каждом алгоритме — целое число.
d. Секретные числа, выбираемые Алисой в каждом алгоритме, — целые числа.
e. Возведение в степень в алгоритме Эль-Гамаля заменено умножением точки на константу.
f. Умножение в алгоритме Эль-Гамаля заменено сложением точек.
g. Инверсия в алгоритме Эль-Гамаля — мультипликативная инверсия в мультипликативной группе; инверсия —заменяется аддитивной инверсией точки на кривой.
h. Вычисление обычно легче в эллиптической кривой, потому что умножение проще, чем возведение в степень, сложение проще, чем умножение, и

нахождение инверсии намного проще в группе эллиптической кривой, чем в мультипликативной группе.

Безопасность метода с использованием эллептической кривой

Чтобы расшифровать сообщение, Ева должна найти значение *r* или *d*.

a. Если Ева знает значение r, она может использовать $P = C_2 - (r \times e_2)$, чтобы найти точку P, относящуюся к исходному тексту. Но для того чтобы найти *r*, Ева должна решить уравнение $C_1 = r \times e_1$. Это значит — найти две точки на кривой, C_1 и e_1. Ева должна найти множитель, который создает C_1 начиная с e_1. Эта проблема известна как проблема логарифма **эллиптической кривой**, единственный известный метод решения этой проблемы — PO – алгоритм Поларда, который неосуществим, если задано большое *r* и *p* в *GF* (*p*) или большое *n* в $GF(2^n)$.

b. Если Ева знает значение *d,* она может использовать $P = C_2 - (d \times C_1)$, чтобы найти точку P, относящуюся к исходному тексту. Поскольку $e_2 = d \times e_1$, это тот же самый тип проблемы, что и в предыдущем пункте. Ева знает значение e_1 и e_2 — она должна найти *d*.

Безопасность криптосистемы с эллиптической кривой зависит от трудности решения проблемы логарифма эллиптической кривой.

Размер модуля

Для того же самого уровня безопасности (затраты на вычисление) модуль *n,* может быть меньшим в эллиптической системе (ECC), чем в RSA. Например, ECC в $GF(2^n)$ с *n,* состоящий из 160 битов, может обеспечить тот же уровень безопасности, как RSA с *n* 1024 бита.

10.6. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Криптографическая система RSA рассматривается в [Sti06], [Sta06], [PHS03], [Vau06], [TW06] и [Mao04]. Криптосистемы Рабина и Эль-Гамаля — в [Sti06] и [Mao 04]. Криптография эллиптической кривой — в [Sti06], [Eng99] и [Bla99].

Сайты

Следующие сайты дают больше информации о темах, рассмотренных в этой лекции.

http: // wwwl.ics.uci.edu / ~ mingl/knapsack.html

www.dtc.umn.edu/~odlyzko/doc/arch/knapsack.survey.pdf

http://en.wikipedia.org/wiki/RSA
citeseer.ist.psu.edu/boneh99twenty.html
www.mat.uniroma3.it/users/pappa/SLIDES/RSA-HRL_05.pdf
http://en.wikipedia.org/wiki/Rabin_cryptosystem
http://en.wikipedia.org/wiki/ElGamaL_encryption
ww..cs.purdue.edu/homes/wspeirs/elgamal.pdf
http://en.wikipedia.org/wiki/Elliptic__curve_cryptography
www.cs.utsa.edu/~rakbani/publications/Akbani-ECC-IEEESMC03.pdf

10.7. Итоги

- Есть два способа достигнуть информационной безопасности: криптография с симметричными ключами и криптография с асимметричными ключами. Эти два способа существуют параллельно и дополняют друг друга; преимущества одного могут дать компенсацию недостаткам другого.
- Концептуальные различия между этими двумя способами базируются на том, как они сохраняют секретность. В криптографии с симметричными ключами секретность должна быть разделена между двумя объектами; в криптографии с асимметричными ключами секретность персональная (неразделенная).
- Криптография с симметричными ключами базируется на подстановке и перестановке символов; криптография с асимметричными ключами базируется на применении математических функций к числам.
- Криптография с асимметричными ключами использует два отдельных ключа: один секретный и один открытый. Шифрование и дешифрование можно представлять себе как запирание и отпирание замков ключами. Замок, который заперт открытым ключом, можно отпереть только соответствующим секретным ключом.
- В криптографии с асимметричным ключом ответственность обеспечения безопасности находится, главным образом, на плечах приемника (Боб), который должен создать два ключа: один секретный и один открытый. Боб несет ответственность за секретный ключ. Открытый ключ может быть распространен сообществу через канал распределения открытого ключа.
- В отличие от криптографии с симметричными ключами, в криптографии с асимметричным ключом исходный текст и зашифрованный текст обрабатываются как целые числа. Сообщение должно кодироваться как целое число (или множество целых чисел) перед шифрованием; целое число (или множество целых чисел) должно быть расшифровано в сообщение после дешифрования. Криптография с асимметричным ключом обычно используется, чтобы зашифровать или расшифровывать маленькие сообщения, такие как ключ шифра для криптографии с симметричными ключами.
- Главная идея криптографии с асимметричным ключом — понятие «лазейка» в односторонней функции (TOWF), которая является такой функцией, что f вычисляется просто, а f^{-1} вычислить невозможно (в смысле сложности вычислений), если не используется лазейка.

- Блестящая идея относительно криптографии общедоступного ключа принадлежит Меркелю и и Хеллману – это ранцевая криптосистема . Когда нам говорят, какие элементы из заранее заданного множества чисел находятся в рюкзаке, мы можем легко вычислить сумму чисел; когда нам сообщают сумму, трудно сказать, какие элементы находятся в рюкзаке, если он не заполнен элементами сверхвозрастающего множества.
- Самый общий алгоритм общедоступного ключа — криптографическая система RSA. RSA использует два числа e и d, где e — общедоступный ключ, а d является частным (секретным). Алиса использует $C = P^e \bmod n$ для того, чтобы создать зашифрованный текст C из исходного текста P; Боб использует $P = C^d \bmod n$, чтобы извлечь исходный текст, переданный Алисой.
- RSA применяет две алгебраических структуры: кольцо и группа. Шифрование и дешифрование выполняются с использованием коммутативного кольца $R = <Z_n^*, +, \times >$ с двумя арифметическими операциями — сложением и умножением. RSA применяет мультипликативную группу $G = <Z_n^*, \times >$ для генерации ключей.
- Никаких разрушительных атак на RSA не было обнаружено. Теоретически предсказано несколько атак, основанных на разложении на множители, выборке шифрованного текста, образце дешифрования, образце шифрования, исходном тексте, модуле и реализации.
- Криптосистема Рабина — вариант криптографической системы RSA. RSA базируется на экспоненциальном сравнении; криптосистема Рабина базируется на квадратичном сравнении. Мы можем представлять себе, что криптосистема Рабина — это RSA, в которой значение $e = 2$ и $d = 1/2$. Криптографическая система Рабина безопасна, пока p и q — большие числа. Сложность криптосистемы Рабина — на том же самом уровне, как и процесс разложения большого числа n на два простых сомножителя p и q.
- Криптосистема Эль-Гамаля базируется на проблеме дискретного логарифма. Криптосистема Эль-Гамаля использует идею первообразных корней в Z_n^*. Шифрование и дешифрование в криптосистеме Эль-Гамаля использует группу $G = <Z_p^*, \times >$. Общедоступный ключ — это два числа, e_1 и e_2, a секретный ключ — это целое число d. Безопасность криптосистемы Эль-Гамаля основана на том, что решение проблемы дискретного логарифма не существует. Однако в литературе была упомянута атака, основанная на малом значении модуля, и атака знания исходного текста.
- Другая криптографическая система, рассмотренная в этой лекции, базируется на эллиптических кривых. Эллиптические кривые являются кубическими уравнениями в двух переменных. Эллиптические кривые на поле вещественных чисел используют специальный класс эллиптических кривых $y^2 = x^3 + ax + b$, где $4a^3 + 27b^2 \neq 0$. Абелева группа была определена с помощью эллиптической кривой с операцией сложения, которая показывает, как две точки на кривой можно сложить, чтобы получить другую точку на этой кривой.

• Криптография эллиптической кривой применяет две алгебраических структуры, абелеву группу и поле. Поле может быть полем вещественных чисел, *GF* (*p*) и GF (2^n). Мы показали, как криптосистема Эль-Гамаля может моделироваться, используя эллиптические кривые в конечном поле. Безопасность криптографии эллиптической кривой зависит от *проблемы логарифма эллиптической кривой,* решение которой неосуществимо при большом значении модуля.

10.8. Набор для практики

Обзорные вопросы

1. Найдите различия между криптосистемами с симметричными ключами и асимметричными ключами.
2. Найдите различия между открытыми и секретными ключами в криптосистеме с асимметричными ключами. Найдите совпадения и различие ключей в криптосистемах с симметричными ключами и с асимметричными ключами.
3. Определите «лазейку» в односторонней функции и объясните её использование в криптографии с асимметричным ключом.
4. Кратко объясните идею ранцевой криптосистемы.
   а. Что является односторонней функцией в этой системе?
   б. Что является лазейкой в этой системе?
   в. Определите открытые и секретные ключи в этой системе.
   d. Опишите безопасность этой системы.
5. Кратко объясните идею криптографической системы RSA.
   а. Что является односторонней функцией в этой системе?
   б. Что является лазейкой в этой системе?
   в. Определите открытые и секретные ключи в этой системе.
   г. Опишите безопасность этой системы.
6. Кратко объясните идею криптосистемы Рабина.
   а. Что является односторонней функцией в этой системе?
   б. Что является лазейкой в этой системе?
   в. Определите открытые и секретные ключи в этой системе.
   г. Опишите безопасность этой системы.
7. Кратко объясните идею криптосистемы Эль-Гамаля.
   а. Что является односторонней функцией в этой системе?
   б. Что является лазейкой в этой системе?
   г. Определите открытые и секретные ключи в этой системе.
   д. Опишите безопасность этой системы.
8. Кратко объясните идею криптографии эллиптической кривой (ECC).
   а. Что является односторонней функцией в этой системе?
   б. Что является лазейкой в этой системе?
   в. Определите открытые и секретные ключи в этой системе.
   г. Опишите безопасность этой системы.

9. Определите эллиптические кривые и объясните их приложения в криптографии.
10. Определите операцию, используемую в абелевой группе, которая обрабатывает точки на эллиптической кривой.

Упражнения

1. Учитывая сверхвозрастающий кортеж b = [7, 11,23,43,87, 173, 357], r =41 и модуль n = 1001, зашифруйте и расшифруйте букву a, используя ранцевую криптосистему. Используйте [7 6 5 1 2 3 4] как таблицу перестановки.
2. В RSA:
   a. Дано n = 221 и e = 5, найдите d.
   b. Дано n =3937 и e =17, найдите d.
   c. Дано p = 19, q = 23 и e = 3, найдите n, $\phi(\mathbf{n})$ и d.
3. Для того чтобы понять безопасность алгоритма RSA, найдите d, если вы знаете, что e =17, а n =187.
4. В RSA дано n и ϕ(n), вычислите p и q.
5. В RSA дано e = 13 и n = 100.
   Зашифруйте сообщение «HOW ARE YOU», применяя 00 к 25 для букв от A до Z и 26 — для пробела. Используйте различные блоки, чтобы сделать P <n.
6. В RSA дано n = 12091 и e = 13. Зашифруйте сообщение «THIS IS THOGH», используя схему кодирования 00 к 26. Расшифровать зашифрованный текст, чтобы найти первоначальное сообщение.
7. В RSA:
   а. Почему Боб не может выбрать 1 как открытый ключ e?
   б. Какова проблема в выборе 2 открытым ключом?
8. Алиса использует открытый ключ RSA Боба (e = 17, n = 19519), чтобы передать сообщение из четырех символов Бобу, применяющему схему A ↔ 0, B ↔ 1... Z ↔ 25 кодирования и декодирования по каждому символу отдельно. Ева перехватывает зашифрованный текст (6625 0 2968 17863) и расшифровывает сообщение, не разлагая на множители модуль. Найдите исходный текст; объясните, почему Ева смогла легко взломать зашифрованный текст.
9. Алиса использует открытый ключ RSA Боба (e = 7, n = 143), чтобы передать исходный текст P = 8, зашифрованный в виде текста C = 57. Покажите, как Ева может использовать атаку выборки текста, если она имеет доступ к компьютеру Боба, чтобы найти исходный текст.
10. Алиса использует общедоступный ключ RSA Боба (e = 3, n = 35) и передает зашифрованный текст 22 Бобу. Покажите, как Ева может найти исходный текст, используя *атаку циклического повторения*.
11. Предложите, как Алиса может предотвратить *атаку связанного сообщения* на RSA.
12. Используя криптосистему Рабина с p = 47 и q =11:
    a. Зашифруйте P = 17 и найдите зашифрованный текст.

б. Используя китайскую теорему об остатках, найдите четыре возможных исходных текста.

13. В криптосистеме Эль-Гамаля дано простое число $p = 31$:
    а. Выберите соответствующие e_1 и d, затем вычислите e_2.
    б. Зашифруйте сообщение «HELLO»; используйте 00 к 25 для кодирования. Используйте различные блоки для того, чтобы сделать $P<p$.
    в. Расшифруйте зашифрованный текст, чтобы получить исходный текст.
14. Что случится в криптосистеме Эль-Гамаля, если C_1 и C_2 будут изменены в течение передачи?
15. Предположим, что Алиса применяет в криптосистеме Эль-Гамаля общедоступный ключ Боба ($e_1 = 2$ и $e_2 = 8$), чтобы передать два сообщения — $P = 17$ и $P' = 37$. Они оба используют то же самое случайное целое число $r = 9$. Ева перехватывает зашифрованный текст и так или иначе находит значение $P = 17$. Покажите, как Ева может применить атаку знания исходного текста, чтобы найти значение P'.
16. В эллиптической кривой E (1, 2) в поле **GF** (11):
    а. Найдите уравнение кривой.
    б. Найдите все точки на кривой и сделайте рисунок, такой же, как рис. 10.14.
    в. Сгенерируйте общедоступный и секретный ключи для Боба.
    г. Выберите точку на кривой как исходный текст Алисы.
    д. Создайте зашифрованный текст, соответствующий исходному тексту Алисы в пункте d.
    е. Расшифруйте зашифрованный текст для Боба, чтобы найти исходный текст, передаваемый Алисой.
17. В эллиптической кривой E (g^4, 1) в поле GF(2^4):
    а. Найдите уравнение кривой.
    б. Найдите все точки на кривой и сделайте рисунок, такой же, как рис. 10.14.
    в. Сгенерируйте общедоступный и секретный ключи для Боба.
    г. Выберите точку на кривой как исходный текст Алисы.
    д. Создайте зашифрованный текст, соответствующий исходному тексту Алисы в пункте г.
    е. Расшифруйте зашифрованный текст для Боба, чтобы найти исходный текст, передаемый Алисе.
18. Используйте ранцевую криптосистему:
    а. Напишите алгоритм для шифрования.
    б. Напишите алгоритм для дешифрования.
19. В RSA:
    а. Напишите алгоритм для шифрования, используя оптимальное асимметричное дополнение шифрования (OAEC).
    б. Напишите алгоритм для дешифрования, используя оптимальное асимметричное дополнение шифрования (OAEC).
20. Напишите алгоритм для атаки циклического повторения на RSA.
21. Напишите алгоритм для сложения двух точек на эллиптической кривой в **GF**(p).
22. Напишите алгоритм для сложения двух точек на эллиптической кривой в **GF**(2^n).

Часть 3. Целостность, установление подлинности и управление ключами

В лекции1 мы видели, что криптография обеспечивает три метода защиты информации: шифры с симметричным ключом, шифры с асимметричным ключом и хэширование. Часть 3 обсуждает криптографические хэш-функции и их приложения. Эта часть также исследует другие проблемы, связанные с темами, о которых мы говорили в частях 1 и 2 — такими как управление ключами. Лекция 11 обсуждает общую идею целостности сообщения и установления подлинности сообщения. Лекция 12 исследует несколько криптографических хэш-функций. Лекция 13 обсуждает цифровые подписи. Лекция 14 показывает идеи и методы установления подлинности объекта. Наконец, лекция 15 обсуждает управление ключами, используемое для криптографии с симметричными ключами и криптографии с асимметричными ключами.

Лекция 11: Целостность сообщения и установление подлинности сообщения

Лекция 11 обсуждает общие идеи, связанные с криптографическими хэш-функциями, которые используются, чтобы создать на основании сообщения дайджест сообщения. Дайджесты сообщения гарантируют целостность сообщения. Затем лекция затем показывает, как простые дайджесты сообщения могут быть усовершенствованы, чтобы подтвердить подлинность сообщения.

Леция 12: Криптографические хэш-функции

Лекция12 исследует несколько стандартных криптографических хэш-функций, принадлежащих двум обширным категориям: функции со сжатием и с блочным шифром, используемым как функция сжатия. Лекция затем описывает одну хэш-функцию для каждой категории и алгоритмы хэширования SHA-512 и Whirlpool.

Лекция 13: Цифровые подписи

Леция 13 обсуждает цифровые подписи и вводит несколько схем цифровой подписи, включая RSA, Эль-Гамаля, Шнорра, DSS и эллиптическую кривую. Лекция также исследует некоторые атаки вышеупомянутых схем и то, как они могут быть предотвращены.

Лекция 14: Установление подлинности объекта

Лекция14 сначала показывает отличие между установлением подлинности сообщения и установлением подлинности объекта. Затем мы поговорим о некоторых методах установления подлинности объекта: использовании пароля, методе «вызов-ответ» и протоколах бесспорного подтверждения. Лекция также включает некоторые сведения по биометрии.

Лекция 15: Управление ключами

Лекция 15 сначала объясняет различные подходы к управлению ключами: использование централизованного распределения ключей (Key Distribution

Center — KDC), Центры сертификации (Certification Authories — CA) и инфраструктура общедоступного ключа (Public-Key Infrastructure — PKI). Мы покажем, как криптография с симметричными ключами и криптография с асимметричными ключами могут дополнить друг друга, чтобы решить некоторые проблемы управления ключами.

Лекция 11. Целостность сообщения и установление подлинности сообщения

Цели и содержание

Эта лекция имеет несколько целей.

- Определить целостность сообщения.
- Определить установление подлинности сообщения.
- Определить критерии для криптографической хэш-функции.
- Определить случайную модель Oracle и ее роль в оценке безопасности криптографических хэш-функций.
- Показать различие между кодом обнаружения модификации и кодом установления подлинности (аутентичности) документа (MDC и MAC).
- Обсудить некоторые общие MAC.

Это первая из трех лекций, посвященных целостности сообщения, установлению подлинности сообщения и установлению подлинности объекта. Здесь мы обсудим общие идеи, связанные с криптографическими хэш-функциями, которые используются, чтобы создать дайджест сообщения из сообщения. Дайджесты сообщения гарантируют целостность сообщения. Затем мы поговорим о том, как простые дайджесты сообщения могут быть модифицированы, чтобы подтвердить подлинность сообщения. Использованию криптографических хэш-функций в стандартной криптографии посвящена лекций 12.

11.1. Целостность сообщения

Системы криптографии, которые мы изучали до сих пор, обеспечивают *тайну (секретность)* или *конфиденциальность,* но не *целостность*. Однако есть случаи, где нам не нужна секретность, но зато необходима целостность (неизменность). Например, Алиса может написать завещание, чтобы распределить свое состояние после ее смерти. Завещание может не быть зашифрованным. После ее смерти любой может посмотреть это завещание. Целостность завещания, однако, должна быть сохранена, ибо Алиса не хочет, чтобы изменяли содержание завещания.

Документ и отпечатки пальцев

Одним из способов сохранить целостность документа мог бы стать способ с помощью *отпечатков пальцев.* Если Алисе надо быть уверенной, что содержание

ее документа не будет изменено, она может поместить отпечаток пальца внизу документа. Ева не может изменить содержание документа или создать ложный документ, потому что она не может подделать отпечаток пальца Алисы. Чтобы гарантировать, что документ не был изменен, отпечаток пальца Алисы на документе можно сравнить с отпечатком пальца Алисы в особом файле. Если они не совпадают, то документ — не от Алисы.

Сообщение и дайджест сообщения

Электронный эквивалент и пары «отпечаток пальца — документ» — это пара *сообщение-дайджест*. Чтобы сохранить целостность сообщения, оно обрабатывается алгоритмом, называемым **криптографической хэш-функцией**. Функция создает сжатое изображение сообщения, которое может использоваться подобно отпечатку пальца. Рисунок 11.1 показывает сообщение, криптографическую хэш-функцию и **дайджест сообщения.**

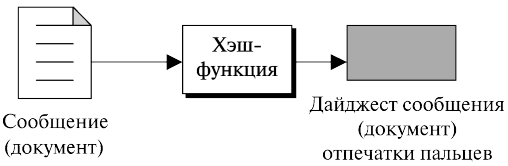


Рис. 11.1. Сообщение и дайджест

Различия

Эти две пары (документ / отпечаток пальца), и (дайджест сообщения / сообщение) имеют некоторые различия. Документ и отпечаток пальца физически связаны вместе, сообщение и дайджест сообщения могут быть разделены (или быть посланы) отдельно, и, что наиболее важно, дайджест сообщения должен быть защищен от изменения.

Дайджест сообщения должен быть защищен от изменения.

Проверка целостности

Чтобы проверить целостность сообщения или документа, мы создаем криптографическую хэш-функцию и сравниваем новый дайджест сообщения с предыдущим. Если они оба — те же самые, мы уверены, что первоначальное сообщение не было изменено. Рисунок 11.2 иллюстрирует идею.

Криптографические критерии хэш-функции

Криптографическая хэш-функция должна удовлетворять три критериям (см. рис. 11.3): **устойчивость к прообразу**, **устойчивость ко второму прообразу** и **устойчивость к коллизиям.**

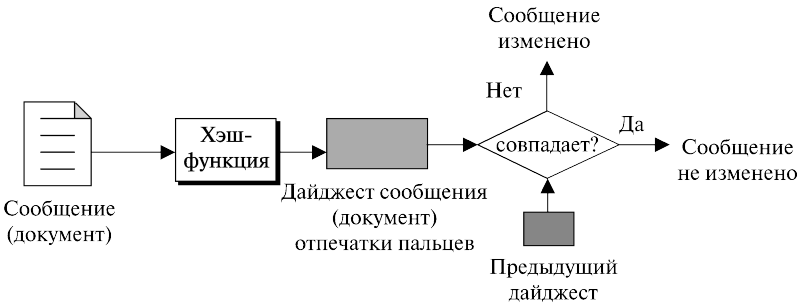


Рис. 11.2. Проверка целостности

Рис. 11.3. Критерии криптографической функции

Устойчивость прообраза

Криптографическая функция должна быть устойчива к прообразу. Если дана хэш-функция h и y = h(M), то для Евы должно быть экстремально трудно найти сообщение, такое, что y = h(M'). Рисунок 11.4 иллюстрирует эту идею.

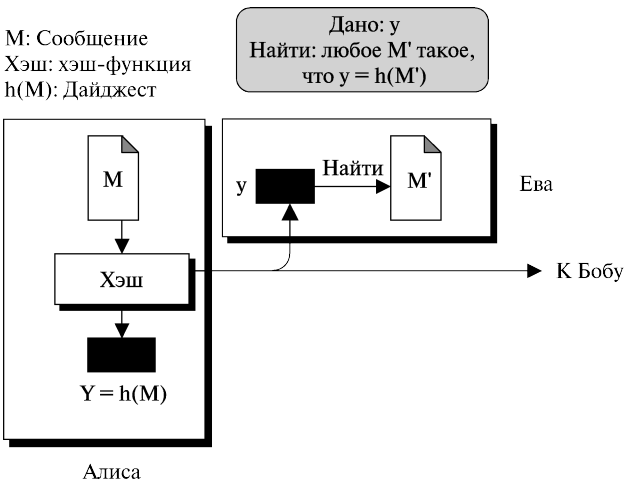


Рис. 11.4. Прообраз

Если хэш-функция — неустойчивый прообраз, Ева может перехватить дайджест h (M), создать сообщение M' и затем передать M' Бобу вместо исходного M.

Атака прообраза

Дано: y = h (M) **Найти: такое M', что y = h (M').**

Пример 11.1

Можем ли мы использовать обычный метод сжатия без потерь, такой, например, как zip, в криптографической хэш-функции?

Решение

Не можем. Метод сжатия без потерь создает сжатое сообщение, которое должно быть обратимо. Вы можете обработать сжатое сообщение, чтобы получить первоначальный текст.

Пример 11.2

Можем ли мы использовать функцию контрольной суммы как криптографическую хэш-функцию?

Решение

Не можем. Функция контрольной суммы — не стойкий прообраз. Ева может найти несколько сообщений, контрольная сумма которых соответствует данной.

Устойчивость ко второму прообразу

Второй критерий, **устойчивость ко второму прообразу,** гарантирует, что сообщение не может легко быть подделанным. Ева не может легко создать другое сообщение, которое преобразуется в тот же самый дайджест. Другими словами, учитывая заданное сообщение и его дайджест, невозможно (или, по крайней мере, очень трудно) создать другое сообщение с тем же самым дайджестом. Рисунок 11.5 иллюстрирует идею.

Ева перехватывает (имеет доступ к) сообщение M и его дайджест h(M). Она создает другое сообщение $M' \neq M$, но h (M) = h(M'). Ева передает M' и h (M') Бобу. Ева подделала сообщение.

Атака второго прообраза

Дана Атака: M и h (M) **Найти: такое $M' \neq M$, чтобы h (M) = h(M').**

Устойчивость к коллизиям

Третий критерий, **устойчивость к коллизиям,** гарантирует, что Ева не может найти два сообщения, которые приводят к тому же самому дайджесту. Здесь противник может создать два сообщения (из рабочего) и привести к тому же дайджесту. Мы увидим позже, как Ева может извлечь выгоду из этой слабости хэш-функции. Предположим, что в течение одного и того же момента времени созданы два различных завещания, которые могут быть приведены к одному тому же дайджесту. Когда наступает время для выполнения завещания, второе (подделанное) завещание представляется наследникам. Поскольку дайджест соответствует обоим завещаниям, подстановка не обнаружена. Рисунок 11.6 иллюстрирует идею. Мы увидим позже, что этот тип атаки намного проще начать, чем два предыдущих ви-

да. Другими словами, мы должны твердо убедиться, что хэш-функция устойчива к коллизиям.

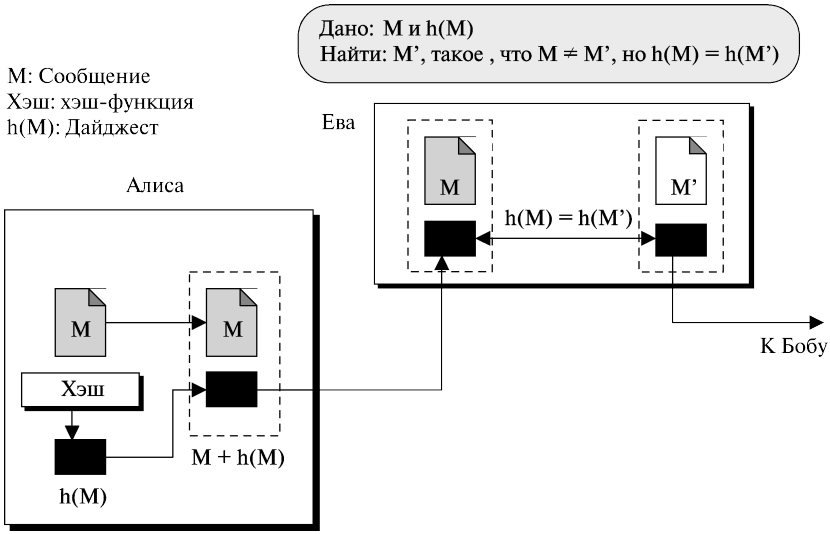


Рис.11.5.Второй прообраз

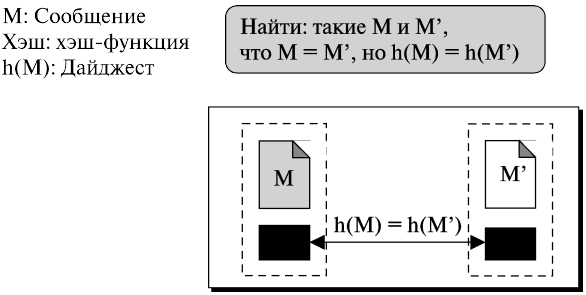


Рис. 11.6. Устойчивость к коллизиям

11.2. Случайная модель Oracle

Случайная модель Oracle была предложена в 1993 г. Белларом (Bellare) и Роджеем (Rogaway). Это идеальная математическая модель для хэш-функции. Функция, которая основана на этой модели, обладает следующими свойствами.

1. Когда поступает новое сообщение любой длины, Oracle порождает и вырабатывает на выходе дайджест-сообщения фиксированной длины, которые состоят из случайных строк нулей и единиц. Это oracle-запись сообщения и дайджест-сообщения.

2. Когда передается сообщение, для которого существует дайджест, oracle просто вставляет дайджест в запись.
3. Дайджест для нового сообщения должен быть выбран независимо от всех предыдущих дайджестов. Это подразумевает, что модель Oracle не может использовать формулу или алгоритм для вычисления дайджеста.

Пример 11.3

Возьмем модель Oracle с таблицей и «правильной» монетой. Таблица имеет два столбца. Левый столбец — сообщения, для которых должны быть выработаны дайджесты. Второй столбец перечисляет дайджесты, созданные для этих сообщений. Примем, что дайджест — всегда 16 битов независимо от размера сообщения. Таблица 11.1 показывает пример такой таблицы, в которой сообщение и дайджест сообщения приведены в шестнадцатеричном исчислении. Модель Oracle уже создала три дайджеста.

Таблица 11.1. Таблица Oracle после создания первых трех дайджестов

Сообщение	Дайджест сообщения
4523AB1352CDEF45126	13AB
723BAE38F2AB3457AC	02CA
AB45CD1048765412AAAB6662BE	A38B

Теперь предположим, что возникают два события:

a. Поступает сообщение AB1234CDS765BDAD для вычисления дайджеста. Oracle проверяет свою таблицу. Этого сообщения нет в таблице, так что сотрудник, использующий Oracle, подбрасывает в воздух свою монету 16 раз. Предположим, что результат — OOPOOOPPOPOOPPPO, в котором буква O представляет «*Орел*», буква P представляет «*Решка*». Oracle интерпретирует O как 1 бит и P как бит 0 и выдает 1101 1100 1011 0001 в двоичном коде либо DCB1 в шестнадцатеричном, как дайджест сообщения для этого сообщения, и складывает сообщение и дайджест в таблице (таблица 11.2).

Таблица 11.2. Таблица Oracle после создания четвертого дайджеста

Сообщение	Дайджест сообщения
4523AB1352CDEF45126	13AB
723BAE38F2AB3457AC	02CA
AB1234CD8765BDAD	DCB1
AB45CD1048765412AAAB6662BE	A38B

6. Сообщение 4523AB 1352CDEF45126 дается для вычисления дайджеста. Oracle проверяет свою таблицу и находит, что есть дайджест для этого сообщения в таблице (первая строка). Oracle просто выдает соответствующий дайджест (13AB).

Пример 11.4

Oracle в Примере 11.3. не может применить формулу или алгоритм, чтобы создать дайджест для сообщения.

Например, вообразим, что Oracle использует формулу h (M) = M mod *n*. Теперь предположим, что Oracle уже выдал h (M_1) и h (M_2). Если новое сообщение представлено как $M_3 = M_1 + M_2$, Oracle не должен вычислить h (M_3). Новый дайджест — только [h (M_1) + h (M_2)] mod n, поскольку:

$$(M3) = (M_1 + M_2) \bmod n = M_1 \bmod n + M_2 \bmod n = [h(M_1) + h(M_2)] \bmod n$$

Это нарушает третье требование: каждый дайджест должен быть выбран беспорядочно на основе сообщения, данного Oracle.

Принцип голубиных ящиков[1]

Первое понятие, с которым мы должны быть знакомы для того, чтобы понять анализ случайной Модели Oracle, — п**ринцип голубиных ящиков**: если *n* ящиков заняты *n* + 1 голубями, то по крайней мере один ящик занят двумя голубями. Обобщенная версия принципа голубиных ящиков: если ящиков *n* заняты *kn* +1 голубями, то по крайней мере один ящик занят *k* + 1 голубем.

Поскольку основная идея хэширования диктует, что дайджест должен быть короче, чем сообщение, согласно принципу голубиных ящиков могут быть конфликты. Другими словами, есть некоторые дайджесты, которые соответствуют больше чем одному сообщению; отношения между возможными сообщениями и возможными дайджестами — «много к одному».

Пример 11.5

Предположим, что сообщения в хэш-функции длиной 6 битов, дайджесты только длиной 4 бита. Тогда возможное число дайджестов (ящики) — от $2^4 = 16$ и возможное число сообщений (голуби) — $2^6 = 64$. Это означает $n = 16$ и $kn + 1 = 64$, так что *k* больше, чем 3. Это говорит о том, что по крайней мере один дайджест соответствует четырем (k + 1) сообщениям.

Проблемы дня рождения

Второе понятие, которое мы должны знать перед анализом случайной модели Oracle, известно как **проблема дня рождения.** Обычно в курсах теории вероятностей сталкиваются с четырьмя различными проблемами дня рождения, и третья из них иногда называется *парадокс дня рождения*. Рисунок 11.7 иллюстрирует смысл каждой проблемы.

[1] Принцип голубиных ящиков — по-английски pigeonhole principle. Второй англоязычный термин — Dirihlet's box Principle (принцип ящика Дирихле). Надо иметь в виду, что англоязычный термин Dirichlet principle (принцип Дирихле) обозначает и другие понятия, не относящиеся к криптографии *(прим. переводчика)*.

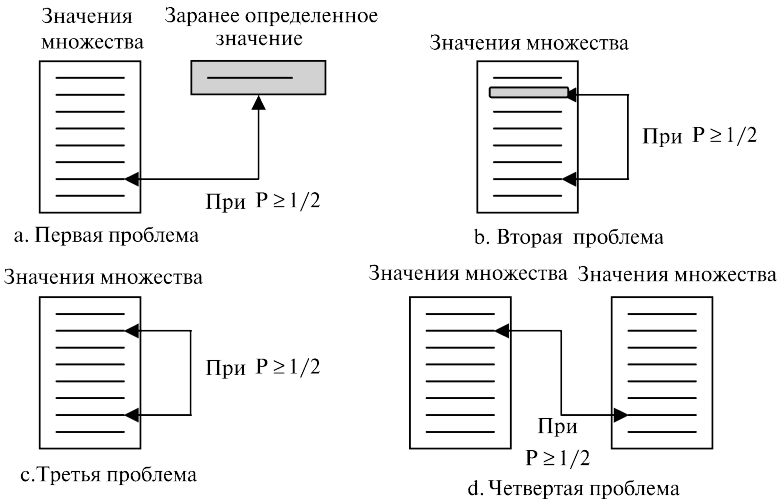


Рис.11.7. Четыре проблемы дня рождения

Описание проблем

Ниже приводятся проблемы дня рождения, выраженные в терминах, которые могут быть применены к хэш-функциям безопасности. Обратите внимание, что термин с *некоторой вероятностью* означает случаи, когда вероятность события $P \geq 1/2$.

Проблема 1. Каково минимальное число k студентов в классной комнате, такое, что *с некоторой вероятностью* по крайней мере один студент имеет заранее заданный день рождения? Эта проблема может быть обобщена следующим образом. Мы имеем однородно распределенную случайную переменную с N возможными значениями (между 0 и $N-1$). Каково минимальное число экземпляров, таких, что *с некоторой вероятностью* по крайней мере один экземпляр равен заранее заданному значению?

Проблема 2. Каково минимальное число k студентов в классной комнате, такое, что *с некоторой вероятностью* по крайней мере один студент имеет тот же самый день рождения, как и студент, выбранный профессором? Эта проблема может быть обобщена следующим образом. Мы имеем однородно распределенную случайную переменную с N возможными значениями (между 0 и $N-1$) Какое минимальное число экземпляров, k, таких, что *с некоторой вероятностью* по крайней мере один экземпляр является равным выбранному?

Проблема 3. Каково минимальное число k студентов в классной комнате, такое, что *с заданной вероятностью* по крайней мере два студента имеют тот же самый день рождения? Эта проблема может быть обобщена следующим образом. Мы имеем однородно распределенную случайную переменную с N возможными значениями (между 0 и $N-1$). Каково минимальное число экземпляров k, таких, что *с некоторой вероятностью* по крайней мере два экземпляра равны?

Проблема 4. Мы имеем два класса, каждый с k студентами. Каково минимальное значение A, такое, чтобы по крайней мере один студент из первой классной комнаты *с некоторой вероятностью* имел тот же самый день рождения, что и студент из второй классной комнаты? Эта проблема может быть обобщена следующим образом. Мы имеем однородно распределенную случайную переменную N со значениями (между 0 и $N - 1$). Мы генерируем два множества случайных значений, каждое величиной k. Каково минимальное число k, такое, что *с некоторой вероятностью* по крайней мере один экземпляр первого множества равен одному образцу во втором множестве?

Результаты решений

Для заинтересованных читателей решения этих проблем даются в приложении E. Результаты приведены в таблице 11.3.

Таблица 11.3. Результаты решений четырех проблем дней рождения

Проблема	Вероятность	Общее значение для k	Значение k при P = 1/2	Число студентов (N=365)
1	$P \approx 1 - e^{-k/N}$	$k \approx \ln[1/(1-P) \times N$	$k \approx 0{,}69 \times N$	253
2	$P \approx 1 - e^{-(k-1)N}$	$k \approx \ln[1/(1-P) \times N + 1$	$k \approx 0{,}69 \times N$	254
3	$P \approx 1 - e^{k(k-1)/2N}$	$k \approx \{2 \ln [1/1-P]\}^{1/2} \times N^{1/2}$	$k \approx 1{,}18 \times N^{1/2}$	23
4	$P \approx 1 - e^{k^2/2N}$	$k \approx \{\ln [1/1-P]\}^{1/2} \times N^{1/2}$	$k \approx 0{,}83 \times N^{1/2}$	16

Затемненное значение, 23, является решением классического парадокса дня рождения; если есть 23 студента в классной комнате, то с *некоторой вероятностью* (с $P \geqslant 1/2$) два студента имеют одинаковый день рождения (игнорируя год их рождения).

Сравнение проблем

Значение k в проблемах 1 или 2 пропорционально N; значение k в проблемах 3 или 4 является пропорциональным $N^{1/2}$. Коротко скажем, что первые две проблемы связаны с атаками прообраза и второго прообраза; третья и четвертая проблемы связаны с атакой коллизии. Сравнение показывает, что намного более трудно начать атаку прообраза или атаку второго прообраза, чем атаку коллизии. Рисунок 11.8 дает граф P при различных k. Для первой и второй проблем показан один граф (значения вероятностей — очень близки). Графы для второй и третьей проблем отличаются сильнее.

Атаки случайной модели Oracle

Чтобы лучше понимать характер хэш-функций и важность случайной модели Oracle, рассмотрим, как Ева может атаковать хэш-функцию, созданную Oracle. Предположим, что хэш-функция создает дайджесты n битов. Тогда дайджест можно представить как случайную переменную, однородно распределенную между 0

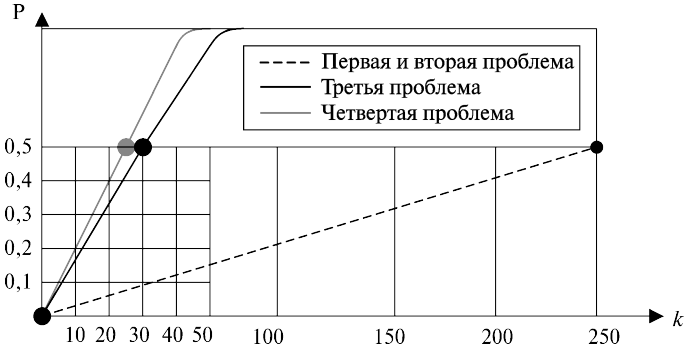


Рис. 11.8. Граф четырех проблем дня рождения

и $N - 1$, в которой $N = 2^n$. Другими словами, есть возможные 2^n значений для дайджеста; каждый раз Oracle случайно выбирает одно из этих значений для сообщения. Обратите внимание: это не означает, что выбор является исчерпывающим. Некоторые значения могут никогда не выбираться, но некоторые могут быть выбраны несколько раз. Мы принимаем, что алгоритм хэш-функции общедоступен и Ева знает размер дайджеста n.

Атака прообраза

Ева перехватила дайджест D = h (M); она хочет найти любое сообщение M', такое, что D = h (M'). Ева может создать список k сообщений и выполнить алгоритм 11.1.

Алгоритм может найти сообщение, для которого D является дайджестом, или может потерпеть неудачу. Какова вероятность успеха этого алгоритма? Очевидно, это зависит от размера списка, k, выбранного Евой. Чтобы найти вероятность, мы используем первую проблему дня рождения. Дайджест, созданный в соответствии с программой, определяет результаты случайной переменной. Вероятность успеха — $P = 1 - e^{-k/N}$.

Алгоритм 11.1. Атака на прообраз

```
Preimage_Attack (D)
{
   for (i =1 to k)
   {
      создать (M[i])
      T ← h(M [i ])  // T - временный дайджест
      if (T = D) return M[i]
   }
   return failure
{
```

Какой должен быть размер k, если Ева должна достигнуть успеха по крайней мере в 50 процентах случаев? Мы показали это значение в таблице 11.3. Для первой проблемы дня рождения: $k \approx 0{,}69 \times N$, или $k \approx 0{,}69 \times 2^n$. Другими словами, чтобы Ева добилась своего более чем в 50 процентах случаев, она должна создать список дайджеста, который пропорционален 2^n.

Сложность атаки прообраза пропорциональна 2^n.

Пример 11.6

Криптографическая хэш-функция использует дайджест 64 бита. Сколько дайджестов Ева должна создать, чтобы найти первоначальное сообщение с вероятностью большей, чем 0,5?

Решение

Число дайджестов, которые будут созданы, — $k \times 0{,}69 \times 2^n = 0{,}69 \times 2^{64}$. Это большое значение. Даже если Ева сможет генерировать 2^{30} (почти один миллиард) сообщений в секунду, требуется $0{,}69 \times 2^{34}$ секунды или больше чем 500 лет. Это означает, что дайджест сообщения размером 64 бита является безопасным относительно атаки прообраза, но, как мы увидим далее, он не защищен от атаки коллизии.

Атака второго прообраза

Ева перехватила дайджест D = h (M) и соответствующее сообщение M.; она хочет найти другое сообщение M', такое, чтобы h(M') = D. Ева может создать список из $k - 1$ сообщения и выполнить Алгоритм 11.2.

Алгоритм 11.2. Атака второго прообраза

```
Second_Preimage_Attack (D, M)
{
    for (i = 1 to k - 1)
    {
        создать (M[i]
        T ← h (M[i])   // T – временный дайджест
        If (T = D) return M[i]
    }
    return failure
}
```

Алгоритм может найти второе сообщение, для которого D является также дайджестом, или может потерпеть неудачу. Какова вероятность успеха этого алгоритма? Очевидно, это зависит от размера списка, k, выбранного Евой. Чтобы найти вероятность положительного исхода алгоритма, мы используем вторую проблему дня рождения. Дайджест, созданный в соответствии с программой, определяет выходное значение случайной переменной. Вероятность успеха – $P \approx 1 - e^{1-(k-1)/N}$. Какой должен быть размер k, если Ева хочет достичь успеха по край-

ней мере в 50 процентах случаев? Мы уже указали это значение в таблице 11.3 для второй проблемы дня рождения: $k \approx 0{,}69 \times N + 1$ или $k \approx 0{,}69 \times 2^n$. Другими словами, чтобы Ева добилась своего более чем в 50 процентах случаев, она должна создать список дайджеста, который пропорционален 2^n.

Сложность атаки второго прообраза пропорциональна 2^n.

Атака коллизии

Ева должна найти два сообщения, M. и M.', такие, что h (M) = h (M.'). Она может создать список сообщений и выполнить алгоритм 11.3.

Алгоритм 11.3. Атака коллизии

```
Collision_Attack
{
   for (i = 1 to k)
   {
      создать (M[i])
      D[i] ← h (M[i])   // D[i] – список создаваемых дайджестов
      for (j = 1 to i - 1)
      {
         if (D[i] = D[j] return (M[i] и M[j] )
      }
   }
   return failure
}
```

Алгоритм может найти два сообщения с одним и тем же дайджестом. Какова вероятность успеха этого алгоритма? Очевидно, это зависит от размера списка, k, выбранного Евой. Чтобы найти вероятность события, мы используем третью проблему дня рождения. Дайджест, созданный в соответствии с программой, определяет выходное значение случайной переменной. Вероятность успеха — $P = 1 - e^{1-(k-1)/2N}$. Какой должен быть размер k, если Ева хочет достичь успеха по крайней мере в 50 процентах случаев? Мы уже указали это значение в таблице 11.3 для третьей проблемы дня рождения: $k \approx 1{,}18 \times N^{1/2}$, или $k \approx 1{,}18 \times 2^{n/2}$. Другими словами, чтобы Ева добилась своего более чем в 50 процентах случаев, она должна создать список дайджеста, который пропорционален $2^{n/2.}$

Сложность атаки коллизии пропорциональна $2^{n/2}$.

Пример 11.7

Криптографическая хэш-функция использует дайджест 64 бита. Сколько дайджестов Ева должна создать, чтобы найти два сообщения с тем же самым дайджестом с вероятностью больше чем 0,5?

Решение

Число дайджестов, которые будут созданы, $k \approx 1{,}18 \times 2^{n/2} \approx 1{,}18 \times 2^{32}$. Если Ева может проверить 2^{20} (почти один миллион) сообщений в секунду, потребуется $1{,}18 \times 2^{12}$ секунд, или меньше чем два часа. Это означает, что дайджест сообщения размера 64 бита небезопасен относительно атаки коллизии.

Дополнительная атака коллизии

Предыдущая атака коллизии не может быть полезна для Евы — ей приходится создать два сообщения, одно реальное и одно фиктивное, которые имеют одно и то же значение хэш-функции. Каждое сообщение должно быть значащим. Предыдущий алгоритм не обеспечивает устойчивость коллизии, решение состоит в том, чтобы создать два значащих сообщения, но добавить избыточность к сообщению или модифицировать сообщение, чтобы изменить содержание слова, или прибавлять некоторые избыточные слова, и так далее, в сообщение, не изменяя значение его хэша. Например, множество сообщений может быть создано из первого сообщения — добавлением пробелов или изменением слова, когда некоторые слова дополняются окончаниями.

Второе сообщение может также создать множество сообщений. Если обозначить первоначальное сообщение M, а фиктивное сообщение — M', Ева создает k различных вариантов M ($M_1, M_2, \ldots, M_k$) и k различных вариантов M' ($M'_1, M'_2, \ldots, M'_k$) . Затем Ева использует алгоритм 11.4 для того, чтобы начать атаку.

Алгоритм 11.4. Дополнительная атака коллизии

```
Alternate_Collision _Attack (M[k], M'[k])
{
   for (i = 1 to k)
   {
      D[i] ← h ([M[i])
      D'[i] ← h [M'[i])
      if (D[i] = D'[j] return (M[i], M'[j])
   }
   return failure
}
```

Какова вероятность успеха этого алгоритма? Очевидно, это зависит от размера списка k, выбранного Евой. Чтобы найти вероятность, мы используем четвертую проблему дня рождения. Два списка дайджеста, созданные в соответствии с программой, определяют два выходных значения случайной величины. Вероятность успеха равна $P \approx 1 - e^{-k^2/N}$. Какой должен быть размер k, если Ева хочет достичь успеха по крайней мере в 50 процентах случаев? Мы уже указали это значение в таблице для четвертой проблемы дня рождения: $k \approx 0{,}83 \times N^{1/2}$ или $k \approx 0{,}83 \times 2^{n/2}$. Другими словами, чтобы Ева добилась своего более чем в 50 процентах случаев, она должна создать список дайджеста, который пропорционален $2^{n/2}$.

Сложность дополнительной атаки коллизии пропорциональна $2^{n/2}$.

Итоги атак

Таблица 11.4 показывает уровень сложности для каждой атаки, если дайджест имеет длину *n* бит.

Таблица 11.4. Уровни сложности для каждого типа атаки

Атака	Значение при P =1/2	Порядок верхнего предела
Прообраз	$k \approx 0{,}69 \times 2^{n}+1$	2^{n}
Второй прообраз	$k \approx 0{,}69 \times 2^{n}+1$	2^{n}
Коллизия	$k \approx 1{,}18 \times 2^{n/2}$	$2^{n/2}$
Дополнительная коллизия	$k \approx 0{,}83 \times 2^{n/2}$	$2^{n/2}$

Таблица 11.4 иллюстрирует, что порядок или сложность атаки для атаки коллизии является намного меньшим, чем для прообраза или атак второго прообраза. Если алгоритм хэша является стойким к атакам коллизии, мы можем не волноваться о безопасности атак прообраза и атак второго прообраза.

Пример 11.8

Первоначально хэш-функции с дайджестом на 64 бита, как полагали, были стойкими к атакам коллизии. Но с увеличением скорости обработки все обнаружили, что эти хэш-функции больше не безопасны. Ева нуждается только в $2^{64/2} = 2^{32}$ испытаний, чтобы начать атаку с вероятностью 1/2 или больше. Предположим, что она может выполнить 2^{20} (один миллион) испытаний в секунду. Она может провести атаку за $2^{32} / 2^{20} = 2^{12}$ секунд (почти час!).

Пример 11.9

MD5 (см. лекцию 12), который был одной из стандартных хэш-функций в течение долгого времени, создает дайджесты в 128 битов. Чтобы провести атаку коллизии, противник должен провести 2^{64} ($2^{128/2}$) испытаний алгоритма коллизии. Даже если противник может выполнить 2^{30} (больше чем один миллиард) испытаний в секунду, требуется 2^{34} секунды (больше чем 500 лет), чтобы провести атаку. Этот тип атаки базируется на случайной модели Oracle. Было доказано, что MD5 может быть атакован за менее чем 2^{64} испытаний — из-за структуры алгоритма.

Пример 11.10

SHA-1 (см. лекцию 12), стандартная хэш-функция, разработанная NIST, создает дайджесты в 160 битов. Чтобы провести атаку коллизии, противник должен исполнить $2^{160/2} = 2^{80}$ испытания в алгоритме коллизии. Даже если противник может выполнить 2^{30} (больше чем один миллиард) испытаний в секунду, требуется 2^{50} секунд (больше чем десять тысяч лет), чтобы начать атаку. Однако исследователи обнаружили некоторые особенности функции, которые позволяют провести атаку на эту хэш-функцию за меньшее время, чем вычисленное выше.

Пример 11.11

Новая хэш-функция, которая, вероятно, станет NIST-стандартом, — SHA-512 (см. лекцию 12) имеет дайджест на 512 битов. Эта функция явно стойкая к ата-

кам коллизии, основанным на случайной модели Oracle. Требуется от $2^{512/2}$ до 2^{256} испытаний, чтобы найти коллизию с вероятностью 1/2.

Атаки структуры

Все обсуждения, связанные с атаками хэш-функций, базировались на идеальной криптографической хэш-функции, которая похожа на Oracle. Хотя этот тип анализа обеспечивает систематическую оценку алгоритмов, практически хэш-функции могут иметь некоторые внутренние структуры, которые могут сделать их намного слабее.

Невозможно найти хэш-функцию, которая создает дайджесты, чтобы она делала их полностью случайными. Противник может иметь другие инструментальные средства для того, чтобы атаковать хэш-функцию. Одно из этих инструментальных средств, например, атака сведения к середине, которая обсуждалась в лекции 6 для двукратного DES. Мы увидим в следующих лекциях, что некоторые хэш-алгоритмы поддаются этому типу атаки. Эти типы хэш-функции далеки от идеальной модели, и нужно их избегать.

11.3. Установление подлинности сообщения

Дайджест сообщения гарантирует целостность сообщения — то есть что сообщение не было изменено. Дайджест сообщения, однако, не подтверждает подлинность передачи сообщения. Когда Алиса передает сообщение Бобу, Боб должен знать, что это сообщение точно прибыло от Алисы. Для того, чтобы обеспечить установление подлинности сообщения, Алиса должна предоставить доказательство, что это сообщение послала именно она, а не самозванец. Дайджест сообщения не может обеспечить такое доказательство. Дайджест, созданный криптографической хэш-функцией, обычно называется *кодом обнаружения модификации* (Modification Detection Code — MDC), — код может обнаружить любое изменение в сообщении. Кроме этого, мы нуждаемся в аутентификации сообщения (установление подлинности происхождения данных) — в *коде установления подлинности сообщения* (Message Authentication Code — MAC).

Код обнаружения модификации

Код обнаружения модификации (MDC) — дайджест сообщения, который может доказать целостность сообщения и подтвердить, что сообщение не было изменено. Если Алиса должна передать сообщение Бобу и хочет быть уверена, что сообщение не будет изменено во время передачи, она может создать дайджест, MDC-сообщение и послать сообщение и MDC Бобу. Боб может создать новый MDC из сообщения и сравнить полученный MDC и новый MDC. Если они одинаковые, значит, сообщение не был изменено. Рисунок 11.9 иллюстрирует идею.

Рисунок 11.9 показывает, что сообщение может быть передано через ненадежный канал. Ева может читать или даже изменять сообщение. MDC, однако, должен быть передан через безопасный канал, такой канал называют *безопасный (safe)*, он не позволяет изменений.

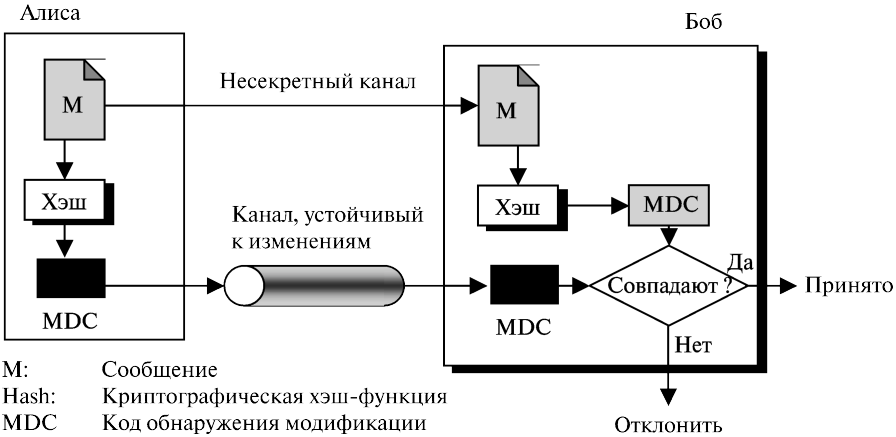


Рис. 11.9. Код обнаружения модификации

Если сообщение и MDC передаются через ненадежный канал, Ева может перехватить сообщение, изменить его, создать новый MDC из сообщения и передать их Бобу. Боб никогда не узнает, что сообщение пришло от Евы. Обратите внимание, что термин *безопасный* может означать, что мы имеем дело с партнером, которому доверяем. Термин *канал* может означать наличие временного промежутка между передачей и приемом. Например, если Алиса создает MDC своего завещания и передает его нотариусу, который сохраняет блокированное сообщение до ее смерти, — она использовала безопасный канал.

Алиса пишет свое завещание и объявляет это публично (ненадежный канал). Алиса делает MDC из сообщения и передает его нотариусу, который сохраняет его до ее смерти (безопасный канал). Хотя Ева может изменить содержание завещания, поверенный может создать MDC завещания и доказать, что версия Евы — подделка. Если хэш-функция криптографии используется для создания MDC, описанные в начале этой лекции три свойства Евой будут потеряны.

Код установления подлинности сообщения (Message Authentication Code — MAC)

Чтобы гарантировать целостность сообщения, подлинность первоначального сообщения и то, что создатель сообщения — Алиса, а не кто-то другой, мы должны изменить код обнаружения модификации (MDC) на **код установления подлинности сообщения (Message Authentication Code — MAC)**. Отличие между MDC и MAC в том, что второй включает секретность между Алисой и Бобом, например секретный ключ, которым Ева не обладает. Рисунок 11.10 иллюстрирует идею.

Алиса использует хэш-функцию, чтобы создать MAC по объединению (конкатенации) ключа и сообщения — h (K|M). Она передает сообщение и MAC Бобу по ненадежному каналу. Боб отделяет сообщение от MAC и затем делает новый MAC из объединения сообщения и ключа засекречивания. Затем Боб сравнивает

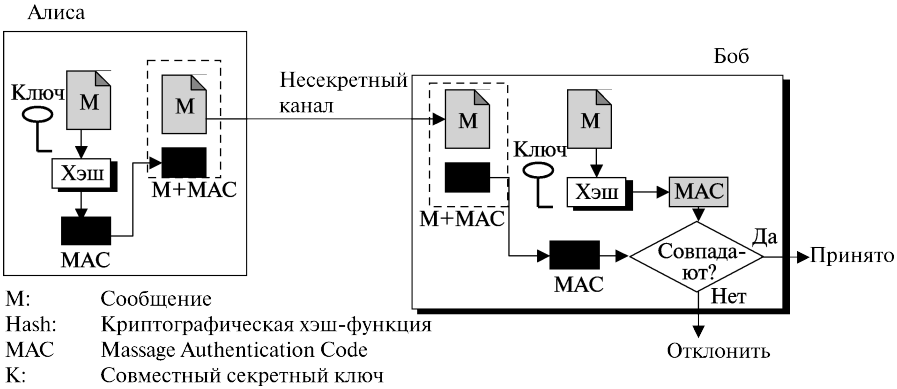


Рис. 11.10. Код установления подлинности сообщения

недавно созданный MAC с полученным. Если оба эти MAC совпадают, то сообщение подлинное и не было изменено противником.

Обратите внимание, что в этом случае нет необходимости использовать два канала. И сообщение, и MAC можно передать по одному, хотя бы и самому ненадежному каналу. Ева может видеть сообщение, но она не может создать новое сообщение, чтобы подделать исходное, потому что Ева не обладает ключом засекречивания между Алисой и Бобом. Она неспособна создать такой же MAC, как это сделала Алиса.

MAC, который мы описали, упоминается как префикс MAC, потому что ключ засекречивания добавлен в конец к началу сообщения. Мы можем иметь MAC с префиксом в конце, — ключ добавлен в конец после конца сообщения. Мы можем комбинировать места префикса и MAC, с тем же самым ключом или двумя различными ключами. Однако MAC в конце сообщения ненадежны.

Безопасность MAC

Предположим, что Ева перехватила сообщение M и дайджест h (K|M). Как Ева может подделать сообщение, не зная ключа засекречивания? Есть три возможных случая.

1. Если размер ключа позволяет полный перебор, Ева может перебрать все возможные ключи в начале сообщения и сделать дайджест (K|M), чтобы найти, что этот дайджест равняется перехваченному. Она уже знает ключ и может успешно заменить сообщение подделанным сообщением по своему выбору.
2. Размер ключа в MAC является обычно очень большим, но Ева может использовать другой инструмент — атаку прообраза, рассмотренную в алгоритме 11.1. Она использует алгоритм, пока не находит X, такой, что h (X) равен MAC, который она перехватила. Теперь она может найти ключ и успешно заменить сообщение подделанным. Поскольку размер ключа обычно очень большой, для полного перебора Ева может только атаковать MAC, который применяет алгоритм прообраза.

3. Получая некоторые пары сообщений и их MAC, Ева может управлять ими, чтобы придумать новое сообщение и его MAC.

Безопасность MAC зависит от безопасности основного хэш-алгоритма.

Вложенный MAC

Чтобы улучшить безопасность MAC, был разработан **вложенный MAC (nested MAC)**, в котором хэширование делается в два шага. На первом шаге ключ конкатенируется (последовательно объединяется) с сообщением и хэшируется, чтобы создать промежуточный дайджест. На втором шаге ключ конкатенируется с промежуточным дайджестом, чтобы создать конечный дайджест. Рисунок 11.11 иллюстрирует общую идею.

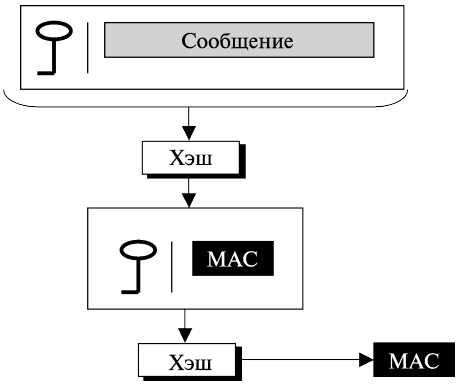


Рис. 11.11. Вложенный MAC

Код аутентификации сообщения, основанный на хэшировании (HMAC)

Национальный институт стандартов США (NIST) разработал стандарт (FIPS 198) для вложенного MAC, который часто называют **HMAC (HASH-BASED MESSAGE AUTHENTICATION CODE)** (его надо отличать от CMAC, который будет рассмотрен в следующем разделе). Реализация HMAC намного более сложна, чем упрощенный вложенный MAC, показанный на рис. 11.11. Есть дополнительные особенности, такие как заполнение. Рисунок 11.12 показывает детали. При реализации HMAC мы проходим следующие шаги:

1. Сообщение разделяется на N блоков, каждый по b битов.
2. Ключ засекречивания дополняется слева нулями, чтобы создать ключ длиной b бит. Обратите внимание: рекомендуется, чтобы ключ засекречивания, прежде чем он будет дополнен, был длиною более чем n бит, где n — размер HMAC.
3. Результат шага 2 складывают по модулю два с константой, называемой **ipad (входной блокнот)**, чтобы создать блок b бит. Значение ipad — $b/8$ — состоит из повторяемой последовательности 00110110 (36 в шестнадцатеричном исчислении).

4. Блок результата присоединим спереди к сообщению из N-блоков. В результате получим $N + 1$ блоков.
5. Результат шага 4 хэшируется, чтобы создать дайджест длиною n-битов. Мы называем этот дайджест промежуточным HMAC.

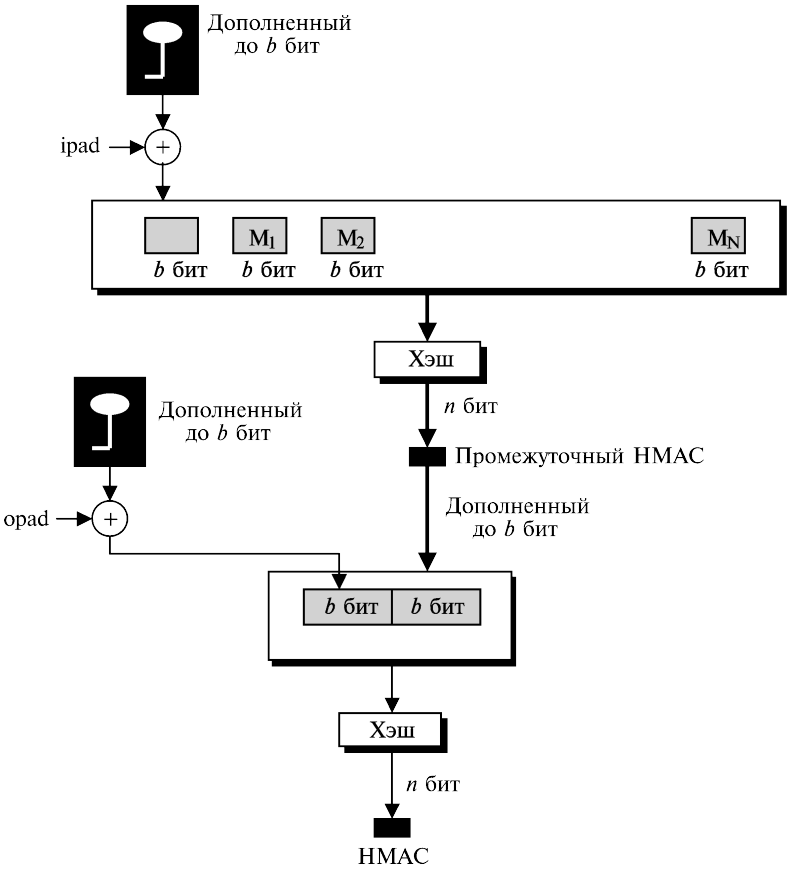


Рис. 11.12. Детали HMAC

6. Промежуточный n-битовый HMAC дополняют слева нулями, чтобы создать b-битовый блок.
7. Шаги 2 и 3 повторяются с другой константой opad **(выходной блокнот).** Значение opad — *b/8* — состоит из повторяемой последовательности 01011100 (5C в шестнадцатеричном исчислении).
8. Блок результата шага 7 присоединим спереди к блоку шага 6.
9. Результат шага 8 хэшируется, тем же самым алгоритмом хэширования, что и в п. 4, чтобы создать конечный n-разрядный HMAC.

CMAC

Национальный институт стандартов и технологии США (NIST) разработал стандарт (FIPS113), названный Алгоритмом установления подлинности данных или кодом аутентификации сообщения, основанный на шифровании базового сообщения — CMAC (Cipher based Message, Authentication Code) или **CBCMAC**. Метод подобен режиму сцепления блоков шифрованного текста (CBC – Cipher Block Chaining), рассмотренному в лекции 8 для шифрования симметричными ключами. Рисунок 11.13 иллюстрирует идею.

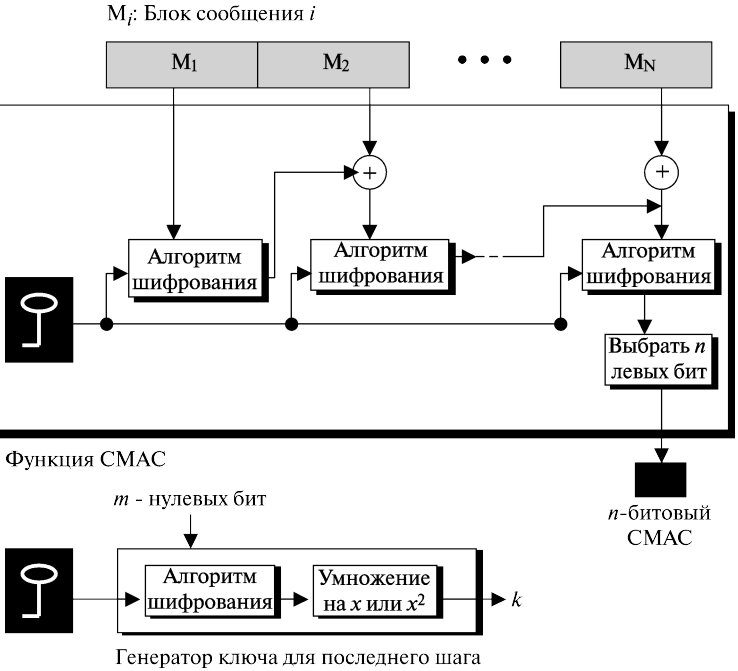


Рис. 11.13. CMAC

Однако смысл здесь состоит не в том, чтобы создавать N блоков зашифрованного текста из N блоков исходного текста. Идея в том, чтобы создать один блок MAC из N блоков исходного текста, используя N раз шифрование с симметричным ключом.

Сообщение разделено на N блоков, каждый длины m бит. Размер CMAC — n бит. Если последний блок — не m бит, он дополняется единичным битом (1), сопровождаемым достаточным количеством нулей (0), чтобы сделать его m-битовым. Первый блок сообщения зашифрован симметричным ключом, чтобы создать m-разрядный блок зашифрованных данных. Этот блок складывается (ИСКЛЮЧАЮЩЕЕ ИЛИ) со следующим блоком, а результат зашифровывается снова, чтобы создать новый m-битовый блок. Процесс продолжается, пока не бу-

дет зашифрован последний блок сообщения. CMAC — это *n* крайних левых бит последнего блока. В дополнение к симметрическому ключу, K, CMAC также использует другой ключ, k, который применяется только на последнем шаге. Этот ключ получен с помощью алгоритма шифрования исходного текста, дополненного m нулевыми битами, и использованием шифро-ключа, K. Результат затем умножен на x, если нет никакого дополнения, или на x^2, если дополнение есть. Умножение проводится в GF (2^m) с неприводимым полиномом степени m, выбранным в соответствии с используемым конкретным протоколом.

Обратите внимание, что эта процедура отличается от CBC (см. лекцию 8), метода, который применяется для получения конфиденциальности. В упомянутом ранее методе выход каждого шифрования передают как зашифрованный текст и в то же самое время складывают (ИСКЛЮЧАЮЩЕЕ ИЛИ) со следующим блоком исходного текста. Здесь же промежуточные зашифрованные блоки не передаются как зашифрованный текст; они только используются для сложения со следующим блоком.

11.4. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Книги [Sti06], [Sta06], [Sch99], [Mao04], [KPS02], [PHS03L] и [MOV96] дают хороший обзор криптографических хэш-функций.

Сайты

Нижеследующие сайты содержат больше информации о темах, рассмотренных в этой лекции.

http://en.wikipedia.org/wiki/Preimage_attack
http://en.wikipedia.org/wiki/Collision_attack#In_cryptography
http://en.wikipedia.org/wiki/Pigeonhole __ principle
csrc.nist.gov/ispab/2005-12/B_Burr-Dec2005-ISPAB.pdf
http://en.wikipedia.org/wiki/Message_authentication_code
http://en.wikipedia.org/wiki/HMAC
csrc.nist.gov/pLiblicationVnps/npsl98/fips-198a.pdf
http://www.faqs.org/rfcs rfc2104.html
http://en.wikipedia.org/wiki/Birthday_paradox

11.5. Итоги

- Отпечаток пальца или дайджест сообщения могут использоваться для того, чтобы гарантировать целостность документа или сообщения. Чтобы гарантировать целостность документа, необходимы документ и отпечаток

пальца; чтобы гарантировать целостность сообщения, необходимы сообщение и дайджест сообщения. Дайджест сообщения надо оберегать от изменения.

- Криптографическая хэш-функция создает дайджест сообщения из сообщения. Функция должна соответствовать трем критериям: устойчивость к прообразу, устойчивость ко второму прообразу и устойчивость к коллизиям.
- Первый критерий — устойчивость к прообразу — означает, что для Евы должно быть чрезвычайно трудно создать любое сообщение, соответствующее этому дайджесту. Второй критерий — устойчивость второго прообраза — гарантирует, что если Ева имеет сообщение и соответствующий дайджест, она не сможет создать второе сообщение, дайджест которого тот же самый, что и у первого. Третий критерий — устойчивость к коллизиям — гарантирует, что Ева не может найти два сообщения, которые хэшируются и приводят к одному и тому же дайджесту.
- Случайная модель Oracle, которая была введена в 1993 г. Белларом и Роджеем, является идеальной математической моделью для хэш-функции.
- Принцип голубиных ящиков устанавливает, что если n ящиков заняты $n+1$ голубем, то по крайней мере один ящик занят двумя голубями. Обобщенная версия принципа голубиных ящиков: если n ящики заняты $kn + 1$ голубем, по крайней мере один ящик занят $k + 1$ голубем.
- Проблемы четырех дней рождения используются, чтобы проанализировать случайную модель Oracle. Первая проблема нужна, чтобы проанализировать атаку прообраза, вторая проблема — чтобы проанализировать атаку второго прообраза, а третья и четвертая проблемы нацелены на атаку коллизии.
- Код обнаружения модификации (MDC) — дайджест сообщения, который может доказать целостность сообщения: что сообщение не было изменено. Чтобы доказать целостность сообщения и установить подлинность происхождения данных, мы должны заменить код обнаружения модификации (MDC) на код установления подлинности сообщения (MAC). Различие между MDC и MAC в том, что MAC включает в себя безопасность передачи между передатчиком и приемником.
- Национальный Институт Стандартов и Технологии США (NIST) выработал стандарт (FIPS 198) для вложенного MAC, который часто называется кодом аутентификации сообщения, основанным на хэшировании — HMAC (хэшированный MAC). NIST также определил другой стандарт (FIPS 113), названный CMAC, или CBCMAC.

11.6. Набор для практики

Обзорные вопросы

1. Покажите различия между целостностью сообщения и установлением подлинности сообщения
2. Определите первый критерий для криптографической хэш-функции.
3. Определите второй критерий для криптографической хэш-функции.

4. Определите третий критерий для криптографической хэш-функции.
5. Определите случайную модель Oracle и дайте описание ее приложений при анализе атак хэш-функций.
6. Установите принцип голубиных ящиков и опишите его приложение при анализе хэш-функций.
7. Определите проблемы четырех дней рождения, рассмотренные в этой лекции.
8. Испробуйте каждый метод дня рождения с одной проблемой из атак хэш-функции.
9. Покажите различия между MDC и MAC.
10. Покажите различия между HMAC и CMAC.

Упражнения

1. В случайной модели Oracle: почему Oracle должен делать запись дайджеста, созданного для сообщения, и присваивать тот же самый дайджест одинаковым сообщениям?
2. Объяснить, почему секретный/открытый ключи не могут использоваться в создании MAC.
3. Игнорируя месяц рождения, сколько попыток в среднем необходимо, чтобы найти человека с такой же датой рождения, как ваша? Примите, что все месяцы имеют 30 дней.
4. Игнорируя месяц рождения, сколько попыток в среднем необходимо, чтобы найти двух человек с одинаковой датой рождения? Примите, что все месяцы имеют 30 дней.
5. Сколько попыток в среднем необходимо, чтобы найти человека того же возраста, что и вы, учитывая группу людей, рожденных после 1950?
6. Сколько попыток в среднем необходимо, чтобы найти двух человек одного и того же возраста, если мы ищем людей, рожденных после 1950?
7. Ответьте на следующие вопросы о семье из шести человек. Предположим, что их дни рождения:
   — однородно распределены в течение дней недели,
   — в течение дней месяца, в течение каждого месяца года,
   — в течение 365 дней года.

   Предположим также, что год состоит точно из 365 дней и каждый месяц — точно из 30 дней.
   a. Какова вероятность, что два из членов семьи имеют один и тот же день рождения? Какова вероятность, что ни один из них не имеет совпадающего дня рождения?
   b. Какова вероятность, что двое из членов семьи рождены в одном и том же месяце? вероятность того, что ни один из них не был рожден в одном и том же месяце?
   c. Какова вероятность, что кто-то из членов семейства рожден в первый день одного из месяцев?
   d. Какова вероятность, что у трех из членов семейства дни рождения приходятся на один и тот же день недели?

8. Какова вероятность совпадения дней рождения в двух классах, одного с *k* студентами и другого с *l* студентами?
9. В классе из 100 студентов какова вероятность, что два или больше студента имеют паспорта с одними и теми же последними четырьмя цифрами?
10. Есть 100 студентов в группе, и профессор разбивает (A, B, C, D, E) по результатам теста. Покажите, что по крайней мере одна группа будет содержать не менее 20 студентов.
11. Требует ли принцип голубиных ящиков случайного распределения голубей по ящикам?
12. Предположим, что Ева решила найти прообраз по алгоритму 11.1. Какое число раз в среднем Ева должна повторить алгоритм?
13. Предположим, что Ева решила найти коллизию по Алгоритму 11.3. Какое число раз, в среднем, Ева должна повторить алгоритм?
14. Предположим, что мы имеем очень простой дайджест сообщения. Наш дайджест сообщения (нереальный) — только одно число между 0 и 25. Дайджест первоначально установлен на 0. Криптографическая хэш-функция складывает текущее значение дайджеста со значением текущего символа (между 0 и 25). Сложение проводится по модулю 26. Идея показана на рис. 11.14. Каково значение дайджеста, если сообщение — «HELLO»? Почему этот дайджест небезопасен?

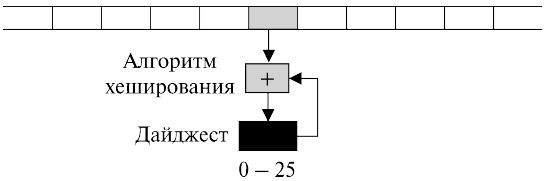


Рис. 11.14 Упражнение 14

15. Попробуем увеличить сложность предыдущего упражнения. Возьмем значение текущего символа, заменим его другим числом и затем сложим с предыдущим значением из дайджеста по модулю 100. Дайджест перво-

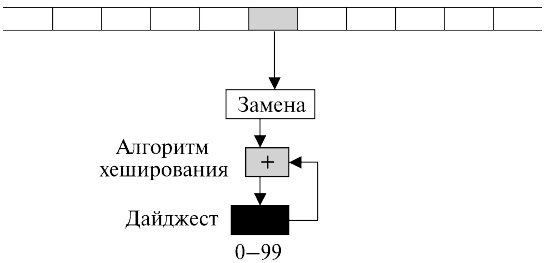


Рис. 11.15 Упражнение 15

начально устанавливается на 0. Рисунок 11.1 показывает идею. Каково значение дайджеста для сообщения «HELLO»? Почему этот дайджест — не безопасный?

16. Используя модульную арифметику, найдите дайджест сообщения. Рисунок 11.16 иллюстрирует процедуру, которая содержит следующие шаги.
    а. Пусть длина дайджеста сообщения равна *n* битам.
    б. Выберите в качестве модуля простое *n*-битовое число p.
    в. Представьте сообщение как двоичное число и дополните сообщение нулями (0), чтобы оно было кратно *m* битам.
    г. Разбейте дополненное сообщение на *N* блоков, каждый по m бит. Обозначим каждый *i*-тый блок X_i.
    д. Выберите начальный дайджест N битов, H_0.
    е. Повторите N раз следующие действия:
    $$H_i = (H_{i-1} + X_i)^2 \bmod p$$
    ж. Дайджест будет равен H_N.
    Какое значение будет иметь дайджест, если сообщение — «HELLO»? Почему этот дайджест не безопасен?

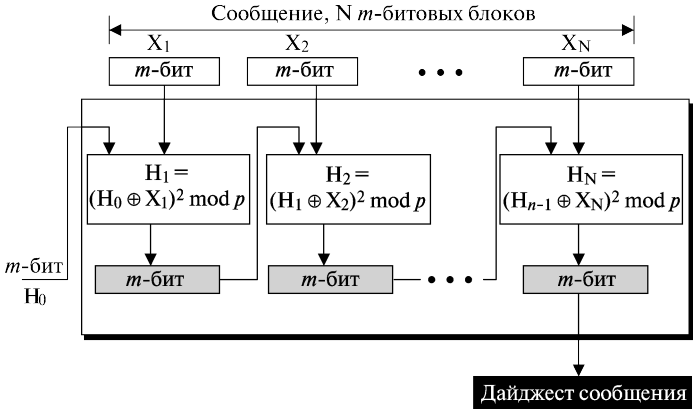


Рис. 11.16. Упражнение 16

17. Ниже описывается хэш-функция, называемая модульной арифметикой безопасного хэширования (Modular Arithmetic Secure Hash — MACH). Напишите алгоритм для вычисления дайджеста заданного сообщения. Найдите дайджест собственного сообщения.
    а. Пусть длина дайджеста сообщения равна *N* бит.
    б. Выберите два простых числа, *p* и *q*. Вычислите $M = pq$.
    в. Представьте сообщение как двоичное число и дополните сообщение нулями, так чтобы сделать его число битов кратным *N/2*. N выбран как число, кратное 16, меньшее, чем число битов в M.
    г. Разделите дополненное сообщение на *m* блоков, каждый по N/2 битов. Обозначим каждый блок X_i.

д. Прибавьте длину сообщения по модулю N/2 как двоичное число к сообщению. Это создаст сообщение длиной m+1 блоков по N/2 битов.

е. Расширьте сообщение, чтобы получить m + 1 блок, каждый по N битов, как показано ниже. Разделите блоки X_1 до X_m на группы по 4 бита. Вставьте 1111 перед каждой группой. Разделите блок X_{m+1} на группы по 4 бита. Вставьте 1010 перед каждой группой. Назовем расширенные блоки $Y_1, Y_2..., Y_{m+1.}$

ж. Выберите начальный дайджест N битов, H_0.

з. Выберите константу K из N битов.

и. Повторите m +1 раз следующие действия (T_i и G_I — промежуточные значения). Символ ‖ обозначает конкатенацию.

$$T_i = ((H_{i+1}, + Y_i,) \| K)^{257} \bmod M. \qquad G_i = H_i \bmod 2^N \qquad H_i = H_{i+1} + G_i;$$

к. Дайджест равен H_{m+1} .

18. Напишите алгоритм в псевдокоде для решения первой проблемы дня рождения (в общей форме).
19. Напишите алгоритм в псевдокоде для решения второй проблемы дня рождения (в общей форме).
20. Напишите алгоритм в псевдокоде для решения третьей проблемы дня рождения (в общей форме).
21. Напишите алгоритм в псевдокоде для решения четвертой проблемы дня рождения (в общей форме).
22. Напишите алгоритм в псевдокоде для HMAC.
23. Напишите алгоритм в псевдокоде для CMAC.

Лекция 12. Криптографические хэш-функции

Цели и содержание

Эта лекция имеет несколько целей.

- Изложить общие идеи криптографических хэш-функций.
- Обсудить схему Меркеля-Дамгарда (Merkle-Damgard) как основу для итеративных хэш-функций.
- Показать различие между двумя категориями хэш-функций: функцией сжатия, которая разработана «на пустом месте», и той, которая использует блочный шифр как функцию сжатия.
- Обсудить структуру SHA-512 как пример криптографической хэш-функции с функцией сжатия, которая делает это «на пустом месте».
- Обсудить структуру Whirlpool как пример криптографической хэш-функции с блочным шифром с функцией сжатия.

12.1. Введение

Рассмотренная в лекции 11 криптографическая хэш-функция получает сообщение произвольной длины и создает дайджест сообщения фиксированной длины. Окончательная цель этой лекции состоит в том, чтобы обсудить детали двух наиболее перспективных криптографических алгоритмов хэширования — *SHA-512* и *Whirlpool*. Однако мы сначала должны обсудить некоторые общие закономерности, которые справедливы в отношении к любой криптографической хэш-функции.

Итеративная хэш-функция

Все криптографические хэш-функции должны создавать дайджест фиксированного размера из сообщения переменного размера. Применять такую функцию лучше всего, используя итерацию. Вместо хэш-функции с вводом переменного размера создана и используется необходимое количество раз функция с вводом фиксированного размера, называемая **функцией сжатия**. Она сжимает n-битовую строку и создает m-битовую строку, где n обычно больше, чем m. Эта схема известна как **итеративная криптографическая функция.**

Схема Меркеля-Дамгарда (Merkle-Damgard)

Схема Меркеля-Дамгарда — итеративная хэш-функция, которая является функцией сжатия, устойчивой к коллизии. Это может быть доказано, но доказательство оставляем как упражнение. Схема показана на рис. 12.1.

Схема использует следующие шаги.

1. Длина сообщения и дополнение добавляются в конец сообщения, чтобы создать увеличенное сообщение, которое может быть равномерно разделено на n-битовые блоки; здесь n – размер блока, который будет обработан функцией сжатия.

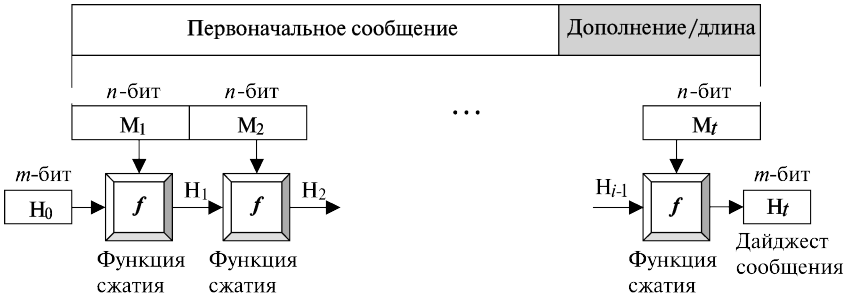


Рис. 12.1. Схема Меркеля-Дамгарда

2. Сообщение тогда рассматривают как *t* блоков, размер каждого состоит из *n* бит. Мы обозначим каждый блок M_1,..., M_t. Мы обозначаем дайджест, созданный при *t* итерациях, — H_1, H_2,...., H_t.
3. Перед стартом итерации дайджест H_0 устанавливается на фиксированное значение, обычно называемое IV (начальное значение или начальный вектор).
4. Функция сжатия при каждой итерации обрабатывает H_{i-1} и M., создавая новый H*i*. Другими словами, мы имеем $H_i = f(H_{i-1}, M_i)$, где *f* — функция сжатия.
5. H_t — функция криптографического хэширования первоначального сообщения, то есть h(M).

Если функция сжатия в схеме Меркеля-Дамгарда устойчива к коллизии, хэш-функция также устойчива к коллизии.

Две группы функций сжатия

Схема Меркеля-Дамгарда сегодня — основан для многих функций криптографического хэширования. Единственная вещь, которую мы должны сделать, – разработать функцию сжатия, которая является устойчивой к коллизиям, и вставить ее в схему Меркеля-Дамгарда. Есть тенденция использовать два различных подхода в разработке хэш-функции. В первом подходе функция сжатия сделана «на пустом месте»: она разработана только для этой цели. Во втором подходе блочный шифр с симметричными ключами служит функцией сжатия.

Хэш-функции, сделанные «на пустом месте»

Множество функций криптографического хэширования использует функции сжатия, которые сделаны «на пустом месте». Эти функции сжатия специально созданы для целей, которым они служат.

Дайджест сообщения (MD) Несколько алгоритмов хэширования были разработаны Роном Ривестом. Они известны в литературе как **MD2**, **MD4** и **MD5,** где MD обозначает Дайджест Сообщения. Последняя версия, MD5, является усилен-

ной версией MD4, которая делит сообщение на блоки по 512 битов и создает дайджест на 128 битов. Оказалось, что дайджест сообщения размером 128 битов — слишком маленький, чтобы быть устойчивым к атаке коллизии.

Алгоритм безопасного хэширования (SHA — Secure Hash Algorithm) Алгоритм безопасного хэширования (SHA) — стандарт, который был разработан национальным Институтом Стандартов и Технологии (NIST — National Institute of Standards and Technology) и издан как Федеральный Стандарт Обработки Информации (FIP 180). Он упоминается в литературе как **Стандарт Безопасного хэширования (SHS — Secure Hash Standard)**. Стандарт главным образом базируется на MD5. В 1995 г. он был пересмотрен под названием FIP 180-1, который включает **SHA-1**. Позже он снова был пересмотрен под названием FIP 180-2, который определяет четыре новых версии: **SHA-224**, **SHA-256**, **SHA-384** и **SHA-512**. Таблица 12.1 дает список некоторых из характеристик этих версий.

Таблица 12.1. Характеристики алгоритмов безопасного хэширования (SHAs)

Характеристики	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Максимальный размер сообщения	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$	$2^{128} - 1$	$2^{128} - 1$
Размер блока	512	512	512	1024	1024
Размер дайджеста сообщения	160	224	256	384	512
Число раундов	80	64	64	80	80
Размер слова	32	32	32	64	64

Все эти версии имеют одну и ту же структуру. SHA-512 будет рассмотрен подробно позже в этой лекции.

Другие Алгоритмы сохранения целостности (RIPMD - RACE[1] Integrity Primitives Evaluation Message Digest). Группа алгоритмов криптографического хэширования **(RIPMD)** имеет несколько версий. **RIPEMD-160** — алгоритм хэширования с дайджестом сообщения на 160 битов. RIPEMD-160 использует ту же структуру, что и MD5, но применяет два варианта выполнения. **HAVAL** — алгоритм хэширования переменной длины с дайджестом сообщения размера 128, 160, 192, 224 и 256. Размер блока — 1024 бита.

Хэш-функции, основанные на блочных шифрах

Итеративная функция криптографического хэширования может использовать блочный шифр с симметричными ключами как функцию сжатия. Основная идея такого подхода: есть несколько безопасных блочных шифров с симметричными ключами, таких, как трехкратный DES или AES, которые могут применяться для создания односторонней функции вместо того, чтобы создавать новую функцию сжатия. Блочный шифр в этом случае только выполняет шифрования. Были предложены несколько схем. Позже мы рассмотрим одну из наиболее перспективных — Whirlpool.

[1] RACE –Research and development for Advanced Communication in Europe — Европейская программа исследований и разработок в области передовых технологий связи (исследовательская программа Европейского союза)

Схема Рабина. Итеративная хэш-функция, предложенная Рабиным, очень проста. **Схема Рабина** базируется на схеме Меркеля-Дамгарда. Функция сжатия заменяется любым алгоритмом шифрования. Блок сообщения используется как ключ; предварительно созданный дайджест используется как исходный текст. Зашифрованный текст — новый дайджест сообщения. Обратите внимание, что размер дайджеста — это размер блочного шифра данных в основной криптографической системе. Например, если DES используется как блочный шифр, размер дайджеста — только 64 бита. Хотя схема очень проста, она может быть раскрыта с помощью атаки «сведения к середине», рассмотренной в лекции 6, поскольку противник может применить алгоритм дешифрования криптографической системы. Рисунок 12.2 показывает схему Рабина.

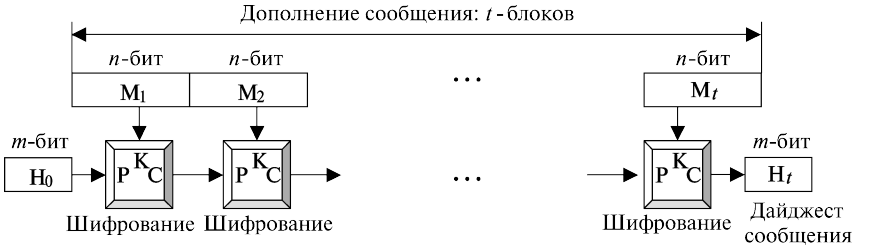


Рис. 12.2. Схема Рабина

Схема Девиса-Мейера (Davies-Mayer). В основном она повторяет схему Рабина, за исключением того, что применяет прямую связь для защиты от атаки «сведения в середину».

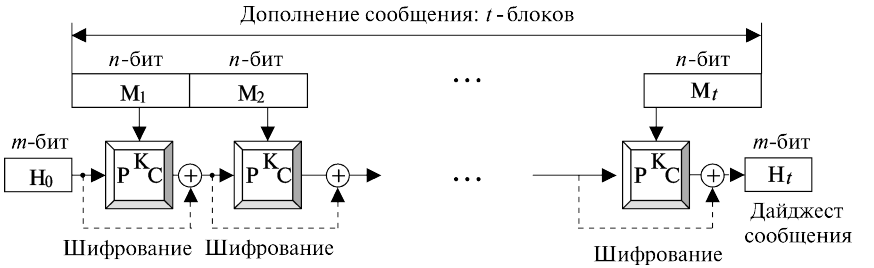


Рис. 12.3. Схема Девиса-Мейера

Схема Матиса-Мейера-Осеаса (Metyas-Mayer-Oseas). Это версия схемы Девиса-Мейера: блоки сообщения применяются как ключи криптосистемы. Схема может быть использована, если блоки данных и ключ шифрования имеют один и тот же размер. Например, AES хорошо подходит для этой цели.

Схема Миагучи-Пренеля — расширенная версия схемы Матиса-Мейера-Осеаса. Чтобы сделать алгоритм более устойчивым к атаке, исходный текст, ключ шифра и зашифрованный текст складываются с помощью ИСКЛЮЧАЮЩЕГО

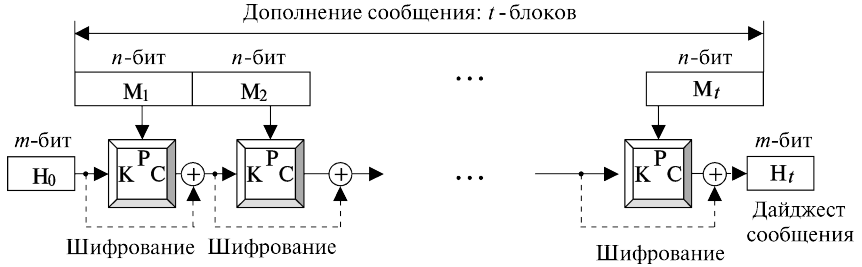


Рис. 12.4. Схема Матиса-Мейера-Осеаса

ИЛИ и создают новый дайджест. Эта схема используется в Whirlpool для создания хэш-функции. На рисунке 12.5 показана схема Миагучи-Пренеля.

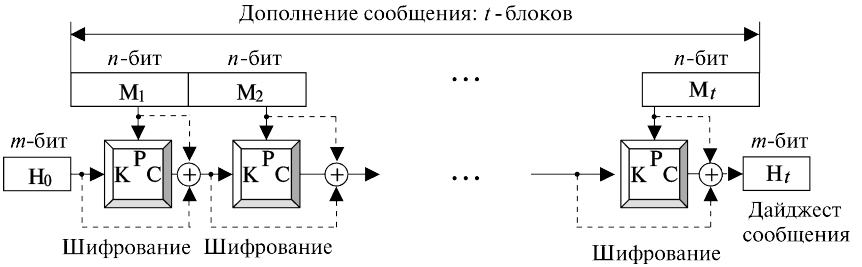


Рис. 12.5. Схема Миагучи-Пренеля

12.2. SHA-512

Версия SHA (Secure Hash Algorithm) — алгоритм безопасного хэширования с 512-битовым дайджестом сообщения. Эта версия похожа на другие алгоритмы этого семейства, которые основаны на схеме Меркеля–Дамгарда. Мы выбрали для рассмотрения особую версию. Она самая поздняя, обладает более полной структурой, чем другие, и наиболее длинным дайджестом сообщения. Если понять эту версию, нетрудно будет усвоить структуру других версий.

Введение

SHA-512 создает дайджест из сообщения, содержащего много блоков. Каждый блок имеет длину 1024 бита, как это показано на рис. 12.6.

Дайджест вначале устанавливается на определенное заранее значение 512 битов. Алгоритм смешивает это начальное значение с первым блоком сообщения, чтобы создать первый промежуточный дайджест сообщения 512 битов. Этот дайджест затем смешивается со вторым блоком, чтобы создать второй промежуточный дайджест. Наконец, $(N - 1)$-ый дайджест смешивается с N-ым блоком — они создают N-ый дайджест. Когда последний блок обработан, результирующий дайджест – это дайджест полного сообщения.

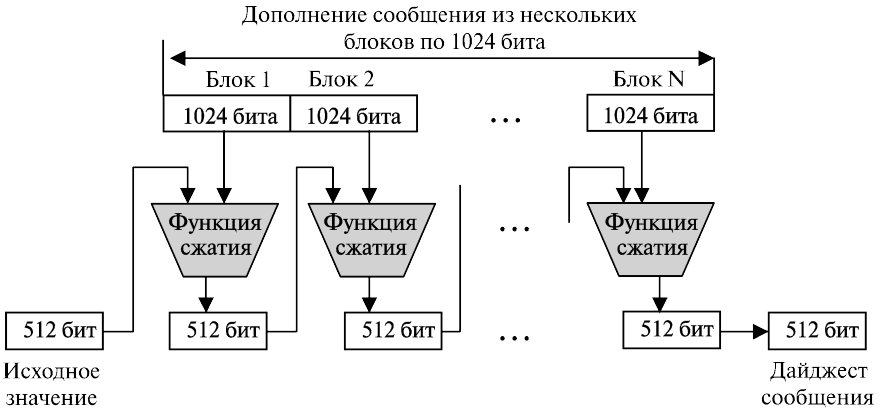


Рис. 12.6. Создание дайджеста сообщения SHA-512

Подготовка сообщения

SHA-512 требует, чтобы длина первоначального сообщения была меньше, чем 2^{128} битов. Если длина сообщения равна или больше, чем 2^{128}, оно не будет обработано SHA-512. Это обычно не проблема, потому что 2^{128} битов превосходят любую возможную сегодня полную емкость хранения любой системы.

SHA-512 создает дайджест сообщения на 512 битов из сообщения меньшего, чем 2^{128}.

Пример 12.1

Этот пример показывает, что ограничение длины сообщения SHA-512 — не серьезная проблема. Предположим, что мы должны передать сообщение длиною 2^{128} бита в секунду. Какое время потребуется для системы коммуникаций со скоростью передачи данных 2^{64} бита в секунду, чтобы передать это сообщение?

Решение

Системы коммуникаций, которая может передать 2^{64} бита в секунду, пока еще не существует. Даже если бы она была, потребовалось бы много лет, чтобы передать это сообщение. Отсюда ясно, что мы не должны волноваться по поводу ограничения длины сообщения для SHA-512.

Пример 12.2

Этот пример также касается длины сообщения в SHA-512. Сколько страниц занимает сообщение 2^{128} бит?

Решение

Предположим, что символ имеет длину 32 или 26 бит. Каждая страница — меньше, чем 2048, или приблизительно 2^{12}, символов. Тогда 2^{128} битов требуют по крайней мере $2^{128}/2^{18}$, или 2^{110} страниц. И снова ясно, что мы не должны волноваться об ограничении на длину сообщения.

Поле длины и заполнение

Прежде чем дайджест сообщения может быть создан, SHA-512 требует сложения поля длины — это целое число без знака на 128 битов, которое определяет длину сообщения в битах, — с сообщением. Это длина первоначального сообщения перед заполнением. Поле целого числа без знака 128 битов можно определить как число между 0 и $2^{128} - 1$, которое является максимальной длиной сообщения, принятого в SHA-512. Поле длины определяет длину первоначального сообщения перед его сложением или заполнением (рис. 12.7).

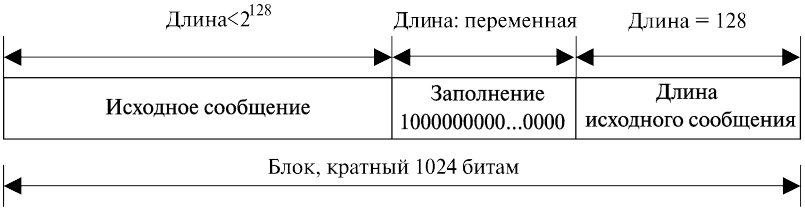


Рис. 12.7. Заполнение и поле длины в SHA-512

Перед сложением поля длины мы должны дополнить первоначальное сообщение, чтобы сделать длину кратной 1024. Для поля длины резервируется 128 битов, как показано на рис. 12.7. Длина области заполнения может быть рассчитана следующим образом. Пусть |M| — длина первоначального сообщения и |P| — длина поля заполнения.

$$(|M| + |P| + 128) = 0 \bmod 1024 \rightarrow |P| = (-|M| - 128) \bmod 1024$$

Формат заполнения – это одна единица (1), сопровождаемая необходимым числом нулей (0).

Пример 12.3

Какое число битов заполнения необходимо, если длина первоначального сообщения — 2590 битов?

Решение

Мы можем вычислить число битов заполнения следующим образом:

$$|P| = (-2590 - 128) \bmod 1024 = -2718 \bmod 1024 = 354$$

Заполнение состоит из одной единицы, сопровождаемой 353 нулями.

Пример 12.4

Нужно ли заполнение, если длина первоначального сообщения уже кратна 1024 битам?

Решение

Да, нужно, потому что мы должны добавить поле длины. Заполнение необходимо, чтобы сделать и новый блок кратным 1024 битам.

Пример 12.5

Каково минимальное и максимальное число битов заполнения, которые можно добавить к сообщению?

Решение

a. Минимальная длина заполнения — 0, и это случается, когда $(-M - 128) \bmod 1024 = 0$; тогда $|M| = -128 \bmod 1024 = 896 \bmod 1024$ бит. Другими словами, последний блок в первоначальном сообщении — 896 битов. Мы добавляем поле длины на 128 битов, чтобы сделать блок полным.

b. Максимальная длина заполнения — 1023, и это случается, когда $(-|M| - 128) = 1023 \bmod 1024$. Это означает, что длина первоначального сообщения — $|M| = (-128 - 1023) \bmod 1024$ или $|M| = 897 \bmod 1024$. В этом случае мы не можем просто добавить область длины, потому что длина последнего блока будет превышать на один бит число 1024. Так что мы нуждаемся в заполнении 127 битами, чтобы закончить этот блок и создать второй блок заполнения 896 битов. Теперь можно добавить поле длины, чтобы сделать этот блок полным.

Слова

SHA-512 оперирует словами; он — **ориентируемый на слово.** Слово определено длиной 64 бита. Это означает, что после того как заполнение и поле длины добавляются к сообщению, каждый блок сообщения состоит из шестнадцати слов по 64 бита. Дайджест сообщения также образуется из слов по 64 бита, но дайджест сообщения — только восемь слов, и слова обозначают A,B, C, D, E, F, G и H, как показано на рис. 12.8.

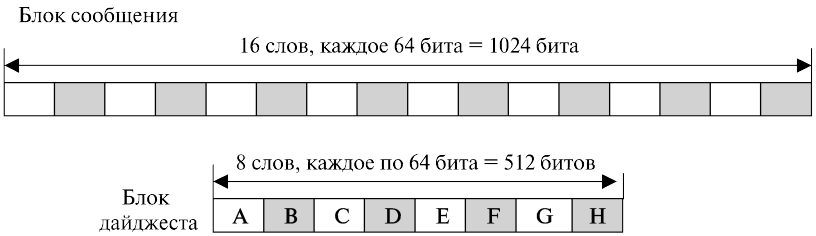
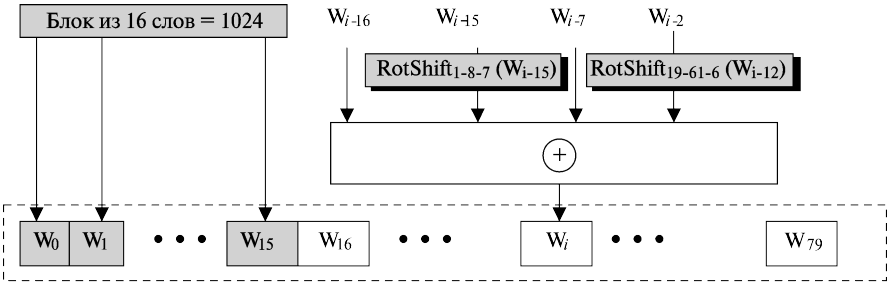


Рис. 12.8. Блок сообщения и дайджест в виде отдельных слов

SHA-512 — алгоритм, ориентированный на слово.
Каждый блок – 16 слов; в дайджесте – только 8 слов.

Расширение слова

Перед обработкой каждый блок сообщения должен быть расширен. Блок образован из 1024 битов, или шестнадцати слов по 64 бита. Как мы увидим позже, в фазе обработки нам нужно 80 слов. Так что блок с 16-ю словами должен быть расширен до 80 слов от W_0 до W_{79} . Рисунок12.9 показывает процесс **расширения слова**. Блок на 1024 бита порождает первые слова; остальная часть слов получается от уже созданных слов согласно операциям, которые показаны на рисунке.



RotShift$_{l\text{-}m\text{-}n}$ (x): $\boldsymbol{RotR_l}(\boldsymbol{x}) \oplus \boldsymbol{RotR_m}(\boldsymbol{x}) \oplus \boldsymbol{ShL_n}(\boldsymbol{x})$

$\boldsymbol{RotR_i}$ (x): Правое циклическое перемещение аргумента x на i бит

$\boldsymbol{ShL_i}$ (x): Левое циклическое перемещение аргумента x на i бит и левое заполнение нулями

Рис.12.9 Расширение слова в SHA-512

Пример 12.6

Показать, как получить W_{60}.

Решение

Каждое слово в диапазоне W_{16} до W_{79} получено в результате обработки четырех слов, созданных предварительно на предыдущих шагах. W_{60} получено как

$$W_{60} = W_{44} \oplus \text{RotShift}_{1\text{-}8\text{-}7}(W_{45}) \oplus W_{53} \oplus \text{RotShift}_{19\text{-}61\text{-}6}(W_{58})$$

Инициализация дайджеста сообщения

Алгоритм использует восемь констант для инициализации дайджеста сообщения. Мы обозначаем их от A_0 до H_0, что соответствует обозначению слов, используемых для дайджеста. Таблица 12.2 показывает значение этих констант.

Таблица 12.2. Значение констант при инициализации дайджеста сообщения SHA-512

Буфер	Значение (шестнадцатеричное)	Буфер	Значение (шестнадцатеричное)
A_0	**6A09E667F3BCC908**	E_0	**510E527FADE682D1**
B_0	**3B67AE8584CAA73B**	F_0	**9B05688C2B3E6C1F**
C_0	**3C6EF372EF94F828**	G_0	**1F83D9ABFB41BD6B**
D_0	**A54FE53A5F1D36F1**	H_0	**5BEOCD19137E2179**

Читатель может задаться вопросом, откуда взяты эти значения. Они рассчитаны из первых восьми простых чисел (2, 3, 5, 7, 11, 13, 17 и 19). Каждое значение —

дробная часть квадратного корня соответствующего простого числа после преобразования к двоичной форме и сохранения только первых 64 битов. Например, восьмое простое число — 19 имеет квадратный корень $(19^{1/2}) = 4{,}35889894354$. Преобразовывая число к двоичной форме только с 64 битами в дробной части, мы имеем

$$(100.0101\ 1011\ 1110...\ 1001)_2 \rightarrow (4{,}5BEOCD19137E2179)_{16}$$

SHA-512 сохраняет дробную часть $(5BEOCD19137E2179)_{16}$ как целое число без знака.

Функция сжатия

SHA-512 создает 512 битов дайджест-сообщения (восемь слов на 64 бита) из сообщения, которое состоит из множества блоков, где каждый блок содержит 1024 бита. Обработка каждого блока данных в SHA-512 включает 80 раундов. Ри-

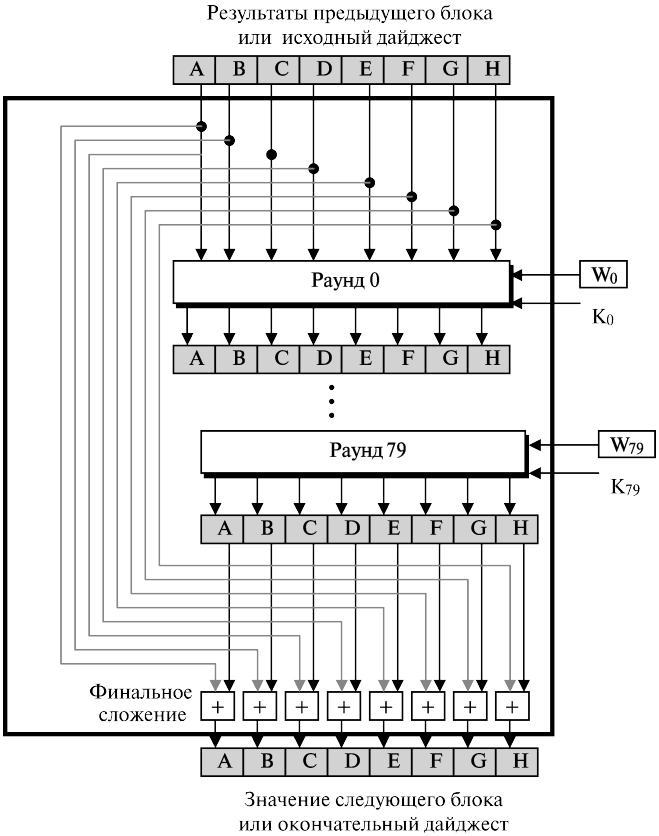


Рис. 12.10. Функция сжатия в SHA-512

сунок 12.10 показывает общую схему сжатия функции. В каждом раунде содержание восьми предыдущих буферов — это одно слово из расширенного блока (W_i), и одна константа на 64 бита (K_i), смешанные вместе. Они обработаны затем, чтобы создать новое множество из восьми буферов. В начале обработки значения восьми буферов сохранены как восемь временных переменных. В конце обработки (после того как сделан шаг 79) эти значения добавляются к значениям, созданным на шаге 79. Мы вызываем эту последнюю операцию *финальным сложением*, как это показано на рисунке 12.10.

Структура каждого раунда

В каждом раунде создаются восемь новых, по сравнению с предыдущим раундом, значений буферов по 64 бита. На рис. 12.11 мы видим, что шесть буферов — точные копии предыдущего раунда, как это показано ниже:

$$A \rightarrow B \qquad B \rightarrow C \qquad C \rightarrow D \qquad E \rightarrow F \qquad F \rightarrow G \qquad G \rightarrow H$$

Два новых буфера, A и E, получают соответствующие значения от некоторых сложных функций, которые включают в себя некоторые значения предыдущих буферов, соответствующее слово для этого раунда (W_i) и константу для этого раунда (K_i). Рисунок 12.11 показывает структуру каждого раунда.

Здесь есть два смесителя, три функции и несколько операторов. Каждый смеситель обрабатывает две функции. Описание функций и операторов приведено ниже.

1. То, что мы называем мажоритарной функцией, является поразрядной функцией. Она использует три соответствующих бита в трех буферах (A, B и C) и вычисляет

$$(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$$

Результат — это значение, которое имеет большинство из трех бит. Если два или три бита равны единице (1) , то результат имеет значение бит 1; иначе он равен 0.

Функция, которую мы называем условной функцией (Conditional) — также поразрядная функция. Она использует три бита, которые содержатся в трех буферах (E, F и G), и вычисляет

$$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$$

Результат подчиняется логике «Если E, то F; иначе G».

3. Функция «циклическое перемещение» (Rotate) обрабатывает три значения одного и того же буфера (A или E) и применяет операцию ИСКЛЮЧАЮЩЕЕ ИЛИ с результатом мажоритарной функции.

$$\text{Rotate (A): } RotR_{28}(A) \oplus RotR_{34}(A) \oplus RotR_{29}(A)$$
$$\text{Rotate (E): } RotR_{28}(A) \oplus RotR_{34}(E) \oplus RotR_{29}(E)$$

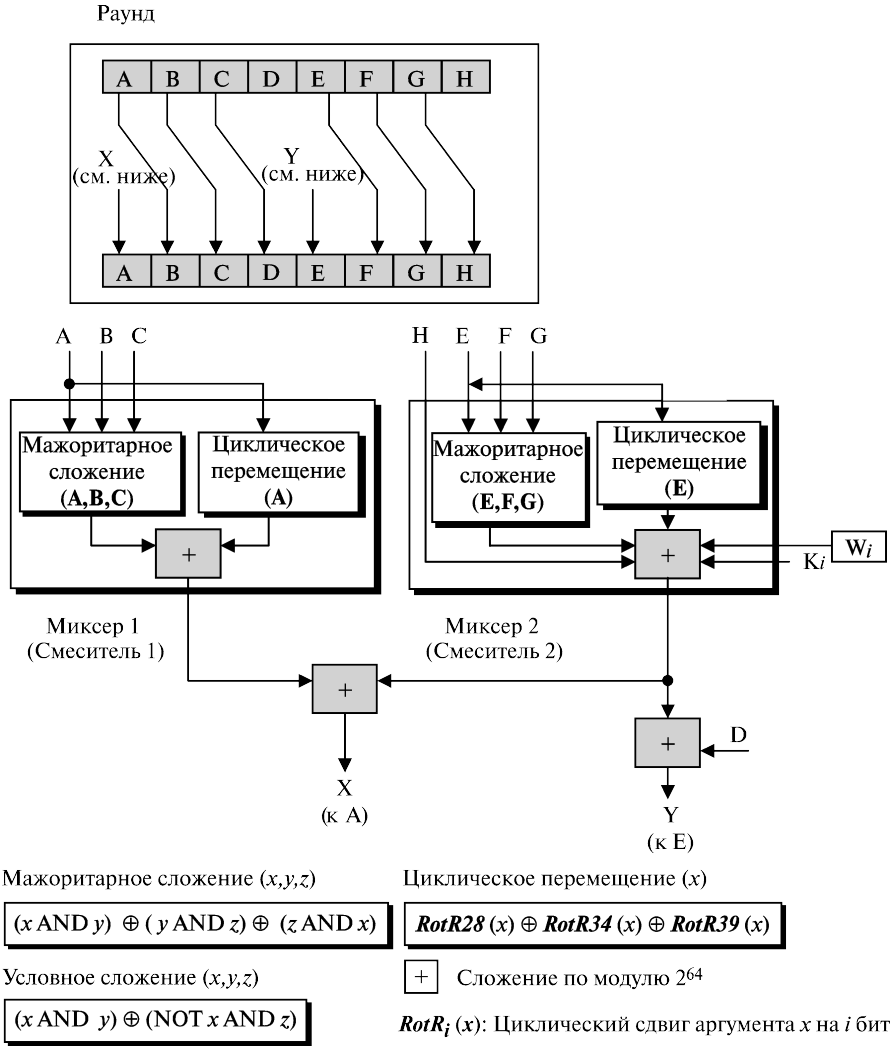


Рис. 12.11. Структура каждого раунда SHA-512

4. Функция «циклическое перемещение вправо» ($RotR_i$ (x)) — та же самая, которую мы использовали в процессе расширения слова.
5. Оператор сложения, применяемый в процессе, – сложение по модулю 2^{64}. Он означает результат сложения двух или больше буферов, содержащих всегда слово на 64 бита.
6. Есть 80 констант, K_0 к K_{79}, каждая по 64 бита, как показано в таблице 12.3 в шестнадцатеричной форме (четыре в каждой строке таблицы). Анало-

гично начальным значениям для восьми буферов, эти значения вычислены из первых 80 простых чисел (2, 3..., 409).

428A2F98D728AE22	7137449123EF65CD	B5COFBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	OFC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CBOA9DCBD41FBD4	76F98DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6EOOBF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670AOE6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766AOABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70DOF89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2DOC8	1E376C085141AB53	2748774CDF8EEB99	34BOBCB5E19B48A8
391COCB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1FOAB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721COC207	EADA7DD6CDEOEB1E	F57D4F7FEE6E178
06F067AA72176FBA	OA637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBEOA15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

Таблица 12.3. Восемьдесят констант, используемых для восьмидесяти раундов в SHA-512

Каждое значение — дробная часть кубического корня из соответствующего простого числа после преобразования этого числа к двоичной форме, сохраняются только первые 64 бита. Например, 80-е простое число – (409). Кубический корень (409) $^{1/3}$ = 7,42291412044. Преобразовывая это число к двоичному виду только с 64 битами в дробной части, мы получаем

$$(111{,}0110\ 1100\ 0100\ 0100...0111)_2 \rightarrow (7{,}6C44198C4A475817).$$

SHA-512 сохраняет дробную часть, $(6C44198C4A475817)_{16}$, как целое число без знака.

Пример 12.7

Мы применяем мажоритарную функцию к значениям буферов A, B и C. Если крайние левые шестнадцатеричные цифры этих буферов — 0x7, 0xA и 0xE соответственно, то какая цифра будет крайней левой частью результата?

Решение

Цифры в двоичной форме — 0111, 1010 и 1110.

a. Первые биты — 0, 1 и 1. Мажоритарная функция равна 1. Мы можем доказать это, используя определение мажоритарной функции:

$$(0 \text{ AND } 1) \oplus (1 \text{ AND } 1) \oplus C\ (1 \text{ AND } 0) = 0 \oplus 1 \oplus 0 = 1$$

b. Вторые биты — 1, 0 и 1. Мажоритарная функция равна 1 .
c. Третьи биты — 1,1 и 1. Мажоритарная функция равна 1.
d. Четвертые биты — 1, 0 и 0. Мажоритарная функция равна 0. Результат — 1110, или 0xE в шестнадцатеричной форме.

Пример 12.8

Мы применяем условную функцию (Conditional) для буферов E, F и G. Если крайние левые шестнадцатеричные цифры этих буферов — 0x9, 0xE и 0xF соответственно, то какая цифра будет крайней левой частью результата?

Решение

Цифры в двоичной форме — 1001, 1010 и 1111.

a. Первые биты — 1,1 и 1. Следовательно, $E_1 = 1$, результат – F_1, который равен 1. Чтобы доказать результат, мы можем также использовать определение функции УСЛОВИЕ (Condition):

$$(1 \text{ AND } 1) \oplus (\text{NOT } 1 \text{ AND } 1) = 1 \oplus 0 = 1$$

b. Вторые биты — 0, 0 и 1. Следовательно, E_2 — 0, результат — F_3, который равен 1.
c. Третьи биты – 0, 1 и 1. Следовательно, E_3 — 0, результат — G3, который равен 1.
d. Четвертые биты — 1, 0 и 1. Следовательно, E_4 — 1, результат — F_4, который равен 0. Результат — 1110, или 0xE в шестнадцатеричной форме.

Анализ

С дайджестом сообщения 512 битов от SHA-512 ожидалось, что он будет более стойким ко всем типам атак, включая атаки коллизии. Он должен был быть лучшим проектом этой версии: более эффективным и более безопасным, чем предыдущие. Однако необходимы были серьезные исследования и испытания, для того чтобы это подтвердить.

12.3. Whirlpool

Whirlpool разработан Винсентом Риджменом (Vincent Rijmen) и Пауло Баретто (Paolo Barreto). Он одобрен европейской организацией **NESSIE** (**New European Schemes for Signature, Integrity and Encryption — Новые европейские схемы подписей, целостности и шифрования**).

Whirlpool — итеративная функция криптографического хэширования, основанная на схеме Миагучи-Пренеля, которая использует блочный шифр с симметричными ключами вместо функции сжатия. Блочный шифр в данном случае является измененным шифром AES, который был приспособлен для этой цели. Рисунок 12.12 показывает хэш-функции Whirlpool.

Подготовка

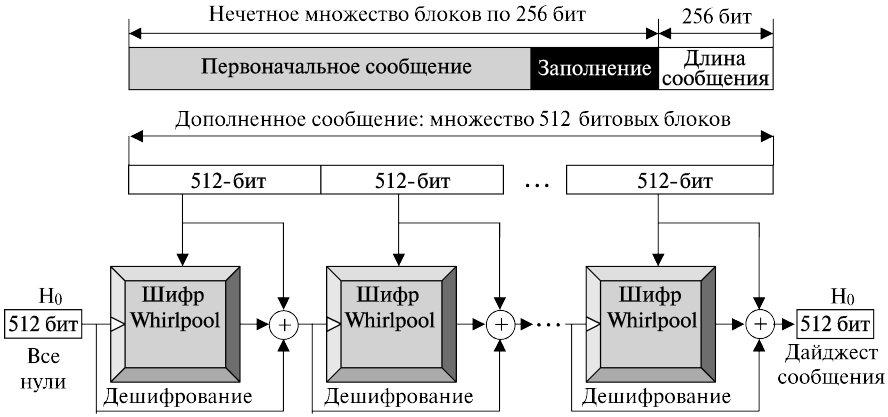


Рис. 12.12. Whirlpool хэш-функция

Перед стартом алгоритма хэширования сообщение должно быть готово к процессу. Whirlpool требует, чтобы длина первоначального сообщения была меньше, чем 2^{256} бита. Сообщение должно быть дополнено прежде, чем начать обработку. Дополнение содержит единственный бит, равный 1, а затем следует необходимое чисел нулевых битов, чтобы сделать длину дополнения нечетным числом, кратным 256 битам. После дополнения добавляется блок 256 битов, чтобы указать длину первоначального сообщения. Этот блок обрабатывается как число без знака.

После дополнения первоначального сообщения и присоединения поля длины увеличенный размер сообщения становится кратным 256 битам или кратным 512 битам. Whirlpool создает дайджест 512 из сообщения, состоящего из многих блоков по 512 бит. Дайджест из 512 бит, H_0, начинается всеми нулями. Это становится ключом шифра для шифрования первого блока. Из зашифрованного текста каждого зашифрованного блока получают ключ шифра для следующего блока после того, как его складывают по модулю два с предыдущим ключом шифра и блоком исходного текста. Дайджест сообщения — конечный зашифрованный текст на 512 битов после последней операции ИСКЛЮЧАЮЩЕЕ ИЛИ.

Шифр Whirlpool

Шифр Whirlpool — шифр не-Файстеля, похож на AES и был, главным образом, разработан как блочный шифр, который используется в алгоритме хэширования. Вместо того чтобы дать полное описание этого шифра, мы будем исходить из того, что читатель знаком с AES по материалам лекции 7. Ниже только приводится сравнение шифра Whirlpool с шифром AES и отмечается их отличие.

Раунды

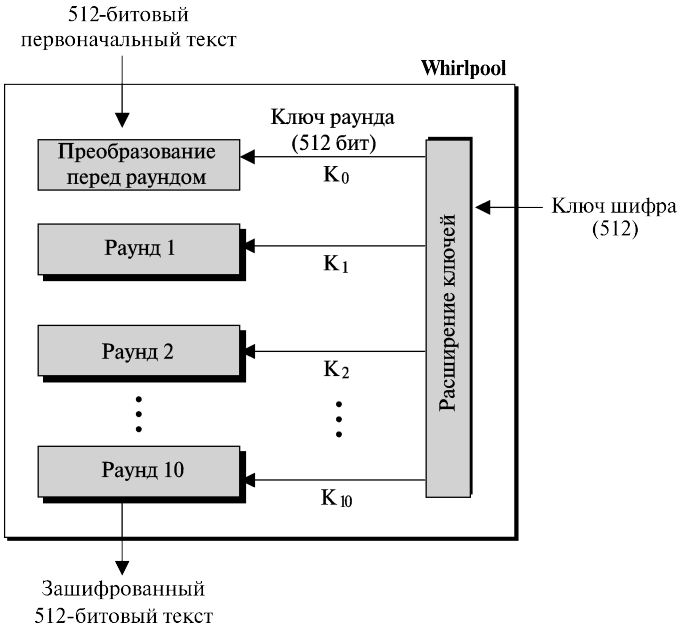


Рис. 12.13. Общая идея шифра Whirlpool

Whirlpool – шифр, который использует 10 раундов. Размер блока и ключевой размер — 512 битов. Шифр применяет 11 ключей раунда $K_0 - K_{10}$, каждый по 512 битов. Рисунок 12.13 показывает общий вид процесса шифрования шифром Whirlpool.

Матрицы состояний и блоки

Подобно шифру AES, шифр Whirlpool использует матрицы состояний и блоки. Однако размер блока или матрицы состояний — 512 битов. Блок рассматривается как строка матрицы длиной 64 байта; матрица состояний — как квадратная матрица 8x8 байтов. В отличие от AES преобразование «блок — матрица состояний» или «матрица состояний — блок» происходят строка за строкой. Рисунок 12.14 показывает блок, матрицу состояний и преобразование в шифр Whirlpool.

Структура каждого раунда

Рисунок 12.15 показывает структуру каждого раунда. Каждый раунд использует четыре преобразования.

SubBytes. Подобно AES, SubBytes обеспечивает нелинейное преобразование. Байт представлен как две шестнадцатеричных цифры. Левая цифра определяет строку, а правая — столбец таблицы подстановки. Две шестнадцатеричных цифры в пересечении строки и столбца — новый байт. Рисунок 12.16 иллюстрирует идею.

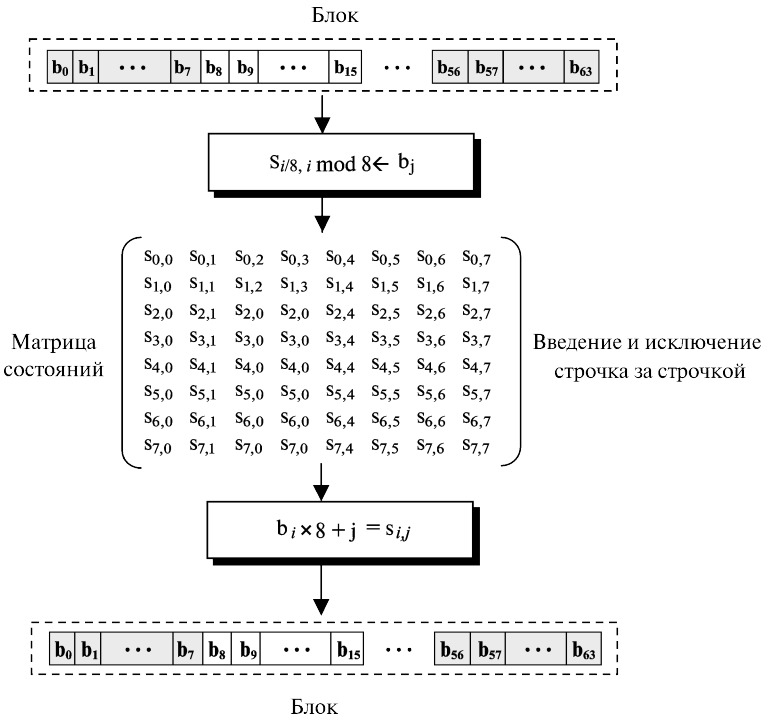


Рис. 12.14. Блок и матрица состояний шифра Whirlpool

В преобразовании SubBytes матрица состояний обрабатывается как матрица байтов 8 × 8. Преобразование делается одновременно только с одним байтом. Содержание каждого байта изменяется, но порядок следования байтов в матрице остается тем же самым. В процессе каждый байт преобразуется независимо; мы имеем 64 различных преобразований байт-к-байту.

Таблица 12.4 показывает таблицу подстановки (S-блок) для преобразования подбайтов. Преобразование обеспечивает эффект перемешивания. Например, два байта, $5A_{16}$ и $5B_{16}$, которые отличаются только одним битом (самый правый бит), преобразованы к $5B_{16}$ и 88_{16}, которые отличаются пятью битами.

Входы в таблице 12.4 могут быть вычислены алгебраически, используя поле $G(2^4)$ с неприводимым полиномом $(x^4 + x + 1)$, как показано на рис. 12.17. Каждая шестнадцатеричная цифра в байте вводится в миниблок (E и E^{-1}). Результаты передаются в другой миниблок R. E-блоки вычисляют степень, равную шестнадцатеричному значению входа; R-миниблок использует псевдослучайный генератор чисел.

$$E(\text{вход}) = (x^3 + x + 1)^{\text{вход}} \bmod (x^4 + x + 1), \text{ если вход} \neq 0 \times F$$
$$E(0 \times F) = 0$$

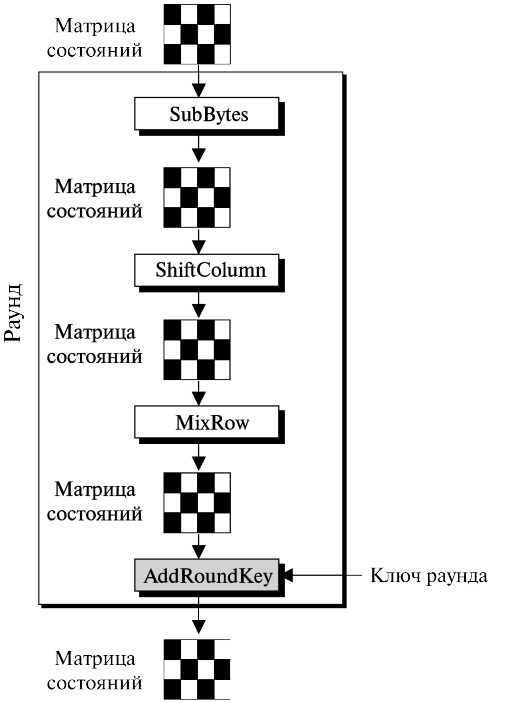


Рис. 12.15. Структура каждого раунда шифра Whirlpool

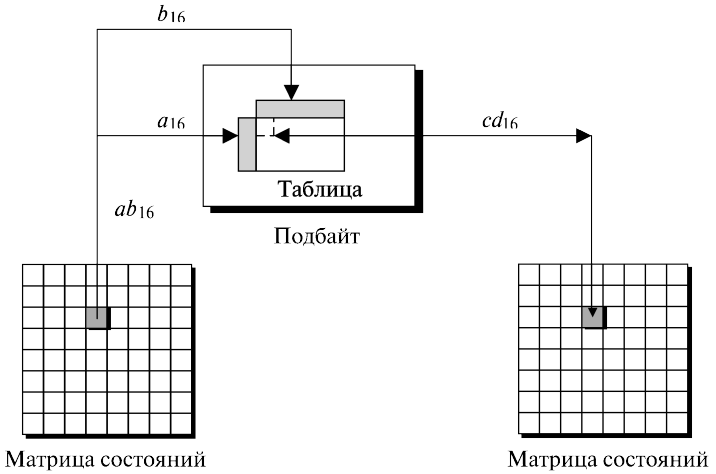


Рис. 12.16. Преобразование *SubBytes* шифра Whirlpool

Таблица 12.4. Таблица преобразования SubBytes (S-Box)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	18	23	C6	E8	87	B8	01	4F	36	A6	D2	F5	79	6F	91	52
1	16	BC	9B	8E	A3	0C	7B	35	1D	E0	D7	C2	2E	4B	FE	57
2	15	77	37	E5	9F	F0	4A	CA	58	C9	29	0A	B1	A0	6B	85
3	BD	5D	10	14	CB	3E	05	67	E4	27	41	8B	A7	7D	95	C8
4	FB	EF	7C	66	DD	17	47	9E	CA	2D	BF	07	AD	5A	83	33
5	63	02	AA	71	C8	19	49	C9	F2	E3	5B	88	9A	26	32	BO
6	E9	OF	D5	80	BE	CD	34	48	FF	7A	90	5F	20	68	1A	AE
7	B4	54	93	2'2	64	F1	73	12	40	08	C3	EC	DB	A1	8D	3D
8	97	00	CF	2B	76	82	D6	1B	B5	AF	6A	50	45	F3	30	EF
9	3F	55	A2	EA	65	BA	2F	CO	DE	1C	FD	4D	92	75	06	8A
A	B2	E6	OE	F	62	D4	A8	96	F9	C5	25	59	84	72	39	4C
B	5E	7S	38	8C	C1	A5	E2	61	B3	21	9C	1E	43	C7	FC	04
C	51	99	6D	0D	FA	DF	7E	24	3B	AB	CE	11	8F	4E	B7	EB
D	3C	S1	94	F7	9B	13	2C	D3	E7	6E	C4	03	56	44	7E	A9
E	2A	BB	C1	53	DC	OB	9D	6C	31	74	F6	46	AC	89	14	E1
F	16	3A	69	09	70	B6	CO	ED	CC	42	98	A4	28	5C	F8	86

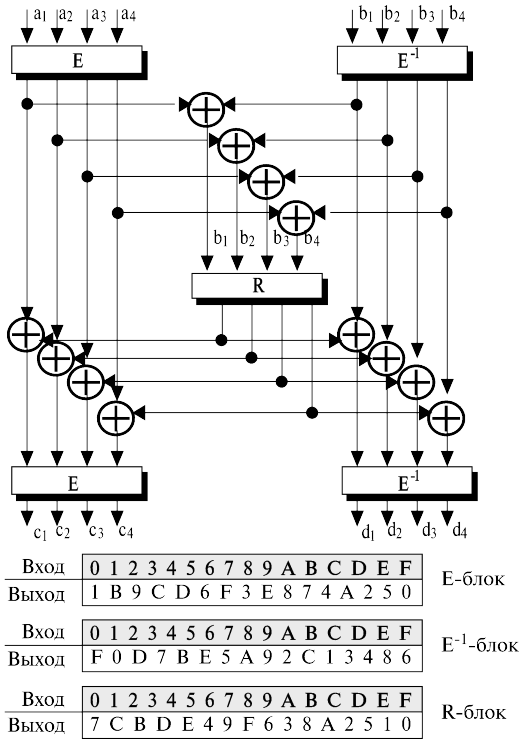
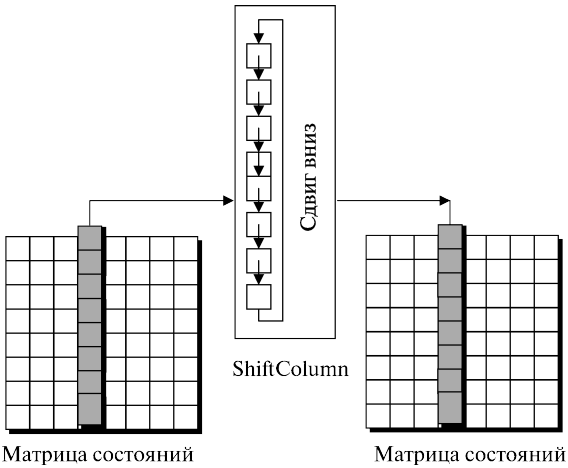


Рис. 12.17. Операция *SubBytes* шифра Whirlpool

E^{-1}-блок — это только инверсия E-блока, где роли входов и выходов изменились. Значения входа-выхода для блоков сведены в таблицу на рис. 12.17.



ShiftColumns. Чтобы обеспечить перестановку, Whirlpool использует преобразование **ShiftColumns**, которое является подобным преобразованию *ShiftRows* в AES, за исключением того, что вместо строк сдвигаются столбцы. Смещение зависит от позиции столбца. Столбец 0 сдвигается на 0 байтов (смещения нет), в то время как столбец 7 сдвигается на 7 байтов. Рисунок 12.18 показывает преобразование смещения.

Рис. 12.18. Преобразование *ShiftColumns* шифра Whirlpool

MixRows. Преобразование **MixRows** имеет тот же эффект, что и преобразование MixColumns в AES: оно рассеивает биты. Преобразование MixRows — матричное преобразование, где байты интерпретируются как слова по 8 битов (или полиномы) с коэффициентами в $GF(2^8)$. Умножение байтов проводится в $GF(2^8)$, но модуль отличается от используемого в AES. Шифр Whirlpool применяет (0x11 D) или $(x^8 + x^4 + x^3 + x^2 + 1)$ как модуль. Сложение слов по 8 битов — то же самое, что ИСКЛЮЧАЮЩЕЕ ИЛИ. На рис. 12.19 представлено преобразование MixRows.

Рисунок показывает умножение единственной строки на матрицу констант; умножение можно провести, умножая всю матрицу состояний на матрицу констант. Обратите внимание, что в матрице констант каждая строка получена с помощью циркулярного сдвига вправо предыдущей строки.

AddRoundKey. Преобразование **AddRoundKey** в шифре Whirlpool делается байт за байтом, потому что каждый ключ раунда — матрица состояний 8 × 8 байт. Рисунок 12.20 показывает этот процесс. Байт матрицы состояний данных складывается в поле $GF(2^8)$ с соответствующим байтом матрицы состояний ключей раунда. Результат — новый байт в новой матрице состояний.

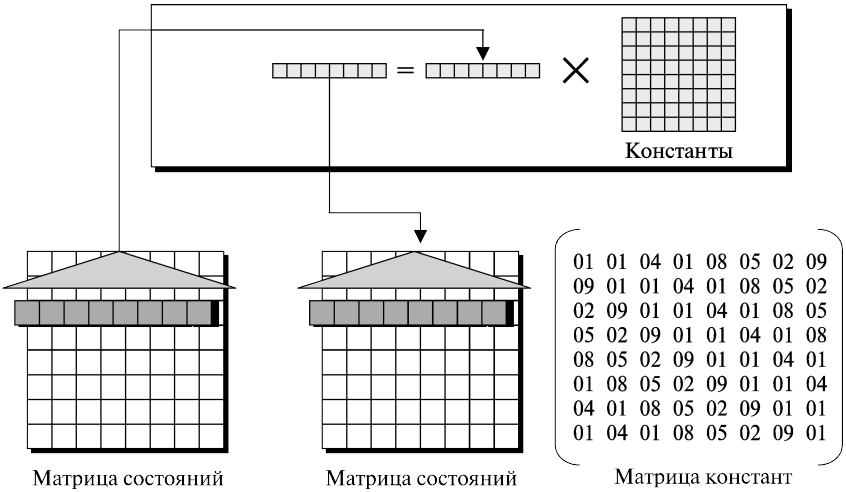


Рис. 12.19. Преобразование *MixRows* шифра Whirlpool

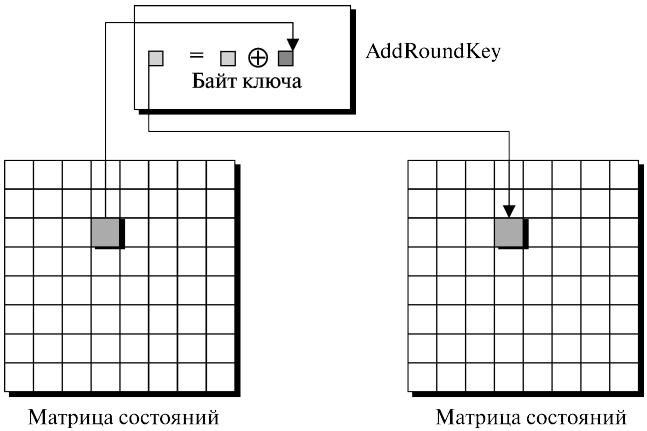


Рис. 12.20. Преобразование *AddRoundKey* шифра Whirlpool

Расширение ключа

Как показывает рис. 12.21, алгоритм расширения ключей в Whirlpool полностью отличается от алгоритма в AES. Вместо того чтобы применять новый алгоритм создания ключей раунда, Whirlpool использует копию алгоритма шифрования (без предраунда), чтобы создать ключи раунда. Выход каждого раунда в алгоритме шифрования есть ключи для этого раунда. На первый взгляд это

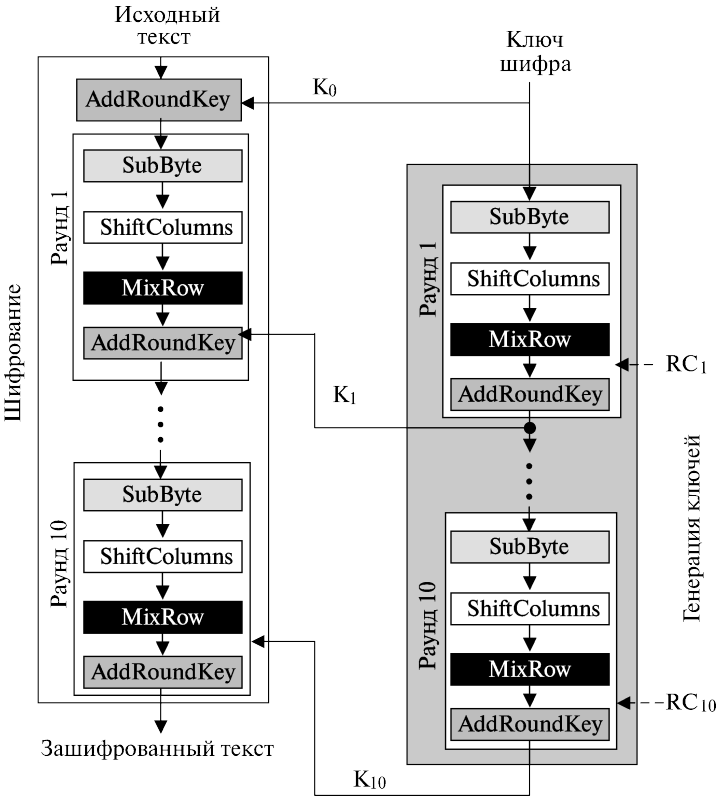


Рис. 12.21. Расширение ключа шифра Whirlpool

напоминает определение, где ключи раунда для алгоритма расширения ключа получаются из него самого. Откуда получается алгоритм расширения? Whirlpool изящно решил эту проблему, используя десять констант раунда (RC) как виртуальные ключи раунда для алгоритма расширения ключей. Другими словами, алгоритм расширения ключей применяет константы как ключи раунда. Алгоритм шифрования использует выход каждого раунда алгоритма расширения ключей как ключи раунда. Алгоритм генерирования ключей обрабатывает ключ шифра как *исходный текст* и зашифровывает его. Обратите внимание, что ключ шифра — также K_0 для алгоритма шифрования.

Константы раунда. Каждая константа раунда RC_r является матрицей 8 × 8, где только первая строка имеет значения, отличные от нуля. Остальная часть входов содержит все нули. Значения для первой строки в каждой матрице констант могут быть вычислены, с применением преобразования SubBytes (таблица 12.4).

$$RC_{round}[\text{строка, столбец}] = \text{Subbytes}\,(8\,(\text{round} - 1) + \text{столбец}) \quad \text{если строка} = 0$$

$$RC_3 = \begin{pmatrix} 1D & E0 & D7 & C2 & 2E & 4B & FE & 57 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{pmatrix}$$

Рис. 12.22. Константы для третьего раунда

RC_{round} [строка, столбец] = 0 если строка ≠ 0

Другими словами, RC_1 использует первые восемь входов в таблице преобразования SubBytes (Таблица 12.4); RC_2 использует вторые восемь входов, и т. д. Рисунок 12.22 показывает пример RC3, где первая строка — третьи восемь входов в таблице SubBytes.

Итоги шифра Whirlpool

В таблице 12.5 приведены основные характеристики шифра Whirlpool.

Таблица 12.5. Основные характеристики шифра Whirlpool

Размер блока: 512 бит
Размер ключа шифра: 512 бит
Число раундов: 10
Расширение ключа: использование шифра непосредственно с константами раунда в качестве ключей раунда
Подстановка: Преобразование SubBytes
Перестановка: Преобразование ShiftColumns
Смешивание: Преобразование MixRows
Константы раунда: кубические корни первых восьмидесяти простых чисел

Анализ

Хотя Whirlpool не был всесторонне изучен или проверен, он базируется на устойчивой схеме Миагучи-Пренеля — (Miyaguchi-Preneel), а для функции сжатия использует шифр, основанный на AES, относительно которой было доказано, что эта криптографическая система — очень стойкая к атакам. Кроме того, размер дайджеста сообщения тот же, что и в SHA-512. Поэтому, как ожидается, Whirlpool будет очень сильной функцией криптографического хэширования. Однако необходимы серьезные испытания и исследования, чтобы подтвердить это. Единственный установленный недостаток — Whirlpool, который использует шифр как

функцию сжатия, не может быть так же эффективен, как SHA-512, особенно когда он реализуется на аппаратных средствах.

12.4. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Несколько книг дают хороший обзор функций криптографического хэширования — [Sti6] [Sta06], [Sch99], [Mao04], [KPS02], [PHS03] и [MOV97].

Сайты

Нижеследующие сайты дают больше информации о темах, рассмотренных в этой лекции.

http://www.unixwiz.net/techtips/iguide-crypto-hashes.html
http://www.faqs.org/rfcs/rfc4231 .html
http://www.itl.nist.gov/fipspubs/fip 180-1.htm
http://www.ietf.org/rfc/rfc3174.txt
http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html

12.5. Итоги

- Все функции криптографического хэширования должны создавать дайджест фиксированного размера из сообщения переменного размера. Такая функция может быть успешнее всего создана с применением итерации. Функция сжатия неоднократно используется, чтобы создать дайджест. Такая схема называется итеративной хэш-функцией.
- Схема Меркеля-Дамгарда (Merkle-Damgard) — итеративная функция криптографического хэширования, устойчивая к коллизиям, если при этом функция сжатия устойчива к коллизиям. Эта схема сегодня — основа для многих функций криптографического хэширования.
- Есть тенденция использовать два различных подхода в проектировании функции сжатия. В первом подходе функция сжатия сделана на пустом месте, т. е. разработана только для этой цели. Во втором подходе блочный шифр с симметричными ключами служит функцией сжатия.
- Множество функций криптографического хэширования использует функции сжатия, которые сделаны на «пустом месте». Эти функции сжатия специально разработаны для этой цели, которую они обслуживают. Некоторые примеры: группа дайджестов сообщения — MD; группа алгоритмов безопасного хэширования — SHA, RIPEMD и HAVAL.
- Итеративная функция криптографического хэширования может использовать блочный шифр с симметричными ключами вместо функции сжа-

тия. Были предложены несколько схем этого подхода, включая схему Рабина, схему Девиса-Мейера, схема Матиса-Мейера-Осеаса и Миагучи-Пренеля.

- Одна из перспективных функций криптографического хэширования — SHA-512 с 512-битовым дайджестом сообщения, основанным на схеме Меркеля-Дамгарда. Она разработана «на пустом месте» только для этой цели.
- Другая перспективная функция криптографического хэширования — Whirlpool, которая одобрена NESSIE. Whirlpool — итеративная функция криптографического хэширования, основанная на схеме Миагучи-Пренеля, которая использует блочный шифр с симметричными ключами вместо функции сжатия. Блочный шифр — измененный и специализированный для этой цели шифр AES.

12.6. Набор для практики

Обзорные вопросы

1. Определите функцию криптографического хэширования.
2. Определите итеративную функцию криптографического хэширования.
3. Опишите идею схемы Меркеля-Дамгарда и объясните, почему эта идея важна для разработки функции криптографического хэширования.
4. Перечислите представителей семейства хэш-функций, которые используют шифр как функцию сжатия.
5. Перечислите некоторые схемы, который были разработаны, чтобы использовать блочный шифр как функцию сжатия.
6. Перечислите главные особенности функции криптографического хэширования SHA-512. Какой типа функции сжатия используется в SHA-512?
7. Перечислите некоторые особенности функции криптографического хэширования. Какая функция сжатия используется в Whirlpool?
8. Сравните характерные особенности SHA-512 и функций криптографического хэширования Whirlpool.

Упражнения

1. В SHA-512 покажите значение поля длины в шестнадцатеричной форме для следующих длин сообщения:
   a. 1000 битов
   b. 10 000 битов
   c. 1000 000 битов
2. В Whirlpool покажите значение поля длины в шестнадцатеричной форме для следующих длин сообщения:
   a. 1000 битов
   b. 10 000 битов
   c. 1000 000 битов
3. Каково дополнение для SHA-512, если длина сообщения:
   a. 5120 битов

b. 5121 бит
c. 6143 бита

4. Каково дополнение для Whirlpool, если длина сообщения:
   a. 5120 битов
   b. 5121 бит
   c. 6143 бита
5. В каждом из следующих случаев покажите, что если два сообщения имеют одни и те же последние блоки, то их последний блок после дополнения поля длины один и тот же:
   a. хэш-функция — SHA-512
   b. хэш-функция — Whirlpool
6. Вычислите G_0 в таблице 12.2, используя седьмое простое число (17).
7. Сравните функцию сжатия SHA-512 без последней операции (конечное сложение) с шифром Файстеля на 80 раундов. Показать совпадения и отличия.
8. Функцию сжатия, используемую в SHA-512 (рис. 12.10), можно представить как шифр с процессом шифрования с 80 раундами, если слова от W_Q до W_{79}, представляют как ключи раунда в одной из схем, рассмотренных в этой лекции (Рабина, Дэвиса-Меейра, Мэтиса-Мейера-Осеаса или Миагучи-Пренеля). Что это напоминает? Подсказка: подумайте об эффекте операции *конечного сложения*.
9. Покажите, что SHA-512 может быть субъектом «атаки сведения к середине», если из функции сжатия удалена операция *конечного сложения*.
10. Составить таблицу, такую же как таблица 12.5, чтобы сравнить AES и Whirlpool.
12. Показать, что третья операция не может быть удалена из десятого раунда в шифре Whirlpool, но она должна быть удалена в шифре AES.
13. Найдите результат операции $\text{RotR}_{12}(x)$, если
    x = 1234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
14. Найдите результат операции $\text{ShL}_{12}(x)$, если
    x = 1234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
15. Найдите результат операции Rotate(x), если
    x = 1234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
16. Найдите результат функции Conditional (x,y,z), если
    x = 1234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
    y = 2234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
    z = 3234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
17. Найдите результат функции Majority (x, y, z), если
    x = 1234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
    y = 2234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
    z = 3234 5678 **ABCD** 2345 34564 5678 **ABCD** 2468
18. Напишите процедуру (в псевдокоде) для вычисления $\text{RotR}_i(x)$ в SHA-512 (рис. 12.9).
19. Напишите процедуру (в псевдокоде), чтобы вычислить ShL_i, (x) в SHA-512 (рис. 12.9).

20. Напишите процедуру (в псевдокоде) для функции Conditional в SHA-512 (рис. 12.11).
21. Напишите процедуру (в псевдокоде) для функции Rotate в SHA-512 (рис. 12.11).
22. Напишите процедуру (в псевдокоде), чтобы вычислить начальный дайджест (значения A_0 до H_0) в SHA-512 (таблица 12.2).
23. Напишите процедуру (в псевдокоде), чтобы вычислить восемьдесят констант в SHA-512 (таблица 12.3).
24. Напишите процедуру (в псевдокоде) для алгоритма расширения слова в SHA-512, показанном на рис. 12.9. Рассмотрите два случая:
    а. Использование массива 80 элементов, содержащих все слова.
    б. Использование массива 16 элементов, содержащего только 16 слов одновременно.
25. Напишите процедуру (в псевдокоде) для функции сжатия в SHA-512.
26. Напишите процедуру (в псевдокоде), чтобы изменить блок 512 бит к 8 × 8 матрицам состояний (рис. 12.4).
27. Напишите процедуру (в псевдокоде), чтобы изменить состояний 8 × 8 матрицу для блоков 512 бит (рис. 12.4).
28. Напишите процедуру (в псевдокоде) для преобразования SubBytes в шифре Whirlpool (рис. 12.16).
29. Напишите процедуру (в псевдокоде) для ShiftColumns преобразования в шифре Whirlpool (рис. 12.18).
30. Напишите процедуру (в псевдокоде) для MixRows преобразования в шифре Whirlpool (рис. 12.19).
31. Напишите процедуру (в псевдокоде) для AddRoundKey преобразования в шифре Whirlpool (рис. 12.20).
32. Напишите процедуру (в псевдокоде) для расширения ключа в шифре Whirlpool (рис. 12.21)
33. Напишите процедуру (в псевдокоде), чтобы создать константы раунда в шифре Whirlpool (Рисунок 12.20).
34. Напишите процедуру (в псевдокоде) для шифра Whirlpool.
35. Напишите процедуру (в псевдокоде) для функции криптографического хэширования Whirlpool.
36. Используйте Internet (или другие доступные ресурсы), чтобы найти информацию о SHA-1. Затем сравните функцию сжатия в SHA-1 с такими же функциями в SHA-512. Каковы совпадения? Каковы отличия?
37. Используйте Internet (или другие доступные ресурсы), чтобы найти информацию о следующих функциях сжатия и сравнить их с SHA-512:
    а. SHA-224
    б. SHA-256
    в. SHA-384
38. Используйте Internet (или другие доступные ресурсы), чтобы найти информацию о RIPEMD и сравнить ее с SHA-512.
39. Используйте Internet (или другие доступные ресурсы), чтобы найти информацию о HAVAL и сравнить ее с SHA-512.

Лекция 13. Цифровая подпись

Цели и содержание

Эта лекция имеет несколько целей.

- Определить понятие цифровой подписи.
- Рассмотреть службы безопасности, обеспеченные цифровой подписью.
- Определить атаки цифровых подписей.
- Обсудить некоторые схемы цифровой подписи, включая RSA, Эль-Гамаля (ElGamal), Шнорра (Schnorr), DSS и эллиптической кривой.
- Описать некоторые приложения цифровых подписей.

Мы все знакомы с понятием подписи. Человек подписывает документ, чтобы показать, что нечто сделано им самим или было им одобрено. Подпись — доказательство получателю, что документ исходит от истинного объекта. Когда клиент подписывает чек, банк уверен, что чек принадлежит клиенту и никому другому. Другими словами, подпись на документе является признаком установления подлинности автора – подлинности документа. Например картина, подписанная художником. Подпись на предмете искусства, если она подлинная, означает, что этот предмет, с большой вероятностью, подлинный.

Когда Алиса передает сообщение Бобу, Боб должен проверить подлинность передатчика; он должен убедиться, что сообщение исходит от Алисы, а не от Евы. Боб может попросить, чтобы Алиса подписала сообщение с помощью электроники. Другими словами, электронная подпись может доказать подлинность Алисы как передатчика сообщения. Мы называем этот тип подписи **цифровая подпись.**

В этой лекции мы сначала рассмотрим некоторые проблемы, связанные с цифровыми подписями, а затем перейдем к рассмотрению различных схем цифровой подписи.

13.1. Сравнение

Рассмотрение начнем с различия между обычными подписями и цифровыми подписями.

Включение

Обычная подпись включена в документ; это — часть документа. Когда мы пишем чек, подпись находится на чеке; это — не отдельный документ. Но когда мы подписываем документ в цифровой форме, мы передаем подпись как отдельный документ. Передатчик передает два документа: сообщение и подпись. Получатель получает оба документа и проверяет, что подпись принадлежит предполагаемому передатчику. Если это доказано, сообщение сохраняется; иначе оно отклоняется.

Метод проверки

Второе отличие между двумя типами подписей — метод подтверждения подписи. В случае обычной подписи, когда получатель получает документ, он сравни-

вает подпись на документе с подписью в архиве. Если они одинаковые, документ подлинный. Получатель должен иметь в архиве копию для сравнения этой подписи. При цифровой подписи – получатель получает сообщение и подпись, но копия подписи не хранится нигде. Получатель должен применить методику проверки комбинации сообщения и подписи, чтобы проверить подлинность.

Отношения

В случае обычной подписи есть отношения «один ко многим» между подписью и документами. Человек использует одну и ту же подпись, чтобы подписать много документов. Для цифровой подписи есть непосредственные отношения между подписью и сообщением. Каждое сообщение имеет свою собственную подпись. Подпись одного сообщения не может использоваться в другом сообщении. Если Боб получает два сообщения, один за другим, от Алисы, он не может использовать подпись первого сообщения, чтобы проверить второе. Каждое сообщение нуждается в новой подписи.

Резервное копирование

Другая разность между двумя типами подписей — качество, названное *резервное копирование*. При обычной подписи копия подписанного документа может отличаться от оригинала. Может быть скорректирована и подписана вновь той же подписью. В цифровой подписи нет такого различия, если нет указателя времени (такого как метка времени) на документе. Например, предположите, что Алиса передает документ, инструктирующий Боба заплатить Еве. Если Ева перехватывает документ и сохраняет подпись, она может воспользоваться этим позже (скопировать подпись), чтобы снова получить деньги от Боба.

13.2. Процесс

Рисунок 13.1 показывает процесс цифровой подписи. Передатчик использует ***алгоритм подписания***, чтобы подписать сообщение. Сообщение и подпись передают приемнику, приемник получает сообщение и подпись и применяет *алгоритм* ***подтверждения*** к комбинации «сообщение — подпись». Если результат истинен, сообщение принято; иначе — отклонено.

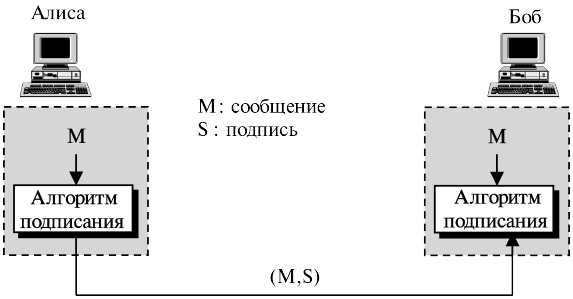


Рис. 13.1. Процесс передачи цифровой подписи

Потребность в ключах

Обычная подпись похожа на частный «ключ», принадлежащий лицу, которое подписывает документ. Подписывающее лицо использует его, чтобы подписать документы; никто другой не имеет эту подпись. Копия подписи находится в архиве подобно открытому ключу; любой может использовать его, чтобы проверить документ, сравнить подпись с первоначальной подписью.

В цифровой подписи подписывающее лицо применяет свой секретный ключ — прикладной алгоритм подписания — для подписания документа. С другой стороны проверяющий использует открытый ключ подписывающего лица — прикладной алгоритм подтверждения — для того, чтобы проверить документ.

Мы можем добавить секретный и открытый ключи к рисунку 13.1, чтобы дать законченное понятие цифровой подписи (см. рис. 13.2). Обратите внимание, что когда документ подписан, любой, включая Боба, может проверить это, потому что каждый имеет доступ к открытому ключу Алисы. Алиса не должна применять свой открытый ключ для того, чтобы подписывать документ, потому что тогда любой сможет подделать ее подпись.

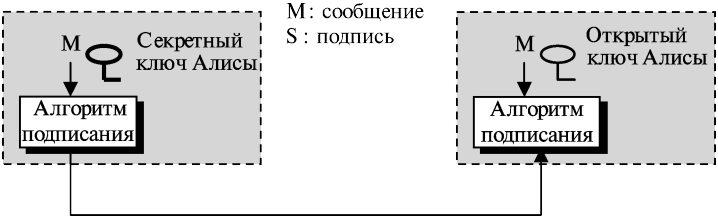


Рис. 13.2. Дополнение передачи цифровой подписи ключом Алисы

Можем ли мы использовать секретный (симметричный) ключ, чтобы и подписать и проверить подпись? Ответ отрицателен по нескольким причинам.

Первая: ключ засекречивания известен только двум объектам (например, Алисе и Бобу). Так, если Алиса должна подписать другой документ и передать его Тэду, она должна применить другой ключ засекречивания.

Вторая: как мы увидим, создавая ключ засекречивания для сеанса, в него включают признаки установления подлинности, которые используют цифровую подпись. Мы получим порочный круг.

Третье: Боб может применить ключ засекречивания, который он использует между собой и Алисой, подписать документ, передать его Тэду и имитировать, что он прибыл от Алисы.

Цифровая подпись нуждается в применении системы открытого ключа. Подписывающее лицо подписывается своим секретным ключом; принимающий проверяет его общедоступным ключом подписывающего лица.

Мы должны понимать различие между секретным и открытым ключами, используемыми в цифровых подписях, и секретным и открытым ключами, которые применяются в криптографической системе для конфиденциальности. В последних секретный и открытый ключи приемника используются в процессе. Передатчик применяет открытый ключ приемника, чтобы зашифровать сообщение; приемник использует свой собственный секретный ключ, чтобы расшифровать сообщение. В цифровой подписи нужен и секретный и открытый ключи передатчика. Передатчик использует свой секретный ключ, приемник — открытый ключ передатчика.

Криптографическая система использует секретный и открытый ключи приемника, цифровая подпись — секретный и открытый ключи передатчика.

Подписание дайджеста

В лекции 10 мы узнали, что криптосистемы с асимметричными ключами очень неэффективны, когда имеют дело с длинными сообщениями. В системе цифровой подписи сообщения обычно длинные, но мы должны использовать асимметрично-ключевые схемы. Решение состоит в том, что подпись дайджеста сообщения — намного короче, чем сообщение. Как мы узнали в лекции 11, тщательно выбранный дайджест сообщения имеет непосредственное отношение к сообщению. Передатчик может подписать дайджест сообщения, а приемник может проверить дайджест сообщения, — эффект тот же самый. Рисунок 13.3 показывает подписание дайджеста в системе цифровой подписи.

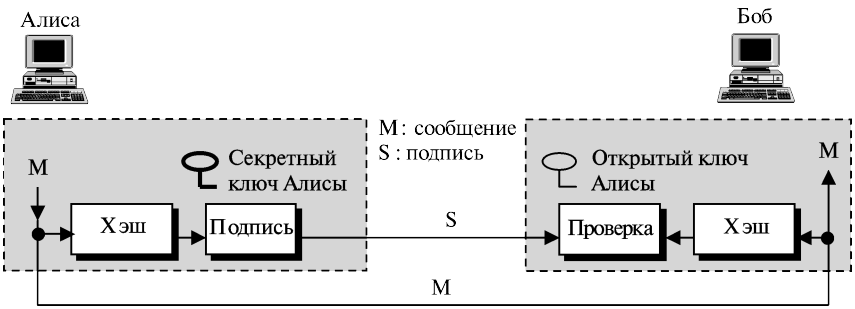


Рис. 13.3. Подписание дайджеста

Дайджест получается из сообщения на стороне Алисы и проходит процесс подписания, используя секретный ключ Алисы. Алиса затем передает сообщение и подпись Бобу. Как мы увидим позже в этой лекции, есть варианты в процессе подписания, которые зависят от системы. Например, могут быть проведены дополнительные вычисления, прежде чем получается дайджест, или может быть использована другая система засекречивания. В некоторых системах подпись имеет множество значений.

На стороне Боба применяется та же самая общедоступная хэш-функция, и сначала из полученного сообщения создается дайджест. Далее вычисляются подпись и дайджест. Процесс подтверждения также проверяет прикладные критерии по результатам вычисления, чтобы определить подлинность подписи. Если подпись подлинна, то сообщение принимается; в противном случае она отклоняется.

13.3. Услуги

Мы обсуждали несколько услуг безопасности в лекции 1, включая *конфиденциальность сообщения, установление подлинности сообщения, целостность сообщения* и *исключение отказа от сообщения*. Цифровая подпись может непосредственно обеспечить последние три; для конфиденциальности сообщения мы все еще нуждаемся в шифровании/дешифровании.

Установление подлинности сообщения

Безопасность схемы цифровой подписи похожа на безопасность обычной подписи (например, она не может быть легко скопирована) и способна обеспечить установление подлинности сообщения (также иногда называется установлением подлинности происхождения данных). Боб может проверить, что сообщение передала Алиса, потому что при проверке используется общедоступный ключ Алисы. Общедоступный ключ Алисы не сможет проверить подпись, подписанную секретным ключом Евы.

Цифровая подпись обеспечивает установление подлинности сообщения.

Целостность сообщения

Целостность сообщения сохраняется, даже если мы подписываем все сообщение, потому что мы не можем получить ту же самую подпись, если сообщение изменено. Схемы цифровой подписи сегодня используют хэш-функцию при подписании и подтверждении, — алгоритмы, которые сохраняют целостность сообщения.

Цифровая подпись обеспечивает целостность сообщения.

Исключение отказа от сообщения

Если Алиса подписывает сообщение и затем отрицает это, может ли Боб доказать, что Алиса фактически подписала его? Например, Алиса передает сообщение банку (Боб) и просит перечислить 10000 $ с ее счета на счет Тэда. Может ли Алиса потом отрицать, что она передала это сообщение? Согласно схеме, которую мы рассматривали до сих пор, Боб мог бы иметь проблему. Боб должен был сохранить подпись в архиве и далее использовать открытый ключ Алисы, чтобы создать

первоначальное сообщение, и доказать, что сообщение в архиве и недавно полученное сообщение является одним и тем же. Это невыполнимо, потому что Алиса может изменить за это время свой секретный или открытый ключ. Она может также утверждать, что сообщение в архиве, содержащее подпись, не подлинное.

Одно из решений — третье лицо (доверенное лицо). Люди могут по договоренности выбрать сторону, которой они доверяют. В будущих лекциях мы увидим, что сторона, которой доверяют, может решить много других проблем относительно служб безопасности и замены ключей. Рисунок 13.4 показывает, как сторона, которой доверяют, может препятствовать отрицанию Алисой передачи данного сообщения.

Алиса создает подпись из своего сообщения (S_A) и передает центру сообщение, которое содержит ее опознавательные признаки, опознавательные признаки Боба, а также подпись. Центр, после проверки правильности открытого ключа Алисы, проверяет с помощью этого ключа Алисы, что сообщение прибыло от Алисы. Затем Центр сохраняет копию сообщения с опознавательными признаками передатчика, опознавательными признаками получателя, а также с меткой времени, в своем архиве. Центр использует свой секретный ключ, чтобы создать из сообщения другую подпись (S_T). Затем центр передает сообщение, новую подпись, опознавательные признаки Алисы и опознавательные признаки Боба — Бобу. Боб проверяет сообщение, используя общедоступный ключ центра, которому он доверяет.

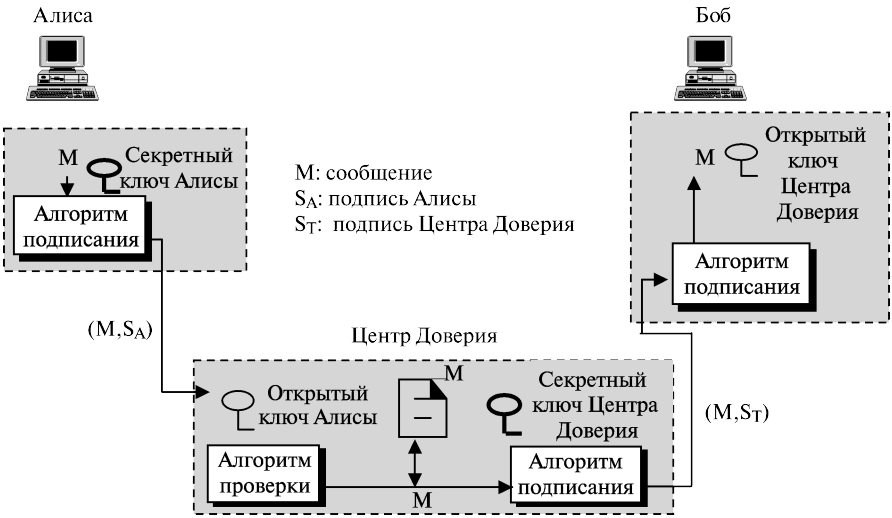


Рис. 13.4. Использование Центра Доверия для исключения отказа от сообщения

Если в будущем Алиса отрицает, что она передала сообщение, центр может предъявить копии сохраненного сообщения. Если сообщение Боба — дубликат сообщения, сохраненного в Центре, Алиса проиграет спор. Чтобы обеспечить

всему этому конфиденциальность, можно добавить к схеме уровень шифрования/дешифрования, как это будет показано в следующей секции.

Исключение отказа от сообщения может быть обеспечено участием третьей стороны, которой доверяют.

Конфиденциальность

Цифровая подпись не обеспечивает конфиденциальную связь. Если конфиденциальность все же требуется, то сообщение и подпись должны быть зашифрованы с использованием любого ключа засекречивания (криптосистема с открытым ключом). Рисунок 13.5 показывает, как этот дополнительный уровень можно добавить к простой схеме цифровой подписи.

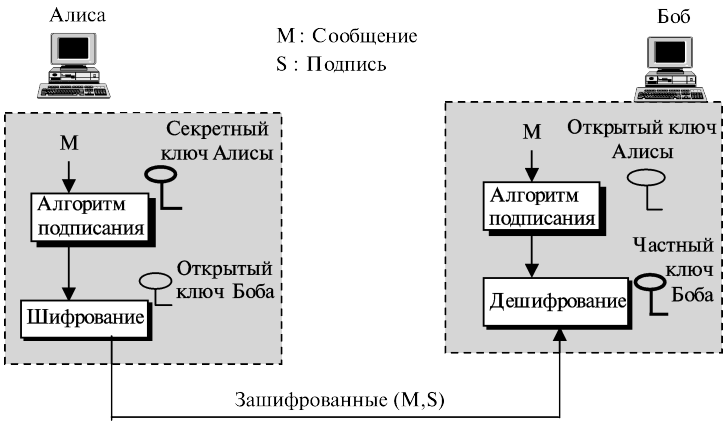


Рис. 13.5. Дополнение конфиденциальности к схеме цифровой подписи

Мы показали асимметрично-ключевое шифрование/дешифрование только для того, чтобы обратить ваше внимание на типы ключей, используемые в каждом конце передачи. Шифрование/дешифрование может также быть сделано симметричным ключом.

Цифровая подпись не обеспечивает секретность. Если есть потребность в секретности, должен быть применен уровень шифрования/дешифрования.

13.4. Атаки цифровой подписи

Эта секция описывает некоторые атаки цифровых подписей и определяет типы подделки.

Типы атаки

Мы рассмотрим три вида атак цифровых подписей:
- атака только на ключ,
- атака при известном сообщении,
- атака по выбранному сообщению.

Атака только на ключ

В **атаке только на ключ** Ева имеет доступ только к общедоступной информации, которую передает Алиса. Для того чтобы подделать сообщение, Ева должна создать подпись Алисы, чтобы убедить Боба, что сообщение прибывает от Алисы. Это можно рассматривать как атаку только зашифрованного текста, которую мы обсуждали при шифровании.

Атака при известном сообщении

В **атаке при известном сообщении** Ева имеет доступ к одной или более паре «подпись — сообщения». Другими словами, она имеет доступ к некоторым документам, предварительно подписанным Алисой. Ева пробует создать другое сообщение и подделать подпись Алисы. Это подобно атаке знания исходного текста, которую мы обсуждали при шифровании.

Атака по выбранному сообщению

В **атаке по выбранному сообщению** Ева так или иначе заставила Алису подписать одно или более сообщений для нее. Ева теперь имеет пару «выбранное сообщение / подпись». Через некоторое время она создает другое сообщение, с содержанием, которое она выбирает в своих интересах, и подделывает подпись Алисы. Это подобно атаке с выборкой исходного текста, которую мы обсуждали при шифровании.

Типы подделки

Если атака успешна, то в результате появляется подделка. Мы можем иметь два типа подделки: экзистенциальная и селективная.

Экзистенциальная подделка

В **экзистенциальной подделке** Ева способна создать правильную пару «подпись — сообщение», но ни одну из подделок она не может реально использовать. Другими словами, документ был подделан, но содержание восстановлено беспорядочно. Этот тип подделки вероятен, но, к счастью, Ева не может извлечь из этого выгоду. Ее подделанное сообщение синтаксически или семантически непонятно.

Селективная подделка

В селективной подделке Ева способна подделать подпись Алисы на сообщении с содержанием, выбранным Евой. Это выгодно для Евы и может быть очень вредно для Алисы, но вероятность такой подделки мала, хотя имеет реальную величину.

13.5. Схемы цифровой подписи

Несколько **схем цифровой подписи** были распространены в течение прошлых нескольких десятилетий. Некоторые из них были реализованы. В этой секции мы поговорим об этих схемах. В следующей секции мы обсуждаем одну из них, которая, вероятно, станет стандартом.

Схема цифровой подписи RSA

В лекции 10 мы рассмотрели, как использовать криптографическую систему RSA для обеспечения секретности. Идея RSA может также применяться для того, чтобы подписать и подтвердить сообщение. В этом случае это называется **схемой цифровой подписи RSA.** Схема цифровой подписи меняет роли секретных и открытых ключей. Первое: применяются секретный и открытый ключи передатчика, а не приемника. Второе: передатчик использует свой собственный секретный ключ для подписи документа; приемник использует открытый ключ передатчика, чтобы проверить этот документ. Если мы сравним схему с обычным способом подписи, мы увидим, что секретный ключ играет роль передатчика собственной подписи, а открытый ключ — роль передатчика копии подписи, которая является общедоступной. Очевидно, Алиса не может использовать открытый ключ Боба, чтобы подписать сообщение, потому что тогда любой другой человек мог бы сделать то же самое. Рисунок 13.6 дает общую идею схемы цифровой подписи RSA.

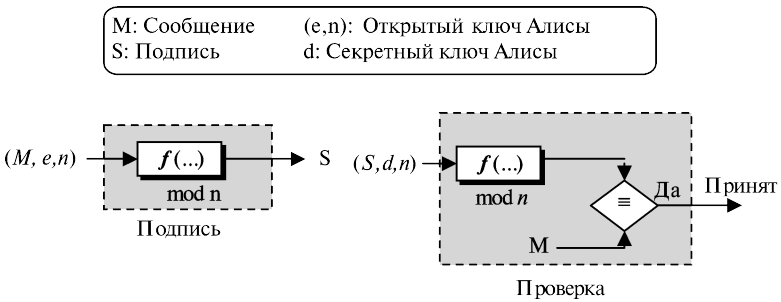


Рис. 13.6. Общая идея схемы цифровой подписи RSA

Подписание и подтверждение сайтов используют ту же самую функцию, но с различными параметрами. Верификатор сравнивает сообщение и вывод функции для сравнения. Если результат истинен, сообщение принято.

Генерация ключей

Генерация ключей в схеме цифровой подписи RSA точно такая же, как и генерация ключей в криптографической системе RSA (см. лекцию 10). Алиса выбирает два простых числа p и q и вычисляет $n = p \times q$. Алиса вычисляет $\phi(n) = (p - 1)(q - 1)$. Затем она выбирает e для общедоступного ключа и вычисляет d для частного ключа, такое, что $e \times d = 1 \bmod \phi(n)$. Алиса сохраняет d и публично объявляет n и e.

В схеме цифровой подписи RSA *d* является секретным; *e* и *n* — открытым.

Подписание и проверка

Рисунок 13.7 показывает схему цифровой подписи RSA.

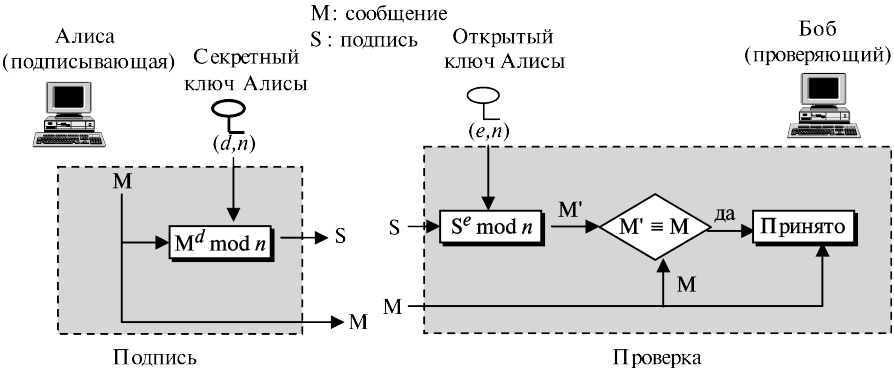


Рис. 13.7. Схема цифровой подписи RSA

Подписание. Алиса на основе сообщения создает подпись, используя частный (секретный) ключ, $S = M^d \bmod n$, и передает сообщение и подпись Бобу.

Проверка. Боб получает M и S. Он применяет общедоступный ключ Алисы к подписи, чтобы создать копию сообщения $M' = S^e \bmod n$. Боб сравнивает значение M' со значением M. Если два значения совпадают, Боб принимает сообщение. Чтобы доказать правильность этой процедуры, мы применяем критерии проверки:

$$M' \equiv M(\bmod n) \quad \rightarrow \quad S^e \equiv M(\bmod n) \quad \rightarrow \quad M^{d \times e} \equiv M(\bmod n)$$

Последнее сравнение справедливо, потому что $d \times e = 1 \bmod \phi(n)$ (см. теорему Эйлера в лекции 9).

Пример 13.1

Для безопасности подписи значения *p* и *q* должны быть очень большими. Как тривиальный пример, предположим, что Алиса выбирает $p = 823$ и $q = 953$ и вычисляет $n = 784319$. Значение $\phi(n)$ — 782544. Теперь она выбирает $e = 313$ и вычисляет $d = 160009$. В этой точка генерация ключей закончена. Теперь вообразим, что Алиса хочет передать сообщение со значением M = 19070 Бобу. Она использует свой частный ключ 160009 для того, чтобы подписать сообщение:

$$M = 19070 \rightarrow S = (19070^{160009}) \bmod 784319 = 210625 \bmod 784319$$

Алиса передает сообщение и подпись Бобу. Боб получает сообщение и подпись. Он вычисляет

$$\mathbf{M'} = 210625^{313} \bmod 784319 = 19070 \bmod 784319 \rightarrow \text{M M} \equiv \text{M'} \bmod n$$

Боб принимает сообщение, потому что он проверил подпись Алисы.

Атаки подписи RSA

Есть некоторые атаки, к которым Ева может обратиться для подделки схемы цифровой подписи RSA Алисы.

Атака только на ключ. Ева имеет доступ только к открытому ключу Алисы, перехватывает пару (M, S) и пробует создать другое сообщение M', такое, что $M' = S^e \bmod n$. Эта проблема по сложности решения равна проблеме дискретного логарифма, которую мы рассмотрели в лекции 9. Это — экзистенциальная подделка и обычно бесполезна для Евы.

Атака при известном сообщении. Здесь Ева использует *мультипликативное свойство* RSA. Предположим, что Ева перехватила две пары подписи сообщения — (M_1, S_1) и (M_2, S_2), которые используют один и тот же секретный ключ. Если $M = (M_1 \times M_2) \bmod n$, тогда $S = (S_1 \times S_2) \bmod n$. Это просто доказать, потому что мы имеем

$$S = (S_1 \times S_2) \bmod n = (M_1^d \times M_2^d) \bmod n = (M_1 \times M_2)^d \bmod n = M^d \bmod n$$

Ева может создать $M = (M_1 \times M_2) \bmod n$ и может создать $S = (S_1 \times S_2) \bmod n$; глупый Боб поверит, что S — подпись Алисы на сообщении M. Эта атака, которая называется иногда *мультипликативной атакой*, проводится очень просто. Однако это — экзистенциальная подделка, так как сообщение M является произведением двух предыдущих сообщений, созданных Алисой, а не Евой; сообщение M обычно бесполезно.

Атака по выбранному сообщению. Эта атака также использует мультипликативное свойство RSA. Ева может так или иначе попросить, чтобы Алиса подписала два законных сообщения M_1 и M_2. С помощью их она позже создает новое сообщение $M = (M_1 \times M_2)$. Ева может позже утверждать, что Алиса подписала M. Такую атаку называют так же, как и предыдущую — *мультипликативная атака*. Это очень серьезная атака схемы цифровой подписи RSA, потому что это — селективная подделка. (Ева может перемножать M_1 и M_2, чтобы получить полезный M).

Подпись RSA на дайджесте сообщения

Как мы обсуждали прежде, подписание дайджеста сообщения и использование сильного алгоритма хэширования имеет несколько преимуществ. В случае RSA процессы подписания и проверки можно сделать намного быстрее, потому что схема цифровой подписи RSA — не что иное, как шифрование с секретным ключом и дешифрование с открытым ключом. Использование сильной графической функции хэширования также делает атаку подписи намного труднее, что мы коротко объясним. Рисунок 13.8 иллюстрирует схему.

Алиса, подписывающее лицо, при первом использовании согласует хэш-функцию, чтобы создать дайджест сообщения $D = h(M)$. Затем она подписывает дайджест, $S = D^d \bmod n$. Сообщение и подпись передают Бобу. Боб, проверяющий, получает сообщение и подпись. Он сначала использует открытый ключ Алисы,

чтобы извлечь (файл) дайджест, D' = S^e mod n. Затем он применяет хэш-алгоритм для того, чтобы получить сообщение D = h(M).

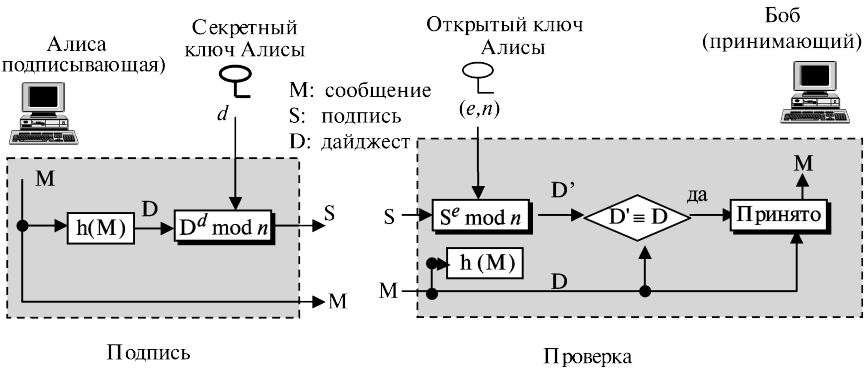


Рис. 13.8. Подпись RSA на дайджесте сообщения

Боб теперь сравнивает эти два дайджеста, D и D'. Если они являются сравнимыми по модулю *n,* он принимает сообщение.

Атаки на подписанные дайджесты RSA

Насколько восприимчива схема цифровой подписи RSA к нападению, когда дайджест уже подписан?

Атака только на ключ. Возможны три случая этой атаки.

a. Ева перехватывает пару (S, M) и пробует найти другое сообщение M', которое создает тот же самый дайджест, h (M) = h (M'). Как мы узнали в лекции 11, если алгоритм хэширования *обладает устойчивостью ко второму прообразу*, эта атака очень трудна.

b. Ева находит два сообщения, M. и M.', такие, что h (M) = h (M'). Она соблазняет Алису подписать h (M), чтобы найти S; теперь Ева имеет пару (M', S), которое прошло подтверждающий тест, но это — подделка. Мы изучали в лекции 11, что если алгоритм хэширования *устойчив к коллизиям*, эта атака очень трудна.

c. Ева случайным подбором может найти дайджест сообщения D, который может соответствовать случайной подписи S. Она тогда находит сообщение M, такое, что D = h (M). Как мы узнали в Лекции 11, если хэш-функция *устойчива к прообразу*, эта атака очень трудно осуществима.

Атака при известном сообщении. Предположим, что Ева имеет две пары подписи сообщения — (M_1,S_1) и (M_2, S_2), которые были созданы с использованием одного и того же секретного ключа. Ева вычисляет $S \equiv S_1 \times S_2$. Если она сможет найти сообщение M, такое, что $h(M) \equiv h(M_1) \times h(M_2)$, она сможет подделать новое сообщение. Однако нахождение M по данному h (M) — это очень трудный процесс, если алгоритм хэширования *устойчив к прообразу.*

Атаки по выбранному сообщению. Ева может попросить, чтобы Алиса подписала два законных сообщения — M_1,и M_2 для нее. Она может создать новую подпись $S \equiv S_1 \times S_2$. Так как Ева может вычислить $h(M) \equiv h(M_1) \times h(M_2)$, если она может найти сообщение M данному h (M), это новое сообщение — подделка. Однако нахождение M по данному h (M) – очень трудный процесс, если алгоритм хэширования *устойчив к прообразу.*

Когда дайджест подписан непосредственно вместо сообщения, восприимчивость схемы цифровой подписи RSA зависит от свойств алгоритма хэширования.

Схема цифровой подписи Эль-Гамаля

Криптосистема Эль-Гамаля была обсуждена в лекции 10. Схема цифровой подписи Эль-Гамаля использует те же самые ключи, но алгоритм различен. Рисунок 13.9 дает общую идею схемы цифровой подписи Эль-Гамаля.

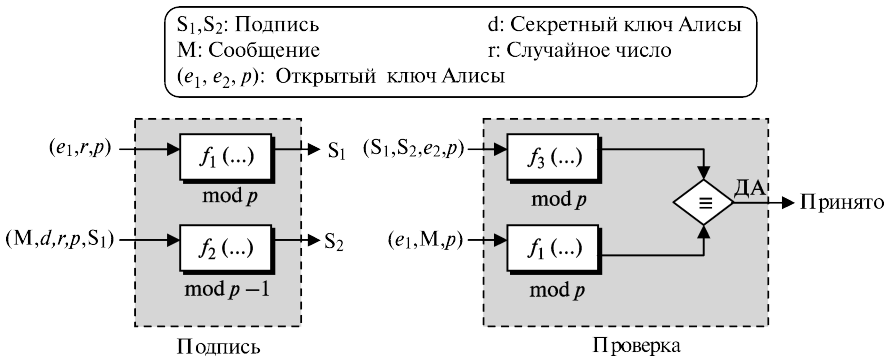


Рис. 13.9. Общая идея схемы цифровой подписи Эль Гамаля

В процессе подписания две функции создают две подписи. На стороне подтверждения обрабатывают выходы двух функций и сравнивают между собой для проверки. Обратите внимание, что одна и та же функция применяется и для подписания, и для проверки, но использует различные входы. Рисунок показывает входы каждой функции. Сообщение — часть входа, для обеспечения функционирования при подписании; оно же — часть входа к функции 1 при подтверждении. Обратите внимание, что вычисления в функциях 1 и 3 проводятся по модулю p, а функции 2 — по модулю $p - 1$.

Генерация ключей

Процедура генерации ключей здесь точно такая же, как та, которая используется в криптографической системе. Выберем достаточно большое простое число p, чтобы в поле Z_{p^*} проблема дискретного логарифма была достаточно труд-

ной. Пусть e_1 — простой элемент в Z_{p^*}. Алиса выбирает свой секретный ключ d, чтобы он был меньше, чем p – 1. Она вычисляет $e_2 = e_1^d$. Открытый ключ Алисы — кортеж (e_1, e_2, p); секретный ключ Алисы — d.

В схеме цифровой подписи Эль-Гамаля (e_1, e_2, p) — открытый ключ Алисы; d —секретный ключ Алисы.

Подтверждение и проверка

Рисунок 13.10 показывает схему цифровой подписи Эль-Гамаля.

Подписывающаяся Алиса может подписать дайджест сообщения, направленный к любому объекту, включая Боба.

1. Алиса выбирает секретное случайное число *r*. Обратите внимание, что хотя открытые и секретные ключи могут использоваться неоднократно, Алиса каждый раз нуждается в новом *r*, когда она подписывает новое сообщение.

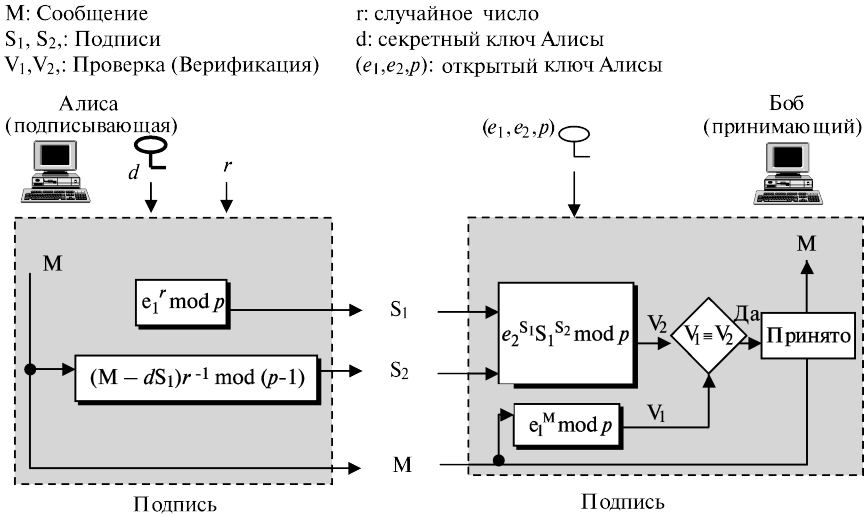


Рис. 13.10. Схема цифровой подписи Эль-Гамаля

2. Алиса вычисляет первую подпись $S_1 = e_1^r \bmod p$.
3. Алиса вычисляет вторую подпись $S_2 = (M - d \times S_1) \times r^{-1} \bmod (p - 1)$, где r^{-1} — мультипликативная инверсия *r* по модулю *p*.
4. Алиса передает M, S_1 и S_2 Бобу.

Проверка. Объект, например Боб, получает M, S_1 и S_2 и может проверить их следующим образом.

1. Боб проверяет, что $0 < S_1 < p$.
2. Боб проверяет, что $0 < S_2 < p - 1$.
3. Боб вычисляет $V_1 = e_1^M \bmod p$.
4. Боб вычисляет $V_2 = e_2^{S_1} \times S_1^{S_2} \bmod p$.

5. Если V_1 является сравнимым по модулю p с V_2, сообщение принято; иначе оно будет отклонено. Мы можем доказать правильность этого критерия проверки, используя $e_2 = e_1^d$ и $S_1 = e_1^r$:

$$V_1 \equiv V_2 \bmod p \rightarrow e_1^{M} \equiv e_2^{S_1} \times S_1^{S_2} \bmod p \equiv (e_1^{d_1})^{S_1} \times (e_1^{r})^{S_2} (\bmod p) \equiv e_1^{dS_1 + rS_2} \bmod p$$

Мы имеем $e_1^{M} \equiv e_1^{dS_1 + rS_2} (\bmod p)$

Поскольку e_1 — первообразный корень, может быть доказано, что вышеупомянутое сравнение справедливо тогда и только тогда, когда $M \equiv [dS_1 + rS_2] \bmod (p-1)$ или $S_2 \equiv [(M - d \times S_1) \times r^{-1}] \bmod (p-1)$, и результат сравнения есть тот же самый S_2, с которого мы начали процесс подписания.

Пример 13.2

Ниже приводится тривиальный пример. Алиса выбрала $p = 3119$, $e_1 = 2$, $d = 127$ и вычислила $e_2 = 2^{127} \bmod 3119 = 1702$. Она выбрала r равным 307. Она объявила e_1, e_2 и p; она сохранила в тайне d. Далее показано, как Алиса может подписать сообщение.

$$M = 320$$
$$S_1 = e_1^{r} = 2^{307} = \bmod 3119$$
$$S_2 = (M - d \times S_1) \times r^{-1} = (320 - 127 \times 2083) \times 307^{-1} = 2105 \bmod 3118$$

Алиса передает M, S_1 и S_2 Бобу. Боб использует открытый ключ, чтобы вычислить, что сообщение подписано Алисой, потому что никто, кроме Алисы, не имеет секретного ключа d.

$$V_1 = e_1^{M} = 2^{30} = 3006 \bmod 3119$$
$$V_2 = d^{S_1} \times S_1^{S_2} = 1702^{2083} \times 2083^{2105} = 3006 \bmod 3119$$

Поскольку V_1 и V_2 являются являются сравнимыми по модулю p, Боб принимает сообщение, и он предполагает, что сообщение было подписано Алисой, потому что никто, кроме нее, не имеет секретного ключа Алисы d.

Пример 13.3

Теперь вообразите, что Алиса хочет передать другое сообщение, M = 3000, Тэду. Она выбирает новое $r = 107$. Алиса передает M., S_1 и S_2 Тэду. Тэд использует общедоступные ключи, чтобы вычислить V_1 и V_2.

$$M = 3000$$
$$S_1 = e_1 = 2^{107} = 2732 \bmod 3119$$
$$S_2 = (M - d \times S_1) r^{-1} = (3000 - 127 \times 2083) \times 107^{-1} = 2526 \bmod 3118$$

$$V_1 = e_1^{M} = 2^{3000} = 704 \bmod 3119$$
$$V_2 = d^{S_1} \times S_1^{S_2} = 1702^{2732} \times 2083^{2526} = 704 \bmod 3119$$

Поскольку V_1 и V_2 являются являются сравнимыми по модулю p, Тэд принимает сообщение; он предполагает, что сообщение подписано Алисой, потому что никто, кроме нее, не имеет секретного ключа Алисы d. Обратите внимание, что сообщение может получить любой человек. Цель состоит не в том, чтобы скрыть сообщение, но в том, чтобы доказать, что его передает Алиса.

Подделка цифровой подписи в схеме Эль-Гамаля

Схема цифровой подписи Эль-Гамаля уязвима к экзистенциальной подделке, но селективную подделку на этой схеме сделать очень трудно.

Подделка только ключа. В этом типе подделки Ева имеет доступ только к открытому ключу. Возможны два варианта.

1. Ева имеет заранее заданное сообщение M. Она должна подделать подпись Алисы на этом сообщении. Ева должна найти две правильных подписи S_1 и S_2 для этого сообщения. Это — селективная подделка.
   a. Ева может выбрать S_2 и вычислить S_1. Она должна иметь $d^{S1} \times S_1^{S2} \equiv e_1^{M} (\text{mod } p)$. Другими словами, $S_1^{S2} \equiv e_1^{M} \times d^{-S1} (\text{mod } p)$ или $S_2 \equiv \log_{S1}(e_1^{M} \times d^{-S1}) (\text{mod } p)$. Это означает вычисление дискретного логарифма, что является очень трудным.
   b. Ева может выбрать S_2 и вычислить S_1. Это намного труднее, чем выполнить часть a.
2. Ева может методом случайного подбора найти три значения, M, S_1 и S_2 , такие, что подпись первого используется для второго. Если Ева может найти два новых параметра x и y, такие, что $M = xS_2 \bmod (p - 1)$ и $S_1 = -y$ $S_2 \bmod (p - 1)$, то она может подделать сообщение, но серьезной выгоды не получит, поскольку это — экзистенциальная подделка.

Подделка при известном сообщении. Если Ева перехватила сообщение M и его две подписи S_1 и S_2, она может найти другое сообщение M', с той же самой парой подписей S_1 и S_2. Однако обратите внимание, что это — экзистенциальная подделка, которая не помогает Еве.

Схема цифровой подписи Шнорра

Проблема схемы цифровой подписи Эль-Гамаля — в том, что p должно быть очень большим, чтобы сделать трудной проблему дискретного логарифма Zp*. Рекомендуется длина p по крайней мере 1024 битов. Можно сделать подпись размером 2048 бит. Чтобы уменьшить размер подписи, Шнорр предложил новую схему, основанную на схеме Эль-Гамаля, но с уменьшенным размером подписи. Рисунок 13.11 дает общую идею **схемы цифровой подписи Шнорра.**

В процессе подписания две функции создают две подписи; в процессе проверки выход одной функции сравнивается с первой подписью для проверки. Рисунок 13.11 показывает входы к каждой функции. Важно то, что схема использует

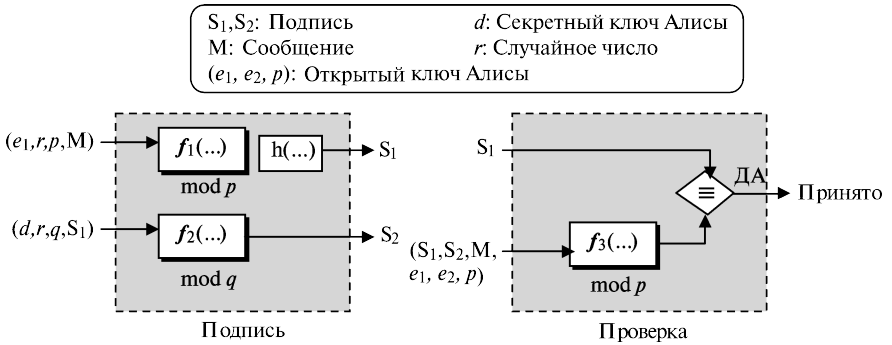


Рис. 13.11 Общая идея схемы цифровой подписи Шнорра

два модуля: p и q. Функции 1 и 3 применяют p; функция 2 — q. Детали входов и функций будут коротко обсуждены далее.

Генерация ключей

Перед подписанием сообщения Алиса должна генерировать ключи и объявить всем открытые ключи.

1. Алиса выбирает простое число p, которое обычно равно по длине 1024 битам.
2. Алиса выбирает другое простое число q, которое имеет такой же размер, что и дайджест, созданный функцией криптографического хэширования (в настоящее время 160 битов, но это может измениться в будущем). Простое число q должно делиться на $(p - 1)$. Другими словами, $(p - 1) = 0 \bmod q$.
3. Алиса выбирает e_1, q-тый корень которого был бы равен 1 mod p. Чтобы сделать это, Алиса выбирает примитивный элемент в Zp, e_0 (см. приложение J) и вычисляет $e_1 = e_0^{(p-1)/q} \bmod p$.
4. Алиса выбирает целое число, d, как свой секретный ключ.
5. Алиса вычисляет $e_2 = e_1^d \bmod p$.
6. Общедоступный ключ Алисы — (e_1, e_2, p, q), ее секретный ключ — (d).

В схеме цифровой подписи Шнорра открытый ключ Алисы — (e_1, e_2, p, q); ее секретный ключ — (d).

Подписание и проверка

Рисунок 13.12 показывает схему цифровой подписи Шнорра.

Подписание

1. Алиса выбирает случайное число r. Обратите внимание, что открытый и секретный ключи могут использоваться для подписи многих сообщений. Но Алиса должна изменять r каждый раз, когда она передает новое сообщение. Обратите внимание также на то, что r должен иметь значение между 1 и q.

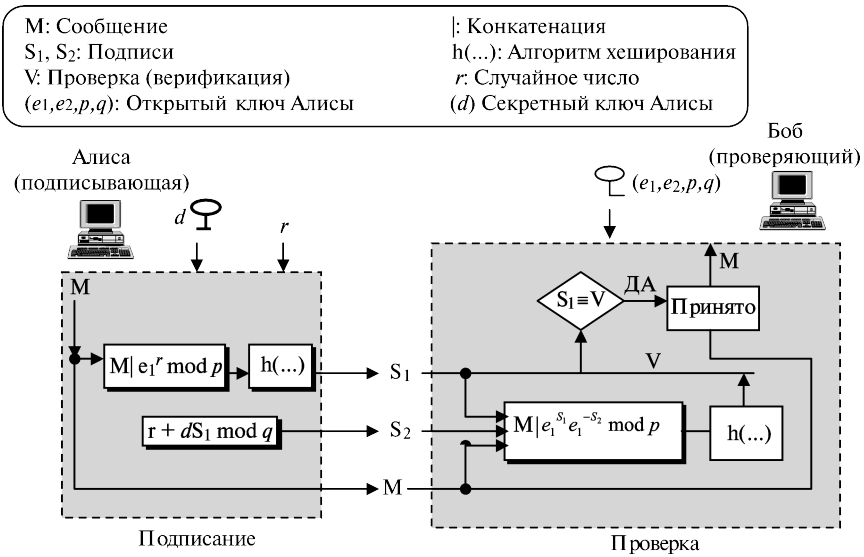


Рис. 13.12. Схема цифровой подписи Шнорра

2. Алиса вычисляет первую подпись $S_1 = h(M \mid e_1^r \bmod p)$. Сообщение присоединяется (конкатенируется) спереди к значению $e_1^r \bmod p$, затем применяется хэш-функция, чтобы создать дайджест. Обратите внимание, что хэш-функция непосредственно не применяется к сообщению, но вместо этого она получается из последовательного соединения M. и $e_1^r \bmod p$.
3. Алиса вычисляет вторую подпись $S_2 = r + d \times S_1 \bmod q$. Обратите внимание, что эта часть — вычисление S_2 — делается в арифметике по модулю q.
4. Алиса передает M., S_1 и S_2.

Верификация (проверка) сообщения. Приемник, например, Боб, получает M, S_1 и S_2.

1. Боб вычисляет $V = h(M \mid e_1^{S_2} e_2^{-S_1} \bmod p)$.
2. Если S_1 конгруэнтно V по модулю p, сообщение принято; иначе оно отклоняется.

Пример 13.4

Вот тривиальный пример. Предположим, что мы выбираем $q = 103$ и $p = 2267$. Обратите внимание на то, что $p = 22 \times q + 1$. Мы выбираем $e_0 = 2$, которое является элементом в Z_{2267}^*. Тогда $(p - 1) / q = 22$, так что мы имеем $e_1 = 2^{22} \bmod 2267 = 354$.

Мы выбираем $d = 30$, тогда $e_2 = 354^{30} \bmod 2267 = 1206$. Секретный ключ Алисы теперь — *(d)*, ее открытый ключ — *(e1,e2,p,q)*.

Алиса хочет передать сообщение M. Она выбирает $r = 11$ и вычисляет $e^r = 354^{11} = 630 \bmod 2267$. Предположим, что сообщение — 1000, и конкатенация (последовательное соединение) означает 1000630. Также предположим, что хэширо-

вание этого значения дает дайджест h (1000630) = 200. Это означает $S_1 = 200$. Алиса вычисляет $S_2 = r + d \times S_1 \bmod q = 11 + 1026 \times 200 \bmod 103 = 11 + 24 = 35$. Алиса передает сообщение M =1000, $S_1 = 200$ и $S_2 = 35$. Проверку оставляем как упражнение.

Подделка по схеме подписи Шнорра

Похоже, что все атаки на схему Эль-Гамаля могут быть применены к схеме Шнорра. Однако схема Шнорра находится в лучшем положении, потому что $S_1 = h(M \mid e_1^r \pmod p)$. Это означает, что хэш-функция применяется к комбинации сообщение и e_1^r, в которой r является секретным.

Стандарт цифровой подписи (DSS)

Стандарт цифровой подписи (DSS – Digital Signature Standard) был принят национальным Институтом Стандартов и Технологии (NIST) в 1994 г. NIST издал DSS как FIPS-186 (FEDERAL INFORMATION PROCESSING STANDARD 186). DSS применяет алгоритм цифровой подписи (DSA), основанный на схеме Эль-Гамаля, с использованием некоторых идей из схемы Шнорра. DSS критиковался со времени его издания. Главная претензия — оценка безопасности проекта DSS. Вторая — размер простого числа, 512 битов. Позже NIST сделал это число переменного размера, чтобы ответить на эту претензию. Рисунок 13.13 дает общую идею схемы DSS.

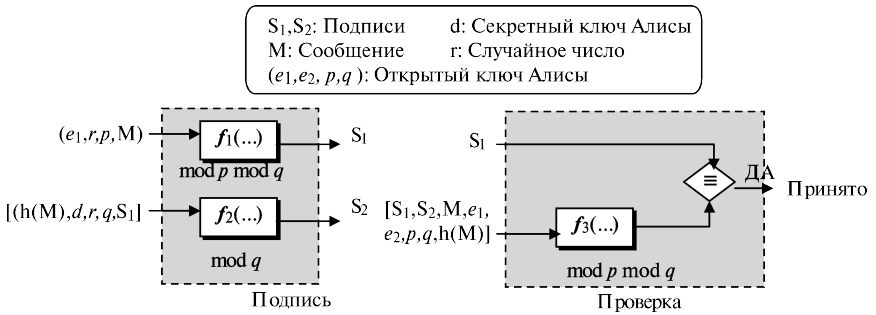


Рис. 13.13. Общая идея схемы DSS

В процессе подписания две функции создают две подписи; в процессе проверки выход одной функции сравнивается с первой подписью для проверки. Это подобно схеме Шнорра, но входы различны. Другое отличие: эта схема использует дайджест сообщения (не само сообщение) как часть входов к функциям 1 и 3. Интересно то, что схема применяет два общедоступных модуля: p и q. Функции 1 и 3 используют оба модуля p и q, функция 2 — только q. Детали входов и функций коротко рассматриваются ниже.

Генерация ключей

Перед подписанием сообщения к любому объекту Алиса должна генерировать ключи и объявить общедоступные ключи.

1. Алиса выбирает простое число длиной p между 512 и 1024 битами. Число битов в p должно быть кратно числу 64.
2. Алиса выбирает простое число на 160 битов q с таким условием, чтобы оно делилось на $(p - 1)$.
3. Алиса использует две группы умножения, $<\mathbf{Z_p}^*, \times>$ и $<Z_q^*, \times>$; вторая — подгруппа первой.
4. Алиса создает e_1, такое, чтобы оно было q-тым корнем 1 по модулю p ($e_1^p = 1 \bmod p$). Она поступает так: выбирает элемент в Z_p, e_0, и вычисляет $e_1 = e_0^{(p-1)/q}$.
5. Алиса выбирает d как секретный ключ и вычисляет $e_2 = e_1^d$.
6. Общедоступный ключ Алисы — (e_1, e_2, p, q), ее секретный ключ – (d).

Подписание и проверка

Рисунок 13.14 показывает схему DSS.

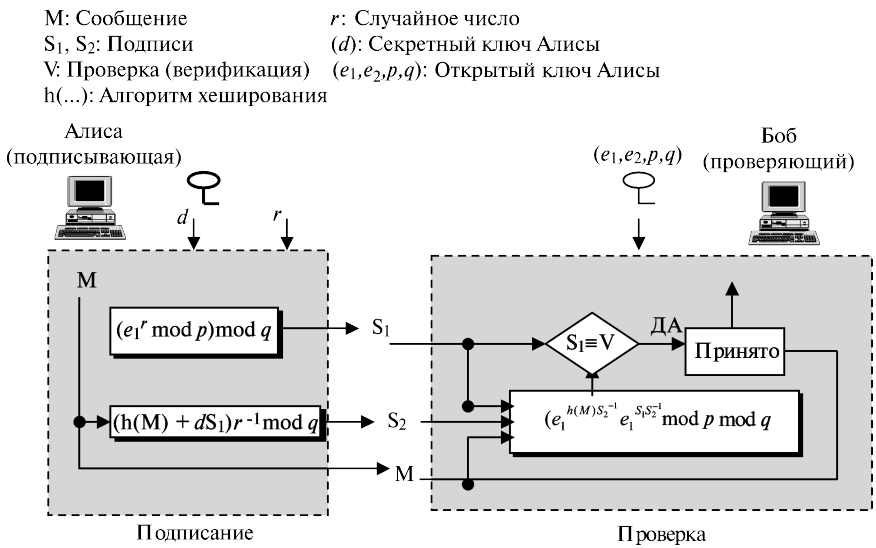


Рис. 13.14. Схема DSS

Подписание. Ниже показаны шаги подписания сообщения.

1. Алиса выбирает случайное число $r (1 \le r \le q)$. Обратите внимание, что хотя открытые и секретные ключи могут быть выбраны один раз и использоваться для того, чтобы подписать много сообщений, Алиса должна выбирать каждый раз новый r, когда она должна подписать новое сообщение.
2. Алиса вычисляет первую подпись $S_1 = (e_1^r \bmod p) \bmod q$. Обратите внимание: значение первой подписи не зависит от M (сообщения).
3. Алиса создает дайджест сообщения h (M).

4. Алиса вычисляет вторую подпись $S_2 = (h(M) + d\ S_1)r^{-1} \bmod q$. Обратите внимание, что вычисление S_2 делается по модулю q.
5. Алиса посылает M, S_1 и S_2 Бобу.

Проверка (верификация). Для проверки сообщения обычно применяются следующие шаги, когда получены M, S_1 и S_2.

1. Боб проверяет S_1 $0 < S_1 < q$.
2. Боб проверяет S_2 $0 < S_2 < q$.
3. Боб вычисляет дайджест M, применяя алгоритм хэширования, используемый Алисой.
4. Боб вычисляет $V = (e_1^{h(M)S_2^{-1}}\ e_2^{S_1S_2^{-1}} \bmod p) \bmod q$.
5. Если S конгруэнтен V, сообщение принимается; иначе — отклоняется.

Пример 13.5

Алиса выбирает $q = 101$ и $p = 8081$. Алиса выбирает $e_0 = 3$ и вычисляет $e_1 = e_0^{(p-1)/q} \bmod p = 6968$. Алиса выбирает $d = 61$ в качестве секретного ключа и вычисляет $e_2 = e_1^d \bmod p = 2038$. Теперь Алиса может передать сообщение Бобу. Предположим, что $h(M) = 5000$, и Алиса выбирает $r = 61$:

$$h(M) = 5000 \quad r = 61$$
$$S_1 = (e_1^r \bmod p) \bmod q = 54$$
$$S_2 = (h(M) + dS_1)r^{-1} \bmod q = 40$$

Алиса передает M, S_1 и S_2 Бобу. Боб использует общедоступные ключи, чтобы вычислить V.

$$S_2^{-1} = 48 \bmod 101$$
$$V = [(6968^{5000\times48} \times 2038^{54\times48} \times 2038^{54\times48}) \bmod 8081] \bmod 101 = 54$$

Поскольку S_1 и V являются сравнимыми, Боб принимает сообщение.

Сравнение DSS и RSA

Вычисление DSS-подписи быстрее, чем вычисление подписей RSA, при использовании того же самого p.

Сравнение DSS и схемы Эль-Гамаля

DSS-подпись — меньше, чем подписи в схеме Эль-Гамаля, потому что q меньше, чем p.

Схема цифровой подписи эллиптической кривой

Наша последняя схема — **схема цифровой подписи эллиптической кривой (ECDSS — Elliptic Curve Digital Signature Scheme)**, которая основана на применении эллиптических кривых, — их мы обсуждали в Лекции 10. Схема иногда упо-

минается как ECDSA (Elliptic Curve Digital Signature Algorithm). Рисунок 13.15 показывает общую идею ECDSS.

В процессе подписания две функции и экстрактор (извлекающее устройство) создают две подписи; в процессе проверки (верификации) обрабатывают выход одной функции (после прохождения через экстрактор) и сравнивают ее с первой подписью для проверки. Функции f_1 *и* f_3 фактически создают точки на кривой. Первая создает новую точку для секретного ключа подписывающего лица. Вторая — новую точку из двух общедоступных ключей подписывающего лица. Каждый экстрактор извлекает первые координаты соответствующей точки в модульной арифметике. Детали входов и функций коротко обсуждаются далее.

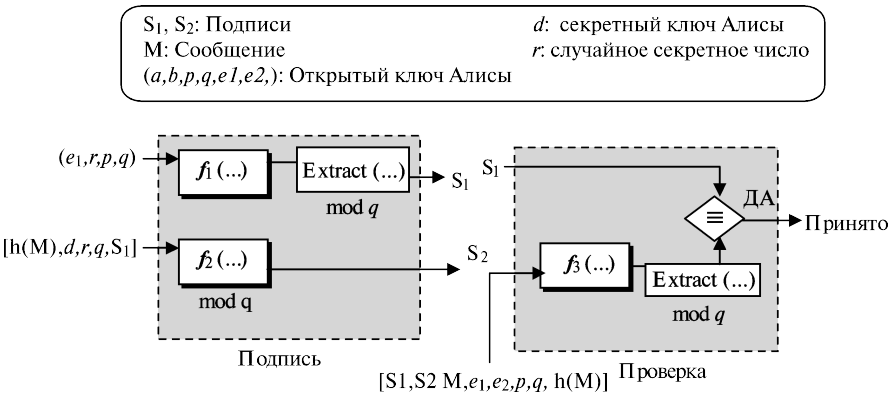


Рис. 13.15. Общая идея схемы DSS

Генерация ключей

Генерация ключей осуществляется следующими шагами.

1. Алиса выбирает эллиптическую кривую E_p (a,b) с простым числом p.
2. Алиса выбирает другое простое число q, чтобы использовать для вычисления.
3. Алиса выбирает секретный ключ d, целое число.
4. Алиса выбирает точку на кривой e_1(.,....)
5. Алиса вычисляет e_2 (..,....) = $d \times e_1$ (.....), другую точку на кривой.
6. Общедоступный ключ Алисы — (a, b, p, q, e_1\, e_2), ее секретный ключ — d.

Подписание и проверка (верификация)

Рисунок 13.16 показывает схему цифровой подписи эллиптической кривой.

Подписание. Процесс подписания состоит главным образом из выбора секретного случайного числа, создания третьей точки на кривой, вычисления двух подписей и передачи сообщения и подписей.

1. Алиса выбирает секретное случайное число, r, между 1 и $q - 1$.
2. Алиса выбирает третью точку на кривой, P (u, v) = $r \times e_1$ (.....).

M: Сообщение
S_1, S_2: Подписи
V: Проверка (верификация)
A,B: Промежуточные результаты
P(u,v), T (x,y): Точка на кривой

r: Случайное число
(d): Секретный ключ Алисы
(d,b,e_1,e_2,p,q): Открытый ключ Алисы
h(...): Алгоритм хеширования

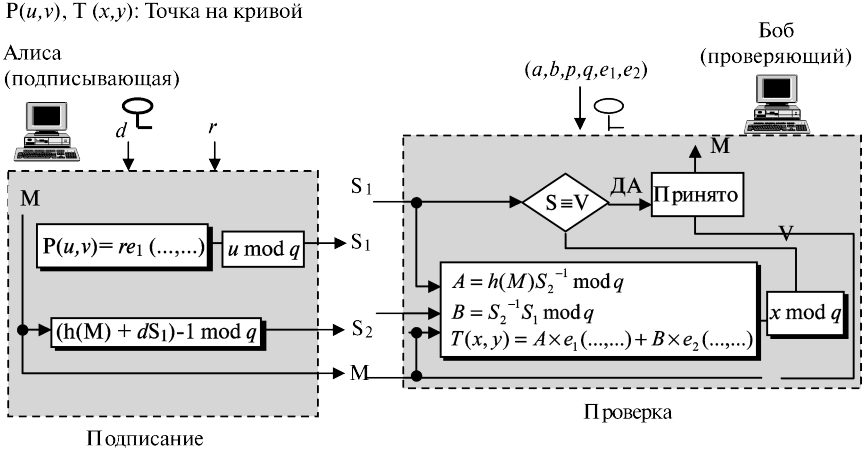


Рис. 13.15. Общая идея схемы ECDSS

3. Алиса использует первые координаты *P (u, v)*, чтобы вычислить первую подпись S_1. Это означает $S_1 = u \bmod q$.
4. Алиса использует дайджест сообщения, свой секретный ключ и секретное случайное число r и S_1, чтобы вычислить вторую подпись $S_2 = (h(M) + d \times S_1)\, r^{-1} \bmod q$,
5. Алиса передает M, S_1 и S_2.

Проверка (верификация). Процесс проверки состоит главным образом из восстановления третьей точки и подтверждения, что первая координата эквивалентна S_1 по модулю q. Обратите внимание, что третья точка была создана подписывающим лицом, использующим секретное случайное число r. Верификатор не имеет этого значения. Ему нужно создать третью точку из дайджеста сообщения, S_1, и S_2.

1. Боб применяет M, S_1 и S_2 для создания двух промежуточных результатов A и B:

$$A = h(M)S_2^{-1} \bmod q \qquad B = S_2^{-1}S_1 \bmod q$$

Затем Боб восстанавливает третью точку $T(x,y) = A \times e_1(...,...) + B \times e_2(...,...)$.

2. Боб использует первую координату из $T(x,y)$, чтобы проверить сообщение. Если $x = S_1 \bmod q$, подпись принимается, иначе — отклоняется.

13.6. Варианты и приложения

В этой секции кратко обсуждаются варианты и приложения для цифровых подписей.

Варианты

Ниже проводится краткое обсуждение некоторых вариантов и дополнений к главному понятию цифровых подписей. Для более четкого понимания читатель может посмотреть специальную литературу.

Подпись с указанием времени

Иногда подписанный документ должен иметь указание времени, чтобы препятствовать действиям противника. Это названо **схемой цифровой подписи с указанием времени.**

Например, Алиса подписывает указание своему банку (условно Бобу) выдать некоторую сумму денег Еве. Документ может быть перехвачен Евой, если на этом документе нет никакого указания времени.

Включение фактической даты и времени на документах может создать проблемы — несинхронизированные часы, отсутствие универсального времени. Одно решение состоит в том, чтобы использовать **nonce (number only once — Nonce) —** одноразовое случайное число, которое может быть задействовано только однажды. Когда приемник получает документ с nonce, он составляет примечание, что число было использовано передатчиком и не может использоваться снова. Другими словами, новый nonce — «настоящее время»; примененный nonce определяет «прошлое время».

Слепые подписи

Иногда мы имеем документ, который хотим подписать, не раскрывая содержание документа подписывающему лицу. Например, ученый хочет сообщить Бобу, что он, возможно, открыл очень важную теорию, которую необходимо заверить у обычного нотариуса. Он просит Алису передать это, не разрешая ей узнать содержание теории. Для этой цели Дэвид Чом (David Caum) разработал некоторые патентованные **слепые схемы цифровой подписи**. Главная идея состоит в следующем.

а. Боб создает сообщение и маскирует (затемняет) его. Боб передает маскированное сообщение Алисе.

б. Алиса подписывает замаскированное сообщение и возвращает подпись на замаскированном сообщении.

в. Боб снимает немаскированную подпись, чтобы получить подпись на первоначальном сообщении.

Слепая подпись, основанная на схеме RSA. Опишем кратко слепую схему цифровой подписи, разработанную Дэвидом Чомом. Маскировка может быть сделана с использованием варианта схемы RSA. Боб выбирает случайное число, *b*, и вычисляет слепое сообщение $B = M \times b^e \bmod n$, в котором *e* является открытым ключом Алисы и *n* — определенным модулем в схеме цифровой подписи RSA. Обратите внимание, что *b* иногда называется «маскирующим коэффициентом», который Боб передает Алисе.

Алиса подписывает маскированное сообщение, используя алгоритм подписания, определенный в RSA. Цифровая подпись – $S_{blind} = B^d \bmod n$, где *d* является секретным ключом Алисы. Обратите внимание, что S_b — подпись на слепой (маскированной) версии сообщения.

Боб просто использует мультипликативную инверсию его случайного числа b, чтобы отделить слепое сообщение от подписи. Подпись — S = $S_b\, b^{-1} \bmod n$. Мы можем доказать, что S — подпись на первоначальном сообщении, как это определено в схеме цифровой подписи RSA:

$$S \equiv S_b b^{-1} \equiv B^d b^{-1} \equiv (M \times b^e)^d b^{-1} \equiv M^d b^{ed} b^{-1} \equiv M^d$$

где S — подпись, если Боб передал первоначальное сообщение, которое будет подписано Алисой.

Предотвращение мошенничества. При слепой подписи появляется опасность, что Боб, может дать Алисе подписать слепое сообщение, которое может навредить ей. Например, сообщение Боба могло быть документом, утверждающим, что Алиса передаст Бобу все свое имущество после ее смерти. Есть по крайней мере три способа предотвратить такой ущерб.

а. Администрация может издать юридический акт, что Алиса не ответственна за подпись любого слепого сообщения, которое нарушает ее интересы.
б. Алиса может запросить документ от Боба, что сообщение, которое она подпишет, не наносит ей ущерба.
в. Алиса могла бы требовать, чтобы Боб доказал свою честность прежде, чем она подпишет ему слепое сообщение

Бесспорные цифровые подписи

Бесспорные схемы цифровой подписи — изящное изобретение Чома и Ван Антверпена (van Antwerpen). Бесспорная схема цифровой подписи имеет три компонента: алгоритм подписания, протокол проверки и протокол отрицания. Алгоритм подписания позволяет Алисе подписывать сообщение. Протокол проверки использует механизм вызова-ответа (рассмотренный в лекции 14), с его помощью Алиса может подтвердить свою подпись. Это предотвращает копирование и распределение подписанного сообщения без одобрения Алисы. Протокол отрицания помогает Алисе отрицать фальшивую подпись. Чтобы потом иметь возможность доказать, что подпись — подделка, Алиса должна принять участие в составлении протокола отрицания.

Приложения

В дальнейшем в этой книге обсуждаются несколько приложений криптографии для сетевой безопасности. Большинство этих приложений непосредственно или косвенно требует использования открытых ключей. Чтобы использовать открытый ключ, человек должен доказать, что он законно владеет этим ключом. По этой причине была разработана идея сертификации и свидетельств сертификации (CA) (см. лекцию 14 и лекцию 15). Свидетельства (CA), чтобы быть признанными, должны быть подписаны. Цифровые подписи используются, чтобы обеспечить такое доказательство. Когда Алиса должна применить открытый ключ Боба, она пользуется свидетельством. Сертифицированные CA используют общедоступный ключ с секретным ключом и проверяемой подписью Алисы. Само свидетельство содержит общедоступный ключ Боба.

Сегодняшние протоколы, которое используют услуги CA, – это IPSec (лекция 18), SSL/TLS (лекция 17) и S/MIME (лекция 16). Протокол PGP поддерживает свидетельства, но они могут быть заменены соглашениями между людьми в сообществе.

13.7. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

[Sti06], [TW06] и [PHS03] подробно рассматривают цифровые подписи.

Сайты

Нижеследующие сайты дают больше информации о темах, обсужденных в этой лекции.

http://www.itl.nist.gcv/npspubs/fip 186.htm
csrc.nist.gov/publiCti ions/tips/tips 186-2/fipsl 86-2-changel
http://en.wikipedia.org /wiki/ElGamal_signature_scheme
csrc.nist.gov/cryptvd/dss/ECDSAVS.pdf
http://en.wikipedia.org/wiki/ElGamal_signature_scheme
http://en.wikipedia.org/wiki/Digital_signature

13.8. Итоги

- Схема цифровой подписи может обеспечить те же самые услуги, что и обычная подпись. Обычная подпись — это часть документа; цифровая подпись — отдельный объект. Чтобы проверять обычную подпись, получатель сравнивает подпись с подписью в архиве; чтобы проверять цифровую подпись, получатель на основе документа проводит процесс подтверждения подписи. Между документом и обычной подписью отношение «один к многим»; Между цифровой подписью и документом отношение «один к одному».
- Цифровые подписи обеспечивают установление подлинности сообщения, а также целостность сообщения, если дайджест сообщения подписан непосредственно вместо сообщения, и исключение отказа от сообщения, если задействовано третье лицо, которому доверяют.
- Цифровая подпись не может обеспечить конфиденциальность сообщения. Если необходима конфиденциальность, кроме схемы цифровой подписи должна быть применена криптографическая система.
- Цифровая подпись нуждается в асимметрично-ключевой системе. В криптографической системе мы используем секретные и открытые ключи приемника; для цифровых подписей мы используем секретные и открытые ключи передатчика.

- Схема цифровой подписи RSA применяет криптографическую систему RSA, но роли секретных и открытых ключей изменяются. Схема цифровой подписи Эль-Гамаля применяет криптосистему Эль-Гамаля (с некоторыми незначительными изменениями), но роли секретных и открытых ключей здесь иные. Схема цифровой подписи Шнорра — модификация схемы Эль-Гамаля, в которой размер подписи может быть меньшим. Стандарт Цифровой подписи (DSS) использует алгоритм цифровой подписи (DSA), который базируется на схеме Эль-Гамаля с некоторыми идеями из схемы Шнорра.
- Схема цифровой подписи с указанием времени разработана для предотвращения повторного использования подписей. Слепые схемы цифровой подписи позволяют Бобу просить Алису подписать документ, не показывая его содержание. Бесспорная схема цифровой подписи нуждается в алгоритме подписания, протоколе проверки и протоколе отрицания для предотвращения копирования и распределения подписанного сообщения без одобрения подписывающего лица.
- Главное приложение цифровых подписей — это Центры сертификации (CA).

13.9. Набор для практики

Обзорные вопросы

1. Сравните и противопоставьте обычную подпись и цифровую подпись.
2. Перечислите службы безопасности, для которых требуется цифровая подпись.
3. Сравните и противопоставьте атаки цифровых подписей с атаками криптографических систем.
4. Сравните и противопоставьте экзистенциальную и селективную подделки.
5. Дайте определение схеме цифровой подписи RSA и сравните ее с криптографической системой RSA.
6. Дайте определение схеме Эль-Гамаля и сравните ее со схемой RSA.
7. Дайте определение схеме Шнорра и сравните ее со схемой Эль-Гамаля.
8. Дайте определение схеме стандарта цифровой подписи (DSS) и сравните ее со схемами Эль-Гамаля и Шнорра.
9. Дайте определение схеме цифровой подписи эллиптической кривой и сравните ее с криптосистемой на основе метода эллиптических кривых.
10. Рассмотрите три варианта цифровых подписей, приведенных в этой лекции, и кратко сформулируйте цель каждой.

Упражнения

1. Используя схему RSA, при $p = 809$, $q = 751$, и $d = 23$, вычислите общедоступный ключ e. Затем:
   а. Подпишите и проверьте сообщение $M_1 = 100$. Получите подпись S_1.
   б. Подпишите и проверьте сообщение с $M_2 = 50$. Получите подпись S_2.
   в. Покажите, что если $M = M_1. \times M_2 = 5000$, то $S = S_1. \times S_2$.

2. Используя схему Эль-Гамаля при $p = 881$ и $d = 700$, найдите значения e_1 и e_2. Выберите $r = 17$. Найдите значение S_1 и S_2, если M = 400.
3. Используя схему Шнорра, при $q = 83$, $p = 997$ и $d = 23$, найдите значения для e_1 и e_2. Выберите $r = 11$. Если M = 400 и h (400) = 100, найдите значение S_1., S_2 и V. Равно ли $S_1 \times V \pmod p$?
4. Используя схему DSS, при $q = 59$, $p = 709$ и $d = 14$, найдите значения для e_1 и e_2. Выберите $r = 13$. Найдите значение S_1 и S_2, если h (M) = 100. Проверьте подпись.
5. Сделайте следующее:
   а. В схеме RSA найдите отношения между размером S и размером n.
   б. В схеме Эль-Гамаля найдите размер S_1 и S_2 в зависимости от размера p.
   в. В схеме Шнорра найдите размер S_1 и S_2 в зависимости от размера p и q.
   г. В DSS-схеме найдите размер S_1 и S_2 в зависимости от размера p и q.
6. NIST-спецификация требует для DSS следующее: если значение $S_2 = 0$, то две подписи должны быть повторно вычислены, используя новый r. Какова причина этого?
7. Что случится, если Ева найдет значение r, используемое подписывающим лицом? Возможно ли это в принципе? Объясните ваш ответ отдельно для схем Эль-Гамаля-Шнорра, DSS.
8. Что случится, если Алиса подпишет два сообщения, используя одно и то же значение n? Объясните ваш ответ отдельно для каждого протокола: Эль-Гамаля, Шнорра или DSS.
9. Покажите пример уязвимости схемы RSA к селективной подделке, когда значения p и q являются маленькими. Используйте $p = 19$ и $q = 3$.
10. Покажите пример уязвимости схемы Эль-Гамаля к селективной подделке, когда значение p мало. Используйте $p = 19$.
11. Покажите пример уязвимости схемы Шнорра к селективной подделке, когда значения p и q являются маленькими. Используйте $p = 29$ и $q = 7$.
12. Покажите пример уязвимости DSS к селективной подделке, когда значения p и q являются маленькими. Используйте $p = 29$ и $q = 7$.
13. В схеме Эль-Гамаля, если Ева может найти значение r, может ли она подделать сообщение? Объясните.
14. В схеме Шнорра, если Ева может найти значение r, может ли она подделать сообщение? Объясните.
15. В DSS-схеме, если Ева может найти значение r, может ли она подделать сообщение? Объясните.
16. Предположим, что значения p q, e_1 и r в схеме Шнорра — те же самые, что и соответствующие значения в DSS-схеме. Сравните значения S_1 и S_2 в схеме Шнорра с соответствующими значениями в DSS-схеме.
17. Объясните, почему в схеме Эль-Гамаля вычисление S_2 делается по модулю p, а вычисление S_2 делается по модулю $p - 1$.
18. Объясните, почему в схеме Шнорра вычисление S_1 делается по модулю p, а вычисление S_2 делается по модулю q.
19. Объясните, почему в схеме DSS вычисление S_1 делается по модулю p и модулю q, а вычисление S_2 делается только по модулю q.
20. В схеме Шнорра докажите правильность процесса проверки.

21. В DSS-схеме докажите правильность процесса проверки.
22. В схеме цифровой подписи эллиптической кривой докажите правильность процесса проверки.
23. Напишите два алгоритма для схемы RSA: один для процесса подписания и один для процесса проверки.
24. Напишите два алгоритма для схемы Эль-Гамаля: один для процесса подписания и один для процесса проверки.
25. Напишите два алгоритма для схемы Шнорра: один для процесса подписания и один для процесса проверки.
26. Напишите два алгоритма для DSS-схемы: один для процесса подписания и один для процесса проверки.
27. Напишите два алгоритма для схемы эллиптической кривой: один для процесса подписания и один для процесса проверки.

Лекция 14. Установление подлинности объекта

Цели и содержание

Эта лекция имеет несколько целей.

- Показать различие между установлением подлинности сообщения и установлением подлинности объекта.
- Определить доказательства, используемые для идентификации.
- Обсудить некоторые методы установления подлинности объекта, например, использование пароля.
- Ввести некоторые протоколы вызова-ответа для установления подлинности объекта.
- Ввести некоторые протоколы подтверждения с нулевым разглашением для установления подлинности объекта.
- Определить биометрию и различающие характеристики между физиологическими и поведенческими методами.

14.1. Введение

Установление подлинности объекта (аутентификация) — методика, которая позволяет одной стороне доказывать подлинность другой стороны. *Объект* может быть человеком, процессом, клиентом или сервером. Объект, подлинность которого должна быть доказана, называется *претендентом;* признаки подлинности (идентификационный код) претендента названы *верификатором.* Когда Боб пробует доказать подлинность Алисы, Алиса — претендент, а Боб — верификатор.

Сопоставление источника данных и установления подлинности объекта

Есть два отличия между *сообщением*, рассмотренным в лекции 13, и *установлением подлинности объекта, (источника данных)*, о котором мы будем говорить в этой лекции.

1. Первое: установление подлинности сообщения (или установление подлинности происхождения данных) не может быть проведено в реальном масштабе времени. Алиса передает сообщение Бобу. Когда Боб проверяет подлинность сообщения, Алиса может участвовать или не участвовать в процессе связи. С другой стороны, когда Алиса запрашивает объект для установления подлинности, соединение для передачи сообщений не устанавливается, пока Алиса не будет опознана Бобом. Алиса должна быть в сети и принять участие в процессе опознавания. Только после того, как она опознана, могут передаваться сообщения между Алисой и Бобом. Установление подлинности источника данных требуется, когда электронную почту передают от Алисы Бобу, равно как и когда Алиса берет наличные из банковского автомата.
2. Второе: установление подлинности сообщения просто подтверждает подлинность одного сообщения. Процесс установления подлинности сооб-

щения может, в случае необходимости, повторяться для каждого нового сообщения. Установление подлинности объекта подтверждает подлинность претендента на всю продолжительность сеанса.

Категории проверки

В установлении подлинности объекта претендент должен идентифицировать себя для верификатора. Этот может быть сделано одним из трех видов свидетелей: *нечто, известное* только претенденту, *нечто, чем обладает* только претендент, или *нечто, свойственное* только претенденту.

Нечто известное. Это — секретная информация известная только претенденту, что может быть проверено верификатором. Примеры: пароль, PIN-код, ключ засекречивания и секретный ключ.

Нечто, чем обладает. Это то, что может доказать опознавательный код претендента. Примеры: паспорт, водительские права, удостоверение личности, кредитная карточка и карточка с интегральной схемой, включающей микропроцессор.

Нечто свойственное. Это свойственные претенденту характеристики. Примеры: обычные подписи, отпечатки пальца, голос, характеристики лица, образец сетчатки глаза и почерк.

Установление подлинности объекта и управление ключами

Эта лекция обсуждает установление подлинности объекта. Следующая лекция рассматривает управление ключами. Эти две темы очень близко связаны; большинство протоколов управления ключами использует протоколы установления подлинности объекта. Именно поэтому в большинстве книг эти две темы обсуждаются вместе. В этой книге для ясности они рассматриваются отдельно.

14.2. Пароли

Самый простой и самый старый метод аутентификации объекта — **аутентификация на основе пароля,** где пароль — «*нечто, что знает*» **претендент**.

Пароль используется, когда пользователь должен обратиться к системе, чтобы задействовать ресурсы системы (вход в систему); пользователь имеет пользовательскую идентификацию, которая открыта, и секретный пароль. Мы разделяем схемы такой аутентификации на две группы: **фиксированный пароль** и **одноразовый пароль.**

Фиксированный пароль

Фиксированный пароль — пароль, который, используется много раз при каждом обращении. Известны несколько схем применения такого пароля.

При элементарном подходе система сохраняет таблицу (файл), в которой отсортированы пользовательские идентификаторы. Чтобы получить допуск к системным ресурсам, в начале сеанса пользователь передает системе свой пользовательский идентификатор и пароль. Система использует идентификацию, чтобы

найти пароль в таблице. Если пароль, передаваемый пользователем, соответствует паролю в таблице, пользователю предоставляют доступ; иначе его заявка отклоняется. Рисунок 14.1 иллюстрирует этот подход.

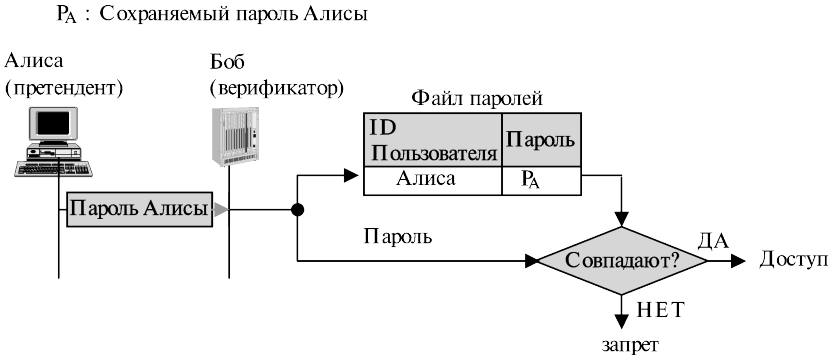


Рис. 14.1. Пользовательский ID и пароль

Атаки

Первый подход. Этот подход содержит несколько видов атак.

Перехват информации. Ева может подсмотреть, как Алиса печатает свой пароль. Большинство систем, в качестве меры безопасности, не показывает символы, которые печатает пользователь. Перехват информации может приобретать более сложную форму. Ева может прослушать линию или перехватить сообщение, таким образом, фиксируя пароль для использования в своих целях.

Захват пароля. Второй тип атаки возникает, когда Ева пробует физически захватить пароль Алисы. Захват может быть предотвращен, если Алиса не записывает пароль, а вместо этого только заучивает его. По этой причине пароль должен быть очень прост и, так или иначе, связан с чем-то знакомым Алисе. Но это делает пароль уязвимым к другим типам атак.

Доступ к файлу пароля. Ева может взломать систему и получить доступ к ID-файлу пароля. Ева может прочитать файл и найти пароль Алисы или даже изменить его. Чтобы предотвратить этот тип атаки, файл может быть защищенным по чтению и записи. Однако большинство систем нуждается в том, чтобы этот тип файла был читаемым. Мы увидим позднее, как второй подход может защитить файл от этого типа атаки.

Подбор. Используя атаку подбора, Ева может зарегистрироваться в системе и попробовать подобрать пароль Алисы, перебирая различные комбинации символов. Пароль особенно уязвим, если пользователю разрешают выбрать короткий пароль (несколько символов). Также уязвима операция, если Алиса выбрала коечто тривиальное, такое, как свой день рождения, имя своего ребенка или имя ее любимого актера. Для того чтобы предотвратить подбор, рекомендуется длинный случайный пароль, что-то не очень очевидное. Однако использование такого случайного пароля может также создать проблему, поскольку Алиса может легко за-

быть такой пароль и обычно возникает желание иметь и хранить где-нибудь копию, что делает пароль уязвимым к захвату.

Второй подход. Более безопасный подход состоит в том, чтобы хранить хэширование пароля (вместо пароля исходного текста) в файле пароля. Любой пользователь может читать содержание файла, но из-за того, что хэш-функция — односторонняя функция, почти невозможно подобрать значение пароля. Рисунок 14.2 показывает ситуацию, когда создается пароль и для его сохранения применяется системное хэширование.

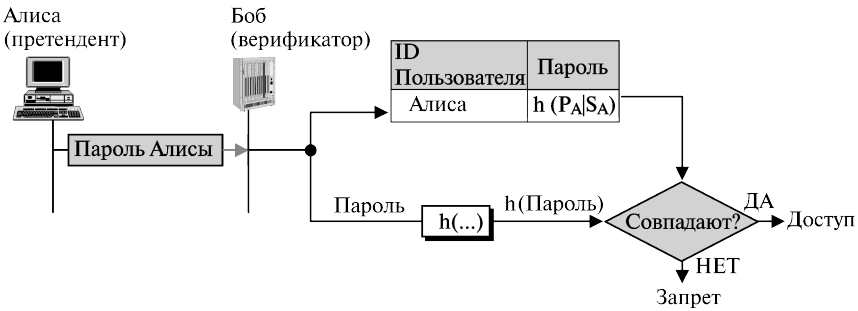


Рис. 14.2. Хеширование пароля

Когда пользователь передает ID и пароль, система выполняет хэширование пароля, а затем сравнивает значение хэшированного пароля со значением, сохраненным в файле. Если они совпадают, пользователю предоставляют доступ; иначе — доступ запрещается. В этом случае файл не должен быть защищен от чтения.

Атака словаря. Хэш-функция препятствует Еве получать доступ к системе даже при том, что Ева имеет файл пароля. Однако есть все еще возможность **атаки словаря.** В этой атаке Ева интересуется нахождением только пароля, независимо от пользовательского ID. Например, если пароль — 6 цифр, Ева может создать список чисел с 6-ю цифрами (000000 к 999999) и затем применить хэш-функцию к каждому числу; результат — список из одного миллиона хэшированных паролей. Она может затем получить файл пароля и провести поиск входов второго столбца, чтобы найти совпадение. Процесс поиска может быть запрограммирован и выполнен автономно на частном компьютере Евы. После того как совпадение найдено, Ева может работать с интересующей ее системой интерактивно и использовать пароль, чтобы обратиться к системе. Третий подход показывает, как затруднить такую атаку.

Третий подход. Третий подход назван **salting** пароля[1]. Когда строка пароля создана, к ней добавляется (с помощью операции «конкатенация») случайная строка, называемая salt. Дополненный пароль и хэшированная информация сохраня-

[1] Salting password – «соление» пароля. В русской литературе по криптографии не встречается соответствующего термина. В большинстве случаев для рассматриваемого процесса защиты пароля применяется термин salting, для применяемого дополнения — salt — *прим. пер.*

ется в файле и хэшируются, а затем снова сохраняются в файле. Теперь, когда пользователь запрашивает доступ, система извлекает salt, конкатенирует (присоединяет) его с полученным паролем, делает хэширование информации результата и сравнивает с хэшированной информацией, сохраненной в файле. Если они совпадают, разрешается доступ к системе; иначе — заявка отклоняется (см. рис. 14.3).

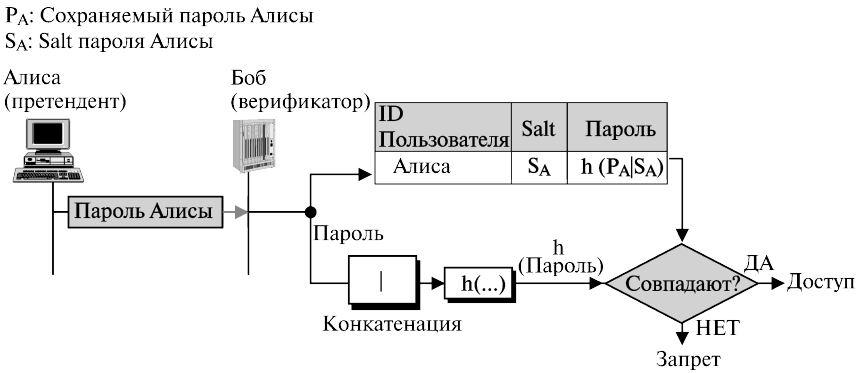


Рис. 14.3. Salt-пароль («соль» пароля)

Рассмотренный выше процесс затруднил атаку словаря. Если первоначальный пароль — 6 цифр, и salt — 4 цифры, то хэширование приводит к значению с 10-ю цифрами. Это означает, что Ева теперь должна сделать список из 10 миллионов комбинаций и провести хэширование для каждой из них. Список хэширования из 10 миллионов входов и сравнение требует намного больше времени, чем в предыдущем случае подбора. Salting очень эффективная процедура, если salt — очень длинное случайное число. Операционная система UNIX использует вариант такого метода.

Четвертый подход. В четвертом подходе два метода идентификации объединены. Хороший пример этого типа установления подлинности — использование карты ATM (платежного автомата) с PIN-кодом (PIN — Personal Identification Number — персональный номер идентификации).

Претендент, имеющий карту, принадлежит к категории *«имеет нечто»*, а PIN-код принадлежит той же категории *«нечто известное»*. PIN — пароль, который увеличивает безопасность карты. Если карта захвачена, она не может использоваться, когда PIN-код не известен. Число PIN-кода, однако, традиционно очень коротко, так что оно легко запоминается владельцем. Но это делает его уязвимым для атаки подбором.

Одноразовый пароль

Одноразовый пароль — это пароль, применяемый единожды. Этот тип пароля делает бесполезным перехват пароля и его дополнение (salting). Здесь рассматриваются три подхода.

Первый подход. При первом подходе пользователь и система согласуют *список паролей*. Каждый пароль в списке может использоваться только единожды.

Есть некоторые недостатки такого подхода, Сначала система и пользователь должны составить и сохранить длинный список паролей. Второе — если пользователь не применяет пароли в заданной последовательности, система должна выполнять длительный поиск для установления соответствия. Однако эта схема делает перехват информации и повторное использование пароля бесполезным. Пароль применим только однажды, и не может применяться снова.

Второй подход. Во втором подходе пользователь и система соглашаются *последовательно обновлять пароль.* Пользователь и система договариваются о первоначальном пароле P_1, действительном только для первого доступа. При первом доступе пользователь генерирует новый пароль, P_2, и зашифровывает этот пароль, используя P_1 как ключ. P_2 — пароль для второго доступа. При втором доступе пользователь генерирует новый пароль, P_3, и зашифровывает его с помощью P_2; P_3 используется для третьего доступа. Другими словами, P_i нужен, чтобы создать P_{i+1}. Конечно, Ева может подобрать первый пароль (P_1), а потом и найти все последующие.

Третий подход. В третьем подходе пользователь и система создают последовательно модифицированный пароль, используя хэш-функцию. В этом подходе, изобретенном Лесли Лампортом (Leslie Lamport), пользователь и система согласуют первоначальный пароль, P_0 и счетчик n. Система вычисляет $h^n(P_0)$, где h^n означает применение хэш-функции n раз. Другими словами,

$$h^n(x) = h(h^{n-1}(x)) \quad h^{n-1}(x) = h(h^{n-2}(x)) \ldots h^2 = h(h(x)) \quad h^1(x) = h(x)$$

Система хранит опознавательный код Алисы, значение n, и значения $h^n(P_0)$. Рисунок 14.4 показывает, как пользователь обращается к системе в первый раз.

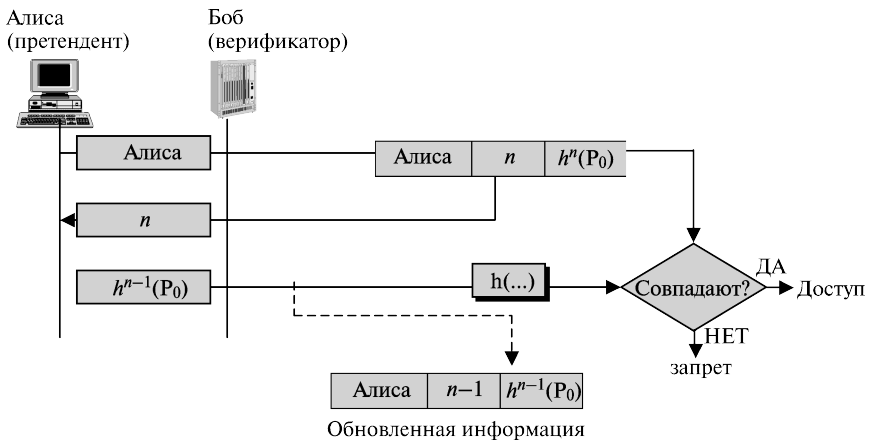


Рис. 14.4. Одноразовый пароль Лампорта

Когда система получает ответ пользователя в третьем сообщении, она применяет хэш-функцию, чтобы сравнить значение с полученными данными и уз-

нать, соответствует ли оно значению хэш-функции и записанному в памяти. Если такое соответствие есть, то предоставляется доступ в систему; иначе заявка отклоняется. Затем система уменьшает значение *n* в памяти и заменяет старое значение пароля $h^n(P_0)$ на новое значение $h^{n-1}(P_0)$.

Когда пользователь обращается к системе второй раз, значение счетчика будет *n* – 1. Третье сообщение от пользователя — теперь $h^{n-2}(P_0)$. Когда система получает это сообщение, она применяет хэш-функцию, чтобы получить $h^{n-1}(P_0)$, которое сравнивается с модифицированным входным сообщением.

Значение *n* во входной информации может быть уменьшено каждый раз, когда есть доступ. Когда значение становится 0, пользователь больше не может обратиться к системе; все должно быть установлено снова. По этой причине значение *n* обычно выбирается достаточно большое, например, такое, как 1000.

14.3. Запрос-ответ

При установлении подлинности пароля претендент доказывает свою идентичность, демонстрируя, что он знает секрет и пароль. Однако из-за того, что претендент сообщает секрет, этот секрет восприимчив к перехвату противником. В **установлении подлинности с помощью запроса-ответа** претендент доказывает, что он *знает* секрет, не посылая его. Другими словами, претендент не передает секрет верификатору; верификатор или имеет его, или находит его.

В установлении подлинности с помощью запроса-ответа претендент доказывает, что знает секрет, не посылая его к верификатору.

Запрос (challenge) — это изменяющееся во времени значение, такое, как случайное число или метка времени, которую передает верификатор. Претендент применяет функцию для преобразования вызова и передает результат, называя его *ответ,* к верификатору. Ответ показывает, что претендент знает секрет.

Запрос — изменяющееся по времени значение, передаваемое верификатором; ответ — результат приложения функции к запросу.

Использование шифра с симметричным ключом

Несколько подходов к установлению подлинности с помощью запроса-ответа используют шифрование с симметричными ключами. Секретность здесь — открытый ключ засекречивания, известный и претенденту и верификатору. Функция — алгоритм шифрования, с помощью которого обрабатывается вызов перед посылкой ответа.

Первый подход. При первом подходе верификатор передает **nonce,** случайное число, используемое только однажды для вызова претендента. **Nonce** должен изменяться; каждый раз создается различное случайное число. Претендент отвечает на вызов, применяя ключ засекречивания, общедоступный претенденту и верификатору. Рисунок 14.5 показывает первый подход.

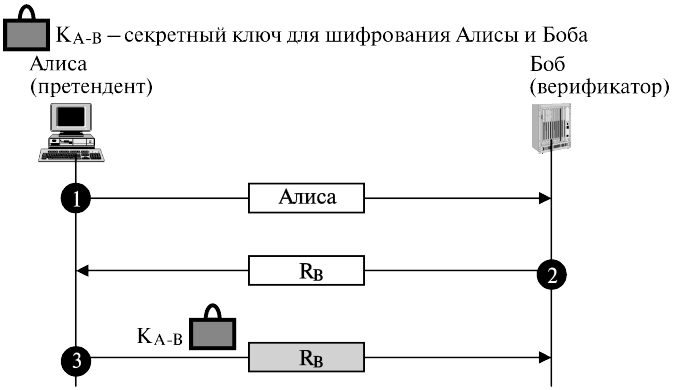


Рис. 14.5. Заявка nonce

Первое сообщение не является частью запроса-ответа — это только сообщение верификатору, что претендент хочет вызвать систему. Второе сообщение — запрос. R_B – это **nonce**, случайно выбранное верификатором (Боб) число-вызов претенденту. Претендент шифрует nonce, используя общедоступный ключ засекречивания, известный только претенденту и верификатору. Он передает результат верификатору. Верификатор расшифровывает сообщение. Если при дешифровании получен тот же самый nonce, как и переданный верификатором, Алисе предоставляется доступ.

Обратите внимание, что в этом процессе претендент и верификатор должны хранить симметричный ключ, используемый в процессе засекречивания. Верификатор должен также сохранить значение nonce до идентификации претендента, пока не будет возвращен ответ.

Читатель, возможно, заметил, что использование nonce предотвращает ответ на третье сообщение Евы. Ева не может ответить на третье сообщение и притвориться, что это новый запрос об установлении подлинности Алисы, потому что как только Боб получает ответ (значение R_B), этот ответ больше не может быть задействован. В следующий раз используется новое значение.

Второй подход. При втором подходе применяется признак, который меняется во времени. Это — метка времени, которая очевидно изменяется со временем. В этом подходе сообщение вызова — текущее время, передаваемое от верификатора к претенденту. Предполагается как необходимое, что часы клиента и сервера синхронизированы и претендент знает текущее время. Это означает, что нет необходимости в сообщении вызова. Первое и третье сообщения могут быть объединены. В результате установление подлинности может быть выполнено с использованием одного сообщения, ответа на неявный вызов — текущее время. Рисунок 14.6 показывает этот подход.

Третий подход. Первый и второй подходы созданы для однонаправленного установления подлинности. Подлинность Алисы устанавливается Бобом, но не устанавливается обратное. Если Алисе надо быть уверенной в подлинности Боба, то нужно иметь процесс двунаправленного установлении подлинности. Рисунок 14.7 показывает такую схему.

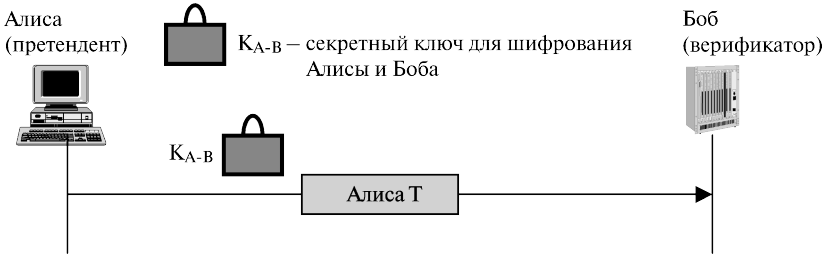


Рис. 14.6. Запрос с меткой времени

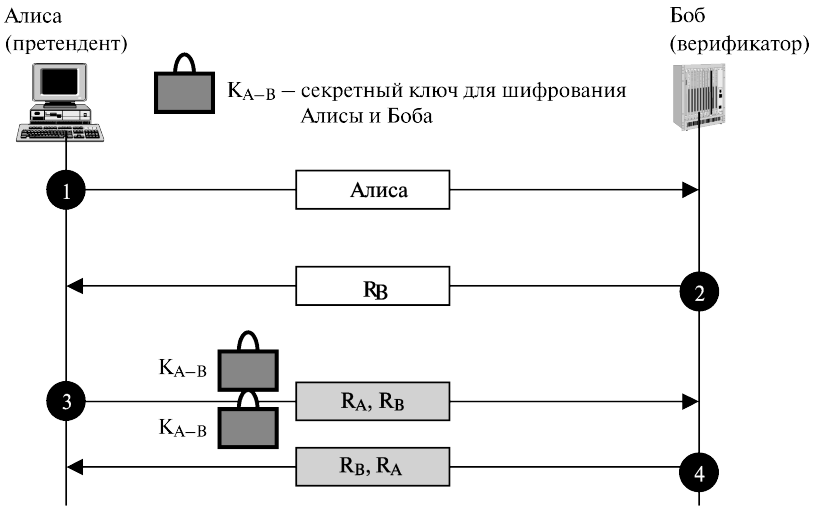


Рис. 14.7. Двунаправленное установление подлинности

Второе сообщение R_B — вызов от Боба Алисе. В третьем сообщении Алиса отвечает на вызов Боба и в то же самое время передает свой вызов R_A Бобу. Третье сообщение — ответ Боба. Обратите внимание, что в четвертом сообщении порядок R_A и R_B изменяется, чтобы предотвратить атаку ответа третьего сообщения противником.

Использование функций ключевого хэширования

Вместо того чтобы использовать шифрование/дешифрование для установления подлинности объекта, мы можем также применять ключевую хэш-функцию (MAC). Эта схема имеет одно преимущество: она сохраняет целостность сообщений вызова и ответа и в то же самое время использует секрет — ключ.

Рисунок 14.8 показывает, как мы можем использовать ключевую хэш-функцию, чтобы создать ответ вызову с меткой времени.

Обратите внимание, что в этом случае метку времени передают и как исходный текст, и как текст, скрэмблированный ключевой хэш-функцией. Когда Боб получает сообщение, он берет исходный текст T, применяет ключевую хэш-функцию и затем, чтобы определить подлинность Алисы, сравнивает свои вычисления с тем, что он получил.

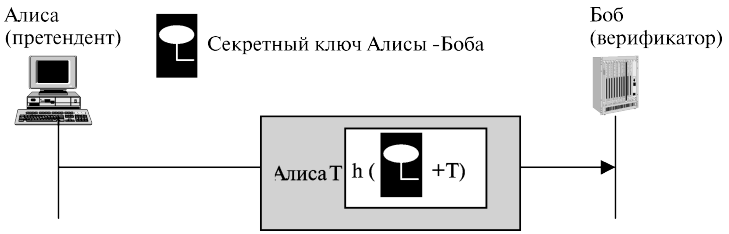


Рис. 14.8. Функция ключевого хэширования

Шифр, использующий асимметричный ключ

Вместо шифра с симметричными ключами мы можем применить для установления подлинности объекта шифр с асимметричными ключами. Здесь секрет — это секретный ключ претендента. Претендент должен показать, что он имеет секретный ключ, связанный с открытым ключом, который доступен каждому. Верификатор должен зашифровать вызов, используя открытый ключ претендента; претендент затем расшифровывает сообщение, используя свой секретный ключ. Этот ответ на запрос — расшифрованный запрос. Далее рассматриваются два подхода: один для однонаправленного установления подлинности и один — для двунаправленного.

Первый подход. В первом подходе Боб зашифровал признак, используя открытый ключ Алисы. Алиса расшифровывает сообщение своим секретным ключом и передает nonce Бобу. Рисунок 14.9 иллюстрирует такой подход.

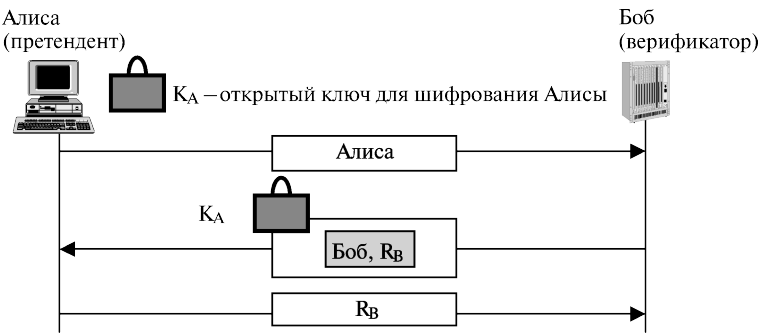


Рис. 14.9. Однонаправленная система проверки подлинности с асимметричным ключом

Второй подход. Во втором подходе используются два открытых ключа, один в каждом направлении. Алиса передает опознавательный код и nonce, зашифрованные открытым ключом Боба. Боб отвечает своим nonce, расшифрованным открытым ключом Алисы. В конечном итоге Алиса отвечает расшифрованным nonce Боба. Рисунок 14.10 иллюстрирует этот подход.

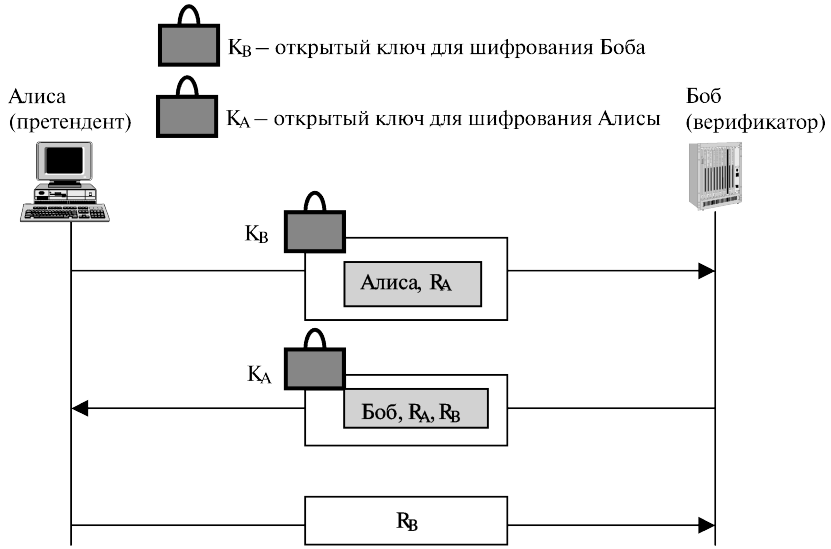


Рис. 14.10. Двунаправленная система проверки подлинности с асимметричным ключом

Использование цифровой подписи

Установление подлинности объекта может также быть достигнуто с помощью цифровой подписи. Когда цифровая подпись применяется для установления подлинности объекта, претендент использует для подписания свой секретный ключ. Здесь показаны два подхода, другие оставляем как упражнения.

Первый подход. При первом подходе, показанном на рис. 14.11, Боб использует исходный текст вызова, а Алиса подписывает ответ.

Второй подход. При втором подходе, показанном на рис. 14.12, Алиса и Боб устанавливают подлинность друг друга.

14.4. Подтверждение с нулевым разглашением

При установлении подлинности пароля претендент должен передать свой секрет (пароль) верификатору; это может привести к перехвату информации Евой. Кроме того, нечестный верификатор может показать пароль другим или использовать его, чтобы исполнить роль претендента.

Рис. 14.11. Одностороннее установление подлинности с помощью цифровой подписи

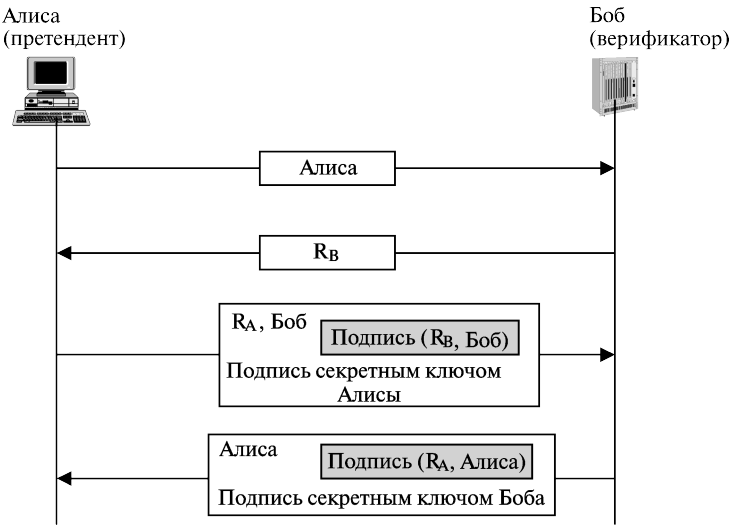


Рис. 14.12. Двустороннее установление подлинности с помощью цифровой подписи

При установлении подлинности объекта методом вызова-ответа секрет претендента не передают верификатору. Претендент применяет некоторую функцию для обработки запроса, которая передана верификатором, но при этом включает свой секрет. В некоторых методах «запроса-ответа» верификатор фактически знает секрет претендента, при этом он может неправильно использоваться нечестной ве-

рификацией. В других методах верификатор может извлечь некоторую информацию о секрете претендента, выбирая заранее запланированное множество запросов.

В **установлении подлинности с нулевым разглашением** претендент не раскрывает ничего, что могло бы создать угрозу конфиденциальности секрета. Претендент доказывает верификатору, что он знает секрет, не раскрывая и не показывая его. В таком случае взаимодействие разработано так, чтобы не привести к раскрытию или предположению о содержании секрета. После обмена сообщениями верификатор только знает, что претендент имеет или не имеет секрет — и ничего больше. В этой ситуации результат — да/нет. Это единственный бит информации.

В установлении подлинности с нулевым разглашением претендент доказывает, что он знает секрет, не показывая его.

Протокол Фиата-Шамира

В **протоколе Фиата-Шамира (Amos Fiat, Adi Shamir)** третье лицо, которому доверяют (см. лекцию 15), выбирает два больших простых числа p и q, чтобы вычислить значение $n = p \times q$. Значение n объявляется общедоступным. Значения n и q сохраняются секретными. Алиса, претендент, выбирает секретное число s между 1 и $n - 1$. Она вычисляет $v = s^2 \bmod n$. Она сохраняет s как свой секретный ключ и регистрирует v как свой общедоступный ключ вместе с третьим лицом. Проверка Алисы Бобом может быть сделана в четыре шага, как показано на рис. 14.13.

1. Алиса-претендент выбирает случайное число r между 0, и $n - 1$ (r называется «обязательство»). Она затем вычисляет значение $x = r^2 \bmod n$ (x называется «свидетельство»).
2. Алиса передает x Бобу как свидетельство.
3. Боб-верификатор передает вызов c Алисе. Значение c равно или 0, или 1.
4. Алиса вычисляет свой ответ $y = rs^c$. Обратите внимание, что r — случайное число, выбранное Алисой на первом шаге, s — ее секретный ключ и c — признак (0 или 1).
5. Алиса передает ответ Бобу, чтобы показать, что она знает значение своего секретного ключа, s. Она подтверждает, что передает именно Алиса.
6. Боб вычисляет y^2 и xv^c. Если эти два значения являются конгруэнтными, то для Алисы значение s означает «она честна»; или она вычислила значение y другим способом («она нечестная»), потому что мы можем легко доказать, что y — тот же самый, как xv^c по модулю n:

$$y^2 = (rs^c)^2 = r^2 s^{2c} = r^2 (s^2)^c = xv^c$$

Раунд состоит из шести шагов; верификация повторяется несколько раз со значением c, равным 0 или 1 (выбираются случайно). В каждом раунде, который будет верифицирован, претендент должен передать результат испытания. Если не проходит хотя бы один раунд, процесс прерывается и формируется сообщение, что Алиса не прошла испытание на подлинность.

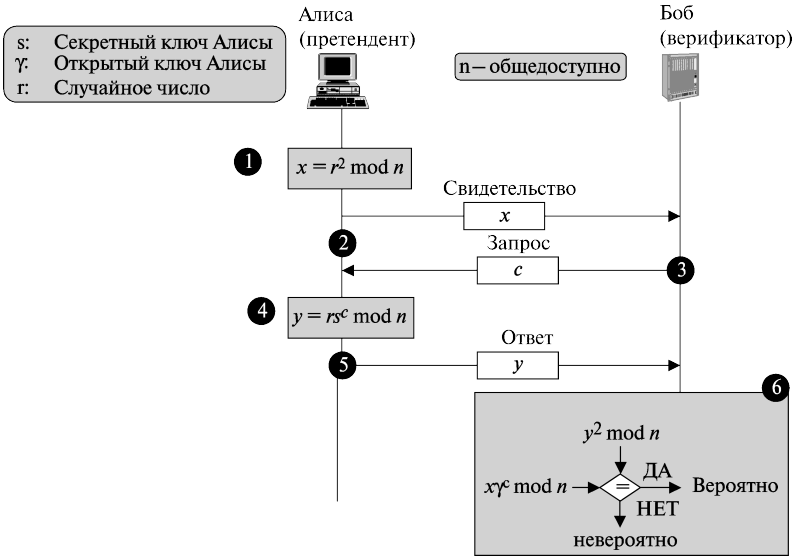


Рис. 14.13 Протокол Фиата-Шамира

Давайте рассмотрим этот тщательно продуманный и интересный протокол. Алиса может быть честна (знает значение *s*) или нечестна (не знает значение *s*). Если она честна, она проходит каждый раунд. Если нет — она может пройти раунд, правильно предсказывая значение вызова. При этом могут возникнуть две ситуации.

1. Алиса предполагает, что значение *c* (выход) будет 1 (предсказание). Она вычисляет $x = r^2/\nu$ и передает *x* как свидетельство.
   а. Если ее предположение правильно (оказалось, что *c* был равен 1), она передает $y = r$ как ответ. Мы можем видеть, что она передаст результат, который соответствует тесту $(y^2 = x\nu^c)$.
   б. Если ее предположение неправильно (оказалось, что *c*, было 0), она не может найти значение *y*, которое соответствует тесту. Она, вероятно, выходит из игры или передаст значение, которое не соответствует ожидаемому результату теста, и Боб прервет процесс.
2. Алиса предполагает, что значение *c* (вызов) будет 0. Она вычисляет $x = r$ и передает *x* как свидетельство.
   в. Если ее предположение правильно (оказалось, что *c* было 0), она передает $y = r$ как ответ. Она передает результат испытания, который соответствует $(y^2 = x\nu^c)$.
   г. Если ее предположение неправильно (оказалось, что *c* было 1), она не может найти значение *y*, которое соответствует тесту. Она, вероятно, выходит из игры или передаст значение, которое не соответствует ожидаемому результату теста, и Боб прервет процесс.

Мы можем видеть, что нечестный претендент имеет 50-процентный шанс на введение в заблуждение верификатора, проводящего испытание (предсказывая

значение вызова). Другими словами, Боб назначает вероятность 1/2 для каждого раунда испытания. Если процесс повторяется 20 раз, вероятность уменьшается до $(1/2)^{20}$ или $9{,}54 \times 10^{-7}$. Другими словами, просто невероятно, что Алиса может правильно предсказать 20 раз.

Пример «пещера Аладдина». Чтобы показать логику вышеупомянутого протокола, Жан-Жак Кискатер (Quisquater) и Гиом Гийу (Gillou) изобрели пример «пещера Аладдина» (рис. 14.14).

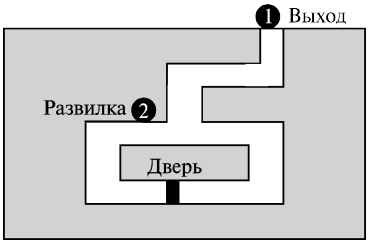


Рис. 14.14. Пещера

Предположим, что есть подземная пещера с дверью в конце, которая может быть открыта только с помощью волшебного слова. Алиса утверждает, что она знает это слово и что она может открыть дверь. Вначале Алиса и Боб стоят у входа (точка 1). Алиса входит в пещеру и достигает разветвления (точка 2). Боб, стоя у входа, не может видеть Алису. Теперь начинается игра.

1. Алиса выбирает, куда идти: или направо, или налево. И говорит об этом Бобу (соответствует передаче свидетельства *x*).
2. После того как Алиса исчезает в пещере, Боб подходит к разветвлению (точка 2) и просит, чтобы Алиса вышла или справа, или слева. Это соответствует передаче вызова (c).
3. Если Алиса знает волшебное слово (свой секретный ключ), она может выйти с запрошенной стороны. Ей, вероятно, придется использовать волшебное слово (если она находится на неправильной стороне), или она может выйти, не используя волшебное слово (если она — на правильной стороне). Однако если Алиса не знает волшебное слово, она может выйти только с правильной стороны, если она разгадала вызов Боба. С вероятностью 1/2 Алиса может убедить глупого Боба, что она знает волшебное слово. Это соответствует ответу *(y)*.
4. Игра повторяется много раз. Алиса победит, если она все время проходит испытания положительно. Вероятность, что она победит в игре, если она не знает волшебное слово, очень низка. Другими словами, $P = (1/2)^N$, где P — вероятность победы, если она не знает волшебное слово. *N* — количество повторений испытания.

Протокол Фейге-Фиата-Шамира

Протокол Фейге-Фиата-Шамира (Feige-Fiat-Shamir) подобен первому подходу за исключением того, что он использует вектор секретных ключей $[s_1, s_2 \ldots s_k]$,

вектор общедоступных ключей ($v_1,v_2,...,v_k$) и векторы признаков ($c_1,c_2,...,c_k$). Секретные ключи выбраны случайно, но они должны быть взаимно простыми с *n*. Общедоступные ключи выбраны так, что $v_i = (s_i^2)^{-1} \bmod n$. На рисунке 14.15 показаны три шага в процессе.

Мы можем доказать, что $y^2v_1^{c_1}v_2^{c_2}...v_k^{c_k}$ имеет то же значение, что и *x:*

$$y^2\gamma_1^{c_1},\gamma^{c_2}{}_2,...,\gamma_k^{c_k} = r^2(s_1^{c_1})^2(s_2^{c_2})^2....(s_k^{c_k})^2\gamma_1^{c_1},\gamma^{c_2}{}_2,...,\gamma_k^{c_k}$$
$$= x(s_1^2)^{c_1}(\gamma_1^{c_1})(s_2^2)^{c_2}(\gamma_2^{c_2})...(s_k^2)^{c_k}(\gamma_k^{c_k})$$
$$= x(s_1^2\gamma_1)^{c_1}(s_1^2\gamma_2)^{c_1}...(s_k^2\gamma_k)^{c_k} = x(1)^{c_1}(1)^{c_2}....(1)^{c_k} = x$$

Три шага составляют раунд; проверка повторяется несколько раз со значением индекса s, равным 0 или 1 (выбирается случайно). Претендент должен провести испытание в каждом раунде, который проверяется. Если претендент ошибается в одном раунде, процесс прерывается и подлинность не подтверждается.

Протокол Кискатера-Гийу

Протокол Кискатера (Quisquater) и Гийу (Gillou) расширяет протокол Фиата-Шамира, в котором может быть использовано меньшее число раундов, чтобы доказать полномочность претендента. Третье лицо, которому доверяют (см. лекцию 15), выбирает два больших простых числа *p* и *q*, чтобы вычислить значение $n = p \times q$.

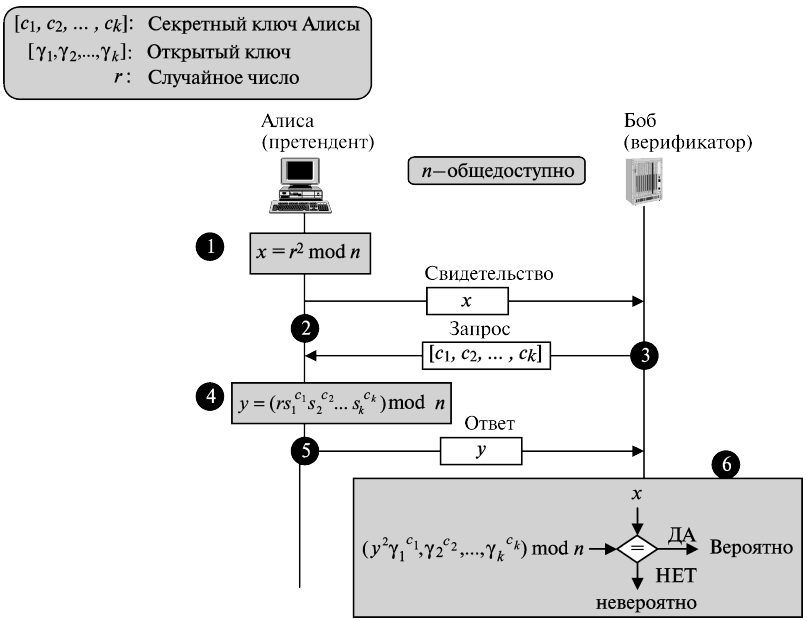


Рис. 14.15. Протокол Фейге-Фиата-Шамира

Сторона, которой доверяют, также выбирает показатель, *e,* который является взаимно-простым с ϕ, где ϕ = (*p* − 1) (*q* − 1). Значения *n* и *e* объявляются общедоступными; значения *p* и *q* сохраняются секретными. Сторона, которой доверяют, выбирает два числа для каждого объекта: число ν, которое является общедоступным, и число *s*, которое является секретным. Однако в этом случае отношения между ν и *s* определяются уравнением $s^e \times \nu = 1 \bmod n$

Три шага составляют раунд; проверка повторяется несколько раз со случайным значением ***c*** (вызов — challenge) между 1 и *e*. Претендент должен передать результат испытания в каждом раунде, который будет проверяться. Если претендент передал неверно хотя бы один раунд, процесс прерывается и сообщение признается неподлинным. Рисунок 14.16 иллюстрирует один раунд.

Равенство может быть доказано так, как показано ниже:

$$y^e \times \gamma^c = (r \times s^c)^e \times \gamma = r^e \times s^{ce} \times \gamma^c = r^e \times (s^s \times \gamma)^c = x \times 1^c = x$$

14.5. Биометрия

Биометрия — измерение физиологических или поведенческих особенностей, которые идентифицируют человека (установление подлинности чего-то, свойственного этому человеку). Работа с показателями биометрии подтверждает, что они не могут быть подделаны, украдены или использованы одновременно с обладателем.

Компоненты

Для биометрии необходимы несколько компонентов, включая устройства фиксации, процессоры и устройства хранения. Устройства фиксации, такие, как читающие устройства (или датчики), измеряют биометрические характеристики. Процессоры преобразуют измеренные данные и заменяют их данными определенного типа, для соответствующей экономии ресурсов компьютера. Устройства хранения сохраняют результат обработки для использования при установлении подлинности.

Регистрация

Перед использованием любых биометрических методов для установления подлинности соответствующие характеристики каждого человека в коллективе, который может быть проверен на подлинность, должны быть доступны в базе данных. Это называется «регистрация».

Установление подлинности

Установление подлинности проводится с помощью верификации или идентификации.

Верификация

При **верификации** характеристики человека отыскиваются для единственной записи в базе данных (соответствие «один к одному»), чтобы найти, является ли он тем, кем сам себя заявляет. Это полезно, например, когда банк должен проверить подлинность клиента, поставившего подпись на чеке.

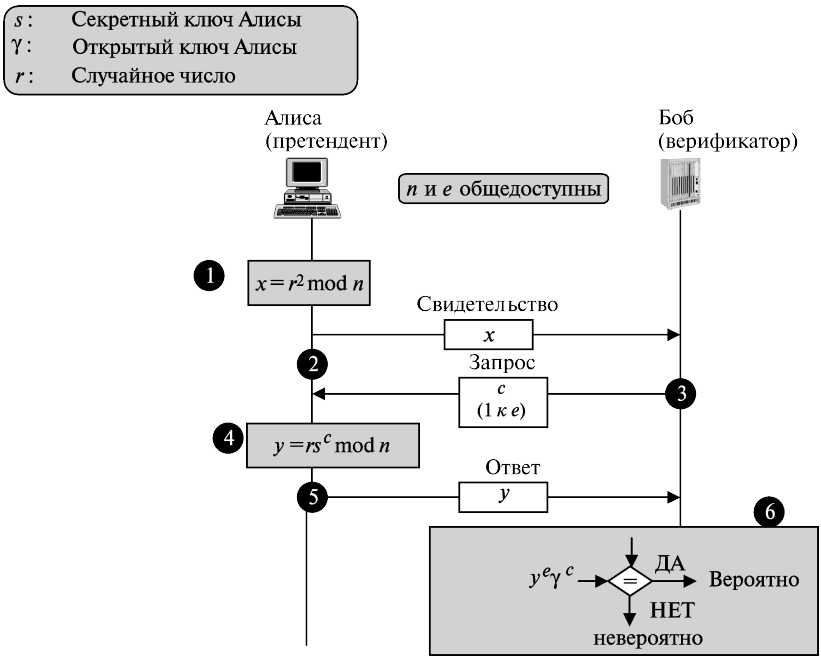


Рис. 14.16. Протокол Кискатера-Гию

Идентификация

При **идентификации** характеристики человека отыскиваются среди всех записей в базе данных (соответствие «один ко многим»), чтобы найти, записан ли он в базе данных. Это полезно, например, когда компания хочет разрешить доступ в здание только своим служащим.

Методы

Методы биометрии могут быть разделены на две широких категории: физиологический и поведенческий. Рис. 14.17 показывает несколько общих методов, входящих в каждую категорию.

Физиологические методы

Физиологические методы измеряют физические характеристики человеческого тела для верификации и идентификации. Чтобы быть эффективными, эти ха-

Рис. 14.17. Биометрика

рактеристики должны быть уникальны среди всех людей или большинства людей. Кроме того, характеристика должна учитывать изменения из-за старения, операции, самочувствия, болезни и так далее. Есть несколько физиологических методов.

Отпечатки пальца. Хотя есть несколько методов для того, чтобы измерить характеристики, связанные с отпечатками пальца, два из них самые общие — *на основе деталей* и *на основе изображения*. Методика на основе деталей — это система создает граф, основанный на расположении отдельных типов узоров (полей папиллярных линий). Граф фиксирует начало и конец отдельных ветвей. В методике на основе изображения система создает изображение кончика пальца и находит подобные изображения в базе данных. Отпечатки пальца использовались в течение долгого времени. Они демонстрируют высокий уровень точности и поддерживают верификацию и идентификацию. Однако отпечатки пальца могут меняться при старении, получении ранения или болезни.

Радужная оболочка глаза. Эта методика снимает образец радужной оболочки глаза, которая уникальна у каждого человека. Обычно требуется применение лазерного (инфракрасного) луча. Это очень точная характеристика, которая сохраняется неизменной в течение всей жизни человека. Она также обеспечивает верификацию и идентификацию. Однако, некоторые болезни глаза, такие, как катаракта, могут изменить вид радужной оболочки.

Сетчатка. Устройства для этой цели исследуют кровеносные сосуды — глазное дно. Однако эти устройства дороги и пока не очень распространены.

Лицо. Существует методика, которая анализирует геометрию лица, основанную на расстоянии между его составляющими, такими, как нос, рот и глаза. Некоторые технологии комбинируют геометрические данные и данные о текстуре кожи. Стандартные видеокамеры и эта методика поддерживают и верификацию, и идентификацию. Однако точность этого метода можно нарушить очками, бородой, усами и влиянием старения.

Руки. Эта методика измеряет характеристик рук, включая форму и длину пальцев. Она может использоваться в закрытом помещении и на открытом воздухе. Однако этот метод лучше подходит для верификации, а не для идентификации.

Голос. Речевое распознавание измеряет основной фон, интонацию и акустический сигнал в голосе. Оно может использоваться вблизи (микрофон) или на расстоянии (аудиоканал). Этот метод главным образом применяется для проверки. Однако точность может быть уменьшена из-за фоновых шумов, болезни или возраста.

ДНК (дезоксирибонуклеиновая кислота, DNA — deoxyribonucleic acid) химически найдена в ядре всех человеческих клеток и в большинстве других организмов. Взятый один раз образец постоянен в течение всей жизни и даже после смерти. Это чрезвычайно точный метод. Он может использоваться и для верификации, и для идентификации. Единственная проблема: идентичные близнецы могут иметь одинаковую ДНК.

Поведенческие методы

Поведенческие методы измеряют некоторые черты человеческого поведения. В отличие от физиологических методов, поведенческие методы должны быть предметом постоянного наблюдения, чтобы гарантировать, что претендент ведет себя нормально, а не пытается исполнять роль кого-то другого.

Подпись. В прошлом подписи использовались в банковском деле, чтобы проверить подлинность клиента, подписавшего чек. Сегодня все еще существует много экспертов, которые могут определить, является ли подпись на чеке или документе той же самой, что и подпись в архиве. Биометрический метод пытается приблизиться к ним. Используются планшеты для анализа подписи и специальные биометрические ручки, чтобы идентифицировать человека. Эти устройства не только сравнивают конечный продукт — подпись, они также измеряют некоторые другие поведенческие черты, такие как время подписания документа и манеру ведения письма во время подписи. Биометрический анализ подписи главным образом используется для верификации.

Нажатие клавиши (ритм печати) — это методика измеряет поведение человека, связанное с работой на клавиатуре. Она может измерить продолжительность нажатия клавиш, время между нажатиями клавиш, число и частоту ошибок, силу давления на клавиши, и так далее. Она — не дорогая, потому что не требует нового оборудования, однако не очень точная, потому что эти черты могут изменяться со временем (люди становятся более быстрыми или более медленными «машинистками»). Точность также зависит от сложности текста.

Точность

Точность биометрических методов измеряется с помощью двух параметров: **коэффициент ложной тревоги (FRR — False Rejection Rate)** и **коэффициент ложной идентификации (FAR — False Acceptance Rate).**

Коэффициент ложной тревоги (FRR)

Он измеряет параметр, который показывает, как часто человек, который должен быть распознан, бывает не распознан системой. FRR — это отношение

ложного отклонения подлинного клиента к общему количеству попыток (в процентах).

Коэффициент ложной идентификации (FAR)

Она измеряет параметр, который показывает, как часто человек, который **не** должен быть распознан, бывает распознан системой. FAR — это отношение ложного установления подлинности клиента к общему количеству попыток (в процентах).

Приложения

Несколько приложений биометрии активно используются. В коммерческих организациях они применяются для разрешения доступа к оборудованию, обращению к информационным системам, в местах продажи, при наблюдении за использованием времени служащего. В юридической системе они применяются в ходе следствия (использование отпечатков пальцев или ДНК) и в судебном анализе. Таможенное управление и управление иммиграции также используют некоторые биометрические методы.

14.6. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Установление подлинности объекта рассматривается в [Sti06], [TW06], [Sal03] и [KPS02].

Сайты

Нижеследующие сайты дают больше информации о темах, обсужденных в этой лекции.

http://en.wikipedia.org/wiki/Challenge-response_authentication
http://en.wikipedia.org/wiki/Password-authenticated_key_agreement
http://rfc.net/rfc2195.html

14.7. Итоги

- Установление подлинности объекта позволяет одной стороне доказывать свою идентичность другой стороне. При установлении подлинности объекта претендент доказывает свою идентичность верификатору, используя один из трех видов свидетельств: «нечто известное», «обладающий чем-то» или «нечто свойственное».
- При установлении подлинности на основе пароля претендент использует строку символов как *нечто известное*. Установление подлинности на осно-

ве пароля может быть разделено на две обширных категории: фиксированный и одноразовый пароли. Атаки установления подлинности на основе пароля включают перехват информации, захват пароля, доступ файла пароля, подбор и атаки словаря.

- При установлении подлинности с помощью вызова-ответа претендент доказывает, что он знает секрет, фактически не открывая его. Установление подлинности с помощью вызова-ответа может использовать шифры с симметричным ключом, функции ключевого хэширования, шифры с асимметричными ключами и цифровые подписи.
- При установлении подлинности с нулевым разглашением претендент не показывает свой секрет; он только доказывает, что знает его.
- Биометрия — измерение физиологических или поведенческих характеристик, чтобы опознать человека по чему-либо свойственному ему. Мы можем разделить биометрические методы на две обширные категории: физиологические и поведенческие. Физиологические методы измеряют физические черты человеческого тела для верификации и идентификации. Поведенческие методы измеряют некоторые черты человеческого поведения.

14.8. Набор для практики

Обзорные вопросы

1. Покажите отличие между установлением подлинности источника данных и установлением подлинности объекта.
2. Перечислите и дайте определение трем типам свидетельств идентификации при установлении подлинности объекта.
3. Покажите отличие между фиксированными и одноразовыми паролями.
4. Каковы преимущества и недостатки использования длинных паролей?
5. Объясните общую идею установления подлинности объекта с помощью вызова-ответа.
6. Дайте определение nonce и расскажите о его использовании в установлении подлинности объекта.
7. Дайте определение атаки словаря и расскажите, как она может быть предотвращена.
8. Покажите отличие между установлениями подлинности объекта вызовом-ответом и подтверждениями с нулевым разглашением.
9. Дайте определение биометрии и покажите отличие между двумя категориями методов.
10. Покажите отличие между двумя параметрами точности, определенными для биометрического измерения в этой лекции.

Упражнения

1. Мы обсуждали фиксированные и одноразовые пароли как два крайних случая. Что вы можете сказать о часто изменяемых паролях? Как вы ду-

маете, может ли эта схема быть реализована в действительности? Каковы преимущества и недостатки такого метода?

2. Как система может предотвратить атаку подбора пароля? Как банк может предотвратить использование PIN в случае, если кто-то нашел или украл банковскую кредитную карточку и пробует использовать ее?
3. Покажите еще два действия процедуры установления подлинности по рис. 14.4.
4. Каковы недостатки использования метки времени по рис. 14.6?
5. Можно ли повторить три сообщения по рис. 14.5, чтобы достигнуть двунаправленного установления подлинности? Объясните.
6. Покажите, как на рис. 14.5 может быть проведено установление подлинности с помощью ключевой хэш-функции.
7. Покажите, как на рис. 14.7 может быть проведено установление подлинности с помощью ключевой хэш-функции.
8. Сравните рис. 14.5 и рис. 14.9 и составьте список совпадений и различий.
9. Сравните рис. 14.7 и рис. 14.10 и составьте список совпадений и различий.
10. Можно ли использовать метку времени с шифром с асимметричными ключами для установления подлинности? Объясните.
11. Сравните и противопоставьте рис. 14.13, рис. 14.15 и рис. 14.16. Составьте список совпадений и различий.
12. Проделайте заново пример «пещеры» для протокола Фейге-Фиата-Шамира.
13. Для $p = 569$, $q = 683$ и $s = 157$ показать три раунда протокола Фиата-Шамира, вычисляя значения и заполняя таблицы.
14. Для $p = 683$, $q = 811$, $s_1 = 157$ и $s_2 = 43215$ покажите три раунда протокола Фейге-Фиата-Шамира, вычисляя значения и заполняя таблицы.
15. Для $p = 683$, $q = 811$ и $v = 157$ покажите три раунда протокола Кискатера-Гийу, вычисляя значения и заполняя таблицы.
16. Нарисуйте рисунок-диаграмму, чтобы показать общую идею трех протоколов для установления подлинности с нулевым разглашением, которые мы обсуждали в этой лекции.
17. В протоколе Фиата-Шамира — какова вероятность, что нечестный претендент правильно ответит на вызов 15 раз?
18. В протоколе Фейге-Фиата-Шамира — какова вероятность, что нечестный претендент правильно ответит на вызов 15 раз?
19. В протоколе Кискатера-Гийу — какова вероятность, что нечестный претендент правильно ответит на вызов 15 раз, если значение свидетельства выбрано между 1 и 15?
20. В двунаправленном подходе к установлению подлинности на рис. 14.10, если позволяются множественные сеансы установления подлинности, Ева перехватывает R_B-nonce от Боба (во втором сеансе) и передает его как nonce Алисы для второго сеанса. Боб, не проверяя, что этот nonce — тот же, как тот nonce, который он передал, зашифровал R_B и вставляет его в сообщение с nonce. Ева использует зашифрованный R_B и притворяется, что она — Алиса, продолжающая первый сеанс и отвечающая зашифрованным R_B. Это — атака отражения. Покажите шаги в этом сценарии.

Лекция 15. Управление ключами

Цели и содержание

Эта лекция имеет несколько целей.

- Объяснить потребность в центре распределения ключей (KDC — Key Distribution Center).
- Показать, как KDC может создать ключ сеанса между двумя сторонами.
- Показать, как две стороны могут использовать протокол соглашения с симметричными ключами для создания между собой ключа сеанса, не прибегая к услугам KDC.
- Описать систему Цербер (Kerberos) как центр распределения ключей (KDC) и протокол идентификации.
- Объяснить потребность в Центрах Сертификации (CA — Certification Authorities) для общедоступных ключей и формат сертификатов в соответствии с рекомендацией X.509.
- Представить идею Инфраструктуры Общедоступного ключа (PKI — Public Key Infrastructure) и объяснить некоторые из ее режимов работы.

В предыдущих лекциях мы обсуждали криптографию с симметричными ключами и с асимметричными ключами. Однако мы еще не обсудили, как распределяются и обслуживаются ключи засекречивания в криптографии с симметричными ключами и открытые ключи в криптографии с асимметричными ключами. Эта лекция затрагивает эти две проблемы.

Во-первых, мы поговорим о распределении симметричных ключей, с использованием третьего лица, которому доверяют. Во-вторых, мы покажем, как две стороны могут установить симметричный ключ между собой, не используя третье лицо, которому доверяют. В-третьих, мы рассмотрим систему Цербер (Kerberos), центры распределения ключей KDC и протокол определения подлинности объекта. В-четвертых, мы обсудим сертификацию общедоступных ключей, с помощью центров сертификации (CA), на основе рекомендаций X 509. Наконец, мы кратко рассмотрим идею относительно Инфраструктуры Общедоступного ключа (PKI) и расскажем о некоторых из ее режимов работы.

15.1. Распределение с симметричными ключами

Для шифрования больших сообщений криптография с симметричными ключами более эффективна, чем криптография с асимметричными ключами. Криптография с симметричными ключами, однако, нуждается в ключе засекречивания, который применяется двумя сторонами.

Если Алиса должна обмениваться конфиденциальными сообщениями с N людьми, она нуждается в N различных ключах. А что, если N людей должно общаться друг с другом? Тогда необходимое общее количество ключей равно N (N – l). Если мы позволяем Алисе и Бобу использовать два одинаковых ключа для двунаправленной связи для обоих направлений, тогда нужно только $N(N-1)/2$ ключей. Это означало бы, что если один миллион человек связывается друг с другом, каждый человек имеет почти один миллион различных ключей. Всего необходим

почти один триллион ключей. Это называется *N*-проблемой, потому что число требуемых ключей для *N* объектов — N^2.

Число ключей — не единственная проблема; распределение ключей — вот другая беда. Алиса и Боб хотят связаться между собой. Им нужен способ обмена ключами засекречивания. Если Алиса хочет связаться с одним миллионом человек, как она может обменяться одним миллионом ключей с одним миллионом человек? Использование Internet — явно не безопасный метод. Очевидно, что мы нуждаемся в эффективном способе поддерживать и распределять ключи засекречивания.

Центр Распределения Ключей: KDC

Практическое решение — привлечение третьего лица, которому доверяют. Оно называется здесь **центром распределения ключей (KDC — Key-Distribution Center).** Чтобы уменьшать число ключей, каждый человек устанавливает открытый ключ засекречивания с KDC, как показано на рис. 15.1.

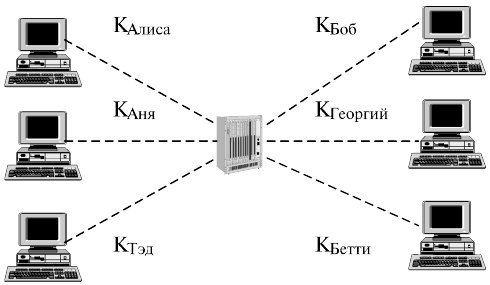


Рис. 15.1. Центр распределения ключей (KDC)

Ключ засекречивания установлен между KDC и каждым членом сообщества. Алиса имеет ключ засекречивания с KDC, который мы называем K_{Alice}. Боб имеет ключ засекречивания с KDC, который мы называем K_{Bob}. Теперь вопрос — то, как Алиса может передать конфиденциальное сообщение Бобу. Процесс следующий.

1. Алиса передает запрос KDC — заявление, что она нуждается в сеансе (временно) и ключе засекречивания между собой и Бобом.
2. KDC сообщает Бобу о запросе Алисы.
3. Если Боб соглашается, между ними создается ключ сеанса.

Ключ засекречивания между Алисой и Бобом, который установлен с KDC, используется, чтобы подтвердить подлинность Алисы и Боба к KDC и препятствовать Еве исполнять роль любого из них. Мы обсудим позже в этой лекции, как установливается ключ сеанса между Алисой и Бобом,.

Когда число людей, использующих KDC (Центр распределения ключей), увеличивается, система становится неуправляемой и срабатывает ее узкое место — число ключей может кончиться. Чтобы решить проблему, мы должны иметь много KDC. Мы можем разделить мир на домены. Каждый домен может иметь один или

более KDCs (для резервной избыточности в случае отказа). Теперь, если Алиса хочет передать конфиденциальное сообщение Бобу, который принадлежит к другому домену, она входит в контакт со своим KDC, который, в свою очередь, входит в контакт с KDC в домене Боба. Два KDCs могут создать ключ засекречивания между Алисой и Бобом. Рисунок 15.2 показывает KDCs, где центры — одного уровня. Мы называем такую организацию центров — «плоское (неиерархическое) множество центров (KDC)».

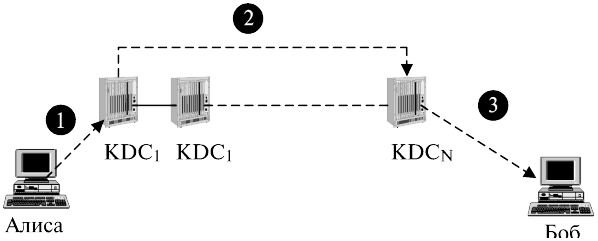


Рис. 15.2. Плоское (неиерархическое) множество KDC

Иерархическое множество центров распределения ключей

Понятие плоского множества KDCs может быть расширено на иерархическую систему KDCs, с одним или более KDCs на верхнем уровне иерархии. Например, может существовать местный KDCs, национальный KDCs и международный KDCs. Когда Алиса должна связаться с Бобом, который живет в другой стране, она передает свой запрос местному KDC; местный KDC ретранслирует запрос к национальному KDC; национальный KDC ретранслирует запрос к международному KDC. Запрос затем транслируется полным путем вниз к местному KDC, где живет Боб. Рисунок 15.3 показывает конфигурацию иерархического множества KDCs.

Ключи сеанса

KDC создает ключ засекречивания для каждого абонента. Этот ключ засекречивания может использоваться только между абонентом и KDC, а не между двумя членами сообщества. Если Алиса должна связаться тайно с Бобом, она нуждается в ключе засекречивания между собой и Бобом. KDC может создать **ключ сеанса** между Алисой и Бобом, используя их ключи с центром. Ключи Алисы и Боба применяются, чтобы подтвердить доступность и полномочность Алисы и Боба к центру и друг к другу перед тем, как будет установлен ключ сеанса. После того как связь закончена, ключ сеанса больше не нужен.

Симметричный ключ сеанса между двумя сторонами используется только однажды.

Были предложены несколько различных подходов, чтобы создать ключ сеанса, используя идеи, рассмотренные в лекции 14 для установления подлинности объекта.

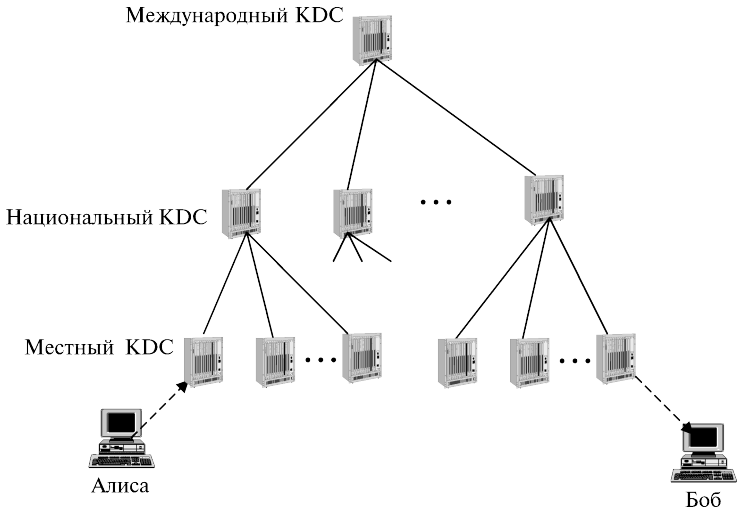


Рис. 15.3. Иерархическое множество центров распределения ключей

Простой протокол, использующий KDC

Давайте посмотрим, как KDS может создать сеансовый ключ K_{AB} между Алисой и Бобом. Рисунок 15.4 показывает предпринимаемые шаги.

1. Алиса передает сообщение исходного текста KDC, чтобы получить симметричный ключ сеанса между собой и Бобом. Сообщение содержит ее зарегистрированный опознавательный код (слово *Алиса* или рисунок) и опознавательный код Боба (слово *Боб* или рисунок). Зашифровано ли сообщение или общедоступно — KDC это не беспокоит.
2. KDC получает сообщение и создает то, что называется **билетом**. Билет зашифрован с помощью ключа Боба (K_B). Билет содержит идентификаторы Алисы и Боба и ключ сеанса (K_{AB}). Билет с копией ключа сеанса передают Алисе. Алиса получает сообщение, расшифровывает его и извлекает ключ сеанса. Она не может расшифровать билет Боба; билет, предназначенный для Боба, недоступен Алисе. Обратите внимание, что сообщение содержит двойное шифрование: зашифрован билет, а также зашифровано полное сообщение. Во втором сообщении Алиса фактически зарегистрирована в KDC, поэтому только Алиса может открыть целое сообщение, используя свой ключ засекречивания с KDC.
3. Алиса передает билет Бобу. Боб открывает билет и знает, что Алиса должна передать ему сообщение, использующее K_{AB} как ключ сеанса. Обратите внимание, что в этом сообщении Боб зарегистрирован в KDC, поэтому только Боб может открыть билет. Поскольку Боб зарегистрирован в KDC, он также зарегистрирован Алисой, которая доверяет KDC. Тем же самым способом Алиса также зарегистрирована Бобом, потому что Боб доверяет KDC, и KDC передал Бобу билет, который включает опознавательный код Алисы.

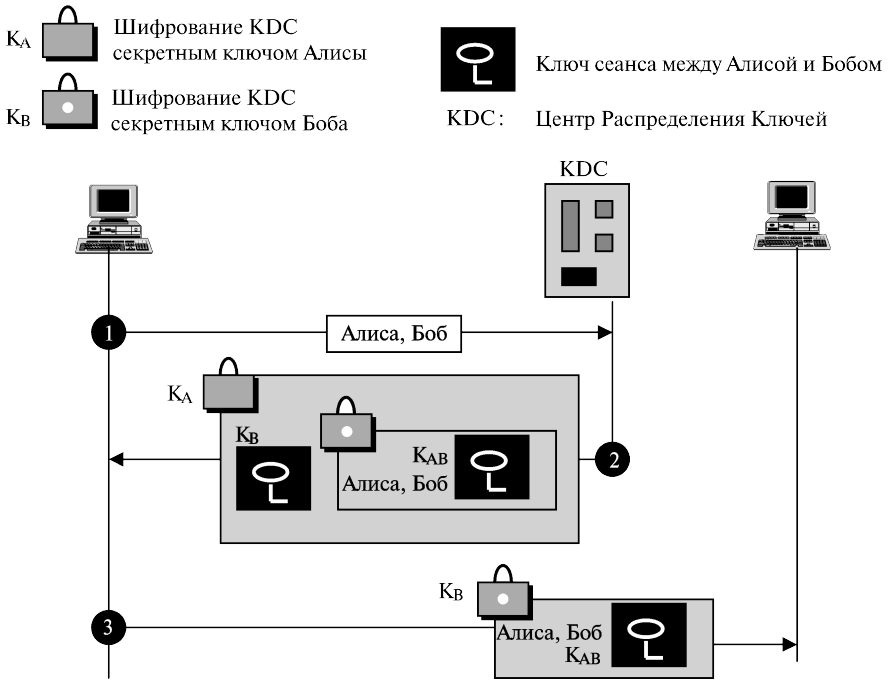


Рис. 15.4. Первый метод, использующий KDC

К сожалению, этот простой протокол имеет недостаток. Ева может применить атаку ответа, рассмотренную раньше, — то есть она может сохранить сообщение шага 3 и использовать его позже.

Протокол Ниидома-Шрёдера

Другой подход — изящный **протокол Ниидома-Шрёдера (Needham-Schreder)**, который является основой многих протоколов. Этот протокол задействует множество действий запроса-ответа между сторонами, чтобы достигнуть безупречного протокола. Ниидом и Шрёдер применяют два nonce: R_A и R_B. Рисунок 15.5 показывает пять шагов, используемых в этом протоколе. Мы кратко представляем каждый шаг.

1. Алиса передает сообщение KDC, в которое включает свой nonce R_A, свой опознавательный код и опознавательный код Боба.
2. KDC передает зашифрованное сообщение Алисы, которое включает once Алисы, опознавательный код Боба, ключ сеанса и зашифрованный билет для Боба. Все сообщение зашифровано ключом Алисы.
3. Алиса передает билет Боба ему.
4. Боб передает свой запрос Алисе (R_B), зашифрованный ключом сеанса.
5. Алиса отвечает на запрос Боба. Обратите внимание, что ответ передается $R_B - 1$ вместо R_B.

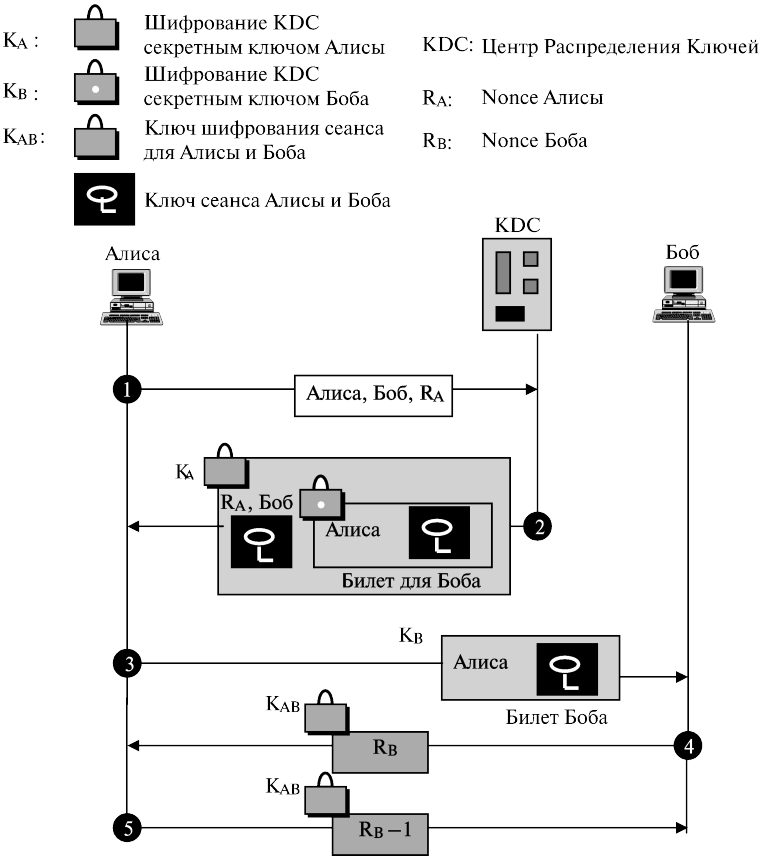


Рисунок 15.5. Протокол Ниидома-Шрёдера

Протокол Отвея-Рисса

Третий подход — **протокол *Отвея-Рисса (Otway-Rees)***, другой, не менее изящный протокол. Рисунок 15.6 показывает этот протокол с пятью шагами.

Ниже кратко описываются шаги этого протокола.

1. Алиса передает сообщение Бобу, которое включает nonce R, идентификационные признаки Алисы и Боба и билет для KDC — в билет входят once Алисы R_A (свидетельство для пользования KDC), копия общего nonce R и идентификаторы Алисы и Боба.
2. Боб создает тот же самый тип билета, но с собственным его, Боба, nonce R_B. Оба билета передают KDC.
3. KDC создает сообщение, которое содержит R, общий nonce, билет для Алисы и билет для Боба; сообщение передают Бобу. Билеты содержат соответствующий nonce, R_A или R_B и ключ сеанса K_{AB}.

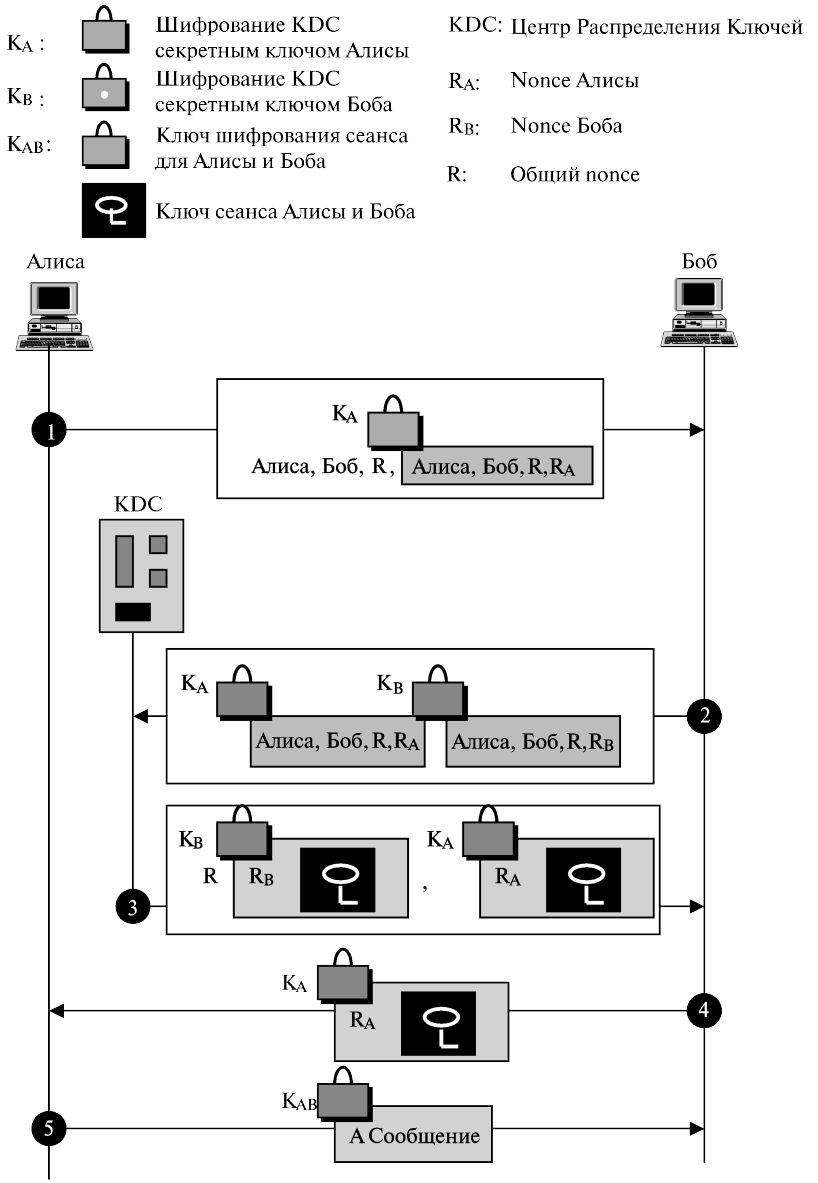


Рисунок 15.6. Протокол Отвея-Рииса

4. Боб передает Алисе ее билет.
5. Алиса передает короткое сообщение, зашифрованное ее сеансовым ключом K_{AB}, чтобы показать, что она имеет ключ сеанса.

15.2. Цербер

Цербер — протокол установления подлинности и в то же самое время — KDC, который стал очень популярным. Несколько систем, включая Windows 2000, используют протокол Цербер. Он назван в честь трехголовой собаки в греческой мифологии, которая охраняет ворота царства мертвых Аида. Первоначально разработанный в MIT, он прошел несколько версий. Мы обсудим только самую популярную версию 4, и кратко объясним отличия между версией 4 и версией 5 (последней).

Серверы

Протокол Цербер включает в себя работу с тремя серверами: сервер аутентификации (AS — Authentication Server), сервер, предоставляющий билет (TGS — Ticket-Granting Server) и реальный сервер (сервер обработки данных), который обеспечивает услуги. В наших примерах и рисунках *Боб* — реальный сервер, а *Алиса* — пользователь, запрашивающий сервер. Рисунок 15.7 показывает отношения между этими тремя серверами.

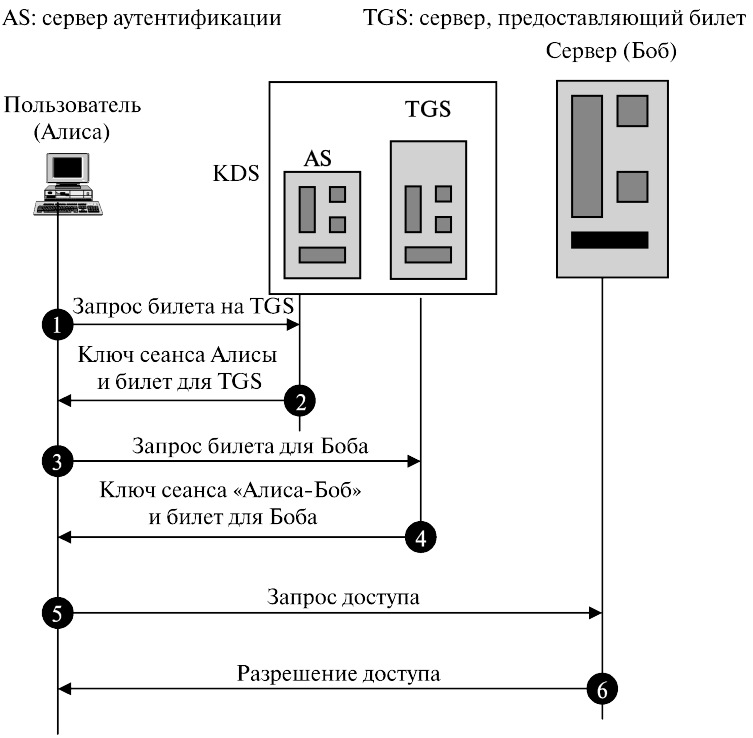


Рис. 15.7. Серверы протокола Цербер

Сервер аутентификации (AS)

Сервер аутентификации (AS) — в протоколе Цербер — KDC. Каждому пользователю, зарегистрированному в AS, предоставляют пользовательский идентификационный код и пароль. AS имеет базу данных с этими идентификационными кодами и соответствующими паролями. AS верифицирует пользователя, выдает ключ сеанса, который используется между Алисой и TGS, и передает билет для TGS.

Предоставляющий билет сервер (TGS)

Предоставляющий билет сервер (TGS) вырабатывает билет для реального сервера (Боба). Он обеспечивает ключ сеанса (K_{AB}) между Алисой и Бобом. Протокол Цербер отделяет верификацию пользователя от выдачи билета.

Этим способом Алиса проверяет свой ID с AS только один раз. В контакт с TGS она может войти много раз, чтобы получить билеты для различных реальных серверов.

Реальный сервер

Реальный сервер (Боб) обеспечивает услуги для пользователя (Алиса). Цербер разработан для взаимодействия с программой «клиент-сервер», такой как, например, протокол передачи файлов FTP (File Transfer Protocol), в котором пользователь использует процесс клиента, чтобы обратиться к процессу сервера. Цербер не применяется для установления подлинности «человек-человек».

Работа

Процесс клиента (Алиса) может обратиться к процессу, функционирующему на реальном сервере (Боб) в шесть шагов, как это показано на рис. 15.8.

1. Алиса передает свой запрос AS в открытом тексте, используя свой зарегистрированный код идентификации.
2. AS передает сообщение, зашифрованное постоянным симметричным ключом Алисы, K_A. AS-сообщение содержит два объекта: ключ сеанса, $K_{A\text{-}TGS}$, который используется Алисой, чтобы войти в контакт с TGS, и билет для TGS, который зашифрован TGS-симметричным ключом ($K_{AS\text{-}TGS}$). Алиса не знает $K_{A\text{-}AS}$, но когда сообщение прибывает, она печатает (сообщает) свой симметричный пароль. Пароль и соответствующий алгоритм вместе создают $K_{A\text{-}AS}$, если пароль правильный. Пароль затем немедленно уничтожают; его не передают по сети, и он не остается в терминале. Он используется только на мгновение, чтобы создать $K_{A\text{-}AS}$. Процесс теперь применяет $K_{A\text{-}AS}$ для того, чтобы расшифровывать передаваемое сообщение $K_{A\text{-}TGS}$ и извлечь билет.
3. Алиса теперь передает три объекта TGS. Первый — билет, полученный от AS. Второй — имя реального сервера (Боб), третий — метку времени, которая зашифрована ключом $K_{A\text{-}TGS}$. Метка времени предотвращает ложный ответ Евы.
4. Теперь TGS передает два билета: каждый содержит ключ сеанса между Алисой и Бобом, $k_{a\text{-}b,}$. Билет для Алисы — зашифрованный $k_{a\text{-}tgs}$, билет

для Боба — зашифрованный с ключом Боба K_{TGS-B}. Обратите внимание, что Ева не может извлечь k_{ab}, потому что Ева не знает K_{A-TGS} или K_{TGS-B}. Она не может ответить на шаг 3, потому что не может заменить метку времени новой меткой. Она не знает K_{A-TGS}, и даже если будет действовать очень быстро и передаст на шаге 3 сообщение прежде, чем истечет метка времени, все равно получит те же самые два билета, которые она не может расшифровать.

5. Алиса передает билет Боба с меткой времени, зашифрованной ключом K_{A-B}.
6. Боб подтверждает, что получил эту информацию, прибавляя 1 к метке времени. Сообщение шифруется ключом K_{A-B} и передается Алисе.

Использование различных серверов

Обратите внимание, что если Алиса должна быть обслужена различными серверами, то первые два шага проверяют идентификационный код Алисы и не повторяются. Повторяя шаги 3-6, Алиса может запросить, чтобы TGS выработал билеты для многих серверов.

Версия 5 Цербер

Незначительные отличия между версией 4 и версией 5 кратко приведены ниже.

1. Версия 5 имеет более длинный указатель времени жизни билета.
2. Версия 5 позволяет возобновлять билеты.
3. Версия 5 может применять любой алгоритм с симметричными ключами.
4. Версии 5 использует различные протоколы для того, чтобы описывать типы данных.
5. Версия 5 имеет заголовки, более длинные, чем версия 4.

Области применения

Цербер позволяет глобальное распределение AS и TGS с каждой системой, называемой *областью*. Пользователь может получить билет для местного сервера или удаленного сервера. Во втором случае, например, Алиса может просить, чтобы ее местный TGS создал ей билет, который мог бы быть принят удаленным TGS. Местный TGS может создать этот билет, если удаленный TGS зарегистрирован в местном TGS, тогда Алиса может использовать удаленный TGS, чтобы обратиться к удаленному реальному серверу.

15.3. Соглашение с симметричными ключами

Алиса и Боб могут создать ключ сеанса между собой, не используя KDC. Этот метод создания ключа сеанса называется соглашением с симметричными ключами. Хотя есть несколько способов выполнить этот процесс, здесь рассматриваются только два общих метода ключевого соглашения: Диффи-Хеллмана (Diffie-Hellman) и «станция-к-станции».

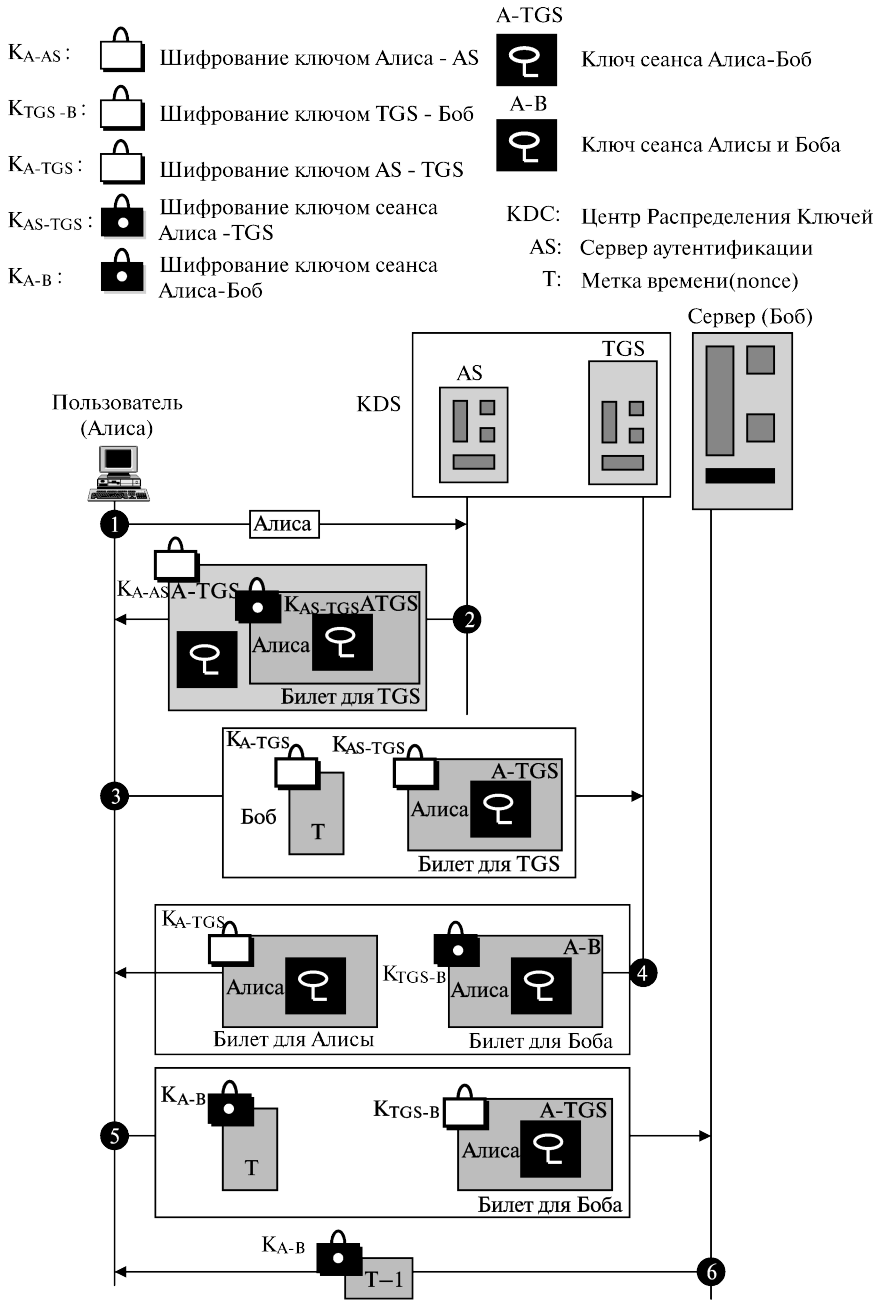


Рис. 15.8. Пример работы Цербера

Ключевое соглашение

В **протоколе Диффи-Хеллмана** две стороны создают симметричный ключ сеанса без KDC. Перед установлением симметричного ключа эти две стороны должны выбрать два числа p и g. Первое число, p, является большим простым числом порядка 300 десятичных цифр (1024 бита). Второе число, g, служит генератором порядка $p - 1$ в группе $<Z_{p^*}, \times>$. Эти два числа (группа и генератор) не должны быть конфиденциальными. Их можно передать через Internet. Они могут быть общедоступны. Рисунок 15.9 показывает процедуру.

Шаги перечислены ниже.

1. Алиса выбирает большое случайное число x, такое, что $0 < x < p - 1$, и вычисляет $R_1 = g^x \bmod p$.
2. Боб выбирает другое большое случайное число y, такое, что $0 < y < p - 1$, и вычисляет $R_2 = g^y \bmod p$.
3. Алиса передает Бобу R_1. Обратите внимание, что Алиса не передает значение x; она передает только R_1.
4. Боб передает Алисе R_2. Снова обратите внимание, что Боб не передает значение y, он передает только R_2.
5. Алиса вычисляет $K = (R_2)^x \bmod p$.
6. Боб также вычисляет $K = (R_1)^y \bmod p$.

$$K = (g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$$

Боб вычисляет $K = (R_1)^y \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p$. Алиса вычисляет $K = (R_2)^x \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$ и получает то же самое

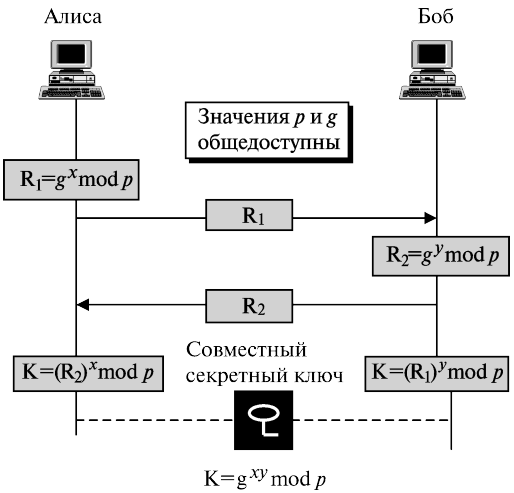


Рис. 15.9. Метод Диффи-Хелмана

значение без Боба, знающего значение x. А Боб получил это значение без Алисы, знающей значение y.

Симметричный (совместный) ключ в методе Диффи-Хеллмана — $K = g^{xy} \bmod p$

Пример 15.1

Приведем тривиальный пример, чтобы ясно понять процедуру. Наш пример использует маленькие числа, но заметим, что в реальной ситуации применяются очень большие числа. Предположим, что $g = 7$ и $p = 23$. Тогда процедура содержит следующие шаги.

1. Алиса выбирает $x = 3$ и вычисляет $R_1 = 7^3 \bmod 23 = 21$.
2. Боб выбирает $y = 6$ и вычисляет $R_2 = 7^6 \bmod 23 = 4$.
3. Алиса передает число 21 Бобу.
4. Боб передает число 4 Алисе.
5. Алиса вычисляет симметричный ключ $K = 4^3 \bmod 23 = 18$.
6. Боб вычисляет симметричный ключ $K = 21^6 \bmod 23 = 18$.

Значение K одно и то же и для Алисы, и для Боба: $g^{x\,y} \bmod p = 7^{18} = 18$.

Пример 15.2

Давайте возьмем более реальный пример. Мы используем программу, чтобы создать случайное целое число 512 битов (идеально — 1024 бит). Целое число p — число с 159 цифрами. Мы также выбираем g, x и y, как показано ниже:

p	76462429856349357218249376595503050747633809672694974892357377286092523566666075542363742330966118003333810619473013095041473870099917804365487858079875 81
g	2
x	557
y	273

Следующая таблица показывает R_1, R_2 и K.

R_1	84492028420665505216172947491035094143433698520012660862863631067673619959280828586700802131859290945140217500319973312945836083821943065966020157955354
R_2	43526283870920037947074711489558162763638911626211555797512337921856631001143571S2083900401818764868417538311653426916302634211067215085589 6255201288594143
K	15563800066452229059622582752327076527321804694442367852032040014640650088793665120425742677660832791101715303867456125221315161097658420012040864336177 40

Анализ протокола Диффи-Хеллмана

Концепция ***Диффи-Хеллмана***, показанная на рис. 15.10, является простой, но изящной. Мы можем представить ключ засекречивания между Алисой и Бобом —

он состоит из трех частей: *g*, *x* и *y*. Первая часть общедоступна. Каждый знает 1/3 ключа — *g*, общедоступное значение. Другие две части нужно узнать у Алисы и Боба. Каждый из них знает одну часть. Алиса добавляет *x* как вторую часть для Боба; Боб добавляет *y* как вторую часть для Алисы. Когда Алиса получает 2/3 полного ключа от Боба, она добавляет последнюю часть, ее *y*, чтобы завершить ключ. Когда Боб получает ключ от Алисы — законченный на 2/3, он добавляет последнюю часть, свое *y*, чтобы завершить ключ. Обратите внимание, что хотя ключ Алисы состоит из *g*, *y* и *x* и ключ Боба состоит из g, *x* и y, эти два ключа — одни и те же, потому что $g^{xy} = g^{yx}$.

Обратите внимание также, что хотя два ключа те же самые, Алиса не может найти значение *y*, используемого Бобом, потому что вычисление сделано по модулю *p*. Алиса получает $g^y \bmod p$ от Боба, но не g^y. Для того чтобы знать значения *y*, Алиса должна использовать дискретный логарифм, который мы обсуждали в предыдущей лекции.

Безопасность протокола

Замена ключа ***Диффи-Хеллмана*** восприимчива к двум атакам: атаке дискретного логарифма и атаке посредника (man-in middle)

Атака дискретного логарифма. Безопасность ключевой станции базируется на трудности проблемы дискретного логарифма. Ева может перехватить R_1 и R_2.

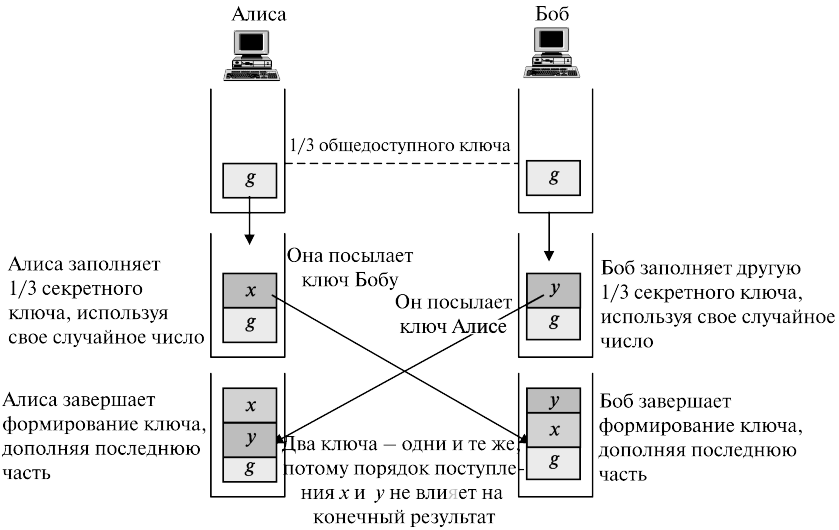


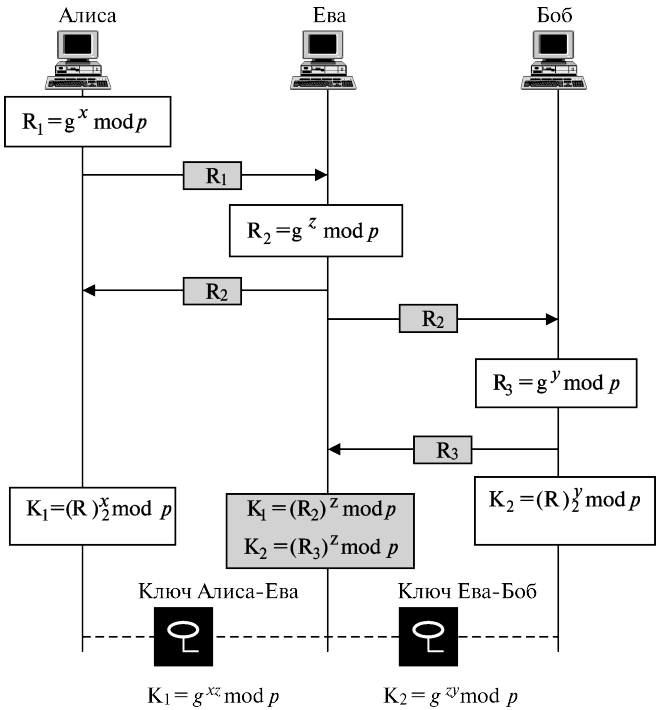
Рис. 15.10. Идея ключа Диффи-Хеллмана

Из $R_1 = g^x \bmod p$ и *y* из $R_2 = g^y \bmod p$ она может затем вычислить симметричный ключ: $K = g^{xy} \bmod p$. Ключ засекречивания больше не является секретным. Чтобы сделать метод Диффи-Хеллмана защищенным от атаки дискретного логарифма, рекомендуется следующее.

1. Простое число p должно быть очень большим (более чем 300 десятичных цифр).
2. Простое число p должно быть выбрано так, чтобы $p - 1$ имел по крайней мере один простой делитель (больше чем 60 десятичных цифр).
3. Генератор должен быть выбран из группы $<Z_{p^*}, \times>$.
4. Боб и Алиса должны уничтожить x и y после того, как они вычислили значение симметричного ключа. Значения x и y должны использоваться только единожды.

Атака «посредника». Этот протокол имеет другую слабость. Еве не надо находить значения x и y, чтобы напасть на протокол. Она может использовать глупость Алисы и Боба, создающих два ключа: один между Бобом и Алисой и другой между Алисой и Бобом. Рисунком 15.11 показывает ситуацию. Может случиться следующее:

1. Алиса выбирает x , вычисляет $R_1 = g^x \bmod p$ и передает R_1 Бобу;
2. Ева, злоумышленник, перехватывает R_1. Она выбирает z, вычисляет $R_2 = g^z \bmod p$ и передает R_2 Алисе и Бобу;
3. Боб выбирает y, вычисляет $R_3 = g^y \bmod p$, передает R_3 Алисе. Ева перехватывает R_3, и Алиса никогда не получит это число;



Ри. 15.11. Атака посредника

4. Алиса и Ева вычисляют $K_1 = g^{xz} \bmod p$, который становится открытым ключом между Алисой и Евой. Алиса, однако, думает, что это —открытый ключ между ней и Бобом;
5. Ева и Боб вычисляют $K_2 = g^{zy} \bmod p$, который становится открытым ключом между Евой и Бобом. Однако Боб думает, что это — открытый ключ между ним Алисой.

Другими словами, создаются два ключа вместо одного: один между Алисой и Евой и один между Евой и Бобом. Когда Алиса посылает данные Бобу, она зашифровывает их ключом K_1 (совместный ключ Алисы иЕвы). Эти данные могут быть расшифрованы и прочитаны Евой. Ева может передать сообщение Бобу, зашифрованное K_2 (совместный ключ между Евой и Бобом); или она может даже изменить сообщение или передать новое сообщение. Боб введен в заблуждение, поскольку уверен, что сообщение пришло от Алисы. Подобный сценарий может случиться и в другом направлении — с Алисой.

Эта ситуация называется **«атака посредника»**, поскольку Ева находится между партнерами и перехватывает R_1, передаваемый Алисой Бобу, и R_3, передаваемый Бобом Алисой. Это — **атака передачи по цепочке**, потому что напоминает короткую линейку добровольцев на пожаре, передающих друг другу ведра с водой, по цепочке от человека человеку.

Следующий метод основан на протоколе Диффи-Хеллмана. Он использует методы установления подлинности, чтобы сорвать эту атаку.

Ключевое соглашение «от станции к станции»

Протокол «от станции к станции» — метод, основанный на методе Диффи-Хеллмана. Он применяет цифровые подписи с сертификатами открытого ключа (см. следующую секцию). Для установки ключа сеанса между Алисой и Бобом используется последовательность, показанная на рис. 15.12.

Имеются следующие шаги.

- После вычисления R_1 Алиса передает R_1 Бобу (шаги 1 и 2 на рис. 15.12).
- После вычисления R_2 и ключа сеанса Боб конкатенирует ID Алисы, R_1 и R_2. Затем он подписывает результат своим секретным ключом. Боб теперь передает R_2, подпись и собственное свидетельство общедоступного ключа Алисе. Подпись зашифрована ключом сеанса (шаги 3, 4 и 5 на рис. 15.12).
- После вычисления ключа сеанса, если подпись Боба проверена, Алиса связывает ID Боба, R_1 и R_2. Затем она подписывает результат своим собственным секретным ключом и передает это Бобу. Подпись зашифрована ключом сеанса (шаги 6, 7 и 8 на рис. 15.12).
- Если подпись Алисы проверена, Боб сохраняет ключ сеанса (шаг 9 на рис. 15.12).

Безопасность протокола «от станции к станции»

Протокол «от станции к станции» предотвращает атаки «посредника». После R_1 Ева не может передать свой собственный R_2 Алисе и притворяться, что это идет от Боба, потому что Ева не может подделать секретный ключ Боба и создать подпись — подпись не может быть проверена общедоступным ключом Боба, оп-

ределенным в свидетельстве. Тем же образом Ева не может подделать секретный ключ Алисы, чтобы подписать третье сообщение, передаваемое Алисой. Сертификату, как мы увидим в следующей секции, можно доверять, потому что он выработан администрацией, которой доверяют.

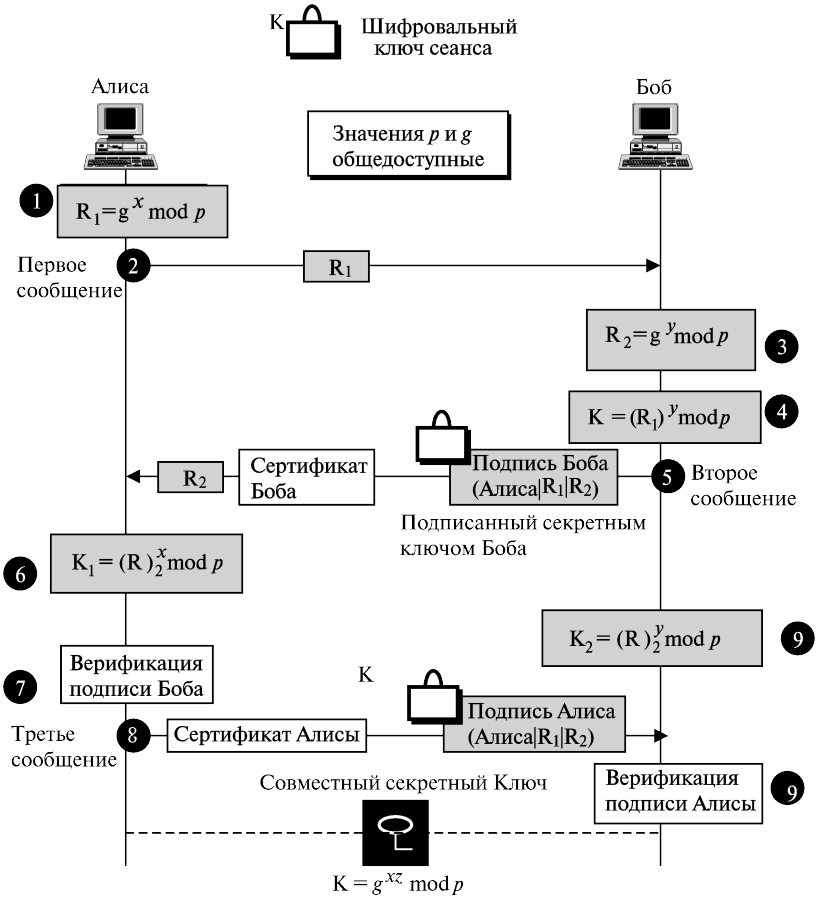


Рис. 15.12. Метод соглашения «от станции — к станции»

15.4. Распределение открытого ключа

В криптографии с асимметричным ключом людям не надо знать симметричный открытый ключ. Если Алиса хочет передать сообщение Бобу, она должна знать только открытый ключ Боба, который является открытым для всех и доступен каждому. Если Боб должен передать сообщение Алисе, он должен знать толь-

ко открытый ключ Алисы, который также известен каждому. В криптографии общедоступного ключа каждый сохраняет секретный ключ и объявляет общедоступный ключ.

В криптографии общедоступного ключа каждый имеет доступ к открытому ключу; открытые ключи доступны всем.

Общедоступные ключи, подобно секретным ключам, должны быть распределены, чтобы быть полезными. Кратко обсудим способ, которым могут быть распределены общедоступные ключи.

Общедоступное объявление

Наивный подход состоит в том, чтобы объявить открытые ключи публично. Боб может поместить свой открытый ключ на своем сайте или объявить его в местной или национальной газете. Когда Алиса должна передать конфиденциальное сообщение Бобу, она может получить открытый ключ Боба из его сайта или из газеты или даже передать сообщение, чтобы попросить его об этом. Рисунок 15.13 иллюстрирует такую ситуацию.

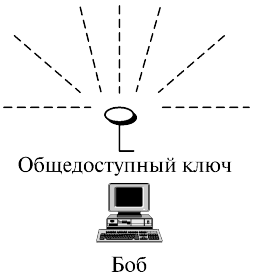


Рис. 15.13. Объявление открытых общедоступных ключей

Этот подход, однако, небезопасен. Он допускает подделку. Например, Ева может сделать такое же общедоступное объявление. Прежде, чем Боб сможет среагировать, может быть нанесен вред. Ева может послать глупой Алисе свое сообщение, которое якобы написано Бобом. Ева может также подписать документ фальшивым секретным ключом и вызвать у каждого предположение, что сообщение было подписано Бобом. Этот подход также уязвим, если Алиса сама запрашивает открытый ключ Боба. Ева может перехватить ответ Боба и заменить его собственным фальшивым открытый ключом.

Центр доверия

Более безопасный подход состоит в том, чтобы иметь центр, которому доверяют и который хранит каталог общедоступных (открытых) ключей: каталог, подобно используемому в телефонной системе, но динамически модифицируемый.

Каждый пользователь может выбрать секретный и открытый ключ, сохранить секретный ключ и опубликовать открытый ключ в каталоге. Центр может предоставить пользовательский регистр и проверить опознавательный код. Каталог может публично рекламироваться центром, которому доверяют. Центр может также ответить на любой запрос об общедоступном ключе. Рисунок 15.14 показывает эту концепцию.

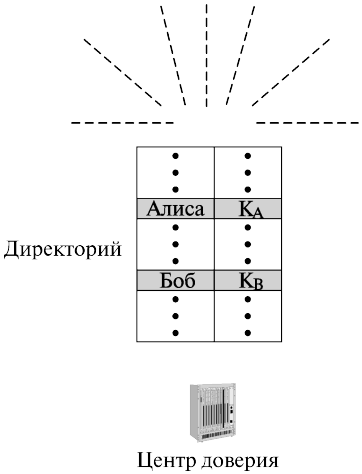


Рис. 15.14. Центр доверия

Управляемый центр доверия

Более высокий уровень безопасности может быть достигнут, если добавить управление распределением открытого ключа. При объявлении открытого ключа можно включить в ответ метку времени и подпись администрации, чтобы предотвратить перехват и переделку ответа. Если Алиса хочет знать открытый ключ Боба, она может передать запрос центру, включая в запрос имя Боба и метку времени. Центр отвечает открытым ключом Боба, первоначальным запросом и меткой времени, подписанной секретным ключом центра. Алиса использует открытый ключ известного всем центра и проверяет метку времени. Если метка времени правильная, она извлекает общедоступный ключ Боба. Рисунок 15.15 показывает один из возможных сценариев.

Центр сертификации

Предыдущий подход может породить высокую нагрузку на центр, если число запросов будет большим. Альтернатива этому — создание с**ертификата (удостоверения) общедоступного ключа.** Боб имеет два желания: он хочет, чтобы люди зна-

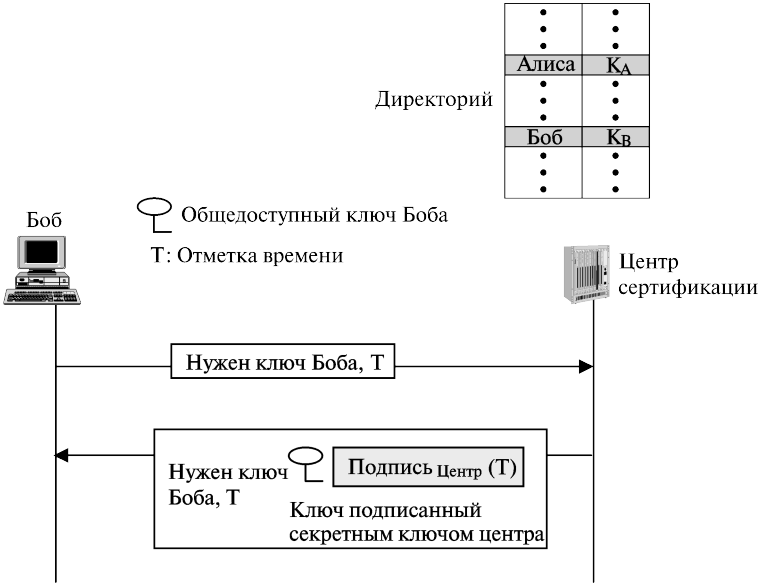


Рис. 15.15. Управляемый Центр Доверия

ли его открытый ключ, и он хочет, чтобы никто не сформировал фальшивый открытый ключ, такой же, как у него. Боб может обратиться в **центр сертификации** (CA — Certification Authority) либо в федеральную или общегосударственную организацию, которая связывает открытый ключ с объектом и выдает сертификат. Центр сертификации имеет известный общедоступный ключ, который не может быть фальшивым. Центр сертификации проверяет идентификацию Боба, используя картинку, или ID, или другое доказательство подлинности заявителя, затем запрашивает открытый ключ Боба и подписывает сертификат с секретным ключом. Центр сертификации подписывает свидетельство своим секретным ключом. Теперь Боб может загрузить подписанное свидетельство. Любой, кто хочет иметь открытый ключ Боба, загружает подписанное свидетельство и использует общедоступный ключ центра, чтобы извлечь общедоступный ключ Боба. Рисунок 15.16 показывает эту концепцию.

X.509

Хотя применение сертификационных центров решило проблему мошенничества при распределении открытого ключа, они создали побочный эффект. Каждое свидетельство может иметь особый формат. Если Алиса хочет использовать программу, чтобы автоматически загрузить различные сертификаты и дайджесты, принадлежащие различным людям, программа не сможет сделать это. Одно свидетельство может иметь открытый ключ в различных форматах. Открытый ключ может быть на первой линейке в одном сертификате и на третьей линейке в дру-

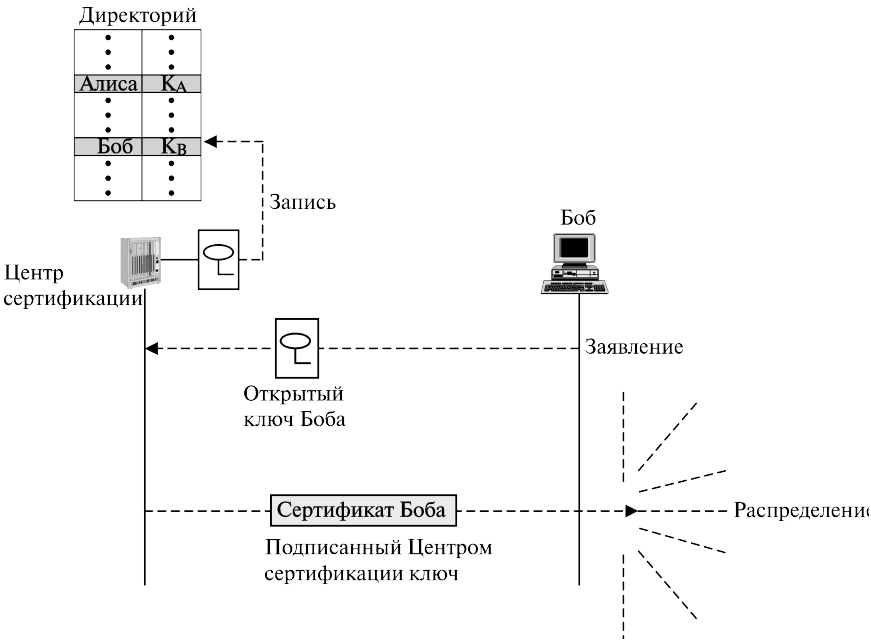


Рис. 15.16. Администрация сертификации

гом. Поэтому то, что надо применять универсально, должно иметь универсальный формат.

Чтобы обеспечить универсальность, ITU (МСЭ) разработал рекомендацию X.509**,** которая была принята в Internet с некоторыми изменениями. X.509 — способ описать сертификат структурированным способом. Он использует известный протокол, называемый ASN.l (Abstract Syntax Notation 1 — Нотация абстрактного синтаксиса 1), — он определяет поля, которые знакомы программистам, использующим C.

Сертификат

Рисунок 15.17 показывает формат сертификата.

Сертификат имеет следующие поля.

- **Номер версии.** Это поле определяет версию сертификата X.509. Номер версии начинается отсчитываться с 0; текущая версия (третья версия) — 2.
- **Серийный номер.** Это поле определяет число, назначаемое каждому сертификату. Значение этого числа является уникальным для каждого выпускаемого свидетельства.
- **Алгоритм подписи ID.** Это поле идентифицирует алгоритм, используемый для подписи сертификата. В этом поле определяется любой параметр, который необходим для подписи.
- **Название выдавшего сертификат.** Это поле идентифицирует центра сертификации, который выдал свидетельство. Название — обычно иерархия строк, которые определяют страну, штат, организацию, отдел и так далее.

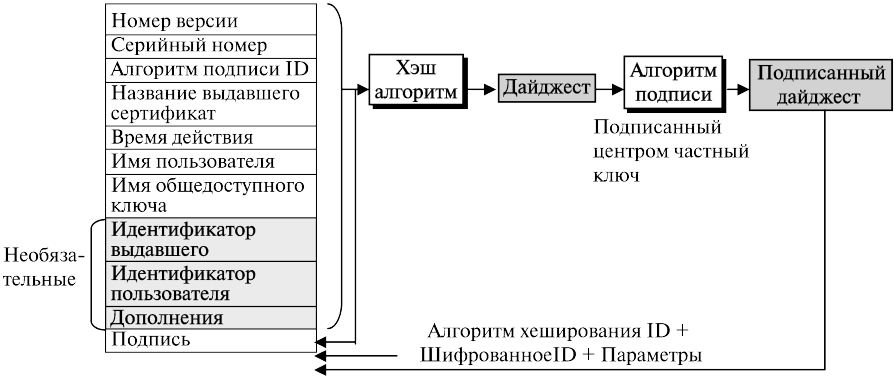


Рис. 15.17. Формат сертификата X.509

- **Срок действия.** Это поле определяет начальное время (не раньше) и последнее время (не позже), когда сертификат считается действительным.
- **Имя пользователя.** Это поле определяет объект, которому принадлежит открытый ключ. Это также иерархия строк. Часть поля определяет то, что называется *общим именем*, которое является фактическим именем обладателя ключа.
- **Имя общедоступного ключа.** Это поле определяет открытый ключ владельца. Это — центральная информация сертификата. Поле также определяет соответствующий алгоритм общедоступного ключа (например, RSA) и его параметры.
- **Уникальный идентификатор выдавшего сертификат.** Это дополнительное поле позволяет двум организациям, выдавшим ключ, иметь одно то и же значение поля *выдавшего*, если *уникальные идентификаторы выдавшего* различны.
- **Уникальный идентификатор пользователя.** Это дополнительное поле позволяет двум различным пользователям иметь одно и то же поле *пользователя*, если *уникальные идентификаторы* пользователя различны.
- **Дополнительное расширение формата.** Это дополнительное поле позволяет выпускающим прикладывать больше частной информации, дополняющей сертификат.
- **Подпись.** Это поле состоит из трех секций. Первая секция содержит все другие поля в сертификате. Вторая содержит дайджест первой секции, зашифрованный с общедоступным ключом сертификационного центра (CA). Третья — идентификатор алгоритма, использованного для создания второй секции.

Возобновление сертификата

Каждое свидетельство имеет срок действия. Если нет никаких проблем с сертификатом, Центр сертификации выдает новый сертификат прежде, чем исте-

кает старый. Этот процесс подобен возобновлению кредитных карточек компанией, выпускающей кредитные карточки; держатель кредитки обычно получает возобновленную карточку прежде, чем срок действия старой истекает.

Аннулирование сертификата

В некоторых случаях сертификат должен быть отменен перед тем, как истечет срок его действия. Например:

а. Секретный ключ (объекта) пользователя, соответствующий открытому ключу, который перечислен в сертификате, возможно, был скомпрометирован.

б. Центр Сертификации больше не желает удостоверять пользователя. Например, свидетельство пользователя касается организации, в которой он больше не работает.

в. Секретный ключ центра сертификации, который может проверять сертификаты, возможно, был скомпрометирован.

В этом случае центр сертификации должен отменить все неистекшие сертификаты.

Аннулирование происходит путем периодического выпуска списка аннулированных сертификатов (CRL — certificate revocation). Список содержит все отменяемые сертификаты, срок которых не истек в день выпуска CRL. Когда пользователь хочет использовать сертификат, он сначала должен проверить каталог соответствующего сертификационного Центра, просмотрев последний список аннулирования сертификатов. Рисунок 15.18 показывает список аннулирования сертификатов.

Список аннулирования сертификатов имеет следующие поля.

- **Алгоритм подписи ID.** Это поле то же самое, как и в сертификате,
- **Название выдавшего сертификат.** Это поле то же самое, как и в сертификате.

Дата модификации. Это поле определяет, когда список был выпущен.

- **Дата последнего обновления.** Это поле определяет следующую дату, когда будет выпущен новый список.

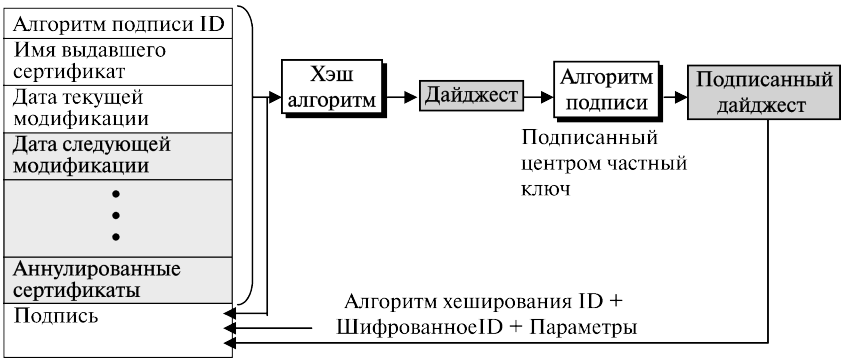


Рис.15.18. Формат аннулирования сертификата X.509

- **Аннулированный сертификат.** Это повторяемый список всех аннулированных сертификатов, у которых не истек срок. Каждый список содержит две части: пользовательский серийный номер сертификата и дату аннулирования.
- **Подпись.** Это поле такое же, как и в списке сертификатов.

Аннулирование с помощью дельта-списка

Чтобы сделать аннулирование более эффективным, был предложен дельта-список аннулирования сертификата (дельта-список CRL — certification list). Дельта-CRL создается и размещается в директории, если есть изменения в период, который начинается с даты последнего обновления сертификата до следующей модификации, — например, если CRL вырабатываются каждый месяц, но есть аннулирования между этим датами. Центр сертификации может создать дельта-список CRL, когда есть изменение в течение месяца. Однако дельта-CRL содержит только изменения, сделанные после последнего CRL.

Инфраструктура открытых ключей (PKI)

Инфраструктура открытых ключей (PKI — Public Key Infrastructures) — модель для создания, распределения и аннулирования сертификатов, основанная на рекомендации X.509. Группа инженерной поддержки сети Интернет (IETF — Internet Engineering Task Force) создала Инфраструктуру общедоступного ключа X.509 (TKIX).

Режимы работы

Для PKI были определены несколько режимов работы. Самые важные из них показаны на рис. 15.19.

- **Выпуск, возобновление и аннулирование сертификатов.** Эти режимы работы были определены в X.509. Поскольку PKIX базируется на X.509, он должен обработать все режимы работы, имеющие отношение к сертификатам.
- **Хранение и модификация ключей.** PKI должен быть местом хранения секретных ключей для тех участников, у которых есть необходимость держать свои секретные ключи где-нибудь в сейфе. В дополнение к этому PKI несет ответственность за обновление этих ключей по запросу участников.

Рис. 15.19. Некоторые режимы PKI

- **Обеспечение услуг другим протоколам.** Как мы увидим в следующих немногих лекциях, некоторые протоколы безопасности Internet, такие, как IPSec и TLS, базируются на услугах PKI.
- **Обеспечение управления доступом.** PKI может обеспечить различные уровни доступа к информации, сохраненной в ее базе данных. Например, организация PKI может обеспечить доступ к полной базе данных для высшего исполнительного руководства, но ограниченный доступ — для служащих.

Модель доверия

Невозможно иметь только один центр сертификации, выпускающий все сертификаты для всех пользователей в мире. Должно быть много центров сертификации (CA), каждый — ответственный за создание, сохранение, издание и аннулирование ограниченного числа сертификатов. **Модель доверия (Trust Model)** определяет правила, которые говорят, как пользователь может проверить сертификат, полученный от Центра Сертификации (CA).

Иерархическая модель. У этой модели — структура типа дерева с корнем CA. Корень CA имеет сертификат, подписанный и выпущенный им самим. Другим CA и пользователям для того, чтобы работать, необходимо доверять ему. Рисунок 15.20 показывает модель доверия такого вида с тремя иерархическими уровнями. В реальной ситуации число уровней может быть больше, чем три.

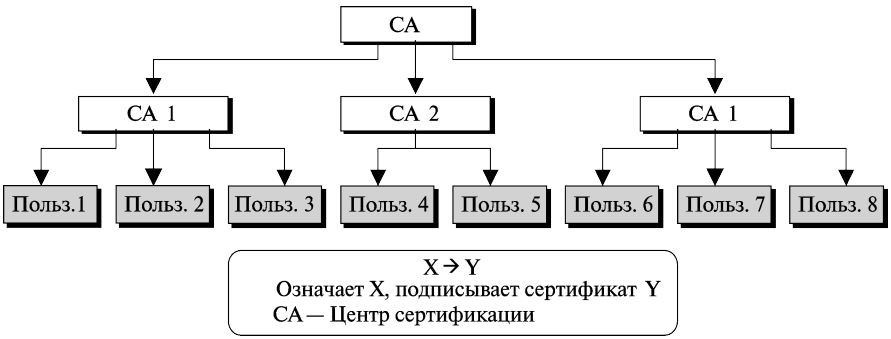


Рис. 15.20. Иерархическая модель PKI

Рисунок показывает, что CA (корень) подписывает сертификаты для CA1, CA2 с CA3; CA1 подписывают сертификаты для Польз.1, Польз. 2, Польз. 3 и так далее. PKI использует особую систему обозначений, которая означает: сертификат выдан администрацией X для объекта Y.

X<<Y>>

Пример 15.3

Показать, как Польз. 1, зная только общедоступный ключ CA (корень), может получить верифицированную копию общедоступного ключа Польз. 3.

Решение

Пользователь 3 передает сертификат по цепочке CA «CA1» и CA1 «Польз. 3», к Польз. 3.

а. Польз.1 подтверждает CA «CA1», используя общедоступный ключ CA.

б. Польз.1 извлекает общедоступный ключ CA1 от CA «CA1».

в. Польз.1 подтверждает CA«Польз. 3», используя общедоступный ключ CA1.

г. Польз.1извлекает общедоступный ключ Польз. 3 из CA «Польз. 3».

Пример 15.4

Некоторые Web-браузеры, такие, как Netscape и Internet Explorer, содержат множество сертификатов, полученных от независимых Центров сертификации (корней) без единственного корня высокого уровня администрации, который мог бы сертифицировать каждый корень. Можно найти список этих корней в Internet Explorer по следующему маршруту: *Инструментальные средства / Опции Интернета / Содержание, Сертификат / Корень доверия* (используя «раскрывающиеся строки»). Пользователь тогда может выбрать любой корень и ознакомиться с его сертификатом.

Модель «каждый с каждым». Иерархическая модель может работать для организации или маленькой группы людей. Большие группы, возможно, нуждаются в нескольких иерархических структурах, соединенных вместе. Один из методов состоит в том, чтобы использовать модель «каждый с каждым» для соединения корней вместе. В этой модели каждый корень связан с каждым другим корнем, как это показано на рис. 15.21.

Рисунок 15.21 показывает, что структура **«каждый с каждым»** соединяет вместе только корни; каждый корень имеет свою собственную иерархическую структуру, изображенную треугольником. Сертификация между корнями — перекрестные сертификаты; каждый корень сертифицирует все другие корни, что означает, что есть N $(N - 1)$ сертификатов. На рис. 15.21 — 4 узла, так что надо $4 \times 3 = 12$ удостоверений. Обратите внимание, что каждая линия имеет двойную стрелку и представляет два сертификата.

Пример 15.5

Алиса имеет дело с администрацией Root 1; Боб имеет дело с администрацией Root 4. Показать, как Алиса может получить верифицированный общедоступный ключ Боба.

Решение

Боб передает цепочку сертификатов от Root 4 к Бобу. Алиса смотрит каталог Root 1, чтобы найти Root 1 сертификаты «Root l» и Root 1 «Root 4» . Используя процесс, показанный на рис. 15.21, Алиса может верифицировать общедоступный ключ Боба.

Сеть доверия. Эта модель, которая используется в PGP (Pretty Good Privacy — очень хорошая конфиденциальность), службе безопасности для электронной почты, рассматривается в лекции 16.

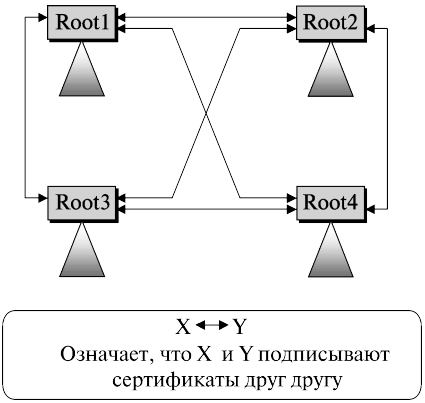


Рис. 15.21. Модель «каждый с каждым»

15.5. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Управление симметричными и асимметричными ключами рассматривается в [Sti06], [KPS02], [Sta06], [Rhe03] и [PHS03].

Сайты

Нижеследующие сайты дают больше информации о темах, обсужденных в этой лекции.

http://en.wikipedia.org/wiki/Needham-Schroeder
http://en.wikipedia.org/wiki/Otway-Rees
http://en.wikipedia.org/wiki/Kerbero s_%28protocol%29
en.wikipedia.org/wiki/Diffie-Hellman
www.ietf.org/rfc/rfc2631.txt

15.6. Итоги

- Для работы в криптографии с симметричными ключами необходимы два общедоступных ключа засекречивания (по одному каждой стороне). Если N людей связались друг с другом, необходимы $N(N-1)/2$ ключей. Число ключей — не единственная проблема; другая проблема — это распределение ключей.

- Практическое решение распределения ключей — использование третьего лица, которому доверяют, называемого Центром распределения ключей (KDC). KDC может создать ключ сеанса (временный) между Алисой и Бобом, используя их ключи связи с центром. Ключи Алисы и Боба применяются, чтобы подтвердить центру подлинность Алисы и Боба.
- Были предложены несколько различных подходов для создания ключей сеанса, которые используют идеи, рассмотренные в лекции 14 для установления подлинности объекта. Два из самых изящных — Протокол Ниидома-Шрёдера, который является основой для многих других протоколов, и Протокол Отвея-Рисса.
- Цербер является и опознавательным протоколом, и KDC. Несколько систем, включая Windows 2000, используют Цербер. В протоколе Цербер участвуют три сервера: опознавательный сервер (AS), сервер, предоставляющий билет (TGS), и реальный сервер данных.
- Алиса и Боб могут создать ключ сеанса между собой, не используя KDC. Этот метод создания ключа сеанса называется соглашением с симметричным ключом. Мы рассмотрели два метода: Диффи-Хеллмана и «от станции к станции». Первый чувствителен к атаке «посредника»; второй — нет.
- Открытые ключи, подобно секретным ключам, должны быть распределены для использования. Администрация по сертификации (CA) обеспечивает сертификатами как доказательством собственности общедоступного ключа. X.509 — рекомендация, которая определяет структуру сертификата.
- Инфраструктура открытого ключа (PKI) — модель для создания, распределения и аннулирования удостоверений, основанных на рекомендации X.509. Группа Инженерной Поддержки Интернет (IETF) создала Инфраструктуру Открытого ключа X.509 (PKIX). Режимы работы PKI включают издание сертификатов, хранение секретного ключа, услуги в соответствии с другими протоколам и управление доступом.
- PKI также определяет модели доверия, отношения между администрациями, выдающими сертификаты. Три модели доверия, рассмотренные в этой лекции, являются иерархическими, «каждый с каждым» и сетью доверия.

15.7. Набор для практики

Обзорные вопросы

1. Перечислите режимы работы KDC.
2. Дайте определение ключа сеанса и покажите, как KDC может создать ключ сеанса между Алисой и Бобом.
3. Дайте определение протокола Цербер и назовите его серверы. Кратко объясните режимы работы каждого сервера.
4. Дайте определение протокола Диффи-Хеллмана и его цель.
5. Дайте определение атаки «посредника».
6. Дайте определение протокола «от станции к станции» и объясните его цель.
7. Дайте определение центра сертификации (CA) и расскажите о его отношении к криптографии общедоступного ключа.

8. Дайте определение рекомендации X.509 и разъясните ее цель.
9. Перечислите режимы работы PKI.
10. Дайте определение модели доверия и рассмотрите некоторые варианты этой модели, которые обсуждались в этой лекции.

Упражнения

11. На рис. 15.4: что случится, если билет для Боба будет зашифрован не на шаге 2 с ключом K_B, но зашифрован вместо этого K_{AB} на шаге 3?
12. Почему нужны четыре nonce в протоколе Ниидома-Шрёдера?
13. Как KDC в протоколе Ниидома-Шрёдера аутентифицирует (удостоверяет) Алису? Как KDC аутентифицирует (удостоверяет) Боба? Как Алиса аутентифицирует (удостоверяет) Боба? Как Боб аутентифицирует (удостоверяет) Алису?
14. Объясните, почему в протоколе Ниидома-Шрёдера Алиса — сторона, которая находится в контакте с KDC, а в протоколе Отвея-Рисса Боб — сторона, которая находится в контакте с KDC.
15. В протоколе Ниидома-Шрёдера есть четыре (R_A, R_B, R_1 и R_2), а в протоколе Отвея-Рисса — только три nonce (R_A, R_B и R). Объясните, почему есть потребность в одном дополнительном once R_2 в первом протоколе.
16. Почему мы нуждаемся только в одной метке времени в протоколе Цербер вместо четырех nonce, как в протоколе Ниидома-Шрёдера, или трех once, как в протоколе Отвея-Рисса?
17. В протоколе Диффи-Хеллмана $g = 7$, $p = 23$, $x = 3$, и $y = 5$.
    а. Какое значение имеет симметричный ключ?
    б. Какие значения имеют R_1 и R_2?
18. Что случится в протоколе Диффи-Хеллмана, если x и y имеют одно и то же значение, то есть Алиса и Боб случайно выбрали одно и то же число? R_1 и R_2 те же самые? Ключи сеанса, вычисленные Алисой и Бобом, имеют одно и то же значение? Приведите пример, чтобы доказать ваши выводы.
19. При тривиальной (не гарантирующей безопасность) смене ключа Диффи-Хеллмана $p = 53$. Найдите соответствующее значение для g.
20. В протоколе «от станции к станции» покажите, что если опознавательный код приемника удален из подписи, протокол становится уязвимым к атаке «посредника».
21. Обсудите заслуживающий доверия корень сертификации, применяющий браузеры.

Часть 4. Безопасность сети

Четвертая часть этой книги сосредоточена на предмете, который является окончательной целью нашей книги: использование криптографии для создания безопасных сетей. Эта часть предполагает, что читатель имеет представление об архитектуре сети Internet и наборе протоколов TCP/IP. Для некоторого введения в предмет можно использовать приложение C как краткий обзор. В списке литературы для дальнейшего изучения читателям указана книга [For06]. Каждая лекция в этой части посвящена обсуждению безопасности на одном из трех уровней набора протоколов TCP/IP: прикладном, транспортным и сетевом уровне. Лекция 16 обсуждает безопасность на прикладном уровне, лекция 17 — на транспортном уровне и лекция 18 — на сетевом уровне.

Лекция 16: Безопасность на прикладном уровне — PGP и S/MIME

Лекция 16 обсуждает два протокола, которые обеспечивают безопасность для электронной почты (e-mail). Очень хорошую конфиденциальность дает PGP (Pretty Good Privacy) — протокол, который обычно используется для персональной почтовой станции.

Безопасное/Многоцелевое расширение почты (S / MIME Secure / Multipurpose Internet Mail Extension) — протокол, который обычно применяется в коммерческих почтовых системах.

Лекция 17: Безопасность на транспортном уровне — SSL и TSL

Лекция 17 сначала доказывает потребность в службах безопасности на транспортном уровне модели Internet. Затем показывает, как может быть обеспечена безопасность на транспортном уровне, используя один из двух протоколов: протокол уровня безопасных разъемов (SSL — Secure Socket Layer) и протокол безопасности транспортного уровня (TLS — Transport Layer Security). Второй протокол — новая версия первого.

Лекция 18: Безопасность на сетевом уровне — IPSec

Лекция посвящена единственному общему протоколу безопасности на сетевом уровне: протоколу безопасного туннелирования IP-пакетов (IPSec — IP Security protocol). Лекция определяет архитектуру IPSec и обсуждает приложение IPSec в транспортных и туннельных режимах. Лекция также обсуждает другие вспомогательные протоколы, например, Интернет-протокол шифрования и идентификации (IKE — Internet Key Exchange), который используется IPSec, определяет станцию ключа Internet, и объясняет, как это применяется IPSec.

Лекция 16. Безопасность на прикладном уровне: PGP И S/MIME

Цели и содержание

Эта лекция имеет несколько целей.

- Объяснить общую структуру почтовой прикладной программы.
- Обсудить, как PGP может обеспечить службы безопасности для электронной почты.
- Обсудить, как S/MIME может обеспечить службы безопасности для электронной почты.
- Определить механизм доверия в PGP и в S/MIME.
- Показать структуру обмена сообщениями в PGP и S/MIME.

Мы обсудим два протокола службы обеспечения безопасности для электронных почт: Очень хорошая конфиденциальность (PGP) и Безопасное/Многоцелевое расширение почты (S/MIME). Понимание каждого из этих протоколов требует общего понимания почтовой системы. Сначала мы поговорим о структуре электронной почты. Затем покажем, как PGP и S/MIME могут дополнить службы безопасности этой структуры. Особое внимание уделяется тому, как PGP и S/MIME могут менять криптографические алгоритмы, ключи засекречивания и сертификаты, не устанавливая сеанс между Алисой и Бобом.

16.1. Электронная почта

Сначала обсудим электронную почту (e-MAIL) как систему.

Архитектура E-MAIL

Рисунок 16.1 показывает самый общий сценарий при одностороннем почтовом обмене. Предположим, что Алиса работает в организации, которую обслуживает почтовый сервер. Каждый служащий связан с почтовым сервером через местную сеть связи (LAN). Или, альтернативно, Алиса могла быть связана с почтовым сервером провайдера (ISP — Information Server Provider) через региональную сеть связи (телефонная линия или кабельная линия). Боб находится также в одной из вышеупомянутых ситуаций.

Администратор почтового сервера на стороне Алисы создает систему организации очереди, которая передает сообщения электронной почты в Интернет одно за другим. Администратор почтового сервера на стороне Боба создает почтовый ящик для каждого пользователя, подключенного к серверу. Почтовый ящик держит полученные сообщения, пока они не будут приняты получателем.

Когда Алиса должна передать сообщение Бобу, она вызывает **агента пользователя** программы (UA), чтобы подготовить сообщение. Она использует другую программу — **почтовый агент** (MTA), — чтобы передать сообщение серверу почты на ее стороне. Обратите внимание, что MTA — программа «клиент-сервер» с клиентом, который, установлен в компьютере Алисы и на сервере, который установлен на сервере почты.

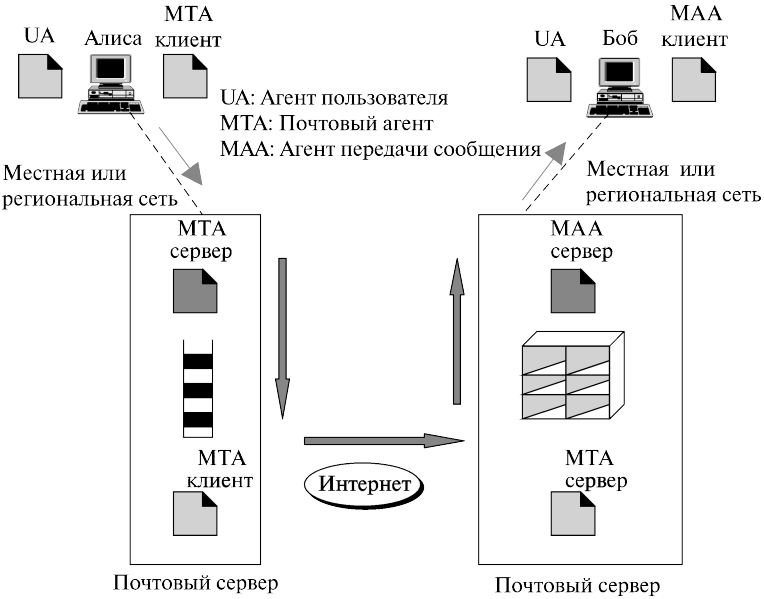


Рис. 16.1. Архитектура e-MAIL

Сообщение, полученное на сервере почты на стороне Алисы, поставлено в очередь со всеми другими сообщениями; очередь создается для каждого направления в соответствующий пункт назначения. В случае Алисы ее сообщение поступает на сервер почты Боба. Клиент-сервер MTA отвечает за почтовую передачу между этими двумя серверами. Когда сообщение достигает сервера почты пункта назначения, оно попадает в почтовый ящик Боба в виде специального файла, который сохраняет сообщение, пока оно не будет извлечено Бобом.

Когда Боб должен извлечь свои сообщения (файл), включая сообщение, передаваемое Алисой, он вызывает другую программу, которую мы называем **агентом доступа к сообщению** (MAA — Massage Access Agent). MAA также разработан как программа «клиент-сервер», установленная и в компьютере Боба, и на сервере почты.

Есть несколько важных положений об архитектуре почтовой системы.

а. Передаваемая электронная почта от Алисы к Бобу накапливается в памяти. Алиса может передать электронную почту сегодня; Боб, будучи занятым, может проверить свою электронную почту три дня спустя. Всё это время электронная почта сохраняется в почтовом ящике Боба, пока он не возьмет ее.

б. Главная связь между Алисой и Бобом проходит две прикладных программы: MTA-клиента в компьютере Алисы и MAA-клиента в компьютере Боба.

в. MTA-программа клиента — *принимающая* программа; клиент помещает сообщение, когда Алиса должна его передать. Программа клиента MAA — *выдающая* программа; клиент перемещает сообщение, когда Боб готов извлечь свою электронную почту.

г. Алиса и Боб не могут непосредственно в данный момент связать MTA-клиента, используемого на стороне передатчика, и MTA-сервер, используемый на стороне приемника. Поэтому требуется, чтобы MTA-сервер функционировал все время, потому что Боб не знает, когда сообщение прибудет. Это практически невозможно, потому что Боб, вероятно, выключает свой компьютер, когда не нуждается в нем.

Почтовая безопасность

Передача электронной почты — одноразовая активность. Характер этой активности отличается от тех, которые мы увидим в следующих двух лекциях. В IPSec или SSL мы предполагаем, что две стороны создают сеанс между собой и обмениваются данными в обоих направлениях. В электронной почте нет никакого сеанса. Алиса и Боб не могут создать сеанс. Алиса передает сообщение Бобу, а когда-нибудь позже Боб читает сообщение и, может быть, сразу не способен передать ответ. Мы будем рассматривать безопасность однонаправленного сообщения, потому что Алиса передает сообщения Бобу полностью независимо от того, что Боб передает Алисе.

Криптографические алгоритмы

Если электронная почта — одноразовое активное действие, как могут передатчик и приемник договориться о криптографическом алгоритме, который они будут использовать для почтовой безопасности? Если нет сеанса и процедуры установления связи, чтобы договориться об алгоритмах относительно шифрования/дешифрования и хэширования, как приемник может знать, какой алгоритм выбран передатчиком для каждого сообщения?

Имеется одно решение для основного протокола — выбирать один алгоритм из заданного множества для каждой криптографической операции и заставить Алису использовать только эти алгоритмы. Это решение очень сужает возможности и ограничивает действия двух сторон.

Лучшее решение для основного протокола — определить множество алгоритмов для каждой операции, которые пользователь может применить в его системе. Алиса включает название (или идентификаторы) алгоритмов, которые она использовала в электронной почте. Например, Алиса может выбрать трехкратный DES для шифрования/дешифрования и MD5 для хэширования. Когда Алиса передает сообщение Бобу, она включает соответствующие идентификаторы для трехкратного DES и MD5 в свое сообщение. Боб получает сообщение и сначала извлекает идентификаторы. Тогда он знает, какой алгоритм использовать для дешифрования и какой — для хэширования.

Для безопасности почты передатчик сообщения должен включить в него название или идентификаторы алгоритмов, используемых в сообщении.

Криптографическая секретность

Та же самая проблема, что и для криптографических алгоритмов, существует и для криптографической секретности (ключи). Если нет переговоров, как

эти две стороны могут установить принципы секретности между собой? Алиса и Боб могли использовать асимметрично-ключевые алгоритмы для установления подлинности и шифрования, которое не требует установления симметричного ключа. Однако, как мы видели, использование асимметрично-ключевых алгоритмов очень неэффективно для шифрования/дешифрования длинного сообщения.

Большинство почтовых протоколов безопасности сегодня требует, чтобы шифрование/дешифрование было сделано с использованием алгоритма с симметричными ключами и одноразовым ключом засекречивания, передаваемого с сообщением. Алиса может создать ключ засекречивания и переслать его с сообщением, которое она передает Бобу. Чтобы защитить ключ засекречивания от перехвата Евой, ключ засекречивания зашифрован открытым ключом Боба. Другими словами, сам ключ засекречивания зашифрован.

Для безопасности почты шифрование/дешифрование делается с использованием симметричного ключевого алгоритма, но ключ засекречивания для расшифровки сообщения зашифрован открытым ключом приемника и передается с сообщением.

Сертификаты

Прежде чем мы обсудим любой почтовый протокол безопасности, нужно рассмотреть в еще одну проблему: некоторые очевидные алгоритмы общедоступного ключа, которые должны использоваться для почтовой безопасности. Например, мы должны зашифровать ключ засекречивания или подписать сообщение. Чтобы зашифровать ключ засекречивания, Алиса нуждается в открытом ключе Боба; для подписи и верификации сообщения Боб нуждается в открытом ключе Алисы. Так что для того, чтобы посылать маленькое заверенное и конфиденциальное сообщение, необходимы два открытых ключа. Как Алиса может быть уверена в открытом ключе Боба, и как Боб может быть уверен в открытом ключе Алисы? Каждый почтовый протокол безопасности имеет различные методы сертификации ключей.

16.2. PGP

Первый протокол, который мы обсудим в этой лекции, называется **Очень хорошей конфиденциальностью** (PGP — Pretty Good Privacy). PGP был изобретен Филом Цимерманном (Phil Zimmermann), чтобы обеспечить секретность, целостность и установление подлинности электронной почты. PGP может применяться, чтобы создать безопасное почтовое сообщение или надежно сохранить файл для будущего извлечения.

Сценарии

Сначала обсудим общую идею PGP, продвигаясь от простого сценария к сложному. Мы используем термин «Данные», чтобы указать сообщение или файл для обработки.

Открытый текст

Самый простой сценарий — это передать почтовое сообщение (или накопленный файл) в исходном тексте, как это показано на рис. 16.2. В этом сценарии нет сохранения целостности сообщения или конфиденциальности. Алиса (передатчик) составляет сообщение и передает его Бобу (приемнику). Сообщение сохраняется в почтовом ящике Боба, пока не будет извлечено им.

Рис. 16.2. Сообщение с обычным текстом

Целостность сообщения

Вероятно, следующее усовершенствование должно позволить Алисе подписывать сообщение. Алиса создает дайджест сообщения и подписывает его своим секретным ключом. Когда Боб получает сообщение, он проверяет его, используя открытый ключ Алисы. Для этого сценария необходимы два ключа. Алиса должна знать свой секретный ключ; Боб должен знать открытый ключ Алисы. Рисунок 16.3 показывает ситуацию.

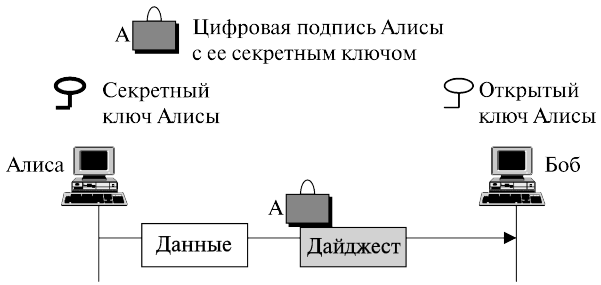


Рис. 16.3. Сообщение с подтверждением подлинности передатчика

Сжатие

Дальнейшее усовершенствование позволяет сжать сообщение и дайджест, чтобы сделать пакет более компактным. Это усовершенствование не имеет никаких преимуществ с точки зрения безопасности, но существенно уменьшает трафик. Рисунок 16.4 показывает новый сценарий.

Конфиденциальность с одноразовым ключом сеанса

Как мы уже говорили раньше, конфиденциальность в почтовой системе может быть достигнута за счет применения обычного шифрования одноразовым ключом сеанса. Алиса может создать ключ сеанса, использовать ключ сеанса для

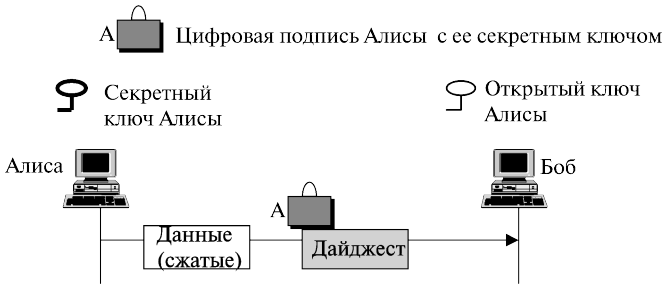


Рис. 16.4. Сообщение со сжатым текстом

шифрования сообщения и дайджеста и передать ключ непосредственно с сообщением. Однако для защиты ключа сеанса Алиса зашифровала его открытым ключом Боба. Рисунок 16.5 показывает ситуацию, когда Боб получает пакет, он сначала расшифровывает ключ, используя свой секретный ключ, чтобы извлечь этот совместный ключ сессии. Затем он использует ключ сеанса, чтобы расшифровать остальную часть сообщения. После расширения (декомпрессации) остальной части сообщения Боб создает дайджест сообщения и проверяет, равен ли он дайджесту, передаваемому Алисой. Если дайджесты равны, то сообщение подлинно.

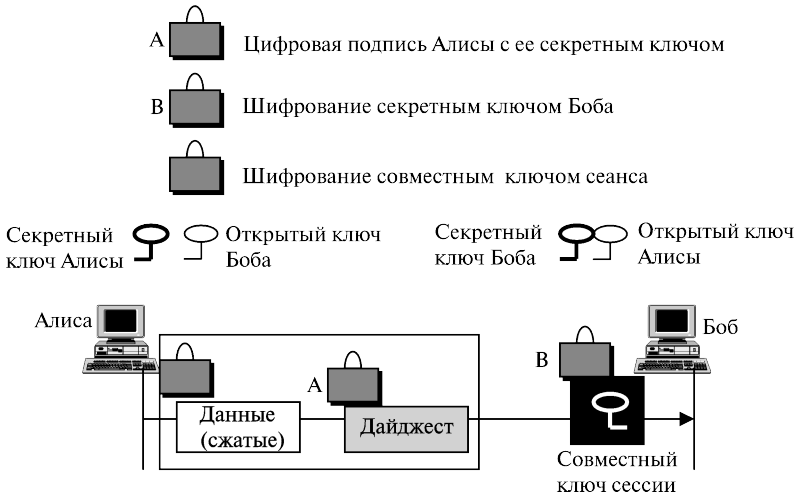


Рис. 16.5. Конфиденциальное сообщение

Преобразование кода

Другая услуга, предоставляемая PGP, — преобразование кода. Большинство почтовых систем позволяет передать сообщение, состоящее только из символов ASCII. Чтобы перевести символы, не входящие в множество ASCII, PGP использу-

ет преобразование Radix-64. Каждый посылаемый символ (после того как будет зашифрован) преобразуется в код Radix-64, который мы обсудим позже в этой лекции.

Сегментация

PGP разрешает сегментацию сообщения после того, как оно было преобразовано к Radix-64, чтобы сделать каждый переданный модуль одинаковым по размеру, в соответствии с основным почтовым протоколом.

Кольца ключей

Во всех предыдущих сценариях мы предполагали, что Алиса должна передать сообщение только Бобу. Но это не единственный вариант передачи. Возможно, Алиса должна передать сообщения многим людям; ей нужны **кольца ключей.** В этом случае Алиса нуждается в кольце открытых ключей, включающем ключ(и), принадлежащий каждому человеку, которому Алиса должна передать или от которого может получить сообщение. PGP-разработчики определили кольцо частных/открытых ключей. Алиса может иметь причины время от времени изменить пару ключей. Другой случай: Алиса, возможно, желает, чтобы ее ключи соответствовали различным группам людей (друзья, коллеги и так далее). Поэтому каждый пользователь должен иметь два множества колец: кольцо частных/открытых ключей и кольца открытых ключей других людей. Рисунок 16.6 показывает сообщество четырех человек, где каждый имеет кольцо пары частных/открытых ключей и, в то же самое время, изображены кольца открытых ключей, принадлежащих другим людям в этом сообществе.

Алиса, например, имеет несколько пар частных/открытых ключей, принадлежащих ей, и открытые ключи, принадлежащие другим людям. Обратите внимание, что каждый может иметь больше чем один открытый ключ. Могут возникнуть два случая.

1. Алиса должна передать сообщение другому человеку в сообществе.
   а. Она использует свой секретный ключ, чтобы подписать дайджест.
   б. Она использует открытый ключ приемника, чтобы зашифровать недавно созданный ключ сеанса.
   в. Она шифрует сообщение и подписанный дайджест с созданным ключом сеанса.
2. Алиса получает сообщение от другого человека, состоящего в этом сообществе.
   а. Она использует свой секретный ключ, чтобы расшифровать ключ сеанса.
   б. Она использует свой ключ сеанса, чтобы расшифровать сообщение и дайджест.
   в. Она использует свой открытый ключ, чтобы проверить дайджест.

PGP-алгоритмы

В PGP используются нижеследующие алгоритмы.

Алгоритмы открытого ключа. Алгоритмы открытого ключа, которые применяются, чтобы подписать дайджесты или зашифровать сообщения, перечислены в таблице 16.1.

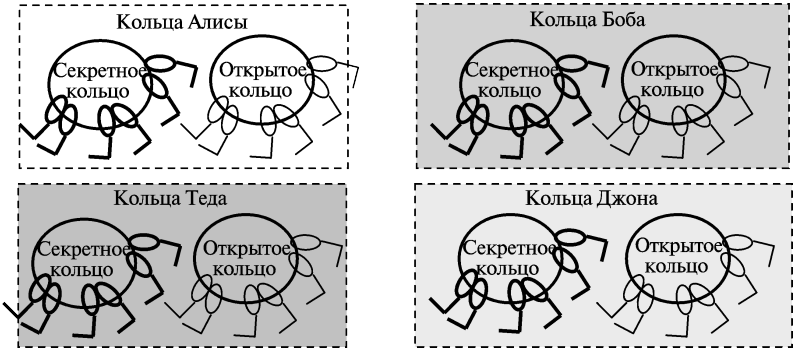


Рис. 16.6. Кольца ключей в PGP

Таблица 16.1. Алгоритмы открытого ключа

ID	Описание
1	RSA (для шифрования и подписи)
2	RSA (только для шифрования)
3	RSA (только для подписи)
16	Эль-Гамаль (только для шифрования)
17	DSS
18	Зарезервировано для эллиптической кривой
19	Зарезервировано для ECDSA
20	Эль-Гамаль (для шифрования и подписи)
21	Зарезервировано для Диффи-Хеллмана
100-110	Секретные алгоритмы

Алгоритмы симметричного ключа. Алгоритмы симметричного ключа, которые используются для удобного шифрования, показаны в таблице 16.2.

Таблица 16.2. Алгоритмы с симметричными ключами

ID	Описание
0	Не зашифровано
1	IDEA
2	Тройной DES
3	CAST-128
4	Blowfish
5	SAFER-SK128
6	Зарезервировано для DES/SK
7	Зарезервировано для AES-128
8	Зарезервировано для AES-192
9	Зарезервировано для AES-256
100-110	Частные алгоритмы

Хэш-алгоритмы. Хэш-алгоритмы, которые используются для создания хэша в PGP, показаны в таблице 16.3.

Таблица 16.3. Хэш-алгоритмы

ID	Описание
1	MD5
2	SHA-1
3	RIPE-MD/160
4	Зарезервировано для SHA двойной ширины
5	MD2
6	TIGER/192
7	Зарезервировано для HAVAL
100-110	Частные алгоритмы

Алгоритмы сжатия. Алгоритмы сжатия, которые используются для сжатия текста, показаны в таблице 16.4.

Таблица 16.4. Методы сжатия

ID	Описание
0	Без сжатия
1	ZIP
2	ZLIP
100-110	Частные методы

PGP-сертификаты

PGP, подобно другим протоколам, как мы видели до сих пор в других системах, использует сертификаты, чтобы подтвердить подлинность открытых ключей. Однако в этом случае процесс полностью отличается от уже рассмотренных

Сертификаты X.509

Протоколы, которые применяют сертификаты X.509, зависят от иерархической структуры доверия. Есть заранее заданная цепочка доверия от корня до любого сертификата. Каждый пользователь полностью доверяет администрации CA на заданном уровне корня. Корень вырабатывает сертификаты для второго уровня, второй уровень CA вырабатывает сертификаты для третьего уровня, и так далее. Каждый партнер, который хочет быть доверенным для выдачи сертификатов, должен быть представлен на одной из ветвей дерева какого-либо CA. Если Алиса не доверяет выпускающему сертификат для Боба, она может обратиться к администрации более высокого уровня вплоть до корня (которому необходимо доверять для того, чтобы система работала). Другими словами, есть один-единственный путь к сертификату от CA, которому полностью доверяют.

В X.509 есть единственный путь от администрации, которому полностью доверяют, к любому сертификату.

PGP-сертификаты

В PGP нет никакой потребности в CA — любой в кольце может подписать сертификат для кого-либо другого в кольце. Боб может подписать сертификат для Теда, Джона, Алисы и так далее. В PGP нет иерархии доверия; нет дерева. Отсутствие иерархической структуры может привести к тому, что Тед может иметь один сертификат от Боба и другой сертификат — от Джона. Если Алиса хочет исследовать линию получения сертификатов для Теда, у нее есть два пути: начинать от Боба и начинать от Джона. Интересно, что Алиса может полностью доверять Бобу, но только частично доверяет Джону. Может быть много путей доверия, приводящих к сертификату от администрации, которой доверяют полностью или частично. В PGP выпускающего сертификат обычно называют *поручителем*.

В PGP может быть много путей доверия, приводящих к сертификату от администрации, которой доверяют полностью или частично.

Доверие и законность

Полная операция PGP базируется на доверии поручителя, доверии сертификата и законности открытых ключей.

Уровни доверия поручителя. При отсутствии центральной администрации очевидно, что кольцо не может быть очень большим, если каждый пользователь в кольце PGP-пользователей не имеет полного доверия к каждому члену сообщества. (Даже в реальной жизни мы не можем полностью доверять каждому человеку, которого мы знаем.). Чтобы решить эту проблему, PGP позволяет различные уровни доверия. Число уровней, главным образом, зависит от реализации, но для простоты давайте назначим три уровня доверия к любому поручителю: *никакой*, *частичный* и *полный*. Уровень доверия поручителя определяет уровни доверия, выработанные поручителем для других людей в кольце. Например, Алиса может полностью доверять Бобу, частично доверять Анне и не доверять Джону. Вообще, в PGP нет механизма, чтобы решить, как принять решение поручителя о заслуживающем доверия партнере; это может сделать только пользователь.

Уровни доверия сертификата, Когда Алиса получает сертификат от поручителя, она хранит сертификат под именем субъекта (сертифицированный объект) и назначает уровень доверия этому сертификату. Уровень доверия сертификата обычно тот же, что и уровень доверия поручителя, который выдал сертификат. Предположим, что Алиса полностью доверяет Бобу, частично доверяет Анне и Джанетт и не имеет никакого доверия Джону. Могут быть следующие сценарии.

1. Боб вырабатывает два сертификата, один для Линды (с открытым ключом K1) и один для Лесли (с открытым ключом K2). Алиса хранит открытый ключ и сертификат для Линды под названием «Линда» и назначает *полный* уровень доверия этому сертификату. Алиса также хранит сертификат и открытый ключ для Лесли под названием «Лесли» и назначает полный уровень доверия этому сертификату.

2. Анна вырабатывает сертификат для Джона (с открытым ключом К3). Алиса хранит этот сертификат и открытый ключ под названием «Джон», но назначает уровень для этого сертификата — *частичный*.
3. Джанетт вырабатывает два сертификата: один для Джона (с общедоступным ключом К3) и один для Ли (с открытым ключом К4). Алиса хранит сертификат Джона под его именем и сертификатом «Ли» под этим именем (Ли), каждый с *частичным* уровнем доверия. Обратите внимание, что Джон теперь имеет два сертификата: один от Анны и один от Джанетт, каждый с *частичным* уровнем доверия.
4. Джон вырабатывает сертификат для Лиз. Алиса может отказаться или сохранить этот сертификат с надписью «*не доверяю*».

Законность ключей. Цель использования поручителя и сертификата доверия — определить законность открытого ключа. Алиса должна знать, насколько законны открытые ключи Боба, Джона, Лиз, Анны и так далее. PGP определяет очень ясную процедуру для того, чтобы определить законность ключей. Уровень законности ключей для пользователя — это взвешенные уровни доверия пользователя. Например, предположим, что мы назначаем следующие веса на уровни доверия сертификата:

1. вес 0 — сертификату, которому не доверяют;
2. вес 1/2 — сертификату с частичным доверием;
3. вес 1 — сертификату с полным доверием.

Тогда для полного доверия объекту Алиса нуждается в одном сертификате, которому доверяет полностью, или в двух частичных сертификатах доверия для этого объекта. Например, Алиса может использовать открытый ключ Джона в предыдущем сценарии, потому что и Анна, и Джанетт выработали сертификат для Джона, каждый с уровнем доверия сертификата 1/2. Обратите внимание, что законность открытого ключа, принадлежащего объекту, не имеет никакого отношения к уровню доверия других людей к этому человеку. Хотя Боб может использовать открытый ключ Джона, чтобы передать сообщение ему, Алиса может не принять ни одного сертификата, выпущенного Джоном, потому что для Алисы Джон имеет уровень «*нет доверия*».

Старт кольца

Вы, возможно, нашли серьезную проблему в вышеупомянутом обсуждении. А если никто не передал сертификат, что он полностью или частично доверяет объекту? Например, как можно решить проблему законности общедоступного ключа Боба, если никто не передал сертификат Боба? В PGP законность ключа, которому доверяют, или объекта, которому частично доверяют, может быть также определена другими методами.

1. Алиса может физически получить открытый ключ Боба. Например, Алиса и Боб могут встретиться лично. И при этом обменяться открытым ключом, написанным на обрывке бумажки или на диске.
2. Если голос Боба распознаваем Алисой, Алиса может по телефону вызвать его и получить его открытый ключ.
3. Лучшее решение, предложенное PGP для Боба, — передать его открытый ключ Алисе электронной почтой. И Алиса, и Боб делают дайджест ключа

16 байтов MD5 (или 20 байтов SHA-1). Дайджест обычно отображается как восемь групп по 4 цифры (или десять групп по 4 цифры) в шестнадцатеричном виде и называется *отпечатком пальца*. Алиса может тогда вызвать Боба и проверить *отпечаток пальца* по телефону. Если ключ изменился или изменен во время почтовой передачи, два отпечатка пальца не будут соответствовать друг другу. Для того чтобы сделать проверку более удобной, PGP создал список слов, каждое из которых представляет комбинацию из 4-х цифр. Когда Алиса вызывает Боба, Боб может объявить эти восемь слов (или десять слов) для Алисы. Слова тщательно выбраны PGP, чтобы избежать путаницы при произношении; например, если *печь* находится в списке, то слово *речь* не включается в список.

4. В PGP ничто не препятствует Алисе получать открытый ключ Боба от CA по отдельной процедуре. Она может тогда вставить открытый ключ в кольцо открытого ключа.

Таблицы кольца ключей

Каждый пользователь, такой как Алиса, сохраняет множество двух колец ключей: одно кольцо секретного ключа и одно кольцо открытого ключа. PGP определяет структуру для каждого из этих колец ключей в форме таблицы.

Рисунок 16.7 показывает формат таблицы кольца секретного ключа.

Секретное кольцо

ID Пользователя	ID Ключа	Открытый ключ	Секретный ключ	Метка времени
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

Рис. 16.7. Формат таблицы кольца секретного ключа.

- **Пользовательский ID.** Пользовательский ID — обычно адрес электронной почты пользователя. Однако пользователь может определять уникальный адрес электронной почты или псевдоним для каждой ключевой пары. Таблица перечисляет пользовательские ID, связанные с каждой парой.
- **ID ключа.** Этот столбец уникально определяет открытый ключ среди открытых ключей пользователя. В PGP ID ключа для каждой пары — первые (самые младшие) 64 бита открытого ключа. Другими словами, ключ ID вычисляется как ключ mod 2^{64}). Ключ ID необходим для работы PGP, потому что Боб может иметь несколько открытых ключей, принадлежащих Алисе, в его кольце открытого ключа. Когда он получает сообщение от Алисы, Боб должен знать, какой ID ключа надо использовать, чтобы проверить сообщение. Ключ ID, который передают с сообщением, как мы увидим, дает возможность Бобу применить заданный открытый ключ для Алисы из своего открытого кольца. Можно спросить, почему не передают полностью открытый ключ. Ответ на это вопрос такой: в криптографии открытого ключа размер открытого ключа может быть очень большой. Передавая только 8 байт, мы уменьшаем размер сообщения.

- **Открытый ключ.** Этот столбец только перечисляет открытые ключи, принадлежащие конкретной паре «секретный ключ / открытый ключ».
- **Зашифрованный секретный ключ.** Этот столбец показывает зашифрованное значение секретного ключа в паре «секретный ключ / открытый ключ». Хотя Алиса — единственный человек, имеющий доступ к своему секретному кольцу, PGP сохраняет только зашифрованную версию секретного ключа. Мы увидим позже, как зашифровывается и расшифровывается секретный ключ.
- **Метка времени.** Этот столбец содержит время и дату создания пары ключей. Это помогает пользователю решать, когда произвести чистку старых пар и когда создавать новые пары.

Пример 16.1

Покажем пример таблицы кольца секретного ключа для Алисы. Мы предполагаем, что Алиса имеет только два пользовательских ID: *alice@some.com* и *alice@anet.net.* Мы также предполагаем, что Алиса имеет два множества пар секретных/открытых ключей, один для каждого пользовательского ID. Таблица 16.5 показывает таблицу кольца секретных ключей для Алисы.

Таблица 16.5. Таблица кольца секретных ключей для примера 16.1

ID пользователя	Ключ ID	Открытый	Секретный ключ ключ	Метка времени шифрования
alice@anet.net	AB13...45	AB13...45...59	**32452398...23**	031505-16:23
alice@some.com	FA23...12	FA23...12...22	**564A4923...23**	031504-08:11

Обратите внимание, что хотя значения ключа ID, открытого ключа и секретного ключа показаны в шестнадцатеричном виде, для метки времени используется формат «*месяц – день – год*». Для метки времени эти форматы служат только для фиксации даты и могут быть различны. Например, в России утвержден формат «*день – месяц – год*».

Рисунок 16.8 показывает формат таблицы кольца общедоступного ключа.

ID пользователя	Ключ ID	Ключ открытый	Доверие к поставщику	Сертификат (ы)	Сертификат доверия	Законность ключа	Метка времени
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Рис. 16.8. Формат таблицы кольца открытого ключа

- **Пользовательский ID.** Как и в таблице 16.7 кольца секретного ключа, пользовательский ID — обычно адрес электронной почты объекта.
- **Ключ ID.** Как и в таблице 16.7 кольца секретного ключа, ключ ID — первые (самые младшие) 64 бита общедоступного ключа.
- **Открытый ключ.** Это — открытый ключ объекта.

- **Доверие к поставщику.** Этот столбец определяет уровень доверия к поставщику. В большинстве реализаций он может иметь только одно из трех значений: *не доверяю, доверяю частично* или *доверяю полностью.*
- **Сертификат(ы).** Этот столбец содержит сертификат или сертификаты, подписанные другим объектам для этого объекта. Пользовательский ID может иметь больше чем один сертификат.
- **Сертификат(ы) доверия.** Этот столбец представляет сертификат доверия (или просто доверие). Если Анна передает сертификат за Джона, PGP ищет вход строки Анны, находит значение доверенного поставщика для Анны, копирует это значение и вставляет его в поле сертификата доверия для Джона.
- **Законность ключей.** Это значение вычисляется PGP, на основе значения сертификата доверия и заранее заданного веса для каждого сертификата доверия.
- **Метка времени.** Этот столбец содержит дату и время создания столбца.

Пример 16.2

Ряд шагов показывает, как формируется таблица кольца открытого ключа для Алисы.

1. Как показано в таблице 16.6, заполнение начинается со строки, где Алиса указывает уровень доверия поставщика, используя N (не доверяю), P (доверяю частично) и F (доверяю полностью). Для простоты принято, что каждый участник (включая Алису) имеет только один пользовательский ID.

Таблица 16.6. Пример 2, стартовая таблица

ID пользователя	Ключ ID	Ключ открытый	Доверие к поставщику	Сертификат(ы) доверия	Сертификат ключа	Законность	Метка времени
Алиса...	AB...	AB...	F			F	.....

Обратите внимание: на основании этой таблицы мы предполагаем, что Алиса выпустила сертификат для себя (неявно). Алиса, конечно, доверяет сама себе полностью. Уровень доверия к поставщику также *полный* и законность ключей — тоже. Хотя Алиса никогда не использует эту первую строку, это необходимо для операции **PGP.**

2. Теперь Алиса добавляет в таблицу Боба. Алиса полностью доверяет Бобу, но для того чтобы получить его открытый ключ, она просит, чтобы Боб передал открытый ключ электронной почтой, так же как и свои «отпечатки пальцев». Затем Алиса вызывает Боба, чтобы проверить его «отпечатки пальцев». Таблица 16.7 показывает это новое событие.

   Обратите внимание, что значение доверия для Боба *полное*, потому что Алиса полностью доверяет Бобу. Значение поля сертификата пусто — это показывает, что этот ключ был получен косвенно, а не в соответствии с сертификатом.

Таблица 16.7. Пример 2 после добавления в таблицу Боба

ID пользователя	Ключ ID	Ключ открытый	Доверие к поставщику	Сертификат(ы) доверия	Сертификат ключа	Законность	Метка времени
Алиса...	AB...	AB...	F			F	.....
Боб...	12...	12...	F			F	.....

3. Теперь Алиса добавляет в таблицу Теда. Теду полностью доверяют. Однако для этого конкретного пользователя Алиса не должна вызвать Теда. Вместо этого Боб, который знает открытый ключ Теда, передает его Алисе, сертификат которой включает открытый ключ Теда, как показано в таблице 16.8.

Таблица 16.8. Пример 2 после добавления Теда в таблицу

ID пользователя	Ключ ID	Ключ открытый	Доверие к поставщику	Сертификат(ы) доверия	Сертификат ключа	Законность	Метка времени
Алиса...	AB...	AB...	F			F	.....
Боб...	12...	12...	F			F	.....
Тэд...	48...	48...	F	Боба	F	F	.....

Обратите внимание: значение поля сертификата показывает, что сертификат был получен от Боба, значение сертификата доверия скопировано PGP с поля доверия к поставщику Боба. Значение поля законности ключей — значение сертификата доверия, умноженного на 1 (вес).

4. Теперь Алиса добавляет к списку Анну. Алиса доверяет Анне частично, но Боб передает сертификат для Анны с отметкой, что полностью доверяет. Таблица 16.9 показывает новое событие.

Таблица 16.9. Пример 2 после добавления Анны в таблицу

ID пользователя	Ключ ID	Ключ открытый	Доверие к поставщику	Сертификат(ы) доверия	Сертификат ключа	Законность	Метка времени
Алиса...	AB...	AB...	F			F	.....
Боб...	12...	12...	F			F	.....
Тед...	48...	48...	F	Боба	F	F	.....
Анна...	71....	71	P	Боба	F	F	.....

Обратите внимание, что значение доверия к поставщику для Анны является частичным, но сертификат доверия и законность ключей являются полными.

5. Теперь Анна представляет Джона, которому не доверяет Алиса. Таблица 16.10 показывает новое событие.

Таблица 16.10. Пример 2 после добавления в таблицу Джона

ID пользователя	Ключ ID	Ключ открытый	Доверие к поставщику	Сертификат(ы) доверия	Сертификат ключа	Законность	Метка времени
Алиса...	AB...	AB...	F			F	.....
Боб...	12...	12...	F			F	.....
Тэд...	48...	48...	F	Боба	F	F	.....
Анна...	71...	71...	P	Боба	F	F	.....
Джон...	31...	31...	N	Анны	P	P	.....

Обратите внимание, что PGP скопировал значение доверия к поставщику Анны (P) и *добавил к таблице* сертификата доверия поле для Джона. Значение поля законности ключей для Джона — 1/2 (P) в этот момент; это означает, что Алиса не должна использовать ключ Джона, пока он не изменится на 1 (F).

6. Теперь Джанетт, неизвестная Алисе, передает сертификат для Ли. Алиса полностью игнорирует это сертификат, потому что она не знает Джанетт. Затем Тед передает сертификат для Джона. Джон, которому доверяет Тед, вероятно, попросил Теда передать этот сертификат. Алиса смотрит на таблицу и находит пользовательское ID Джона с соответствующим ключом ID и открытым ключом. Алиса не добавляет другую строку к таблице; она только изменяет таблицу, как показано в таблице 16.11.
   Поскольку Джон имеет два сертификата в таблице Алисы, и его значение законности ключей — 1, Алиса может использовать его ключ. Но Джон все еще ненадежен. Обратите внимание, что Алиса может продолжить добавлять входы к таблице.

Таблица 16.11. Пример 2 после того, как для Джона был получен сертификат

ID пользователя	Ключ ID	Ключ открытый	Доверие к поставщику	Сертификат(ы) доверия	Сертификат ключа	Законность	Метка времени
Алиса...	AB...	AB...	F			F	.....
Боб...	12...	12...	F			F	.....
Тэд...	48...	48...	F	Боба	F	F	.....
Анна...	71...	71...	P	Боба	F	F	.....
Джон...	31...	31...	N	Анны Боба	P F	F	.....

Модель доверия в PGP

Как предложил Циммерман, мы можем создать модель доверия для любого пользователя в кольце, с пользователем в качестве центра активности. Такая модель может выглядеть, как показано на рис. 16.9. Рисунок показывает модель доверия для Алисы в некоторый момент. Диаграмма может измениться с любыми изменениями таблицы кольца общедоступного ключа.

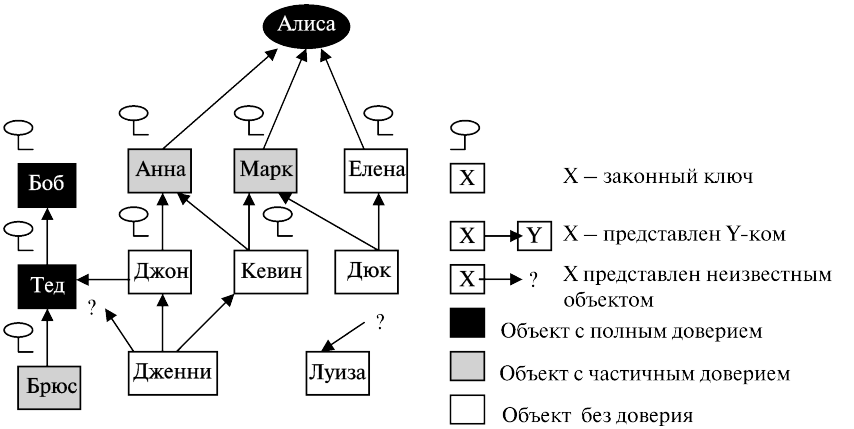


Рис. 16.9. Модель доверия

Рассмотрим рисунок. Здесь показаны три объекта с полным доверием в кольце Алисы (сама Алиса, Боб и Тед) и три объекта с частичным доверием (Анна, Марк и Брюс). Есть также шесть объектов без доверия. Девять объектов имеют законный ключ. Алиса может зашифровать сообщение к любому из этих объектов или проверить, что подпись получена от одного из этих объектов (ключ Алисы никогда не используется в этой модели). Есть также три объекта, которые не имеют никаких законных ключей с Алисой.

Боб, Анна и Марк сделали свои ключи законными, посылая их электронной почтой и подтверждая «отпечатками пальцев» по факсу. Елена передала сертификат от CA, потому что ей не доверяет Алиса и проверка по телефону невозможна. Хотя Теду полностью доверяют, он дал Алисе сертификат, подписанный Бобом. Джон передал Алисе два сертификата, один — подписанный Тедом и один — Анной. Кевин передал два сертификата Алисе, один — подписанный Анной и один — Марком. Каждое из этих удостоверений дает Кевину половину уровня законности; поэтому ключ Кевина законен. Дюк передал два удостоверения Алисе, одно — подписанное Марком и другое — Еленой. Так как Марку «полудоверяют», а Елене не доверяют, Дюк не имеет законного ключа. Дженни передала четыре сертификата, один — подписанный объектом, которому «полудоверяют», два — незаконными объектами и один — неизвестным объектом. Дженни имеет не много шансов, чтобы ее ключ признали законным. Луиза передала один сертификат, подписанный неизвестным объектом. Заметим, что Алиса может сохранить имя Луизы в таблице в случае, если в будущем прибудет сертификат для Луизы.

Сеть доверия

PGP может в конечном счете сделать **сеть доверия** между группой людей. Если каждый объект вводит все больше и больше объектов другим объектам, кольцо открытого ключа для каждого объекта становится большим и большим, и объекты в кольце могут тогда передавать безопасную электронную почту друг другу.

Аннулирование ключей

Аннулирование ключей может стать необходимым для объекта, если надо отменить его или ее открытый ключ в кольце. Это может случиться, если владелец ключа чувствует, что ключ скомпрометирован (например, украден) или слишком старый, чтобы быть безопасным. Чтобы отменять ключ, владелец может передать сертификат аннулирования, подписанный им самим. Сертификат аннулирования должен быть подписан старым ключом и распространен всем людям в кольце, которые использовали общедоступный ключ.

Распаковка информации из колец

Как мы видели, передатчик и приемник каждый имеет два кольца ключей: один частный и один открытый. Давайте посмотрим, какая информация нужна для того, чтобы послать и получить сообщение, извлеченное из этих колец.

Сторона передатчика

Предположим, что Алиса посылает электронную почту Бобу. Алиса нуждается в пяти единицах информации: ключ ID открытого ключа, который она использует, ее секретный ключ, ключ сеанса, ID открытого ключа Боба и открытый ключ Боба. Для того чтобы получить эти пять единиц информации, Алиса должна передать PGP четыре единицы информации: ее пользовательский ID (для электронной почты), ее фразу-пароль (последовательность нажатия клавиш с возможными паузами) и пользовательский ID Боба (см. рисунок 16.10).

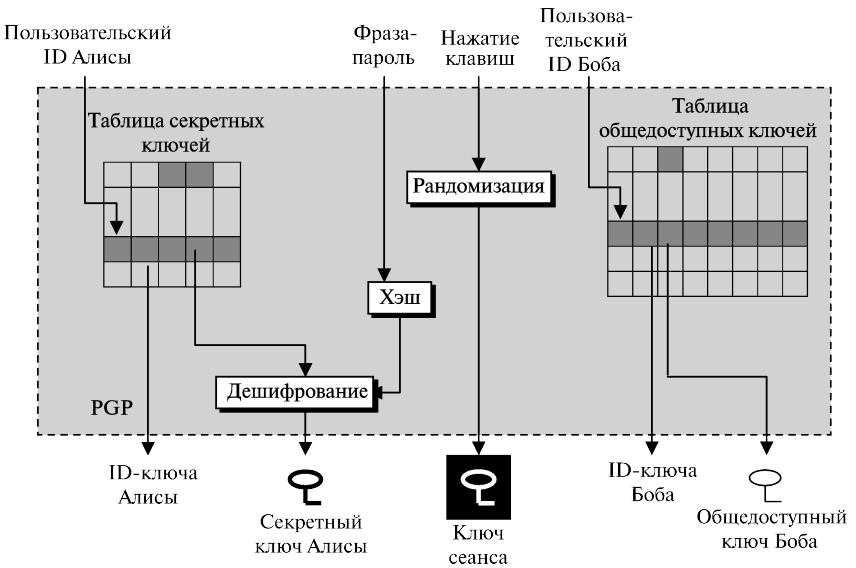


Рис. 16.10. Распаковка информации из колец на стороне передатчика

ID открытого ключа Алисы (полученный с сообщением) и ее секретный ключ (для подписи сообщения) хранятся в таблице кольца секретного ключа. Алиса выбирает пользовательский ID (user ID — ее адрес электронной почты), который она хочет использовать как индекс к этому кольцу. PGP извлекает ключ ID и зашифрованный секретный ключ. PGP использует заранее заданный алгоритм дешифрования и ее хэширования, фразу-пароль (как ключ), чтобы расшифровать этот секретный ключ.

Алиса также нуждается в секретном ключе сеанса. Ключ сеанса в PGP — случайное число с размером, который определен в алгоритме шифрования/дешифрования. PGP использует генератор случайных чисел, чтобы создать случайный ключ сеанса; начальное число — множество произвольных нажатий клавиши, напечатанных Алисой на ее клавиатуре. Каждое нажатие клавиши преобразовано в 8 бит, а каждая пауза между нажатиями клавиши преобразована в 32 бита. Комбинация проходит сложный генератор случайных чисел, чтобы создать очень достоверное случайное число как ключ сеанса. Обратите внимание, что ключ сеанса в PGP — одноразовый случайный ключ (см. приложение K), используемый только один раз.

Алиса также нуждается в ID ключа Боба (чтобы послать его с сообщением) и в открытом ключе Боба (чтобы зашифровать ключ сеанса). Эти две части информации извлекаются из таблицы кольца открытых ключей, используя пользовательское ID Боба (его адрес электронной почты).

Сторона приемника

На стороне приемника Бобу нужны три части информации: секретный ключ Боба (расшифровывает ключ сеанса), ключ сеанса (чтобы расшифровать данные) и ключ Алисы (чтобы проверить подпись) (см. рис. 16.11).

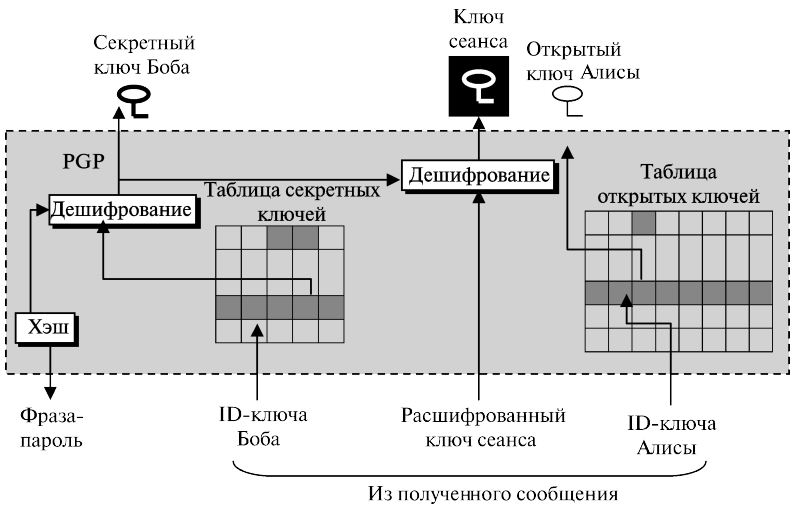


Рис. 16.11. Распаковка информации из колец на стороне приема

Боб использует ID своего открытого ключа, передаваемый Алисой, чтобы определить, что его соответствующий секретный ключ может расшифровать ключ сеанса. Эта часть информации может быть извлечена из таблицы кольца секретного ключа Боба. Секретный ключ, однако, зашифрован при записи в память. Боб должен использовать фразу-пароль и хэш-функцию, чтобы расшифровать ключ.

Зашифрованный ключ сеанса передали с сообщением; Боб применяет свой расшифрованный секретный ключ, чтобы расшифровать ключ сеанса.

Боб использует ID ключа Алисы, переданный с сообщением, чтобы извлечь открытый ключ Алисы, который хранится в таблице кольца открытого ключа Боба.

PGP-пакеты

Сообщение в PGP состоит из одного или более пакетов. В течение эволюции PGP формат и число типов пакета изменились. Подобно другим протоколам, известным до сих пор, PGP имеет типовой заголовок, который применяется к каждому пакету. Типовой заголовок нынешней версии имеет только два поля, как показано на рис. 16.12.

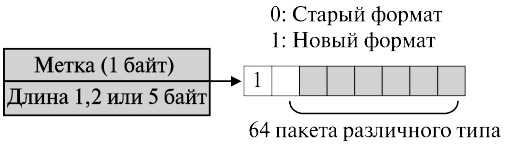


Рис. 16.12. Формат заголовка пакета

- **Метка (тег).** Нынешний формат для этого поля определяет метку как флажок на 8 битов; первый бит (самый старший) — всегда 1. Второй бит — 1, если мы используем последнюю версию. Остающиеся шесть бит могут определить до 64 различных типов пакетов, как показано в таблице 16.12.
- **Длина.** Поле длины определяет длину полного пакета в байтах. Размер этого поля является переменным; он может быть 1, 2 или 5 байтов.

Таблица 16.12. Некоторые обычно используемые типы пакетов

Значение	Тип пакета
1	Ключ сеанса, зашифрованный открытым ключом
2	Пакет подписи
5	Пакет секретного ключа
6	Пакет открытого ключа
8	Пакет сжатых данных
9	Пакет данных, зашифрованный секретным ключом
11	Пакет литеральных (буквенных данных)
13	Пакет пользовательского ID

Приемник может определить число байтов поля длины, на основании значения байта, идущего сразу после поля метки.

а. Если значение байта после поля метки меньше, чем 192, то поле длины — только один байт. Длина текстового блока (пакет минус заголовок) вычисляется как

длина текстового блока = первый байт

б. Если значение байта после поля метки между 192 и 223 (включая крайние значения), то поле длины — два байта. Длина текстового блока может быть вычислена как

длина текстового блока = (первый байт – 192) << 8 + второй байт + 192

в. Если значение байта после поля метки между 224 и 254 (включая крайние значения), то поле длины — один байт. Этот тип поля длины определяет только длину части текстового блока (частичная длина текстового блока). Частичная длина текстового блока может быть вычислена как

частичная длина текстового блока = 1<< (первый байт & 0 x 1F)

Обратите внимание, что формула означает $1 \times 2^{(\text{первый байт \& 0x1F})}$. Степень — это фактически значение пяти самых правых битов. Поскольку поле — между 224 и 254 включительно значение пяти самых правых битов — между 0 и 30 включительно. Другими словами, частичная длина текстового блока может быть между единицей (2^0) и 1 073 741 824 (2^{30}). Когда пакет представлен несколькими частичными текстовыми блоками, применима частичная длина текстового блока. Каждая частичная длина текстового блока определяет одну часть длины. Последнее поле длины не может быть частичной длиной текстового блока созданного сообщения. Например, если пакет имеет четыре части, он может иметь три частичных поля длины и одно поле длины другого типа.

г. Если значение байта после поля метки — 255, то поле длины состоит из пяти байтов. Длина текстового блока вычисляется как

Длина текстового блока = второй байт <<24 | третий байт << 16| четвертый байт << 8 |пятый байт

Пакет литеральных[1] **данных.** Пакет буквенных данных переносит или содержит текущие данные, которые передаются или сохраняются. Этот пакет — самый элементарный тип сообщения; то есть, он не может нести никакой другой пакет. Формат пакета показан на рис. 16.13.

[1] Литерал – тип данных для переноса или сохранения определенного заранее типа данных (вещественные, булевы, переменные и т. д.).

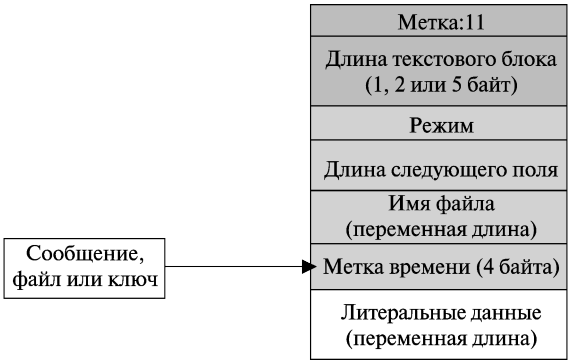


Рис. 16.13. Пакет буквенных данных

- **Режим.** Это однобайтовое поле определяет, как данные написаны в пакете. Значение этого поля может быть «b» для двоичных данных, «t» — для текста, или любое другое значение, определенное для собственных целей.
- **Длина следующего поля.** Это однобайтовое поле определяет длину следующего поля (поля имени файла).
- **Имя файла.** Это поле переменной длины определяет название файла или сообщения в виде строки ASCII.
- **Метка времени.** Это четырехбайтовое поле определяет время создания или последней модификации сообщения. Значение может быть 0 — это означает, что пользователь выбирает опцию «не определять время».
- **Литеральные данные.** Это поле переменной длины, переносящее фактические данные (файл или сообщение) в тексте или двоичном виде (в зависимости от значения поля режима).
- **Сжатый пакет данных.** Этот пакет, переносящий пакеты сжатых данных. Рисунок 16.14 показывает формат пакет сжатых данных.

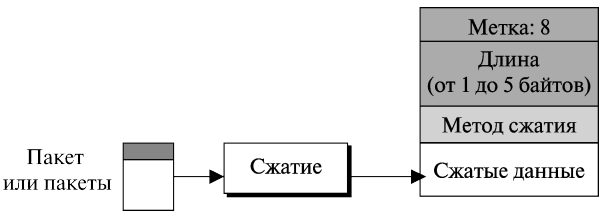


Рис. 16.14. Пакет сжатых данных

- **Метка (метод сжатия).** Это однобайтовое поле определяет метод сжатия, используемый для сжатия данных (следующее поле). Значения, определенные для этого поля, пока 1 (ZIP) и 2 (ZLIP). Также реализация может применять другие экспериментальные методы сжатия. Метод ZIP обсуждается в приложении M.

- **Сжатые данные.** Это поле переменной длины переносит данные после сжатия. Обратите внимание, что в этом поле может быть один пакет данных или последовательное соединение двух или более пакетов. Общая ситуация — единственный пакет литеральных данных или комбинация пакета подписи, сопровождаемого пакетом литеральных данных.
- **Пакет данных, зашифрованных ключом засекречивания.** Этот пакет переносит данные от одного пакета или комбинации пакетов, которые были зашифрованы, с использованием обычного алгоритма с симметричными ключами. Обратите внимание, что пакет, несущий одноразовый ключ сеанса, передается перед этим пакетом. Рисунок 16.15 показывает формат пакета зашифрованных данных.

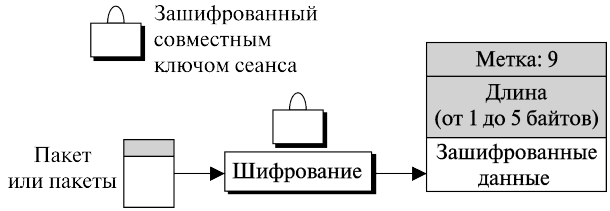


Рис. 16.15. Пакет зашифрованных данных

- **Пакет подписи.** Пакет подписи мы уже обсуждали раньше, когда рассматривали защиту целостности данных. Рисунок 16.16 показывает формату пакета подписи.

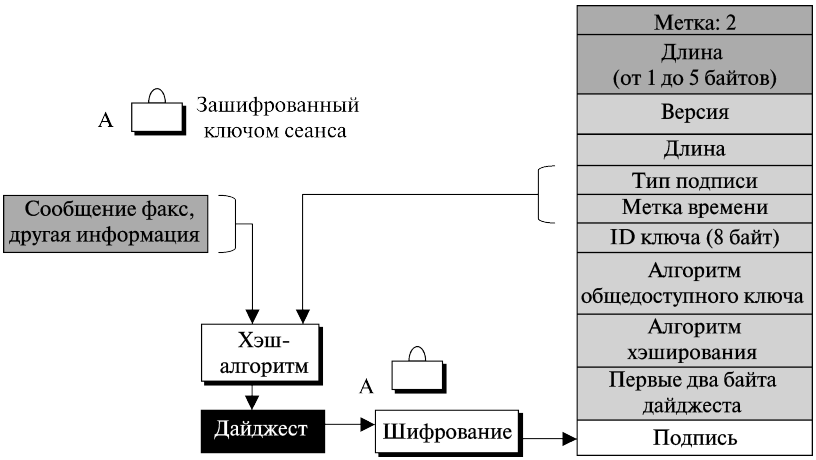


Рис. 16.16. Пакет подписи

- **Версия.** Это однобайтовое поле определяет используемую версию PGP.

- **Длина.** Это поле сначала было выделено для того, чтобы показать длину следующих двух полей, но теперь размер этих полей установлен, поэтому значение этого поля равно 5.
- **Тип подписи.** Это однобайтовое поле определяет цель подписи. Оно документирует подпись. Таблица 16.13 показывает некоторые типы подписи.

Таблица 16.13. Некоторые значения подписи

Значение	Подпись
0x00	Подпись двоичного документа (сообщение или файл)
0x01	Подпись текстового документа (сообщение или файл)
0x10	Общий сертификат пользовательского ID и пакета открытого ключа. Подписывающее лицо не указывает никаких данных о владельце ключа
0x11	Персональный сертификат пользовательского ID и пакет открытого ключа. Не проводится верификация владельца ключа
0x12	Случайный сертификат пользовательского ID и пакет открытого ключа. Некоторая случайная верификация владельца ключа
0x13	Положительный сертификат пользовательского ID и пакет открытого ключа. Делается существенная верификация
0x30	Подпись аннулирования сертификата. Она удаляет более ранний сертификат (от Ox10 до Ox13)

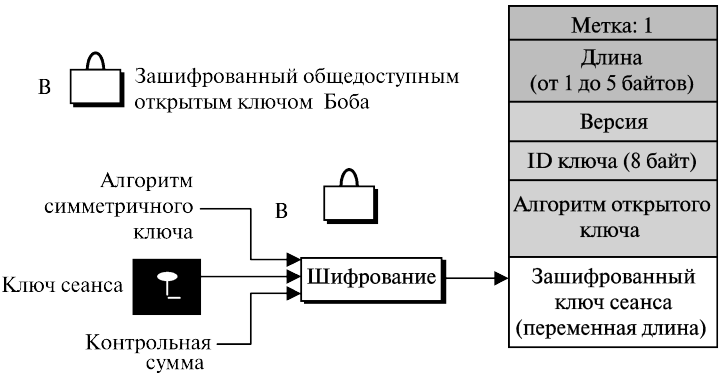
- **Метка времени.** Это четырехбайтовое поле, которое определяет время, когда подпись была вычислена.
- **Ключ ID.** Это восьмибайтовое поле определяет ID открытого ключа подписывающего лица. Оно указывает верификатору, какой открытый ключ подписывающего лица должен быть использован, чтобы расшифровать дайджест.
- **Алгоритм открытого ключа.** Это однобайтовое поле дает код для алгоритма открытого ключа, который применялся для шифрования дайджеста. Верификатор использует тот же самый алгоритм, чтобы расшифровать дайджест.
- **Алгоритм хэширования.** Это однобайтовое поле дает код для алгоритма хэширования, обычно создает дайджест.
- **Первые два байта дайджеста сообщения.** Эти два байта используются как своего рода контрольная сумма. Они гарантируют, что приемник использует правильный ключ ID, чтобы расшифровать дайджест.
- **Подпись.** Это поле переменной длины. Оно содержит зашифрованный дайджест, подписанный передатчиком.

Пакет ключа сеанса, зашифрованный открытым ключом.

Этот пакет применяется, чтобы передать ключ сеанса, зашифрованный открытым ключом приемника. Формат пакета показан на рис. 16.17.

- **Версия.** Это однобайтовое поле определяет используемую версию PGP.

• **Ключ ID.** Это восьмибайтовое поле определяет ID общедоступного ключа передатчика. Он указывает приемнику, какой общедоступный ключ передатчика должен использоваться, чтобы расшифровать ключ сеанса.



• **Алгоритм открытого ключа.** Это однобайтовое поле дает код для алгоритма открытого ключа, использованного для шифрования ключа сеанса. Приемник применяет тот же самый алгоритм, чтобы расшифровать ключ сеанса.

Рис. 16.17. Пакет ключа сеанса

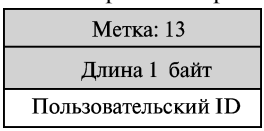
• **Сеанс шифрования.** Это область переменной длины, которая содержит зашифрованное значение ключа сеанса, созданного отправителем и посланного приемнику. Шифрование основано на следующих средствах:
а. симметричный алгоритм шифрования с одним октетом;
б. ключ сеанса;

в. контрольная сумма с двумя октетами равняется сумме октетов ключей предыдущих сеансов.

Пакет открытого ключа. Этот пакет содержит открытый ключ отправителя. Формат пакета показан на рис. 16.18.

Рис. 16.18. Пакет открытого ключа

- **Версия.** Эта однобайтовая область определяет используемую версию PGP.
- **Метка времени.** Эта четырехбайтовая область определяет время, когда был создан ключ.
- **Законность.** Эта двухбайтовая область показывает число дней, в продолжении которых ключ является действительным. Если значение равно 0, это означает, что действие ключ не заканчивается.
- **Алгоритм открытого ключа.** Эта однобайтовая область дает код для алгоритма открытого ключа.
- **Открытый ключ.** Эта область переменной длины содержит открытый ключ. Его содержание зависит от алгоритма открытого ключа.



Пакет пользовательского ID. Этот пакет идентифицирует пользователя и обычно связывает пользователя и содержание с открытым ключом передатчика. Рисунок 16.19 показывает формат пакет пользовательского ID. Обратите внимание, что поле длины общего заголовка — только один байт.

Рис. 16.19. Пакет пользовательского ID

- **Пользовательский ID.** Эта строка переменной длины определяет пользовательский ID передатчика. Это обычно имя пользователя, сопровождаемое адресом электронной почты.

PGP-сообщения

Сообщение в PGP — комбинация упорядоченных и/или вложенных пакетов. Даже притом, что не все комбинации пакетов могут составлять сообщение, список комбинаций пакетов достаточно длинный. В этой секции мы приведем несколько примеров, чтобы проиллюстрировать идею.

Зашифрованное сообщение

Зашифрованное сообщение может быть последовательностью двух пакетов: пакета ключа сеанса и симметрично зашифрованного пакета. Последний обычно представляет собой вложенный пакет. Рисунок 16.20 показывает такую комбинацию.

Обратите внимание, что пакет ключа сеанса содержит только единственный пакет. Зашифрованный пакет данных состоит из сжатого пакета. Сжатый пакет состоит из пакета литеральных данных. Последний содержит литеральные данные.

Подписанное сообщение

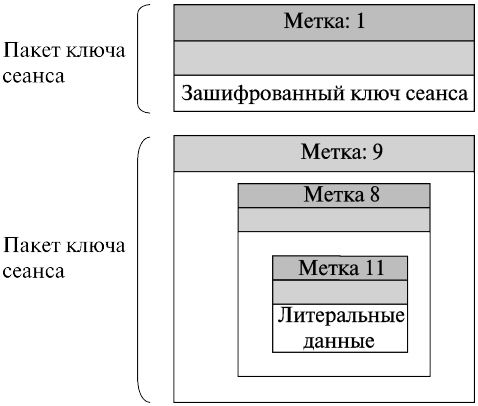


Рис. 16.20. Зашифрованное сообщение

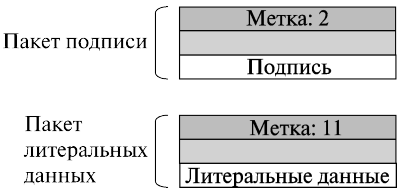
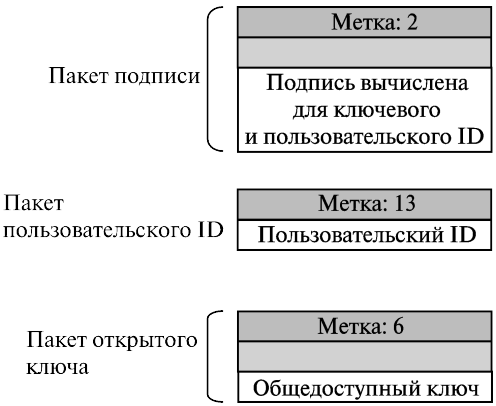


Рис. 16.21. Подписанное сообщение

Подписанное сообщение может быть комбинацией пакета подписи и литерального пакета, как это показано на рис. 16.21.



Сообщение сертификата

Хотя сообщение сертификата может принимать множество форм, один простой пример — комбинация пользовательского ID пакета и пакета открытого ключа, как показано на рис. 16.22. Подпись здесь вычислена для последовательного соединения ключевого и пользовательского ID.

Рис. 16.22. Сообщение сертификата

Приложения PGP

PGP широко применялся для персональной электронной почты. Это использование, вероятно, продолжится.

16.3. S/MIME

Другая служба безопасности разработана для электронной почты **Безопасное/Многоцелевое расширение почты (S/MIME — Secure/Multipurpose Internet Mail Extinction)**. Этот протокол является расширением **Многоцелевого расширения почты (MIME — Multipurpose Internet Mail Extinction)**. Для лучшего понимания S/MIME кратко изложим MIME. Затем обсудим S/MIME как дополнение к MIME.

MIME

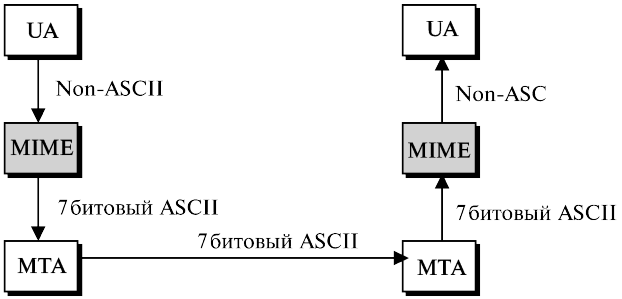
Электронная почта имеет простую структуру. Однако за эту простоту приходится платить. Почта может передать сообщения только в формате NVT ASCII на 7 битов. Другими словами, она имеет некоторые ограничения. Например, она не может работать с языками, которые не поддержаны символами ASCII (такими как арабский язык, китайский, французский, немецкий, иврит, японский, русский). Также она не может использоваться для передачи двоичных файлов или видео- и аудиоданных.

MIME — это дополнительные протоколы, которые позволяют данным, не передаваемым с помощью ASCII, проходить по электронной почте. MIME преобразовывает такие данные на стороне передатчика к данным NVT ASCII и поставляет их клиенту MTA по сети Интернет. Сообщение на приемной стороне преобразуется снова к первоначальному виду.

Мы можем представлять себе MIME как множество программных функций, которые преобразовывают данные, не передаваемые в ASCII, к данным ASCII, и наоборот, как показано на рис. 16.23.

MIME определяет пять заголовков (содержания), которые можно добавить к первоначальной электронной почте, чтобы определить параметры преобразования:

1. Version — MIME (версия);
2. Content — Type (Содержание – тип);
3. Content — Transfer – Encoding (Содержание – Передача – Шифрование);
4. Content — ID (Содержание – ID);
5. Content — Description (Содержание – Описание).



UA	User Agent	Почтовый агент
MIME	Multipurpose Internet Mail Extension	Многоцелевое расширение электронной Интернет почты
MTA	Mail Transfer Agent	Агент передачи электронной почты
ASCII	American Standard for Information Interchange	Американский стандартный код для обмена информацией
Non-ASCII		Не ASCII

Рис.16.23. MIME

Заголовок e-mail
Заголовок e-mail MIME – Version 1.1 Content Type: type/subtype Content –Transfer _ Encoding: encoding type Content –Is: message Id Content – Description: textual explanation of nontextual contents E-mal текстовый блок
E-mail текстовый блок

Рис. 16.24. MIME заголовок

Рисунок 16.24 показывает заголовки MIME. Далее дается более подробное описание каждого из заголовков.

MIME-версия

Этот заголовок определяет версию используемого MIME. Текущая версия имеет номер 1.1.

Version MIME: 1.1

Content-Type

Этот заголовок определяет тип данных, используемых в текстовом блоке сообщения (в «теле» сообщения), и подтип содержания, отделенный наклонной чертой. В зависимости от подтипа заголовок может содержать другие параметры.

Content – Type: <type/ subtype; parameters>

MIME позволяет семь различных типов данных. Они перечислены в таблице 16.14 и ниже описаны более подробно.

- **Текст.** Первоначальное сообщение находится в формате ASCII на 7 битов, и преобразование MIME не требуется. Есть два подтипа, в настоящее время используемые: *исходное* и *HTML*.
- **Многоэлементный.** Текстовый блок содержит множественные независимые части. Многоэлементный заголовок должен определить границу между каждой частью. Для этой цели используется параметр. Параметр — строковый символ, который ставится перед каждой частью, для отдельной линии и с предшествующими двумя дефисами. Текстовый блок использует граничный символ, которому также предшествуют два дефиса.

Для этого типа определены четыре подтипа: *смешанный, параллельный, дайджест и альтернатива*.

В смешанном подтипе части должны быть представлены получателю точно в том же порядке, как и в сообщении.

Таблица 16.14. *Типы и подтипы данных в MIME*

Тип	Подтип	Описание
	Обычный (Plain)	Неформатированный
	HTML	Формат HTML
Многоэлементный (Multipart)	Смешанный (Mixed)	Блок содержит упорядоченные части данных различного типа.
	Параллельный (Parallel)	То же самое, что выше, но неупорядоченное.
	Дайджест (Digest)	Тот же самый, что смешанный, но по умолчанию тип – Message / RFC822
	Альтернативный (Alternative)	Части различных версий одного и того же сообщения.
Сообщение (Message)	RFC822	В блоке инкапсулировано сообщение
	Частичное (Partial)	Блок – это фрагмент другого – большего сообщения.
	Внешний текст (External-Body)	Блок – это ссылка на другое сообщение.
Изображение (Image)	JPEG	Изображение в формате JPEG
	GIF	Изображение в формате GIF
Видео (Video)	MPEG	Видео в формате MPEG
Аудио (Audio)	Основное (Basic)	Одиночный канал голосового сообщения в полосе 8 КГц
Приложение (Application)	Язык описания (PostScript)	Язык описания – Adobe PostScript
	Поток октетов (Octet-stream)	Обычные двоичные данные (восьмибитовые байты).

Каждая часть имеет различный тип и определяет границы. Параллельный подтип подобен смешанному подтипу, за исключением того, что порядок следования частей не играет роли. Подтип дайджеста также подобен смешанному подтипу за исключением того, что по умолчанию тип/подтип (type/subtype) имеет значение сообщение (message)/RFC822, как это определено ниже. В альтернативном подтипе то же самое сообщение повторено, используя различные форматы. Далее — пример многоэлементного сообщения, использует смешанный подтип:

Content-Type: multipart/mixed; boundary=xxxx
—xxxx
Content -Type: text/plain;
—xxxx
Content –Type: image/gif;
.....
—xxxx—

- **Сообщение.** В типе сообщения содержание — полное самостоятельное сообщение почты, либо часть сообщения почты, либо указатель на сообщение. В настоящее время используются три подтипа (*RFC822, частичные (partial)* и *внешнее тело (external-body))*. Подтип RFC822 применяется, если в содержание включено сообщение, формирующее другое сообщение (включая заголовок и тело). Частичный подтип используется, если первоначальное сообщение было фрагментировано в несколько различных почтовых сообщений и данное сообщение почты — это один из фрагментов. Фрагменты должны быть повторно собраны в пункте назначения MIME. К сообщению добавляются три параметра: *ID, номер* и *общее количество.* ID идентифицирует сообщение и присутствует во всех фрагментах. Номер определяет порядок и число фрагментов, которые включает в себя первоначальное сообщение. Ниже приводится пример сообщения с тремя фрагментами:

Content-Type: message/partial;
id=”forouzan@challenger.atc.flida.edu”;
number=1;
total=3;
.....
.....

Подтип «внешний текст» указывает, что информация не содержит настоящего сообщения, а только ссылку (указатель) на первоначальное сообщение. Параметры этого подтипа определяют, как получить доступ к первоначальному сообщению. Ниже приведен пример:

Content- Type: message/ external-body; name=”report.text”;
site=”fhda.edu”;
access-type=”ftp”;
.....

.....

- **Изображение.** Первоначальное сообщение — неподвижное изображение — указывает на то, что нет никакой мультипликации. В настоящее время используется два подтипа: *Объединенная Экспертная группа по фотографии (JPEG — Joint Photographic Expert Group)*, который позволяет сжать изображение, и *Формат Обмена Графическими Файлами (GIF — Graphics Interchange Format)*.
- **Видео.** Первоначальное сообщение — изменяющееся по времени изображение (мультипликация). Единственный подтип — Группа экспертов по движущимся изображениям *(MPEG — Moving Picture Experts Group)*. Если движущееся изображение сопровождается звуковым рядом, его нужно послать отдельно, используя звуковой (audio) заголовок «содержание — тип».
- **Аудио.** Первоначальное сообщение является звуковым. Основным является единственный подтип, который использует стандартный аудиоканал на 8 кГц.
- **Приложение.** Первоначальное сообщение — тип данных, не определенных предварительно. В настоящее время применяется только два подтипа: *PostScript* и *поток октетов*. PostScript (язык описания страниц) используется, когда данные находятся в формате Adobe *PostScript*. Поток октетов нужен, когда данные должны интерпретироваться как последовательность байтов по 8 битов (двоичный файл).

Content — Transfer — Encoding (Содержание — Передача — Кодирование)

Этот заголовок определяет метод, используемый для кодирования сообщения в виде нулей и единиц для транспортировки через сеть.

Content – Transfer – Encoding: <type>

Пять типов методов шифрования приведены в таблице 16.15.

Таблица 16.15. Content — Transfer — Encoding (Содержание — Передача — Шифрование)

Тип	Описание
7 бит	NVT ASCII символы и короткие строки
8 бит	Символы, не отображаемые в ASCII (Non-ASCII); символы и короткие линейки
Двоичный	Символы, не отображаемые в ASCII (Non-ASCII); символы и нелимитированные линейки
Radix-64	6-битовые блоки, зашифрованные по 8 бит, в символы ASCII, использующие преобразование Radix-64 conversion
Ограниченная печатная строка	Символы, не отображаемые в ASCII (Non-ASCII) и зашифрованные как эквивалентные знаки кода ASCII

- **7bit.** Это 7 бит, закодированные в NVT ASCII. Хотя никакого специального преобразования здесь не требуется, но необходимо, чтобы длина линейки не превышала 1000 символов.
- **8bit.** Это закодированные по 8 битов символы не-ASCII, которые можно передать по каналу, но длина линейки не должна превышать 1000 символов. MIME в этом случае ничего не кодирует; основной SMTP позволяет передачу 8-битовых символов не-ASCII . Поэтому этот тип не рекомендуется. Предпочтительней применять типы Radix 64 и «приспособленный для печати».
- **Двоичный.** Это закодированные по 8 битов символы не-ASCII, которые можно передать по каналу, но разрешается длина линейки более 1000 символов. MIME в этом случае ничего не кодирует; основной протокол SMTP позволяет передачу двоичных данных. Поэтому этот тип не рекомендуется. Предпочтительней применять типы Radix 64 и «ограниченную печатную строку».
- **Radix-64.** Этот тип позволяет передавать данные, состоящие из байтов, когда самый высокий разрядный бит — не обязательно равный нулю. Radix-64 преобразовывает этот тип данных к символам типа «ограниченная печатная строка», которые можно тогда послать как символы ASCII или любой тип символов, поддерживаемый основными алгоритмами передачи почты. Radix-64 разделяет двоичные (потоки бит) в блоки по 24 бита. Каждый блок затем разделяется на четыре секции, каждая состоит из 6 бит (см. рис. 16.25). Каждая секция на 6 битов интерпретируется как один символ согласно Таблице 16.16.
- **Ограниченная печатная строка (Quoted-printable).** Radix-64 — избыточная схема кодирования: то есть 24 бита преобразуются в четыре символа и в конечном счете посылаются как 32 бита. Мы имеем избыточность 25 процентов. Если данные состоят главным образом из символов ASCII с небольшой маленькой частью не-ASCII, мы можем использовать кодирование типа Quoted-printable («ограниченная печатная строка»). Если это символ ASCII, то его посылают без преобразования. Если символ — не ASCII, его посылают как три символа. Первый символ — знак равенства (=). Следующие два символа — шестнадцатеричное представление байта. На рисунке 16.26 показан пример.

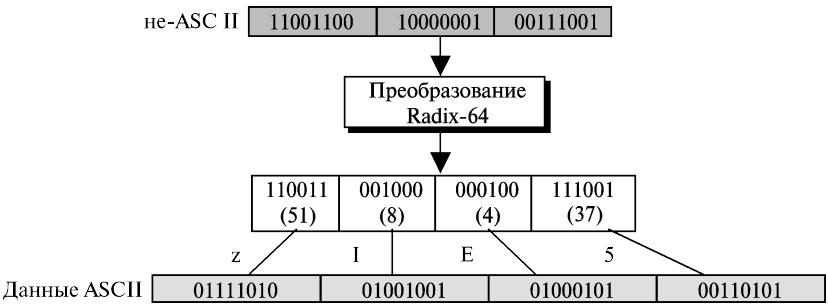


Таблица 16.16. Таблица кодирования Radix-64

Значение	Код	Значение	Код	Значение	Код	Значение	Код	Значение	Код	Значение	Код
0	**A**	11	**L**	22	**W**	33	**h**	44	**S**	55	**3**
1	**B**	12	**M**	23	**X**	34	**i**	45	**t**	56	**4**
2	**C**	13	**N**	24	**Y**	35	**J**	46	**u**	57	**5**
3	**D**	14	**0**	25	**Z**	36	**k**	47	**V**	58	**6**
4	**E**	15	**P**	26	**a**	37	**1**	48	**W**	59	**7**
5	**F**	16	**Q**	27	**b**	38	**m**	49	**x**	60	**8**
6	**G**	17	**R**	28	**c**	39	**n**	50	**y**	61	**9**
7	**H**	18	**S**	29	**d**	40	**0**	51	**Z**	62	**+**
8	**I**	19	**T**	30	**e**	41	**P**	52	**0**	63	**/**
9	**J**	20	**U**	31	**f**	42	**q**	53	**1**		
10	**K**	21	**V**	32	**g**	43	**r**	54	**2**		

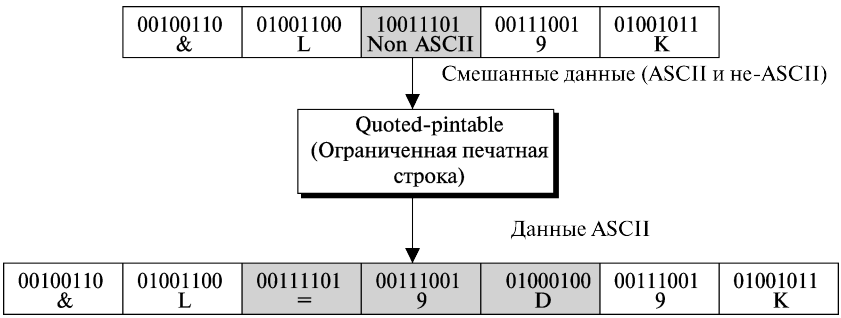


Рис. 16.26. Ограниченная печатная строка (Quoted printable)

Рис. 16.25. Преобразование Radix-64

Content-Id (Содержание-Id)

Этот заголовок уникально идентифицирует целое сообщение среди многих сообщений

Content-Id : id =<content-id> (содержание id)

Content-Description (Содержание-Описание)

Этот заголовок определяет, является ли блок информации неподвижным изображением, аудио или видео.

Content – Description : <описание>

S/MIME

S/MIME добавляет некоторые новые типы заголовков-содержания, чтобы включить службы безопасности в MIME. Все эти новые типы включают параметр «application/pkcsP-mime», в котором «pkcs (Public Key Cryptography Specification)» определяет «Спецификацию криптографии открытого ключа».

Синтаксис криптографического сообщения

Чтобы определять услуги безопасности, такие как конфиденциальность или целостность, можно добавить к типам содержания MIME, S/MIME определитель **криптографический синтаксис сообщения** (CMS —Cryptographic Message Syntax). Синтаксис в каждом случае определяет точную схему кодирования каждого типа содержания. Ниже рассматриваются типы сообщения и различные подтипы, которые могут быть созданы из этих сообщений. Для детального изучения можно прочитать [RFC3369] и [3370].

Тип содержания данных — произвольная строка. Созданный объект называется *Данными*.

Signed — Data Content Type (тип содержания — подписанные данные). Этот тип обеспечивает только целостность данных. Он содержит любой тип и нулевое или большее количество подписей. Кодируемый результат — *объект*, называемый *signedData*. Рисунок 16.27 показывает процесс создания объекта этого типа. В процессе используются следующие шаги.

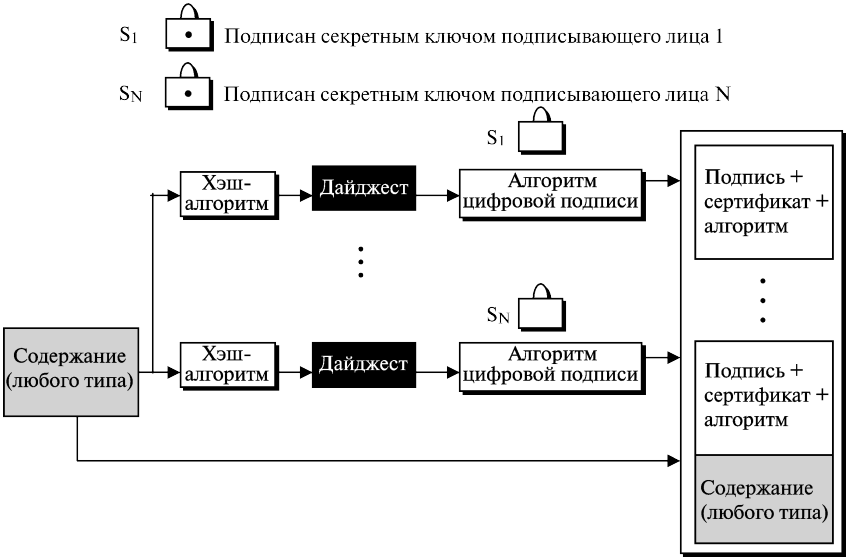
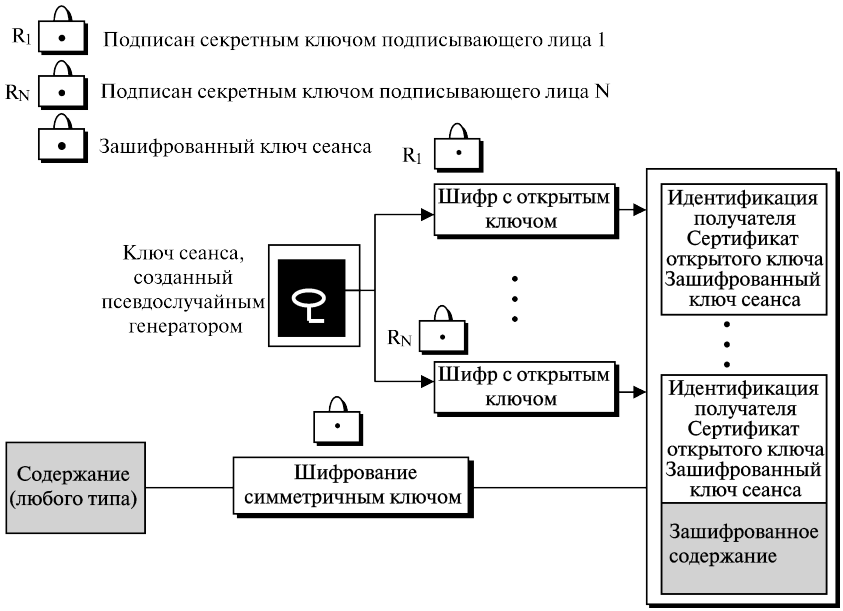


Рис. 16.27. Содержание типа «подписанные данные»

1. Для каждого подписывающего лица дайджест сообщения создает заголовок-содержание, используя заданный алгоритм хэширования, который выбирается некоторым подписывающим лицом.
2. Каждый дайджест сообщения подписывается секретным ключом подписывающего лица.
3. Содержание, значения подписи и сертификата, а также алгоритмы затем собираются для создания объекта *signedData*.

Enveloped — Data Content Type (тип содержания — конверт данных). Этот тип используется, чтобы обеспечить секретность сообщения. Он содержит любой тип от нуля и далее зашифрованных ключей и сертификатов. Кодируемый результат — *объект*, называемый *envelopedData*. Рисунок 16.28 показывает процесс создания объекта этого типа.



1. Создается псевдослучайный ключ сеанса для алгоритмов с симметричными ключами.
2. Для каждого получателя копия ключа сеанса зашифрована с открытым ключом каждого получателя.
3. Содержание зашифровано, используя определенный алгоритм и созданный ключ сеанса.
4. Зашифрованное содержание, зашифрованный ключ сеанса. Использованный алгоритм и сертификаты кодируются с применением RADIX-64.

Рис. 16.28. Содержание типа «конверт данных»

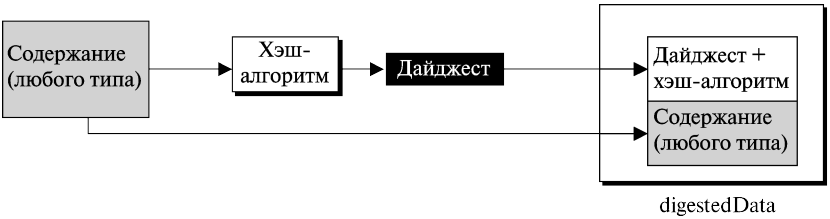


Рис. 16.29. Содержание типа дайджест данных

Digest-Data Content Type (*Содержание типа «дайджест данных»*). Этот тип применяется, чтобы обеспечить целостность сообщения. Результат обычно используется как содержание типа «конверт данных». Кодируемый результат — *объект*, называемый *digestedDate*. Рисунок 16.29 показывает процесс создания объекта этого типа.

1. Дайджест сообщения вычислен на основании содержания.
2. Дайджест сообщения, алгоритм и содержание добавляются вместе, чтобы создать *digestData*.

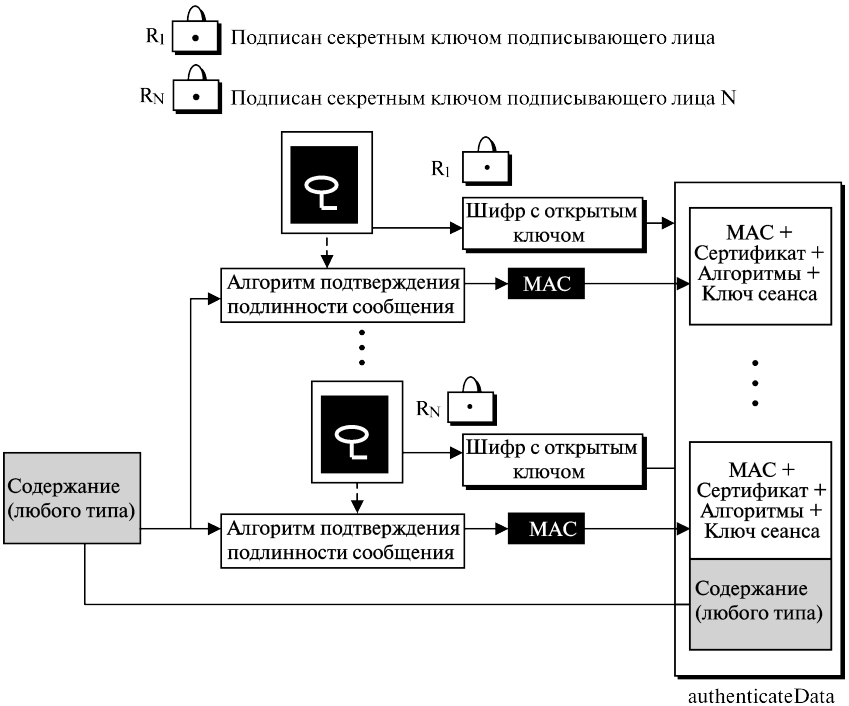


Рис. 16.30. Содержание типа «подтверждение подлинности»

Encrypted-Data Content Type (Тип содержания «зашифрованные данные»). Этот тип применяется, чтобы создать зашифрованную версию содержания любого типа. Хотя он похож на тип содержания «конверт данных», но не имеет информации о получателе. Он может использоваться, чтобы хранить зашифрованные данные, вместо того, чтобы передавать их. Процесс очень прост: пользователь применяет любой ключ (нормально, исходя из пароля) и любой алгоритм, чтобы зашифровать содержание. Зашифрованное содержание сохраняется без записи ключа или алгоритма. Созданный объект называется *encryptedDuta*.

Authenticated-Data Content Type (тип содержания — подтверждение подлинности данных). Этот тип используется, чтобы обеспечить установление подлинности данных. Объект называется *authenticatedData*. Рисунок 16.30 показывает процесс.

1. С применением псевдослучайного генератора для каждого получателя генерируется ключ кода, подтверждающий подлинность сообщения (MAC — Message Authentication Code).
2. Ключ кода, подтверждающего подлинность сообщения, зашифрован открытым ключом получателя.
3. Код, подтверждающий подлинность сообщения, создан для содержания.
4. Содержание, код, подтверждающий подлинность сообщения, алгоритм и другие данные собраны вместе в формате объекта с именем authenticatedData.

Таблица 16.17. Криптографические алгоритмы для S/MIME

Алгоритм	**Передатчик должен поддерживать**	**Приемник должен поддерживать**	**Передатчик может поддерживать**	**Приемник может поддерживать**
Алгоритм зашифрованного содержания	Triple DES	Triple DES		l. AES 2. RC2/40
Алгоритм шифрования ключа сеанса	RSA	RSA	Diffie-Hellman (Диффи-Хеллман)	Diffie-Hellman (Диффи-Хеллман)
Хэш-алгоритм	SHA-1	SHA-1		MD5
Алгоритм шифрования дайджеста	DSS	DSS	RSA	RSA
Алгоритм определения подлинности		HMAC с SHA-1		

Управление ключами

Управление ключами в S/MIME — это комбинация управления ключами, используемого в X.509 и PGP. S/MIME использует сертификат открытого ключа, который подписан удостоверяющей администрацией, определенной X.509. Однако пользователь несет ответственность по поддержке сети доверия для проверки подписи, как это определено PGP.

Криптографические алгоритмы

S/MIME определяет несколько криптографических алгоритмов, как это показано в таблице 16.17. Термин «должен» означает абсолютное требование; термин «может» означает рекомендацию.

Ниже показан пример конверта данных, в котором маленькое сообщение зашифровано с использованием трехкратного DES.

Content-Type: application/pkcs7-imme; mime-type=enveloped-data
Content-Transfer-Encoding: Radix-64
Content-Description: attachment
Name= "report.txt";
cb32ut67f4bhijHU21oi87eryb0287hmnklsgFDoY8bc659GhIGfH6543mhjkdsaH
23YjBnmNybmlkzjhgfdyhGe23Kjk34XiuD678Esl6se09jy76jHuytTMDcbnmlkjg
fFdiuyu67.S543mOn3tiG34un12P2454Hoi87e2rybOH2MjN6KuyrlsgFDoY897f
k923jljk1301 XiuD6gh78EsUyT23y

Приложения S/MIME

Предполагается, что S/MIME будет выбран промышленностью для обеспечения безопасности коммерческой электронной почты.

16.4. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Электронная почта рассматривается в [For06] и [For07]. PGP рассматривается в [Sta06], [KPS02J] и [Rhe03]. S/MIME рассматривается в [Sta06] и [lRhe03].

Сайты

Нижеследующие сайты дают больше информации о темах, обсужденных в этой лекции.

http://axion.physics.ubc.ca/pgp-begin.html
csrc.nist.gov/publications/nistpubs/800-49/sp800-49.pdf
www.faqs.org/rfcs/rfc2632.html

16.5. Итоги

- Поскольку при использовании электронной почтовой связи отсутствует сеанс, передатчик сообщения должен включить в сообщение названия или

идентификаторы алгоритмов, используемых в электронной почте. В электронной почте шифрование/дешифрование делается при помощи алгоритма с симметричными ключами, но при этом ключ засекречивания для расшифровки сообщения зашифрован открытым ключом приемника и передается с сообщением.

- Первый протокол, рассмотренный в этой лекции, называется «Очень хорошей конфиденциальностью» (PGP). Он был изобретен Филом Циммерманом для обеспечения электронной почты услугами секретности, целостности и установления подлинности. PGP может использоваться для того, чтобы создать безопасное почтовое сообщение или надежно хранить файл для будущего чтения.
- В PGP Алиса нуждается в кольце открытых ключей для каждого человека, с которым она переписывается. Она также нуждается в кольце принадлежащих ей частных/открытых ключей.
- В PGP не нужны Центры Сертификации; любой в кольце может подписать сертификат для кого-либо еще в этом кольце. В PGP нет иерархии доверия; нет дерева иерархии. Может быть много путей от администрации, которой полностью или частично доверяют, к любому объекту.
- Работа PGP базируется на доверии поручителя, уровне доверия и законности открытых ключей. PGP организует сеть доверия между группами людей.
- PGP определил несколько типов пакетов: литеральных данных, сжатый пакет данных, пакет данных, зашифрованный ключом засекречивания, пакет подписи, пакет ключа сеанса, зашифрованный открытым ключом, пакет открытого ключа и пользовательский ID пакета.
- В PGP мы имеем несколько типов сообщений: зашифрованное сообщение, подписанное сообщение и сообщение сертификата.
- Другая служба безопасности, разработанная для электронной почты:
- Безопасное/Многоцелевое расширение Интернет-почты (S/MIME). Протокол многоцелевого расширения Интернет-почты (MIME) — это протокол, который является дополнительным протоколом и позволяет не-ASCII данные посылать через электронную почту. По отношению к MIME S/MIME дополняется некоторыми новыми типами содержания для обеспечения службы безопасности.
- Криптографический синтаксис сообщения (CMS) определяет несколько типов сообщений — они создаются на основе новых типов содержания, которые добавляются к MIME. В этой лекции упоминались несколько типов сообщения, таких как: тип содержания данных, тип содержания подписанных данных, тип содержания конверта данных, тип содержания дайджеста данных, тип содержания зашифрованных данных и тип содержания данных, подтверждающих подлинность.
- Управление ключами в S/MIME — это комбинация управления ключами, используемая X.509 и PGP. S/MIME использует открытый ключ, подписанный удостоверяющей администрацией.

16.6. Набор для практики

Обзорные вопросы

1. Объясните, как Боб, когда он получает PGP-сообщение от Алисы, узнает, какие криптографические алгоритмы она использовала.
2. Объясните, как Боб узнает, какой криптографический алгоритм Алиса использовала, когда получает S/MIME-сообщение от нее.
3. Объясните, как Боб и Алиса в PGP обмениваются секретным ключом шифрования сообщений.
4. Объясните, как Боб и Алиса обмениваются секретным ключом шифрования сообщений.
5. Сравнить сходство и различия сертификатов в PGP и S/MIME. Объясните сеть доверия, которая применяется для сертификации в PGP и в S/MIME.
6. Назовите семь типов пакетов, используемых в PGP, и объясните их цели.
7. Назовите три типа сообщений в PGP и объясните их цели.
8. Назовите все типы содержания, определенные CMD, и их цели.
9. Сравните совпадения и отличия управление ключами в PGP и S/MIME.

Упражнения

1. Боб получает PGP-сообщение. Как он может узнать тип пакета, если значение метки:
   а. 8
   б. 9
   в 2
2. В PGP в почтовом сообщении можно использовать два различных алгоритма открытого ключа для шифрования и подписи. Как это определяется в сообщении, передаваемом от Алисы к Бобу?
3. Ответьте на следующие вопросы о значениях метки в PGP.
   а. Может ли пакет со значением метки 1 содержать другой пакет?
   б. Может ли пакет со значением метки 6 содержать другой пакет?
4. Какие типы пакетов нужно передать в PGP, чтобы обеспечить следующие услуги безопасности:
   а. Конфиденциальность
   б. Целостность сообщения
   в. Определение подлинности
   г. Исключение отказа от факта принятия сообщения
   д. Комбинация а и б
   е. Комбинация а и в
   ж. Комбинация а, б и в
   з. Комбинация а, б, в и г.
5. Какой тип содержания в S/MIME обеспечивает следующие услуги безопасности:
   а. конфиденциальность
   б. целостность сообщения
   в. установление подлинности
   г. исключение отказа от факта принятия сообщения

Лекция 17. Безопасность на транспортном уровне: SSL И TLS

Цели и содержание

Эта лекция имеет несколько целей.

- Обсудить потребности в услугах безопасности на транспортном уровне модели Интернет.
- Обсудить общую архитектуру SSL.
- Обсудить общую архитектуру TLS.
- Сравнить и показать отличия SSL и TLS.

Безопасность транспортного уровня обеспечивает услуги безопасности «из конца в конец» для приложений, которые используют протоколы транспортного уровня, такие как TCP. Основные идеи предназначены для того, чтобы обеспечить услуги безопасности на сети Интернет. Например, когда в сети имеются интерактивно работающие онлайн(online)-магазины, то желательны следующие услуги безопасности:

1. Клиент должен убедиться, что сервер принадлежит фактическому продавцу, а не самозванцу. Клиент не хочет сообщать самозванцу номер кредитной карточки (установление подлинности объекта).
2. Клиент и продавец должны быть убеждены, что содержание сообщения не изменено в течение передачи (целостность сообщения).
3. Клиент и продавец должны быть убеждены, что самозванец не перехватит чувствительную информацию, такую как номер кредитной карточки (конфиденциальность).

Сегодня применяются в основном два протокола обеспечения безопасности на транспортном уровне: **Протокол «Уровень безопасных розеток»** (SSL — Secure Socket Layer) и **Протокол Безопасности Транспортного уровня** (TLS — Transport Layer Security). Мы сначала обсудим SSL, затем TLS, а потом их сравним и покажем их отличия друг от друга. Рисунок 17.1 показывает место SSL и TLS в модели Интернет (модель протоколов TCP/IP).

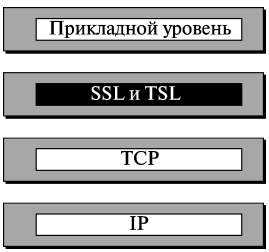


Рис. 17.1. Место SSL и TSL в модели Интернет

Одна из целей этих протоколов состоит в том, чтобы обеспечить сервер и клиента услугами установления подлинности, конфиденциальности и целост-

ности данных. Прикладной уровень программ клиент-сервер (client-server), таких как **Язык передачи гипертекста (HTTP)**, который использует услуги TCP, может инкапсулировать свои данные в пакеты SSL. Если сервер и клиент согласованы с функционирующими программами SSL (или TLS), то клиент может использовать URL https: // ... вместо http:// ..., для того чтобы разрешить сообщениям HTTP инкапсулироваться в пакеты SSL (или TLS). Например, номера кредитной карточки могут быть безопасно переданы через Интернет для онлайн-покупателей.

17.1. SSL-архитектура

SSL разработан, чтобы обеспечить безопасность и услуги сжатия данным, сгенерированным прикладным уровнем. Как правило, SSL может получить данные от любого протокола прикладного уровня, но обычно он получает их от протокола HTTP. Данные, полученные от приложения, сжаты (дополнительно), подписаны и зашифрованы, а затем их передают к протоколу транспортного уровня, такому как TCP. Фирма Netscape разработала SSL в 1994 году . Версии 2 и 3 были выпущены в 1995 году. В этой лекции мы рассмотрим только SSL V. 3.

Услуги

SSL обеспечивает несколько услуг для данных, полученных от прикладного уровня.

Фрагментация
Сначала SSL делит данные на блоки 2^{14} байтов или меньше.

Сжатие
Каждый фрагмент данных сжат с применением одного из методов сжатия без потери методом, согласованным по договору между клиентом и сервером. Эта услуга является дополнительной.

Целостность сообщения
Чтобы сохранять целостность данных, SSL использует ключевую хэш-функцию для создания кода проверки подлинности (MAC).

Конфиденциальность
Чтобы обеспечить конфиденциальность, первоначальные данные и код проверки подлинности (MAC) зашифрованы, с применением криптографии с симметричными ключами.

Организация кадра
К зашифрованной полезной нагрузке добавляется заголовок. Полезную нагрузку затем передают достоверному протоколу транспортного уровня.

Алгоритмы смены ключей

Как мы увидим позднее, для обмена подлинными и конфиденциальными сообщениями клиенту и серверу нужны шесть криптографических объектов секретности (четыре ключа и два вектора инициализации). Однако чтобы создать их, между этими двумя сторонами должен быть установлен один предварительный главный секретный код (pre-master secret). SSL определяет шесть методов обмена ключами, чтобы установить этот предварительный объект секретности: NULL, RSA, анонимный Диффи-Хеллман (Diffie-Hellman), кратковременный Диффи-Хеллман, фиксированный Диффи-Хеллман и Fortezza, как это показано на рис. 17.2.

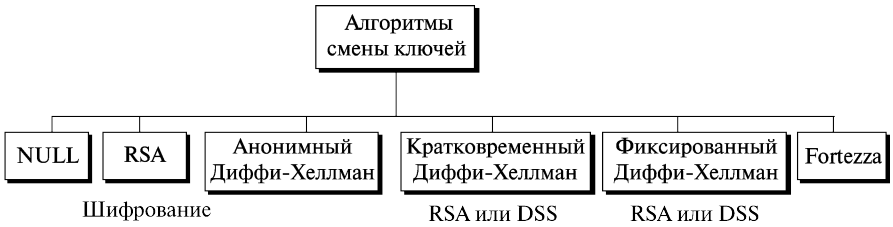


Рис. 17.2. Методы замены ключей

NULL (Пустой указатель)

В этом методе нет никакой смены ключей. Между клиентом и сервером не установлен предварительный главный секретный код.

И клиент и сервер должны знать значение предварительного главного секретного кода.

RSA

В этом методе предварительный главный секретный код — 48-байтовое случайное число, созданное клиентом, зашифрованное открытым ключом RSA-сервера и передаваемое серверу. Сервер должен передать свой сертификат шифрования/дешифрования RSA. Рисунок 17.3 иллюстрирует идею.

Рис. 17.3. RSA-смена ключа; открытый ключ сервера

Анонимный протокол Диффи-Хеллмана

Это самый простой и наиболее ненадежный метод. Предварительный главный секретный код устанавливают между клиентом и сервером, используя протокол Диффи-Хеллмана. При этом передают половину ключа в исходном тексте — это называется **анонимным протоколом Диффи-Хеллмана**, потому что ни одна сторона не известна другой. Как мы уже обсуждали, самый серьезный недостаток этого метода — возможность атаки «посредника». Рисунок 17.4 иллюстрирует идею анонимного метода.

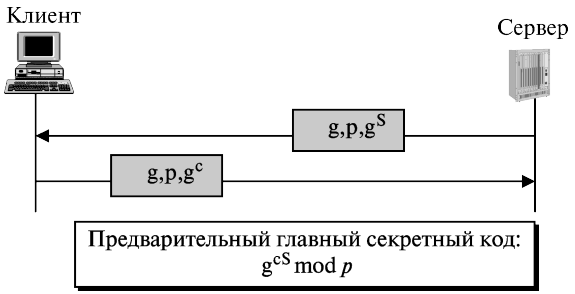


Рис. 17.4. Анонимный протокол Диффи-Хеллмана смены ключей

Кратковременный метод Диффи-Хеллмана

Чтобы сорвать атаку «посредника», может быть применена **кратковременная смена ключей методом Диффи-Хеллмана.** Каждая сторона передает ключ Диффи-Хеллмана, подписанный своим секретным ключом. На приемной стороне должны проверить подпись, используя открытый ключ передатчика. Обмен открытыми ключами для проверки использует либо RSA-, либо DSS-сертификат цифровой подписи. Рисунок 17.5 иллюстрирует идею.

Рис. 17.5. Кратковременный протокол Диффи-Хеллмана смены ключей

Фиксированный метод Диффи-Хеллмана

Другое решение — фиксированный метод Диффи-Хеллмана. Все объекты в группе могут подготовить фиксированные параметры (g и p). Затем каждый объект может создать фиксированную половину ключа (g^x). Для дополнительной безопасности каждая отдельная половина ключа Диффи-Хеллмана вставляется в сертификат, проверенный центром сертификации (CA). Другими словами, две стороны отдельно не обмениваются полуключами; CA передает полуключи в специальном сертификате RSA или DSS. Когда клиент должен вычислить предварительный главный секретный код, он использует свой собственный фиксированный полуключ и полуключ сервера, полученный в сертификате. Сервер делает то же самое, но в обратном порядке. Обратите внимание, что в этом методе не передаются сообщения смены ключей, а происходит только обмен сертификатами.

Fortezza

Fortezza (образован из итальянского слова «крепость») — зарегистрированная торговая марка американского Агентства Национальной безопасности (NSA — National Security Agency). Это семейство протоколов безопасности, разработанных для Отдела Защиты. Мы здесь не обсуждаем Fortezza из-за его сложности.

Алгоритмы шифрования/дешифрования

Есть несколько возможностей выбора алгоритма шифрования/дешифрования. Мы можем разделить алгоритмы на 6 групп, как это показано на рис. 17.6. Все протоколы блока используют 8-байтовый вектор инициализации (IV), кроме Fortezza, который применяет 20 байтов IV.

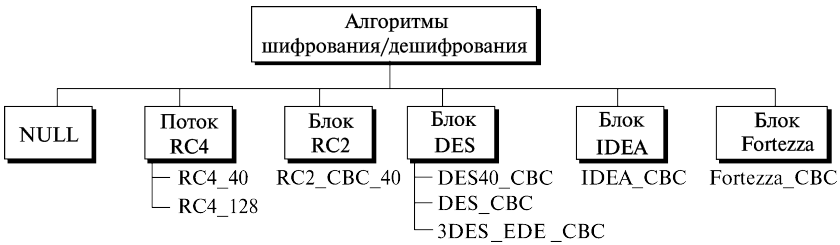


Рис. 17.6. Алгоритмы шифрования/дешифрования

NULL

NULL — категория, которая просто определяет отсутствие алгоритма шифрации/дешифрации.

Блок RC

В блочном режиме RC определены два потока алгоритма: RC4-40 (ключ на 40 битов) и RC4-128 (ключ на 128 битов).

Блок RS

В блочном режиме RC определен один алгоритм: RC2_CBC_40 (ключ на 40 битов). CBC (Cipher Block Chaining) — сцепление шифрованных блоков.

DES

Все алгоритмы DES определены в блочном режиме. DES40_CBC использует ключ на 40 битов. Стандартные DES определены как DES_CBC. 3DES_EDE_CBC используют ключ на 168 битов.

IDEA

В блочном режиме IDEA определен один алгоритм — IDEA_CBC, с ключом на 128 битов.

Fortezza

В блочном режиме Fortezza определен один алгоритм — FORTEZZA_CBC, с ключом на 96 бит.

Алгоритмы хэширования

SSL применяет алгоритмы хэширования, чтобы обеспечить целостность сообщения (установление подлинности сообщения). Имеются хэш-функции, показанные на рис. 17.7.

Рис. 17.7.

Null (Пустой указатель)

Две стороны могут отказаться использовать алгоритм хэширования. В этом случае сообщение не заверено.

MD5

Две стороны могут выбрать MD5 как алгоритм хэширования. В этом случае применяется алгоритм хэширования MD5 — 128-битовый.

SHA-1

Две стороны могут выбрать SHA как алгоритм хэширования. В этом случае используется алгоритм хэширования SHA-1 на 160 битов.

Набор шифров

Комбинация смены ключей, хэширования и алгоритмов шифрования определяет **набор шифров** для каждого сеанса SSL. Таблица 17.1 показывает наборы, применяемые в Соединенных Штатах. Мы не включили наборы, которые используются для экспорта. Обратите внимание, что в списке находятся не все комбинации смены ключей, целостности сообщения и установления подлинности сообщения.

Каждый набор начинается термином «SSL», сопровождаемым алгоритмом смены ключей. Слово «WITH» отделяет алгоритм смены ключей от алгоритмов шифрования и хэширования.

Например,

```
SSL_DHE_RSA_WITH_DES_CBC_SHA
```

определяет DHE_RSA (кратковременный метод Диффи-Хеллмана (Diffie-Hellman ephemeral)) с цифровой подписью RSA для смены ключей, DES_CBC — в качестве алгоритма шифрования и SHA — как алгоритм хэширования.

Обратите внимание, что сокращения DH (Diffie-Hellman) — это фиксированный метод Диффи-Хеллмана, DHE (Diffie-Hellman Ephemeral) — это кратковременный метод Диффи-Хеллмана и DH-anon (anonymous Diffie-Hellman) — это анонимный метод Диффи-Хеллмана.

Таблица 17.1. Список набора шифров SSL

Набор шифров	**Смена ключей**	**Шифрование**	**Хэш**
SSL-NULL-WITH-NULL- NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-1
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
SSL RSA WITH 3DES EDE CBC SHA	RSA	3DES	SHA-1
SSL_DH_anon_WITH_RC4_128_MD5	DH anon	RC4	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH anon	DES	SHA-1
SSL_DH_anon_WITH_3DES_EDE_ CBC SHA	DH anon	3DES	SHA-1
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE RSA	DES	SHA-1
SSL_DHE_RSA_WITH_3DES_ EDE CBC SHA	DHE RSA	3DES	SHA-1
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE DSS	DES	SHA-1
SSL_DHE_DSS_WITH_3DES_EDE_ CBC_SHA	DHE DSS	3DES	SHA-1
SSL_DH_RSA_WITH_DES_CBC_SHA	DH RSA	DES	SHA-1
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH RSA	3DES	SHA-1
SSL_DH_DSS_WITH_DES_CBC_SHA	DH DSS	DES	SHA-1
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH DSS	3DES	SHA-1
SSL_FORTEZZA_DMS_WITH_NULL_SHA	Fortezza	NULL	SHA-1
SSL_FORTEZZA_DMS_WITH_FORTEZZA_ CBC_SHA	Fortezza	Fortezza	SHA-1
SSL_FORTEZZA_DMS_WITH_RC4_ 128_SHA	Fortezza	RC4	SHA-1

Алгоритмы сжатия

Как мы уже говорили, сжатие является дополнительной услугой в SSLv3. Для SSLv3 не определен алгоритм сжатия. Поэтому заданным по умолчанию методом сжатия служит NULL. Однако система может использовать любой алгоритм сжатия по выбору сторон.

Генерирование криптографических параметров

Чтобы обеспечить целостность и конфиденциальность сообщения, в SSL необходимо иметь: шесть криптографических объектов секретности, четыре ключа и два инициализирующих вектора (IV). Клиенту нужно: один ключ для передачи сообщения установления подлинности (HMAC — HASH-BASED MESSAGE AUTHENTICATION CODE), один ключ для шифрования и один IV для шифрования блока. Сервер нуждается в том же самом. SSL требует, чтобы ключи для одного направления отличались от ключей для другого направления. Если будет атака в одном направлении, она не затронет другое направление. Для генерации параметров используют следующую процедуру.

1. Клиент и сервер обмениваются двумя случайными числами, одно из которых создано клиентом, а другое — сервером.
2. Клиент и сервер обмениваются одним предварительным главным секретным кодом, используя один из алгоритмов смены ключей, которые мы обсуждали раньше.
3. Создается 48-байтовый **главный секретный код (master secret)** из **предварительного главного секретного кода (pre-master secret)**, с применением хэш-функций (SHA-1 и MD5), как это показано на рис. 17.8.

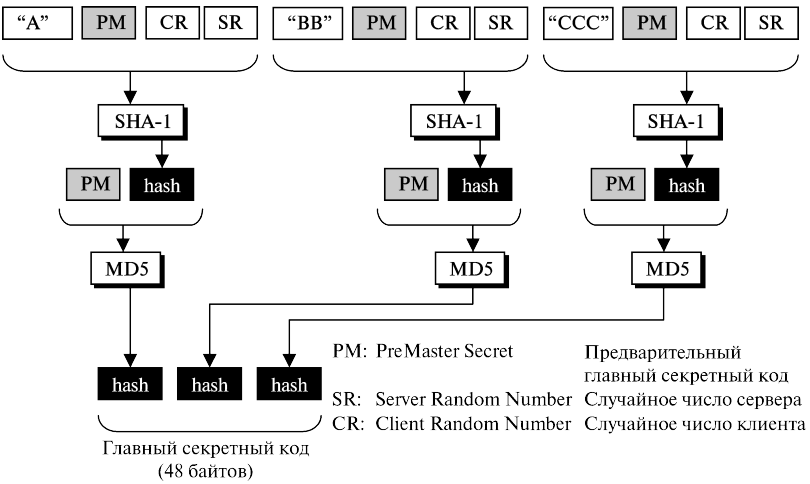
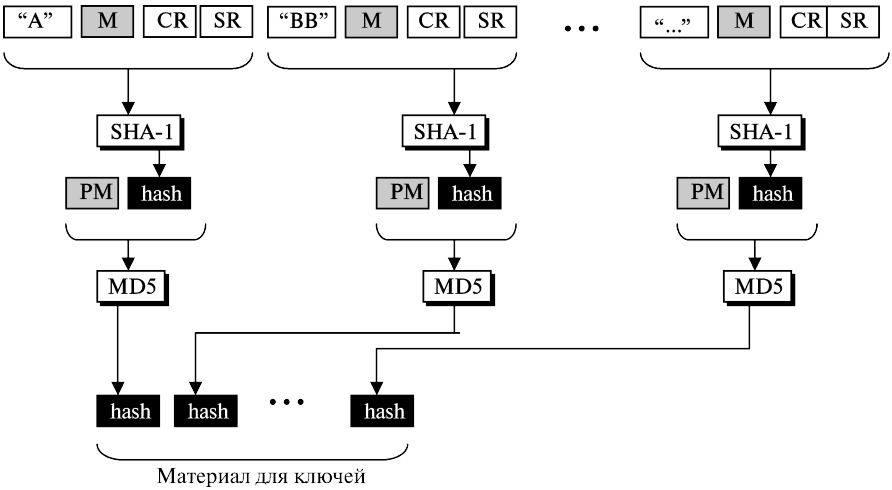


Рис. 17.8 Вычисление главного секретного кода из предварительного главного секретного кода

4. Главный секретный код используется для того, чтобы создать **материал для ключей (key material)**, который имеет переменную длину. Для этого применяют то же самое множество хэш-функций, что и в предыдущем случае, и подставляют спереди различные константы, как это показано на рис. 17.9. Алгоритм повторяется, пока не получится материал для ключа адекватного размера.
   Обратите внимание, что длина блока материала для ключей зависит от выбранного набора шифра и размера ключей, необходимых для этого набора.



M: Master Secret — Главный секретный код
SR: Server Random Number — Случайное число сервера
CR: Client Random Number — Случайное число клиента

Рис. 17.9 Вычисление матриала для ключей из главного секретного кода

5. Из материала для ключей извлекаются шесть различных ключей, как показано на рис. 17.10.

Сеансы и соединение

SSL отличает **соединение** от **сеанса.** Давайте рассмотрим эти два термина. Сеанс — связь между клиентом и сервером. После того как сеанс установлен, эти две стороны имеют общую информацию, такую как идентификатор сеанса, сертификат, подтверждающий подлинность каждого из них (в случае необходимости), метод сжатия (если необходимо), набор шифров и главный секретный код. Эта информация используется для того, чтобы создать ключи для сообщения, содержащего шифр установления подлинности.

Для двух объектов, чтобы начать обмен данными, установление сеанса необходимо, но не достаточно; они должны создать между собой соединение. Два объ-

Auth Key	Authentication Key	Ключ аутентификации
Enc. Key:	Encryption Key	Ключ Шифрования
IV	Initialization Vector	Вектор Инициализации

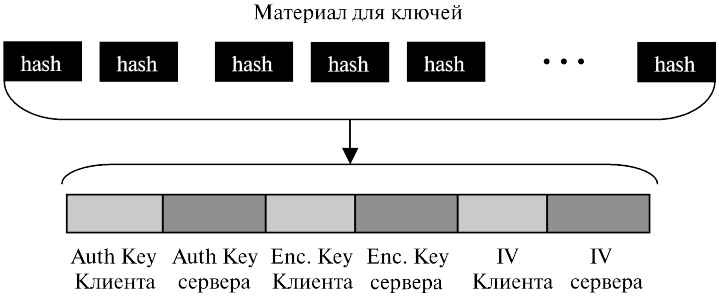


Рис. 17.10 Извлечение криптографических секретных кодов из материала для ключей

екта обмениваются двумя случайными числами и создают, используя главный секретный код, ключи и параметры, необходимые для того, чтобы обмениваться сообщениями, включая установление подлинности и секретность.

Сеанс может состоять из многих соединений. Соединение между двумя сторонами может быть закончено и восстановлено в пределах одного и того же сеанса. Когда соединение закончено, эти две стороны могут также закончить сеанс, но это необязательно. Сеанс может быть приостановлен и продолжен снова.

Чтобы создавать новый сеанс, эти две стороны должны пройти процесс переговоров. Чтобы возобновлять старый сеанс и создавать только новое соединение, эти две стороны могут пропустить часть переговоров, что уменьшает время вхождения в связь. Не надо создавать главный секретный код, когда сеанс продолжается.

Разделение сеанса от соединения предотвращает высокую стоимость создания главного секретного кода. Если мы разрешаем приостановления и продолжения сеанса, процесс вычисления главного секретного кода может быть устранен. Рисунок 17.11 иллюстрирует идею сеанса и соединения в этом сеансе.

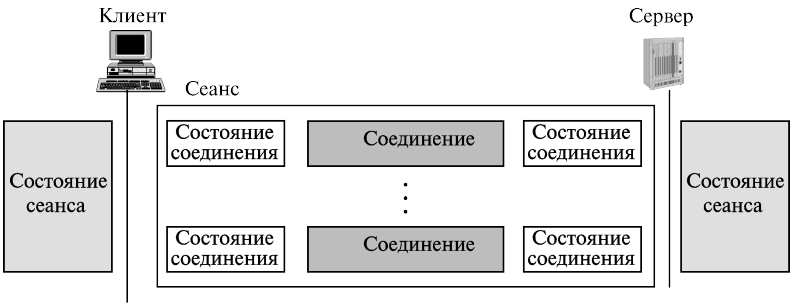


Рис. 17.11. Сеанс и соединение

В сеансе одна сторона играет роль клиента и другая — роль сервера. При соединении обе стороны имеют равные роли, они равны по уровню.

Состояние сеанса

Сеанс определяется состоянием сеанса — это множество параметров, установленных между сервером и клиентом. Таблица 17.2 показывает список параметров для состояния сеанса.

Таблица 17.2. Параметры состояния сеанса

Параметры	Описание
ID сеанса	Случайное 8-битовое число, выбранное сервером и определяющее сеанс
Сертификат уровня	Сертификат типа X509 .v.3. Этот параметр может быть пустым (null)
Метод сжатия	Метод сжатия
Набор шифров	Согласованный набор шифров
Главный секретный код	48-байтовый секретный код
Возможность повторения	Флаг «Да, Нет», который разрешает новое соединение в старом сеансе

Состояние соединения

Подключение определяется состоянием соединения — это множество параметров, установленных между двумя равными по уровню объектами. Таблица 17.3 показывает список параметров для состояния соединения.

SSL использует два признака, чтобы отличить криптографическую секретность: *писать* и *читать*. Термин *писать* определяет ключ, используемый для того, чтобы подписать или зашифровать исходящее сообщение. Термин *читать* определяет ключ, используемый для того, чтобы подтвердить или расшифровывать прибывающие сообщения. Обратите внимание: *писать*-ключ клиента — тот же самый, что и ключ-*читать* сервера; ключ-*читать* клиента — тот же самый, что и ключ-*писать* сервера.

Клиент и сервер имеют шесть различных криптографических объекта: три объекта секретности *читать* и три *писать*. Секретность *читать* для клиента та же самая, что и секретность *писать* для сервера, и наоборот.

Таблица 17.3. Параметры состояния соединения

Параметры	Описание
Случайные числа клиента и сервера	Последовательность байтов, выбранная для каждого соединения серверу и клиенту

Записанный сервером секретный код подлинности сообщения	Ключ кода установления подлинности сообщения исходящего сервера для сохранения целостности сообщения.. Используется сервером для подписи, а клиентом для верификации
Записанный клиентом секретный код установления подлинности сообщения	Ключ кода установления подлинности сообщения исходящего сервера для сохранения целостности сообщения.. Используется сервером для подписи, а клиентом для верификации
Секретный код, записанный сервером	Ключ шифрования исходящего сервера для сохранения целостности сообщения
Секретный код, записанный клиентом	Ключ шифрования исходящего сервера для сохранения целостности сообщения
Вектор инициализации	Блочные шифры в режиме «цепочки блочных шифров» I (CBC) используют векторы инициализации. (IV). Для каждого шифровального ключа путем переговоров определен один вектор инициализации, который используется первым блоком обмена ключами. Зашифрованный текст из блока используется как вектор инициализации (IV) для следующего блока
Порядковый номер	Каждая сторон имеет порядковый номер. Он начинается с 0 и увеличивается на 1. Он не должен быть больше $2^{64} - 1$

17.2. Четыре протокола

Мы обсудили идею относительно SSL, не показав, как SSL выполняет свои задачи. SSL содержит четыре протокола на двух уровнях, как это изображено на рис. 17.12. Протокол передачи записей — переносящий информацию. Он переносит на транспортный уровень сообщения от трех других протоколов, а также данные, поступающие от прикладного уровня. Сообщения из протокола записей — это полезная нагрузка для транспортного уровня, обычно TCP. Протокол установления соединения обеспечивает параметры безопасности для Протокола записей. Он устанавливает набор шифров и задает ключи и параметры безопасности.

Он также подтверждает, если необходимо, подлинность сервера клиенту и подлинность клиента серверу. Протокол изменения параметров шифрования используется, чтобы передавать сигналы для подготовки к криптографической безопасности. Аварийный протокол нужен, чтобы известить о ситуациях, отклоняющихся от нормы. Все эти протоколы мы кратко рассмотрим в этой секции.

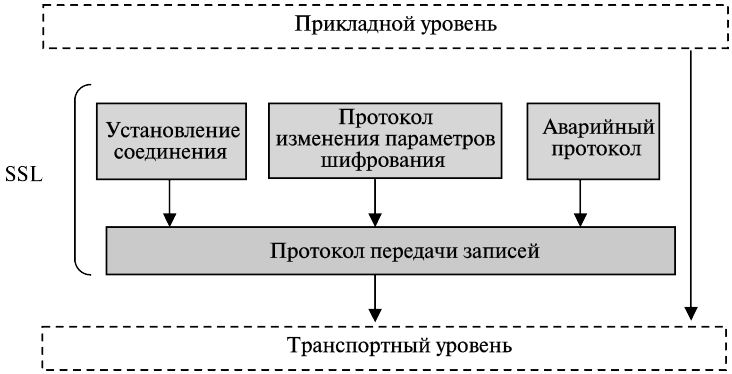


Рис. 17.12. Четыре протокола SSL

Протокол установления соединения

Протокол установления соединения используется при передаче сообщений, чтобы договориться, если это необходимо, о составе шифров от сервера к клиенту и от клиента к серверу и обменяться информацией ради обеспечения криптографической безопасности. Процедура установления связи происходит в 4 фазы; она проиллюстрирована на рис. 17.13.

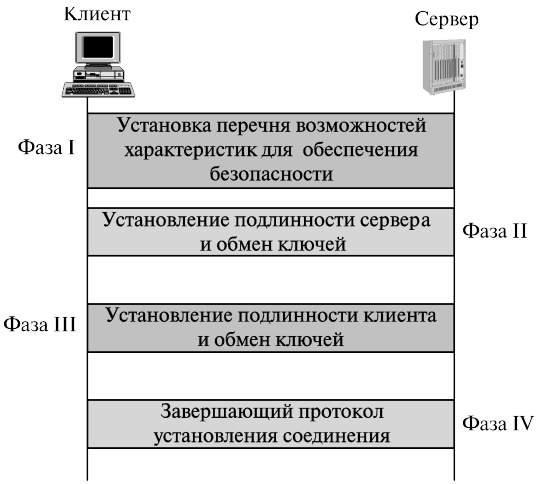


Рис. 17.13. Протокол установления соединения

Фаза 1: Установление характеристик для обеспечения безопасности

В Фазе 1 клиент и сервер объявляют свои характеристики безопасности, которые нужны и удобны для обоих. В этой фазе устанавливается и выбирается ID

сеанса. Стороны согласуют конкретный метод сжатия. Наконец, выбирают два случайных числа: одно выбирается клиентом и другое — сервером, чтобы создать главный секретный код. В этой фазе стороны обмениваются двумя сообщениями: ClientHello и ServerHello. Рисунок 17.14 содержит дополнительные детали Фазы 1.

ClientHello. Клиент посылает сообщение. Оно содержит следующую информацию.

а. Самый высокий номер версии SSL, которую может поддержать клиент.
б. 32-байтовое случайное число (от клиента), которое будет использоваться для генерации главного секретного кода (мастер кода).
в. ID сеанса, который определяет сеанс.
г. Набор шифров, который определяет список алгоритмов, поддерживаемых клиентом.
д. Список методов сжатия, которые клиент может поддержать.

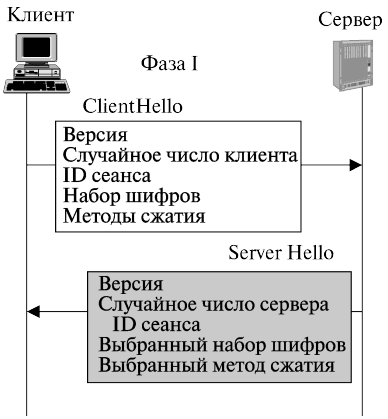


Рис. 17.14. Протокол установления соединения, Фаза I

ServerHello. Сервер отвечает клиенту ссообщением ServerHello, оно содержит:

а. Номер версии SSL. Это два номера версии: наиболее высокий номер, поддерживаемый клиентом, и наиболее высокий, поддерживаемый сервером.
б. 32-байтовое случайное число (от сервера), которое будет использоваться для генерации главного секретного кода (мастер кода).
в. ID сеанса, который определяет сеанс.
г. Выбранный шифр из списка клиента.
д. Выбранный метод сжатия из списка клиента.

После Фазы 1 клиент и сервер знают следующее:

- **Версия SSL**
- **Алгоритмы для смены ключей, установления подлинности сообщения и шифрования**
- **Метод сжатия**
- **Два случайных числа для генерации ключей**

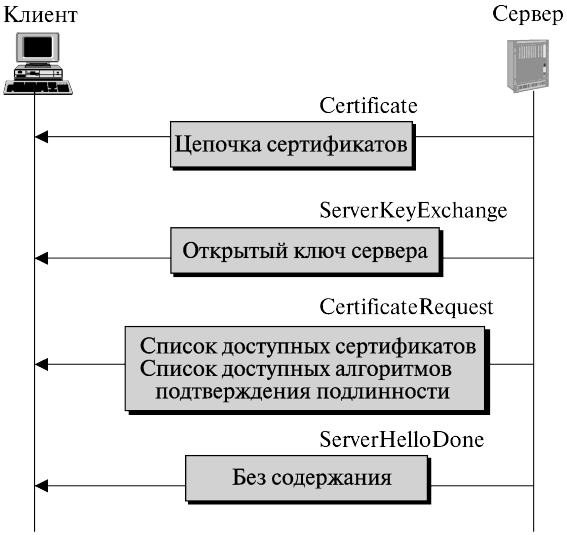


Рис. 17.15. Протокол установления соединения, Фаза II

Фаза II: Смена ключей сервера и установление его подлинности

В фазе II сервер, если необходимо, подтверждает свою подлинность. Передатчик может передать свой открытый ключ и может также запросить сертификат клиента. В конце сервер объявляет, что процесс serverHello окончен. Рисунок 17.15 дает дополнительные сведения о Фазе II.

Certificate (сертификат) — если это требуется, сервер передает сообщение **certificate**, чтобы подтвердить свою подлинность. Сообщение включает список сертификатов типа X.509. Если алгоритм смены ключей — анонимный Диффи-Хеллман (Diffie-Hellman), то сертификат не нужен.

ServerKeyExchange. После сообщения **Certificate** сервер передает сообщение ServerKeyExchange, которое включает в себя его вклад в предварительный главный секретный код. Если метод смены ключей — RSA или метод «фиксированный Диффи-Хеллман», то такого сообщение не требуется.

CertificateRequest — сервер может потребовать, чтобы клиент подтвердил свою подлинность. В этом случае сервер передает CertificateRequest сообщение в Фазе II, в котором запрашивает от клиента подтверждение в Фазе III. Сервер не может запросить сертификат от клиента, если клиент использует метод «анонимный Диффи-Хеллман».

ServerHelloDone — последнее сообщение в Фазе II. Оно является сигналом клиенту, что Фаза II закончена и что клиент должен запустить Фазу III.

После Фазы II

- **Клиенту подтверждена подлинность сервера.**
- **Если требуется, то клиент знает открытый ключ сервера.**

Давайте тщательно рассмотрим установление подлинности сервера и смену ключей в этой фазе. Первые два сообщения здесь базируются на методе смены ключей. Рисунок 17.16 показывает четыре из шести методов обмена ключей, которые мы обсуждали раньше. Мы не включили Нулевой (NULL) метод, потому что в нем нет никакого обмена. Мы не включили метод Fortezza, потому что в этой книге мы его глубоко не рассматривали.

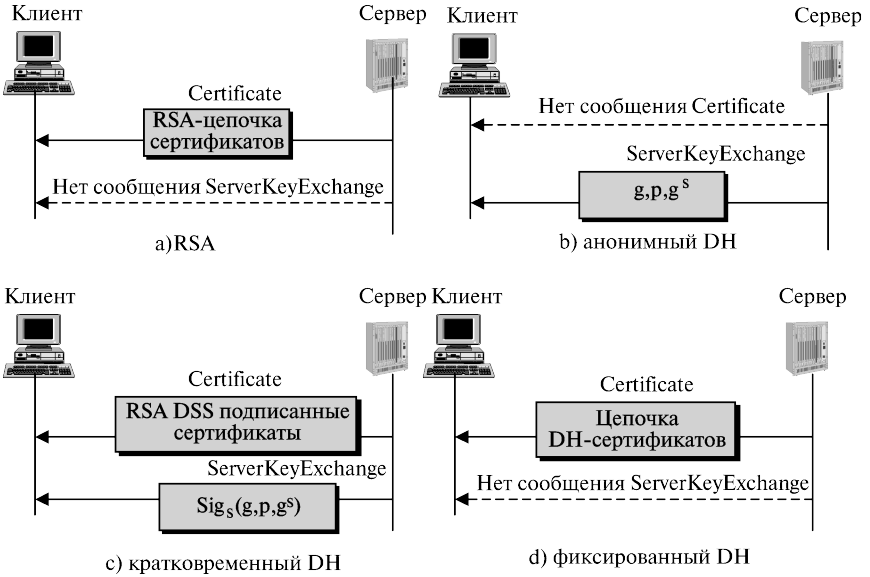


Рис. 17.16. Четыре случая Фазы II

- **RSA.** В этом методе в первом сообщении сервер передает сертификат своего открытого ключа RSA шифрования/дешифрования. Второе сообщение, однако, пустое, потому что не сгенерирован предварительный главный секретный код — он передается клиентом в следующей фазе. Обратите внимание, что сертификат открытого ключа подтверждает подлинность сервера клиенту. Когда сервер получает предварительный главный секретный код, он расшифровывает его секретным ключом. Владение секретным ключом для сервера — доказательство, что сервер — это тот объект, который находился в сертификате открытого ключа, переданном в первом сообщении.
- **Анонимный DH (DH_{anonym}).** В этом методе не требуется сообщения **Certificate**. Анонимный объект не имеет сертификата. В ServerKeyExchange сообщении сервер передает параметры метода Диффи-Хеллмана и свой полуключ. Обратите внимание, что здесь сервер не подтверждает свою подлинность.
- **Кратковременный DH (DHE).** В этом методе сервер передает либо RSA-, либо DSS-сертификат цифровой подписи. Секретный ключ, связанный с

сертификатом, позволяет серверу подписать сообщение; открытый ключ позволяет получателю проверить подпись. Во втором сообщении сервер передает параметры Диффи-Хеллмана и полуключ, подписанный его секретным ключом. В этом методе сервер подтверждает клиенту свою подлинность не потому, что он передает сертификат, а потому, что подписывает параметры и ключи своим секретным ключом. Владение секретным ключом — доказательство того, что сервер — объект, который находится в сертификате. Если самозванец копирует и передает сертификат клиенту, симулируя, что он — сервер, запрошенный в сертификате, он не сможет подписать второе сообщение, потому что не имеет секретного ключа.

- **Фиксированное DH (DH).** В этом методе сервер передает либо RSA-, либо DSS-сертификат цифровой подписи, в который включает свой зарегистрированный полуключ DH. Второе сообщение — пустое.
  Сертификат подписан секретным ключом CA и может быть проверен клиентом, использующим открытый ключ CA. Другими словами, CA подтвердил клиенту подлинность и заверил, что полуключ принадлежит серверу.

Фаза III: Смена ключей клиента и установление его подлинности

Фаза III предназначена подтвердить подлинность клиента. От клиента серверу можно передать до трех сообщений, как показано на рис. 17.17.

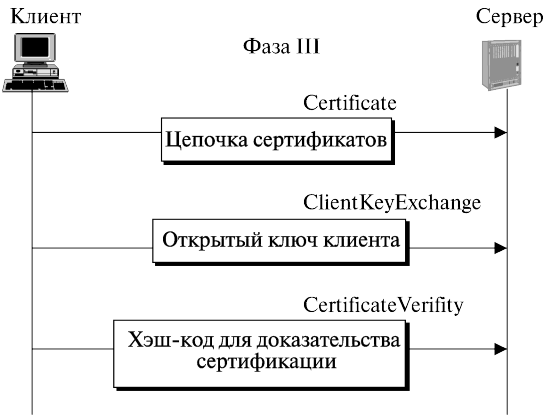


Рис. 17.17. Протокол установления соединения, Фаза III

Сертификат. Чтобы сертифицировать себя на сервере, клиент передает сообщение Certificate. Обратите внимание, что его формат — тот же самый, что и у сообщения Certificate, передаваемого сервером в Фазе II, но содержание различно. В данном случае Certificate включает цепочку сертификатов, которые сертифицируют клиента. Такое сообщение передают, только если сервер запросил сертификат в Фазе II. Если есть запрос и клиент не имеет сертификата, чтобы передать его, тогда передается аварийное сообщение (часть аварийного протокола, кото-

рый будет обсуждаться позже) с предупреждением, что сертификата нет. Сервер может продолжить сеанс или может решить прервать его.

ClientKeyExchange. После передачи сообщения Certificate клиент передает сообщение ClientKeyExchange, которое включает в себя вклад в предварительный главный секретный код. Содержание этого сообщения базируется на используемом алгоритме смены ключей. Если метод — RSA, клиент создает полный предварительный главный секретный код и зашифровывает его открытым ключом RSA сервера. Если метод — анонимный или кратковременный Диффи-Хеллман, клиент передает свой полуключ. Если метод — Fortezza, клиент передает параметры Fortezza. Содержание этого сообщения пусто, если метод — «фиксированный Диффи-Хеллман».

Верификация сертификата. Если клиент передал сертификат, объявляющий, что имеет открытый ключ в сертификате, он должен доказать, что знает соответствующий секретный ключ. Это необходимо, чтобы сорвать попытки самозванца, который передает сертификат и утверждает, что он исходит от клиента. Доказательство владения секретным ключом он представляет, создавая сообщение и подписывая его секретным ключом. Сервер может проверить сообщение переданным ему открытым ключом, чтобы гарантировать, что сертификат фактически принадлежит клиенту. Обратите внимание, что это возможно, если в сертификат включены необходимые подписанные полномочия; в том числе пара ключей — открытый и секретный. Сертификат для фиксированного Диффи-Хеллмана не может быть проверен таким путем.

После Фазы III

- **Клиент проверен на подлинность сервером.**
- **Клиент и сервер знают предварительный главный секретный код.**

Рассмотрим более подробно установление подлинности клиента и смену ключей в этой фазе. Основой данного метода здесь являются три сообщения. Рисунок 17.18 показывает четыре из шести методов, которые мы рассматривали раньше. Опять мы исключили метод NULL и метод Fortezza.

- **RSA.** В этом случае не передается сообщение Certificate, если сервер явно не запросил его в Фазе II. ClientKeyExchange включает в себя предварительный главный серетный код, который зашифрован открытым ключом RSA, полученным в Фазе II.
- **Анонимный DH.** В этом методе не передается сообщение Certificate. Сервер не имеет права запросить сертификат в Фазе II, потому что и клиент и сервер — анонимны. В ClientKeyExchange-сообщении сервер передает параметры Диффи-Хеллмана и свой полуключ. Обратите внимание, что в этом методе клиент не проверен на подлинность сервером.
- **Кратковременный DH.** В этом методе клиент обычно имеет сертификат. Сервер должен передать его RSA- или DSS-сертификат (на основе согласованного множества шифров). В ClientKeyExchange сообщении клиент подписывает параметры **DH**, а также и свой полуключ, и передает их. Подлинность клиента подтверждается серверу подписью второго сообщения. Если клиент не имеет сертификата, а сервер запрашивает его, клиент пере-

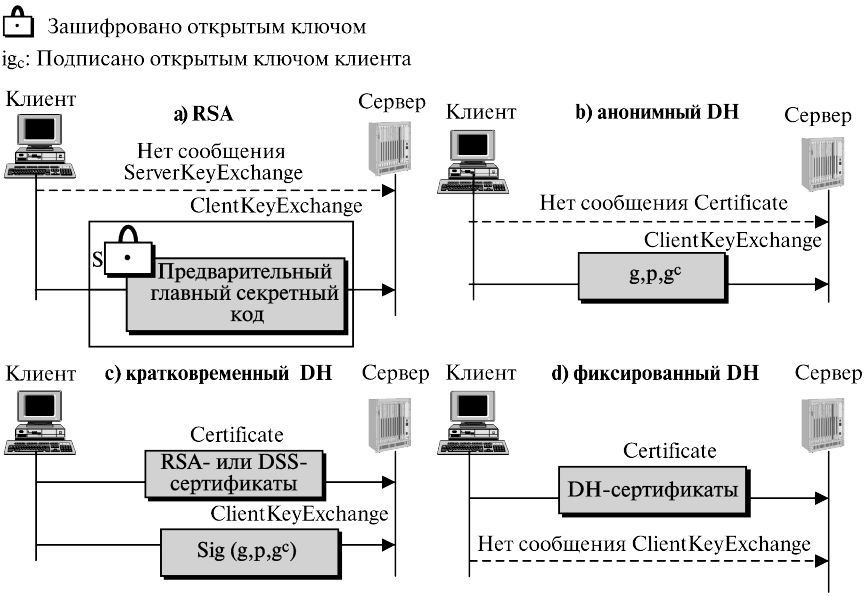


Рис. 17.18. Четыре случая Фазы III

дает аварийное сообщение. Если это приемлемо для сервера, клиент передает параметры DH и ключ в исходном тексте. Конечно, в этой ситуации клиент не проверен сервером на подлинность.

- **Фиксированный DH.** В этом методе клиент обычно передает сертификат **DH** в первом сообщении. Обратите внимание, что здесь второе сообщение пустое. Клиент подтверждает свою подлинность серверу, посылая сертификат **DH**.

Фаза IV: Завершение и окончание

В Фазе IV клиент и сервер передают сообщения, чтобы изменить спецификацию шифра и закончить процедуру установления связи. В этой фазе происходит обмен четырьмя сообщениями, как это показано на рис. 17.19.

ChangeCipherSpec. Клиент передает сообщение ChangeCipherSpec, чтобы показать, что он передал весь набор шифров и параметры для перехода из состояния ожидания в активное состояние. Это сообщение — фактически часть протокола ChangeCipherSpec, который мы обсудим позже.

Finished. Это сообщение также передает клиент. Сообщение Finished объявляет об окончании процедуры установления связи клиентом.

ChangeCipherSpec. Сервер передает ChangeCipherSpec-сообщение, чтобы показать, что он также обменялся набором шифров и параметрами для перехода из состояния ожидания в активное состояние. Это сообщение — часть протокола ChangeCipherSpec, который будет рассмотрен позже.

Finished. Наконец, сервер передает сообщение Finished, чтобы показать, что процедура установления связи полностью закончена.

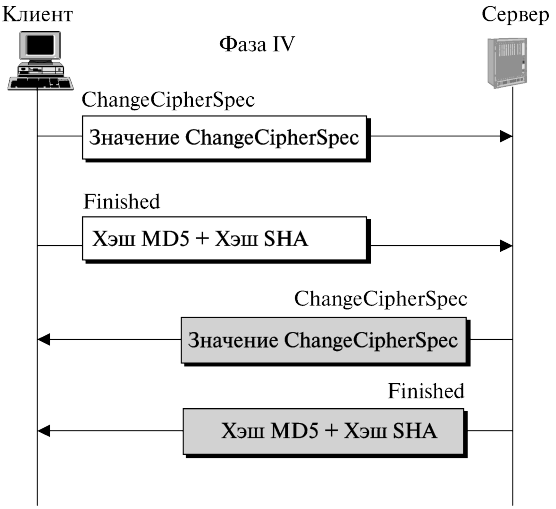


Рис. 17.19. Протокол установления соединения, Фаза IV

Протокол ChangeCipherSpec

Мы видели, что переговоры о наборе шифров и генерация криптографической секретности формируются постепенно по ходу выполнения протокола установления соединения. Возникает вопрос: когда две стороны могут использовать эти параметры секретности? SSL утверждает, что стороны не могут использовать эти параметры или секретность, пока они не передали или не получили специальное сообщение: ChangeCipherSpec — сообщение, которым обмениваются клиент и сервер в ходе выполнения протокола установления соединения и которое определено в протоколе ChangeCipherSpec. По этой причине объекты не посылают или не получают сообщение. Передатчик и приемник нуждаются не в одном, а в двух состояниях. Одно — состояние ожидания, при котором сохраняются параметры и секретности. Другое — активное состояние, при котором параметры и секретность используются протоколом передачи записей, чтобы подписаться/проверить или зашифровать/расшифровывать сообщения. Кроме того, каждое состояние содержит два множества значений: *читать* (входящее), *писать* (исходящее).

Протокол ChangeCipherSpec определяет процесс перемещения значений между состоянием ожидания и активным состоянием. Рисунок 17.20 показывает гипотетическую ситуацию, с гипотетическими значениями, чтобы только проиллюстрировать концепцию.

На рисунке показано только несколько параметров. Перед обменом сообщениями ChangeCipherSpec в состоянии ожидания заполнены только значения столбцов.

Сначала клиент передает ChangeCipherSpec-сообщение. После этого клиент перемещает параметры «писать» (исходящие) из состояния ожидания в активное состояние. Теперь он может использовать эти параметры, чтобы подписаться или

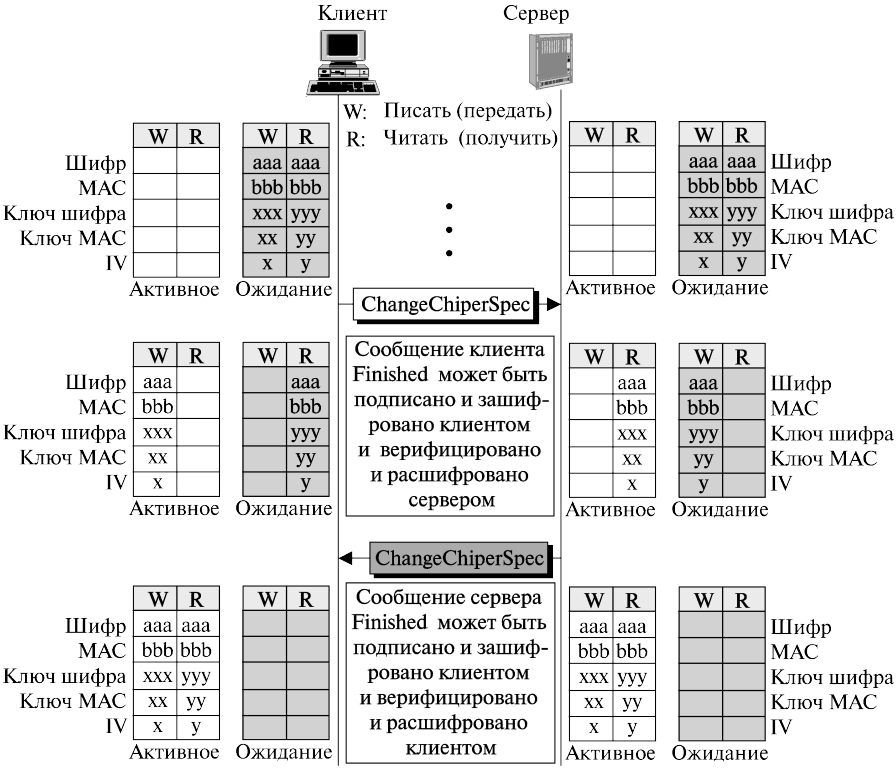


Рис.17.20. Передвижение параметров из состояния ожидания в активное состояние

зашифровать исходящее сообщение. После получения этого сообщения сервер перемещает параметры «читать» (входящие) из состояния ожидания в активное состояние. Теперь сервер может верифицировать и расшифровывать сообщение. Это означает, что сообщение Finished, передаваемое клиентом, может быть подписано клиентом и расшифровано сервером.

Сервер передает сообщение ChangeCipherSpec после получения от клиента сообщения Finished. После передачи этого сообщения он перемещает параметры «писать» (входящие) из состояния ожидания в активное состояние. Сервер может теперь использовать эти параметры для того, чтобы подписать или зашифровать исходящие сообщения. После того как клиент получает это сообщение, он перемещает первый абзац (входящий) из состояний ожидания в активное состояние. Теперь клиент может проверить (верифицировать) и расшифровать сообщения.

Конечно, после обмена сообщениями Finished обе стороны могут работать в обоих направлениях, используя активные параметры для чтения/записи.

Аварийный протокол

SSL использует **аварийный протокол** для того, чтобы известить об ошибках и ненормальном состоянии устройств. Имеется только один тип аварийного сообщения, которое описывает проблему и ее уровень (опасное или полный выход из строя). Таблица 17.4 показывает типы аварийных сообщений, определенных для SSL.

Таблица 17.4. Аварийные сообщения, определенные для SSL

Цифровое обозначение	Описание	Значение
0	CloseNotify	Передатчик больше не будет посылать сообщений
10	UnexpectedMessage	Получено несоответствующее сообщение
20	BadRecordMAC	Получено некорректное MAC
30	DecompressionFailure	Невозможно получить несжатый текст
40	HandshakeFailure	Передатчик не может закончить установление соединения
41	NoCertificate	Клиент не сертифицировал послание
42	BadCertificate	Полученный сертификат искажен
43	UnsupportedCertificate	Тип полученного сертификата не поддерживается
44	CertificateRevoked	Подписавший аннулирует сертификат
45	CertificateExpired	Сертификат просрочен
46	CertificateUnknown	Сертификат неизвестен
47	Illegal Parameter	Поле не может быть обработано

Протокол передачи записей

Протокол передачи записей доставляет сообщения от верхнего уровня (протокол установления соединения, ChangeCipherSpec-протокол, аварийный протокол) или прикладных уровней. Сообщения фрагментированы и произвольно сжаты; MAC добавляется к сжатому сообщению, используя согласованный алгоритм хэш. Сжатый фрагмент и MAC зашифрованы с применением согласованного алгоритма шифрования. В конце к зашифрованному сообщению добавляется заголовок SSL. Рисунок 17.21 иллюстрирует этот процесс в передатчике. Процесс в приемнике имеет обратный порядок.

Обратите внимание, однако, что этот процесс может быть выполнен, только когда криптографические параметры находятся в активном состоянии. Сообщения, передаваемые перед перемещением из состояния ожидания в активное состояние, не подписаны и не зашифрованы.

В следующих секциях мы увидим некоторые другие сообщения в протоколе установления соединения, которые используются некоторыми определенными типами хэширования для обеспечения целостности сообщения.

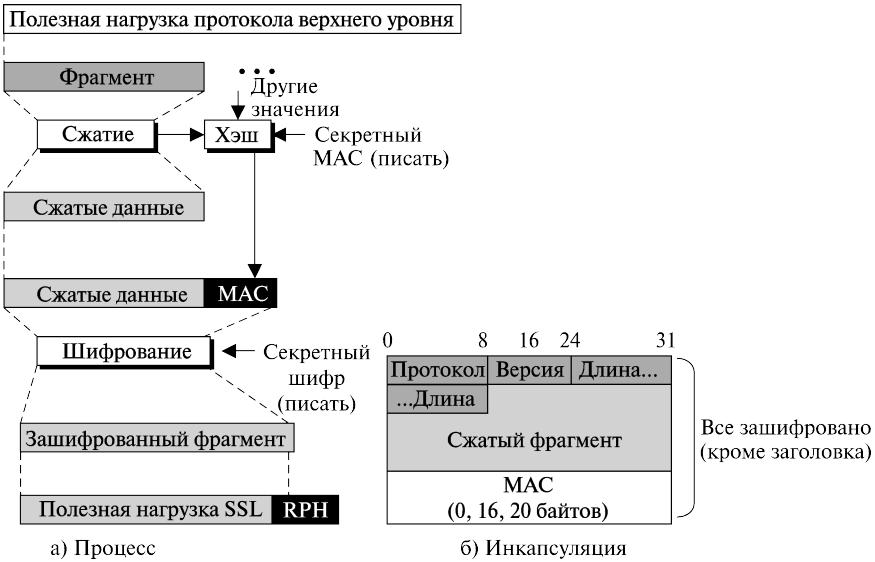


Рис. 17.21. Процесс работы протокола передачи записей

Фрагментация/Объединение

В передатчике сообщение от прикладного уровня фрагментируется в блоки по 2 байта, при этом последний блок может быть меньше. В приемнике фрагменты объединяются вместе, чтобы создать точную копию первоначального сообщения.

Сжатие/Расширение

В передатчике все фрагменты прикладного уровня сжаты с использованием метода сжатия, о котором договариваются во время процедуры установления связи. Метод сжатия не должен вносить потери (расширенный фрагмент должен быть точной копией первоначального фрагмента). Размер фрагмента не должен превысить 1024 байта. Некоторые методы сжатия работают только с заранее заданными размерами блоков, и если размер блока меньше, то блок дополняется. Поэтому размер сжатого фрагмента может быть больше, чем размер первоначального фрагмента. В приемнике сжатый фрагмент расширяется, чтобы создать точную копию оригинала. Если размер расширенного фрагмента превышает 2^{14}, вырабатывается аварийное сообщение «неправильное расширение» (fatal decompression). Обратите внимание, что сжатие/расширение в SSL — дополнительные функции.

Подписание/Подтверждение

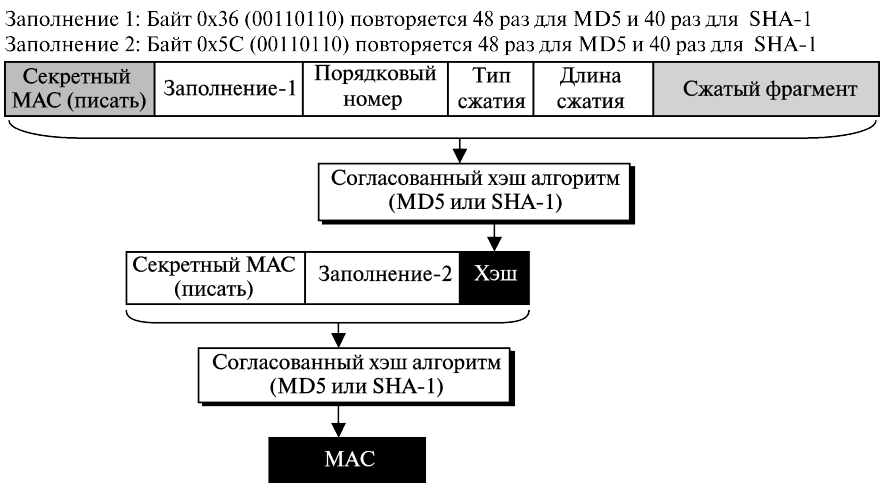


Рис. 17.22. Вычисление MAC

В передатчике метод подтверждения подлинности определяется в течение установления соединения (NULL, MD5 или SHA-1) и создает подпись (MAC), как это показано на рис. 17.22.

Алгоритм хэширования применяется дважды. Сначала хэширование создается путем последовательных соединений (конкатенации) следующих значений:

а. секретный MAC (писать) (ключ подтверждения подлинности для исходящих сообщений);

б. заполнение 1, которое представляет байт 0x36, повторяемый 48 раз для MD5 и 40 раз для SHA-1;

в. порядковый номер этого сообщения;

г. тип сжатия, который определяет протокол верхнего уровня, обеспечившего сжатие фрагмента;

д. длина сжатия, которая указывает длину сжатого фрагмента;

ж. сжатый фрагмент.

Второй раз завершающее хэширование (MAC) создается последовательным соединением (конкатенацией) следующих значений:

а. секретный MAC (писать);

б. заполнение 2, которое представляет байт 0x5C, повторяемый 48 раз для MD5 и 40 раз для SHA-1;

в. создание хэша из результата первого шага.

В приемнике проводится верификация — вычисление нового хэша и сравнение его с полученным хэшированием.

Шифрование/Дешифрование

В передатчике сжатый фрагмент и хэш зашифрованы с применением секретного шифра (писать). В приемнике полученное сообщение расшифровывается с

использованием секретного шифра (читать). При шифровании к блоку добавляется заполнение, чтобы сделать размер сообщения пригодным для шифрования — кратным числом размера блока.

Создание кадра

В передатчике после шифрования добавляется заголовок протокола передачи записей. Заголовок удаляется в приемнике перед дешифрованием.

17.3. Форматы сообщения SSL

Мы уже знаем,что сообщения из трех протоколов и данные от прикладного уровня инкапсулируются (вставляются) в сообщения протокола передачи записей. Другими словами, сообщение в протокол передачи записей инкапсулируется из четырех различных источников на стороне передатчика. На стороне приемника протокол передачи записей извлекает сообщения и доставляет их различным

пунктам назначения. Протокол передачи записей имеет общий заголовок, который добавляется к каждому сообщению, прибывающему от источников, как показано на рис. 17.23.

Рис. 17.23. Общий заголовок протокола передачи записей

Ниже приводится описание полей.

- **Протокол.** Это 1-байтовое поле определяет источник или пункт назначения инкапсулированного сообщения. Оно используется для мультиплексирования и демультиплексирования. Значения — 20 (ChangeCipherSpec-протокол), 21 (аварийный протокол), 22 (протокол установления соединения) и 23 (данные от прикладного уровня).
- **Версия.** Это 2-байтовое поле определяет версию SSL; один байт — первая цифра версии и другой — вторая. Текущая версия SSL — 3.0.
- **Длина.** Это 2-байтовое поле определяет размер сообщения (без заголовка) в байтах.

ChangeCipherSpec-протокол

Как мы говорили раньше, протокол ChangeCipherSpec имеет одно сообщение: ChangeCipherSpec. Это сообщение содержит только один байт, инкапсулированный в сообщение протокола передачи записей со значением 20, как показано на рис. 17.24.

Рис. 17.24. Сообщение ChangeCipherSpec (CCS)

Однобайтовое поле в сообщении названо *CCS*, и его значение в настоящее время равно 1.

Аварийный протокол

Аварийный протокол, как мы говорили прежде, имеет одно сообщение — рапорт об ошибках в процессе. Рисунок 17.25 показывает инкапсуляцию этого единственного сообщения в Протоколе передачи записей со значением 21.

Рис. 17.25. Аварийное сообщение

Два поля аварийного сообщения разъясняются ниже.

- **Уровень.** Это однобайтовое поле определяет уровень ошибки. В настоящее время определены два уровня: предупреждение и полный отказ.
- **Описание.** Однобайтовое описание определяет тип ошибки.

Протокол установления соединения

Несколько сообщений были определены для протокола установления соединения. Все эти сообщения имеют четырехбайтовый типовой заголовок, показанный на рис. 17.26. Рисунок приводит заголовок протокола передачи записей и типовой заголовок для протокола установления соединения. Обратите внимание, что значение поля протокола — 22.

Рис. 17.26. Типовой заголовок в протоколе установления соединения

- **Тип.** Это однобайтовое поле определяет тип сообщения. В настоящее время определены десять типов, которые перечислены в таблице 17.5.

Таблица 17.5. Типы сообщений установления соединения

Тип	Сообщение
0	HelloRequest
1	ClientHello
2	ServerHello
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

- **Длина.** Это трехбайтовое поле определяет длину сообщения (исключая длину типа и поля длины). Читатель может задать вопрос: почему мы нуждаемся в двух полях длины — одном в общем заголовке протокола передачи записей и одном в типовом заголовке для сообщений установления соединения? Ответ: сообщение протокола передачи записей может доставить два сообщения установления соединения в одно и то же время, если нет потребности передать другое сообщение между ними.

Сообщение HelloRequest

Сообщение HelloRequest, которое используется редко, является запросом от сервера к клиенту для перезапуска сеанса. Это может быть необходимо, если в

сервере обнаружены сбои и необходим новый сеанс. Например, если сеанс становится таким длинным, что это угрожает его безопасности, сервер может передать рассматриваемое сообщение. Клиент тогда должен передать сообщение ClientHello и договориться о параметрах безопасности. Рисунок 17.27 показывает формат такого сообщения. Это — четыре байта со значением типа 0. Он не имеет никакого текстового блока, так что значение поля длины — также 0.

Сообщение ClientHello

Сообщение ClientHello — первое сообщение в обмене при установлении соединения. На рис. 17.27 показан формат сообщения.

Рис. 17.27. Сообщение HelloRequest

Поле «тип» и поле длины предварительно были уже обсуждены. Ниже приводится краткое описание других полей.

Рис. 17.28. Сообщение ClientHello

- **Версия.** Это 2-байтовое поле показывает версии используемого SSL. Версия 3.0 — для SSL и 3.1 — для TLS. Обратите внимание, что значение версии, например, 3.0, сохраняется в двух байтах: 3 в первом байте и 0 — во втором.
- **Случайное число клиента.** Это 32-байтовое поле используется клиентом, чтобы передать случайное число, которое создает параметры безопасности.
- **Длина ID сеанса.** Это 1-байтовое поле определяет длину ID сеанса (следующее поле). Если нет ID сеанса, значение этого поля — 0.
- **ID сеанса.** Значение этого поля переменной длины — 0, когда клиент начинает новый сеанс. ID сеанса инициируется сервером. Однако если клиент хочет возобновить предварительно остановленный сеанс, он может включить предварительно определенный ID сеанса в этом поле. Протокол определяет для ID сеанса максимум 32 байта.
- **Длина набора шифров.** Это 2-байтовое поле определяет длину предложенного клиентом списка набора шифров (следующее поле).
- **Список набора шифров.** Это поле переменной длины дает список набора шифров, поддерживаемых клиентом. Поле перечисляет набор шифров от наиболее предпочтительных к наименее предпочтительным. Каждый набор шифров кодируется как двухбайтовое число.
- **Длина методов сжатия.** Это 1-байтовое поле определяет длину списка предложенных клиентом методов сжатия (следующее поле).
- **Список методов сжатия.** Это поле переменной длины дает список методов сжатия, которые поддерживает клиент. Поле перечисляет методы от наиболее предпочтительных к наименее предпочтительным. Каждый метод кодируется как однобайтовое число. Сейчас единственный метод — метод

Рис. 17.29. Сообщение ServerHello

NULL («нет сжатия»). В этом случае значение длины методов сжатия — 1, и список метода сжатия имеет только один элемент со значением 0.

Сообщение ServerHello

Сообщение ServerHello — ответ сервера на сообщение ClientHello. Формат подобен сообщению ClientHello, но с меньшим количеством полей. Рисунок 17.29 показывает формат сообщения.

Поле версии — то же самое, что и в сообщении ClientHello. Поле случайного числа сервера определяет значение, выбранное сервером. Длина ID сеанса и поле ID

сеанса — те же самые, что и в сообщении ClientHello. Однако ID сеанса — обычно пробел (и длина обычно устанавливается на 0), если сервер не возобновляет старый сеанс. Другими словами, если сервер позволяет возобновление сеанса, он вставляет соответствующее значение в поле ID сеанса, которое используется клиентом (в сообщении ClientHello), когда клиент желает повторно открыть старый сеанс.

Поле выбранного набора шифров определяет единственный набор шифров, который выбран сервером из списка, посланного клиентом. Поле методов сжатия определяет выбранный сервером метод из списка, посланного клиентом.

Сообщение Certificate

Сообщение Certificate может быть передано клиентом или сервером, чтобы передать список сертификатов открытого ключа. Рисунок 17.30 показывает формат.

Рис. 17.30. Сообщение Certificate

Значение поля «тип» — 11. Текстовый блок сообщения включает следующие поля.

- **Длина цепочки сертификатов.** Это трехбайтовое поле показывает длину цепочки сертификатов. Это поле избыточно, потому что его значение — всегда на 3 меньше, чем значение поля длины.
- **Цепочка сертификатов.** Это поле переменной длины содержит цепочку сертификатов открытого ключа, который использует клиент или сервер. Для каждого сертификата есть два дополнительных поля:
  а. трехбайтовое поле длины;
  б. сам сертификат — поле переменного размера.

Сообщение ServerKeyExchange

ServerKeyExchange-сообщение передают от сервера к клиенту. Рисунок 17.31 показывает общий формат.

Сообщение содержит ключи, сгенерированные сервером. Формат сообщения зависит от набора шифров, выбранного в предыдущем сообщении. Клиент, который получает сообщение, должен интерпретировать его согласно предыдущей информации. Если сервер передал сообщение сертификата, то сообщение содержит подписанный параметр.

Сообщение CertificateRequest

CertificateRequest-сообщение передают от сервера к клиенту. Сообщение просит, чтобы клиент подтвердил серверу свою подлинность, а также подлинность одного из используемых сертификатов и одной из удостоверяющих администраций, названной в сообщении. Рисунок 17.32 показывает общий формат.

Рис. 17.31. Сообщение ServerKeyExchange

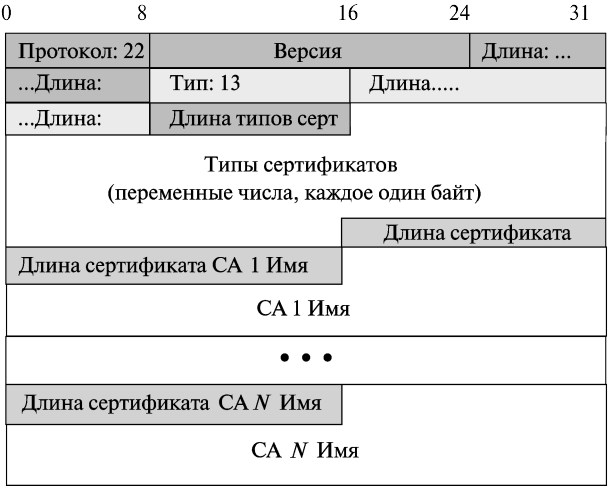


Рис. 17.32. Сообщение CertificateRequest

Значение поля типа — 13. Текстовый блок сообщения включает следующие поля.

- **Длина типов сертификатов (Len of Cert Types).** Это однобайтовое поле показывает длину типов сертификатов.
- **Типы сертификатов (Certificate Types).** Это поле переменной длины содержит список сертификата открытого ключа, который принят сервером. Каждый тип — один байт.

- **Длина сертификатов подтверждения подлинности (Length of CAs).** Это двухбайтовое поле показывает длину списка сертификатов, удостоверяющих подлинность (остальная часть пакета).
- **Длина сертификата подлинности *x*. Имя. (Length of CA *x*. Name).** Это двухбайтовое поле определяет длину *x*-ого сертификата и название администрации. *x* может принимать значение от 1 до *N*.
- **Сертификат *x*, подтверждающий подлинность. Имя. (CA *x*. Name).** Это поле переменной длины определяет название *x*-ого сертификата администрации. x может принимать значение от 1 до *N*.

Сообщение ServerHelloDone

ServerHelloDone-сообщение — последнее сообщение, посылаемое во второй фазе процедуры установления связи. Сообщение сигнализирует, что Фаза II не доставляет никакой дополнительной информации. Рисунок 17.33 показывает формат.

Рис. 17.33. Сообщение ServerHelloDone

Сообщение CerivicateVerify

Сообщение CerivicateVerify — это последнее сообщение Фазы III. В этом сообщении клиент доказывает, что фактически имеет секретный ключ, связанный с его сертификатом открытого ключа. Чтобы доказать это, клиент создает хэш всех сообщений установления соединения, посланных перед этим сообщением, и под-

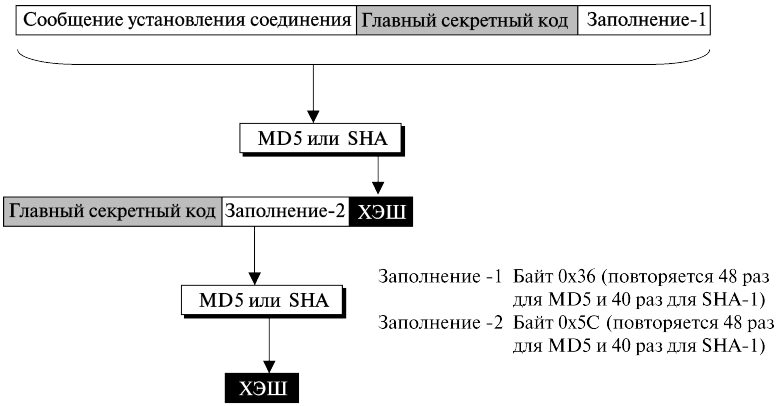


Рис. 17.36. Сообщение ClientKeyExchange

писывает их, используя алгоритм MD5 или SHA-1, основанный на типе сертификата клиента. Рисунок 17.34 показывает формат.

Рис. 17.34. Сообщение CerivicateVerify

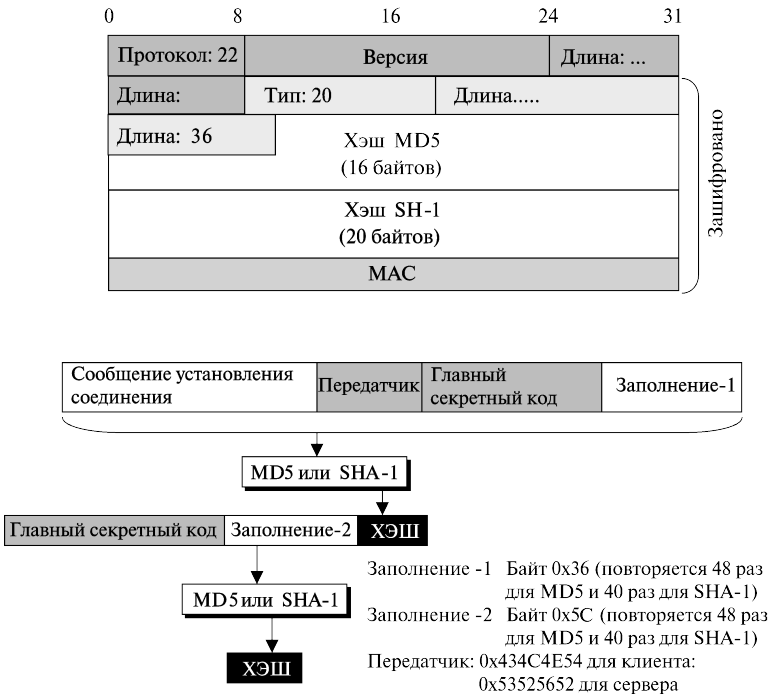
Если секретный ключ клиента связан с DSS-сертификатом, то хэширование базируется только на алгоритме SHA-1, и длина хэш — 20 байтов. Если секретный ключ клиента связан с сертификатом RSA, то есть два (конкатенированных) хэша: один — основанный на MD5 и другой — основанный на SHA-1. Полная длина — 16 + 20 = 36 байтов. Рисунок 17.35 показывает вычисления хэша.

Сообщение ClientKeyExchange

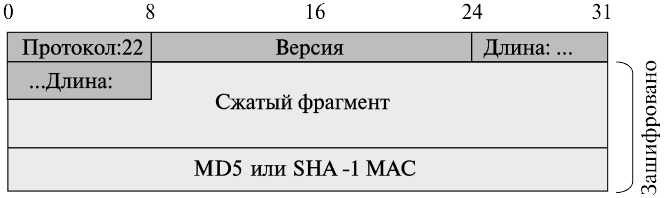
ClientKeyExchange — второе сообщение, посылаемое в течение третьей фазы процедуры установления связи. В этом сообщении клиент обеспечивает ключи. Формат сообщения зависит от заданных алгоритмов смены ключей, выбранных двумя сторонами. Рисунок 17.36 показывает общую идею формата.

Рис. 17.35. Вычисление хэш для сообщения CerivicateVerify

Сообщение Finished



Сообщение Finished показывает, что переговоры закончены. Оно содержит все сообщения, которыми обменивались в течение процедуры установления связи, сопровождаемой указанием типа передатчика (клиент или сервер) и дополнением главного секретного кода. Точный формат зависит от типа используемого



набора шифра. Общий формат показан на рис. 17.37 — последовательное соединение (конкатенация) двух хэш. На рисунке 17.38 изображено, как вычисляется каждый из них. Обратите внимание, что когда клиент или сервер передают сообщение Finished, они уже передали сообщение ChangedCipherSpec. Другими словами, криптографическая секретная информация (писать) находится в активном состоянии, клиент или сервер могут ее обработать. Сообщение Finished подобно фрагменту данных, прибывающему от прикладного уровня. Сообщение Finished может быть заверено (используя MAC из набора шифров) и зашифровано (используя алгоритм шифрования набора шифров).

Прикладные данные

Протокол передачи записей добавляет подпись в конце фрагмента (возможно, сжатый MAC), прибывающий от прикладного уровня, и затем зашифровывает фрагмент и MAC.

Рис. 17.37. Сообщение Finished

Рис. 17.38. Вычисление хеш для сообщения Finished

После добавления общего заголовка со значением протокола 23 сообщение передачи записи передано. Обратите внимание, что общий заголовок не зашифрован. Рисунок 17.39 показывает общий формат.

Рис. 17.39. Сообщение протокола передачи записей для прикладных данных

17.4. Безопасность транспортного уровня

Безопасность транспортного уровня (TLS — Transport Layer Security) — протокол IEFT, стандартная версия протокола SSL. Эти два протокола очень похожи, но имеют небольшие отличия. Вместо того чтобы описывать TLS полностью, в этой секции мы только отметим отличия между протоколами TLS и SSL.

Версии

Первое отличие — номер версии (основное, но незначительное). Текущая версия SSL — 3.0; текущая версия TLS — 1.0. Другими словами, SSLv3.0 совместим с TLSv 1.0.

Набор шифров

Другое незначительное отличие между SSL и TLS — отсутствие поддержки Fortezza. TLS не поддерживает Fortezza для смены ключей или для шифрования/дешифрования. Таблица 17.6 показывает набор шифров для TLS.

Генерация криптографической секретности

Генерация криптографической секретности в TLS более сложная, чем в SSL. TLS сначала определяет две функции: функцию расширения данных и псевдослучайную функцию. Рассмотрим их.

Функция расширения данных

Функция расширения данных использует заранее заданный код аутентификации на основе хэширования (HMAC-HASH-BASED MESSAGE AUTHENTICATION CODE), или MD5, или SHA-1 для того, чтобы расширить информацию засекречивания. Эту функцию можно рассматривать как функцию, содержащую множество секций, где каждая секция создает одно значение хэширования. Расширенная секретность — последовательное соединение значений хэширования. Каждая секция использует два HMAC, информацию засекречивания и начальное число. Функция расширения данных — это формирование цепочки в виде многих секций. Однако чтобы сделать следующую секцию зависимой от предыдущей, второе начальное число — фактически выход первого HMAC предыдущей секции, как это показано на рис. 17.40.

Псевдослучайная функция

TLS определяет **псевдослучайную функцию (PRF — PseudoRandom Function)**, чтобы получить комбинацию двух функций расширения данных: одна из них использует MD5 и другая — SHA-1. На **PRF** поступает три части информации: секретный код, метка и начальное число.

Таблица 17.6. Набор шифров для TLS

Набор шифров	Замена ключей	Шифрование	Хэш
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
TLS_RSA_W1TH_RC4_128_MD5	RSA	RC4	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-!

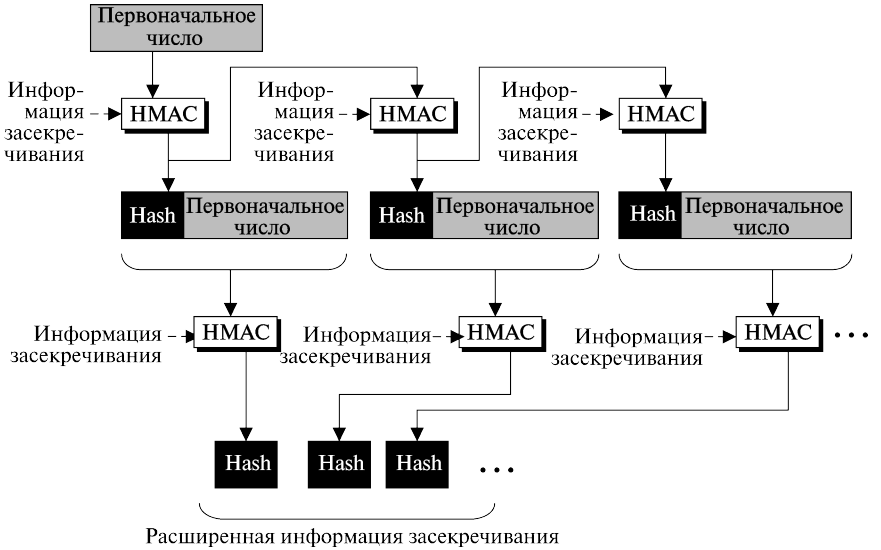
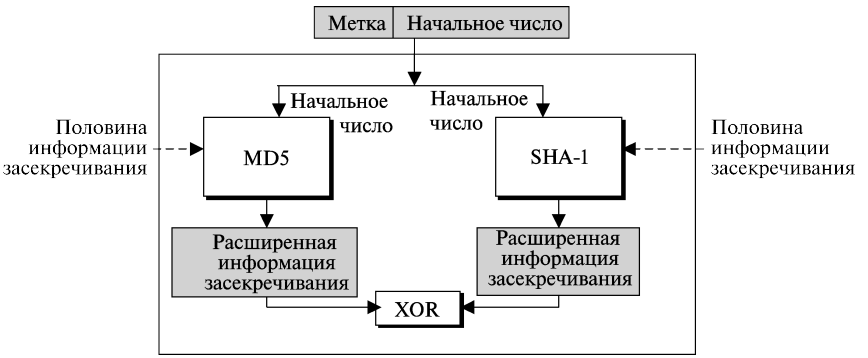


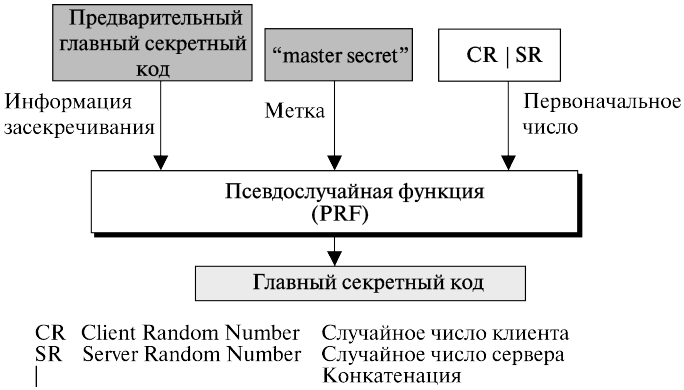
Рис. 17.40 Функция расширения данных

TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES	SHA-1
TLS_DH_anon_WITH-RC4_128_MD5	DH_anon	RC4	MD5
TLS_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-1



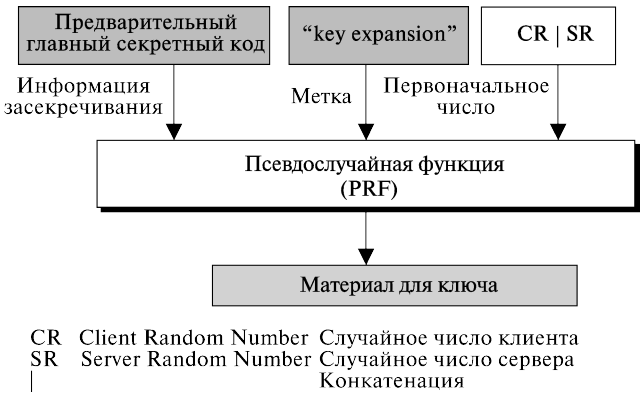
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES	SHA-1
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-1

TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-1
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-1
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1



TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-1

Метка и начальное число связаны и служат начальным числом для каждой функции расширения данных. Информация засекречивания разделена на две части; каждая часть используется как информация засекречивания для каждой функции расширения данных. Выходы двух функций расширения данных складывают по модулю два, чтобы создать конечную расширенную информацию засекречивания. Обратите внимание, что поскольку хэш создается MD5 и SHA-1, он имеет различные размеры, поэтому должны быть созданы дополнительные сек-



ции функций на базе MD5, чтобы сделать два вывода с одинаковым размером. Рисунок 17.41 показывает идею применения PRF.

Рис. 17.41. PRF

Главный секретный код

TLS применяет функцию PRF, чтобы создать главный секретный код из предварительного главного секретного кода. Это можно сделать, используя предварительный главный секретный код как информацию засекречивания, строку «главный секретный код» — как метку и последовательное соединение информации (конкатенацию) случайного числа клиента и случайное число сервера — как начальное число. Обратите внимание, что метка — фактически код ASCII строки «главного секретного кода». Другими словами, метка определяет выход для создания главного секретного кода. Рисунок 17.42 иллюстрирует идею.

Рис. 17.42. Генерация главного секретного кода

Материал для ключей

TLS использует функцию PRF, чтобы создать материал для ключей от главного секретного кода. На сей раз информация засекречивания содержит: главный секретный код; метку — это строка «расширение ключа»; и начальное число — конкатенацию случайного числа сервера и случайного числа клиента, как это показано на рис. 17.43.

Рис. 17.43. Генерация материала для ключа

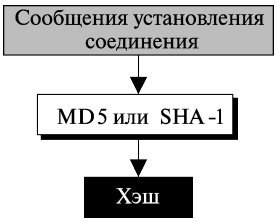
Аварийный протокол

TLS поддерживает все аварийные сигналы, определенные в SSL, за исключением NoCertificate. TLS также добавляет к списку SSL некоторые новые. Таблица 17.7 показывает полный список аварийных сигналов, поддерживаемых TLS.

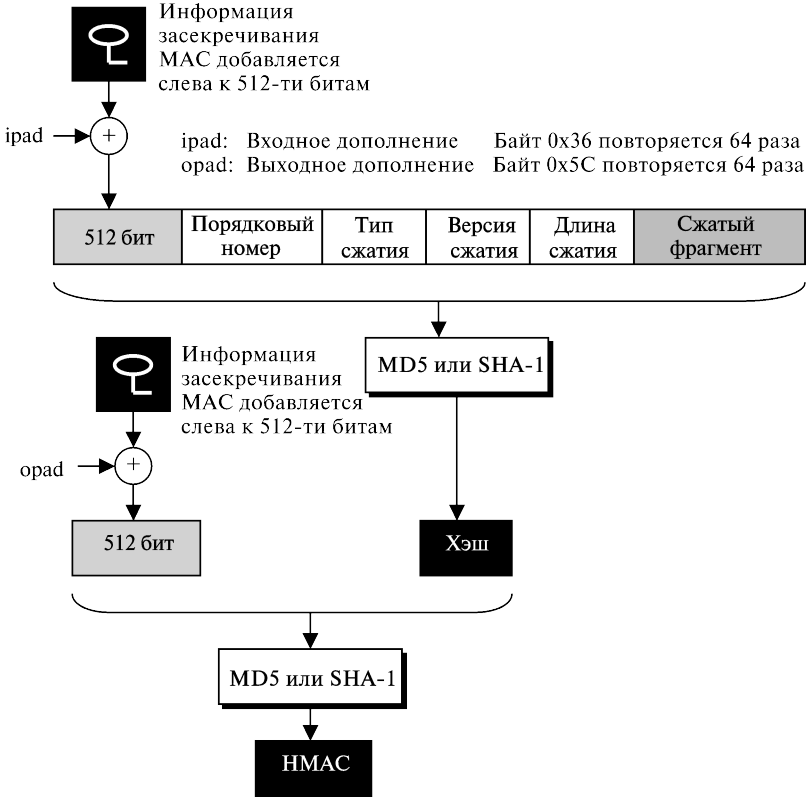
Таблица 17.7. Аварийные сигналы, определенные для TLS

Значение	Описание	Содержание
0	*CloseNotify*	Передатчик не будет посылать сообщений
10	*UnexpectedMessage*	Получено несоответствующее сообщение
20	*BadRecordMAC*	Получен некорректный MAC
21	*DecryptionFailed*	Дешифрованное сообщение недействительно
22	*RecordOverFlow*	Размер сообщения больше чем $2^{14} + 2048$
30	*DecampressionFailure*	Невозможно соответствующее расширение сообщения
40	*HandsHakeFailure*	Передатчик не может завершить установление соединения
42	*BadCertificate*	Полученный сертификат искажен

43	*UnsupportedCertificate*	Полученный тип сертификата не поддерживается
44	*CertificateRevoked*	Подписавший аннулировал сертификат
45	*CertificateExpired*	Срок сертификата истек
46	*CertificateUnknown*	Сертификат неизвестен



47	*IllegalParameter*	Поле выходит за допустимые пределы или не совместимо с другими
48	*UnknownCA*	CA не может быть идентифицировано
49	*AcessDenied*	Нежелательно для продолжения переговоров
50	*Decode Error*	Полученное сообщение не может быть декодировано
51	*DecryptError*	Расшифровка зашифрованного текста недействительна
60	*ExportRestriction*	Проблемы согласования с ограничениями в США
70	*ProtocolVersion*	Эта версия протокола не поддерживается
71	*InsufficientSecurity*	Требуется больший набор секретных шифров
80	*InternalError*	Местная ошибка
90	*UserCanceled*	Данная сторона хочет прекратить переговоры



100	*NoRenegotiation*	Сервер не может снова начать переговоры по установлению соединения

Протокол установления соединения

TLS вносит некоторые изменения в протокол установления соединения. Были специально изменены детали сообщения CertificateVerify и сообщения Finished.

Сообщение CertificateVerify (сертификат верифицирован)

В SSL хэширование, используемое в Сообщении CertificateVerify, — это хэширование с двумя шагами, а именно: сообщение установления соединения плюс заполнение и главный секретный код. TLS упростил процесс: в нем хэшируется только сообщение установления соединения, как показано на рис. 17.44.

Рис. 17.44. Хэш для сообщения CertificateVerify в TLS

Сообщение Finished

Было также изменено вычисление хэш для сообщения Finished. TLS использует PRF, чтобы вычислить два хэша, применяемых для сообщения Finished, как это показано на рис. 17.45.

Протокол передачи записей

Единственное изменение в протоколе передачи записей — использование HMAC, чтобы подписать сообщение. TLS применяет MAC, как это определено в лекции 11, для того чтобы создать HMAC. TLS также добавляет версию протокола (названную сжатой версией) к тексту, который будет подписан. Рисунок 17.46 показывает, как формируется HMAC.

Рис. 17.45. Хэш для сообщения Finished в TLS

Рис.17.46 HMAC для TLS

17.4. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце.

Книги

[Res], [Tho00], [Sta06], [Rhe03] и [PHS03] рассматривают SSL и TLS.

Сайты

Нижеследующий сайт дает больше информации о темах, обсужденных в этой лекции.

http://www.ietf.org/rfc/rfc2246.txt

17.7. Итоги

- Протокол безопасности транспортного уровня обеспечивает услуги безопасности из конца в конец для приложений, которые пользуются протоколами транспортного уровня, такими как, например, TCP. На сегодняшний день преобладает применение двух протоколов: протокол «Уровень Безопасных Розеток» (SSL — Secure Sockets Layer) и протокол «Безопасность Транспортного уровня» (TLS — Transport Layer Security).
- SSL или TLS обеспечивают такие услуги, как фрагментация, сжатие, целостность сообщения, конфиденциальность и создание кадра данных, полученных от прикладного уровня. Как правило, SSL (или TLS) может получить прикладные данные от любого протокола прикладного уровня, но работает протокол обычно с HTTP.
- Комбинация алгоритмов смены ключей, хэширования и алгоритм шифрования определяют набор шифров для каждого сеанса.

- Для того чтобы обмениваться заверенными и конфиденциальными сообщениями, клиенту и серверу необходимо иметь шесть единиц криптографической секретности (четыре ключа и два вектора инициализации).
- В SSL (или TLS) отличают подключение и сеанс. В сеансе одна сторона играет роль клиента, а другая — роль сервера; при подключении обе стороны играют одинаковые роли.
- SSL (или TLS) определяет четыре протокола на двух уровнях: протокол установления соединения, протокол ChangeCipherSpec, аварийный протокол и протокол передачи записей. Протокол установления соединения использует несколько сообщений, чтобы договориться о наборе шифров, подтвердить подлинность сервера для клиента и клиента для сервера, если это необходимо, и обмениваться информацией для организации криптографической секретности. Протокол ChangeCipherSpec определяет процесс перемещения информации между состоянием ожидания и активным состоянием. Аварийный протокол передает извещения об ошибках и ситуациях, отклоняющихся от нормальных. Протокол передачи записей доставляет сообщения от верхнего уровня (протокол установления соединения, аварийный протокол, ChangeCipherSpec-протокол) или прикладного уровня.

17.8. Набор для практики

Обзорные вопросы

1. Перечислите услуги, обеспеченные SSL или TLS.
2. Объясните, как в SSL создается из предварительного главного секретного кода главный секретный код.
3. Объясните, как в TLS создается из предварительного главного секретного кода главный секретный код.
4. Объясните, как в SSL создается из главного секретного кода материал для ключей.
5. Объясните, как в TLS создается из главного секретного кода материал для ключей.
6. Покажите различия между сеансом и соединением.
7. Перечислите цель четырех протоколов, определенных в SSL или TLS.
8. Определите цель каждой фазы в протоколе установления соединения.
9. Сравните и противопоставьте протоколы установления соединения в SSL и TLS.
10. Сравните и противопоставьте протоколы передачи записей в SSL и TLS.

Упражнения

1. Какова длина материала для ключей, если набор шифров — один из перечисленных ниже:
   a. SSL_RSA_WITH_NULL_MD5
   b. SSL_RSA_WITH_NULL_SHA

c. TLS_RSA_WITH_DES_CBC_SHA
d. TLS_RSA_WITH_3DES_EDE_CBC_SHA
e. TLS_DHE_RSA_WITH_DES_CBC_SHA
f. TLS_DH _RSA _3DES _EDE_CBC_SHA

2. Покажите число повторных модулей, необходимых для каждого случая в Упражнении 1 (см. рис. 17.9).
3. Сравните вычисление главного секретного кода в SSL с таким же процессом в TLS. В SSL предварительный главный код применяется в вычислении три раза, в TLS — только единожды. Какое вычисление более эффективно по объему и по времени?
4. Сравните вычисление материала для ключей в SSL и TLS. Ответьте на следующие вопросы:

а. Какое вычисление обеспечивает большую безопасность?
б. Какое вычисление более эффективно по объему и времени?

5. Вычисление материала для ключей в SSL требует нескольких итераций, в TLS этого не делается. Как может TLS вычислять материал для ключей переменной длины?
6. Когда сеанс продолжается с новым соединением, SSL не требует проведения полной процедуры установления связи. Покажите сообщения, которыми необходимо будет обменяться в частичной процедуре установления связи.
7. Когда сеанс продолжен, какая из следующей криптографической информации для засекречивания должна быть повторно вычислена?
   а. предварительный главный секретный код
   б. главный секретный код
   в. ключи подтверждения подлинности
   г. ключи шифрования
   д. IV (первоначальный вектор)
8. Что случится в процессе на рис. 17.20, если сервер передает сообщение ChangeCipherSpec, а клиент не передает? Какие сообщения могут быть переданы в протоколе установления соединения? Какие не могут быть переданы?
9. Сравните вычисление MAC в SSL и TLS (см. рис. 17.22 и рис. 17.46). Какое из них более эффективно?
10. Сравните вычисление хэширования для сообщения CertificateVerify в SSL и TLS (см. рис. 17.35 и рис. 17.44). Какое более эффективно?
11. Сравните вычисление хэширования для сообщения Finished в SSL и TLS (см. рис. 17.38 и рис. 17.45). Ответьте на следующие вопросы:
    а. Которое из них более безопасно?
    б. Которое из них более эффективно?
12. TLS использует PRF для всех вычислений хэш кроме сообщения CertificateVerify. Объяснить это исключение.
13. Большинство протоколов записывают в виде формулы, чтобы показать вычисления криптографической секретности и хэширования. Например, в SSL вычисление главного секретного кода (см. рис. 17.8) отображается следующим образом (последовательная конкатенация записывается в виде линеек):

Master Secret = MD5 (pre-master | SHA-1 («A» |pre-master I CR I SR)) |
MD5 (pre-master| SHA-1 («A» | pre-master | CR I SR)) |
MD5 (pre-master |SHA-1 («A» | pre-master | CR | SR))

Master Secret	Главный секретный код
pre-master	Предварительный главный секретный код
|	Конкатенация
CR	Случайное число клиента
SR	Случайное число сервера

Запишите в виде формул следующие процессы:

а. Материал для ключей в SSL (рис. 17.9)
б. MAC в SSL (рис. 17.22)
в. Вычисление хэширования для сообщения CertificateVerify в SSL (рис. 17.35)
г. Вычисление хэширования для сообщения Finished в SSL (рис. 17.38)
д. Расширение данных в TLS (рис. 17.40)
ж. PRF в TLS (рис. 17.41)
з. Главный секретный код в TLS (рис. 17.42)
и. Материал для ключей в TLS (рис. 17.43)
к. Вычисление хэширования для сообщения CertificateVerify в TLS (рис. 17.44)
л. Вычисление хэширования для Finished сообщения в TLS (рис. 17.45)
м. MAC в TLS (рис. 17.46)

14. Покажите, как SSL или TLS реагируют на атаку воспроизведения. То есть покажите, как SSL или TLS отвечает нападавшему, который пытается имитировать одно или более сообщений установления соединения (предварительно записав сообщение).
15. Покажите, как SSL или TLS реагируют на атаку грубой силы. Может ли злоумышленник использовать исчерпывающий компьютерный поиск и найти ключ шифрования в SSL или TLS? Какой протокол более безопасен в этом отношении — SSL или TLS?
16. Каков риск использования ключей короткой длины в SSL или TLS? Какую атаку злоумышленник может применить, если ключи коротки?
17. Действительно ли SSL или TLS относительно безопасны к атаке «посредника»? Может ли злоумышленник создать материал для ключей между клиентом и самим собой и между собой и сервером?

Лекция 18. Безопасность на сетевом уровне: IP SEC

Цели и содержание

Эта лекция имеет несколько целей.

- Определить архитектуру IPSec.
- Обсудить приложение IPSec в транспортном и туннельном режимах.
- Обсудить, как IPSec может использоваться, чтобы обеспечить только установление подлинности.
- Обсудить, как IPSec может использоваться, чтобы обеспечить и конфиденциальность, и установление подлинности.
- Определить службы обеспечения безопасности (SA — Security Association) и объяснить, как они реализованы для IPSec.
- Определить протокол обмена ключами (IKE — Internet Key Exchange) и объяснить, как он используется в IPSec.

В двух предыдущих лециях мы обсудили безопасность на прикладном и транспортном уровнях. Однако такая безопасность в некоторых случаях не может быть достаточной.

Во-первых, не все программы «клиент-сервер» защищены на прикладном уровне; например, PGP и S/MIME защищают только электронную почту.

Во-вторых, не все программы «клиент-сервер» на прикладном уровне используют протокол TCP, который может быть защищен SSL или TLS; некоторые программы применяют обслуживание UDP.

В-третьих, много приложений, таких как протоколы маршрутизации, непосредственно работают с протоколом IP; они нуждаются в службах безопасности на уровне IP.

Безопасный IP (IPSec) — совокупность протоколов, разработанных Группой Инженерной Поддержки сети Интернет (IETF Internet Engineering Task Force), чтобы обеспечить безопасность передачи пакетов на сетевом уровне. Сетевой уровень в Интернете упоминается часто как Интернет-протокол — Internet Protocol (IP). Протокол IPSec помогает создавать заверенные и конфиденциальные пакеты для уровня IP, как это показано на рис. 18.1.

Рис. 18.1. Набор протоколов TCP/IP и IPSec

IPSec может быть полезен в нескольких областях. Во-первых, он может увеличить безопасность программ «клиент-сервер», таких как электронная почта, которая использует свои собственные протоколы безопасности. Во-вторых, он может увеличить безопасность программ «клиент-сервер», которые применяют службы безопасности на транспортном уровне, — например, HTTP. Он может обеспечить безопасность программ «клиент-сервер», которые не пользуются службами безопасности транспортного уровня. Он может обеспечить безопасность для программ установления связи «от-узла-к-узлу», таких как маршрутизация.

18.1. Два режима

IPSec работает в двух различных режимах — транспортном и туннельном.

Транспортный режим

В **транспортном режиме** IPSec защищает информацию, доставляемую от транспортного уровня к сетевому уровню. Другими словами, транспортный режим защищает полезную нагрузку сетевого уровня, и полезная нагрузка должна быть инкапсулирована в сетевой уровень, как это показано на рис. 18.2.

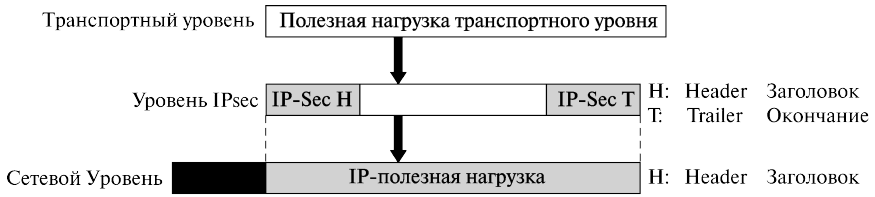


Рис.18.2. Транспортный режим IPSec

Обратите внимание, что транспортный режим не защищает заголовок IP. Другими словами, транспортный режим не защищает весь пакет IP, а только пакет транспортного уровня (полезная нагрузка P-уровня). В этом режиме IPSec-заголовок (и конечная метка) добавляется к информации, прибывающей от транспортного уровня. Заголовок IP добавляется позже.

IPSec в транспортном режиме не защищает заголовок IP, а только информацию, прибывающую от транспортного уровня.

Транспортный режим обычно применяется, когда мы нуждаемся в защите данных на участке «хост-хост» («из конца в конец»). Передающий хост использует IPSec, чтобы подтвердить подлинность и/или зашифровать полезную нагрузку, освобожденную от информации транспортного уровня. Приемный хост использует IPSec, чтобы проверить установление подлинности и/или расшифровать пакет IP и доставить его транспортному уровню. Рисунок 18.3 иллюстрирует эту концепцию.

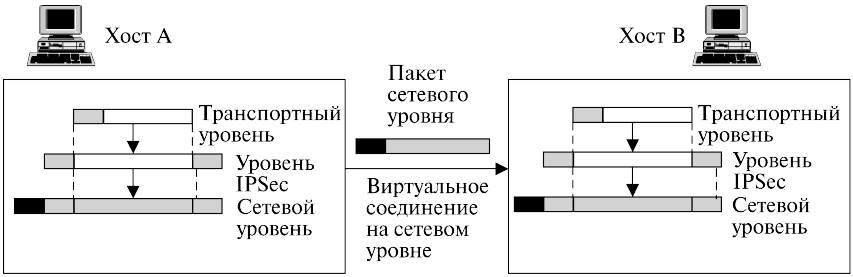


Рис. 18.3. Действия транспортного режима

Туннельный режим

В **туннельном режиме** IPSec защищает весь пакет IP. Он обрабатывает пакет IP (включая заголовок), применяя методы безопасности IPSec к полному пакету, и затем добавляет новый заголовок IP, как это показано на рис. 18.4.

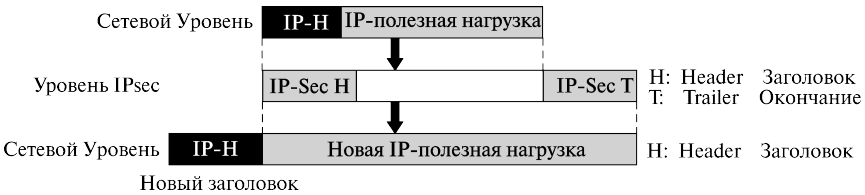


Рис. 18.4. Туннельный режим IPSec

Новый заголовок IP, как мы увидим, содержит иную информацию, нежели первоначальный заголовок IP. Туннельный режим обычно используется между двумя маршрутизаторами, между хостом и маршрутизатором или между маршрутизатором и хостом, как это показано на рис. 18.5. Другими словами, туннельный режим применяется, когда либо передатчик, либо приемник не является хостом.

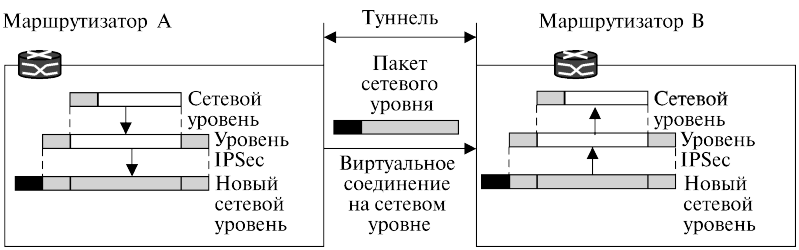


Рис. 18.5. Действия туннельного режима

Весь первоначальный пакет защищен от вмешательства между передатчиком и приемником, как будто весь пакет проходит мнимый туннель.

IPSec в туннельном режиме защищает первоначальный заголовок IP.

Сравнение

В транспортном режиме уровень IPSec располагается между транспортным уровнем и сетевым уровнем. В туннельном режиме поток проходит от сетевого уровня до уровня IPSec, а затем снова возвращается назад к сетевому уровню. Рисунок 18.6 сравнивает эти два режима.

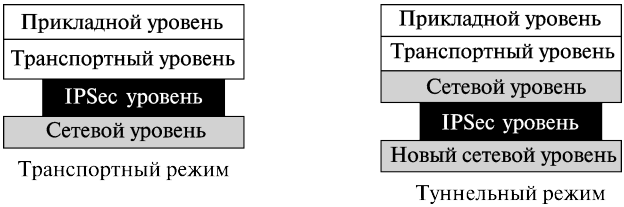


Рис. 18.6. Сравнение транспортного и туннельного режимов

18.2. Два протокола безопасности

IPSec определяет два протокола: протокол «Заголовок аутентификации (AH — Authentication Header)» и протокол «Полезная нагрузка со встроенной защитой (ESP — Encapsulating Security Payload)». Их цель — обеспечить установление подлинности и/или шифрование для пакетов на уровне IP.

Заголовок аутентификации (AH)

Протокол «Заголовок аутентификации» (AH) разработан для того, чтобы подтвердить подлинность хоста источника и гарантировать целостность полезной нагрузки, которую переносит пакет IP. Протокол использует хэш-функцию и симметричный ключ, чтобы создать дайджест сообщения; дайджест вставляется в заголовок аутентификации.

Затем AH вставляют на соответствующее место, в зависимости от режима (транспортный или туннельный). Рисунок 18.7 показывает поля и позиции заголовка аутентификации в транспортном режиме.

Когда дейтаграмма IP переносит заголовок аутентификации, первоначальное значение в поле протокола заголовка IP устанавливается в значение 51. Поле в заголовке аутентификации (следующее поле заголовка) содержит первоначальное значение поля протокола (тип полезной нагрузки, которую несет дейтаграмма IP). Добавление заголовка аутентификации проводится следующими шагами.

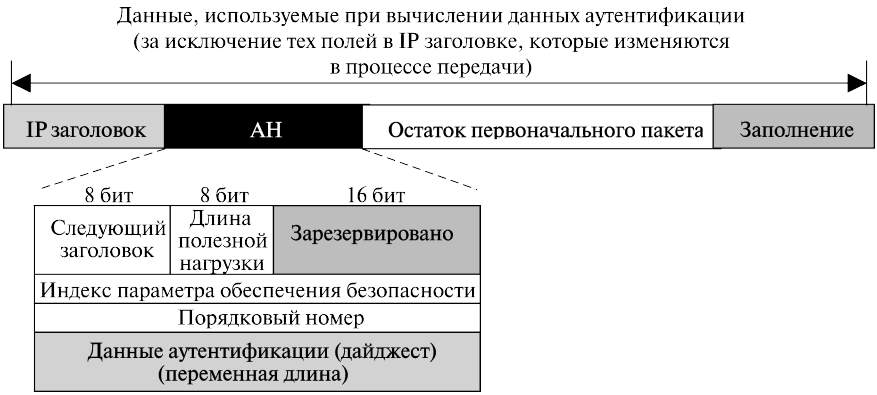


Рис. 18.7. Протокол «Заголовок аутентификации (AH)»

1 Заголовок аутентификации добавляется к полезной нагрузке с полем аутентификации данных, установленным на 0.
2. Заполнение добавляется, если нужно сделать полную длину сообщения для конкретного алгоритма хэширования.
3. Хэширование проводится на всем пакете. Однако в вычисление дайджеста сообщения (данные аутентификации) включены только те поля IP-заголовка, которые не изменяются в течение передачи.
4. Данные аутентификации вставляются в заголовок аутентификации.
5. Заголовок IP добавляется после изменения значения поля протокола на 51.

Краткое описание каждого поля дано ниже.

- **Следующий заголовок.** Поле «следующий заголовок» имеет 8 битов и определяет тип полезной нагрузки, которую несет IP-дейтаграмма (такие как TCP, UDP, ICMP или OSPF). Поле выполняет ту же самую функцию, что и поле протокола в заголовке IP перед инкапсуляцией. Другими словами, процесс копирует значение поля протокола в дейтаграмме IP в поле «следующий заголовок». Значение поля протокола в новой дейтаграмме IP теперь установлено на 51, чтобы показать, что пакет переносит заголовок аутентификации.
- **Длина полезной нагрузки.** Название этого поля из 8 битов вводит в заблуждение. Оно не определяет длину полезной нагрузки; оно задает длину заголовка аутентификации в числах, кратных 4-м байтам, но не включает первые 8 байтов.
- **Индекс параметра обеспечения безопасности.** Это поле на 32 бита (SPI — Security Parameter Index) играет роль идентификатора виртуального канала и для всех пакетов, посылаемых в течение соединения и называемых Службы обеспечения безопасности трафика (Security Association). Они будут рассмотрены позже.
- **Порядковый номер.** Порядковый номер на 32 бита обеспечивает информацию о порядке последовательности дейтаграмм. Порядковые номера предотвращают повторение. Обратите внимание, что номер не повторяется

при повторной передаче пакета. Порядковый номер не циклический и не повторяет цикла после того, как он достигает 2^{32}. Для обновления номера должно быть установлено новое соединение.

- **Данные аутентификации.** Наконец, поле данных аутентификации — результат применения хэш-функции ко всей IP-дейтаграмме, исключая поля, которые меняются в течение транзита (например, поле «время жизни»).

Протокол AH обеспечивает установление подлинности источника и целостность данных, но не конфиденциальность.

Полезная нагрузка со встроенной защитой (ESP)

Протокол AH не обеспечивает секретность, а только установление подлинности источника и целостность данных. Для IPSec был определен альтернативный протокол **Полезная нагрузка со встроенной защитой** (ESP — Encapsulating Security Payload), который гарантирует установление подлинности источника, целостность и секретность. ESP добавляет заголовок и конечную метку. Обратите внимание, что данные аутентификации ESP добавляются в конце пакета — это делает их вычисление более простыми. Рисунок 18.8 показывает размещение заголовка ESP и конечной метки.

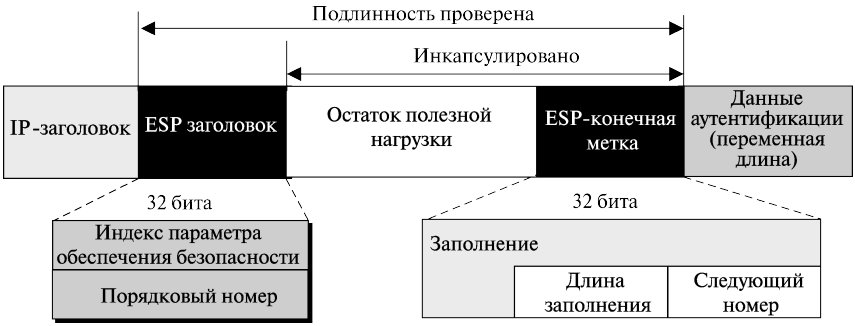


Рис. 18.8. Протокол «Полезная нагрузка со встроенной защитой» (ESP)

Когда дейтаграмма IP переносит заголовок ESP и конечную метку, значение поля протокола в заголовке IP равно 50. Поле в конечной метке ESP (поле следующего заголовка) содержит первоначальное значение поля протокола (тип полезной нагрузки, которую несет дейтаграмма IP, такой как TCP или UDP). Процедура ESP выполняется следующими шагами.

1. Конечная метка ESP добавляется к полезной нагрузке.
2. Полезная нагрузка и конечная метка зашифровываются.
3. Добавляется заголовок ESP.
4. Заголовок ESP, полезная нагрузка и конечная метка ESP используются, чтобы создать данные аутентификации.

5. Данные аутентификации добавляются в конце конечной метки ESP.
6. Заголовок IP добавляется после изменения значения протокола на 50.

Поля заголовка и конечной метки следующие.

- **Индекс параметра обеспечения безопасности.** Поле индекса параметра обеспечения безопасности на 32 бита совпадет с тем, которое определено для протокола AH.
- **Порядковый номер.** Поле порядкового номера на 32 бита совпадет с тем, которое определено для протокола AH.
- **Заполнение** — это поле переменной длины (от 0 до 255 байтов), состоящее из нулей и служащее заполнением.
- **Длина заполнения.** Поле длины заполнения на 8 битов определяет число байтов заполнения между 0 и 255; максимальное значение используется редко.
- **Следующий заголовок.** Поле следующего заголовка на 8 битов совпадет с тем, которое определено для протокола AH. Оно выполняет ту же самую задачу, как и поле протокола в заголовке IP перед инкапсуляцией.
- **Данные аутентификации.** Наконец, поле данных аутентификации — результат применения схем аутентификации к частям дейтаграммы. Обратите внимание на отличие между данными аутентификации в AH и ESP. В AH часть заголовка IP включена в вычисление данных аутентификации, а в ESP — нет.

ESP обеспечивает установление подлинности источника, целостность данных и секретность.

IPv4 и IPv6

IPSec поддерживает и IPv4, и IPv6. В IPv6, однако, AH и ESP — часть расширения заголовка.

Сравнение AH и ESP

Протокол ESP был разработан, когда протокол AH был уже в использовании. ESP умеет то, что AH делает только с дополнительными функциональными возможностями (секретность). Вопрос: почему же тогда мы по-прежнему нуждаемся в AH? Определенного ответа нет. Однако реализация AH включена в несколько коммерческих продуктов, то есть AH останется частью Интернет, пока эти продукты не будут постепенно выведены из употребления.

Услуги, обеспечиваемые IPSec

Эти два протокола, AH и ESP, могут обеспечить несколько услуг безопасности для пакетов на сетевом уровне. Таблица 18.1 показывает список услуг, доступных для каждого из этих протоколов.

Таблица 18.1. Услуги IPSec

Услуги	**AH**	**ESP**
Управление доступом Access control	ДА	ДА
Установление подлинности сообщения (целостность сообщения) Message authentication (message integrity)	ДА	ДА
Установление подлинности объекта (установление подлинности источника данных) Entity authentication (data source authentication)	ДА	ДА
Конфиденциальность Confidentiality	**НЕТ**	ДА
Защита от атаки воспроизведения Replay attack protection	ДА	ДА

Управление доступом

IPSec обеспечивает управление доступом, косвенно использующее базу данных услуг обеспечения безопасности трафика (SAD — Security Association Database), как мы это увидим в следующей секции. Когда пакет достигает пункта назначения и атрибуты службы обеспечения безопасности трафика, установленные для этого пакета, отсутствуют, пакет бракуется.

Целостность сообщения

Целостность сообщения сохраняется и в AH, и в ESP. Дайджест данных создается и посылается передатчиком, который будет проверен приемником.

Установление подлинности объекта

Службы обеспечения безопасности трафика и дайджест ключевого хэширования данных, посланных передатчиком, подтверждают подлинность передатчика данных и в AH, и в ESP.

Конфиденциальность

Шифрование сообщения обеспечивает конфиденциальность в ESP. AH однако, конфиденциальность не гарантирует. Если конфиденциальность необходима, нужно использовать ESP вместо AH.

Защита от атаки воспроизведения

В обоих протоколах предотвращается атака воспроизведения за счет применения порядковых номеров и скользящего окна приемника. Каждый IPSec-заголовок содержит уникальный порядковый номер — когда установлены Службы обеспечения безопасности трафика. Числа начинаются от 0 и увеличиваются, пока не достигают значения $2^{32} - 1$ (размер поля порядкового номера — 32 бита). Когда порядковый номер достигает максимума, он сбрасывается в 0, и в то же самое время удаляются старые Службы обеспечения безопасности трафика (см. следующую секцию) и устанавливаются новые. Чтобы предотвращать пакеты дубликата обработки, IPSec использует фиксированный размер окна приемника. Размер окна приемника задан по умолчанию значением 64. Рисунок 18.9 показывает окно ответа. Окно имеет фиксированный

размер W. Затемненные пакеты показывают, что полученные пакеты были проверены и аутентифицированы.

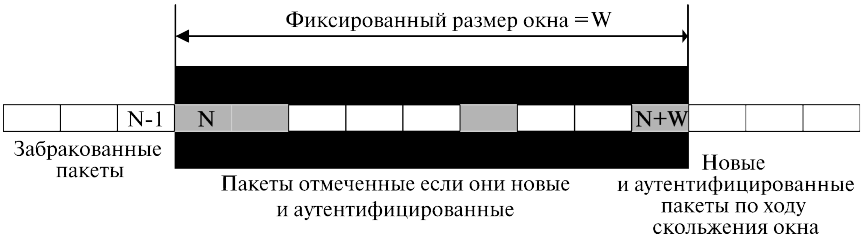


Рис. 18.9 Окно ответа

Когда пакет достигает приемника, в зависимости от значения порядкового номера может произойти одно из трех событий:.

1. Порядковый номер пакета — меньше чем *N*. Тогда пакет размещается налево от окна и будет забракован. Он либо является дубликатом, либо время его прибытия истекло.
2. Порядковый номер пакета — между *N* и *(N* + W – 1) включительно. Тогда пакет размещается в окне. В этом случае, если пакет новый (неотмеченный) и он содержит аутентификационный тест, порядковый номер отмечается, и пакет принимается. Иначе (если пакет не новый) он бракуется.
3. Порядковый номер пакета больше, чем *(N* + W – 1). Тогда пакет размещается справа от окна. В этом случае, если пакет аутентифицирован, соответствующий порядковый номер отмечается и окно перемещается (скользит) вправо и занимает отмеченный порядковый номер. Иначе (если не аутентифицирован) пакет бракуется. Обратите внимание, что это может случиться, если пакет прибывает с порядковым номером, намного большим, чем *(N* + W) (очень далеким от правого края окна). В этом случае скольжение вправо может привести к попаданию немаркированных порядковых номеров налево от окна. Эти пакеты, когда они прибывают, никогда не будут приниматься; их время истекло. Например (на рис. 18.9), если пакет прибывает с порядковым номером *(N* + W + 3), окно перемещается и левый край начнется с *(N* + 3). Это означает, что порядковый номер (N + 2) теперь вне окна. Если пакет прибывает с этим порядковым номером, он будет забракован.

18.3. Услуги обеспечения безопасности трафика

Услуги обеспечения безопасности трафика — очень важный аспект IPSec. IPSec требует между двумя хостами логических отношений, называемых **услуги обеспечения безопасности трафика** (SA — Security Association). В этом разделе вначале рассмотрим идею, а затем покажем, как она используется в IPSec.

Идея услуг обеспечения безопасности трафика

Услуга обеспечения безопасности трафика (SA — Security Association) — соглашение между двумя сторонами для создания безопасного канала между ними. Предположим, что Алиса должна однонаправленно связаться с Бобом. Если Алиса и Боб интересуются только аспектом конфиденциальности и безопасности, они могут получить общедоступный ключ засекречивания для связи между собой. Мы можем сказать, что эздесь задействованы две услуги обеспечения безопасности трафика (SA's) между Алисой и Бобом: одна SA — исходящая и одна SA — входящая. Каждый из них хранит значение ключа и имя алгоритма шифрования/дешифрования. Алиса использует алгоритм и ключ, чтобы зашифровать сообщение Бобу; Боб использует алгоритм и ключ, когда он должен расшифровать сообщение, полученное от Алисы. Рисунок 18.10 показывает эти простые услуги SA.

Услуги обеспечения безопасности трафика могут быть расширены, если наши две стороны нуждаются в гарантии целостности сообщения и установлении подлинности. Тогда каждое такое сообщество нуждается в дополнительных данных, например, таких как алгоритм для целостности сообщения, ключ и другие параметры. Все может быть намного сложнее, если стороны использовали различные протоколы, которые имеют разные алгоритмы и параметры — например, IPSec AH или IPSec ESP.

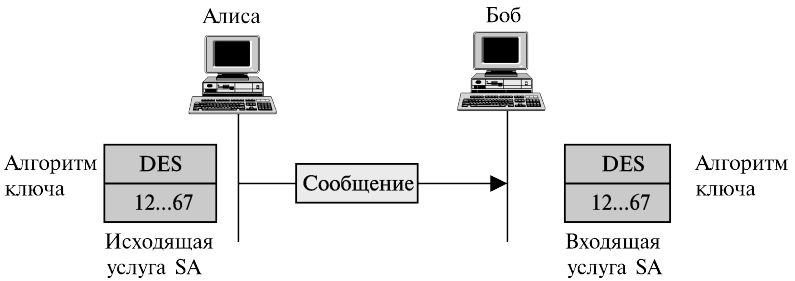


Рис. 18.10. Простые услуги обеспечения безопасности (SA)

База данных услуг обеспечения безопасности

Услуги обеспечения безопасности могут быть организованы очень сложно. Это особенно справедливо, если Алиса хочет передавать сообщения многим людям, а Боб должен получать сообщения от многих людей. Кроме того, каждая сторона должен иметь и входящие, и исходящие SA's, чтобы позволить осуществлять двунаправленную связь. Другими словами, мы нуждаемся во множестве SA's, которые могут быть собраны в базу данных. Эта база данных называется **Базой данных Услуг обеспечения безопасности (SAD — Security Association Database)**. Базу данных можно представлять как двумерную таблицу с каждой строкой, определяющей единственную SA. Обычно есть две SAD's: одна — входящая и одна — исхо-

дящая. Рисунок 18.11 показывает концепцию исходящих и входящих SAD's для одного объекта.

< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							

База данных услуг обеспечения безопасности

SPI:	Security Parameter Index	Индекс параметра обеспечения безопасности
DA:	Destination Address	Адрес пункта назначения
AH/ESP:	Information for either one	Информация для любого одного
P:	Protocol	Протокола
Mode:	IPSec Mode Flag	Флажок Режима IPSec
SN:	Sequence Number	Порядковый номер
OF:	Overflow Flag	Флажок Переполнения
ARW:	Anti-Replay Window	Окно анти-воспроизведения
LT:	Lifetime	Время жизни
MTU:	Path MTU (Maximum Transfer Unit)	Максимальный передаваемый блок

Рис. 18.11 База данных услуг обеспечения безопасности

Когда хост должен передать пакет, который должен доставить IPSec-заголовок, хост должен найти соответствующий исходящий SAD и найти информацию для того, чтобы применить услуги безопасности к пакету. Точно так же, когда хост получает пакет, который должен доставить IPSec-заголовок, хост должен найти соответствующий вход во входящем SAD, найти нужную информацию для проверки безопасности пакета. Этот поиск должен быть задан так: приемный хост должен убедиться, что для обработки пакета используется правильная информация, чтобы обработать пакет. Каждый вход во входящем SAD выбирается, используя тройной индекс: индекс параметра обеспечения безопасности, адрес пункта назначения и протокол.

Индекс параметра обеспечения безопасности. Индекс параметра обеспечения безопасности (SPI) — число на 32 бита, которое определяет SA в пункте назначения. Как мы увидим позже, SPI определен в течение SA-переговоров. Тот же самый SPI включен во все IPSec-пакеты, принадлежащие одному и тому же входящему SA.

Адрес пункта назначения. Второй индекс — адрес пункта назначения хоста. Мы должны помнить, что хост в Интернете обычно имеет один индивидуальный адрес пункта назначения, но у него могут быть несколько адресов групповой рассылки. IPSec требует, чтобы SA был уникален для каждого адреса пункта назначения.

Протокол. IPSec имеет два различных протокола безопасности: AH и ESP. Чтобы отделить параметры и информацию, используемую для каждого протокола, IPSec требует, чтобы пункт назначения определял различный SA для каждого протокола.

Входы для каждой строки называются параметрами SA. Типичные параметры показаны в таблице 18.2.

Таблица 18.2. Типичные параметры SA

Счетчик порядкового номера Sequence Number Counter	Это значение на 32 бита, которое используется, чтобы генерировать порядковые номера для AH- или ESP-заголовка
Переполнение порядкового номера Sequence Number Overflow	Это флажок, который определяет варианты состояния в случае переполнения порядкового номера
Окно антивоспроизведения Anti-Replay Window	Оно обнаруживает входящий воспроизведенный пакет AH или пакет ESP
Информация AH AH Information	Эта секция содержит информацию для протокола AH: 1. Алгоритм аутентификации 2. Ключи 3. Время жизни ключа 4. Другие связанные параметры
Информация ESP ESP Information	Эта секция содержит информацию для протокола ESP: 1. Алгоритм шифрования 2. Алгоритм аутентификации 3. Ключи 4. Время жизни ключа 5. Вектор инициализации 6. Другие связанные параметры
Время жизни SA Lifetime	Определяет время жизни для SA
Режим IPSec IPSec Mode	Определяет режим, транспортный или туннельный
Путь MTU Path MTU	Определяет путь МАКСИМАЛЬНЫЙ ПЕРЕДАВАЕМЫЙ БЛОК (фрагментацию).

18.4. Стратегия безопасности

Другой аспект импорта IPSec — **Стратегия безопасности (SP — Security Policy)**, которая определяет тип безопасности, предоставляемой пакету, когда его нужно передать или когда его нужно принять. Перед тем как использовать SAD, рассмотренный в предыдущем разделе, хост должен определить заранее заданную стратегию обслуживания этого пакета.

База данных стратегии безопасности

Каждый хост, который применяет IPSec-протокол, должен хранить **Базу данных стратегии безопасности (SPD — Security Policy Database).** При этом, как и

раньше, необходимо иметь входящую SPD и исходящую SPD. К каждому входу в SPD можно обратиться, используя индекс из шести позиций: исходный адрес, адрес пункта назначения, название протокола, исходный порт и порт пункта назначения, как показано на рис. 18.12.

Индекс	Стратегия
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	

SA:	Source Address	Исходный Адрес
DA:	Destination Address	Адрес пункта назначения
P:	Protocol	Протокол
SPort:	Source Port	Исходный Порт
Dport:	Destination port	Порт Пункта назначения

Рис. 18.12. База данных Стратегии Безопасности (SPD)

Источник и адреса пункта назначения могут быть индивидуальные, групповой рассылки или групповой символ (wildcard[1]) — их имя обычно определяет в Интернете объект доменной системы имен (DNS). Протокол является или AH, или ESP. Источник и порты пункта назначения — адреса порта для процесса, функционирующего в хостах пункта назначения и источнике.

Исходящая SPD

Когда пакет нужно передать, то работают с исходящим SPD. Рисунок 18.13 показывает обработку пакета передатчиком.

Вход исходящего SPD состоит из шестизначного индекса; выход содержит один из трех результатов.

1. **Скинуть.** Это означает, что пакет, определенный этим индексом, нельзя передать; он отбрасывается.
2. **Обход.** Это означает, что нет никакой стратегии для пакета с этим индексом стратегии; пакет передают в обход, не применяя действий с заголовком безопасности.
3. **Применить.** В этом случае применяется работа с заголовком безопасности. При этом могут возникнуть две ситуации:
   a. Если исходящую SA уже установили, возвращается тройной индекс SA, который выбирается соответствующей SA из исходящего SAD. Формируется AH или заголовок ESP; шифрование, установление подлинности или оба этих действия применяются в соответствии с выбранной SA. Пакет передается.

[1] wildcard – в англоязычной литературе понимается как символ, указывающий, что подходит любой символ из группы. Например, при поиске нужных файлов знак «*» обозначает «любая буква».

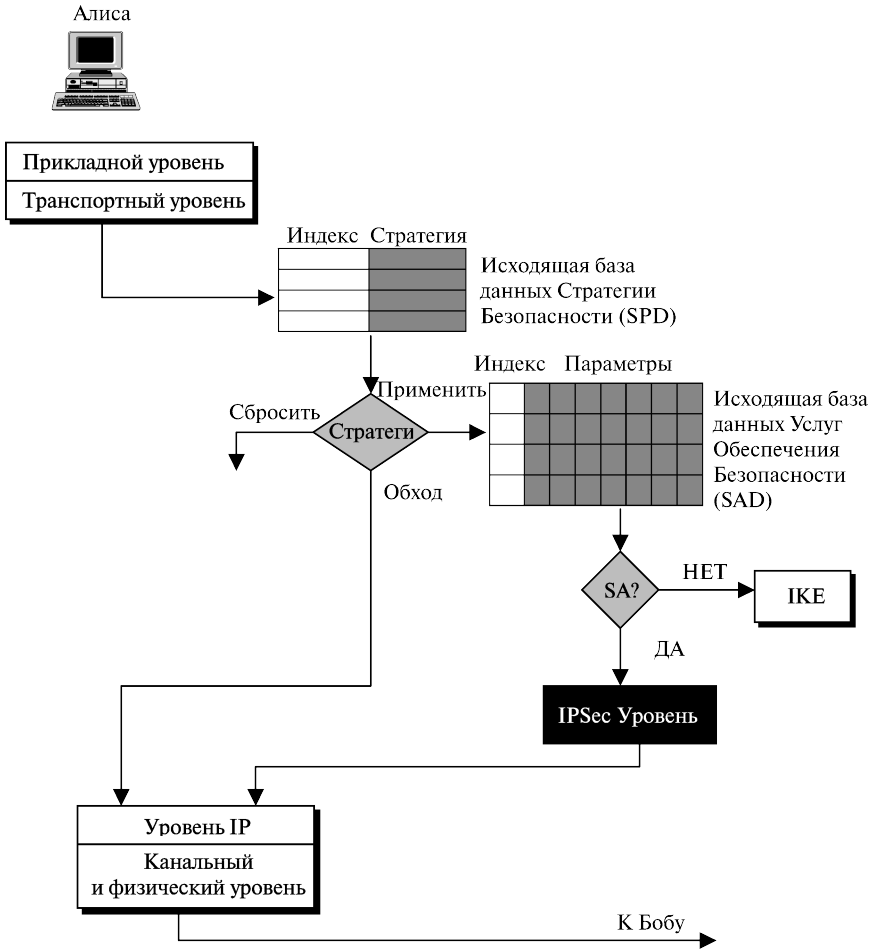


Рис. 18.13 Исходящий процесс

б. Если исходящая SA еще не установлена, то вызывается протокол Интернет обмена ключами (IKE — Internet Key Exchange) (см. следующую секцию), чтобы создать исходящую и входящую SA для этого трафика. Исходящая SA добавляется источником к исходящей SAD; входящая SA добавляется пунктом назначения к входящей SAD.

Входящий SPD

Когда пакет прибывает, то проводится работа с входящей SPD. К каждому входу во входящей SPD также обращаются, используя тот же самый шестикратный индекс. Рисунок 18.14 показывает обработку пакета приемником.

Вход к входящему SPD — шестикратный индекс; выход — один из трех результатов.

1. **Сбросить.** Это означает, что пакет, определенный стратегией, должен быть отброшен.
2. **Обход.** Это означает, что нет никакой стратегии для пакета с этим индексом стратегии; пакет обрабатывается, игнорируя информацию от AH или заголовок ESP. Пакет доставляют транспортному уровню.
3. **Применить.** В этом случае заголовок безопасности должен быть обработан. Здесь могут возникнуть две ситуации.
   а. если входящая SA уже установлена, возвращается тройной индекс SA, который выбирается соответствующей SA из входящего SAD. Применяются дешифрование, установление подлинности или оба этих действия. Если пакет передает критерии безопасности, AH или заголовок ESP забракован, и пакет доставляют транспортному уровню;
   б. если SA еще не установлена, то пакет должен быть забракован.

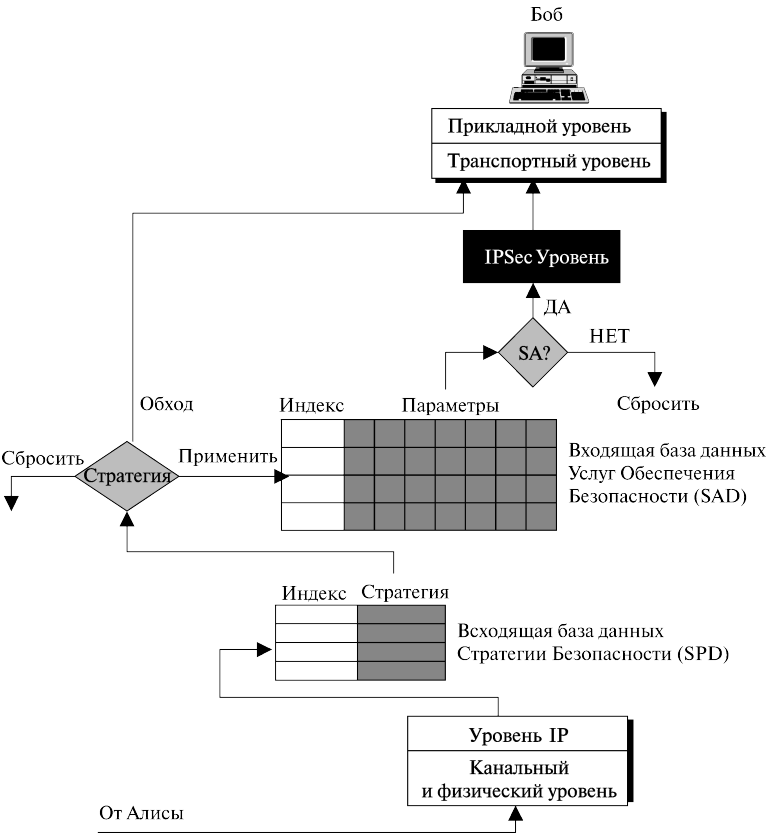


Рис. 18.14 Входящий процесс

18.5. Протокол интернет-обмена ключами (IKE)

Протокол Интернет-обмена ключами **(IKE — Internet Key Exchange)** должен создавать и входящие, и исходящие услуги обеспечения безопасности. Как мы обсуждали в предыдущей секции, когда пакет IP должен быть передан между равными уровнями, тогда обращаются к базе данных стратегии безопасности (SPDB), чтобы видеть, есть ли SA для такого типа трафика. Если нет такой SA, вызывается IKE, чтобы установить ее.

Протокол Интернет-обмена ключами создает услуги обеспечения безопасности SA's для протокола IPSec.

IKE — сложный протокол, основанный на трех других протоколах: OAKLEY, SKEME и ISAKMP, как показано на рис. 18.15.

Рис. 18.15. Компоненты протокола управления ключами в Интернете

Протокол **Oakley** был разработан Хиллари Орманом. Это протокол создания ключа, основанный на методе смены ключей Диффи-Хеллмана, но с некоторыми усовершенствованиями. Oakley — протокол со свободным форматом, в том смысле, что он не определяет формат сообщений, которыми будут обмениваться стороны. В этой лекции мы не обсуждаем протокол Oakley непосредственно, но показываем, как IKE использует его идеи.

SKEME, разработанный Хьюго Кравчиком, является еще одним протоколом для смены ключей. Он использует шифрование открытым ключом для установления подлинности объекта в протоколе смены ключей. Мы коротко рассмотрим, как один из методов, используемых IKE, базируется на SKEME.

Internet-услуги обеспечения безопасности и протокол управления ключами (ISAKMP) являются протоколом, разработанным Агентством Национальной безопасности (NSA), который фактически осуществляет обмен сообщениями, определенными в IKE. Он определяет некоторые пакеты, протоколы и параметры, которые позволяют проводить обмен сообщениями IKE в стандартизированных, отформатированных сообщениях, чтобы создать SA's. Мы обсудим ISAKMP в следующей секции как протокол, осуществляющий IKE.

В этой секции мы поговорим непосредственно об IKE как о механизме для создания услуг безопасности в SA's в IPSec.

Улучшенный протокол управления ключами Диффи-Хеллмана

Идея смены ключей в IKE базируется на протоколе Диффи-Хеллмана. Этот протокол обеспечивает ключ сеанса между двумя равными по уровню процессами, без необходимости существования любых предварительных средств безопасности.

В лекции 15 мы обсудили метод Диффи-Хеллмана; концепция этого метода суммирована на рис. 18.16.

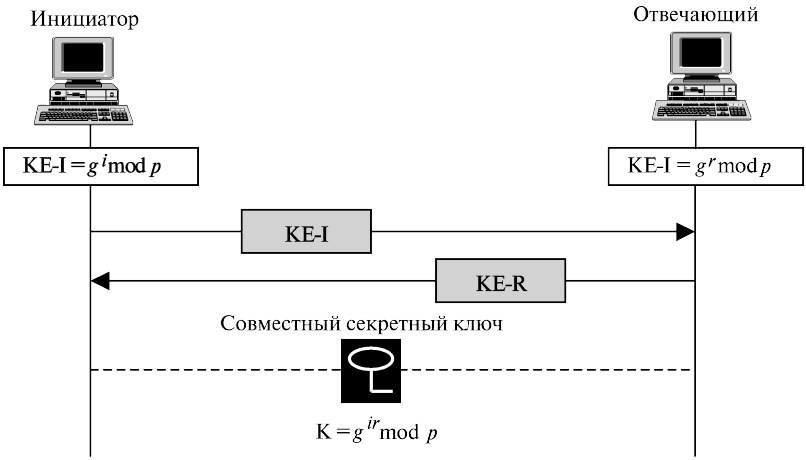


Рис. 18.16. Обмен ключами по методу Диффи-Хеллмана

В первоначальном методе обмена ключей Диффи-Хеллмана две стороны создают симметричный ключ сеанса, чтобы обмениваться данными. При этом им не надо помнить или хранить ключ для будущего использования. Перед установлением симметричного ключа эти две стороны должны выбрать два числа: p и g. Первое число, p, является большим простым числом порядка 300 десятичных цифр (1024 бита). Второе число, g, — генератор в группе $<Z_p^*, \times>$. Алиса выбирает большое случайное число i и вычисляет KE-I = $g^i \bmod p$. Она передает KE-I Бобу. Боб выбирает другое большое случайное число r и вычисляет KE-R = $g^r \bmod p$. Он передает KE-R Алисе. Напоминаем, что KE-I и KE-R, как полуключи метода Диффи-Хеллмана, сгенерированы для равных по уровню процессов. Они должны быть объединены вместе, чтобы создать полный ключ, K = $g^{ir} \bmod p$ у. K — это симметричный ключ для сеанса.

Протокол Диффи-Хеллмана имеет некоторые слабости, которые должны быть устранены прежде, чем он станет приемлемым для обмена ключей в Интернет.

Засоряющая атака

Первая проблема с протоколом Диффи-Хеллмана — **засоряющая атака** или *атака отказа в обслуживании.* Злоумышленник может передать много полуклю-

чей (g^x mod q) сообщения Бобу, симулируя, что они из различных источников. Тогда Боб должен вычислить различные ответы (g^y mod q) и в то же самое время вычислить полный ключ (g^{xy} mod q); это загрузит Боба настолько, что он может прекратить отвечать на любые другие сообщения. Он будет отказывать в обслуживании клиентам. Такое может случиться, потому что протокол Диффи-Хеллмана требует большого времени для вычислений.

Чтобы предотвратить атаку засорения, мы можем добавить к протоколу два дополнительных сообщения и вынудить эти две стороны передать **cookies**[1].

Рисунок 18.17 показывает обработку, которая может предотвратить засоряющую атаку. **Cookies** — результат хэширования уникальных идентификаторов процессов, равных по уровню (таких как адрес IP, число порта и протокол, секретное случайное число, известное сторонам, которые генерирует **cookies**, и метка времени).

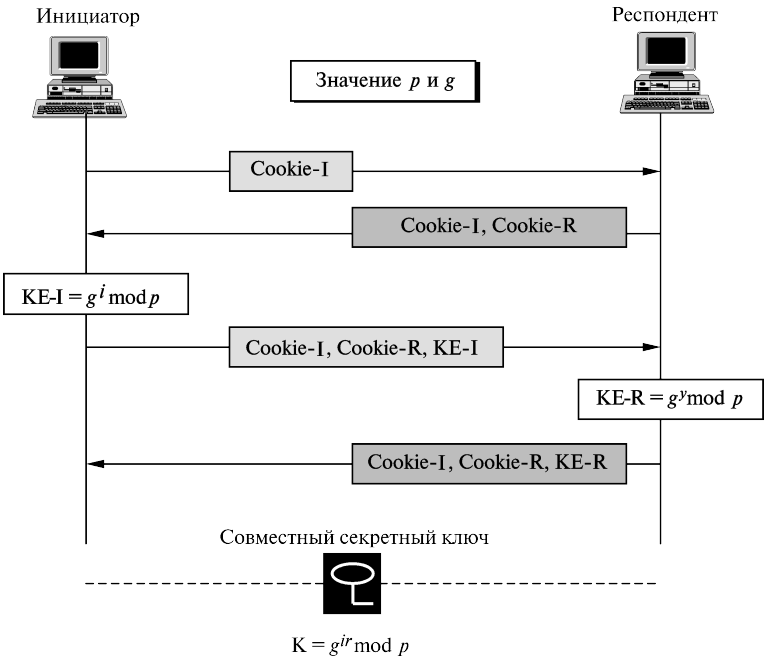


Рис. 18.17. Метод Диффи-Хеллмана с cookies

Инициатор передает собственное cookie; ответная сторона — свой cookie. Оба cookies повторяются в неизмененном виде в каждом последующем сообщении. Вычисления полуключей и ключа сеанса отложены до возвращения cookies.

[1] Cookies (на сленге – «плюшки») – небольшой фрагмент данных, содержащих предысторию обращений данного пользователя к данному WWW-серверу; автоматически создается сервером на машине пользователя.

Если любой из процессов этого уровня — хакер, делающий попытку засоряющей атаки, cookies не возвращается; соответствующая сторона не тратит время и усилие на вычисление полуключа или ключа сеанса. Например, если инициатор — хакер, использующий фиктивный адрес IP, то инициатор не получает второе сообщение и не может передать третье сообщение. Процесс прерывается.

Чтобы защититься от засоряющей атаки, IKE использует cookies.

Атака воспроизведения

Подобно другим протоколам, которые мы рассматривали до сих пор, протокол Диффи-Хеллмана не устойчив к **атаке воспроизведения**; информация от одного сеанса может быть записана без расшифровки и воспроизведена в будущем сеансе злоумышленником. Ради предотвращения этой атаки мы можем добавить nonce к третьему и четвертому сообщениям, чтобы сохранить свежесть сообщения.

Чтобы защититься против атаки воспроизведения, IKE использует nonce.

Атака «посредника»

Третий и наиболее опасный вид атаки протокола Диффи-Хеллмана — атака «посредника», предварительно рассмотренная в лекции 15. Ева может войти в середину диалога и создать один ключ между Алисой и собой и другой ключ между Бобом и собой. Сорвать эту атаку не так просто, как первые две. Мы должны подтвердить аутентификацию каждой стороны.

Алиса и Боб должны убедиться, что сохранена целостность сообщений и что оба аутентифицированы по отношению друг к другу.

Аутентификация обмена сообщений (целостность сообщения) и аутентификация сторон (аутентификация объекта) требует, чтобы каждая сторона доказала, что у нее есть требуемый код идентификации. Чтобы сделать это, каждый должен доказать, что он обладает секретностью.

Чтобы защититься против атаки «посредника», IKE требует, чтобы каждая сторона показала, что она обладает секретностью.

В IKE секретность может заключаться в одном из следующих сочетаний ключей:

а. предварительный открытый ключ засекречивания;

б. предварительно известная пара открытого ключа шифрования/дешифрования. Объект должен показать, что сообщение, зашифрованное объявленным открытым ключом, может быть расшифровано им с помощью соответствующего секретного ключа;

в. предварительно известная пара открытого ключа цифровой подписи. Объект должен показать, что он может подписать сообщение своим секретным ключом, который может быть проверен его объявленным открытым ключом.

Фазы IKE

IKE создает SA's для обмена сообщениями протокола, такого как IPSec. IKE, однако, должен обмениваться конфиденциальными и аутентифицированными сообщениями. Какие SA's обеспечивает протокол IKE для себя? Можно догадаться, что он требует бесконечной цепочки SA's: IKE должен создать SA's для IPSec, протокол X должен создать SA's для IKE, протокол Y должен создать SA's для протокола X, и так далее. Для того чтобы решить эту дилемму и в то же время оставить IKE независимым от протокола IPSec, разработчики IKE разделили IKE на две фазы. В фазе I IKE создает SA's для фазы II. В фазе II IKE создает SA's для IPSec или некоторых других протоколов.

Фаза I является базовой; фаза II задана для протокола.

IKE разделен на две фазы: фаза I и фаза II. Фаза I создает SA's для фазы II; фаза II создает SA's для протокола обмена данными, например, такого как IPSec.

Однако остается вопрос: как защищена фаза 1? В следующей секции мы покажем, как фаза 1 использует SA, который сформирован постепенно. Более ранние сообщения заменяются в исходном тексте; более поздние сообщения заменяются созданными из ранних сообщений.

Фазы и режимы

Чтобы учесть разнообразие методов обмена, IKE определил для фаз режимы. В настоящее время есть два режима для фазы I: *основной режим* и *энергичный режим*. Единственный режим для фазы II — *быстрый режим*. Рисунок 18.18 показывает отношения между фазами и режимами.

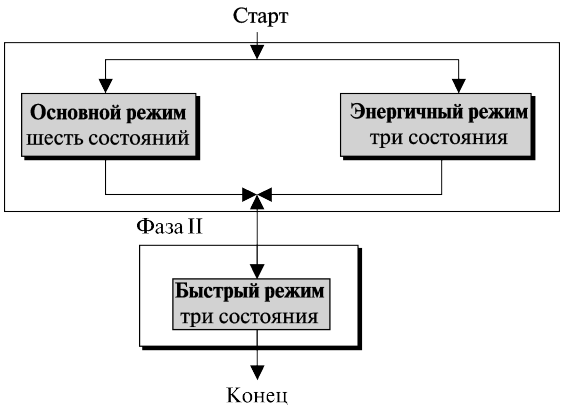


Рис. 18.18. Фазы IKE

В зависимости от характера предварительной секретности между этими двумя сторонами, режимы фазы I могут использовать один из четырех различных методов аутентификации: метод предварительного открытого ключа засекречивания, метод первоначального открытого ключа, метод пересмотренного открытого ключа или метод цифровой подписи, как это показано на рис. 18.19.

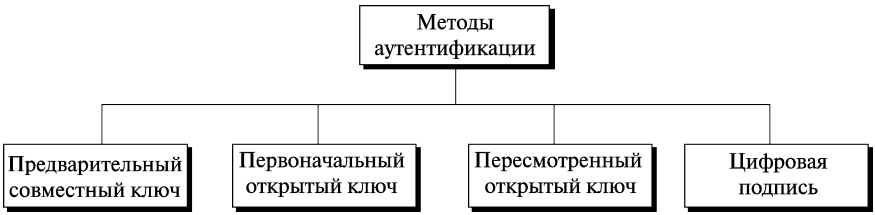


Рис. 18.19. Методы основного или энергичного режима

Фаза I: основной режим

В **основном режиме** инициатор и респондент обмениваются шестью сообщениями. В первых двух сообщениях они обмениваются cookies (чтобы защитить против засоряющей атаки и договориться о параметрах SA): инициатор передает ряд предложений; респондент выбирает одно из них. Когда они обменяются первыми двумя сообщениями, инициатор и респондент знают параметры SA и уверены, что другая сторона существует и засоряющая атака не возникнет.

В третьих и четвертых сообщениях инициатор и респондент обычно обмениваются своими полуключами (g^i и g^r метода Диффи-Хеллмана) и их nonce (для защиты ответа). В некоторых методах обмениваются другой информацией, о которой мы поговорим позже. Обратите внимание, что полуключи и nonce не передаются с первыми двумя сообщениями, потому что две стороны должны сначала получить гарантию, что засоряющая атака невозможна.

После обмена третьим и четвертым сообщениями каждая сторона может вычислить общую секретность между ними в дополнение к ее отдельному дайджесту хэширования. Общая секретность SKEYID (ID ключа засекречивания) зависит от метода вычисления, как показано ниже. В уравнениях ***prf*** (псевдослучайная функция) — хэш-функция ключа, которая определена в течение фазы переговоров.

SKYID = *prf (*предварительный совместный ключ*,* N-I | N-R)	(метод предварительного совместного ключа)
SKYID = *prf(*N-I | N-R, g^{ir})	(метод открытого ключа)
SKYID = *prf((hash (*N-I | N-R), Cookie –I | Cookie –R)	(цифровая подпись)

Другая общая секретность вычисляется следующим образом:

SKYID_d = *prf((*SKEYID, $g^{i\,r}$| Cookie –I | Cookie –R| 0)

SKYID_a = *prf((*SKEYID, SKEYID_d | $g^{i\,r}$| Cookie –I | Cookie –R| 1)

SKYID_e = *prf((*SKEYID, SKEYID_a | $g^{i\,r}$| Cookie –I | Cookie –R| 2)

SKEYID_d (derived — производный ключ) — ключ для создания других ключей. SKEYID_a — ключ аутентификации, и SKEYID_e применяется для ключа шифрования; обе эти секретности используются в течение фазы переговоров. Первый параметр (SKEYID) вычисляется для каждого метода обмена ключами отдельно. Второй параметр — конкатенация различных данных. Обратите внимание, ключ для *prf* — всегда SKEYID.

Эти две стороны также вычисляют два дайджеста хэширования, HASH-I и HASH-R, которые используются в главном режиме трех из этих четырех методов. Вычисления показаны ниже:

HASH-I = *prf((*SKEYID, KE-I | KE-R | Cookie –I | Cookie –R |SA-I | ID-I)

HASH-R = *prf((*SKEYID, KE-I | KE-R | Cookie –I | Cookie –R |SA-I | ID-R)

Обратите внимание, что первый дайджест применяет ID-I, в то время как второй — ID-R. Оба используют SA-I — полные SA данные, посланные инициатором. Ни один из них не включает предложение, выбранное респондентом, поскольку нужно защитить предложение, посланное инициатором, от изменений злоумышленника. Например, злоумышленник мог бы попробовать передать список предложений более уязвимых, чтобы облегчить себе атаку. Точно так же, если SA не включен, злоумышленник мог бы изменить выбранное предложение на другое, благоприятное для себя. Обратите внимание, что одна сторона не должна знать ID другой стороны при вычислении хэш.

После вычисления ключей и хэширования каждая сторона передает хэш другой стороне, чтобы подтвердить свою подлинность. Инициатор передает HASH-I респонденту как доказательство, что она — Алиса. Только Алиса знает секретность аутентификации, и только она может вычислить HASH-I. Если HASH-I, вычисленный Бобом, соответствует HASH-I, посланному Алисой, она аутентифицирована. Тем же самым способом Боб может подтвердить свою подлинность Алисе, посылая HASH-R.

Обратите внимание, что здесь есть одна тонкость. Когда Боб вычисляет HASH I, он нуждается в ID Алисы, и наоборот. В некоторых методах ID передают с помощью предыдущих сообщений; в других — с хэшем либо с хэшем и с ID, зашифрованным SKEYID_e.

Метод предварительного совместного ключа

В методе предварительного совместного ключа симметричный ключ используется для аутентификации равноправных партнеров друг для друга. Рисунок 18.20 показывает аутентификацию совместным ключом в главном режиме.

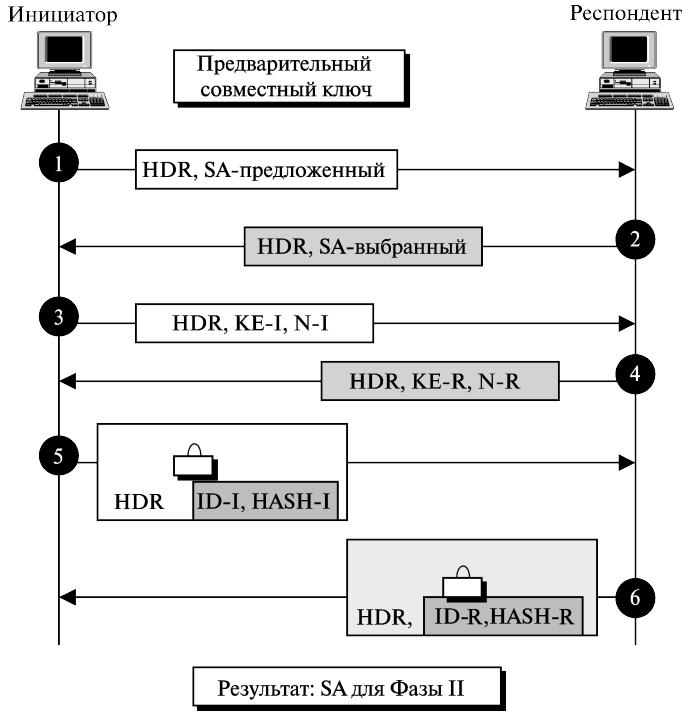


Рис. 18.20. Основной режим метода с предварительным совместным секретным ключом

В первых двух сообщениях инициатор и респондент обмениваются cookies (в общем заголовке и параметрах SA). В следующих двух сообщениях они обмениваются полуключами и nonces (см. лекцию 15). Теперь эти две стороны могут создать SKEYID и два ключевых хэша (HASH-I и HASH-R). В пятом и шестом сообщениях две стороны обмениваются созданными хэшами и их ID. Чтобы защищать ID и хэши, последние два сообщения зашифрованы с SKEYID_e.

Обратите внимание, что предварительный совместный ключ обеспечивает секретность между Алисой (инициатором) и Бобом (респондентом). Ева (злоумышленник) не имеет доступа к этому ключу. Ева не может создать SKEYID и поэтому не может создать ни HASH-I, ни HASH-R. Обратите внимание также, что обмен ID должен быть сделан в сообщениях 5 и 6, чтобы обеспечить вычисление хэша.

С этим методом есть одна проблема. Боб не может расшифровать сообщение, если он не знает предварительный совместный ключ, — то есть он должен знать, кто Алиса (знать ее ID). Но ID Алисы зашифрован в сообщении 5. Разработчик этого метода утверждает, что ID в этом случае должен быть в адресе каждой стороны.

Это — не проблема, если Алиса находится в постоянном хосте (адрес IP установлен). Однако если Алиса переходит из одной сети в другую, это уже проблема.

Первоначальный метод открытого ключа

В первоначальном методе открытого ключа инициатор и респондент доказывают их подлинность, показывая, что они обладают секретным ключом, связанным с их объявленным открытым ключом. Рисунок 18.21 иллюстрирует обмен сообщениями при использовании метода первоначального открытого ключа.

Первые два сообщения — те же, что в предыдущем методе. В третьем сообщении инициатор передает свой полуключ, once и ID. В четвертом сообщении респондент поступает аналогично. Однако nonce и ID зашифрованы открытым ключом приемника и расшифрованы секретным ключом приемника. Как видно из рис. 18.21, nonce и ID зашифрованы отдельно, потому что, как мы увидим позже, они кодируются отдельно от остальных нагрузок.

Отличие между этим методом и предыдущим в том, что обмен ID проводится в третьем и четвертом сообщении вместо пятого и шестого сообщений. Пятое и шестое сообщения только доставляют хэши.

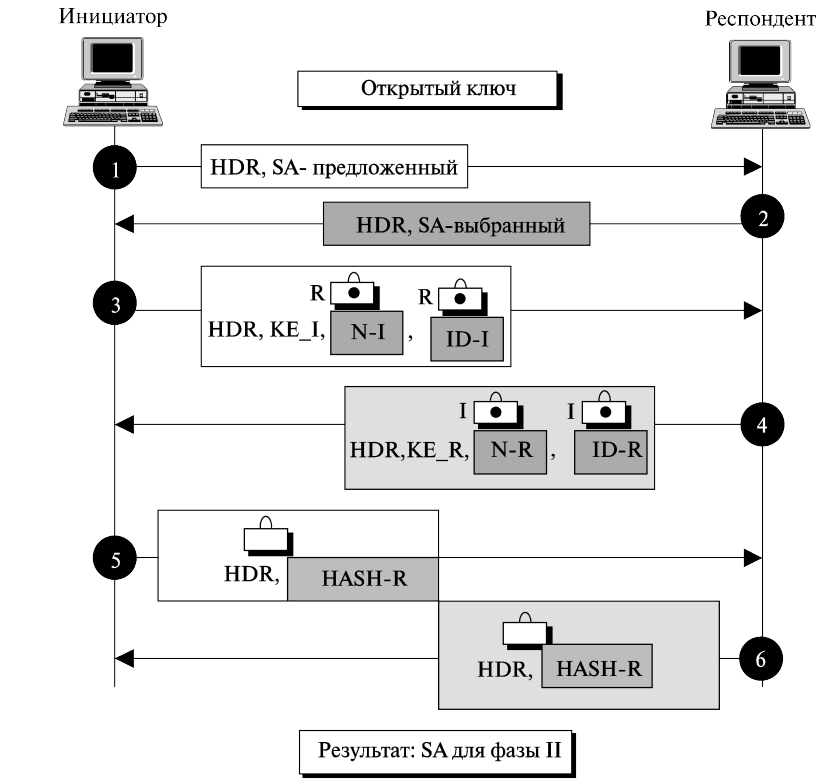
Вычисление SKEYID в этом методе базируется на хэшировании nonce и симметричного ключа. Хэширование nonce используется как ключ для функции HMAC. Обратите внимание, что здесь мы используем двойное хэширование. Хотя SKEYID, а, следовательно, его хэш непосредственно не зависит от секретности, которой обладает каждая сторона, они связаны косвенно. SKEYID зависит от nonce, а nonce может быть расшифрован только секретным ключом (секретность) приемника. Следовательно если вычисленный хэш соответствует полученному, он доказывает, что каждая сторона — тот, кем он себя утверждает.

Пересмотренный метод открытого ключа

Метод первоначального открытого ключа имеет некоторые недостатки. Во-первых, две операции шифрования/дешифрования открытого ключа накладывают тяжелую нагрузку на инициатора и респондента. Во-вторых, инициатор не может послать свой сертификат, зашифрованный открытым ключом респондента, так как любой мог сделать это с ложным сертификатом. Метод был пересмотрен так, чтобы открытый ключ использовался только для создания временного ключа засекречивания, как показано на рис. 18.22.

Обратите внимание, что два временных секретных ключа созданы из хэша nonce и cookies. Инициатор использует открытый ключ респондента, чтобы передать его nonce. Респондент дешифрирует nonce и вычисляет временный ключ инициатора, после чего могут быть расшифрованы полуключ, ID и дополнительный сертификат. Два временных ключа засекречивания, K-I и K-R, вычисляются следующим образом:

$$\text{K-I} = \textit{prf}(\text{N} - \text{I}, \text{Cookie-I}) \qquad \text{K-R} = \textit{prf}(\text{N} - \text{R}, \text{Cookie-R})$$



KE-I (KE-R): Полуключ инициатора (респонедента)
N-I(N-R): Nonce инициатора (респонедента)
ID-I (ID-R): ID инициатора (респонедента)
HASH-I, (HASH-R) Хеш инициатора (респонедента)
HDR: Общий заголовок включающий coocies
Cert-I (Cert-R) Хеш инициаторов (респонедента)

I Засекреченный открытый ключ инициатора

R Засекреченный открытый ключ респондента

Засекреченный SKEYID_e

Рис. 18.21. Главный режим метода с первоначальным открытым ключом

Метод цифровой подписи

В этом методе каждая сторона показывает, что она обладает сертифицированным секретным ключом, связанным с цифровой подписью. Рисунок 18.23 иллюстрирует обмен сообщениями при этом методе. Он похож на метод предварительного совместного ключа во всем, кроме процедуры вычисления SKEYID.

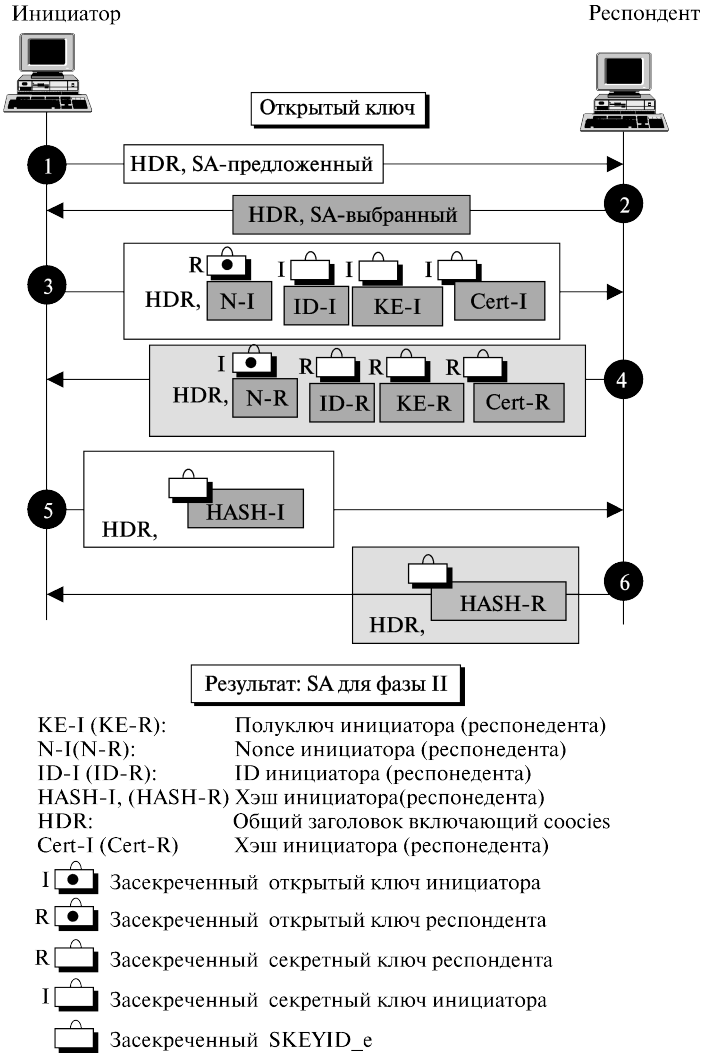


Рис. 18.22. Главный режим метода с пересмотренным открытым ключом

Обратите внимание, что в этом методе передача сертификата является необязательной операцией. Сертификат можно передать, потому что он может быть зашифрован SKEYID_e, который не зависит от ключа подписи. В сообщении 5 инициатор подписывает всю информацию обмена в сообщениях 1-4 своим ключом подписи. Респондент верифицирует подпись, используя открытый ключ инициатора, который подтверждает подлинность инициатора. Аналогично, в сообще-

нии 6 респондент подписывает всю информацию обмена своим ключом подписи. Инициатор верифицирует подпись.

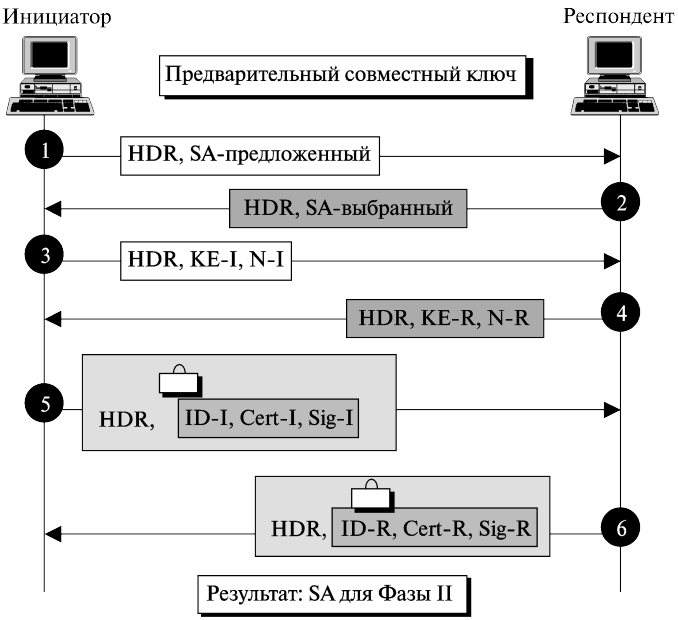


Рис. 18.23. Главный режим метода цифровой подписи

Фаза I: энергичный режим

Каждый **энергичный режим** — сжатая версия соответствующего главного режима. Вместо шести сообщений в обмене участвуют только три. Сообщения 1 и 3 объединены в одно — первое сообщение. Сообщения 2, 4 и 6 объединены во второе сообщение. Сообщение 5 передают как третье сообщение. Идея та же самая, что и в главном режиме.

Метод предварительного совместного ключа

Рисунок 18.24 показывает метод предварительного совместного ключа в энергичном режиме. Обратите внимание, что после получения первого сообще-

ния респондент может вычислить SKEYID и, следовательно, HASH-R. Но инициатор не может вычислить SKEYID, пока не получит второе сообщение. HASN-I в третьем сообщении может быть зашифровано.

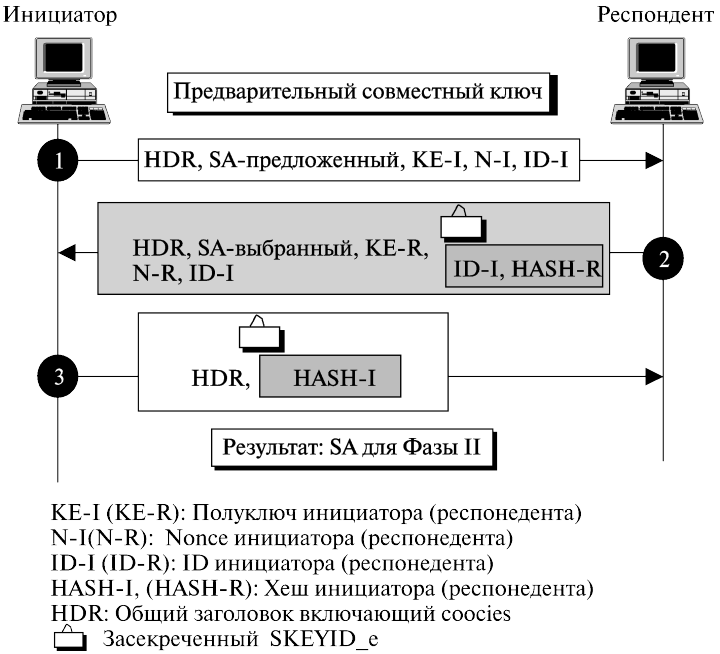


Рис. 18.24. Энергичный режим метода с предварительным совместным ключом

Метод первоначального открытого ключа

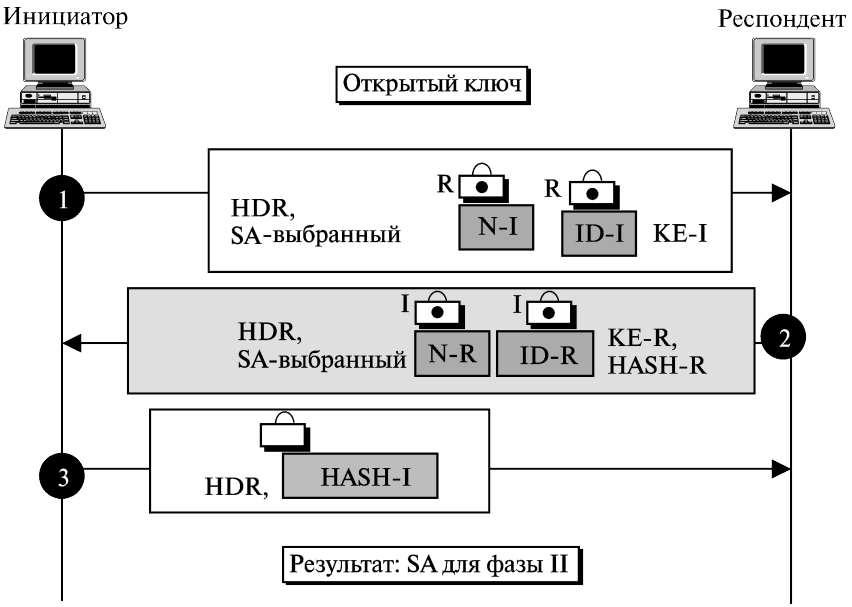
Рисунок 18.25 показывает обмен сообщениями с использованием метода первоначального открытого ключа в энергичном режиме. Обратите внимание, что респондент может вычислить SKEYID и HASH-R после получения первого сообщения, но инициатор должен ждать, пока не получит второе сообщение.

Пересмотренный метод открытого ключа

Рисунок 18.26 показывает пересмотренный метод открытого ключа в энергичном режиме. Идея — та же самая, что и в главном режиме, за исключением того, что некоторые сообщения объединены.

Метод цифровой подписи

Рисунок 18.27 показывает метод цифровой подписи в энергичном режиме. Идея — та же самая, что и в главном режиме, за исключением того, что некоторые сообщения объединены.



KE-I (KE-R): Полуключ инициатора (респонедента)
HDR: Общий заголовок, включающий coocies
N-I(N-R): Nonce инициатора (респонедента)
ID-I (ID-R): ID инициатора (респонеденнта)
HASH-I, (HASH-R): Хеш инициатора (респонеденнта)
I Засекречивание с помощью открытого ключа инциатора
R Засекречивание с помощью открытого ключа респодента
Засекреченние с помощью SKEYID_e

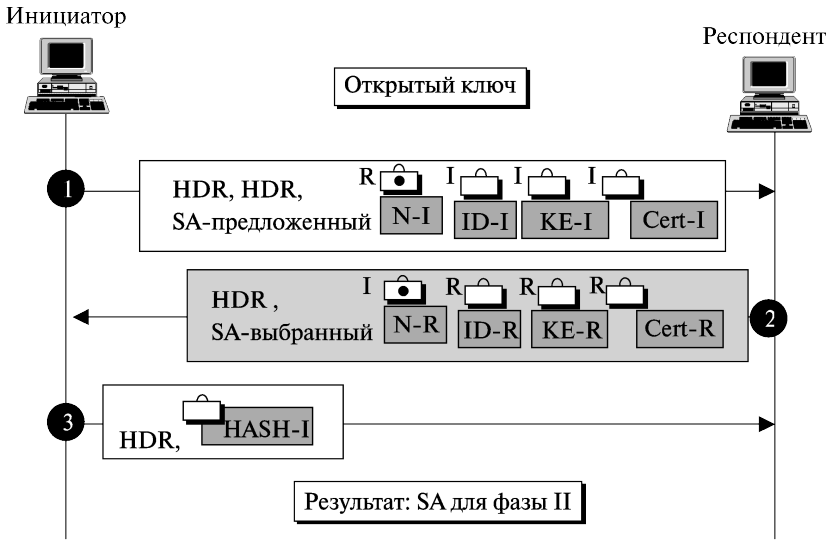
Рис. 18.25. Энергичный режим метода с первоначальным ключом

Фаза II: быстрый режим

После того как услуги безопасности (SA's) были созданы или в главном режиме, или в энергичном режиме, может быть начата фаза II. Для фазы II в настоящее время есть только один режим — *быстрый режим*. Этот режим находится под управлением SA's IKE, созданного фазой 1. Каждый метод быстрого режима может начать работу любым главным или агрессивным режимом.

Быстрый режим использует SA's IKE, чтобы создать IPSec SA's (или SA's для любого другого протокола). Рисунок 18.28 показывает обмен сообщениями в ходе быстрого режима.

В фазе II любая сторона может быть инициатором: то есть инициатор фазы II может быть инициатором или респондентом фазы 1.



KE-I (KE-R): Полуключ инициатора (респонедента)
N-I(N-R): Nonce инициатора (респонедента)
ID-I (ID-R): ID инициатора (респонедента)
HASH-I, (HASH-R): Хеш инициатора (респонедента)
HDR: Общий заголовок включающий coocies
Cert-I (Cert-R): Хэш инициатора (респонедента)

I Засекреченный открытый ключ инициатора
R Засекреченный открытый ключ респондента
R Засекреченный секретный ключ респондента
I Засекреченный секретный ключ инициатора
Засекреченный SKEYID_e

Рис. 18.26. Энергичный режим метода с пересмотренным открытым ключом

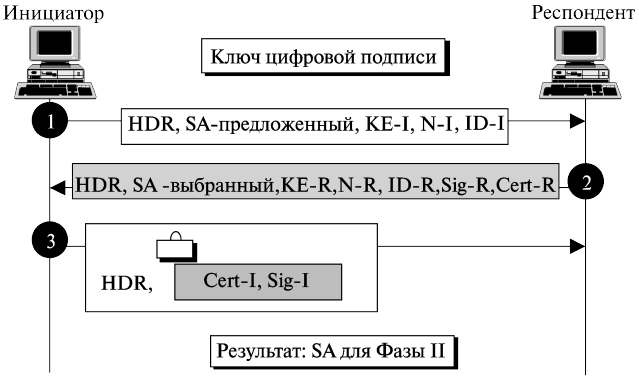
Инициатор передает первое сообщение, которое включает в себя ключевой HMAC HASH 1 (будет рассмотрен позже), полный SA, созданный в фазе 1, новый nonce (N–I), дополнительный новый полуключ Диффи-Хеллмана (KE-I) и иногда ID обеих сторон. Второе сообщение похоже, но переносит ключевой HMAC HASH2, nonce респондента (N-R), и, если есть, то полуключ Диффи-Хеллмана, созданный респондентом. Третье сообщение содержит только HMAC HASH3 ключа.

Сообщения аутентифицированы с помощью использования трех HMAC ключа: HASH1, HASH2 и HASH3. Они вычисляются следующим образом:

HASH1 = ***prf*** (SKEYID_d, MsgID | SA | N-I)

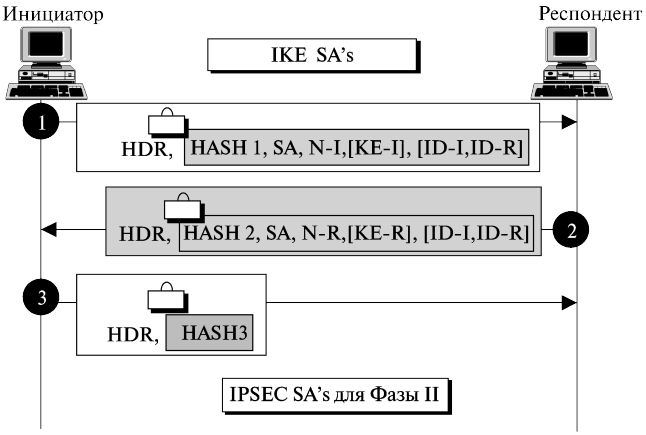
HASH2 = ***prf*** (SKEYID_d, MsgID | SA | N-R)

HASH3 = ***prf*** (SKEYID_d, 0 | MsgID [SA | N-I | N-R)



HDR: Общий заголовок включающий cookies
Sig-I: Цифровая подпись инициатора
Sig-R: Цифровая подпись респондента
Cert-I (Cert-R): Сертификат инициатора (респондента)
N-I (N-R): Nonce инициаторов (респонедента)
KE-I (KE-R): Полуключ инициаторов (респонедента)
ID-I (ID-R): ID инициатора (респонедента)
Засекреченный SKEYID_e

Рис. 18.27. Энергичный режим метода цифровой подписи



KE-I (KE-R): Полуключ инициаторов (респонедента)
N-I(N-R): Nonce инициаторов (респонедента)
ID-I (ID-R): ID инициаторов (респонеденнта)
HASH-I, (HASH-R): Хэш инициатора (респодента)
HDR: Общий заголовок включающий cookies
Засекреченный SKEYID_e

Рис. 18.28. Быстрый режим

Каждый HMAC включает сообщение IKE (MsgID), используемое в заголовке ISAKMP. Включение MsgID предотвращает одновременное создание фазы II и возможное столкновение.

Все три сообщения зашифрованы для конфиденциальности, используя SKEYID_e, созданный в течение фазы 1.

Идеальная прямая безопасность (PFS)

После установления IKE SA и вычисления SKEYID_d в фазе 1 все ключи для быстрого режима получены из SKEYID_d. Так как из единственной фазы I фаза II может быть получена много раз, безопасность фазы II является уязвимой, если злоумышленник имеет доступ к SKEYID_d. Чтобы воспрепятствовать этому, IKE применяет опцию **идеальная прямая безопасность (PFS — Perfect Forward Security)**. В этой опции происходит обмен дополнительным полуключом Диффи-Хеллмана, и в результате совестный ключ (g^{ir}) используется в вычислении материала для ключей (см. следующий раздел) для IPSec. PFS эффективен, если ключ Диффи-Хеллмана после вычисления материала для ключа немедленно удален в каждом быстром режиме.

Материалы для ключей

После обмена в фазе II SA для IPSec создан, включая материал для ключа, K, который может использоваться в IPSec. Его значение получено следующим образом:

$K = \mathit{prf}(\text{SKEYID\_d}, \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$ (без PFS)
$K = \mathit{prf}(\text{SKEYID\_d}, g^{ir} \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$ (с PFS)

Если длина K слишком коротка для конкретного выбранного шифра, создается последовательность ключей, где каждый ключ получается из предыдущего, и ключи конкатенируются для того, чтобы сделать длинный ключ. Мы показываем случай без PFS; для варианта с PFS мы должны добавить g^{ir}.

Созданный материал для ключей — однонаправленный; каждая сторона создает свой различный материал для ключей, поэтому используемый в каждом направлении материал различен.

$K_1 = \mathit{prf}(\text{SKEYID\_d}, \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$
$K_2 = \mathit{prf}(\text{SKEYID\_d}, K_1 \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$
$K_3 = \mathit{prf}(\text{SKEYID\_d}, K_2 \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$
.....
$K = K_1 \mid K_2 \mid K_3 \mid$

Материал для ключей, созданный после фазы II, — однонаправленный; есть один ключ для каждого направления.

SA-алгоритмы

В заключение этого раздела приведем алгоритмы, с помощью которых договариваются в течение первых двух обменов сообщениями в IKE.

Группы Диффи-Хеллмана

Первые переговоры включают группу Диффи-Хеллмана, используемую для того, чтобы обмениваться полуключами. Она состоит из пяти групп, как это показано в таблице 18.3.

Таблица 18.3. Группы Диффи-Хеллмана

Значение	Описание
1	Группа возведения в степень по модулю с модулем 768 битов
2	Группа возведения в степень по модулю с модулем 1024 бита
3	Группа эллиптической кривой с размером поля 155 битов
4	Группа эллиптической кривой с размером поля 185 битов
5	Группа возведения в степень по модулю с модулем 1680 битов

Алгоритмы хэширования, которые используются для аутентификации, показаны в таблице 18.4.

Таблица 18.4. Алгоритмы хэширования

Значение	Описание
1	MD5
2	SHA
3	Tiger
4	SHA2-256
5	SHA2-384
6	SHA2-512

Алгоритмы шифрования

Алгоритмы шифрования, которые применяются для обеспечения конфиденциальности, показаны в таблице 18.5. Все они обычно используются в режиме CBC.

Таблица 18.5. Алгоритмы шифрования

Значение	Описание
1	DES
2	IDEA
3	Blowfish
4	RC5
5	3DES
6	CAST
7	AES

18.6. ISAKMP

Протокол управления ключами и услуг безопасности в Интернете — **ISAKMP (Internet Security Association and Key Management Protocol)** — разработан для обмена сообщениями в протоколе шифрования и идентификации — (IKE).

Общий заголовок

Формат общего заголовка показан на рис. 18.29.

Рис. 18.29. Общий заголовок ISAKMP

- **Cookie инициатора.** Это поле на 32 бита определяет cookie объекта, который инициирует установление SA, уведомление SA или удаление SA.
- **Cookie респондента.** Это поле на 32 бита определяет cookie ответа респондента. Значение этого поля — 0, когда инициатор передает первое сообщение.
- **Следующая полезная нагрузка.** Это поле на 8 битов определяет тип полезной нагрузки, который следует непосредственно за заголовком. Мы обсуждаем различные типы полезной нагрузки в следующем разделе.
- **Главная версия.** Этот номер размером 4 бита определяет главную версию протокола. В настоящее время значение этого поля — 1.
- **Младший номер версии.** Этот номер размером 4 бита идет после главной версии и дополняет версию протокола. В настоящее время значение этого поля — 0.
- **Тип обмена.** Это поле на 8 битов определяет тип обмена, который осуществляется ISAKMP-пакетами. Мы обсудили различные типы обмена в предыдущем разделе.
- **Флажки.** Это — поле на 8 битов, в котором каждый бит определяет опцию обмена. Пока определены только три самых младших бита. Бит шифрования, когда он установлен на 1, указывает, что остальная часть полезной нагрузки будет зашифрована с использованием ключа шифрования и алгоритма, определенного SA. Договорный бит, когда он установлен на 1, определяет, что перед установлением SA материал шифрования не получен. Когда бит аутентификации установлен на 1, он определяет, что остальная часть полезной нагрузки хоть и не зашифрована, но аутентифицирована для сохранения целостности.
- **ID Сообщения.** Это поле на 32 бита — уникальный опознавательный код сообщения, которое определяет протокол. Это поле используется только в течение второй фазы переговоров и установлено на 0 в течение первой фазы.
- **Длина сообщения.** Поскольку к каждому пакету можно добавлять различные полезные нагрузки, длина сообщения может быть различна для каждого пакета. Это поле на 32 бита определяет длину всего сообщения, включая заголовок и все полезные нагрузки.

Полезные нагрузки

Полезные нагрузки разработаны для того, чтобы доставлять сообщения. Таблица 18.6 показывает типы полезных нагрузок.

Таблица 18.6. Типы полезных нагрузок

Типы	Название	Краткое описание
0	None (Нет)	Используется, чтобы показать конец множества полезных нагрузок
1	SA	Используется для того, чтобы запустить начало процесса переговоров
2	Proposal (предложение)	Содержит информацию, используемую в течение переговоров о SA
3	Transform (преобразовать)	Определяет секретные преобразования для создания безопасного канала
4	Key Exchange (обмен ключами)	Доставляет данные, используемые для генерации ключей
5	Identification (идентификация)	Доставляет данные идентификации в соединениях равного уровня
6	Certification (Сертификация)	Доставляет сертификат открытого ключа
7	Certification Request (Запрос сертификата)	Используется, чтобы запросить сертификат другой стороны
8	Hash (хэширование)	Доставляет данные, сгенерированные хэш-функцией
9	Signature (подпись)	Доставляет данные, сгенерированные функцией подписи
10	Nonce	Доставляет беспорядочно сгенерированные данные, такие как nonce
11	Notification (уведомление)	Доставляет сообщения об ошибках или состоянии услуг безопасности (SA)
12	Delete (удалить)	Доставляет SA, который удалил передатчик
13	Vendor (производитель)	Определяет расширения спецификации производителя

Каждая полезная нагрузка имеет типовой заголовок и некоторые заданные поля. Формат общего заголовка показан на рис. 18.30.

- **Следующая полезная нагрузка.** Это поле на 8 битов идентифицирует тип следующей полезной нагрузки. Когда такой нагрузки нет, значение этого поля — 0. Обратите внимание, что нет поля типа для текущей полезной нагрузки. Тип текущей полезной нагрузки определен предыдущей полезной нагрузкой или общим заголовком (если полезная нагрузка — первая).
- **Длина полезной нагрузки.** Это поле на 16 битов определяет длину полной полезной нагрузки (включая типовой заголовок) в байтах.

Рис. 18.30. Общий заголовок полезной нагрузки

SA полезная нагрузка

SA полезная нагрузка используется, чтобы договориться о параметрах безопасности. Однако эти параметры не включены в SA полезную нагрузку; они находятся в двух других полезных нагрузках *(proposal — предложение* и *transform — преобразование),* которые мы обсудим позже. SA полезная нагрузка сопровождается одной или несколькими *полезными нагрузками proposal (предложение)*, и каждая полезная нагрузка предложения сопровождается одной или больше *полезными нагрузками transform (преобразование)*. SA полезная нагрузка только определяет поле *домен интерпретации* и поле *ситуация*. Рисунок 18.31 показывает формат SA полезной нагрузки.

Рис. 18.31. SA полезная нагрузка

Поля в общем заголовке уже обсуждались. Описание полей SA полезной нагрузки дано ниже.

- **Домен интерпретации (DOI — Domen Interpretation).** Это — поле на 32 бита. Для фазы 1 значение 0 для этого поля определяет общий SA; значение 1 определяет IPSec.
- **Ситуация.** Это — поле переменной длины, определяющее ситуацию, в которой проводятся переговоры.

Полезная нагрузка «Предложение»

Полезная нагрузка «Предложение» (*proposal*), начинает процесс переговоров. Хотя это само по себе не задает никаких параметров, она определяет идентификацию протокола и индекс параметра обеспечения безопасности (SPI). Параметры для переговоров посылаются в составе полезной нагрузки *«Преобразование»* (*transform*), которая следует за полезной нагрузкой предложения. Каждая полезная нагрузка предложения сопровождает одну или более полезных нагрузок преобразования, которые создают альтернативные множества параметров. Рисунок 18.32 показывает формат полезной нагрузки предложения.

Поля в общем заголовке уже обсуждались. Описания других полей даны ниже.

Рис. 18.32. Полезная нагрузка «Предложение» (proposal)

- **Предложение #**. Инициатор определяет номер предложения так, чтобы респондент мог сослаться на него. Обратите внимание, что полезная нагрузка SA может включить несколько полезных нагрузок предложения. Если все предложения принадлежат одному и тому же множеству протоколов, номер предложения должен быть одним и тем же для каждого протокола во множестве. В других случаях предложения должны иметь различные номера.
- **ID Протокола.** Это поле на 8 битов определяет протокол для переговоров. Например, Фаза I IKE = 0, ESP = 1, AH = 2, и т. д.
- **Размер SPI.** Это поле на 8 битов определяет размер индекса параметра безопасности (SPI) в байтах.
- **Число преобразований.** Это поле на 8 битов определяет число полезных нагрузок преобразования, которые будут следовать за этой полезной нагрузкой предложения.
- **SPI.** Это поле переменной длины фактический SPI (см.размер SPI). Обратите внимание, что если SPI не заполняет пространство на 32 бита, заполнение не добавляется.

Полезная нагрузка «Преобразование»

Полезная нагрузка «Преобразование» фактически доставляет признаки SA переговоров. Рисунок 18.33 показывает формат полезной нагрузки «преобразование».

Поля в общем заголовке уже обсуждались. Описания других полей даны ниже.

- **Преобразование #**. Это поле на 8 битов определяет номер преобразования. Если есть больше чем одна полезная нагрузка преобразования в полезную нагрузку предложения, то каждая должна иметь свой собственный номер.
- **ID преобразования.** Это поле на 8 битов определяет идентификацию полезной нагрузки.
- **Признаки.** Каждая полезная нагрузка преобразования может доставить несколько признаков (атрибутов). Каждый атрибут непосредственно имеет три или два подполя (см. рис. 18.33). Подполе *типа признака* определяет тип признака как определенного в домене интерпретации — DOI. Подполе *длины признака*, если оно имеется, определяет значение длины признака. Поле *значения признака* — два байта в короткой форме или переменной длины в длинной форме.

Рис. 18.33. Полезная нагрузка «Преобразование»

Полезная нагрузка «обмена ключами»

Полезная нагрузка «обмена ключами» применяется при обмене сообщениями, в которых требуется передать предварительные ключи, используемые для создания ключей сеанса. Например, она может быть нужна, чтобы передать полуключ Диффи-Хеллмана. Рис. 18.34 показывает формат полезной нагрузки «обмена ключами».

Рис. 18.34. Полезная нагрузка «обмена ключами»

Поля в общем заголовке уже обсуждались. Описание поля ключа засекречивания (KE) дано ниже.

KE. Это поле переменной длины переносит данные, необходимые для того, чтобы создавать ключ сеанса.

Полезная нагрузка «Идентификация»

Полезная нагрузка «Идентификация» позволяет объектам передать свои параметры идентификации друг другу. Рисунок 18.35 показывает формат полезной нагрузки «Идентификация».

Рис. 18.35. Полезная нагрузка «Идентификация»

Поля в общем заголовке обсуждались. Описания других полей даны ниже.

- **Тип ID.** Это поле на 8 битов задает домен (область) интерпретации DOI и определяет тип используемого ID.
- **Данные ID.** Это поле на 24 бита обычно устанавливается на 0.
- **Данные идентификации.** Фактический идентификатор каждого объекта доставляется в этом поле переменной длины.

Полезная нагрузка «Сертификация»

В любое время в течение процесса обмена объект может передать свой сертификат (для открытого ключа шифрования/дешифрования или ключа подписи). Хотя включение *полезной нагрузки «Сертификация»* в процесс обмена является обычно необязательным, оно должно быть предусмотрено, если нет безопасного списка-указателя (директории), доступного для распределения сертификатов. Рисунок 18.36 показывает формат полезной нагрузки «сертификация».

Рис. 18.36. Полезная нагрузка «сертификация»

Поля в общем заголовке уже обсуждались. Описания других полей приведены ниже.

- **Сертификат кодирования.** Это поле на 8 битов определяет кодирование (тип) сертификата. Таблица 18.37 показывает типы, определенные в настоящее время.
- **Данные сертификата.** Это поле переменной длины, содержащее фактическое значение сертификата. Обратите внимание, что предыдущее поле неявно определяет размер этого поля.

Полезная нагрузка «Запрос сертификата»

Каждый объект может явно запросить сертификат от другого объекта, используя полезную нагрузку *«запроса сертификата»*. Рисунок 18.37 показывает формат такой полезной нагрузки.

Таблица18.7. Типы сертификации

Значение	Тип
0	Нет
1	Сертификат в виде X.509
2	Сертификат по алгоритму PGP
3	Ключ подписанный DNS
4	Сертификат X.509 – Подпись
5	Сертификат X.509 – Обмен ключами
6	Маркеры Цербера (Cerberus token)
7	Список аннулирования сертификатов
8	Список административного аннулирования
9	сертификат SPKI (Simple Public Key Infrastructure)
10	X.509 сертификат – Признаки

Рис. 18.37. Полезная нагрузка «Запрос сертификата»

Поля в общем заголовке уже обсуждались. Определения других полей приведено ниже.

- **Тип сертификата.** Это 8-битовое поле определяет тип сертификата в полезной нагрузке сертификата.
- **Администрация сертификата.** Это — поле переменной длины, которое определяет администрацию для выданного типа сертификата.

Полезная нагрузка «Хэширование»

Полезная нагрузка *«Хэширование»* содержит данные, сгенерированные хэш-функцией, как описано в процедуре обмена IKE. Данные хэширования гарантируют целостность сообщения или состояния. Рисунок 18.38 показывает формат полезной нагрузки «Хэширование».

Рис. 18.38. Полезная нагрузка «Хэширование»

Поля в общем заголовке уже обсуждались. Описание последнего поля дано ниже.

- **Данные хэширования.** Это — поле переменной длины, которое содержит данные хэширования, сгенерированные с применением хэш-функции к части сообщения или состояний ISAKMP.

Полезная нагрузка «Подпись»

Полезная нагрузка *«Подпись»* содержит данные, сгенерированные, с применением процедуры цифровой подписи по некоторой части сообщений или состояний ISAKMP. Рисунок 18.39 показывает формат полезной нагрузки «Подпись».

Рис. 18.39. Полезная нагрузки «Подпись»

Поля в общем заголовке уже обсуждались. Описание последнего поля дано ниже.

- **Подпись.** Это поле переменной длины содержит дайджест, следующий из применения подписи к части сообщений или состояний ISAKMP.

Полезная нагрузка Nonce

Полезная нагрузка *Nonce* содержит случайные данные для использования nonce, чтобы обеспечить живучесть сообщения и предотвратить атаку воспроизведения. Рисунок 18.40 показывает формат полезной нагрузки nonce.

Рис. 18.40. Полезная нагрузка Nonce

Поля в общем заголовке уже обсуждались. Описание последнего поля дано ниже.

- **Nonce.** Это поле переменной длины содержит значения nonce.

Полезная нагрузка «Уведомление»

В течение процесса переговоров иногда одна сторона должна сообщить другой стороне о состоянии или об ошибках. Полезная нагрузка *«Уведомление»* разра-

ботана для этих двух целей. Рисунок 18.41 показывает формат полезной нагрузки «Уведомление».

Рис. 18.41. Полезная нагрузка «Уведомление»

Поля в общем заголовке уже обсуждены. Описания других полей даны ниже.

- **DOI.** Это поле на 32 бита — то же самое, что определено для полезной нагрузки услуг обеспечения безопасности (SA).
- **ID протокола.** Это поле на 8 битов — то же самое, что определено для полезной нагрузки «Предложение».
- **SPI размер.** Это поле на 8 битов — то же самое, что определено для полезной нагрузки «Предложение».
- **Тип сообщения «Уведомление».** Это поле на 16 битов определяет состояние или тип ошибки, о которой нужно передать сообщение. Таблица 18.8 дает краткое описание этих типов.
- **SPI.** Это поле переменной длины — такое же, как определено для полезной нагрузки «Предложение».
- **Данные уведомления.** Это поле переменной длины может доставить дополнительное текстовое сообщение о состоянии или ошибках. Типы ошибок перечислены в таблице 18.8. Значения 31 до 8191 зарезервированы для будущего использования и значения от 8192 до 16383 — для частного применения.

Таблица 18.8 Типы уведомления

Значение	Описание	Описание (рус.)
1	INVALID-PAYLOAD-TYPE	Недопустимый тип полезной нагрузки
2	DOI-NOT-SUPPORTED	Не поддерживается
3	SITUATION-NOT-SUPPORTED	Ситуация не поддерживается
4	INVALID-COOKIE	Недопустимое cookie
5	INVALID-MAJOR-VERSION	Недопустимая главная версия
6	INVALID-MINOR-VERSION	Недопустимый младший номер версии

7	INVALID-EXCHANGE-TYPE	Недопустимый тип обмена
Значение	**Описание**	**Описание (рус.)**
8	INVALID-FLAGS	Недопустимые флажки
9	INVALID-MESSAGE-ID	Недопустимый ID сообщения
10	INVALID-PROTOCOL-ID	Недопустимый ID протокола
11	INVALID-SPI	Недопустимый SPI
12	INVALID-TRANSFORM-ID	Недопустимый ID преобразования
13	ATTRIBUTE-NOT-SUPPORTED	Атрибут не поддерживается
14	NO-PROPOSAL-CHOSEN	Предложение не выбрано
15	BAD PROPOSAL-SYNTAX	Плохой синтаксис предложения
16	PAYLOAD-MALFORMED	Неправильно сформированная полезная нагрузка
17	INVALID-KEY-INFORMATION	Недопустимая информация ключа
18	INVALID-ID-INFORMATION	Недопустимая информация ID
19	INVALID-CERT-ENCODING	Недопустимое шифрование сертификата
20	INVALID-CERTIFICATE	Недопустимый сертификат
21	CERT-TYPE-UNSUPPORTED	Неподдерживаемый тип сертификата
22	INVALID-CERT-AUTHORITY	Недопустимая администрация сертификата
23	INVALID-HASH-INFORMATION	Недопустимая информация хэширования
24	AUTHENTICATION-FAILED	Ошибочная аутентификация
25	INVALID-SIGNATURE	Недопустимая подпись
26	ADDRESS-NOTIFICATION	Уведомление адреса
27	NOTIFY-SA-LIFETIME	Уведомление о времени жизни SA
28	CERTIFICATE-UNAVAILABLE	Сертификат недоступен
29	UNSUPPORTED EXCHANGE-TYPE	Неподдерживаемый тип обмена
30	UNEQUAL-PAYLOAD-LENGTHS	Несоответствующая длина полезной нагрузки

Таблица 18.9 содержит список уведомлений состояния. Значения от 16385 до 24575 и от 40960 до 65535 зарезервированы для будущего использования, значения от 32768 до 40959 — для частного применения.

Таблица 18.9. Значения уведомлений состояния

Значение	**Описание**
16384	Подключено
24576-32767	DOI — заданные коды интерпретации домена

Полезная нагрузка «Удаление»

Полезная нагрузка «Удаление» используется объектом, который удалил один или более SA's и должен сообщить равным по уровню объектам, что он эти

SA's больше не поддерживает. Рисунок 18.42 показывает формат полезной нагрузки «удаление».

Рис. 18.42. Полезная нагрузка «Удаление»

Поля в общем заголовке уже были рассмотрены. Описания других полей приводятся ниже.

- **DOI.** Это поле на 32 бита — то же самое, что определено для полезной нагрузки SA (услуг обеспечения безопасности).
- **ID Протокола.** Это поле на 8 битов — то же самое, что определено для полезной нагрузки предложения.
- **SPI размер.** Это поле на 8 битов — то же самое, что определено для полезной нагрузки предложения.
- **Номер SPI's.** Это поле на 16 битов определяет номер SPI's. Каждый, кто удаляет полезную нагрузку, может известить об удалении нескольких SAs.
- **SPIs.** Это поле переменной длины определяет SPI's, удаленные SA's.

Полезная нагрузка «Поставщик»

ISAKMP позволяет обмен информацией, учитывающей особенности данного поставщика. Рисунок 18.43 показывает формат *полезной нагрузки «Поставщик»*.

Рис. 18.43. Полезная нагрузка «Поставщик»

Поля в типовом заголовке уже обсуждались. Описание последнего поля дано ниже.

- **ID поставщика.** Это поле переменной длины определяет константу, используемую поставщиком.

18.7. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце.

Книги

[DH03], [Fra0l], [KPS02], [Res0l], [Sta06], [Rhe03] полностью рассматривают IPSec.

Сайты

Нижеследующие сайты дают больше информации о темах, обсужденных в этой лекции.

http://www.unixwiz.net/techtips/iguide-ipsec.html
http://www.ietf.org/rfc/rfc2401.txt
http://rfc.net/rfc240l.html

18.8. Итоги

- Безопасность IP (IPSec) — совокупность протоколов, разработанных IETF (Группа Инженерной поддержки сети Интернет) для того, чтобы обеспечить безопасность передачи пакетов на сетевом уровне.
- IPSec работает в транспортном или туннельном режиме. В транспортном режиме IPSec защищает информацию, доставляемую от транспортного уровня к сетевому уровню, но не защищает заголовок IP. В туннельном режиме IPSec защищает весь пакет IP, включая первоначальный заголовок IP.
- IPSec определяет два протокола: протокол заголовка аутентификации (AH) и полезную нагрузку со встроенной защитой нагрузка (ESP). Эти протоколы обеспечивают аутентификацию, шифрование или и то и другое для пакетов на уровне IP. Протокол заголовка аутентификации (AH) подтверждает подлинность хоста источника и гарантирует целостность полезной нагрузки, которую несет пакет IP. Протокол «полезная нагрузка со встроенной защитой» (ESP) обеспечивает исходную аутентификацию, целостность и секретность. ESP добавляет к формату заголовок и конечную метку.
- IPSec косвенно обеспечивает управление доступом, используя базу данных услуг обеспечения безопасности (SAD).
- В IPSec стратегия безопасности (SP) определяет, какая безопасность должна быть обеспечена пакету в передатчике или в приемнике. IPSec использует множество стратегий безопасности, называемых базой данных стратегии безопасности (SPD).
- Смена ключей в Интернете (IKE) — протокол, предназначенный для создания услуг обеспечения безопасности (SA) для входящих и исходящих соединений. IKE создает SA's для IPSec.

- IKE — сложный протокол, который базируется на трех других протоколах: Oakley, SKEME и ISAKMP.
- IKE работает в двух фазах: фаза I и фаза II. Фаза I создает SA's для фазы II; фаза II создает SA's для протокола обмена данными, такого как IPSec.
- ISAKMP-протокол разработан для того, чтобы доставить сообщение для протокола IKE.

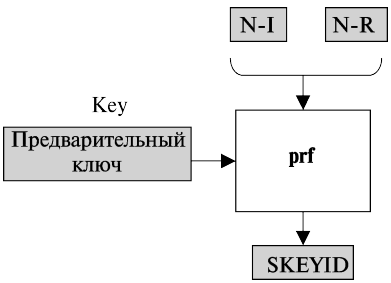
18.9. Набор для практики

Обзорные вопросы

1. Покажите различия между двумя режимами IPSec.
2. Определите протокол AH и услуги безопасности, которые он обеспечивает.
3. Определите протокол ESP и услуги безопасности, которые он обеспечивает.
4. Определите услуги обеспечения безопасности (SA) и объясните их цель.
5. Определите SAD и объясните его отношение к услугам обеспечения безопасности.
6. Определите стратегию безопасности и объясните ее цель в отношении к IPSec.
7. Определите IKE и объясните, почему этот протокол необходим в IPSec.
8. Определите фазы IKE и цели каждой фазы.
9. Определите ISAKMP и его отношение к IKE.
10. Перечислите типы полезной нагрузки ISAKMP и цель каждого типа.

Упражнения

1. Хост получает аутентифицированный пакет с порядковым номером 181. Окно ответа имеет промежуток от 200 до 263. Что хост сделает с пакетом? Каков будет промежуток окна после этого события?
2. Хост получает аутентифицированный пакет с порядковым номером 208. Окно ответа имеет промежуток от 200 до 263. Что хост сделает с пакетом? Каков будет промежуток окна после этого события?



3. Хост получает аутентифицированный пакет с порядковым номером 331. Окно ответа имеет промежуток от 200 до 263. Что хост сделает с пакетом? Каков будет промежуток окна после этого события?
4. Диаграмма для вычисления SKEYID при методе предварительного совместного ключа показана на рис. 18.44. Обратите внимание, что ключ к функции prf в этом случае — предварительный совместный ключ.

Рис. 18.44. Упражнение 14

   а. нарисуйте подобную диаграмму SKEYID для метода открытого ключа.
   б. нарисуйте подобную диаграмму SKEYID для метода цифровой подписи.

5. Нарисуйте диаграмму, подобную рис. 18.44, для приводимых ниже случаев; ключ в каждом случае — SKEYID.
   a. SKEYID_a
   b. SKEYID_d
   c. SKEYID_e
6. Нарисуйте диаграмму, подобную рис. 18.44, для приводимых ниже случаев, ключ в каждом случае — SKEYID.
   а. HASH-1
   б. HASH-R
7. Нарисуйте диаграмму, подобную рис. 18.44, для следующего случая; ключ в каждом случае — SKEYID_d.
   а. HASH 1I
   б. HASH2
   в. HASH3
8. Нарисуйте диаграмму, подобную рис. 18.44, для следующего случая; ключ в каждом случае — SKEYID _d.
   а. K для случая без PFS
   б. K для случая с PFS
9. Повторите упражнение для случая, в котором длина K — слишком мала.
10. Начертите диаграмму и покажите ISAKMP-пакеты, которыми обменялись инициатор и респондент, использующие метод *предварительного совместного ключа* в *главном режиме* (см. рис. 18.20). Используйте по крайней мере два пакета предложения с двумя пакетами преобразования для каждого предложения.
11. Повторите упражнение 10, используя *метод первоначального открытого ключа* в *главном режиме* (см. рис. 18.21).
12. Повторите упражнение 10, используя *пересмотренный* метод *открытого ключа* в *главном режиме* (см. рис. 18.22).
13. Повторите упражнение 10, используя метод *цифровой подписи* в *главном режиме* (см. рис. 18.23).
14. Повторите упражнение 10 в *энергичном режиме* (см. рис. 18.24).
15. Повторите упражнение 11 в *энергичном режиме* (см. рис. 18.25).
16. Повторите упражнение 12 в *энергичном режиме* (см. рис. 18.26).
17. Повторить упражнение 13 в *энергичном режиме* (см. рис. 18.27).

18. Нарисуйте диаграмму и покажите фактические ISAKMP-пакеты, которыми обменялись инициатор и респондент в *быстром режиме* (см. рис. 18.28).
19. Сравните методы предварительного совместного ключа в *главном* и *энергичном* режимах. Что сделано в *энергичном* режиме для безопасности? Каково увеличение эффективности?
20. Сравните общие методы открытого ключа в *главном* и *энергичном* режимах. Что сделано в агрессивном режиме относительно безопасности? Каково увеличение эффективности?
21. Сравните пересмотренные методы открытого ключа в *главном* и *энергичном* режимах. Что сделано в агрессивном режиме относительно безопасности? Каково увеличение эффективности?
22. Сравните метод цифровой подписи в *главном* и *энергичном* режимах. Что сделано в энергичном режиме относительно безопасности? Каково увеличение эффективности?
23. В главном и энергичном режиме — мы предполагаем, что злоумышленник не может вычислить SKEYID. Приведите доводы в пользу этого предположения.
24. В фазе I IKE идентификатор обычно определяется как адрес IP. В предварительном общедоступном методе предварительный совместный ключ — также функция адреса IP. Покажите, как это может создать порочный круг.
25. Сравните методы для главного режима и покажите, какой метод позволяет обменяться защищенными ID.
26. Повторите упражнение для энергичных методов.
27. Покажите, как IKE реагирует на атаку воспроизведения в главном режиме, — то есть покажите, как IKE отвечает нападающему, который пытается воспроизвести одно или более сообщений в главном режиме
28. Покажите, как IKE реагирует на атаку воспроизведения в энергичном режиме, — то есть покажите, как IKE отвечает нападающему, который пытается воспроизвести одно или более сообщений в энергичном режиме.
29. Покажите, как IKE реагирует на атаку воспроизведения в быстром режиме, — то есть покажите, как IKE отвечает нападающему, который пытается воспроизвести одно или более сообщений в быстром режиме.
30. Покажите, как IPSec реагирует на атаку грубой силы. Если злоумышленник может сделать исчерпывающий компьютерный поиск, сможет ли он найти ключ шифрования для IPSec?

Приложение A. ASCII

Американский Стандартный Код для Информационного Обмена (ASCII American Standard Code for Information Interchange) — код на 7 битов, который был разработан, чтобы обеспечить коды для 128 символов, как это показано в таблице А.1.

Таблица А.1. Коды ASCII

Hex	Символ	Hex	Символ	Hex	Символ	Hex	Символ	Hex	Символ	Hex	Символ
00	null	18	CAN	30	0	48	H	60	'	78	x
01	SOH	19	EM	31	1	49	I	61	a	79	y
02	STX	1A	SUB	32	2	4A	J	62	b	7A	z
03	ETX	1B	ESC	33	3	4B	K	63	c	7B	{
04	EOT	1C	FS	34	4	4C	L	64	d	7C	|
05	ENQ	1D	GS	35	5	4D	M	65	e	7D	}
06	ACK	1E	RS	36	6	4E	N	66	f	7E	~
07	BEL	1F	US	37	7	4F	0	67	g	7F	DEL
08	BS	20	SP	38	8	50	P	68	h		
09	HT	21	!	39	9	51	Q	69	i		
OA	LF	22	"	3A	:	52	R	6A	j		
OB	VT	23	#	3B	;	53	S	6B	k		
OC	FF	24	$	3C	<	54	T	6C	1		
OD	CR	25	%	3D	=	55	U	6D	m		
OE	SO	26	&	3E	>	56	V	6E	n		
OF	SI	27	'	3F	?	57	W	6F	o		
10	DLE	28	(	40	@	58	X	70	p		
11	DC1	29	)	41	A	59	Y	71	q		
12	DC 2	2A	*	42	B	5A	Z	72	r		
13	DC 3	2B	+	43	C	5B	[	73	s		
14	DC 4	2C	,	44	D	5C	\	74	t		
15	NAK	2D	-	45	E	5D	]	75	u		
16	SYN	2E	o	46	F	5E	^	76	v		
17	ETB	2F	/	47	G	5F	-	77	w		

Hex – шестнадцатеричное значение.

Приложение В. Стандарты и организации по стандартизации

Стандарты являются основными инструментами в создании и поддержании открытого и конкурентоспособного рынка для изготовителей оборудования и гарантией обеспечения национальной и международной способности к взаимодействию технологий. Стандарты обеспечивают рекомендации изготовителям, поставщикам, правительственным агентствам и другим поставщикам услуг, а также гарантию взаимосвязанности, необходимой для сегодняшнего рынка и в международных связях.

В.1. Стандарты интернета

Стандарт Интернета — полностью проверенная жизнью спецификация, которой полезно и необходимо придерживаться тем, кто работает с Интернетом. Это формализованное регулирование правил, сопровождающее все области работы с Интернетом. Есть строгая процедура, в соответствии с которой спецификация достигает состояния стандарта Интернета. Спецификация начинается как эскиз Интернета. **Эскиз (черновик) Интернета** — рабочий документ (работа в процессе выполнения). Он не имеет официального статуса без официального состояния и удаляется из базы данных после шестимесячного времени жизни. По рекомендации администрации сети Интернет эскиз может быть издан как **Запрос о комментарии (запрос на комментарии).** Каждый запрос на комментарий редактируется, получает назначенный номер и делается доступным для всех заинтересованных сторон. Запросы на комментарии проходят уровни готовности и разбиты на категории согласно их уровню и требованиям каждого уровня готовности.

Уровни готовности

Запрос на комментарии в течение его времени жизни относят к одному из шести **уровней готовности:** предложенный стандарт, эскиз стандарта, стандарт Интернета, исторический, экспериментальный и информационный, как показано на рис. В.1.

Предложенный стандарт

Предложенный стандарт — это спецификация, которая хорошо понятна и вызывает достаточный и стабильный интерес сообществ Интернета. На этом уровне спецификация обычно проверяется и реализуется несколькими различными группами.

Черновой стандарт

После по крайней мере двух успешных независимых и совместимых реализаций предложенный стандарт переходит на более высокий уровень чернового стандарта. Если не обнаруживаются трудности и не возникает проблем, черновой стандарт с определенными модификациями обычно становится стандартом Интернета.

Рис. В.1. Уровни готовности RFC

Стандарт Интернета

Черновой стандарт после демонстрации успешной реализации становится стандартом Интернета.

Исторические

Исторические запросы на комментарии существенны для исторической перспективы. Если старые версии стандартов были заменены новыми, или вышли из употребления, или никогда не проходили необходимые уровни готовности, они приобретают статус исторических RFC (запросов на комментарии).

Экспериментальные

Запрос на комментарии классифицируют как экспериментальный, если он содержит сведения об экспериментальных исследованиях, интересных для Интернет-сообщества.

Информационные

Запрос на комментарии классифицируют как информационный, если он содержит документы, которые являются не стандартами или любыми другими несогласованными документами. Они могут быть обучающими программами, связанными с Интернет.

Уровни требований

Запросы на комментарии разделены на пять **уровней требований**: требуемые, рекомендуемые, выбранные, ограниченного использования и нерекомендованные, как это показано на рис. В.2.

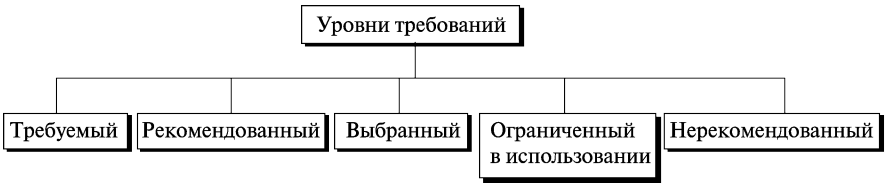


Рис. В.2. Уровни требований RFC

Требуемый

Запрос на комментарии (RFC) помечен как *требуемый*, если он реализован всеми системами Интернета, чтобы достигнуть между ними минимального соответствия.

Рекомендованный

Запрос на комментарии (RFC) помечается как *рекомендованный*, если не требуется минимального соответствия между системами; его рекомендуют из-за его полезности в некоторых системах.

Выбираемый

Запрос на комментарии (RFC) помечается как *выбираемый*, если он не требуемый и не рекомендованный. Однако система может использовать его для собственных целей.

Ограниченное использование

Запрос на комментарии (RFC) помечен как *ограниченный для использования*, если он должен применяться только в ограниченных ситуациях. Большинство экспериментальных запросов на комментарии подпадает под эту категорию.

Нерекомендованный

Запрос на комментарии, помеченный как *нерекомендованный*, является не соответствующим для общего использования. Обычно под эту категорию подпадает исторический (устаревший) запрос на комментарии.

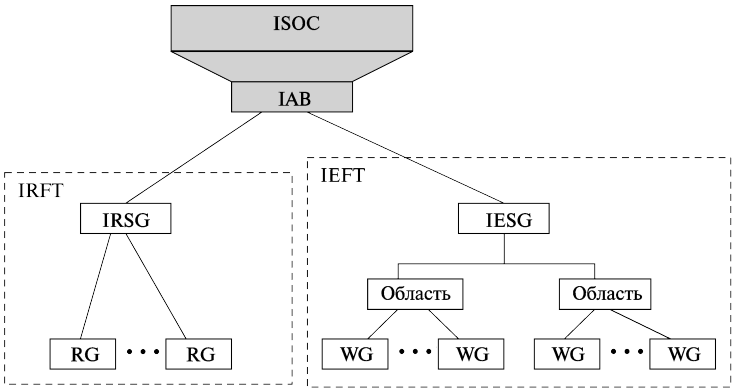
Запросы на комментарии могут быть найдены на сайте www.faqs.org/rfcs

Администрация сети Интернет

Сеть Интернет начиналась прежде всего как исследовательская работа. После этого она быстро развивалась и получила более широкое применение в коммерческой деятельности. Различные группы, которые координируют проблемы Интернета, определяли и определяют этот рост и развитие. Рисунок В.3 показывает общую организацию администрации сети Интернет.

Сообщество пользователей сети Интернет (ISOC)

Сообщество Пользователей сети Интернет (ISOC — Internet Society) — международная некоммерческая организация, созданная в 1992 году, чтобы обеспечить поддержку процессу разработки стандартов сети Интернет. ISOC выполняет функцию путем управления и контроля за другими административными единицами, такими как IAB, (IETF, IRTF и ICANN (см. следующие разделы). ISOC курирует исследования по теме Интернета, а также содействует образованию, касающемуся Интернет.



ISOC	Internet Society	Сообщество Пользователей Интернет
IAB	Internet Architecture Board	Совет по архитектуре Интернет
IRFT	Internet Research Task Force	Исследовательская группа сети Интернет
IEFT	Internet Engineering Task Force	Инженерная группа сети Интернет
IRSG	Internet Research Steering Group	Группа руководства исследованиями в сети Интернет
IESG	Internet Engineering Steering Group	Инженерная группа по разработке стандартов Интернет
RG	Research Group	Исследовательская группа
WG	Working Group	Рабочая группа

Рис. В.3. Администрация сети Интернет

Совет по архитектуре сети Интернет (IAB)

Совет по архитектуре сети Интернет (IAB — Internet Architecture Board) — технический советник ISOC. Главные цели IAB состоят в том, чтобы наблюдать за непрерывным развитием протокола TCP/IP, собирать сведения и служить техническим консультантом для исследований, проводимых членами Интернет-сообщества. IAB выполняет эти функции, используя два первичных компонента: Инженерную группу сети Интернет (IEFT) и Исследовательскую группу сети Интернет (IRTF) . Другая область ответственности IAB — редактирование запросов на комментарии (RFT), рассмотренных ранее в этом приложении. IAB также осуществляет внешние связи между администрацией и другими организациями стандартизации и форумами.

Инженерная группа сети Интернет *(IETF)*

Инженерная группа сети Интернет (IETF — Internet Engineering Task Force) — форум рабочих групп, управляемых Инженерной группой по разработке стандар-

тов сети Интернет (IESG — Internet Engineering Steering Group). IETF отвечает за определение рабочих проблем и разработку предложений по решению этих проблем. IETF также предлагает и рассматривает спецификации, предназначенные для разработки стандартов Интернет. Рабочие группы собраны по областям проблем, и каждая концентрируется на заданной теме. В настоящее время были определены девять областей проблем: вопросы разработки приложений, протоколы сети Интернет, маршрутизация, процессы функционирования, пользовательские услуги, управление сетью, транспортные вопросы сети, следующее поколение протокола сети Интернет (IPng) и безопасность.

Исследовательская группа сети Интернет (IRTF)

Исследовательская группа сети Интернет (Internet Research Task Force) — форум рабочих групп, управляемых Группой руководства исследованиями в сети Интернет (IRSG — Internet Research Steering Group). IRTF сосредоточивается на долгосрочных исследовательских темах, связанных с сетевыми протоколами Интернет, приложениями, архитектурой и технологией.

Корпорация Интернет для назначения названий и номеров (ICANN)

Корпорация Интернет для назначения названий и номеров **(ICANN — Internet Corporation for Assigned Names and Numbers)** — частная некоммерческая корпорация, управляемая международным советом, которая несет ответственность за управление именами доменов Интернета и адресами.

Центр сетевой информации (NIC)

Центр сетевой информации (NIC — Network Information Center) несет ответственность за распределение информации о протоколах TCP/IP.

В.2. Другие организации по стандартизации

Рассмотрим информацию о нескольких других организациях стандартизации, которые упомянуты в тексте.

NIST

Национальный Институт Стандартов и Технологии (NIST) — часть Отдела Торговли Соединенных Штатов. NIST вырабатывает стандарты в форме Федерального Стандарта Обработки Информации (FIPS — Federal Information Processing Standard).

Это процесс проходит следующими шагами.

1. NIST издает Федеральный Стандарт Обработки Информации (FIPS) в *Федеральном Регистре* (правительственное издание) и сайт NIST для общедоступного обзора и комментариев. Объявление также определяет крайний срок приёма комментариев (обычно спустя 90 дней после объявления).
2. После окончания срока группа экспертов в NIST рассматривает комментарии и делает любые необходимые изменения.
3. Рекомендованный Федеральный Стандарт Обработки Информации (FIPS) передают министру торговли для одобрения.

4. Одобренный Федеральный Стандарт Обработки Информации (FIPS) издается в *Федеральном Регистре* и на сайте NIST.

Международная Организация по Стандартизации (ISO)

Международная Организация по Стандартизации (Международная Организация по Стандартизации) — многонациональная группа, членство в которой независимо, главным образом от комитетов создания стандартов различных правительств во всем мире. Международная Организация по Стандартизации работает над развитием сотрудничества в областях научной, технологической и экономической деятельности.

ITU-T (International Telecommunication Union — Telecommunication Standards Sector)

Международный Союз по Телекоммуникациям — Сектор Телекоммуникационных Стандартов (ITU-T) — часть международного Телекоммуникационного Союза (ITU). Сектор посвящен исследованию и установлению стандартов для телекоммуникаций, в частности, для систем передачи данных и телефонной связи.

ANSI

Американский национальный Институт Стандартов (ANSI — American National Standards Institute) — полностью частная, некоммерческая корпорация, не связанная с американским федеральным правительством. Однако все действия ANSI предпринимаются в первую очередь с целью улучшения благосостояния Соединенных Штатов и его граждан.

IEEE (ИИЭР)

Институт инженеров по электротехнике и радиоэлектронике (ИИЭР) — самое крупное профессиональное техническое общество в мире. Действуя в международном масштабе, оно стремится продвигать теорию, развивать творческий потенциал и следить за качеством продуктов в областях электротехники, электроники и радио, так же как во всех других связанных с ними ветвях разработки. Одна из его целей — наблюдение за развитием и принятием международных стандартов в области компьютеров и связи.

EIA

Электронная Ассоциация Отраслей промышленности (EIA) — подобная ANSI некоммерческая организация, которая оказывает поддержку компаниям в промышленности, производящей электронику. Её действия в дополнение к развитию стандартов включают общедоступное образование и лоббирование. В области информационной технологии EIA внесла существенный вклад в развитие стандартов для передачи данных.

Приложение C. Набор протоколов TCP/IP

Модель организации сети, используемая в сети Интернет сегодня, — **протокол управления передачей / протокол взаимодействия сетей (TCP/IP)**, или **набор протокола TCP/IP.** Как показано на рис. C.1, набор состоит из пяти уровней: *прикладной, транспортный, сетевой, звена данных (канальный)* и *физический.*

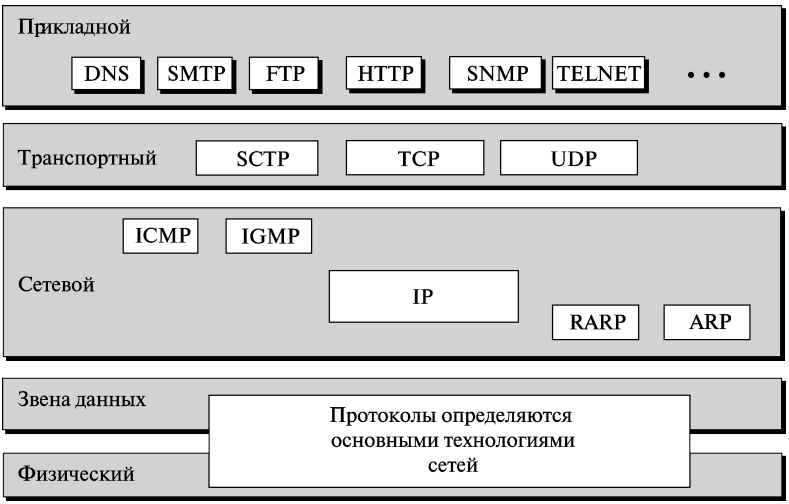


Рис. C.1. Модель протоколов TCP/IP

TCP/IP — иерархический протокол, составленный из интерактивных модулей, каждый из которых обеспечивает функциональные возможности. Термин *иерархический* означает, что каждый протокол верхнего уровня использует услуги одного или нескольких протоколов более низкого уровня.

C.1. Уровни в TCP/IP

В этом разделе мы кратко описываем функции каждого уровня в наборе протокола TCP/IP.

Прикладной уровень

Прикладной уровень позволяет пользователю (или человеку, или программному обеспечению) обращаться к сети. Он обеспечивает интерфейсы пользователя и поддержку услуг, таких как передача файлов, электронная почта и работа с удаленным сервером.

Прикладной уровень отвечает за обеспечение услуг пользователю.

- **Доменная система имен (DNS — Domain Name System**). Доменная система имен — прикладная программа, которая предоставляет услуги другим прикладным программам. Она находит логический адрес (сетевой уровень) по заданному адресу пользователя (прикладной уровень).
- **Простой протокол передачи почты (SMTP — Simple Mail Transfer Protocol).** SMTP — протокол, используемый для электронной почты. Электронная почта рассматривалась в лекции 16.
- **Протокол передачи файлов (FTP — File Transfer Protocol).** FTP — протокол передачи файлов в сети Интернет. Он применяется для того, чтобы передать большие файлы от одного компьютера к другому.
- **Протокол передачи гипертекста (HTTP — Hypertext Transfer Protocol).** HTTP — протокол, который обычно используется, чтобы обратиться к Word Wide Web (Всемирная Паутина).
- **Простой протокол управления сетью (SNMP — Simple Network Management Protocol**). SNMP — официальный протокол управления в Интернет.
- **Оконечная сеть (TELNET — Terminal Network).** TELNET — удаленная прикладная программа входа в систему. Пользователь может использовать TELNET, чтобы соединиться с удаленным хостом и применять доступные услуги.

Транспортный уровень

Транспортный уровень отвечает за доставку всего сообщения «от процесса-к-процессу». Процесс — прикладная программа, функционирующая на хосте.

Транспортный уровень отвечает за доставку сообщения от одного процесса до другого.

Традиционно транспортный уровень был представлен в TCP/IP двумя протоколами: TCP и UDP. Новый протокол транспортного уровня SCTRP (Stream Control Transmission Protocol) был разработан в ответ на новые потребности некоторых вновь созданных приложений (например, систем реального времени).

- **Протокол пользовательских дейтаграмм (UDP).** UDP — наиболее простой из двух стандартных транспортных протоколов TCP/IP. Это протокол типа «процесс-процесс», который добавляет к данным верхнего уровня только адреса порта, контрольную сумму для контроля ошибок и информацию длины.
- **Протокол управления передачей (TCP).** TCP обеспечивает приложениям полный набор услуг транспортного уровня. TCP — достоверный протокол транспортного потока. Термин *протокол потока* в этом контексте означает средства, ориентированные на соединение: соединение должно быть установлено между обоими участниками обмена прежде, чем любой из них сможет передать данные. На передающем конце в процессе передачи TCP делит поток данных на меньшие модули, называемые *сегментами*. Каждый сегмент включает в себя порядковый номер, чтобы после приема установить

порядок следования сегментов, вместе с номерами подтверждения для полученных сегментов. Сегменты переносят через сеть Интернет информацию в виде дейтаграмм IP. В конце приема TCP собирает из сегментов каждую дейтаграмму и упорядочивает их следование на основе порядковых номеров.

- **Транспортный протокол управления потоком (SCTP — STREAM CONTROL TRANSMISSION PROTOCOL).** SCTP обеспечивает поддержку новым приложениям, таким как IP-телефония. Это — протокол транспортного уровня, который объединяет положительные свойства UDP и TCP.

Сетевой уровень

Сетевой уровень предназначен для доставки пакетов от источника в пункт назначения, вероятно через множество физических сетей (линий связи). Сетевой уровень гарантирует, что каждый пакет будет доставлен от его исходной точки к его конечному пункту назначения. Некоторые обязанности сетевого уровня включают логическую адресацию и маршрутизацию.

Сетевой уровень предназначен для доставки отдельных пакетов от хоста источника до хоста пункта назначения.

- **Internet-протокол (IP).** IP — механизм передачи, используемый в соответствии с протоколами TCP/IP. Это ненадежное без установления соединения обслуживание с *лучшими намерениями*. Термин «*с лучшими намерениями*» означает, что IP не обеспечивает проверки ошибок или выбор оптимального маршрута. IP учитывает ненадежность основных уровней и «прилагает все усилия», чтобы передать информацию к пункту назначения, но без гарантий. IP транспортирует данные в пакетах, называемых *дейтаграммами*, каждая из которых транспортируется отдельно. Дейтаграммы могут перемещаться по различным маршрутам и могут прибыть не в исходной последовательности или оказаться продублированными. IP не сохраняет порядок и список маршрутов и не имеет никаких средств для того, чтобы исправить дейтаграммы, однажды прибывшие в пункт назначения. Ограниченные функциональные возможности IP нельзя рассматривать как слабость. IP обеспечивает только функции передачи. Пользователь может добавить те средства, которые необходимы для данного приложения, и таким образом обеспечить максимальную эффективность.
- **Протокол определения адресов (ARP — Address Resolution Protocol).** Протокол определения адресов используется, чтобы связать IP-адрес с физическим адресом. На обычной физической сети каждое устройство идентифицировано физическим адресом или адресом станции, обычно закрепленным на сетевой карте интерфейса (NIC — Network Interface Card). Протокол определения адресов используется, чтобы найти физический адрес узла, когда известен его Интернет-адрес.
- **Протокол определения адресов обратного адреса (RARP — Reverse Address Resolution Protocol)** позволяет хосту обнаруживать его адрес в сети Интернет, когда он знает только свой физический адрес. Это используется, ког-

да компьютер связывается с сетью впервые или когда компьютер загружается без диска.

- **Протокол Управляющих сообщений сети Интернет (ICMP – Internet Control Message Protocol).** ICMP – механизм, используемый хостами и другими промежуточными устройствами, чтобы передать уведомление о дейтаграммных проблемах назад передатчику. ICMP передает запрос и ошибку, извещая о сообщениях.
- **Протокол управления группами сети Интернет (IGMP — Internet Group Message Protocol).** Протокол управления группами (пользователей) сети Интернет применяется для того, чтобы облегчить одновременную передачу сообщения группе получателей.

Уровень звена передачи данных

Уровень звена передачи данных преобразовывает физический уровень, простое средство передачи, — в достоверную линию связи. Это дает возможность сделать физический уровень свободным от ошибок (виртуально) при передаче информации к верхнему уровню (сетевой уровень). Некоторые из задач уровня звена передачи данных — цикловая синхронизация, физическая адресация, управление потоком, контроль ошибок и управление доступом.

Уровень звена передачи данных предназначен для того, чтобы перемещать кадры от одного участка (узла) к следующему.

Физический уровень

Физический уровень координирует функции доставки потока бит по физической среде. Физический уровень определяет: физические характеристики интерфейсов и характеристики среды передачи, представление битов, скорость передачи данных, синхронизацию битов и физическую топологию сети.

Физический уровень предназначен для передачи отдельных битов от одного участка (узла) к следующему.

С.2. Адресация

В Интернете используются четыре различных уровня адресов, предназначенные для работы в TCP/IP: **заданный адрес**, **адрес порта**, **логический адрес** и **физический адрес**, как это показано на рис. С.2.

Заданный адрес

Связь на прикладном уровне осуществляется с применением заданных адресов: адресов, принадлежащих заданным протоколам прикладного уровня. Например, используется один адрес электронной почты, чтобы передать электронную почту.

Адрес порта

Сегодня компьютеры — устройства, которые могут выполнять одновременно много задач. Конечная цель связи в Интернете — процесс, устанавливающий связь с другим процессом. Например, компьютер А работает с компьютером С, используя TELNET. В то же самое время компьютер обменивается сообщениями с компьютером В, пользуясь протоколом передачи файлов (FTP). Чтобы этих процессы могли функционировать одновременно, должен быть метод отметки различных процессов. Другим словами, процессы нуждаются в адресации. В архитектуре TCP/IP метка, назначаемая процессу, называется адресом порта. Адрес порта в TCP/IP — 16 битов длиной.

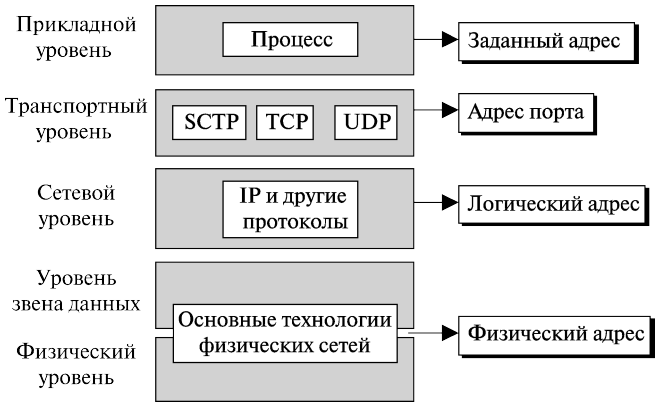


Рис. С.2. Адреса в TCP/IP

Логический адрес

Логические адреса необходимы для универсальных служб связи, которые не зависят от основных физических сетей. Нужна универсальная система адресации, в которой каждый хост может быть идентифицирован уникально, независимо от основной физической сети. Логические адреса разработаны для этой цели. Логический адрес (адрес IP) в Интернете — в настоящее время это адрес на 32 бита, который может уникально определить хост, подключенный к Интернету. Никакие два хоста, получившие адреса и работающие в Интернете, не могут иметь один и тот же адрес IP.

Физический адрес

Физический адрес, также известный как адрес связи, является адресом узла, определенного в данной физической сети. Он включен в кадр, используемый уровнем звена передачи данных. Это — адрес самого низкого уровня. Физические адреса управляются администрацией по физической сети. Размер и формат этих адресов изменяются в зависимости от сети.

Приложение D. Элементарная теория вероятностей

Теория вероятностей играет очень важную роль в криптографии, потому что она обеспечивает лучший способ количественно определить степень неопределенности, а криптография в большинстве случаев основана на неопределенности. Это приложение рассматривает основные концепции теории вероятностей, которые необходимы, чтобы понять некоторые темы, рассмотренные в этой книге.

D.1. Введение

Мы начинаем с некоторых определений, аксиом и свойств.

Определения

Случайный эксперимент

Эксперимент может быть определен как любое действие, которое на некоторые изменения на входе отвечает изменениями на выходе. **Случайный эксперимент** — эксперимент, в котором одно и то же значение на входе может дать различные значения на выходе. Другими словами, выход не может быть однозначно определен значением на входе. Например, когда мы бросаем правильную монету два раза, вход (монета) один и тот же, но выход («орел» или «решка») может быть различен.

Результаты

Каждый выход случайного эксперимента называется **результатом.** Например, когда бросается шестисторонняя игральная кость, возможными событиями могут быть выпадение одного из перечисленных ниже чисел — 1, 2, 3, 4, 5 и 6.

Типовое пространство

Типовое пространство элементарных событий (S) является множеством всех возможных результатов случайного эксперимента. При бросании монеты пространство элементарных событий имеет только два элемента, S = {«орел», «решка»}. При бросании игровой кости типовое пространство имеет шесть элементов, S = {1, 2, 3, 4, 5, 6}. Типовое пространство иногда называют вероятностным пространством, случайным пространством или полным множеством (универсумом).

События

Когда случайный эксперимент закончен, нас интересует полученное подмножество типового пространства, не обязательно состоящее из одного результата. Например, когда игральная кость брошена, нас может интересовать выпадение числа 2, либо любого четного числа, либо числа меньше, чем 4. Каждый из этих возможных результатов можно представлять как **событие.** Событие A является подмножеством типового пространства, содержащим один результат. Все ранее упомянутые события могут быть определены следующим образом.

a. Получение 2 (простой результат): $A_1 = \{2\}$.

b. Получение четного числа: $A_2 = \{2, 4, 6\}$.

c. Получение числа меньше, чем 4: $A_3 = \{1, 2, 3\}$.

Определение вероятности

Главная идея в теории вероятностей — идея появления события. Но какова вероятность данного события? Этот вопрос обсуждался в течение многих столетий. Недавно математики пришли к соглашению, что мы можем определить вероятности событий, используя три метода: классический, статистический и вычислительный.

Классическое определение вероятности

В **классическом определении вероятности** вероятность события — число, интерпретируемое как P (A) = n_A/n, где n — общее количество возможных результатов и n_A — число возможных результатов, связанных с событием A. Это определение полезно, только если каждый результат одинаково вероятен.

Пример D.I

Мы бросаем «правильную монету». Какова вероятность, что результатом этого эксперимента будет «орел»?

Решение

Общее количество возможных результатов — 2 («орел» или «решка»). Число возможных результатов, связанных с этим событием, — 1 (выпадает «решка»). Поэтому мы имеем P («решка») = $n_{\text{решка}}/n = 1/2$.

Пример D.2

Мы бросаем игральную кость. Какова вероятность выпадения цифры 5?

Решение

Общее количество возможных результатов — 6, S = {1, 2, 3, 4, 5, 6}. Число возможных результатов, связанных с этим событием, — 1 (выпадение 5). Поэтому мы имеем P (5) = $n_5/n = 1/6$.

Статистическое определение вероятности

В **статистическом определении вероятности** эксперимент выполняется n раз при равных условиях. Если событие возникает m раз и n является разумно большим, вероятность события — это число, интерпретируемое как P (A) = m/n. Это определение полезно, когда события не одинаково вероятны.

Пример D.3

Мы бросаем «неправильную монету» 10 000 раз и получаем «орел» 2600 раз и «решку» 7400 раз. Поэтому P («орел») = 2600/10 000 = 0,26 и P (решка) = 7400/10 000 = 0,74.

Вычислительное определение вероятности

В вычислительном определении вероятности определяют вероятность события исходя из вероятностей других событий, используя аксиомы и свойства, приведенные ниже.

Аксиомы

Аксиомы вероятности не могут быть доказаны. Они вводятся как предположения при их использовании в теории вероятностей. Следующие три аксиомы — это фундаментальные аксиомы теории вероятностей.

- **Аксиома 1.** Вероятность события — это неотрицательное значение: $P(A) \geq 0$.
- **Аксиома 2.** Вероятность случайного пространства равна 1: $P(S) = 1$. Другими словами, при испытаниях всегда возникает один из возможных в пространстве результатов.
- **Аксиома 3.** Если A_1, A_2, A_3... — пара непересекающихся событий, то

$$P(A_1, \text{ или } A_2, \text{ или } A_3, \text{ или } ...) = P(A_1) + P(A_2) + P(A_3) + ...$$

События A_1, A_2, A_3...являются попарно непересекающимися событиями, если появление каждого из них не изменяет вероятность появления других.

Свойства

Если принять вышеупомянутые аксиомы, можно доказать список свойств. Ниже приведен минимальный список свойств, требуемый для понимания разделов в этой книге (доказательства следует искать в книгах по теории вероятностей).

- Вероятность события находится всегда между 0 и 1: $0 \leq P(A) \leq 1$.
- Вероятность никакого результата — 0: $P(\overline{S}) = 0$. Другими словами, если мы бросаем игральную кость, вероятность, что одно из чисел будет равно 7, равно 0 (невозможное событие).
- Если $\overline{A}$ — дополнение A, то $P(\overline{A}) = 1 - P(A)$. Например, если вероятность выпадения цифры 2 при бросании игральной кости — 1/6, вероятность неполучения цифры 2 — это (1 – 1/6).
- Если A — подмножество B, то $P(A) \leq P(B)$. Например, когда мы бросаем игральную кость P (2 или 3), — меньше чем P (2 или 3 или 4).
- Если события A, B, C... независимы, то

$$P(A \text{ и } B \text{ и } C \text{ и } ...) = P(A) \times P(B) \times P(C) \times ...$$

Условная вероятность

Появление одного события может быть связано с появлением другого события. **Условную вероятность** события B, при условии возникновения события A, записывают в виде P (B|A). Может быть доказано, что

$$P(B \mid A) = P(A \text{ и } B)/P(A).$$

Примечание: если A и B — независимые события, то P (B|A) = P (B).

Пример D.4

Бросается игральная кость. Если нам сообщают, что результат — четное число, какова вероятность, что это — 4?

Решение

P (4) = P (4)/P (четн). Если число является четным, есть только один способ получить число 4, P (4) = 1/6. P (четн) = P (2, или 4, или 6) = 3/6. Поэтому,

$$P(4 \mid \text{четн}) = (1/6) / (3/6) = 1/3$$

Обратите внимание, что условная вероятность ‘P (4 | четн) больше, чем P (4).

D.2. Случайные переменные

Переменная может принять различные значения. Переменные, значения которых зависят от результатов случайного эксперимента, названы **случайными переменными**.

Непрерывные случайные переменные

Случайные переменные, которые могут принимать бесконечное число значений, называются **непрерывными случайными переменными**. Мы в криптографии обычно не интересуемся этим типом случайных переменных.

Дискретные случайные переменные

В криптографии нас интересуют случайные эксперименты с конечным числом результатов (такие как с игральной костью). Случайные переменные, связанные с этим типом эксперимента, называются **дискретными случайными переменными.** Дискретная случайная переменная — отображение множества результатов на множество значений реальных чисел. Например, мы можем отобразить результаты подбрасывания монеты («орел», «решка») на множество чисел {0, 1}.

Приложение E. Проблемы дня рождения

Проблемы дня рождения обсуждались в лекции 11. В этом приложении даются общие решения четырех проблем дней рождения — мы используем теорию вероятностей, некоторые положения которой кратко приведены в приложении D. Для того чтобы упростить решения, применяются следующие математические уравнения:

$1 - x \approx e^{-x}$ // Ряд Тейлора при малом x
$1 + 2 + ... + (k - 1) \approx k(k - 1)/2$
$k(k - 1) \approx k^2$

E.1. Четыре проблемы

Мы представляем решения четырех проблем, обсужденных в лекции 11.

Первая проблема

Мы имеем множество выборок значений размером k, в которых каждый элемент может иметь только одно из N равновероятных значений. Каков должен быть минимальный размер выборки k в множестве, чтобы с вероятностью $P > 1/2$ по крайней мере один из элементов был бы равен заранее определенному значению?

Чтобы решить проблему, мы сначала находим вероятность P, что по крайней мере один элемент равен заранее заданному значению. Затем задаем вероятность к 1/2, чтобы найти минимальный размер элемента.

Вероятность

Чтобы найти вероятность P, мы делаем четыре шага.

1. Если $P_{эл}$ — вероятность того, что выбранный элемент равен заранее заданному значению, то $P_{эл} = 1/N$, потому что элемент может с равной вероятностью принимать любое из значений N.
2. Если $Q_{эл}$ — вероятность, что выбранный элемент не равен заранее заданному значению, то $Q_{эл} = 1 - P_{эл} = (1 - 1/N)$.
3. Если каждый элемент независим (справедливое предположение) и Q — вероятность, что ни один элемент не равен заранее заданному значению, то $Q = Q_{эл}^k = (1 - 1/N)^k$.
4. Наконец, если P — вероятность того что, по крайней мере один элемент равен заранее определенному значению, то $P = 1 - Q$ или $P = 1 - (1 - 1/N)^k$.

Размер выборки

Теперь мы находим минимальный размер выборки k, чтобы вероятность появления элемента была $P \geq 1/2$. Как показано ниже, значение k должно быть $k > \ln 2 \times N$:

$$P = 1-(1-1/N)^k \geq 1/2 \rightarrow (1-1/N)^k \leq 1/2$$

$$(1-1/N)^k \leq 1/2 \rightarrow (e^{-k/N}) \leq 1/2 \quad \textit{Используя аппроксимацию } 1-x \approx e^{-x} \textit{ при } x = 1/N$$

$$(e^{-k/N}) \leq 1/2 \rightarrow e^{k/N} \geq 2 \rightarrow k/N \geq \ln 2 \times N$$

Первая проблема
Вероятность: $P = 1 - (1 - 1/N)^k$ Размер выборки : $k \geq \ln 2 \times N$

Вторая проблема

Вторая проблема — та же самая, что и первая, за исключением того, что заранее заданное значение — один из элементов множества. Это означает, что мы можем найти результат второй проблемы, если заменим k на $k - 1$, потому что после выбора одного элемента из выборки элементов остается только $k - 1$ элементов. Поэтому $P = 1 - (1 - 1/N)^{k-1}$ и $k > \ln 2 \times N + 1$.

Вторая проблема
Вероятность: $P = 1 - (1 - 1/N)^{k-1}$ Размер выборки $k \geq \ln 2 \times N + 1$.

Третья проблема

В третьей проблеме мы должны найти минимальный размер выборки элементов k, такой, чтобы с вероятностью $P > 1/2$ по крайней мере два элемента имели одно и то же значение. Чтобы решить проблему, мы сначала находим соответствующую вероятность P, затем задаем вероятность 1/2, чтобы найти минимальный размер выборки.

Вероятность

Здесь мы используем не такую стратегию, как в предыдущих случаях.

1. Мы определяем вероятности появления элементов по одному. Предположим, что P является вероятностью, с который выборка элементов i имеет то же самое значение, что и одна из предыдущих выборок, и Q — вероятность, с который выборка i имеет значение, отличающееся от всех предыдущих выборок.
   a. Поскольку перед первой выборкой нет ни одной другой $P_1 = 0$ и $Q_1 = 1 - 0 = 1$.
   b. Поскольку перед второй выборкой есть одна другая выборка, первая и вторая выборки могут иметь одно из значений из N элементов. $P_2 = 1/N$ и $Q_2 = (1 - 1/N)$.
   c. Поскольку перед третьей выборкой есть две выборки, то каждая из этих двух выборок может иметь одно из значений из N элементов, $P_3 = 2/N$ и $Q_3 = (1 - 2/N)$.
   d. Продолжая эти рассуждения, мы получим $P_k = (k - 1)/N$ и $Q_k = (1 - (k - 1)/N)$.

2. Предположим, что все выборки независимы, тогда вероятность Q, с которой все выборки имеют различные значения,

$$Q = Q_1 \times Q_2 \times Q_3 \times \cdots \times Q_K = 1 \times (1 - 1/N) \times (1 - 2/N) \times \cdots \times (1 - (k-1)/N)$$
$$Q = (e^{-1/N}) \times (e^{-2/N}) \times \cdots \times (e^{-(k-1)/N}) \text{ Используем аппроксимацию } 1 - x \approx e^{-x} \text{ при } x = i/N$$
$$Q = e^{-k(k-1)/2N} \text{ Используем уравнение } 1 + 2 + \cdots + (л - 1) = k(k-1)/2$$
$$Q = e^{-k^2/2N} \text{ Используем аппроксимацию } k(k-1) \approx k^2$$

3. Наконец, если P — вероятность, с которой по крайней мере две выборки имеют одни и те же значения, то мы имеем P = 1 – Q или $P = 1 - e^{-k^2/2N}$

Размер выборки

Теперь мы можем найти минимальный размер выборки с вероятностью $P \geq 1/2$. Она равна $k \geq (2 \times \ln 2)^{1/2}$ или $k \geq 1{,}18 \times N^{1/2}$, доказательство приведено ниже:

$$P = 1 - e^{-k^2/2N} \geq 1/2 \rightarrow e^{-k^2/2N} \leq 1/2$$
$$e^{-k^2/2N} \leq 1/2 \rightarrow e^{-k^2/2N} \geq 2 \rightarrow k^2/2N \geq \ln 2 \rightarrow k \geq (2 \times \ln 2)^{1/2} \times N^{1/2}$$

Третья проблема
Вероятность: $P = 1 - e^{-k^2/2N}$ Размер выборки: $k \geq (2 \times \ln 2)^{1/2} \times N^{1/2}$

Четвертая проблема

В Четвертой проблеме мы имеем две выборки равного размера, *k*. Мы должны найти минимальное значение *k*, такое, что с вероятностью $P \geq 1/2$ по крайней мере одна из выборок в первом множестве имеет то же самое значение, что и выборка во втором множестве. Чтобы решить проблему, мы сначала находим соответствующую вероятность P. Затем мы задаем вероятность 1/2, чтобы найти минимальный размер выборки.

Вероятность

Мы решаем это с использованием стратегии, подобной той, которую мы использовали для первой проблемы.

1. Согласно первой проблеме, вероятность, что все выборки в первом множестве имеют значения, отличающиеся от значений первой выборки во втором множестве, — $Q_1 = (1 - 1/N)^k$.
2. Вероятность, что все выборки в первом множестве имеют значения, отличающиеся от первых и вторых выборок во втором множестве, – $Q_2 = (1 - 1/N)^k \times (1 - 1/N)^k$.
3. Мы продолжаем рассуждения, чтобы показать, что вероятность, что все выборки в первом множестве имеют значения, отличающиеся от любой выборки во втором множестве:

$$Q_k = (1-1/N)^k \times (1-1/N)^k \times \cdots \times (1-1/N)^k \rightarrow Q_k = (1-1/N)^{k^2}$$

$$Q_k = (1-1/N)^{k^2} \rightarrow Q_k = e^{-k^2/N} \quad \textit{Используя аппроксимацию } 1-x \approx e^{-x} \textit{ при } x = 1/N$$

4. Наконец, если P — вероятность, что по крайней мере одна выборка из первого набора имеет то же самое значение, что и одна из выборок во втором наборе, то $P = 1 - Q_k$ или $P = 1 - e^{-k^2/N}$

Размер выборки

Теперь мы находим минимальный общий размер выборок, как это показано ниже:

$$P = 1 - e^{-k^2/N} \geq 1/2 \rightarrow e^{-k^2/N} \leq 1/2 \rightarrow e^{k^2/N} \leq 2$$

$$e^{-k^2/N} \leq 1/2 \rightarrow e^{k^2/N} \geq 2 \rightarrow k^2/N \geq \ln 2 \rightarrow k \geq (\ln 2)^{1/2} \times N^{1/2}$$

Четвертая проблема

Вероятность: $P = 1 - e^{-k^2/2N}$ Размер выборки: $k \geq (2 \times \ln 2)^{1/2} \times N^{1/2}$

E.2. Итоги

Таблица E.1 дает выражения для вероятности (P) и размера выборки (k) для каждой из этих четырех проблем.

Таблица E.1. Итоги решений для четырех проблем дня рождения

Проблема	Вероятность	Общее выражение для k	Значение k при $P \geq 1/2$
1	$P \approx e^{-k/N}$	$k \approx \ln[1/(1-P)] \times N$	$k \approx 0{,}69 \times N$
2	$P \approx 1 - e^{(k-1)/N}$	$k \approx \ln[1/(1-P)] \times N+1$	$k \approx 0{,}69 \times N + 1$
3	$P \approx e^{-k^2/2N}$	$k \approx [2\ln(1/(1-P)]^{1/2} \times N^{1/2}$	$k \approx 1{,}18 \times N^{1/2}$
4	$P \approx e^{-k^2/N}$	$k \approx [\ln[(1/(1-P)]^{1/2} \times N^{1/2}$	$k \approx 0{,}83 \times N^{1/2}$

Приложение F. Теория информации

В этом приложении мы обсуждаем несколько концепций *теории информации*, которые связаны с темами, рассмотренными в этой книге.

F.1. Измерение информации

Как мы можем измерить информацию в событии? Сколько информации нам доставляет событие? Давайте ответим на эти вопросы с помощью примеров.

Пример F.1

Вообразите человека, сидящего в комнате. Глядя из окна, он может ясно видеть, что сияет солнце. Если в этот момент он получает сообщение (событие) от соседа, который говорит «Хороший день», это сообщение содержит какую-либо информацию? Конечно, нет! Человек уже уверен, что это день и погода хорошая. Сообщение не уменьшает неопределенности его знаний.

Пример F.2

Вообразите, что человек купил лотерейный билет. Если друг звонит, чтобы сказать, что он выиграл первый приз, это сообщение (событие) содержит информацию? Конечно, да! Сообщение содержит много информации, потому что вероятность выигрыша первого приза является очень маленькой. Приемник сообщения потрясен.

Вышеупомянутые два примера показывают, что есть отношения между полноценностью события и ожиданиями приемника. Если приемник удален от места события, когда событие случается, сообщение содержит много информации; иначе — это не так. Другими словами, информационное содержание сообщения обратно пропорционально связано с вероятностью возникновения этого сообщения. Если событие очень вероятно, оно не содержит никакой информации (Пример F.1); если оно является маловероятным, оно содержит много информации (Пример F.2).

F.2. Энтропия

Предположим, что S — распределение вероятностей конечного числа событий (См. приложение D). Энтропия или неопределенность в S может быть определена[1] как:

$$\mathbf{H(s) = \Sigma\, \mathit{P(s)} \times [\log_2 1/(\mathit{p(s)}]\ \mathit{бит}}$$

где $x \in S$ — возможный результат одного испытания. Обратите внимание, что, если $P(s) = 0$, то мы будем считать, что $P(s) \times [\log_2 1/(p(s)]$ равно 0, чтобы избежать деления на 0.

1 В математической литературе принят немного другой вид формулы, отображающей энтропию: $H(S) = -\Sigma p(s)\log_2 p(s)$

Пример F.3

Предположим, что мы бросаем правильную монету. Результаты — «орел» и «решка», каждый с вероятностью 1/2, и это означает

$$H(S) = P(\text{орел}) \times [\log_2 1/ (P(\text{решка})] + P(\text{решка}) \times [\log_2 1 / (P(\text{решка})]$$
$$H(S) = (1/2) \times [\log_2 1 / (1/2)\ 1 + (1/2) \times [\log_2 1 / (1/2)] = 1 \text{ бит}$$

Этот пример показывает, что результат бросания «правильной» монеты дает нам 1 бит информации (неопределенность). При каждом бросании мы не знаем, каков будет результат, поскольку две возможности одинаково вероятны.

Пример F.4

Предположим, что мы бросаем «неправильную» (поврежденную) монету. Результаты выпадения «орла» и «решки» следующие: P («орел») = 3/4 и P («решка») = 1/4. Это означает, что

$$H(S) = (3/4)\ x\ [\log_2 1 / (3/4)] + (1/4)\ x\ [\log_2 1 / (1/4)] = 0{,}8 \text{ бит}$$

Этот пример показывает, что результат бросания неправильной монеты дает нам только 0,8 битов информации (неопределенность). Количество информации здесь меньше, чем количество информации в Примере F.3, потому что мы ожидаем получить «орлов» большее число раз, чем «решек».

Пример F.5

Теперь предположим, что мы бросаем полностью неправильную монету, в которой результат является всегда «орел», P («орел») = 1 и P («решка») = 0. Энтропия в этом случае

$$H(S) = (1)\ x\ [\log_2 1)] + (0)\ x\ [\log_2 1 / (0)] = (1)\ x\ (0) + (0) = 0$$

В этом эксперименте нет никакой информации (неопределенности). Мы знаем, что результатом всегда будет «орел»; энтропия — 0.

Максимальная энтропия

Может быть доказано, что для распределения вероятностей с *n* возможными результатами максимальная энтропия может быть достигнута, только если все вероятности равны (все результаты одинаково вероятны). В этом случае максимальная энтропия

$$\mathbf{H_{max} = \log_2 n \text{ бит}}$$

Другими словами, энтропия любого множества вероятностей имеет верхний предел, который определяется этой формулой.

Пример F.6

Предположим, что бросается шестигранная игральная кость. Энтропия испытания равна

$$H(S) = \log_2 6 \approx 2{,}58 \text{ битов}$$

Минимальная энтропия

Можно доказать, что для распределения вероятностей с *n* возможными результатами, получается минимальная энтропия тогда и только тогда, когда все время получается один из результатов. В этом случае минимальная энтропия

$$\mathbf{H_{min}\ (S) = 0 \text{ битов}}$$

Другими словами, эта формула определяет нижний предел энтропии для любого набора вероятностей.

Энтропия любого набора вероятностей находится между 0 бит и $\log_2 n$ бит, где *n* — число возможных результатов.

Интерпретация энтропии

Энтропию можно воспринимать как число бит, которым можно представить каждый результат из множества вероятностей, в том случае, когда результаты одинаково вероятны. Например, когда возможное случайное распределение имеет восемь возможных результатов, каждый результат может быть представлен в виде трех бит (от 000 до 111). Когда мы получаем результат эксперимента, мы можем сказать, что получили 3 бита информации. Энтропия этого набора вероятностей — также 3 бита ($\log_2 8 = 3$).

Совместная энтропия

Когда мы имеем два набора распределения вероятностей, S_1 и S_2, мы можем определить совместную энтропию $H(S_1, S_2)$ как

$$\mathbf{H\ (S_1, S_2) = \Sigma\Sigma\ P(\mathit{x,y})) \times [\log_2 1/P\ (x, y)] \text{ бит}}$$

Условная энтропия

Мы часто должны знать неопределенность распределения вероятностей S_1, при условии получения результата, который определяется неопределенностью распределения вероятности S_2. Она называется условной энтропией $H(S_1| S_2)$. Может быть доказано, что

$$\mathbf{H\ (S_1| S_2) = H\ (S_1, S_2) - H\ (S_2) \text{ бит}}$$

Другие соотношения

Приведем здесь без доказательства некоторые другие соотношения для энтропии:

1. H (S_1, S_2) = H (S2 | S_1) + H (S_1) = H (S_1| S_2) + H (S2)
2. H (S_1, S_2) ≤ H (S_1) + H (S2)
3. H (S_1| S_2) ≤ H (S_1)
4. H (S_1, S2, S3) = H (S_1| S2, S3) + H (S_1, S3)

Второе и третье соотношения справедливы, если S_1 и S_2 статистически независимы.

Пример F.7

В криптографии, если **P** — распределение вероятностей исходного текста, **C** – распределение вероятностей зашифрованного текста и **K** – распределение вероятностей ключей, то **H (K|C)** может интерпретироваться как сложность атаки зашифрованного текста, в которой знание C может привести к знанию K.

Пример F.8

В криптографии, учитывая исходный текст и ключ, детерминированный алгоритм шифрования создает уникальный зашифрованный текст, что означает **H (C | K, P) = 0.** Также учитывая зашифрованный текст и ключевой алгоритм дешифрования, создается уникальный исходный текст, что означает **H (P | K, C) = 0**. Если дан зашифрованный текст и исходный текст, ключ также определяется уникально: **H (K | P, C) = 0**.

Совершенная секретность

В криптографии, если P, K и C — пространства выборки вероятности исходного текста, зашифрованного текста и ключа соответственно, то мы имеем H (P|C) ≤ H (P). Это может быть интерпретировано так: неопределенность P данного C меньше или равна неопределенности P. В большинстве криптографических систем, справедливо отношение H (P|C)< H (P), что означает, что перехват зашифрованного текста уменьшает знание, которое требуется для того, чтобы найти исходный текст. Криптографическая система обеспечивает **совершенную секретность**, если соблюдается соотношение H (P|C)=H (P), — это означает, что неопределенность исходного текста и данного зашифрованного текста – одна и та же неопределенность исходного текста. Другими словами, Ева не получает никакой информации, перехватив зашифрованный текст; она по-прежнему должна исследовать все возможные варианты.

Криптографическая система обеспечивает совершенную секретность, если H (P | C) = H (P).

Пример F.9

В предыдущих лекциях мы утверждали, что *одноразовый* шифр *блокнота* обеспечивает совершенную секретность. Докажем этот факт, используя предыду-

щие соотношения энтропии. Предположим, что алфавит — только 0 и 1. Если длина сообщения — L, может быть доказано, что ключ и зашифрованный текст состоят из 2^L символов, в которых каждый символ является одинаково вероятным. Следовательно, H (K) = H (C) = $\log_2 2^L$ = L.

Используя отношения, полученные в примере F.8, и то, что H (P, K) = H (P) + H (K), потому что P и K независимы, мы имеем

H (P, K, C) = H (C|P, K) + H (P, K) = H (P, K) = H (P) + H (K)
H (P, K, C) = H (K|P, C) + H (P, C) = H (P, C) = H (P|C) + H (C)
Это означает, что H (P | C) = H (P)

Пример F.10

Шеннон показал, что в криптографической системе, если (1) ключи возникают с равной вероятностью и (2) для каждого исходного текста и каждого зашифрованного текста есть уникальный ключ, то криптографическая система обеспечивает совершенную секретность. Доказательство использует тот факт, что в этом случае распределения вероятностей ключей, исходного текста и зашифрованного текста имеют один и тот же размер.

F.3. Энтропия языка

Интересно связать концепцию энтропии с естественными языками, такими как английский язык. В этом разделе мы касаемся некоторых пунктов, связанных с энтропией языка.

Энтропия произвольного языка

Предположим, что язык составлен из *N* букв и все буквы имеют равную вероятность появления. Мы можем сказать, что энтропия этого языка – $H_L = \log_2 N$. Например, если мы используем двадцать шесть прописных букв (от A до Z), чтобы передать наше сообщение, то энтропия, или информация, содержащаяся в каждой букве, равна $H_L = \log_2 26 = 4{,}7$ битов. Другими словами, от каждой буквы мы получаем 4,7 бита информации. Это означает, что мы можем кодировать буквы на этом языке, применяя слова по 5 битов; вместо того чтобы посылать букву, мы можем передать одно слово из 5 битов.

Энтропия английского языка

Энтропия английского языка — намного меньше, чем 4,7 бита, по двум причинам (если мы используем только прописные буквы). Первое: буквы возникают с неодинаковой вероятностью. Лекция 3 показывает частоту появления букв в английском языке. Буква E возникнет намного более вероятно, чем буква Z. Второе: существование диграмм (сочетаний по две буквы) и триграмм (сочетаний по три буквы) уменьшает количество информации в полученном тексте. Если мы получаем букву Q, вероятнее всего, что следующая буква — U. Также, если мы получа-

ем пять последовательных букв SELLI, то вероятно, что следующие две буквы будут QG. Эти два факта уменьшают энтропию английского языка. Шеннон показал, что среднее значение энтропии английского языка[1] равно 1,50.

Избыточность

Избыточность языка была определена как

$$\mathbf{R = 1 - H_L / (\log_2 N).}$$

В случае английского языка, используя только прописные буквы, мы получим R = 1 – 1,50/4,7 = 0,68. Другими словами, в английском сообщении есть 70-процентная избыточность. Алгоритм может сжать английский текст до 70 процентов, не теряя содержания.

Интервал однозначности

Другое определение, введенное Шенноном, — **интервал однозначности**. Интервал однозначности — минимальная длина зашифрованного текста, n_0, которая требуется Еве, чтобы уникально определить ключ (за достаточно большое число повторений) и в конечном счете вычислить исходный текст. Интервал однозначности определен как

$$\mathbf{n_0 = H\,(K) / [R \times H\,(P)]}$$

Пример F.11

Шифр подстановки использует множество ключей, состоящих из 26 ключей, и алфавит из 26 символов. Используя избыточность 0,70 для английского языка, определяем интервал однозначности:

$$n_0 = (\log_2 26!) / (0{,}70 \times \log_2 26) = 27$$

Это означает, что зашифрованный текст должен содержать по крайней мере 27 символов, для того чтобы Ева могла уникально найти исходный текст.

Пример F.12

Шифр сдвига использует множество из 26 ключей и алфавит из 26 символов. Используя избыточность 0,70 для английского языка, интервал однозначности определяем следующим соотношением:

$$n_0 = (\log 26)/0{,}70 \times \log_2 26) = 1{,}5$$

Это означает, что Еве необходимо иметь по крайней мере 2 символа зашифрованного текста, чтобы уникально найти исходный текст. Конечно, это весьма приблизительная оценка. В фактической ситуации Ева нуждается в большем количестве символов, чтобы нарушить код.

[1] А.Н. Колмогоров в своей статье «Три подхода к определению понятия количества информации» приводит оценку энтропии русского языка на основе словаря С.И. Ожегова –это 1,9 ± 0,1.

Приложение G. Список неприводимых и примитивных полиномов

Из лекции 4 мы узнали, что *неприводимый полином* в $GF(2^n)$ — полином степени n, который не может быть разложен на множители — полиномы со степенью меньше, чем *n*. Мы также узнали из лекции 5, что *примитивный полином* — это неприводимый полином, который является делителем числа $2^e + 1$, где *e* — наименьшее целое числа в форме $e = 2^k - 1$ и $k \geq 2$. Это означает, что примитивный полином — обязательно неприводимый полином, но неприводимый полином — не обязательно примитивный полином. Таблица G.1 показывает неприводимые и примитивные полиномы для степеней 1 – 8. В круглых скобках показаны неприводимые, но не примитивные полиномы.

Таблица G.1. Неприводимые и примитивные полиномы

n	Полиномы (в шестнадцатеричном формате)									
1	3	2								
2	7									
3	B	D								
4	13	19	(1F)							
5	25	29	2F	37	3B	3D				
6	43	(45)	49	57	5B	61	6D	73		
7	83 D3	87 D4	91 E5	9D EF	A7 Fl	AB F7	B9 FD	BF	Cl	CB
8	(11B) 169 (1BD)	11D 171 1CF	12B (177) (1D7)	12D (17B) (1DB)	(139) 187 1E7	(13F) (18B) (1F3)	14D (19F) 1F5	15F (1A3) (1F9)	163 1A9	165 (1B1)

Чтобы найти полином, представленный в таблице шестнадцатеричным числом, сначала запишите число в двоичном виде, а затем преобразуйте его в полином.

Пример G.1

Найдите первый примитивный полином степени 7.

Решение

Первое число для степени 7 - 83 в шестнадцатеричном виде, которое является и неприводимым, и примитивным полиномом. Целое число 83 в шестнадцатеричном виде эквивалентно 1000 0011 в двоичном. Соответствующий полином — $x^7 + x + 1$.

Пример G.2

Найдите первый неприводимый полином, который не примитивен.

Решение. Полином степени 6, который будет первым непримитивным полиномом степени 6, — (45) в шестнадцатеричном виде. Целое число 45 в шестнадцатеричном виде эквивалентно 100 0101 в двоичном виде (обратите внимание, что мы должны сохранить только 7 битов). Соответствующий полином — $x^6 + x^2 + 1$.

Пример G.3

Найдите второй неприводимый полином степени 8, который не примитивен.

Решение

Второй непримитивный полином степени 8 — (139) в шестнадцатеричном виде. Целое шестнадцатеричное число эквивалентно 1 0011 1001 в двоичном виде (обратите внимание, что мы должны сохранить только 9 битов). Соответствующий полином — $x^8 + x^5 + x^4 + x^3 + 1$.

Приложение Н. Простые числа, меньшие чем 10 000

Это приложение перечисляет простые числа, меньшие чем 10000. В каждой таблице каждое число в первом столбце является числом простых чисел в соответствующем диапазоне для этой строки, например, от 0 до 100 число простых чисел 25 и т. д.

Таблица Н.1. Список простых чисел в диапазоне 1-1000

25	2 3 5 7 11 13 17 19 23 29 31 3 741 43 47 53 59 61 67 71 73 79 83 89 97
21	101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199
16	211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
16	307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397
17	401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499
14	503 509 521 523 541 547 557 563 569 571 577 587 593 599
16	601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691
14	701 709 719 727 733 739 743 751 757 761 769 773 787 797
15	809 811 821 823 827 829 839 853 857 859 863 877 881 883 887
16	907 91 1 919 929 937 941 947 953 967 971 977 983 991 997
Всего число простых чисел в диапазоне 1-1000 равно 168	

Таблица Н.2 Список простых чисел в диапазоне 1001-2000

16	1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097
12	1103 1109 1117 1123 1 129 1 151 1 153 1163 1 171 1181 1187 1193
15	1203 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297
11	1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399
17	1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499
12	1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597
15	161 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699
12	1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789
12	1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889
13	1901 1907 1913 1931 1933 1949 1951 1973 1979 1987 1993 1997 1999
Всего число простых чисел в диапазоне 1001-2000 равно 134	

Таблица Н.3. Список простых чисел в диапазоне 2001-3000

14	2003 20111 2017 2027 2029 2039 2053 2063 2069 2081 2083 2087 2089 2099
10	2111 2113 21292131 21372141 214321532161 2179
15	2203 2309 2207 2311 2213 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297
15	2309 2311 2333 2339 2341 2347 2351 2357 2371 2377 2381 2383 2389 2393 2399
10	2411 2417 2423 2437 2441 2447 2459 2467 2473 2477

11	2503 2521 2531 2539 2543 2549 2551 2557 2579 2591 2593
15	2609 2707 2617 2711 2621 2633 2647 2657 2659 2663 2671 2677 2683 2687 2689 2693 2699
14	2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797
12	2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897
11	2903 2909 2917 2927 2939 2953 2957 2963 2969 2971 2999

Всего число простых чисел в диапазоне 2001-3000 равно 127

Таблица Н.4. Список простых чисел в диапазоне 3001-4000

12	3001 3011 3019 3023 3037 3041 3049 3061 3067 3079 3083 3089
10	3109 3119 3121 3137 3163 3167 3169 3181 3187 3191
11	3203 3209 3217 3221 3229 3251 3253 3257 3259 3271 3299
15	3301 3307 3313 3319 3323 3329 3331 3343 3347 3359 3361 3371 3373 3389 3391
11	3407 3413 3433 3449 3457 3461 3463 3467 3469 3491 3499
14	3511 3517 3527 3529 3533 3539 3541 3547 3557 3559 3571 3581 3583 3593
13	3607 3613 3617 3623 3631 3637 3643 3659 3671 3673 3677 3691 3697
12	3701 3709 3719 3727 3733 3739 3761 3767 3769 3779 3793 3797
11	3803 3821 3823 3833 3847 3851 3853 3863 3877 3881 3889
11	3907 3911 3917 3919 3923 3929 3931 3943 3947 3967 3989

Всего число простых чисел в диапазоне 3001-4000 равно 120

Таблица Н.5. Список простых чисел в диапазоне 4001-5000

15	4001 4003 4007 4013 4019 4021 4027 4049 4051 4057 4073 4079 4091 4093 4099
9	4111 4127 4129 4133 4139 4153 4157 4159 4177
16	4201 4211 4217 4219 4229 4231 4241 4243 4253 4259 4261 4271 4273 4283 4289 4297
9	4327 4337 4339 4349 4357 4363 4373 4391 4397
11	4409 4421 4423 4441 4447 4451 4457 4463 4481 4483 4493

12	4507 4513 4517 4519 4523 4547 4549 4561 4567 4583 4591 4597
12	4603 4621 4637 4639 4643 4649 4651 4657 4663 4673 4679 4691
12	4703 4721 4723 4729 4733 4751 4759 4783 4787 4789 4793 4799
8	4801 4813 4817 4831 4861 4871 4877 4889
15	4903 4909 4919 4931 4933 4937 4943 4951 4957 4967 4969 4973 49874993 4999

Всего число простых чисел в диапазоне 4001-5000 равно 119

Таблица Н.6. Список простых чисел в диапазоне 5001-6000

12	5003 5009 5011 5021 5023 5039 5051 5059 5077 5081 5087 5099
11	5101 5107 5113 5119 5147 5153 5167 5171 5179 5189 5197
10	5209 5227 5231 5233 5237 5261 5273 5279 5281 5297
10	5303 5309 5323 5333 5347 5351 5381 5387 5393 5399
13	5407 5413 5417 5419 5431 5437 5441 5443 5449 5471 5477 5479 5483
13	5501 5503 5507 5519 5521 5527 5531 5557 5563 5569 5573 5581 5591
12	5623 5639 5641 5647 5651 5653 5657 5659 5669 5683 5689 5693
10	5701 5711 5717 5737 5741 5743 5749 5779 5783 5791
16	5801 5807 5813 5821 5827 5839 5843 5849 5851 5857 5861 5867 5869 5879 5881 5897
	5903 5923 5927 5939 5953 5981 5987

Всего число простых чисел в диапазоне 5001-6000 равно 114

Таблица Н.7. Список простых чисел в диапазоне 6001-7000

12	6007 6011 6029 6037 6043 6047 6053 6067 6073 6079 6089 6091
11	6101 61136121 6131 6133 6143 6151 6163 6173 6197 6199
13	6203 6211 6217 6221 6229 6247 6257 6263 6269 6271 6277 6287 6299

15 6301 6311 6317 6323 6329 6337 6343 6353 6359 6361 6367 6373 6379 6389 6397

8	6421 6427 6449 6451 6469 6473 6481 6491
11	6521 6529 6547 6551 6553 6563 6569 6571 6577 6581 6599
10	6607 6619 6637 6653 6659 6661 6673 6679 6689 6691
12	6701 6703 6709 6719 6733 6737 6761 6763 6779 6781 6791 6793
12	6803 6823 6827 6829 6833 6841 6857 6863 6869 6871 6883 6899
13	6907 6911 6917 6947 6949 6959 6961 6967 6971 6977 6983 6991 6997

Всего число простых чисел в диапазоне 6001-7000 равно 117

Таблица Н.8. Список простых чисел в диапазоне 7001-8000

9 7001 7013 7019 7027 7039 7043 7057 7069 7079

10 7103 7109 7121 7127 7129 7151 7159 7177 7187 7193

11 7207 7211 7213 7219 7229 7237 7243 7247 7253 7283 7297

9 7307 7309 7321 7331 7333 7349 7351 7369 7393

11 7411 7417 7433 7451 7457 7459 7477 7481 7487 7489 7499

15 7507 7517 7523 7529 7537 7541 7547 7549 7559 7561 7573 7577 7583 7589 7591

12 7603 7607 7621 7639 7643 7649 7669 7673 7681 7687 7691 7699

10 7703 7717 7723 7727 7741 7753 7757 7759 7789 7793

10 7817 7823 7829 7841 7853 7867 7873 7877 7879 7883

10 7901 7907 7919 7927 7933 7937 7949 7951 7963 7993

Всего число простых чисел в диапазоне 7001-8000 равно 107

Таблица Н.9. Список простых чисел в диапазоне 8001-9000

11 8009 8011 8017 8039 8053 8059 8069 8081 8087 8089 8093

10 8101 8111 8117 8123 8147 8161 8167 8171 81798191

14 8209 8219 8221 8231 8233 8237 8243 8263 8269 8273 8287 8291 8293 8297

Приложение I. Простые множители целых чисел, меньшие чем 1000

Это приложение помогает находить целые числа, которые являются простыми сомножителями чисел, меньших, чем 1000. Таблицы I.1 и I.2 дают наименьшие простые сомножители. Эти таблицы не включают в себя четные целые числа (очевидно, что у них наименьший простой сомножитель — число 2) и целые числа, которые оканчиваются на 5 (у них наименьший простой сомножитель — 5). Обратите внимание, что если наименьший сомножитель целого числа не дается, тогда это число – простое (его наименьший сомножитель — это оно само).

Чтобы найти все сомножители целого числа, меньшие, чем 1000, сначала найдите наименьший сомножитель, затем разделите число на этот коэффициент. Затем снова используйте таблицу для поиска наименьшего сомножителя числа, полученного в результате деления. Это будет второй простой сомножитель, и так далее.

Пример I.1

Чтобы найти все простые сомножители числа 693, мы используем следующие шаги:

1. Наименьший простой сомножитель 693 есть 3; 693/3 = 231.
2. Наименьший простой сомножитель 231 есть 3; 231/3 = 77.
3. Наименьший простой сомножитель 77 есть 7; 77/7 = 11.
4. Целое число 11 — само простое число. Поэтому $693 = 3^2 \times 7 \times 11$.

Пример I.2

Чтобы найти простые сомножители числа 722, мы используем следующие шаги.

1. Число является четным, и наименьший простой сомножитель, очевидно, равен 2; 722/2 = 361.
2. Наименьший простой сомножитель 361 есть 19; 361/19 = 19.
3. Целое число 19 — само простое число. Поэтому $722 = 2 \times 19^2$.

Пример I.3

Чтобы найти все наименьшие простые сомножители числа 745, мы используем следующие шаги.

1. Число делится без остатка на 5, так что наименьший простой сомножитель, очевидно, есть 5; 745/5 = 149.
2. Целое число 149 — само простое число. Поэтому $745 = 5 \times 149$.

Таблица I.1. Наименьший сомножитель целого числа в диапазоне 1–500 (Н.С. означает «наименьший сомножитель»)

Целое	Н.С.	Целое	Н.С.	Целое	Н.С.	Целое	Н.С.	Целое	Н.С.
1	–	101	–	201	3	301	7	401	–
3	–	103	–	203	7	303	3	403	13
7	–	107	–	207	3	307	–	407	11
9	3	109	–	209	11	309	3	409	–
11	–	111	3	211	–	311	–	411	3
13	–	113	–	213	3	313	–	413	7
17	–	117	3	217	7	317	–	417	3
19	–	119	7	219	3	319	11	419	–
21	3	121	11	221	13	321	3	421	–
23	–	123	3	223	–	323	17	423	3
27	3	127	–	227	–	327	3	427	7
29	–	129	3	229	–	329	7	429	3
31	–	131	–	231	3	331	–	431	–
33	3	133	7	233	–	333	3	433	–
37	–	137	–	237	3	337	–	437	19
39	3	139	–	239	–	339	3	439	–
41	–	141	3	241	–	341	11	441	3
43	–	143	11	243	3	343	7	443	–
47	–	147	3	247	13	347	–	447	3
49	7	149	–	249	3	349	–	449	–
51	3	151	–	251	–	351	3	451	11
53	–	153	3	253	11	353	–	453	3
57	3	157	–	257	–	357	3	457	–
59	–	159	3	259	7	359	–	459	3
61	–	161	7	261	3	361	19	461	–
63	3	163	–	263	–	363	3	463	–
67	–	167	11	267	3	367	–	467	–
69	3	169	13	269	–	369	3	469	7
71	–	171	3	271	–	371	7	471	3
73	–	173	–	273	3	373	–	473	11
77	7	177	3	277	–	377	13	477	3
79	–	179	–	279	3	379	–	479	–
81	3	181	–	281	–	381	7	481	13
83	–	183	3	283	–	383	–	483	3
87	3	187	11	287	7	387	–	487	–
89	–	189	3	289	17	389	7	489	13
91	7	191	–	291	3	391	17	491	–
93	3	193	–	293	–	393	3	493	17
97	–	197	–	297	3	397	–	497	7
99	3	199	–	299	13	399	3	499	–

Таблица I.2. Наименьший сомножитель целого числа в диапазоне 501–1000 (Н.С. означает «наименьший сомножитель»)

Целое	Н.С.	Целое	Н.С.	Целое	Н.С.	Целое	Н.С.	Целое	Н.С.
501	3	601	–	701	–	801	3	901	17
503	–	603	3	703	19	803	11	903	3
507	–	607	–	707	7	807	3	907	–
509	3	609	3	709	–	809	–	909	3
511	7	611	13	711	3	811	–	911	–
513	3	613	–	713	13	813	3	913	11
517	11	617	–	717	3	817	19	917	7
519	3	619	–	719	–	819	3	919	–
521	–	621	3	721	7	821	–	921	3
523	–	623	7	723	3	823	–	923	13
527	17	627	3	727	^	827	–	927	3
529	23	629	17	729	3	829	–	929	–
531	3	631	–	731	17	831	3	931	7
533	13	633	3	733	–	833	7	933	3
537	3	637	7	737	11	837	3	937	–
539	7	639	3	739	–	839	–	939	3
541	–	641	–	741	3	841	29	941	–
543	3	643	–	743	–	843	3	943	23
547	–	647	–	747	3	847	7	947	–
549	3	649	11	749	7	849	3	949	13
551	19	651	3	751	–	851	23	951	3
553	7	653	–	753	3	853	–	953	–
557	–	657	3	757	–	857	–	957	3
559	13	659	–	759	3	859	–	959	7
561	3	661	–	761	–	861	3	961	31
563	–	663	3	763	7	863	–	963	3
567	3	667	23	767	13	867	3	967	–
569	–	669	3	769	–	869	11	969	3
571	–	671	11	771	3	871	13	971	–
573	3	673	–	773	–	873	3	973	7
557	–	677	–	777	3	877	–	977	–
579	3	679	7	779	19	879	3	979	11
581	7	681	3	781	11	881	–	981	3
583	11	683	–	783	3	883	–	983	–
587	–	687	3	787	–	887	–	987	3
589	19	689	13	789	3	889	7	989	23
591	3	691	–	791	7	891	3	991	–
593	–	693	3	793	3	893	19	993	3
597	3	697	–	797	–	897	3	997	–
599	–	699	3	799	17	899	29	999	3

Приложение J. Список первых первообразных корней для простых чисел, меньших чем 1000

Таблица J.1 показывает первый первообразный корень по модулю простого числа первообразных корней для простых чисел, меньших чем 1000.

Таблица J.1

Простое	Корень	Простое	Корень	Простое	Корень	Простое	Корень	Простое	Корень	Простое	Корень	Простое	Корень
2	1	103	5	241	7	401	3	571	3	739	3	919	7 .
3	2	107	2	251	6	409	21	577	5	743	5	929	3
5	7	109	6	257	3	419	2	587	2	751	3	937	5
7	3	113	2	263	5	421	2	593	3	757	2	941	2
11	2	127	3	269	2	431	7	599	7	761	6	947	2
13	2	131	2	271	6	433	5	601	7	769	11	953	3
17	3	137	3	277	5	439	15	607	3	773	2	967	5
19	2	139	2	281	3	443	2	613	2	787	2	971	2
23	5	149	2	283	3	449	3	617	3	797	2	977	3
29	2	151	6	293	2	457	13	619	2	809	3	983	5
31	3	157	5	307	5	461	2	631	3	811	3	991	6
37	2	163	2	31 1	17	463	3	641	3	821	2	997	7
41	6	167	5	313	10	467	2	643	11	823	3		
43	3	173	2	317	2	479	13	647	5	827	2		
47	5	179	2	331	3	487	3	653	2	829	2		
53	2	181	9	337	10	491	2	659	2	839	11		
59	2	191	19	347	2	499	7	671	2	853	2		
61	2	193	5	349	2	503	5	673	5	857	3		
67	2	197	2	353	2	509	2	677	9	859	2		
71	2	199	3	359	7	521	3	683	5	863	5		
73	5	211	2	367	6	523	2	691	3	877	2		
79	3	323	3	373	2	541	2	701	2	881	3		
83	2	227	2	379	2	547	2	709	2	883	2		
89	2	229	6	383	5	557	2	719	11	887	5		
97	5	!33	3	389	2	563	2	727	5	907	2		
101	2	239	7	397	5	569	3	733	6	911	17		

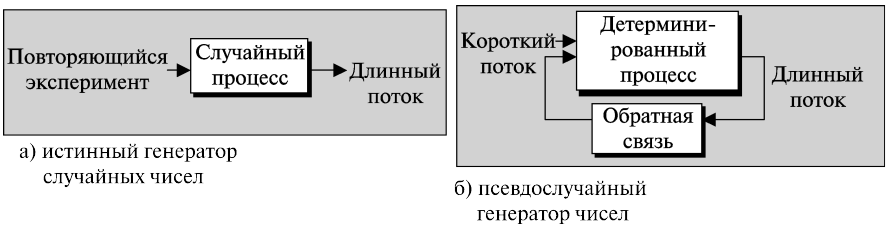
Приложение K. Генератор случайных чисел

Криптография и случайность имеют тесную связь. В приложении F, *Теория информации*, мы упоминали, что совершенная секретность может быть достигнута, если ключ алгоритма шифровки — действительно случайное число. Есть два подхода к получению длинного потока случайных битов.

1. Использование естественного случайного процесса, такого как многоразовое бросание монеты и интерпретация результата «орел» или «решка», как значения битов 0 или 1.
2. Использование детерминированного процесса с информацией обратной связи.

Первый подход назван **истинным генератором случайных чисел (TRNG — True Random Number Generator).**

Второй назван **псевдослучайным генератором числа (PRNG — Pseudorandom Number Generator).** Рисунок K.1 показывает эти два подхода.



а) истинный генератор случайных чисел

б) псевдослучайный генератор чисел

Рис. K.1. Истинный генератор случайных чисел и псевдослучайный генератор чисел

K.1. Истинный генератор случайных чисел (TRNG)

При бросании правильной монеты непрерывно возникает совершенный случайный поток битов, но это неприменимо на практике. Есть много естественных источников, которые могут произвести истинные случайные числа, такие как тепловой шум в электрическом резисторе или время ответа механического или электрического процесса после передачи команды. Эти природные ресурсы использовались в прошлом, и некоторые из них были внедрены в коммерческую деятельность. Однако есть несколько недостатков такого подхода. Процесс обычно медленный, и если необходимо, один и тот же случайный поток не может быть повторен.

K.2. Генератор псевдослучайных чисел (PRNG)

Случайный поток битов может быть получен с использованием детерминированного процесса при введении короткого случайного потока (начального числа). Генератор псевдослучайных чисел применяет такой подход. Сгенериро-

ванное число не случайно, потому что процесс, который его создает, детерминирован. Генераторы псевдослучайных чисел могут быть разделены на две широких категории: конгруэнтные генераторы и генераторы, использующие криптографические шифры. Мы обсуждаем некоторые генераторы в каждой категории.

Конгруэнтные генераторы

Несколько методов используют некоторые конгруэнтные отношения.

Линейный конгруэнтный генератор

В информатике самая общая методика для того, чтобы производить псевдослучайные числа, — линейный конгруэнтный метод, введенный Лехмером (Lehmer). Рисунок K.2 показывает этот метод, который рекурсивно создает последовательность псевдослучайных чисел, используя линейное конгруэнтное уравнение $x_{i+1} = (ax_i + b) \bmod n$, где x_0 называется начальным числом (seed) — это число между 0 и $n - 1$.

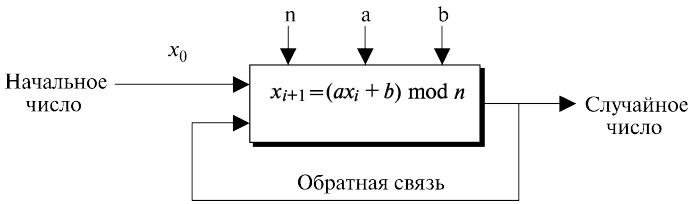


Рис. K.2. Линейный конгруэнтный генератор псевдослучайных чисел

Последовательность является периодической, где период зависит от того, как тщательно выбраны коэффициенты a и b. Идеально период должен быть такого размера, как модуль n.

Пример K.1

Предположим $a = 4$, $b = 5$, $n = 17$ и $x_{i0} = 7$. Последовательность — 16, 1, 9, 7, 16, 1, 9, 7..., которая есть явно неудовлетворительная псевдослучайная последовательность; её период — только 4.

Критерии. Для приемлемого генератора псевдослучайных чисел (PRNG) в течение прошлых нескольких десятилетий были разработаны несколько критериев.

1. Период должен быть равен n (модулю). Это означает, что прежде чем целые числа в последовательности начинают повторяться, должны быть сгенерированы все целые числа между 0 и $n - 1$.
2. Последовательность в каждый период должна быть случайна.
3. Процесс генерации должен быть удобен для реализации на компьютере. Большинство компьютеров сегодня эффективно, когда применяется арифметика, использующая слова по 32 бита.

Рекомендации, основанные на предыдущих критериях: рекомендуется выбрать коэффициенты конгруэнтного уравнения и значения модуля исходя из следующих соображений.

1. Оптимальный выбор модуля, *n,* — это наибольшее простое число, близкое к размеру слова, используемого в компьютере. Рекомендуется использовать тридцать первое простое число Мерсенны как модуль: $n = M_{31} = 2^{31} - 1$.
2. Чтобы создавать период, равный значению модуля, значение первого коэффициента, *a*, должно быть первообразным корнем главного модуля. Хотя целое число 7 — первообразный корень M_{31}, рекомендуют использовать 7^k, где *k* — целое число, взаимно-простое с $(M_{31} - 1)$. Некоторые рекомендованные значения для *k* — это 5 и 13. Это означает, что $(a = 7^5)$ или $(a = 7^{13})$.
3. Вторая рекомендация: для эффективного применения компьютера значение второго коэффициента *b* должно быть нулевым.

Линейный конгруэнтный генератор:

$x_{i+1} = ax_i \bmod n$, где $n = 2^{31} - 1$ и $a = 7^5$ или $a = 7^{13}$

Безопасность. Последовательность, сгенерированная линейным конгруэнтным уравнением, показывает приемлемую случайность (если следовать предыдущим рекомендациям). Последовательность полезна в некоторых приложениях, где требуется только случайность (таких как моделирование); она бесполезна в криптографии, где желательны и случайность, и безопасность. Поскольку число *n* общедоступно, последовательность может быть атакована Евой с использованием одной из двух стратегий:

a. если Ева знает значение начального числа (x_0) и коэффициент *a*, она может легко восстановить целую последовательность;
b. если Ева не знает значение x_0 и *a*, она может перехватить первые два целых числа и использовать следующие два уравнения, чтобы найти x_0 и *a:*

$$x_1 = ax_0 \bmod n \quad x_2 = ax_1 \bmod n$$

Генератор квадратичных вычетов

Чтобы получить менее предсказуемую псевдослучайную последовательность, был введен генератор квадратичных вычетов (см. лекцию 9), $x_{i+1} = x_i^2 \bmod n$, где x_0 называют начальным числом, — число между 0 и $n - 1$.

Генератор Blum Blum Shub

Простой, но эффективный метод создания генератора псевдослучайных чисел назван ***Blum Blum Shub (BBS)*** по имени его трех изобретателей.

BBS использует уравнение квадратичного вычета, но это — псевдослучайный генератор бит вместо генератора псевдослучайных чисел; он генерирует последовательность битов (0 или 1). Рисунок К.3 показывает идею этого генератора.

Ниже приведены шаги генерации.

1. Найдите два больших простых числа *p* и *q* в форме $4k + 3$, где *k* — целое число (*p* и *q* являются конгруэнтными 3 mod 4) .

2. Выберите модуль $n = p \times q$.
3. Выберите случайное целое число r, которое является взаимно-простым с n.
4. Вычислите начальное число как $x_0 = r^2 \bmod n$.
5. Сгенерируйте последовательность $x_{i+1} = x_i^2 \bmod n$.
6. Возьмите самый младший бит сгенерированного случайного целого числа (LSB — Least Significant Bit) как случайный бит.

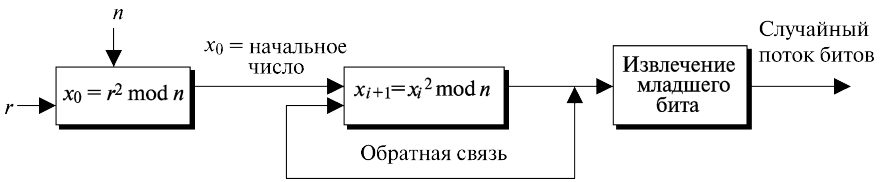


Рис. K.3. Blum Blum Shub (BBS) генератор псевдослучайных чисел

Безопасность. Может быть доказано, что если p и q известны, i-тый бит в последовательности можно найти как самый младший бит:

$$x_i = x_0^{2i \bmod [(p-1)(q-1)]} \bmod n$$

Это означает, что если Ева знает значение p и q, она может найти значение i-того бита, пробуя все возможные значения n (значение n обычно общедоступно). Тем самым сложность у этого генератора — та же самая, как у разложения на множители n. Если n является достаточно большим, последовательность безопасна (непредсказуема). Было доказано, что при очень большом n Ева не может предсказать значение следующего бита в последовательности, даже если она знает значения всех предыдущих битов. Вероятность каждого принятия значений для каждого бита, 0 или 1, — очень близка к 50 процентам.

Безопасность BBS зависит от трудности разложения на множители *n*.

Генераторы на основе криптографической системы

Криптографические системы, такие как шифр для процесса шифрования или хэш-функция, могут также быть использованы для генерации случайного потока битов. Мы кратко покажем две системы, которые применяют алгоритмы шифрования.

ANSI X9.17 генератор псевдослучайных чисел (PRNG)

ANSI X9.17 определяет криптографически сильный генератор псевдослучайных чисел, использующий тройной 3DES с двумя ключами (шифрация – дешифрация – шифрация), рисунок K.4 иллюстрирует этот проект. Обратите внимание, что первое псевдослучайное число это — 64-битовое начальное число, используемое как инициирующий вектор (IV); остальная часть псевдослучайных чисел использует начальное число, показанное как *следующие IV*. Такой же ключ

засекречивания на 112 битов (K_1 и K_2 в 3DES) применяется для всех трех 3DES-шифров.

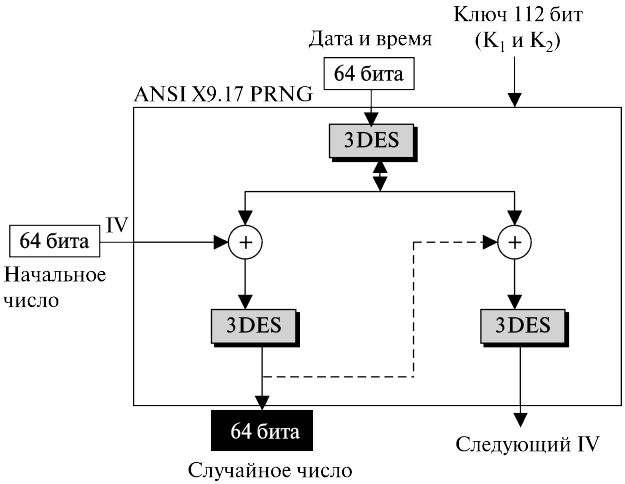


Рис. К.4. ANSI X9.17 генератор псевдослучайных чисел

На рис. К.4 конфигурация — режим *сцепления блоков шифрованного текста* (*CBC*), который мы описали согласно рис. 8.3 в лекции 8. Режим X9.17 применяет два каскада формирования цепочки блока. Исходный текст для каждого каскада поступает от выхода первого 3DES, который использует дату и время как исходный текст на 64 бита. Зашифрованный текст, созданный вторым 3DES, — случайное число; зашифрованный текст, созданный третьим 3DES, – следующий инициирующий вектор IV для следующего случайного числа.

Строгость X9.17 определяется следующими фактами.

1. Ключ — 112 (2 × 56) бит.
2. Ввод даты и времени на 64 бита обеспечивает хорошую метку времени, предотвращающую атаку воспроизведения.
3. Система обеспечивает превосходный эффект рассеивания и перемешивания с помощью шести шифрований и трех дешифрований.

PGP генератор псевдослучайных чисел (PRNG)

PGP (очень хорошая конфиденциальность) берет ту же самую идею, что и X9.17 с несколькими изменениями. Сначала PGP PRNG использует семь каскадов вместо двух. Второе: шифр является или IDEA, или CAST 128 (не рассмотренный в этой книге). Третье: ключ — обычно 128 битов. PGP PRNG создает три случайных числа на 4 бита: первое используется как инициирующий вектор IV секретности (для связи, работающей с PGP, но не для PRNG), второй и третий конкатенируются, чтобы создать секретный ключ 128 битов (для связи, работающей PGP). Рисунок К.5 показывает эскиз PGP PRNG. Строгость PGP PRNG за-

дана в размере его ключа и в том, что оригинал IV (начальное число) и ключ засекречивания на 128 битов могут быть сгенерированы от 24-байтовой истинно случайной переменной.

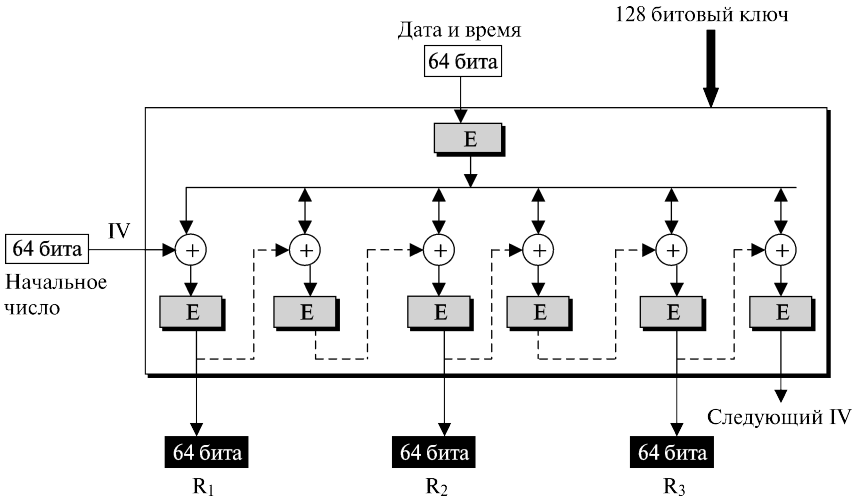


Рис. K.5. PGP-генератор псевдослучайных чисел

Приложение L. Сложность

L.1. Сложность алгоритма

В криптографии мы нуждаемся в инструменте для анализа вычислительной сложности алгоритма. Нам нужно, чтобы алгоритмы шифрования (или дешифрования) имели низкий уровень сложности (были эффективными). Еще условие: чтобы алгоритм, с точки зрения криптоанализа (перехвата кода), имел высокий уровень сложности (был не эффективен). Другими словами, мы хотим выполнять шифрование и дешифрование как можно быстрее, но чтобы злоумышленник при этом не смог никогда, даже с использованием компьютера, расшифровать сообщение.

Сложность алгоритма обычно базируется на двух типах ресурсов. **Пространственная сложность** алгоритма рассматривает объем памяти, которая должна хранить алгоритм (программу) и данные. **Сложность по времени** алгоритма рассматривает сумму времени, необходимую для выполнения алгоритма (программы) от начала до получения результата.

Сложность побитовой операции

В остальной части этого приложения мы будем говорить только о сложности по времени, которая имеет большее влияние на работу алгоритма, поскольку этот более общий показатель к тому же проще измерять. Сложность по времени алгоритма зависит от конкретного компьютера — того, на котором алгоритм должен быть выполнен. Чтобы сделать сложность независимой от компьютера, была определена **сложность побитовой операции** $f(n_b)$, — она подсчитывает число элементарных операций, которые должен выполнить компьютер, чтобы создать результат при n_b-битовом вводе. Элементарная обработка бита — это время, которое требуется компьютеру, чтобы сложить, вычитать, умножать или делить два единственных бита или сдвигать единственный бит.

Пример L.1

Какова сложность побитовой операции, которая выполняет функцию сложения двух целых чисел?

Решение

Сложность операции — $f(n_b) = n_b$, где n_b — число битов, которые представляют большое целое число. Если значение этого числа — N, то $n_b = log_2 N$.

Пример L.2

Какова сложность побитовой операции для функции, которая умножает два целых числа?

Решение

Хотя сегодня есть более быстрые алгоритмы умножения двух целых чисел, традиционно число разрядных операций предполагает, что эта сложность равна n_b^2, где n_b — число битов, представляющих большее целое число. Сложность поэтому — $f(n_b) = n_b^2$.

Пример L.3

Какова сложность побитовой операции для функции, складывающей два целых числа, каждое из которых содержит d десятичных цифр?

Решение

Максимальное значение множества цифр d разрядного десятичного числа — $N = 10^d - 1$ или $N \approx 10^d$. Биты во входном слове — $n_b = \log_2 N = \log_2 10^d =, d \times \log_2 10$. Тогда сложность равна $f(n_b) = d \times \log_2 10$. Например, *если* $d = 300$ цифр, $f(n_b) = 300 \log_2 10 \approx 997$ побитных операции.

Пример L.4

Какова сложность побитовой операции для функции, которая вычисляет $B = A^C$ (если A<C)?

Решение

Предположим, что число битов в C – n_b ($C = 2^{n_b}$ или $n_b = \log_2 C$). Обычный метод возведения в степень использует умножение числа самого на себя C раз. Каждая операция умножения нуждается *в* n_b побитовых операций (при использовании обычного алгоритма умножения). Поэтому сложность — $f(n_b) = C \times n^2_b = 2^{n_b} \times n_b^2$. Например, если C находится в диапазоне 2^{1024} ($n_b = 1024$), обычный метод возведения в степень имеет сложность

$$f(n_b) = 2^{1024} \times 10242 = 2^{1024} \times (2^{10})^2 = 2^{1044}$$

Это означает, что если компьютер может сделать 2^{20} (почти один миллион) побитных операций в секунду, для выполнения этой операции ему потребуется $2^{1044} / 2^{20} = 2^{1024}$ секунд.

Пример L.5

Какова сложность побитовой операции функции, которая вычисляет $B = A^C$ (если A < C) с использованием показательного алгоритма (метод «возведения в квадрат и умножения», рассмотренный в лекции 9)?

Решение

Мы показали в лекции 9, что быстрый показательный алгоритм использует максимум $2n_b$ умножений, где n_b — число битов в двоичном представлении C. Каждый оператор умножения требует n_b^2 побитных операций. Поэтому сложность — $f(n_b) = 2^{n_b} \times n^2 = 2n_b^3$. Например, в диапазоне 2^{1024} ($n_b = 1024$), быстрый показательный алгоритм дает нам

$$f(n_b) = 2 \times 1024^3 = 2^1 \times (2^{10})^3 = 2^{31}$$

Это означает, что если компьютер может сделать 2^{20} (почти один миллион) операций в секунду, он затратит на выполнение этой операции $2^{31}/2^{20} = 2^{11}$ секунды (почти 34 минуты). Сегодня компьютер может выполнить эту операцию гораздо быстрее.

Асимптотическая сложность

Цель оценки сложности состоит в том, чтобы измерить поведение алгоритмов, когда число n_b бит на входе является очень большим. Например, предположим, что ниже показаны оценки сложности двух алгоритмов:

$$f_1(n_b) = 5 \times 2^{n_b} + 5n_b \text{ и } f_2(n_b) = 2^{n_b} + 4$$

Когда n_b мало, эти два алгоритма ведут себя иначе, когда n_b является большим (приблизительно 1000). В последнем случае два алгоритма ведут себя почти одинаково. Причина в том, что элементы 5, $5n_b$ и 4 настолько малы по сравнению с элементом 2^{n_b}, что они могут полностью игнорироваться. Мы можем сказать, что для больших n_b $f_1(n_b) = f_2(n_b) = 2^{n_b}$. Другими словами, мы интересуемся $f(n_b)$, когда n_b приближается *к* бесконечности.

Большие О-нотации

Применив асимптотическую сложность, мы можем определить стандартный масштаб сложности функций с дискретными значениями переменных и определить сложность алгоритма, используя одно из этих значений. Один из общих стандартов называется **большой О-нотацией (Big O-Notation).** В этом стандарте $f(n_b) = O(g(n_b))$, где $g(n_b)$ — функция n_b, полученная из $f(n_b)$ с использованием следующих трех теорем.

- **Первая теорема.** Если мы можем найти константу K, такую, что $f(n_b) < K \times g(n_b)$, то мы имеем $f(n_b) = O(g(n_b))$. Эта теорема может быть легко реализована, если применить следующие два простых правила.
  a. Установите коэффициенты n_b в функции $f(n_b)$ на 1.
  b. Сохраните наибольший элемент в $f(n_b)$ как $g(n_b)$ и удалите другие. Элементы оцениваются от наиболее низкого к самому высокому, как это показано ниже:

(1). $(\log n_b)$, (n_b), $(n_b \log n_b)$, $(n_b \log n_b \log \log n_b)$, (n_b^2), (n_b^3),....., (n_b^k), (2^{nb}), $(n_b!)$

- **Вторая теорема.** Если $f_1(n_b) = \mathbf{O}(g_1(n_b))$ и $f_2(n_b) = \mathbf{O}(g_2(n_b))$, то
  $$f_1(n_b) + f_2(n_b) = \mathbf{O}(g_1(n_b) + g_2(n_b))$$
- **Третья теорема.** Если $f_1(n_b) = (g_1(n_b))$ и $f_2(n_b) = \mathbf{O}(g_2(n_b))$, то
  $$f_1(n_b) \times f_2(n_b) = \boldsymbol{O}(g_1(n_b) \times g_2(n_b))$$

Пример L.6

Найдите большие О-нотации для $f(n_b) = n_b^5 + 3n_b^2 + 7$.

Решение

Обратите внимание, что $f(n_b) = n_b^5 + 3n_b^2 + 7n_b^0$. Применение первого правила первой теоремы дает $g(n_b) = n_b^5 + n_b^2 + 1$. Применение второго правила дает нам $g(n_b) = n_b^5$. Большая **О**-нотация — $\mathbf{O}(n_b^5)$.

Пример L.7

Найдите большие О-нотации для $f(n_b) = (2^{n_b} + n_b^5) + (n_b \log_2 n_b)$.

Решение

Мы имеем $f_1(n_b) = (2^{n_b} + n_b^5)$ и $f_2(n_b) = (n_b \log_2 n_b)$. Поэтому $g_1(n_b) = 2^{n_b}$ и $g_2(n_b) = n_b \log_2 n_b$.

Применяя вторую теорему, мы имеем $g(n_b) = 2^{n_b} + . n_b \log_2 n_b$. Снова применяя первую теорему, мы получаем $g(n_b) = 2^{n_b}$. Большая **О**-нотация — $\mathbf{O}(2^{n_b})$.

Пример L.8

Найдите Большую **O**-нотацию для $f(n_b) = n_b!$ (n_b факториал).

Решение

Мы знаем $n_b! = n_b! \times (n_b! - 1) \times \ldots\ldots \times 2 \times 1$. Каждый элемент имеет максимальную сложность $O(n_b)$. Согласно третьей теореме, полная сложность — n_b раз $O(n_b)$, или $O(n_b{}^{n_b})$.

Иерархия сложности

Предыдущее обсуждение позволяет нам ранжировать алгоритмы, основанные на их сложности побитовой операции. Таблица L.1 дает общие уровни иерархии, используемые в литературе.

Таблица L.1. Иерархия сложностей и большие O-нотации (Big-O-notations)

Иерархия	**Большая- O-нотация**
Константа	$O(1)$
Логарифмическая	$O(\log n_b)$
Полином	$O(n^c)$, где c — константа
Субэкспоненциальная	$O(2^{p(\log n_b)})$, где p — полином в $\log n_b$
Экспоненциальная	$O(2^{n_b})$
Суперэкспоненциальная	$O(n_b{}^{n_b})$ или $O(2^{2^{n_b}})$

Алгоритм с *постоянной*, *логарифмической* и *полиномиальной* сложностью считают выполнимым для любого размера. Алгоритм с *показательной* и *суперэкспоненциальной* сложностью считают неосуществимым, если n_b очень большое. Алгоритм с *субэкспоненциальной* сложностью (такой как $O(2^{(\log n_b)^2})$) выполним, если n_b не является очень большим.

Пример L.9

Как показано в примере L.4, сложность обычного возведения в степень — $f(n_b) = 2^{n_b} \times n_b{}^2$. Большая **O**-нотация для этого алгоритма — $O(2^{n_b} \times n_b{}^2)$, больше, чем показательный. Этот алгоритм неосуществим, если n_b выбрать очень большим.

Пример L.10

Как показано в примере L.5, сложность быстрого показательного алгоритма $f(n_b) = 2n_b{}^3$ большая $\mathbf{O}(n_b{}^3)$-нотация для этого алгоритма, которая является полиномиальной. Этот алгоритм выполним; он используется в криптографической системе RSA.

Пример L.11

Предположим, что криптографическая система имеет длину ключа n_b бит. Чтобы провести атаку грубой силы для этой системы, противник должен проверить 2^{n_b} различных ключа. Это означает, что алгоритм должен пройти 2^{n_b} шагов.Если N — число побитных операций для каждого шага, сложность алгоритма

должна быть $f(n_b) = N \times 2^{nb}$. Даже если N — константа, сложность этого алгоритма показательна, $O(2^{nb})$. Поэтому для большого n_b атака неосуществима. В лекции 6 мы видели, что DES с ключом на 56 битов уязвимы к атаке грубой силы, но для 3DES с ключом 112 бит она неосуществима. В лекции 7 мы также убедились, что AES с ключом на 128 битов имеет иммунитет к этой атаке.

L.2. Сложность проблемы

Теория сложности также рассматривает вопрос определения *сложности проблемы* до того, как алгоритм будет разработан. Чтобы оценить сложность проблем, могут использоваться две версии теоретических машин Тьюринга: детерминированная и недетерминированная. Недетерминированная машина может решить более трудные проблемы подбором первого решения и затем его проверкой.

Две широких категории

Теория сложности делит все проблемы на две широких категории: **неразрешимые проблемы** и **разрешимые проблемы.**

Неразрешимые проблемы

Неразрешимая проблема — проблема, для которой нет алгоритма решения. Алан Тьюринг показал, что известная *проблема остановки* неразрешима. Проблема остановки может быть просто изложена следующим образом: «Если дано описание алгоритма, входная информация и машина Тьюринга, то нет алгоритма, который мог бы определить, завершится ли когда-нибудь выполнение этого алгоритма (остановится ли в конечном счете машина)». В математике и информатике есть несколько неразрешимых проблем.

Разрешимые проблемы

Проблема *разрешима*, если может быть написан алгоритм для решения этой проблемы. Соответствующий алгоритм, однако, может быть или не быть выполнимым. Если проблема может быть решена с использованием алгоритма полиномиальной сложности или меньше, это называется разрешимой в разумное время проблемой или просто *разумной проблемой* (*tractable problem)*. Если проблема может быть решена с применением алгоритма показательной сложности, ее называют неразрешимой в разумное время или *тяжелой проблемой* (*intractable problem)*.

P, NP и coNP. Теория сложности делит разумные проблемы на три (возможно, накладывающиеся друг на друга) класса — **P**, **NP** и **coNP.** Как показано на рис. L.I, NP и coNP — перекрывающиеся, а P-класс находится на пересечении этих классов. Проблемы в классе P (P — *polynomial) полиномиальные* и могут быть решены детерминированной машиной Тьюринга в полиномиальное время. Проблемы NP класса (NP — *nondeterministic polynomial* для *недетерминированного полинома)* могут быть решены недетерминированной машиной Тьюринга в полиномиальное время. Проблемы в классе coNP (coNP — *complementary nondeterministic polynomial — дополнительного недетерминированного полинома)* — такие проблемы, которые могут быть решены недетерминированно путем дополнения машины Тьюринга. На-

пример, проблема, которая решает, может ли целое число быть разложено на множители в два простых числа, есть дополнение проблемы, которая может решить, является ли число простым. Другими словами, утверждение «может быть разложен на множители» является эквивалентным доказательству «непростое число».

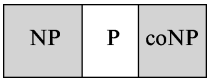


Рис. L.1. Классы P, NP и coNP

L.3. Вероятностные алгоритмы

Если проблема тяжелая, для ее решения мы можем найти вероятностный алгоритм. Хотя вероятностные алгоритмы не гарантируют, что решение свободно от ошибок, вероятность ошибки может быть сделана очень маленькой, с помощью повторения алгоритма с применением нескольких различных параметров. Вероятностные алгоритмы могут быть разделены на две категории: *Монте-Карло* и *Лас-Вегас*.

Алгоритм Монте-Карло

Алгоритм Монте-Карло выдает решения «да/нет»: выход алгоритма является или *да*, или *нет*. Алгоритм *Монте-Карло, смещенный к ДА*, дает результат *ДА* с вероятностью 1 (нет ошибки); он дает результат *НЕТ* с вероятностью *e* (возможная ошибка). *Смещенный к НЕТ*, алгоритм *Монте-Карло*, выдаёт результат *НЕТ* с вероятностью 1 (нет ошибки); это дает результат *ДА* с вероятностью *e* (возможна ошибка). Мы рассмаривали вероятностные методыи в лекции 9.

Алгоритм Монте-Карло, смещенный к *ДА*, может проверить, является ли целое число простым. Если алгоритм возвращает «простое» мы уверены, что целое число является простым; если он возвращает «составной объект», число может быть простым с очень небольшой вероятностью.

Алгоритм Лас-Вегаса

Алгоритм Лас-Вегаса — алгоритм, который либо успешен, либо ошибочен. Если он успешен, он всегда возвращает правильный ответ. Если он терпит неудачу, никто за это не отвечает.

Приложение M. ZIP

PGP (лекция 16) использует методику сжатия данных ZIP, созданную Джином Гэйлеем, Марком Адиром и Ричардом Уользом. Она базируется на алгоритме, называемом LZ77 (Lempel-Ziv 77), который был изобретен Джакопом Зивом (Jacop Ziv) и Абрахамом Лемпэлем (Abraham Lempel). В этом приложении мы кратко обсуждаем LZ77 как основу ZIP.

M. 1. Кодирование LZ77

Кодирование LZ77 — пример **кодирования на основе словаря**. Идея в том, что нужно создать словарь (таблицу) строк, используемых в течение сеанса связи. Если и передатчик, и приемник имеют копию словаря, то уже встречавшиеся строки могут быть заменены их индексами в словаре, чтобы уменьшить количество переданной информации.

Хотя идея кажется простой, она довольно сложна в реализации. Первое: как создать словарь для каждого сеанса? Он не может быть универсальным из-за своей длины. Второе: как приемник может приобрести словарь, созданный передатчиком? Если вы передаете словарь, вы посылаете дополнительные данные, которые вредят цели сжатия.

Практический алгоритм, который использует идею адаптивного кодирования на основе словаря, — LZ77-алгоритм. Мы приводим основную идею этого алгоритма с примером, но не копаемся в деталях различных версий и реализаций. В нашем примере предположим, что нам нужно передать следующую строку. Мы выбрали эту строку, чтобы упростить обсуждение.

BAABABBBAABBBBAA

Для нашей простой версии LZ77-алгоритма процесс разделен на две фазы: сжатие строки и расширение (декомпрессия строки).

Сжатие

В этой фазе есть два параллельных события: создание индексированного словаря и сжатие строки символов. Алгоритм извлекает из памяти несжатую строку, у которой наименьшая подстрока — та, что не может быть найдена в словаре. Затем алгоритм сохраняет копию этой подстроки в словаре (как новый вход) и назначает значение индекса. Сжатие возникает, когда подстрока, если это не последний символ, заменена индексом, найденным в словаре. Процесс затем вставляет индекс и последний символ подстроки в сжатую строку. Например, если подстрока — ABBB, вы ищете в словаре ABB. Вы находите, что индекс для ABB — 4; сжатая подстрока поэтому будет 4B. Рисунок M.1 показывает процесс для нашей типовой строки.

Рассмотрим несколько шагов на рис. M.1.

- **Шаг 1.** Процесс извлекает из первоначальной строки наименьшую подстроку, который нет в словаре. Поскольку словарь пуст, наименьший символ —

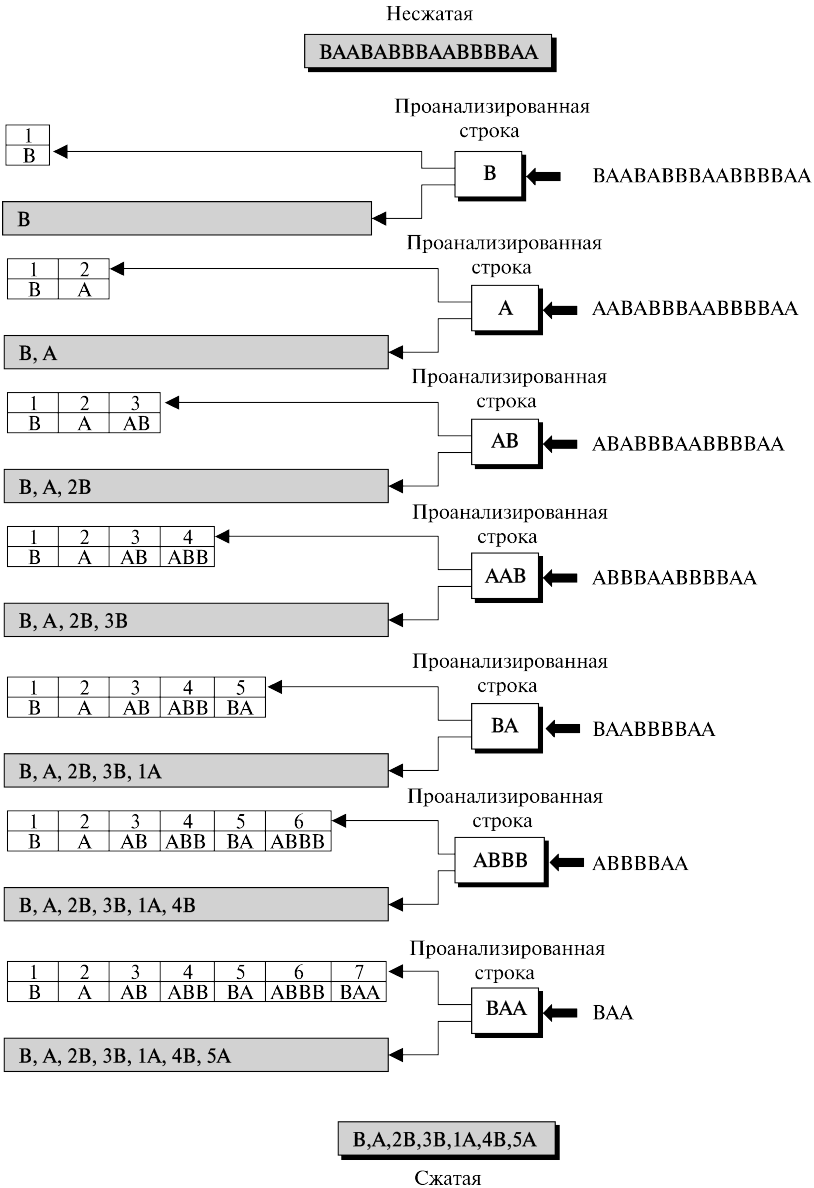


Рис. М.1. Пример LZ77 кодирование

один символ (первый символ B). Процесс хранит его копию как первый вход в этом словаре. Его индекс – 1. Часть этой подстроки не может быть заменена индексом из словаря (потому что это только один символ). Потому про-

цесс вставляет B (без индекса) в сжатую строку. Пока сжатая строка имеет только один символ: B. Первоначальная строка сохраняется не в сжатом виде — без первого символа.

- **Шаг 2.** Процесс извлекает из оставшейся строки следующую наименьшую подстроку, которой нет в словаре. Эта подстрока — символ A, который не находится в словаре. Процесс хранит копию этого символа как второй вход в словаре. Часть этой подстроки не может быть заменена индексом из словаря (потому что это только один символ). Процесс вставляет в сжатую строку A. Теперь сжатая строка имеет два символа: B и A (мы поставили запятые между подстроками в сжатой строке, чтобы показать разделение).
- **Шаг 3.** Процесс извлекает из остающейся строки следующую наименьшую подстроку, которой нет в словаре. Эта ситуация отличается от двух предыдущих шагов. Следующий символ (A) находится в словаре, так что процесс извлекает два символа (AB), а этой комбинации в словаре нет. Процесс сохраняет копию AB как третий вход в словарь. Процесс теперь находит индекс входа в словаре и создает сжатую подстроку без последнего символа (AB без последнего символа, т. е. без A). Индекс для комбинации сжатой строки — 2, так что процесс вставляет в сжатую строку комбинацию 2B.
- **Шаг 4.** Затем процесс извлекает подстроку ABB (потому что и AB уже в словаре есть). Копия ABB сохраняется в словаре с присвоением индекса 4. Процесс находит индекс подстроки без последнего символа (AB) — это 3. В сжатую строку вставляется комбинация 3B. Вы, возможно, заметили, что в трех предыдущих шагах мы фактически не достигли никакого сжатия, потому что заменили один символ одним (в первом A на A и B на B во втором шаге) и два символа на два (AB на 2B на третьем шаге). Но на этом шаге мы фактически уменьшили число символов (ABB стал 3B). Если первоначальная строка имеет много повторений (это предположение справедливо в большинстве случаев), мы можем заметно уменьшить число символов.

Каждый из остающихся шагов подобен одному из предыдущих четырех шагов, и мы предоставляем читателю возможность выполнить их самому. Обратите внимание, что словарь используется передатчиком, чтобы найти индексы. Его не передают приемнику; приемник должен создать словарь для себя сам, как мы увидим это в следующем разделе.

Декомпрессия

Декомпрессия — инверсия процесса сжатия. Процесс извлекает подстроки из сжатой строки и пробует заменять индексы соответствующими входами в словаре, который вначале является пустым и создается постепенно. Главное, что когда индекс получен, далее надо использовать вход словаря к соответствующему индексу. Рисунок М.2 показывает процесс декомпрессии.

Рассмотрим несколько шагов на рисунке М.2.

- **Шаг 1.** Анализируется первая подстрока сжатой строки. Это — B без индекса. Поскольку подстрока не находится в словаре, она добавляется к словарю.

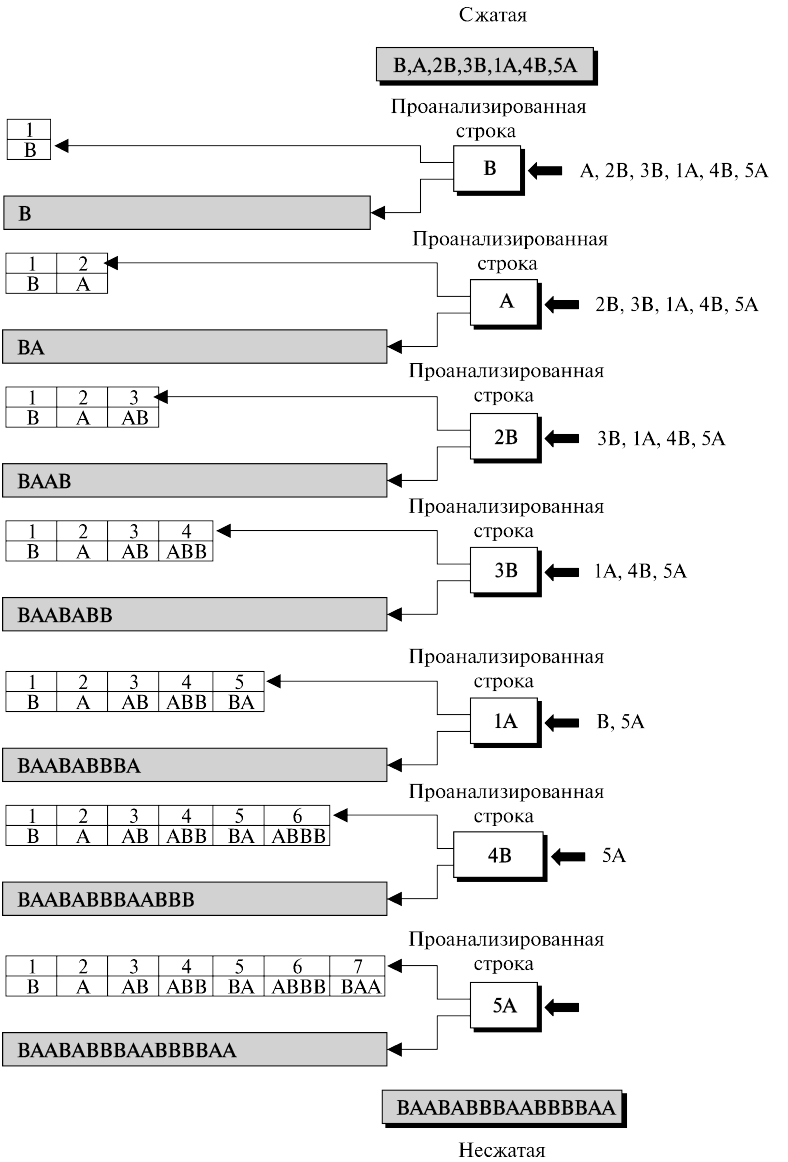


Рис. M.2. Пример LZ77-декодирования

Подстрока (B) вставляется в несжатую (декомпрессированную) строку.

- **Шаг 2.** Анализируется вторая подстрока (A); ситуация такая же, как и на шаге 1.

Теперь несжатая строка имеет два символа (BA), и словарь имеет два входа.

- **Шаг 3.** Анализируется третья подстрока (2B). Процесс ищет в словаре и заменяет индекс 2 подстрокой A. К несжатой строке добавляется новая подстрока (AB), и AB добавляется к словарю.
- **Шаг 4.** Анализируется четвертая подстрока (3B). Процесс ищет в словаре и заменяет индекс 3 с подстрокой AB. Подстрока ABB теперь добавляется к несжатой строке, и ABB добавляется к словарю.

Мы не будем делать анализ последних трех шагов, а добавим как задание для самостоятельных упражнений.

Как вы заметили, мы используем для обозначения индекса десятичные числа, такие, как 1 или 2. В действительности индекс записывается в двоичном виде (возможно, в данном случае длиной 3) для лучшей эффективности.

Приложение N. Дифференциальный и линейный криптоанализ DES

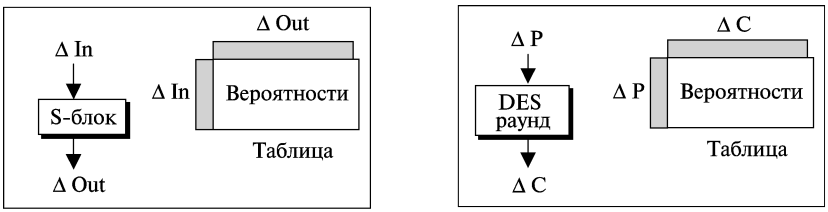
В этом приложении мы кратко обсуждаем две проблемы, которые связаны с шифром DES, рассмотренным в лекции 6: дифференциальный и линейный криптоанализ. Полное рассмотрение этих двух проблем невозможно в этой книге. Приложение дает общее представление и отсылает заинтересованных читателей к возможностям познакомиться с проблемой поближе в соответствующей литературе.

N.1. Дифференциальный криптоанализ

Дифференциальный криптоанализ для DES был изобретен Бихамом (Biham) и Шамиром (Shamir). В этом криптоанализе злоумышленник концентрируется на *атаках с выборкой исходного текста*. Анализ использует *разность* в прохождении различных входных сигналов через устройство или программу шифрации. Термин *разность* здесь применяется, чтобы рассмотреть с помощью операции ИСКЛЮЧАЮЩЕЕ ИЛИ несовпадение двух различных входных сообщений (исходные тексты). Другими словами, злоумышленник анализирует, как P ⊕ P' различаются при обработке в каждом раунде.

Вероятностные отношения

Идея относительно дифференциального криптоанализа базируется на вероятностных отношениях между входными разностями и разностями выхода. Два отношения представляют конкретный интерес в анализе: *дифференциальный профайл* и *характеристика раунда*, как это показано на рис. N.1.



а. Дифференциальный профайл б. Характеристика раунда

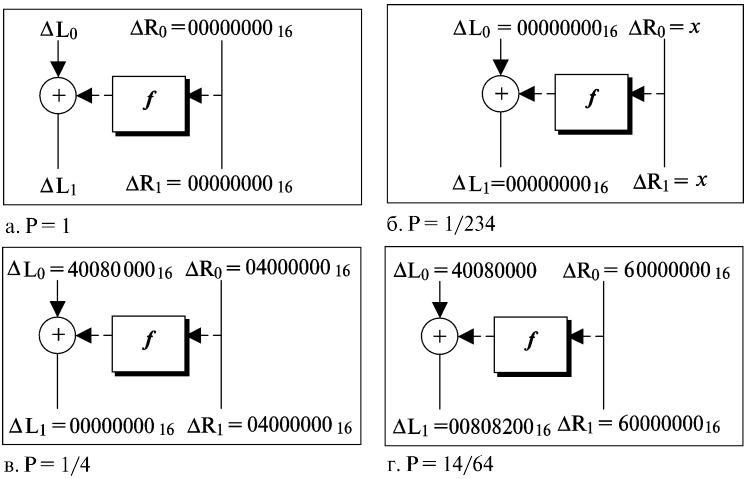
Рис. N.1. Дифференциальный профайл и характеристика раунда в DES

Дифференциальный профайл

Дифференциальный профайл (он же профайл ИСКЛЮЧАЮЩЕЕ ИЛИ) показывает вероятностное отношение между входными разностями и разностями выхода S-блока. Мы обсуждали этот профайл для простого S-блока в лекции 5 (см. таблицу 5.5). Подобные профайлы могут быть созданы для каждого из восьми S-блоков в DES.

Характеристика раунда

Характеристика раунда подобна дифференциальному профайлу, но вычисляется для целого раунда. Она показывает вероятность, с которой одна входная разность создала бы разность определенного выхода. Обратите внимание, что характеристика одна и та же для каждого раунда, потому что любое отношение, которое включает разности, не зависит от ключей раунда. Рисунок N.2 показывает четыре различные характеристики раунда.



а. P = 1 б. P = 1/234

в. P = 1/4 г. P = 14/64

Рис. N.2. Некоторые характеристики раунда для дифференциального криптоанализа

Хотя существует много характеристик для раунда, рисунок N.2 показывает только четыре из них. В каждой характеристике мы разделили входные разности и разности выхода в левые и правые секции. Каждая левая или правая разность состоят из 32 битов или восьми шестнадцатеричных цифр. Все эти характеристики могут быть найдены использующими программами, которые могут найти отношение входа-выхода в раунде DES. Рисунок N.2а показывает, что входная разность (x, 00000000_{16}) дает на выходе разность (x, 00000000_{16}) с вероятностью 1. Рисунок N.2б показывает ту же самую характеристику, как рисунок N.2а, за исключением того, что левые и правые вход и выход поменялись местами; вероятность изменится чрезвычайно. Рисунок N.2в показывает, что входная разность (40080000_{16}, 04000000_{16}) дает разность выхода (00000000_{16}, 0400000_{16}) с вероятностью 1/4. Наконец, рисунок N.2г показывает, что входная разность (00000000_{16}, 60000000_{16}) дает разность выхода (00808200_{16} 6000000_{16}) с вероятностью 14/64.

Трехраундная характеристика

После создания и хранения однораундных характеристик анализатор может комбинировать различное количество раундов, чтобы создать множественную характеристику раунда. Рисунок N.3 показывает случай трехраундной DES.

На рис. N.3, мы использовали три смесителя и только два устройства замены, потому что последний раунд не нуждается ни в каком устройстве замены, как уже говорилось в лекции 5. Характеристики, показанные в смесителях первых и третьих раундов, те же самые, как и на рисунке N.2b. Характеристика смесителя во втором раунде — та же самая, что и на рис. N.2a. Очень интересно отметить, что точки, в этом конкретном случае, разности входа и выхода — те же самые ($\Delta L_3 = \Delta L_0$ и $\Delta R_3 = \Delta R_0$).

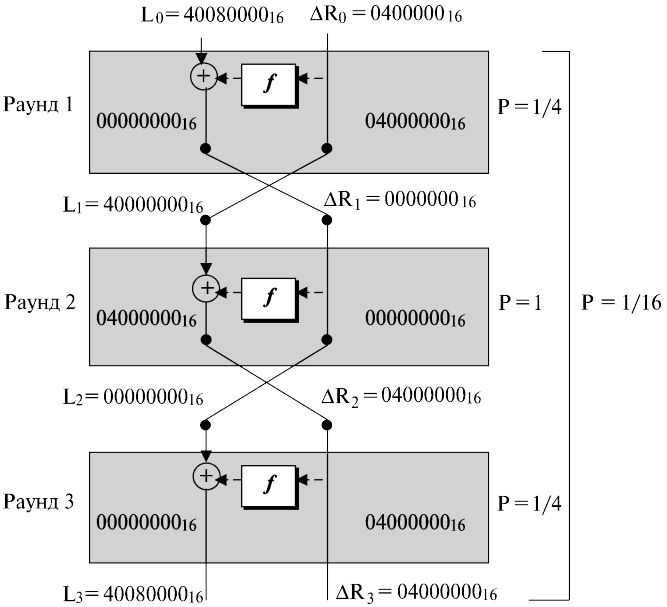


Рис. N.3. Трехраундная характеристика

Шестнадцатираундная характеристика

Для шифра с шестнадцатью раундами можно скомпилировать много различных характеристик. Рисунок N.4 показывает пример. На этом рисунке шифр DES состоит из восьми секций с двумя раундами. Каждая секция использует характеристики *а* и *б* на рис. N.2. Ясно, что если последние раунды не имеют устройства замены, вход *(x,* 0) создает выход (0, *x)* с вероятностью (1/234) [8].

Атака

Для примера предположим, что Ева использует характеристику по рисунку N.4, чтобы напасть на DES с шестнадцатью раундами. Ева каким-то способом провоцирует Алису, чтобы зашифровать много исходных текстов в форме (*x,* 0), в которой левая половина — *x* (различные значения) и правая половина — 0. Ева затем сохраняет все зашифрованные тексты, полученные от Алисы, в форме (0, *x*). Обратите внимание, что 0 здесь означает 00000000_{16}.

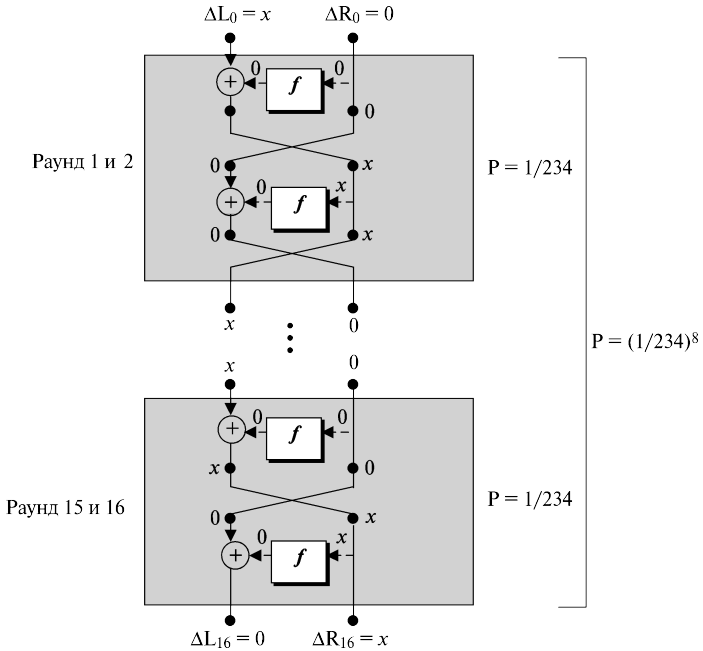


Рис. N.4. Шестнадцатираундная характеристика для дифференциального криптоанализа

Нахождение ключа шифра

Окончательная цель злоумышленника в дифференциальном криптоанализе состоит в том, чтобы найти ключ шифра. Для этого нужно найти ключи каждого раунда от основания до вершины (от K_{16} до K_1).

Нахождение последних ключей раунда

Если злоумышленник имеет достаточно много пар исходного текста / зашифрованного текста (каждый с различными значениями x), он может использовать отношения в последнем раунде, $0 = f(K_{16},x)$ и найти некоторые из битов в K_{16}. Это можно сделать, выбирая самые вероятные значения.

Нахождение других ключей раунда

Ключи для других раундов можно найти, используя другие характеристики или применяя атаки грубой силы.

Безопасность

Известно, что необходимы 2^{47} выборки пар исходного текста / зашифрованного текста, чтобы напасть на DES с 16 раундами. Найти такое огромное число

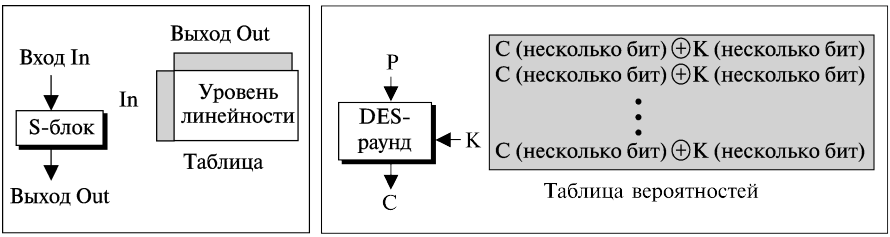
выбранных пар чрезвычайно трудно в ситуациях реальной жизни. Это означает, что DES неуязвимы для этого типа атаки.

N.2. Линейный криптоанализ

Линейный криптоанализ для DES был разработан Матцуи. Это — *атака знания исходного текста*. Анализ использует распространение конкретного набора битов через устройство шифрования.

Отношения линейности

Линейный криптоанализ основан на отношениях линейности. В этом типе криптоанализа представляют интерес два набора отношений: линейные профайлы и характеристики раунда, как показано на рис. N.5.



а. Дифференциальный профайл б. Характеристика раунда

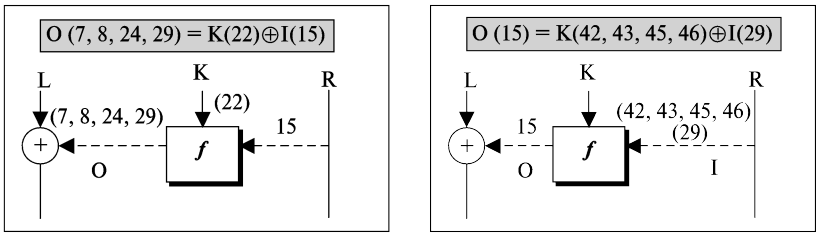
Рис. N.5. Линейный профайл и характеристика раунда в DES

Линейный профайл

Линейный профайл показывает уровень линейности между входом и выходом S-блока. Мы видели в лекции 5, что в S-блоке каждый бит выхода — функция всех входных битов. Желательное свойство в S-блоке достигнуто, если каждый бит выхода — нелинейная функция всех входных битов. К сожалению, эта идеальная ситуация не существует в DES; некоторые биты выхода — линейная функция некоторых комбинаций входных битов. Другими словами, можно найти, что некоторые комбинации битов входа-выхода могут быть отображены между собой, используя линейную функцию. Линейный профайл показывает уровень линейности (или нелинейности) между входом и выходом. Криптоанализ может создать восемь различных таблиц, по одной для каждого S-блока, в которых первый столбец показывает возможные комбинации входов по шесть бит, 00_{16} до $3F_{16}$. Первая строка показывает возможные комбинации выходов по четыре бита, 0_{16} до F_{16}. Входы показывают уровень линейности (или нелинейности) данного проекта. Мы не можем углубляться в детали того, как измеряется уровень линейности, но входы с высокого уровня из линейности интересны для криптоанализа.

Характеристика раунда

Характеристика раунда в линейном криптоанализе показывает комбинации входных битов, битов ключей раунда и битов выхода для того, чтобы определить линейное отношение. Рисунок N.6 изображает две различные характеристики раунда. Система обозначений, используемая для каждого случая, определяет биты, которые складываются по модулю два. Например, O (7, 8, 24, 29) означает операцию исключающее ИЛИ 7-х, 8-х, 24-х и 29-х битов, выходящих из функции; K (22) означает 22-й бит в ключе раунда; I (15) означает 15-й бит, входящий в функцию.



a. P=52/64 b. P=42/64

Рис. N.6. Некоторые характеристики раунда для линейного криптоанализа

Ниже показаны отношения для частей а и б рисунка N.6, использующих индивидуальные биты.

Часть a: O (7) ⊕ O (8) ⊕ O (24) ⊕O (29) = I (15) ⊕ K (22)
Часть b: F (15) = I (29) ⊕ K (42) ⊕ K (43) ⊕ K (45) ⊕ K (46)

Трехраундная характеристика

После создания и хранения однораундных характеристик анализатор может комбинировать различные раунды, чтобы создать множественную характеристику раунда. Рисунок N.7 показывает случай трехраундной DES, в которой раунды 1 и 3 используют одну и ту же характеристику, как это изображено на рис. N.6a, а в раунде 2 использована произвольная характеристика.

Цель линейного криптоанализа состоит в том, чтобы найти линейное отношение между некоторыми битами в паре «исходный текст / зашифрованный текст» и ключ. Давайте посмотрим, можем ли мы установить такое отношение для DES с 3-мя раундами, изображенной на рис. N.7.

Раунд 1: R_1 (7, 8, 24, 29) = L_0 (7, 8, 24, 29) ⊕ R_o (15) ⊕K_1 (22)
Раунд 3: L_3 (7, 8, 24, 29) = L_2 (7, 8,24, 29) ⊕ R2 (15) ⊕ K_3 (22)

Но L_2 — тот же самый, что и R_1, и R_2 — тот же самый, что и R_3. После замены L_2 на R_1 и R_2 на R_3 во втором отношении мы получим:

L_3 (7, 8, 24, 29) = R_1, (7, 8, 24, 29) ⊕ R_3 (15) ⊕ K_3 (22)

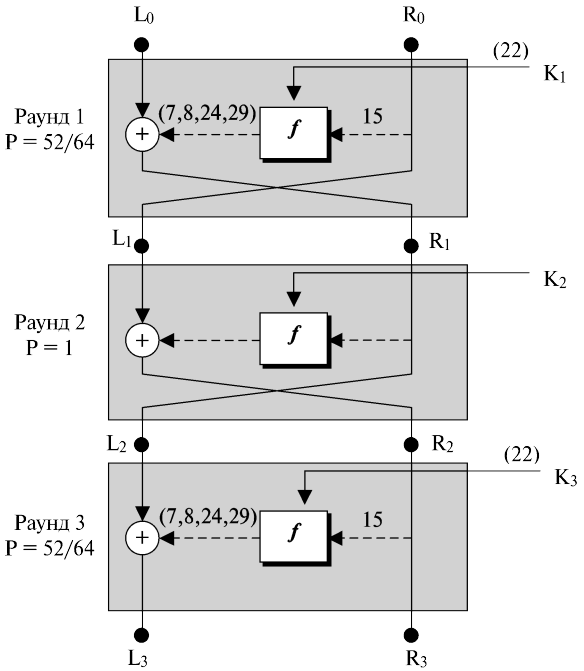


Рис. N.7. Трехраундные характеристики для линейного криптоанализа

Мы можем заменить R_1 на его эквивалентное значение в раунде 1, в результате имеем

$$L_3(7, 8, 24, 29) = L_0 (7, 8, 24, 29) \oplus R_0 (15) \oplus K_1 (22) \oplus R_3 (15) \oplus K_3 (22)$$

Отношения между битами входа и выхода для всей системы из трех раундов после преобразований:

$$L_3 (7, 8, 24, 29) \oplus R_3 (15) = L_0 (7, 8, 24, 29) \oplus Ro (15) \oplus K_1 (22) \oplus K_3 (22)$$

Другими словами, мы имеем

$$C (7, 8, 15, 24, 29) = P (7, 8, 15, 24, 29) \oplus K_1 (22) \oplus K_3 (22)$$

Вероятность

Один интересный вопрос: как найти вероятность трехраундных (или n-раундных) DES. Матцуи (Matsui) показал, что вероятность в этом случае

$$\mathbf{P = 1/2 + 2^{n-1} \prod (p_i - 1/2),}$$

где n является числом раундов, P — вероятность каждой характеристики раунда и P — полная вероятность. Например, полная вероятность для трехраундного анализа на рис. N.7:

$$P = 1/2 + 2^{3-1}[(52/64 - 1/2) \times (1 - 1/2) \times (52/64 - 1/2)] \approx 0{,}695$$

Шестнадцатираундная характеристика

16-раундовая характеристика может также быть скомпилирована, чтобы обеспечить линейные отношения между некоторыми битами исходного текста, некоторыми битами зашифрованного текста и некоторыми битами в ключах раунда.

$$C(\text{некоторые биты}) = P(\text{некоторые биты}) \oplus K_1(\text{некоторые биты}) \oplus \ldots \oplus K_{16}(\text{некоторые биты})$$

Атака

После нахождения и сохранения многих отношений между некоторыми битами исходного текста, битами зашифрованного текста и битами ключей раунда Ева может обратиться к некоторым парам исходного текста / зашифрованного текста (атака знания исходного текста) и использовать соответствующие биты из сохраненных характеристик, чтобы найти биты в ключах раунда.

Безопасность

Известно, что для того чтобы напасть на 16-раундовый DES, необходимы 2^{43} известных пар исходного текста / зашифрованного текста. Линейный криптоанализ выглядит более вероятным, чем дифференциальный криптоанализ, по двум причинам. Первая: число шагов у него меньше. Вторая: он более прост для атаки знания исходного текста, чем для атаки с выборкой исходного текста. Однако и такая атака все еще далека от того, чтобы ее серьезно опасаться тем, кто работает с DES.

Приложение О. Упрощенный DES (S-DES)

Упрощенный DES (S-DES) разработан профессором Эдуардом Шаефером (Edward Schaefer) Университета Санта-Клары и является образовательным инструментом для помощи студентам при изучении структуры DES — для шифрования и дешифрования с использованием блочных шифров и ключей с небольшим количеством битов. Читатели могут изучать это приложение перед чтением лекции 6.

О.1. Структура S-DES

S-DES — блочный шифр, как это показано на рис. О.1.

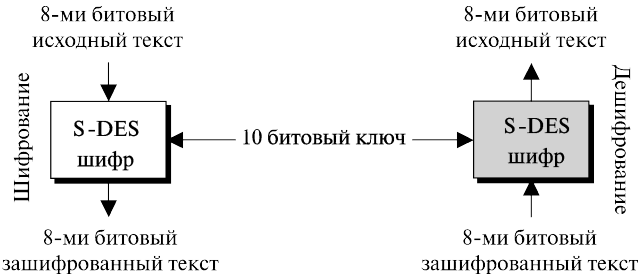


Рис. О.1. Шифрование и дешифрование в S-DES

На стороне шифрования S-DES принимает исходный текст по 8 битов и создает зашифрованный текст по 8 битов; на стороне дешифрования S-DES принимает зашифрованный текст на 8 битов и создает 8-битовый открытый текст. Один и тот же ключ шифра на 10 битов используется и для шифрования, и для дешифрования.

Остановимся на шифровании, а потом обсудим дешифрование. Процесс шифрования состоит из двух перестановок (P-блоки), которые мы называем начальными и конечными перестановками (их также называют IP и IP^{-1}), и двух раундов Файстеля. Каждый раунд использует различные ключи раунда по 8 битов, сгенерированные от ключа шифра согласно заранее заданному алгоритму, описанному позже в этом приложении. Рисунок О.2 показывает элементы шифра S-DES на стороне шифрования.

Начальная и конечная перестановки

Рисунок О.3 показывает начальные и конечные перестановки (P-блоки). Каждая из этих перестановок получает входную информацию на 8 битов и переставляет их согласно заранее заданному правилу. Эти перестановки — прямые перестановки, которые инверсны друг другу, как это было сказано в лекции 5. Эти две перестановки не имеют никакого криптографического значения в S-DES. Они включены в S-DES, чтобы совместить их с полным DES.

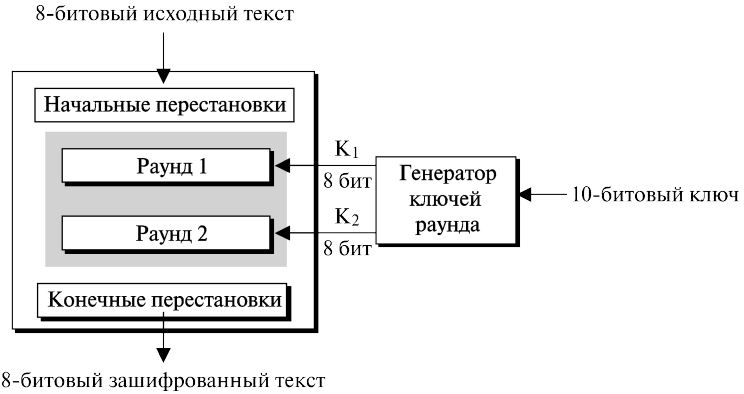


Рис. О.2. Общая структура S-DES устройства шифрования

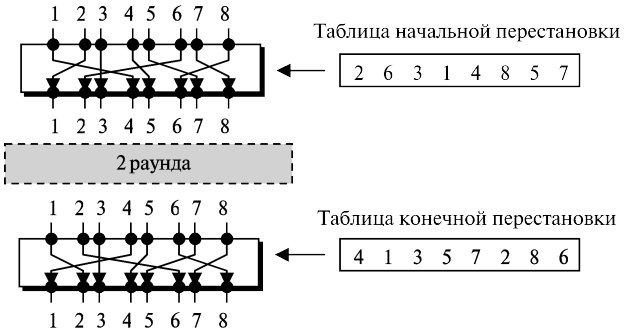


Рис. О.3. Начальная и конечная перестановки (IP и IP^{-1})

Раунды

S-DES использует два раунда. Каждый раунд S-DES — это шифр Файстеля, как это показано на рис. О.4

Раунд получает L_{I-1} и R_{I-1} от предыдущего раунда (или начального блока перестановки) и создает L_I и R_I, которые поступают в следующий раунд (или конечный блок перестановки). Как мы говорили в лекции 5, мы можем принять, что каждый раунд имеет два элемента шифра: смеситель и устройство замены. Каждый из этих элементов является обратимым. Устройство замены, очевидно, является обратимым. Оно меняет левую половину текста с правой половиной. Смеситель является обратимым из-за операции ИСКЛЮЧАЮЩЕЕ ИЛИ. Все необратимые элементы собраны в функции $f(R_{I-1}, K_I)$.

Функция S-DES

Основа S-DES — функция S-DES. Функция S-DES применяет ключ на 8 битов к самым правым 4 битам (R_{I-1}), чтобы формирует выход на 4 бита. Эта функ-

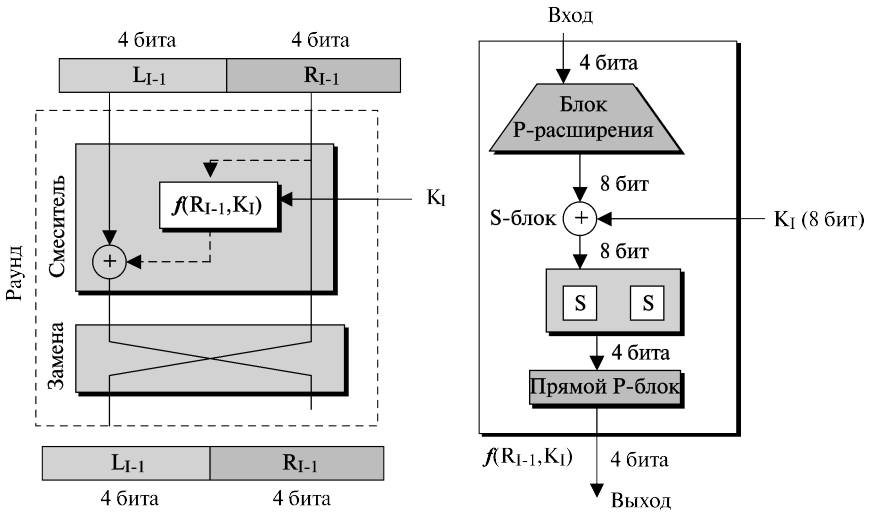


Рис. О.4. Раунд S-DES (сторона шифрования)

ция составлена из четырех секций: P-блока расширения, отбеливателя, группы S-блоков и прямого P-блока, как показано на рис. О.4.

P-блок расширения — R_{I-1} вход на 4 бита, и K_I — ключ на 8 битов, так что мы сначала должны расширить R_{I-1} до 8 битов.

Хотя отношения между входом и выходом могут быть определены математически, S-DES использует таблицу, чтобы определить P-блок, как это показано на рис. О.5. Обратите внимание, что число портов выхода — 8, но диапазон значения — только 1-4. Некоторые из входов идут больше чем в один выход.

Отбеливатель (ИСКЛЮЧАЮЩЕЕ ИЛИ). После перестановки расширения S-DES используют операцию ИСКЛЮЧАЮЩЕЕ ИЛИ для сложения расширенной правой секции и ключа раунда. Обратите внимание, что ключи раунда используются только в этой операции.

S-блоки. S-блоки делают реальное смешивание (перемешивание). S-DES используют два S-блока, каждый с входом на 4 бита и выводом на 2 бита (см. рис. О.6).

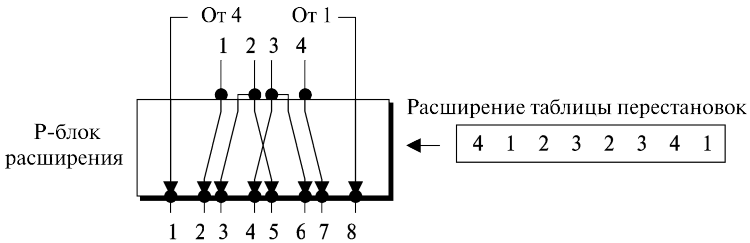
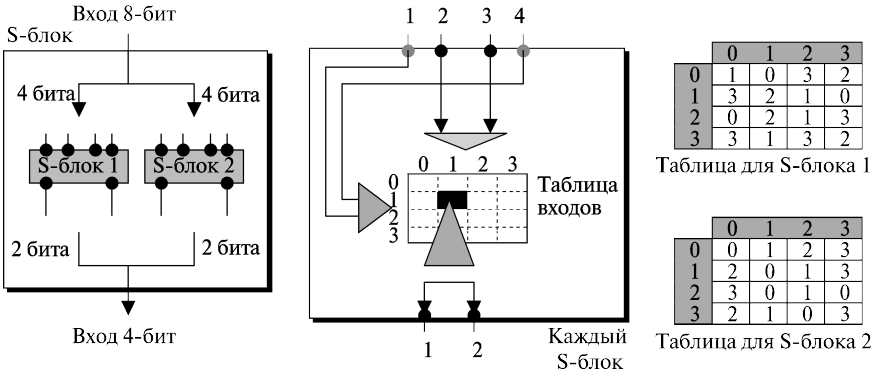


Рис. О.5. P-блок расширения



	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

Таблица для S-блока 1

	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

Таблица для S-блока 2

Рис. О.6. S-блоки

Данные по 8 битов от второй операции разделены на две части по 4 бита, и каждый кусок подается в свой блок. Результат каждого блока — отрезок на 2 бита; когда они объединяются, в результате получается текст на 4 бита. Подстановка в каждом блоке соответствует заранее определенным правилам, основанным на записях в таблицах 4 x 4. Комбинация входных битов 1 и 4 определяет одну из четырех строк; комбинация битов 2 и 3 определяет один из четырех столбцов, как показано на рис. О.6.

Поскольку каждый S-блок имеет свою собственную таблицу, необходимы две таблицы, как это видно на рис. О.6, чтобы определить выход этих блоков. Значения входов (номер строки и номер столбца) и значения выходов на рисунке даются в виде десятичных чисел, чтобы сохранить пространство. Они должны быть изменены на двоичные значения.

Пример 0.1

На вход S-блока 1 подается число 1010_2. Каким будет выход?

Решение

Если мы запишем вместе первый и четвертый биты, мы получим 10 в двоичном виде или 2 в десятичном числе. Остающиеся биты — 01 в двоичном виде, — это 1 в десятичном числе. Мы ищем последовательно значение 2, столбец 1, на рис. 0.6 (S-блок 1). Результат в десятичном виде — 2, а в двоичном — это 10. Таким образом, вход 1010_2 дает выход 10_2.

Прямая перестановка. Последняя операция в функции S-DES — прямая перестановка с входом на 4 бита и выходом на 4 бита. Отношения входа-выхода для этой операции показаны на рисунке О.7 и следуют тем же общим правилам, что и предыдущие таблицы перестановки.

Генерация ключей

Генератор ключей раунда создает два ключа на 8 битов из ключа шифра на 10 битов.

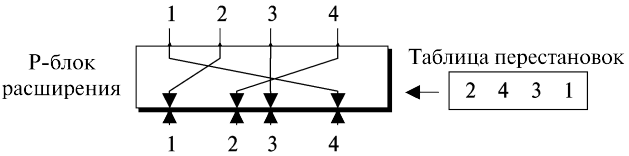


Рис. О.7. Р-блок прямой перестановки

Прямая перестановка

Первый процесс — прямая перестановка. Этот процесс переставляет 10 битов в ключе согласно заранее заданной таблице, как это показано на рис. О.8.

Сдвиг влево

После прямой перестановки ключ разделяется на две части по 5 битов. Каждая часть сдвигается влево (циклический сдвиг) на *r* бит, где *r* — номер раунда

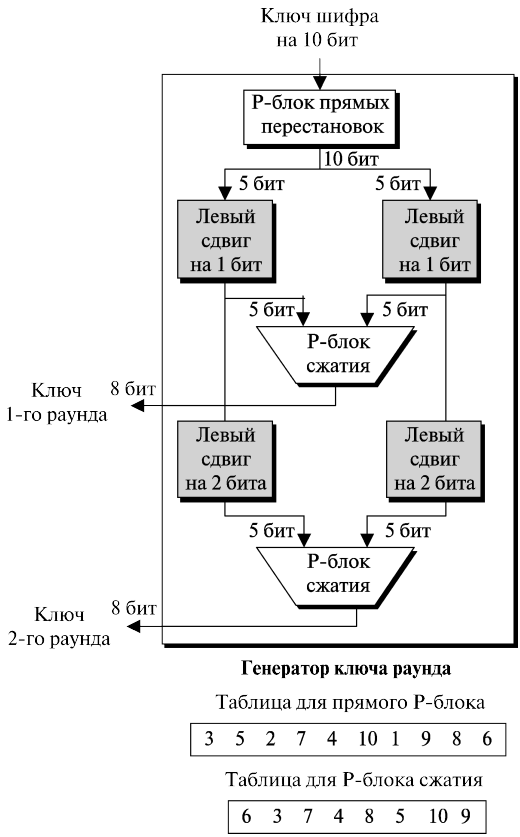


Рис. О.8. Генератор ключей

(1 или 2). Эти две части затем объединяются, чтобы сформировать модуль на 10 битов (см. лекцию 5 для обсуждения операции сдвига).

Перестановка сжатия

Перестановка сжатия (P-блок) изменяет 10 битов на 8 битов, которые используются как ключ для раунда. Таблица перестановки сжатия показана на рис. О.8.

Пример О.2

Таблица О 1 приводит три случая генерации ключей.

Таблица О.1

Шаги	**Случай 1**	**Случай 2**	**Случай 3**
Ключ шифра	**1011100110**	**0000000000**	**1111111111**
после перестановки	1100101110	0000000000	1111111111
После разделения	L: 11001 R:01110	L: 00000 R:00000	L: 11111 R: 11111
Раунд 1:			
Сдвинутые ключи:	L:10011 R:11100	L:00000 R:00000	L:11111
Комбинация ключей:	1001111100	0000000000	R:111111111111111
Ключ раунда 1:	**10111100**	**00000000**	**11111111**
Раунд 2:			
Сдвинутые ключи:	L: 01110 R: 10011	L: 00000 R: 00000	L:11111 R 11111
Комбинация ключей:	0111010011	0000000000	1111111111
Ключ раунда 2:	**11010011**	**00000000**	**11111111**

Случаи 2 и 3 показывают, что ни одна из операций, используемых в процессе генерации ключей, не эффективна, если ключ шифра состоит из всех нулей или всех единиц. Нужно избегать такого типа ключей шифра, как уже обсуждалось в лекции 6.

S-DES очень уязвим к атаке грубой силы из-за размера ключа (10 битов).

О.2. Шифр и обратный шифр

Используя смесители и устройства замены, мы можем создать шифр и обратный шифр в два раунда. Шифр применяется на стороне шифрования; обратный шифр — на стороне дешифрования. Чтобы сделать шифр и обратный шифр, используются похожие алгоритмы. Раунд 2 имеет только смеситель и не имеет устройства замены. Это показано на рис. О.9.

Мы говорили в лекции 5, что смеситель самообратим, так же как и устройство замены. Конечные и начальные перестановки также инверсны друг другу. Левая часть исходного текста на стороне шифрования, l_0, зашифрована как L_2; L_2 на стороне дешифрования расшифрована как l_0. Аналогичная ситуация — с правой частью.

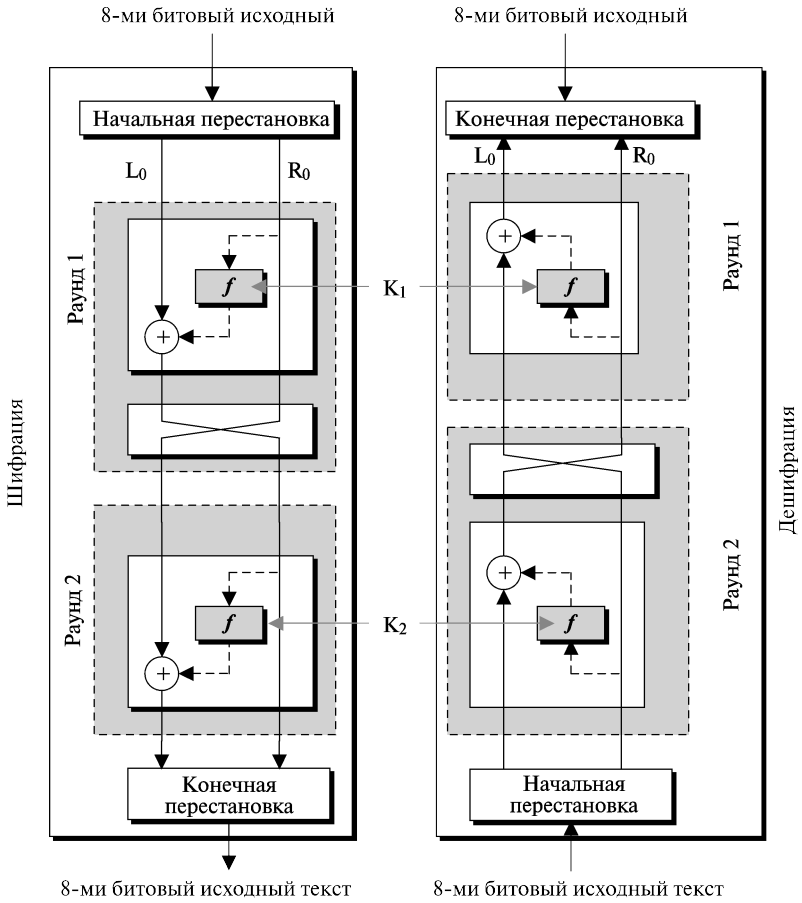


Рис.О.9. S-DES шифр и обратный шифр

Очень важно запомнить, что ключи шифров раунда (K_1 и K_2), должны использоваться в обратном порядке. На стороне шифрования в раунде 1 применяют K_1, а в раунде 2 применяют K_2; на стороне дешифрования в раунде 1 нужен K_2, а в раунде 2 — K_1.

Во втором раунде нет устройства замены.

Пример О.3

Мы выбираем случайный блок исходного текста и случайный ключ. Определяем, какой блок зашифрованного текста получится, если

Исходный текст: 11110010 Ключ:1011100110 Зашифрованный текст: 11101011

Посмотрим результат каждого раунда и текста, созданного до и после раундов. Таблица О.2 показывает результат шагов перед началом раунда. Исходный текст проходит начальную перестановку, чтобы создать различные 8 битов. После этого шага текст разбит на два части, L_0 и R_0. Таблица показывает результаты двух раундов, которые включают смешивание и заполнение (исключая второй раунд). Результаты последних раундов (L_2 и R_2) объединены.

Наконец, текст проходит конечную перестановку, чтобы получить зашифрованный текст.

Таблица О.2

Начальная обработка	Исходный текст: **11110010** После IP:10111001 L_0: 1011 R_0: 1001	Ключ шифра: **1011100110**
Раунд 1	L_1: 1001 R_1: 0111	Ключ раунда: 10111100
Раунд 2	L_2: 1011 R_2 : 0 111	Ключ раунда: 11010011
Конечная обработка	Перед IP^{-1} 10110111 Зашифрованный текст: **11101011**	

Стоит упомянуть здесь некоторые важные положения. Сначала правая часть каждого раунда — та же самая, что и левая секция из следующего раунда. Причины в том, что правая часть проходит смеситель без изменения, но устройство замены перемещает ее левую часть. Например, R_1 проходит через смеситель второго раунда без изменения, но затем эта часть становится L_2 из-за устройства замены. Интересно то, что эта точка не имеет устройства замены в последнем раунде. Поэтому R_1, становится R_2 вместо того чтобы стать L_2.

Из-за малого количества раундов S-DES более уязвим к криптоанализу, чем DES.

Приложение P. Упрощенный AES (S-AES)

P.1. Структура S-AES

Упрощенный AES (S-AES), разработан профессором Эдвардом Шаефером (Edward Schaefer) в Университете Санта-Клары, является образовательным инструментом и предназначен помочь студентам изучать структуру AES с использованием меньших блоков и ключей. Читатели могут изучить это приложение перед чтением лекции 7.

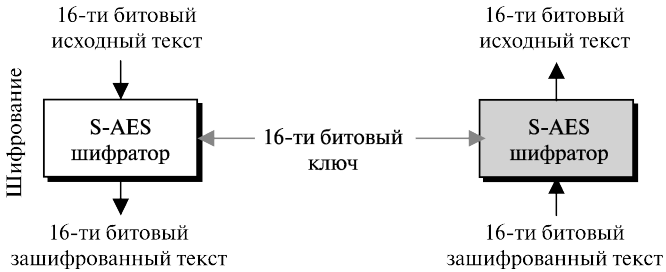


Рис. P.1 Шифрация и дешифрация в S-AES

На стороне шифрования S-AES получают исходный текст на 16 битов и создают зашифрованный текст на 16 битов; на стороне дешифрации S-AES получают зашифрованный текст на 16 битов и создают исходный текст на 16 битов. Ключ шифра на 6 битов используется и для шифрования, и для дешифрования.

Раунды

S-AES —шифр не-Файстеля, который зашифровывает и расшифровывает блок данных в 16 битов. Он использует предварительный раунд преобразования и два раунда. Ключ шифра — также 16 битов. Рисунок P.2 показывает общий вид для алгоритма шифрования (называемого «шифр») и алгоритма дешифрования (называемого «обратный шифр»). Они совпадают, но ключи раунда применяются в обратном порядке.

На рисунке P.2 ключи раунда, которые созданы алгоритмом расширения ключей, всегда содержат 16 бит, — они такого же размера, как и блок зашифрованного текста или исходный текст. В S-AES есть три ключа раунда — K_0, K_1 и K_2.

Модули данных

Для представления данных S-AES, как показано на рис. P.3, используют пять модулей измерения: биты, полубайты, слова, блоки и состояния.

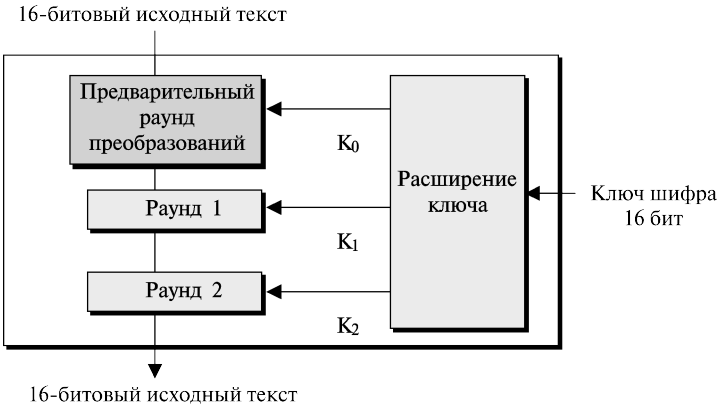


Рис. Р.2. Общий вид блока шифрования S-AES

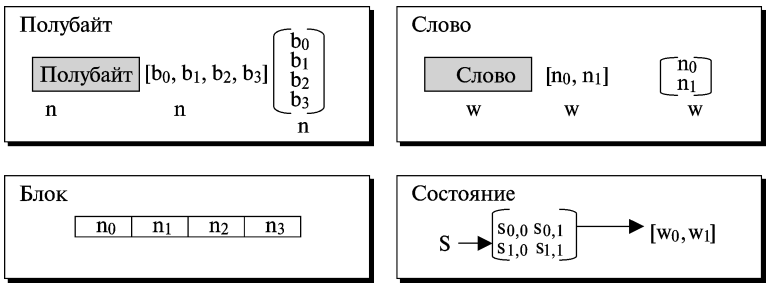


Рис. Р.3. Блоки данных используемых в S-AES

Бит

В S-AES *бит* — двоичная цифра, которая принимает значение либо 0, либо 1. Для обозначения бита мы будем применять латинскую букву b.

Полубайт

Полубайт — группа из 4 битов, которая может быть обработана как один объект, матрица-строка на 4 бита или матрица-столбец на 4 бита. Когда полубайт обрабатывается как матрица-строка, биты располагаются в матрице слева направо; когда полубайт обрабатывается как матрица-столбец, биты располагаются в матрице сверху вниз. Чтобы обозначить полубайт, мы используем строчную полужирную букву **n**. Обратите внимание, что полубайт — фактически одна шестнадцатеричная цифра.

Слово

Слово — группа 8 битов, которая может быть обработана как один объект: матриц-строка из двух полубайт или матрица-столбец из двух полубайт. Когда

слово обрабатывается как матрица-строка, полубайты располагаются в матрице слева направо; когда слово рассматривается как матрица-столбец, полубайты располагаются в матрице сверху вниз. Чтобы обозначить слово, мы применяем строчную полужирную букву **w**.

Блок

S-AES шифрует и расшифровывает блоки данных. *Блок* в S-AES – группа из 16 битов. Однако блок может быть представлен как матрица-строка из 4 полубайтов.

Состояние

В S-AES блок данных называется также *состоянием*. Мы используем полужирную заглавную букву **S**, чтобы обозначить состояние. Состояния, подобно блокам, состоят из 16 битов, но обычно они обрабатываются как матрицы по 4 полубайта. В этом случае каждый элемент состояния обозначается как $s_{r,c}$, где r (0 или 1) определяет строку и c (0 или 1) определяет столбец. В начале шифра полубайты в блоке данных состояния располагаются столбец за столбцом и в каждом столбце сверху вниз. В конце шифра полубайты состояния извлекаются тем же способом, как показано на рис. P.4.

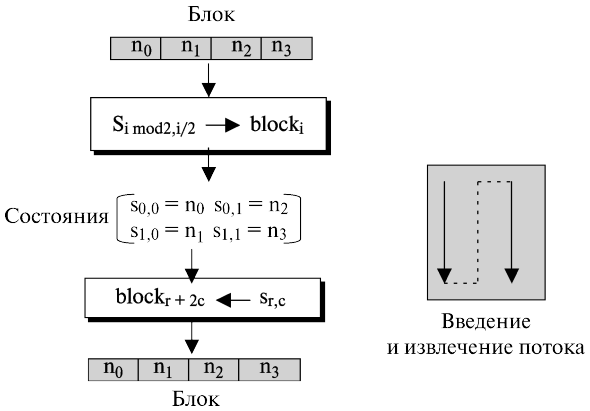


Рис. P.4. Преобразования «Блок – состояние» и «состояние – блок»

Пример P.1

Давайте посмотрим, как можно блок в 16 битов представить как матрицу 2 × 2. Пусть текстовый блок имеет вид 1011 0111 1001 0110. Сначала мы представим блок как 4 полубайта. Затем заполняем матрицу состояний, «столбец за столбцом», как это показано на рис. P.5.

Структура каждого раунда

Рисунок P.6 показывает, что каждое преобразование получает состояние и создает другое состояние, которое используется для следующего преобразования

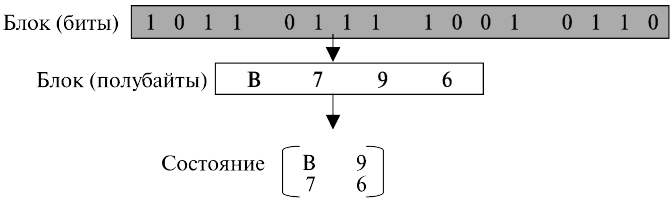


Рис. Р.5. Изменение шифрованного текста в состояние

или следующего раунда. Секция предварительного преобразования (перед раундом) применяет одно преобразование (AddRoundKey); последний раунд использует только три преобразования (преобразование MixColumns отсутствует).

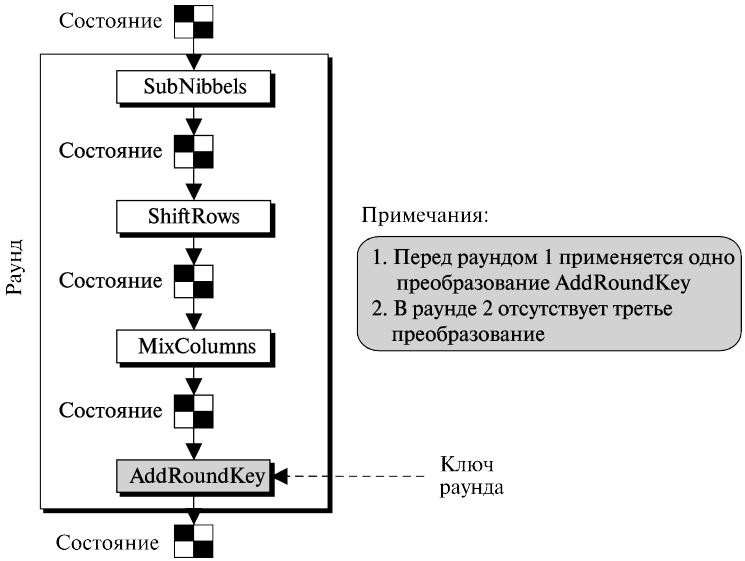


Рис. Р.6. Структура каждого раунда на стороне шифрования

На стороне дешифрования используются: обратные преобразования InvSubNibbles, InvShiftRows, InvMixColumns и AddRoundKey (этот процесс является самообратимым).

Р.2. Преобразования

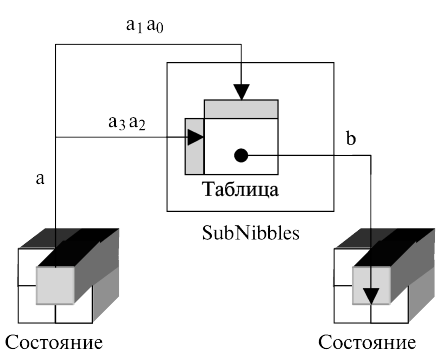
Чтобы обеспечить безопасность, S-AES применяют четыре типа преобразований: подстановка, перестановка, смешивание и сложение ключа. Ниже мы обсудим каждое из них.

Подстановка

Подстановка сделана для каждого полубайта (модуль данных на 4 бита). Для преобразований каждого полубайта используется только одна таблица, что означает, что если два полубайта одинаковые, то и результат преобразования одинаковый. В этом приложении преобразование определено процессом поиска таблицы.

Подполубайты

Первое преобразование, **SubNibbles**, применяется на стороне шифрования. Для того чтобы заменять полубайты, мы интерпретируем каждый полубайт как 4 бита. Левые 2 бита определяют строку, а правые 2 бита определяют столбец таблицы подстановки. Шестнадцатеричная цифра в пересечении строки и столбца — новый полубайт. Рисунок P.7 показывает идею.



$a_1 a_0$

$a_3 a_2$	00	01	10	11
00	9	4	A	B
01	D	1	8	5
10	6	2	0	3
11	C	E	F	7

Таблица SubNibbles

$a_1 a_0$

$a_3 a_2$	00	01	10	11
00	A	5	9	B
01	1	7	8	F
10	6	0	2	3
11	C	4	D	E

Таблица InvSubNibbles

Рис. P.7. Преобразование SubNibble

В преобразовании SubNibbles состояния обрабатываются как матрицы полубайтов 2 × 2. Преобразование осуществляется с одним полубайтом одновременно. Содержание каждого полубайта изменяется, но порядок полубайтов в матрице остается тем же самым. В процессе обработки каждый полубайт преобразуется независимо: есть четыре различных преобразования от полубайта к полубайту.

SubNibbles содержит четыре независимых преобразования от полубайта к полубайту.

Рисунок P.7 также показывает таблицы подстановки (S-блоки) для преобразования SubNibbles. Преобразование SubNibbles обеспечивает эффект перемешивания. Например, два полубайта, A_{16} и B_{16}, которые отличаются только одним битом (самый правый бит), преобразованы к 0_{16} и 3_{16}, которые отличаются двумя битами.

InvSubNibbles

InvSubNibbles – инверсия преобразования SubNibbles. На рис. P.7 также показано обратное преобразование. Мы можем легко проверить, что эти два преобразования инверсны друг другу.

Пример P.2

Рисунок P.8 показывает, как преобразовано состояние, применяющее преобразование SubNibbles. На рисунке также показано преобразование InvSubNibbles, которое создает первоначальное состояние. Обратите внимание, что если два полубайта имеют одинаковые значения, в результате их преобразования получается одинаковое значение — поскольку каждый полубайт использует одну и ту же таблицу.

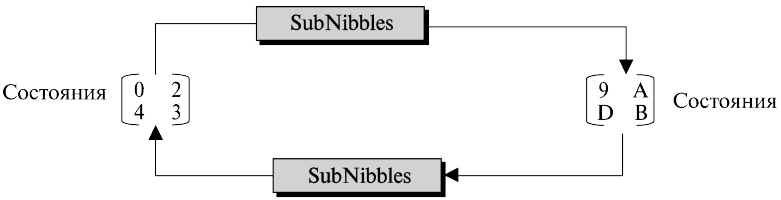


Рис. P.8 Преобразование SubNibbles

Перестановка

Другое преобразование, которое проводится в раунде, — сдвиг, переставляющий полубайты. Преобразование сдвига в S-AES делается на уровне полубайта; порядок следования битов в полубайте не изменяется.

ShiftRows

Преобразование при шифровании, называемое ShiftRows, сдвигает биты строки влево. Число сдвигов зависит от номера строки (0, 1) в матрице состояний. Это означает, что строка 0 не сдвигается вообще, а строка 1 сдвигается на 1 полубайт. Рисунок P.9 показывает преобразование сдвига. Обратите внимание, что ShiftRows-преобразование работает одновременно только с одной строкой.

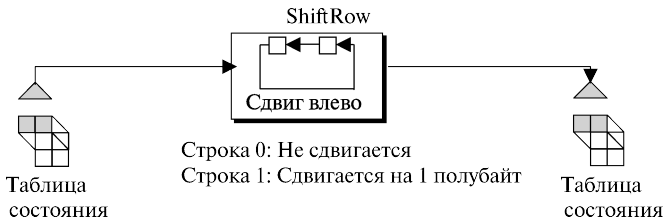


Рис. P.9. Преобразование ShiftRows

InvShiftRows

При дешифровании преобразование называется InvShiftRows и производит сдвиг вправо. Число сдвигов равно номеру строки в матрице состояний (0, 1).

Преобразования ShiftRows и InvShiftRows инверсны друг другу.

Пример P.3

Рисунок P.10 показывает, как преобразуется состояние с использованием ShiftRows. Рисунок также показывает как преобразование InvShiftRow создает первоначальное состояние.

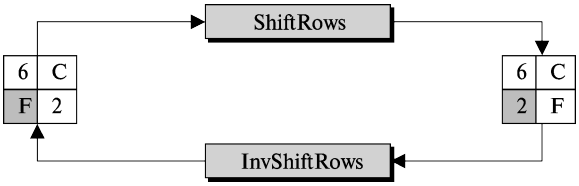


Рис. P.10. Преобразование ShiftRows в примере P.3

Смешивание

Подстановка обеспечивает преобразование SubNibbles и изменяет значение полубайта исходя только из первоначального значения полубайта и входа в таблицу; процесс не включает соседние полубайты. Мы можем сказать, что преобразование SubNibbles происходит *внутри полубайта*. Перестановка, обеспеченная ShiftRows-преобразованием, обменивает полубайты, не переставляя биты в байтах. Мы можем сказать, что ShiftRows — преобразование *обмена полубайтами*. Нам нужно также преобразование *между полубайтами*, которое изменяет биты в полубайте, учитывая биты в соседних полубайтах. Мы должны смешать полубайты, чтобы обеспечить рассеивание на побитовом уровне.

Смешивающееся преобразование изменяет содержание каждого полубайта, используя 2 полубайта одновременно и объединяя их, чтобы создать 2 новых полубайта. Чтобы гарантировать, что каждый новый полубайт отличается от других (даже если старые полубайты одинаковы), комбинация сначала умножает каждый полубайт на различную константу, а затем смешивает их. Смешивание может быть обеспечено матричным умножением. Как мы обсуждали в лекции 2, когда мы умножаем квадратную матрицу на матрицу-столбец, результат — новая матрица-столбец. Каждый элемент в новой матрице зависит от двух элементов старой матрицы после того, как они умножены на значение строки в матрице констант.

MixColumns

Преобразование MixColumns работает на уровне столбца; оно преобразовывает каждый столбец состояния в новый столбец. Преобразование — фактически матричное умножение столбца состояния на квадратную матрицу констант. Полубайты в столбце состояний и в матрице констант интерпретируются как слова по 4 бита (или полиномы) с коэффициентами в GF(2).

Умножение байтов делается в $GF(2^4)$ по модулю $(x^4 + x + 1)$ или (10011). Сложение — это операция ИСКЛЮЧАЮЩЕЕ ИЛИ на 4 бита. Рисунок P.11 показывает преобразование MixColumns.

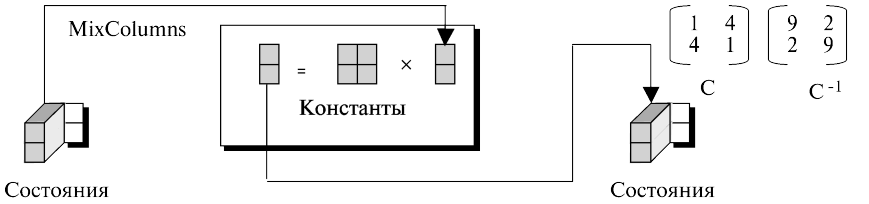


Рис. P.11 Преобразование MixColumn

InvMixColumns

InvMixColumns-преобразование — в основном такое же, как преобразование MixColumns. Если две матрицы констант инверсны друг другу, то несложно доказать, что эти два преобразования инверсны друг другу.

MixColumns- и InvMixColumns-преобразования инверсны друг другу.

Рисунок P.12 показывает, как преобразуется состояние, используя преобразование MixColumns. Мы также видим здесь, что преобразование InvMixColumns создает первоначальный текст.

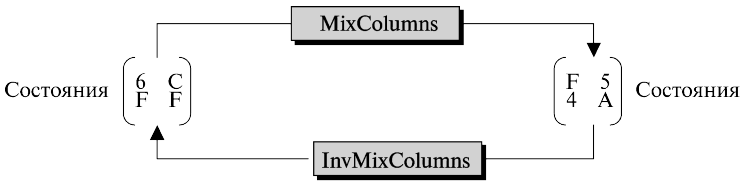


Рис. P.12. Преобразование Mix Column в примере 7.5.

Обратите внимание, что равные байты в старом состоянии неравны в новом состоянии. Например, два байта F во второй строке изменены на 4 и A.

Дополнение ключа

Вероятно, самое важное преобразование — это то, которое включает ключ шифра. Все предыдущие преобразования используют известные алгоритмы, которые являются обратимыми.

Если ключ шифра не добавляется к состоянию в каждом раунде, для противника очень просто найти исходный текст, учитывая зашифрованный текст. В этом случае ключ шифра — единственная гарантия безопасности между Алисой и Бобом.

S-AES использует процесс, называемый *расширением ключа* (он будет рассмотрен позже в этом приложении), который из ключа шифра создает три ключа раунда. Каждые ключ раунда имеет длину 16 битов – он применяется как два слова по 8 битов. При дополнении ключа и состояниякаждое слово рассматривают как матрицу-столбец.

AddRoundKey

AddRoimdKey обрабатывает один столбец одновременно. Это преобразование похоже на MixColumns. MixColumns умножает квадратную матрицу констант на каждый столбец состояний.

AddRoundKey складывает слово ключа раунда с каждой матрицей-столбцом состояний. Операции в MixColumns — матричное умножение; операции в AddRoundKey — матричное сложение. Сложение выполнено в поле GF(2^4). Поскольку сложение и вычитание в этом поле одинаковы, преобразование AddRoundKey самоинверсно. Рисунок Р.13 показывает преобразование AddRoundKey.

Преобразование AddRoundKey самоинверсно.

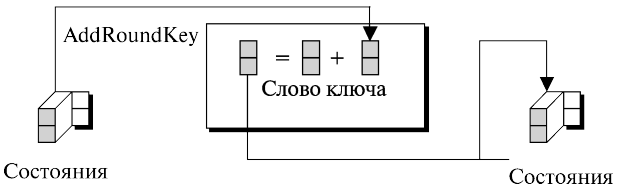


Рис. Р.13. Преобразование AddRoundKey

Р.3. Расширение ключа

Процедура *расширения ключа* создает три ключа раунда по 16 битов из единственного ключа шифра на 16 битов. Первые ключи раунда используются для предварительного преобразования (AddRoundKey); оставшиеся ключи применяются для последнего преобразования (AddRoundKey) в конце раунда 1 и раунда 2.

Процедура расширения ключей создает слово ключа раунда последовательно – слово за словом, где слово — массив 2 полубайта. Процедура создает 6 слов, которые обозначаются $\mathbf{w_0, w_1, w_2, \ldots\ldots, w_5}$.

Создание слов в S-AES

Рисунок Р.14 показывает, как создаются 6 слов из первоначального ключа.

Процесс создания слов в S-AES следующий:

1. Первые два слова ($\mathbf{w_0, w_1}$) создаются из ключа шифра. Ключи шифра представляют как массив из 4 полубайтов ($\mathbf{n_0 - n_3}$). Первые 2 полубайта ($\mathbf{n_0, n_1}$) образуют слово w_0; следующие 2 полубайта ($\mathbf{n_2, n_3}$) образуют слово w_1. Другими словами, конкатенация слов в этой группе копирует ключ шифра.
2. Остальная часть слов (w_i $i = 2 - 5$) создается следующим образом.
   a. Если *(i* mod 2) = 0, $w_i = t_i \oplus w_{i-2}$. Здесь t_i — временное слово, является результатом применения к w_i двух процедур: SubWord и RotWord операции

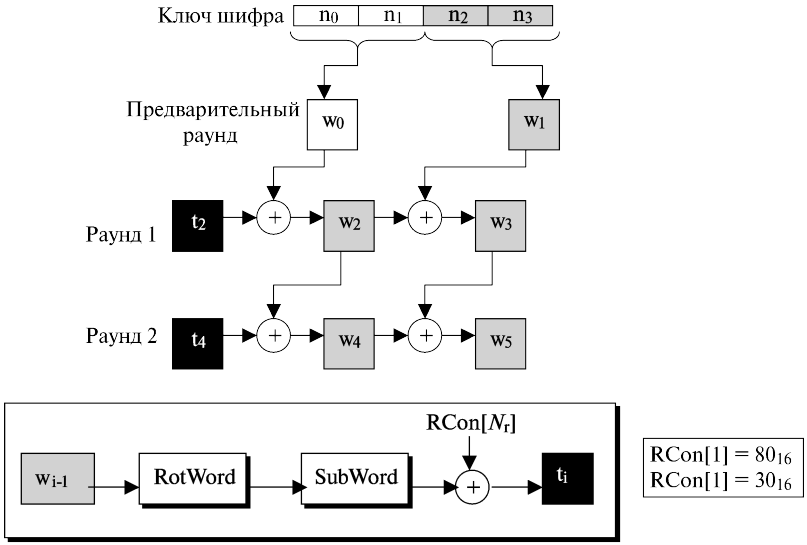


Рис. Р.14 Создание слов в S-AES

ИСКЛЮЧАЮЩЕЕ ИЛИ к результату с константой раунда Rcon[N_r], где N_r — номер раунда. Другими словами, мы имеем

$$\mathbf{t}_i = \text{SubWord}(\text{RotWord}(w_{i-1})) \oplus \mathbf{RCon}[N_r]$$

Слова w_2 и w_4 создаются, используя этот процесс.

б. Если *(i* mod 2) ≠ 0, $\mathbf{w_i = w_{i-1} \oplus w_{i-2}}$. На рисунке Р.14 это означает, что каждое слово создается от слова слева и слова сверху. Слова **w3** и **w5** создаются, используя этот процесс.

RotWord

RotWord (вращение слова) — процедура, подобная преобразованию ShiftRows, но она применяется только к одной строке. Процедура получает слово как массив 2 полубайтов и сдвигает каждый полубайт влево с полным вращением (полный оборот) слова. В S-AES это заполнение 2 полубайтов в слове.

SubWord

SubWord (слово-заменитель) – процедура является подобной преобразованию SubNibbels, но она применяется только к 2 полубайтам. Процедура берет каждый полубайт в слове и заменяет его другим полубайтом. Она применяет таблицы SubNibbels, показанные на рис. Р.7.

Константы раунда

Каждая константа раунда, RC, является значением, содержащим 2 полубайта, в котором самый правый полубайт всегда нуль. Рисунок P.14 также показывает значение RC

Пример P.4

Таблица P.1 показывает, как вычисляется ключ для каждого раунда. При этом предполагается, что Алиса и Боб согласовали ключ 2475_{16}.

Таблица P.1. Пример расширения ключа

Раунд	Значение t_i	Первое слово в раунде	Второе слово в раунде	Ключ раунда
0		$w_0 = 24$	$w_1 = 75$	$K_0 = 2475$
1	$t_2 = 95$	$w_2 = 95 \oplus 24 = B1$	$w_2 = B1 \oplus 75 = C4$	$K_0 = B1C4$
2	$t_4 = EC$	$w_4 = B1 \oplus EC = 5D$	$W_5 = 5D \oplus C4 = 99$	$K_2 = 5D99$

В каждом раунде второе слово вычисляется очень просто. Для вычисления первого слова мы должны сначала вычислить значение временного слова (t_i), как это показано ниже:

RotWord (75) = 57 → SubWord (57) = 15 → $t_2 = 15 \oplus$ **RC[1]** $= 15 \oplus 80 = 95$
RotWord (C4) = 4C → SubWord (4C) =DC → $t_4 = DC \oplus$ **RC [2]** $= DC \oplus 30 = EC$

P.4. Шифры

Теперь давайте посмотрим, как S-AES используют четыре типа преобразований для шифрования и дешифрования. Алгоритм шифрования будем называть *шифратор,* а алгоритм дешифрования — *обратный шифратор*.

S-AES — шифр не-Файстеля, и это означает, что каждое преобразование или группа преобразований должны быть обратимыми. Кроме того, шифратор и обратный шифратор должны применять эти операции таким способом, при котором они отменяют друг друга. Ключи раунда должны использоваться в обратном порядке. Чтобы выполнить эти требования, преобразования применяются в установленном порядке в шифраторе и обратном шифраторе, как это показано на рис. P.15.

Первое: порядок использования SubNibbles и ShiftRows изменяется в обратном шифраторе. Второе: порядок MixColumns и AddRoundKey в обратном шифраторе также изменяется. Это отличие в порядке необходимо, чтобы сделать каждое преобразование в шифре соответствующим по порядку его инверсии в обратном шифраторе. Следовательно, алгоритм дешифрования в целом — инверсия алгоритма шифрования. Обратите внимание, что ключи раунда используются в обратном порядке.

Пример P.5

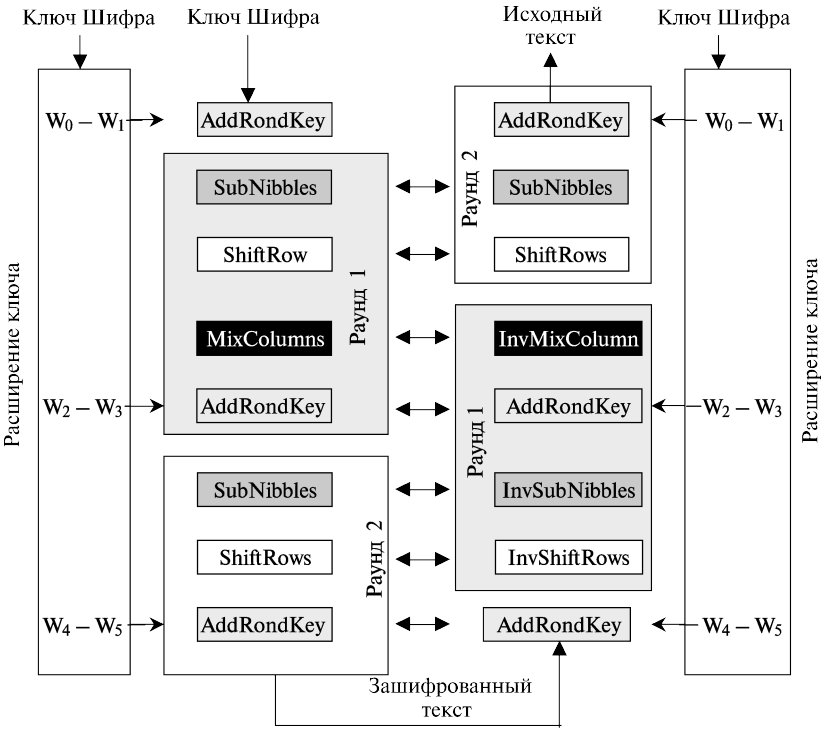
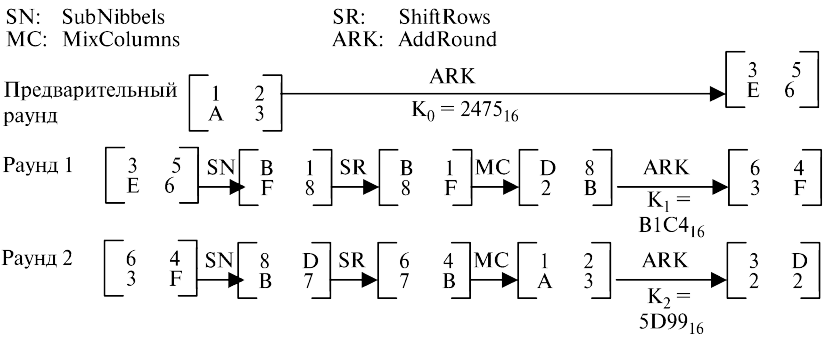


Рис. Р.15. Шифратор и обратный шифратор исходного проекта

Мы выбираем случайный блок исходного текста, ключ шифра, применяемого в примере Р.4, и определяем, какой получается блок зашифрованного текста:



Исходный текст: $1A23_{16}$ Ключ: 2475_{16} Зашифрованный текст: $3AD2_{16}$

Рисунок Р.16 показывает значение состояний в каждом раунде. Мы используем ключи раунда, сгенерированные в примере Р.4.

Приложение Q. Некоторые доказательства

Это приложение содержит некоторые доказательства для теорем, используемых в лекциях 2 и 9. Почти все они короткие и неформальные. Они рассчитаны на студентов, изучающих курс криптографии. Читатель, заинтересованный в более детальном изучении, может пополнить свои знания из книг, посвященных теории чисел.

Q.1. Лекция 2

Эта лекция содержит некоторые доказательства теорем по теории делимости, евклидовых алгоритмов и сравнений.

Теория делимости

Ниже — доказательства нескольких теорем по теории делимости.

Теорема Q.I: Уравнение деления (алгоритм)
Для целого числа a и b, где $b > 0$, существуют целые числа q и r, такие, что $a = q \times b + r$.

Доказательство:
Рассмотрим арифметическую прогрессию в форме

$$..., -3 \times b, -2 \times b, -1 \times b, 0 \times b, 1 \times b, 2 \times b, 3 \times b, ...$$

Очевидно, что целое число или является равным одному из членов этой прогрессии, или находится между двумя последовательными членами. Другими словами, $a = q \times b$ + r, где $q \times b$ – член в вышеупомянутой прогрессии и r — смещение от этого члена.

Теорема Q.2.
Если a | 1, тогда a = ±1.

Доказательство:
$a \mid 1 \rightarrow .1 = a \times x$, где x — целое число.
Это означает: ($x = 1$ и $a = 1$) или ($x = -1$ и $a = -1$).
Поэтому: $a = \pm 1$.

Теорема Q.3.
Если a | b и b | a, тогда a= ±b.

Доказательство:
$a \mid b \rightarrow b = x \times a$, где x — целое число.
$b \mid a \rightarrow = y \times b$, где y — целое число.

Мы имеем $a = y \times (x \times a) = (y \times x) \times a. \rightarrow y \times x = 1$.
Это означает: ($x = 1$ и $y = 1$) или ($x = -1$ и $y = -I$).
Поэтому: $a = y \times b \rightarrow \pm b$.

Теорема Q.4.
Если a | b и b | c, тогда a | c.

Доказательство:
$a \mid b \rightarrow b = x \times a$, где x — целое число.
$b \mid c \rightarrow c = y \times b$, где — y целое число.
Мы имеем $c = y \times (x \times a) = (y \times x) \times a$.
Поэтому $a \mid c$.

Теорема Q.5.
Если $a \mid b$ и $a \mid c$, тогда $a \mid (b + c)$.

Доказательство:
$a \mid b \rightarrow b = x \times a$, где x — целое число.
$a \mid c \rightarrow c = y \times a$, где y — целое число.
Мы имеем $b + c = (x + y) \times a$.
Поэтому $a \mid (b + c)$.

Теорема Q.6.
Если $a \mid b$ и $a \mid c$, тогда $a \mid (m \times b + n \times c)$, где m и n — произвольные целые числа.

Доказательство:
$a \mid b \rightarrow b = x \times a$, где x — целое число.
$a \mid c \rightarrow c = y \times a$, где y — целое число.
Мы имеем $(m \times b + n \times c) = m \times (x \times a) + n \times x\ (y \times a) = (m \times x + n \times y) \times a$.
Поэтому $a \mid (m \times b + n \times c)$.

Евклидовы алгоритмы

Мы использовали евклидов и расширенный евклидов алгоритмы в лекции 2. Ниже даны доказательства двух теорем, связанных с этими алгоритмами.

Теорема Q.7.
Если a = b × q + r (r — остаток от деления a на b), то НОД (a, b) = НОД (b, r).

Доказательство

Предположим, что E — набор всех общих делителей a и b. Каждый элемент E делит a и b, поэтому он делит $r = a - b \times q$. Это означает, что E — набор всех общих делителей a, b, и r.

Предположим, что F — набор всех общих делителей b и r. Каждый элемент F делит b и r; поэтому делит $a = b \times q + r$. Это означает, что F — набор всех общих делителей a, b и r.

Это означает, что $E = F \rightarrow a, b$ и r имеют одинаковый набор общих делителей. Поэтому НОД (a, b) = НОД ($b,$ r).

Как мы видели в лекции 2, эта теорема — основание евклидового алгоритма для нахождения наибольшего общего делителя двух целых чисел.

Теорема Q.8.

Если a и b — целые числа и оба не равны нулю, то существуют целые числа x и y, такие, что НОД (a, b) = $x \times a + y \times b$.

Доказательство:

Предположим, что D — набор всех значений ($x \times a + y \times b$), а d есть наименьшее значение, отличное от нуля.

Мы можем записать $a = q \times d + r \rightarrow r = a - q \times d = (1 - q \times x)a + (-q \times y)\ b,$ где $0 \leqslant r < d$.

Это подразумевает, что r входит в D. Но поскольку $r < d,$ то или r = 0, или $d \mid a$.

Подобным путем мы можем показать, что $d \mid b$.

Поэтому d — общий делитель a и b.

Любой другой делитель a и b делит $d = x \times a + y \times b$. Поэтому d должен быть НОД (a,b).

Как мы видели в лекции 2, эта теорема — основание расширенного евклидового алгоритма.

Сравнение

Ниже даются доказательства некоторых теорем о сравнении, используемые в лекции 2.

Теорема Q.9.

Если $a,\ b$ и n — целые числа и $n > 0$, то $a \equiv b \pmod{n}$, тогда и только тогда, когда существует целое число q, такое, что $a = q \times n + b$.

Доказательство:

Если $a \equiv b \pmod{n}$, то $n \mid (a - b),$ это означает, что есть целое число q, такое, что $a - b = q \times n$.

Поэтому мы имеем $a = q \times n + b$.

Если есть целое число q , такое, что $a = q \times n + b$, тогда $a - b = q \times n,$ что означает $n \mid (a - b)$.

Поэтому мы имеем $a \equiv b \pmod{n}$.

Теорема Q.10.

Если a, b, c и n — целые числа при $n > 0$, такие, что $a \equiv b \pmod{n}$, то

a. $a + c \equiv b + c \pmod{n}$.
b. $a - c \equiv b - c \pmod{n}$.
c. $a \times c \equiv b \times c \pmod{n}$.

Доказательство: Обратите внимание, что $a \equiv (\text{mod } n) \rightarrow n \mid (a - b)$.
а. $(a+c) - (b + c) = a - b$. Поскольку $n \mid (a - b)$, $n \mid (a + c) - (b + \text{c})$.
Поэтому $a + c \equiv b + c$ (mod n).
б. $(a - c) - (b - \text{c}) = a - b$. Поскольку $n \mid (a - b)$, n $\mid (a - c) - (b - \text{c})$
Поэтому $a - c \equiv b - c$ (mod n).
в. $(a \times \text{c}) - (b \times c) = (a - b) \times \text{c}$. Поскольку $n \mid (a - b)$, $n \mid (a - b) \times c$.
Поэтому $a \times c \equiv b \times c$ (mod n).

Теорема Q.11.
Если a, b, c, d и n — целые числа при n> 0, такие, что a × b (mod n) и c ≡ d (mod n), то
a. $a + c \equiv b + d$ (mod n).
b. $a - c \equiv b - d$ (mod n)
c. $a \times c \equiv b \times d$ (mod n).

Доказательство:
Обратите внимание, что
a × b mod n → $(a - b)$ → (a-b) $= k \times n$; $c \equiv d$ (mod n) → $(c - d) = l \times n$
а. $(a + c) - (b + d) = (a - b) + (\text{c} - d) = \text{k} \times n + l \times n = (k + l) \times n$.
Поэтому $(a + c) \equiv b + d$ (mod n).
б. $(a - \text{c}) - (b - d) = (a - b) - (c - d) = k \times n - l\, x\, n = (k - l\,) \times n$.
Поэтому $a - c \equiv b - d$ (mod n).
в. $a \times c - (b - d) = c \times (a - b) + b \times (c - d) = (\text{c} \times k + b \times l) \times \text{n}$.
Поэтому $a \times c \equiv b \times d$ (mod n)

Q.2. Лекция 9

В этом разделе приводятся некоторые доказательства теорем, используемых в лекции 9. Мы не будем приводить длинных доказательств, таких как доказательство китайской теоремы об остатках, — интересующимся студентам рекомендуем посмотреть книги по теории чисел.

Простые числа

Мы докажем только одну теорему о простых числах.

Теорема Q.I2.
Если n — составной объект, то есть простой делитель p, такой, что $p \leqslant \sqrt{n}$.

Доказательство:
Поскольку n — составной объект, $n = a \times b$.
Если p — наименьший простой делитель n, тогда p ⩽ a и p ⩽ b.
Поэтому $p^2 \leqslant a \times b$ или $p^2 < n \rightarrow p \leqslant \sqrt{n}$

Эта теорема используется в решете Эратосфена, чтобы найти все простые сомножители n.

Phi-функция Эйлера

Ниже приводятся три доказательства, связанные с phi-функцией Эйлера.

Теорема Q.I3.
Если p — простое число, тогда $\phi(p) = p - 1$.

Доказательство:
Поскольку p — простое число, все целые числа, меньшие, чем p, взаимно простые по отношению к p.
Поэтому $\phi(p) = p - 1$.

Эта теорема — часть phi-функции Эйлера.

Теорема Q.14.
Если p — простое число и e — положительное целое число, тогда
$\phi(p) = p^e - p^{e-1}$

Доказательство:
Целые числа, которые не являются взаимно простыми с p^e – $(1 \times p)$, $(2 \times p)$ .., $(p^{e-1} \times p)$. Все они целые числа и имеют общий делитель p с p^e. Общее количество этих целых чисел — p^{e-1}. Остальная часть целых чисел является взаимно простой с p^e.
Поэтому $\phi(p) = p^e - p^{e-1}$

Теорема Q.15.
Если n — составной объект с разложением на простые множители $\prod p_i{}^{e_i}$, то $\phi(n) = \prod (p_i{}^{e_i} - p_i{}^{e_i-1})$

Доказательство:
Доказательство базируется на факте, что $\phi(n)$ — *мультипликативная функция,* в которой m и n являются взаимно простыми. Поскольку элементы в разложении n на простые множители *взаимно простые*, $\phi(\prod p_i{}^{e_i}) = \prod \phi(p_i{}^{e_i})$.
Поэтому $\phi(n) = \prod (p_i{}^{e_i} - p_i{}^{e_i-1})$

Эта теорема — обобщение phi-функции Эйлера.

Малая теорема Ферма

Ниже приводятся две теоремы, которые относятся к малой теореме Ферма.

Теорема Q.I6.
Если p — простое число и a — положительное целое число, взаимно простое с p, то $a^{p-1} \equiv 1 \pmod{p}$.

Доказательство:

Может быть доказано, что вычеты элементов a, $2a$, .., $(p-1)\,a$ по модулю p равны 1, 2 ..., $(p-1)$, но не обязательно в том же самом порядке.

В результате $a \times 2a \times ... \times (p-1)$ равно $[(p-1)]!\ a^{p-1}$

В результате $1 \times 2 \times ... \times (p-1)$ равно $[(p-1)]!$

Это означает $[(p\text{-}1)]!\ a^{p-1} \equiv [(p-1)]!\ (\text{mod } p)$,

Сокращая обе стороны тождества на $(p-1)!$, мы получаем $a^{p-1} \equiv 1(\text{mod } p)$.

Эта теорема – первая версия Малой теоремы Ферма.

Теорема Q.I7.

Если p — простое число и a — положительное целое число, то $a^p \equiv a\ (\text{mod } p)$.

Доказательство:

Если a и p взаимно-простые, используя результат предыдущей теоремы, мы умножаем обе стороны сравнения, чтобы получить $a^p \equiv a\ (\text{mod p})$.

Если $p \mid a$, то $a^p \equiv a \equiv 0\ (\text{mod } p)$.

Эта теорема – вторая версия теоремы Ферма.

Теорема Эйлера

Ниже приводится доказательство одной теоремы, связанной с первой версией теоремы Эйлера. Вторую версию мы доказали в лекции 9.

Теорема Q.18.

Если n и a являются взаимно-простыми, то $a^{\phi(n)} \equiv 1\ (\text{mod } n)$.

Доказательство:

Предположим, что элементы в $Z\,n^* - r_1, r_2, .., r_{\phi(n)}$.

Мы создаем другой набор $ar_1, ar_2, ..., ar, ar_{\phi(n)}$ умножая каждый элемент в $Z\,n^*$ на a. Может быть доказано, что каждый элемент в этом новом наборе является конгруэнтным элементу в Zn^* (не обязательно в том же самом порядке).

Таким образом, $ar_1 \times ar_2 \times, ar, a \times r_1, r_2, .., r_{\phi(n)}\ (\text{mod } n)$

Мы имеем $a^{\phi(n)}\ [r_1 \times r_2 \times .. \times r_{\phi(n)}] \equiv r_1 \times r_2 \times ... \times r_{\phi(n)}\ (\text{mod } n)$

Откуда $a^{\phi(n)} \equiv 1\ (\text{mod } n)$.

Основная теорема арифметики

Ниже приводится частичное доказательство основной теоремы арифметики.

Теорема Q.19

Любое положительное целое число n больше чем 1 может быть представлено, как произведение простых чисел.

Доказательство

Мы используем индукцию. Первое утверждение (база индукции) $n = 2$, является простым числом. Предположим, что все положительные целые числа меньше, чем n может быть представлены как произведение простых чисел. Мы докажем, что n может также быть представлено как произведение простых чисел.

Может иметь два случая: n – простое число, или n – составной объект.

1. Если n является простым, оно может быть представлено как произведение из одного этого простого числа.
2. Если n – составной объект, то мы можем написать $n = a \times b$. Поскольку a и b – оба меньше чем n, каждое из них может быть представлено как произведение простых чисел согласно нашему предположению. Поэтому, n может быть представлено как произведение простых чисел.

Эта теорема — частичное доказательство основной теоремы арифметики. Чтобы полностью доказать эту теорему, мы должны показать, что это произведение уникально. Но мы рекомендуем посмотреть полное доказательство в книгах по теории чисел.

Глоссарий

A5/1. Член семейства A5 шифров потока, используется в Глобальной Системе Подвижной Связи (GSM).

AddRoundKey. В AES — операция, которая складывает слово ключ раунда с каждой матрицей-столбцом состояния.

CBC-MAC. См. MAC.

CMAC (Chipper-based Message Authentication Code). Стандартный контроль подлинности сообщения (MAC — Message Authentication Code), который определил NIST (FIPS 113) как алгоритм аутентификации данных. Метод подобен режиму сцепления блоков шифрованного текста.

HAVAL — алгоритм хэширования переменной длины с дайджестом сообщения размера 128, 160, 192, 224 и 256. Размер блока — 1024 бита.

InvMixColumns. В AES — инверсия операции MixColumns, используется в обратном шифре.

InvSubBytes. В AES — инверсия операции SubBytes, используется при дешифрации. **InvShiftRows.** В AES — инверсия операции ShiftRows, используется при дешифрации.

MixColumns. В AES — операция, которая преобразовывает каждый столбец состояния к новому столбцу.

MixRows. В Whirlpool — операция, подобная MixColumns в AES, за исключением того, что смешиваются строки вместо столбцов.

nonce — случайное число.

Oakley. Протокол смены ключей, разработанный Хилари Ормано ; это – улучшенный метод Диффи-Хеллмана.

P-блок (P-box). Компонент в современном блочном шифре, который перемещает биты.

P-блок сжатия. P-блок с n входами и m выходами, где $n > m$.

RC4. Ориентированный на байт шифр потока, спроектированный Рональдом Ривестом.

Rijndael. Современный блочный шифр, разработанный бельгийскими исследователями Джоном Даеменом и Винсентом Риджментом и выбранный NIST как Усовершенствованный стандарт шифрования (AES).

RotWord. В AES — операция, подобная операции ShiftRows, работающая только с одной строкой слова в процессе расширения ключа.

S-блок (S-box). Компонент в блочном шифре, который заменяет биты на входе новыми битами в выходе.

SHA-1. SHA с блоком 512 битов и дайджестом 160 битов.

SHA-224. SHA с блоком 512 битов и дайджестом 224 бита.

SHA-256. SHA с блоком 512 битов и дайджестом 256 битов.

SHA-384. SHA с блоком 1024 битов и дайджестом 384 бита.

SHA-512. SHA с блоком 1024 битов и дайджестом 512 битов.

ShiftColumns. В Whirlpool — операция, подобная ShiftRows-преобразованию в AES, за исключением того, что вместо строк сдвигаются столбцы.

ShiftRows. В AES — преобразование, которое сдвигает байты.

SubBytes. В AES — преобразование, которое использует таблицу, чтобы провести операцию подстановки байтов.

SubWord. В AES — процедура, подобная преобразованию SubBytes, но используемая только с одной строкой.

X.509. Рекомендация, разработанная ITU и принятая Интернетом, которая определяет структуру сертификации.

Whirlpool. Криптографическая система, основанная на измененном AES.

A

Абелева группа (abelian group). Коммутативная группа.

Аварийный протокол (Alert protocol). В SSL и TLS — протокол, извещающий об ошибках и отклонениях от правильной работы.

Агент доступа к сообщению (MAA — Message Access Agent). Программа клиента, которая накапливает и сохраняет сообщения от сервера.

Агент пользователя (UA — User Agent). Компонент в почтовой системе, который готовит сообщение и электронный конверт.

Агентство Национальной безопасности (NSA — National Security Agency) — американское агентство, собирающее сведения безопасности.

Аддитивная инверсия (additive inverse). В модульной арифметике a и b аддитивно инверсны друг другу, если $(b + a) \bmod n = 0$.

Аддитивный шифр (additive cipher). Самый простой моноалфавитный шифр, в котором каждый символ зашифровывается путем сложения его значения с ключом.

Активная атака (active attack). Атака, которая может изменить данные или повредить систему.

Алгоритм верификации (verifying algorithm). Алгоритм, который проверяет законность цифровой подписи на стороне приемника.

Алгоритм декодирования (decryption algorithm) — алгоритм, используемый для декодирования.

Анонимный протокол Диффи-Хеллмана (anonymous Diffie-Hellman). Первоначальный протокол Диффи-Хеллмана в SSL и TLS.

Ассоциативность (associativity). В алгебраической структуре, если a, b и c — элементы основного набора и $\bullet$ обозначает одну из операций, свойство ассоциативности гарантирует, что $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.

Алгебраическая структура (algebraic structure). Структура, состоящая из заранее определенного набора операций и элементов. Примеры алгебраических структур — группы, кольца и поля.

Алгоритм декодирования (decryption algorithm) — алгоритм, используемый для декодирования.

Алгоритм безопасного хэширования (SHA — Secure Hash Algorithm) — ряд стандартов хэш-функции, разработанных NIST. Был издан как FIPS 180. Он в основном базируется на MD5.

Алгоритм возведения в квадрат и умножения (square-and-multiply algorithm). Быстрый метод возведения в степень, в котором используются две операции — возведение в квадрат и умножение — вместо одной операции умножения.

Анализ трафика (traffic analysis). Тип атаки конфиденциальности, в которой нападающий получает некоторую информацию, контролируя сетевой трафик.

Атака анализом времени (timing attack). Атака RSA, основанная на быстром показательном алгоритме. Атака использует факт, что время, которое требуется на каждую итерацию, больше, если бит равен 1.

Атака анализом мощности (power attack). В RSA — атака, подобная атаке анализом времени, при которой используется измерение мощности в течение процесса дешифрования.

Атака воспроизведения (replay attack). См. *переигрывание*.

Атака грубой силы (brute-force attack). Тип атаки, прикоторой нападающий пробует использовать все возможные ключи, чтобы найти ключ шифра.

Атака знания исходного текста (known-plaintext attack). Атака, при которой нападающий использует набор известных исходных текстов и их соответствующих зашифрованных текстов, чтобы найти ключ шифра.

Атака короткого блокнота (short-pad attack). Атака RSA, при которой злоумышленник может найти исходный текст, если он имеет два экземпляра соответствующих зашифрованных текстов, каждый из которых создан с различным коротким заполнением.

Атака короткого сообщения (short-message attack). Атака RSA, при которой нападавший знает набор возможных исходных текстов и зашифровывает их так, чтобы найти эквивалент перехваченного зашифрованного текста.

Атака малого значения ключа дешифрации (low-private-exponent attack). В RSA — атака, которая может быть начата, если конкретный образец ключа имеет малое значение.

Атака на нескрытое сообщение (unconcealed message attack). Атака RSA, основанная на отношениях перестановки между исходным текстом и зашифрованным текстом; нескрытое сообщение — это сообщение, которое при шифровании воспроизводит само себя.

Атака на секретную информацию (security attacks). Атаки, угрожающие целям безопасности.

Атака передачи по цепочке. См. *атака «посредника»*.

Атака по теореме Куперсмита (Coppersmith's theorem attack). Тип атаки RSA, которая может быть предпринята, если значение показателя степени ключа шифрования — маленькое.

Атака сведения к середине (meet-in-the-middle attack). В двойной шифровке — атака, когда пытаются найти исходный текст и зашифрованный текст; при этом считают, что шифрование первого и дешифрование второго содержит аналогичные операции.

Атака с выборкой зашифрованного текста (chosen-ciphertext attack). Тип атаки, при которой противник перехватывает набор зашифрованных текстов и находит соответствующие исходные тексты. Затем он анализирует пары зашифрованного текста / исходных текстов, чтобы найти ключ шифра.

Атака по выбранному сообщению (chosen-message attack). Атака, при которой нападавший каким-то способом заставил Алису подписать одно или более сообщений. Нападавший позже создает другое сообщение, с содержанием, которое он хочет передать, и подставляет на нем подпись Алисы.

Атака по образцу (pattern attack). Атака шифра на транспозицию, при которой используются повторные образцы, созданные в зашифрованном тексте.

Атака при известном сообщении (known-message attack). Атака цифровой подписи, при которой нападавший имеет доступ к одной или более парам подписей сообщения.

Атака с выборкой исходного текста (chosen-plaintext attack). Тип атаки, при которой противник перехватывает набор исходных текстов и находит соответствующие зашифрованные тексты. Затем он анализирует пары исходного текста / зашифрованного текста, чтобы найти ключ шифра.

Атака связанного сообщения (related message attack). Атака RSA, при которой два связанных зашифрованных текста используются, чтобы найти два связанных исходных текста, когда открытый ключ имеет малую длину. Обнаружена Франклином Рейтером.

Атака словаря (dictionary attack). Атака, при которой злоумышленник интересуется обнаружением только пароля независимо от пользовательского ID.

Атака отказа в обслуживании (denial of service). Единственная цель атаки — готовность системы. Атака может замедлить или прервать работу системы.

Атака «посредника» (man-in-the-middle attack). Атака протокола Диффи-Хеллмана, в который нападающий вводит в заблуждение две стороны, вовлеченные в обмен по протоколу, создавая два ключа сеанса: один между первой стороной и нападающим, а другой — между нападающим и второй стороной.

Атака только зашифрованного текста (ciphertext-only attack). Тип атаки, в которой злоумышленник имеет для анализа только перехваченный зашифрованный текст.

Атака только на ключ (key-only attack) — атака цифровой подписи, в которой нападающий имеет доступ только к открытому ключу.

Атака широковещательной рассылки (broadcast attack). Тип атаки на RSA, которая может быть начата, если один объект передает одно и то же маленькое сообщение группе получателей с одним и тем же самым низким уровнем шифрования.

Атака циклического повторения (cycling attack). Тип атаки на RSA, которая использует тот факт, что зашифрованный текст — перестановка исходного текста, и непрерывное шифрование зашифрованного текста в конечном счете заканчивается исходным текстом.

Аутентификация (authentication). Служба безопасности, которая проверяет на другом конце линии идентификационный код участника обмена.

Аутентификация на основе пароля (password-based authentication) — самый простой и самый старый метод аутентификации объекта, в котором пароль используется, чтобы идентифицировать претендента.

Аутентификация сообщения (message authentication). Доказательство подлинности передатчика без установления соединения.

Аутентификационный обмен (authentication exchange). Механизм безопасности, в котором два объекта обмениваются набором сообщений, чтобы доказать друг другу подлинность своей идентификации.

Аутентификация объекта (entity authentication). Методика, предназначенная обеспечивать одной стороне доказательство подлинности идентификатора дру-

гой стороны. Объект — подлинность идентификатора, которая должна быть доказана, — называется претендентом; сторона, которая пытается проверить идентификатор претендента, называется верификатором.

Аутентификация с помощью вызова-ответа (challenge-response authentication). Метод аутентификации, в котором претендент доказывает, что он *знает* секрет, не посылая его.

Аффинный шифр (affine cipher) — шифр, который объединяет аддитивные и мультипликативные шифры.

Б

Байт (byte). Группа из восьми битов, она же октет.

База данных услуг обеспечения безопасности (SAD — Security Association Database). Двухмерная таблица, где каждая строка определяет одиночные услуги обеспечения безопасности (SA).

Безопасность транспортного уровня (TLS — Transport Layer Security). Версия IETF протокола SSL.

Безопасный механизм смены ключей (SKEME — Secure Key Exchange Mechanism). Протокол, разработанный для смены ключей, который использует открытый ключ для шифрования объектов аутентификации.

Безопасное/многоцелевое расширение почты (S/MIME — Secure/Multipurpose Internet Mail Extension). Расширение к MIME, создавалось для обеспечения безопасности электронной почты.

Бесконечная группа (infinite group). Группа с бесконечным числом элементов.

Бесспорные подписи (undeniable signatures). Схема подписи с тремя компонентами: алгоритм подписания, протокол проверки и протокол отрицания.

Билет (ticket). Зашифрованное сообщение, предназначенное для объекта B, но посланное объекту A для доставки.

Бинарная операция (binary operation). Операция, которая имеет два сигнала на входе и создает один сигнал на выходе.

Биометрия (biometrics). Измерение физиологических или поведенческих особенностей, которые идентифицируют человека.

Бит (bit) — двоичная цифра со значением 0 или 1.

Блок (block). Группа битов, обрабатываемая как один модуль.

Блокнот вывода (opad — output pad). Второе заполнение, используемое в алгоритме HMAC.

Блочный шифр (block cipher) — тип шифра, в котором блоки исходного текста зашифрованы по одному с использованием одного и того же ключа шифра.

В

Взаимно простые (relatively prime). Два целых числа называются взаимно простыми, если их наибольший общий делитель — 1.

Вложенное сообщение с кодом аутентификации (nested MAC). MAC с двумя шагами.

Вмешательство (snooping). Неправомочный доступ к конфиденциальной информации. Атака цели конфиденциальности в информационной безопасности.

Внешняя обратная связь режим (OFB — Output FeedBack) — режим работы, подобный CFB (Cipher FeedBack), но здесь регистр сдвига модифицируется предыдущим r-битовым ключом.

Возможно слабые ключи (possible weak keys). Набор 48 ключей в DES, где каждый ключ создает только четыре различных ключа раунда.

Входной блокнот (ipad-input pad). Первое заполнение, используемое в алгоритме HMAC.

Выборочная подделка (selective forgery) — тип подделки, в которой подделывающий способен подделать подпись передатчика на сообщении с содержанием, выборочно вставленным самим подделывающим.

Вычет (residue). Остаток.

Г

Генератор ключей (key generator). Алгоритм, который создает ключи раунда из ключа шифра.

Генерирование ключей раунда (round-keys generation). В современном блочном шифре — процесс, который создает ключи раунда из ключа шифра.

Главный секретный код (master secret). В SSL — 48-байтовый секретный код, созданный из *предварительного главного секретного кода*.

Готовность (availability). Компонент информационной безопасности, который требует, чтобы информация, созданная и сохраненная организацией, была доступна разрешенным объектам.

Группа (group). Алгебраическая структура с только одной бинарной операцией, которая удовлетворяет четырем свойствам: замкнутости, ассоциативности, существованию тождественности и существованию инверсии.

Группа инженерной поддержки сети Интернет (IETF — Internet Engineering Task Force). Группа специалистов, работающая над разработкой и развитием набора протокола TCP/IP и Internet.

Группа перестановки (permutation group). Группа, в которой набор элементов является всеми перестановками элементов.

Д

Дайджест сообщения (message digest). Строка фиксированной длины, созданная при применении хэш-функции к сообщению.

Дайджест сообщения (MD — Message Digest). Набор нескольких алгоритмов хэширования, разработанных Роном Ривестом и обозначаемых как MD2, MD4 и MD5.

Дайджест сообщения оценки стойкости примитивов целостности (RACE RIPMED — RACE Integrity Primitives Evaluation Message Digest). Алгоритм хэширования с несколькими версиями, разработанный организацией RACE.

Двукратный DES (2DES — double DES). Алгоритм шифрования, который использует два экземпляра шифров для шифрования и два экземпляра обратных шифров для дешифрования.

Делимость (divisibility). Если a и b — целые числа и $a \neq 0$, мы говорим, что a делит b, если есть целое число k, такое, что $b = k \times a$.

Декодирование (decoding). Этот термин имеет много определений. В этой книге подразумевается одно из значений: преобразование n-битового целого числа в 2^n-разрядную строку с единственной единицей. Положение этой единственной единицы соответствует значению целого числа.

Детерминант (determinant). Скалярное значение, определенное для квадратной матрицы. Матрица обратима, если ее детерминант является отличным от нуля.

Дешифрование (decryption). Преобразование зашифрованного текста, чтобы создать первоначальный исходный текст.

Дискретный логарифм (discrete loatithm). Целое число *d* называется дискретным логарифмом *a* по основанию *r* , если $r^d \equiv a \pmod{n}$, где *r* — первообразный корень *n* и *a* и *n* — взаимно простые.

Дистрибутивность (distributivity). В алгебраической структуре с двумя операциями □ и • дистрибутивность □ • означает, что для всех *a*, *b* и *c* — элементов основного набора, — мы имеем $a \square (b \bullet c) = (a \square b) \bullet (a \square c)$ и $(a \bullet b) \square c = (a \square c) \bullet (b \square c)$.

Дифференциальный криптоанализ (differential cryptanalysis). Тип *атаки с выборкой исходного текста*, введенный Бихамом (Biham) и Шамиром (Shamir), использующий дифференциальный профайл S-блоков, чтобы напасть на составной шифр.

Домен дайджеста сообщения (message digest domain). Набор возможных результатов функции криптографического хэширования.

Доверенность (notarization). Механизм безопасности, при котором одна часть сообщества выбирает третью доверенную сторону для управления связью между двумя объектами.

Дополнение ключа (key complement). Строка, полученная инверсией каждого бита в ключе.

Е

Евклидов алгоритм (Euclidean's algorithm) — алгоритм нахождения наибольшего общего делителя двух положительных целых чисел.

З

Заголовок аутентификации (AH — Authentication Header). Протокол в IPSec, который обеспечивает целостность сообщения и аутентификацию.

Замкнутость (closure). В алгебраической структуре, если *a* и *b* — элементы основного набора и • обозначает одну из операций, свойство замкнутости гарантирует, что $c = a \bullet b$ — также член этого набора.

Заполнение трафика (traffic padding). Механизм безопасности, в котором в трафик вставлены некоторые фиктивные данные, чтобы сорвать атаку анализа трафика.

Засоряющая атака (clogging attack). Тип атаки в методе Диффи-Хеллмана, в которой злоумышленник может передать много полуключей одной из сторон, симулируя, что они — от различных источников. Атака может в конечном счете кончиться отказом в обслуживании.

Зашифрованный текст (ciphertext). Сообщение после того, как оно было зашифровано.

И

Идеальная прямая безопасность (PFS — Perfect Forward Security). Свойство криптографической системы, при которой раскрытие какого-то аспекта секретности не ставит под угрозу безопасность будущей связи.

Инфраструктура открытого ключа (PKI — Public-Key Infrastructure). Модель для создания и распределения сертификатов, основанная на X.509.

Имитация источника (masquerading). Тип атаки на целостность информации, в которой нападающий выдает себя за кого-то другого.

Интернет-протокол шифрования и идентификации (IKE — Internet Key Exchange). Протокол, предназначенный для создания услуг обеспечения безопасности в IPSec.

Исключение отказа от сообщения (nonrepudiation). Услуга безопасности, которая защищает против атаки отказа от сообщения либо передатчика, либо приемника данных.

Испытание на делимость (divisibility test). Самый элементарный детерминированный метод для проверки простоты чисел, в котором число объявляется простым, если оно не делится на все числа, меньшие, чем $\sqrt{n}$.

Испытание простоты чисел (primality test). Детерминированный или вероятностный алгоритм, который определяет, является ли положительное целое число простым числом.

Исходный текст (plaintext) — сообщение перед шифрованием или после хешифрования.

Итеративная криптографическая хэш-функция (iterated cryptographic hash function). Хэш-функция, в которой функция фиксированного размера создается и используется необходимое количество раз.

К

Квадратичный вычет (QR — Quadratic Residue). Коэффициент в уравнении $x^2 = a \pmod p$, где уравнение имеет два решения.

Квадратичный невычет (QNR — Quadratic NonResidue). Коэффициент в уравнении $x^2 = a \pmod p$, где уравнение не имеет решения.

Квадратичное сравнение (quadratic congruence). Конгруэнтное уравнение формы $ax^2 + bx + c = 0 \pmod n$.

Квадратная матрица (square matrix). Матрица, у которой число строк равно числу столбцов.

Китайская теорема об остатках (CRT — Chinese Remainder Theorem). Теорема, в которой доказывается, что существует уникальное решение к набору конгруэнтных уравнений с одной переменной, если модули взаимно простые.

Ключ (key). Набор значений, которые применяются в алгоритме для шифрования/дешифрования.

Ключ сеанса (session key) — секретный одноразовый ключ между двумя сторонами.

Ключ сдвига (shift cipher). Тип аддитивного шифра, в котором ключ определяет смещение символов к концу алфавита.

Код аутентификации сообщения (MAC — Message Authentication Code). Код, который включает информацию для обеспечения безопасности между двумя сторонами.

Код аутентификации сообщения, основанный на хешировании (HMAC — Hash-Based Message Authentication Code). Стандарт, выработанный NIST (FIPS198) для вложенного MAC.

Кодирование (encoding). Этот термин имеет много определений. В этой книге принято одно из значений: преобразование 2^n-разрядной строки с единствен-

ной единицей (1) к целому n-битовому числу. Позиция единственной единицы (1) определяет значение целого числа.

Кодирование Radix64 (Radix 64 encoding). Система кодирования, в которой двоичные данные разделены на блоки по 24 бита. Каждый блок разделен затем на четыре секции на 6 битов. Каждая секция из 6 битов интерпретируется как один символ в системе — ограниченная печатная строка.

Кое-что известное (something known). Секретная информация, известная только претенденту, которая может быть проверена верификатором для аутентификации объекта.

Кое-что свойственное (something inherent). Характеристики, используемые для аутентификации претендента, такие как обычные подписи, отпечатки пальцев, голос, лицевые характеристики, образец сетчатки глаза и почерк.

Кое-что, чем обладаю (something possessed). Нечто, принадлежащее претенденту, что может его идентифицировать, например паспорт, водительские права, удостоверение личности, кредитная карточка или карточка с интегральной схемой, включающей микропроцессор.

Кольцо (ring). Алгебраическая структура с двумя операциями. Первая операция должна удовлетворить все пять свойств, требуемых для абелевой группы. Вторая операция должна удовлетворить только первые два. Кроме того, вторая операция должна быть дистрибутивна (распределенная) по отношению к первой.

Кольцо ключей (key ring). Набор общедоступных или секретных ключей, используемых в PGP.

Комбинированная операция (combine operation). Операция в некоторых блочных шифрах, которая связывает два блока равной длины, чтобы создать новый блок.

Коммутативная группа (commutative group). Группа, в которой бинарная операция удовлетворяет коммутативному свойству.

Коммутативность (commutativity). В алгебраической структуре, если a и b — элементы основного набора и $\cdot$ обозначает одну из операций, коммутативное свойство состоит в том, что $a \cdot b = b \cdot a$.

Композиция (composition). Композиция двух функций f и g определяется как g $(f(x))$. Это означает, что сначала функция f применяется к переменной x и затем функция g применяется ко всему диапазону значений f.

Конечная группа (finite group). Группа с конечным числом элементов.

Конечное поле (finite field). Поле с конечным числом элементов.

Конфиденциальность (confidentiality). Цель применения средств безопасности, которая определяет процедуры, чтобы скрыть информацию от неправомочного объекта.

Конфиденциальность данных (data confidentiality) — служба безопасности, предназначенная для защиты данных от атак раскрытия, вмешательства и анализа трафика.

Коэффициент (coefficient). В полиноме — постоянное значение в каждом элементе.

Коэффициент ложной идентификации (FAR — False Acceptance Rate). Параметр, указывающий, как часто система признает неавторизованного пользователя.

Коэффициент ложной тревоги (FRR — False Rejection Rate). Параметр, указывающий, как часто система будет не в состоянии опознать человека, который должен быть распознан.

Кратковременный метод Диффи-Хеллмана (ephemeral Diffie-Hellman). Версия протокола смены ключей Диффи-Хеллмана, в которой каждая сторона передает ключ Диффи-Хеллмана, подписанный ее секретным ключом.

Криптография (cryptography). Наука и искусство преобразования сообщений с целью сделать их гарантированно устойчивыми к атакам.

Криптографический синтаксис сообщения (Cryptographic Message Syntax) — синтаксис, используемый в S/MIME, который определяет точную схему кодирования содержания каждого типа.

Криптографическая система с симметричным ключом (symmetric-key cryptosystem). Криптографическая система, в которой единственный ключ засекречивания используется и для шифрования, и для дешифрования.

Криптографическая система эллиптических кривых (elliptic curves cryptosystem). Криптосистема с асимметричными ключами на основе эллиптических кривых.

Криптографическая система RSA (RSA cryptosystem). Самый общий алгоритм с использованием открытого ключа, изобретенный Ривестом (Rivest), Шамиром (Shamir) и Адельманом (Adieman).

Криптоанализ (cryptanalysis). Наука и искусство «взлома» кодов.

Криптосистема с асимметричными ключами (asymmetric-key cryptosystem). Криптографическая система, которая использует два различных ключа для шифрования и дешифрования: открытый ключ — для шифрования и секретный ключ — для дешифрования.

Криптосистема Рабина (Rabin's cryptosystem). Вариант RSA криптографической системы, изобретенный Рабиным, в котором значение e и d установлено на 2.

Криптосистема Эль-Гамаля (ElGamal's cryptosystem) — криптосистема с асимметричными ключами, изобретенная Эль-Гамалем (ElGamal), которая базируется на проблеме дискретного логарифма.

Л

Лавинный эффект (avalanche effect) — желательная характеристика в шифре, в котором малое изменение в исходном тексте или ключе кончается большим изменением в зашифрованном тексте.

«Лазейка» (trapdoor). Особенность алгоритма, которая позволяет злоумышленнику обходить безопасность, если он знает эту особенность.

«Лазейка» в односторонней функции (TOWF — Trapdoor One-Way Function). Особенность односторонней функции, при которой кто-то может полностью изменить результат, если знает эту лазейку.

Линейные диофантовы уравнения (linear Diophantine's equations) — уравнение двух переменных формы $ax + bx = c$.

Линейное сравнение. В этом тексте — уравнение формы $ax \equiv b \pmod{n}$.

Линейный криптоанализ (linear cryptanalysis). Атака знания исходного текста, при которой используется линейная аппроксимация, чтобы анализировать блочный шифр.

Линейный регистр сдвига с обратной связью (LFSR — Linear FeedBack Shift Register). Регистр сдвига с обратной связью, в котором функция обратной связи — линейна.

Линейный S-блок (linear S-box). S-блок, в котором каждый выход является линейной функцией входов.

М

Малая теорема Ферма (Fermat's little theorem). В первой версии: если p — простое число и a — целое число, такое, что p не делит a, то $a^{p-1} = 1 \bmod p$. Во второй версии: если p — простое число и a — целое число, то $a^p = a \bmod p$.

Материал для ключей (key material). В SSL и TLS — строка переменной длины, из которой извлекаются необходимые ключи и параметры для связи.

Матрица (matrix). Прямоугольный массив $l \times$ m. элементов, в которых l — число строк, а m – число столбцов.

Матрица состояний (state). В AES — модуль данных в промежуточных каскадах, состоит из матрицы 16 байтов. В S-AES модуль данных состоит из 4 полубайтов.

Матрица-столбец (column matrix). Матрица только с одним столбцом.

Матрица-строка (row matrix) — матрица с единственной строкой.

Машина «Энигма» (Enigma machine) — шифровальная машина, базировалась на принципе роторных шифров. Использовалась немецкой армией в течение Второй мировой войны.

Международный Союз Электросвязи — Сектор Стандартизации (ITU-T — International Telecommunication Union-Telecommunication Standardization Sector). Группа международных экспертов, ответственная за стандарты связи.

Метод испытания простоты чисел квадратным корнем (square root primality test method). Метод испытания простоты чисел, основанный на факте, что квадратный корень положительного целого числа по модулю n — только +1 или −1, если n — простое.

Метод испытания простоты чисел тестом Ферма (Fermat's primality test method). Испытание простоты чисел, основанное на Малой теореме Ферма.

Метод Поларда $p - 1$ разложения на множители (Polard's p−1- factorization method). Метод, который находит для некоторого числа простой сомножитель p, при условии, что $p - 1$ не имеет коэффициента большего, чем заранее заданное значение B, называемое границей. Метод разработан Джоном М. Поллардом.

Метод Поларда rho разложения на множители (Polard's *rho* factorization method). Метод, который находит для некоторого числа простой сомножитель p. Значения, получаемые алгоритмом, повторяются, создавая форму, подобную на графике греческой букве «ро». Метод разработан Джоном М. Поллардом.

Метод разложения на множители проверкой делением (trial division factorization method). Самый простой и наименее эффективный алгоритм нахождения коэффициентов положительного целого числа, в котором пробуют все положительные целые числа, начинающиеся с 2, чтобы найти одно число, которое делит n.

Метод разложения на множители Ферма (Fermat's factorization method). Метод разложения на множители, в котором целое число n делится на два положительных целых числа a и b так, чтобы $n = a \times b$.

Механизмы безопасности (security mechanisms). Восемь механизмов, рекомендованных ITU-T, чтобы обеспечить службы безопасности: шифровка, целостность данных, цифровая подпись, аутентификация, заполнение трафика, управление маршрутизацией, доверенность и управление доступом.

Многоалфавитный шифр (polyalphabetic cipher). Шифр, в котором каждое появление символа может иметь различное значение замены.

Многоцелевое расширение почты (MIME — Multipurpose Internet Mail Extension). Протокол, который позволяет передавать данные не-ASCII через электронную почту.

Множество целых чисел (set of integers – Z). Набор всех целых чисел от отрицательной бесконечности до положительной бесконечности.

Модификация (modification). Тип атаки на целостность информации, в которой нападающий задерживает, удаляет или изменяет информацию, чтобы получить от этого выгоду для себя.

Модуль (modulus). Делитель в модульной арифметике.

Модульная арифметика (modular arithmetic). Тип арифметики, в которой при делении целого числа другим используется только один из выходов, остаток r, а частное отбрасывается.

Моноалфавитный шифр (monoalphabetic cipher). Шифр подстановки, в котором символ в исходном тексте всегда изменяется на один и тот же символ в зашифрованном тексте, независимо от его позиции в тексте.

Моноалфавитный шифр подстановки (monoalphabetic substitution cipher) — шифр, в котором ключ отображает каждый символ исходного текста в соответствующий символ зашифрованного текста.

Мультипликативная инверсия (multiplicative inverse). В модульной арифметике: a и b — мультипликативные инверсии друг друга, если $(a \times b) \bmod n = 1$.

Мультипликативный шифр (multiplicative cipher). Шифр, в котором алгоритм шифрования проводится умножением исходного текста на ключ, и алгоритм дешифрования проводится делением зашифрованного текста на ключ.

Н

Набор шифров (cipher suite). В SSL и TLS — комбинация смены ключей, хеширования и алгоритмов шифрования.

Наибольший общий делитель — НОД (GCD — Greatest Common Divisor). Наибольшее целое число, которое может делить два целых числа a и b.

Наименьший вычет (least residue). Остаток в модульной арифметике.

Начальное число (seed). Начальное значение, используемое в генераторе псевдослучайных чисел или применяемое для того, чтобы загрузить информацию в ячейки в регистр сдвига.

Национальный Институт Стандартов и Технологии (NIST — National Institute of Standards and Technology). Агентство в американском правительстве, которое развивает стандарты и технологию.

Начальный вектор (IV initial vector). Блок, используемый некоторыми режимами работы, чтобы инициализировать первую итерацию.

Нелинейный S-блок (nonlinear S-box). S-блок, в котором есть по крайней мере один выход, который определяется нелинейной функцией входов.

Неприводимый полином (irreducible polynomial). Полином степени n, не имеющий полинома-делителя степени меньше, чем n. Неприводимый полином не может быть разложен на полиномы со степенью меньше, чем n.

Несингулярная эллиптическая кривая (nonsingular elliptic curve). Эллиптическая кривая, в уравнении которой $x^3 + ax + b = 0$ имеется три различных корня.

Несинхронный шифр потока (nonsynchronous stream cipher) — шифр потока, в котором каждый ключ потока зависит от предыдущего исходного текста или зашифрованного текста.

Новые европейские схемы подписей, целостности и шифрования (NESSIE — New European Schemes for Signatures, Integrity and Encryption) — Европейская научно-исследовательская разработка, имеющая цель идентифицировать безопасные криптографические алгоритмы.

О

Область ключей (key domain). Возможный набор ключей для шифра.

Обман (spoofing). См. *имитация источника.*

Обнаружение модификации (modification detection). Дайджест сообщения, который может доказать целостность сообщения.

Обратимая функция (invertible function). Функция, которая связывает каждый элемент в диапазоне точно с одним элементом в домене.

Обратный шифр (inverse cipher). Алгоритм дешифрования.

Общая атака модуля (common modulus attack) — тип атаки RSA, которая может быть начата, если сообщество использует общий модуль.

Общедоступный ключ засекречивания (shared secret key). Ключ, используемый в криптографии с асимметричным ключом.

Ограниченная печатная строка (quoted-printable). Используемая схема кодирования, когда данные состоят главным образом из символов ASCII с незначительной частью не-ASCII. Если символ — ASCII, его передают без изменений. Если символ — не-ASCII, его передают как три символа. Первый символ — знак равенства (=). Следующие два символа — шестнадцатеричные представления байта.

Одноразовый блокнот (one-time pad) — шифр, в котором ключ является случайной последовательностью символов, имеющей ту же самую длину, что и исходный текст.

Одноразовый пароль (one-time password). Пароль, который используется только единожды.

Односторонняя функция (OWF — One-Way Function) — функция, которая может быть легко вычислена, но вычисление ее инверсии неосуществимо.

Оператор модуля (mod). Оператор, используемый в модульной арифметике, чтобы создать остаток.

Оператор сравнения (congruence operator). Оператор (≡), используемый в уравнении сравнения.

Операция разбиения (Split Operation). Операция в блочном шифре, которая разбивает блок пополам, создавая два блока равной длины.

Операция циклического сдвига (circular shift operation). Операция в современных блочных шифрах, которая удаляет k бит с одного конца и вставляет их на другом конце блока.

Оптимальное асимметричное дополнение шифрования (OAEP — Optimal Asymetric Encryption Padding). Метод, предложенный группой RSA и некоторыми поставщиками. Он применяет усложненную процедуру, чтобы дополнить сообщение для шифрования, использующего RSA.

Основной режим (main mode). В IKE — любой режим? в котором используется обмен с шестью сообщениями.

Отказ от сообщения (repudiation). Тип информационной атаки на целостность, которая может быть выполнена одной из двух сторон обмена сообщениями и состоящая в отрицании факта передачи или приема сообщения.

Открытый ключ (public key). В криптосистеме с асимметричными ключами — ключ, используемый для шифрования. В цифровой подписи такой ключ применяется для проверки.

Очень хорошая конфиденциальность (PGP — Pretty Good Privacy) — протокол, изобретенный Филом Циммерманом, чтобы обеспечить электронную почту секретностью, целостностью и аутентификацией.

П

Пассивная атака (passive attack). Тип атаки, в которой цель нападающего состоит в том, чтобы получить информацию; атака не изменяет данные и не вредит системе.

Первообразный корень (primitive root). В группе $G = <Zn*, \times>$, когда порядок элемента тот же самый, что и $\phi(n)$, этот элемент называется первообразным корнем группы.

Переигрывание (replaying). Тип атаки информационной целостности, в которой нападающий перехватывает сообщение и, не расшифровывая, позднее посылает его снова для повторного использования.

Перемешивание (confusion). Желательное свойство блочного шифра, которое скрывает отношения между зашифрованным текстом и ключом. Оно позволяет расстроить планы противника, который пробует использовать зашифрованный текст, чтобы найти ключ.

«Плюшки» (cookies). Текст, который содержит некоторую информацию о приемнике и должен быть возвращен без изменений передатчику.

Подгруппа (subgroup). Поднабор H группы G является подгруппой G, если H — группа относительно операций на G.

Подключение (connection). В протоколах SSL и TLS — процесс, который позволяет двум объектам обмениваться двумя случайными числами и создавать ключи и параметры, необходимые для связи.

Подписывающие алгоритм (signing algorithm). В схеме подписи — процесс, используемый передатчиком.

Подписи с указанием времени (time-stamped signatures). Цифровая подпись с меткой времени, не дает возможность противнику повторно использовать сообщение (переиграть сообщение).

Поле (field). Алгебраическая структура с двумя операциями, в которых вторая операция удовлетворяет все пять свойств, определенных для первой операции, за исключением того, что нейтральный элемент первой операции не имеет инверсии относительно второй операции.

Поле Галуа (Galois's field). См. *конечное поле*.

Полезная нагрузка со встроенной защитой (ESP — Encapsulating Security Payload). Протокол в IPSec, который обеспечивает аутентификацию источника, целостность и секретность.

Полином (polynomial). Выражение формы $a_n x^n + a_{n-1}x^{n-1} + ... + a_0 x^0$, где $a_i\, x^i$ называется i-тым элементом, а a_i называется коэффициентом i-того элемента.

Полуслабые ключи (semi-weak keys). Набор шести ключей в DES, где каждый ключ создает только два различных ключа раунда и каждый из них повторяется восемь раз.

Порядок группы (order of a group). Число элементов в группе.

Порядок элемента (order of an element). В группе — наименьшее положительное целое число n, такое, что $a^n = e$.

Почтовый сервер (MTA — Message Transfer Agent). Компонент электронной почты, который передает сообщения через Интернет.

Предварительный главный секретный код (pre-master secret). В SSL — информация безопасности которой обмениваются клиент и сервер перед вычислением главного секретного кода.

Предоставляющий билет сервер (TGS — Ticket-Granting Server). В Цербере — сервер, который создает билеты для реального сервера.

Претендент (claimant). В аутентификации объекта — объект, опознавательный код которого должен быть доказан.

Примитивный полином (primitive polynomial). Неприводимый полином, который делит $x^e + 1$, где e — наименьшее целое число в форме $e = 2^k - 1$.

Принцип голубиных ящиков (pigeonhole principle) — принцип, который утверждает, что если n ящиков заняты $n + 1$ голубем, то по крайней мере один ящик занят двумя голубями.

Принцип Керкхоффа (Kerckhoff's principle). Принцип в криптографии, согласно которому нужно всегда предполагать, что противник знает алгоритм шифрования/дешифрования. Поэтому сопротивление шифра атакам должно базироваться только на сохранении тайны ключа.

Проблема дня рождения (birthday problem) — классическая проблема определения вероятности того, что n людей имеют отличающиеся даты дней рождения, где $n < 365$.

Проблема логарифма эллиптических кривых (elliptic curves logarithm problem). Эта проблема состоит в нахождении по двум заданным точкам e_1 и e_2 множителя на эллиптической кривой r, такого, что $e_2 = r \times e_1$, где e_1 и e_2 — две точки, заданные на этой кривой.

Простое число (prime). Положительное целое число, которое точно делится без остатка только двумя целыми числами, на 1 и на само себя.

Протокол безопасности Интернета (IPSec Internet Protocol Security). Набор протоколов, разработанных IETF, который обеспечивает безопасность пакетов на уровне сети.

Протокол Гийу-Кискатера (Guillou-Quisquater's protocol). Дополнение протокола Фиата-Шамира, в котором может использоваться меньшее число раундов, чтобы доказать подлинность абонента.

Протокол Диффи-Хеллмана (Diffie-Hellman's protocol). Протокол создания ключей сеанса без использования центра распределения ключей (KDC).

Протокол Ниидома-Шрёдера (Needham-Schroeder's protocol). Протокол смены ключей, использующий центр распределения ключей (KDC), где применяется многократный интерактивный обмен между сторонами сообщениями «вызов-ответ».

Протокол Отвея-Рисса (Otway-Rees's protocol). Протокол смены ключей, подобный протоколу Ниидома-Шредера, но более сложный.

Протокол «от станции к станции» (station-to-station protocol). Метод создания ключей сеанса, основанный на протоколе Диффи-Хеллмана. Этот метод использует удостоверения открытого ключа, чтобы предотвратить атаки «посредника».

Протокол передачи гипертекста (HTTP — Hypertext Transfer Protocol). Обслуживает прикладной уровень, чтобы извлекать web-документы.

Протокол передачи записей (Record Protocol). В SSL и TLS — протокол, который доставляет сообщения от верхнего уровня.

Протокол установления соединения (handshake protocol). В SSL и TLS — протокол, который использует сообщения, чтобы договориться о наборе шифров, подтвердить подлинность сервера клиенту и клиенту — подлинность сервера, а также обменяться информацией для обеспечения криптографической безопасности.

Протокол Фейге-Фиат-Шамира (Feige-Fiat-Shamir's protocol). Метод установления подлинности (аутентификация) с нулевым разглашением, подобный протоколу Фиат-Шамира, но использующий вектор секретных ключей.

Протокол Фиат-Шамира (Fiat-Shamir's protocol). Метод установления подлинности с нулевым разглашением, изобретенный Фиатом и Шамиром.

Протокол ChangeCipherSpec. Протокол в SSL и TLS, который позволяет переход от ожидающего решения состояния к активному состоянию.

Прямые P-блоки (straight P-Boxes). P-блок с n входами и n выходами.

Псевдопростое число (pseudoprime). Число, которое прошло несколько испытаний на простоту чисел, но не гарантированно является простым числом.

Псевдослучайная функция (PRF — PseudoRandom Function). В TLS — функция, которая комбинирует две функции расширения данных: одну использует в MD5 и другую — в SHA-1.

P

Разложение на множители (factorization). Нахождение всех простых сомножителей целого числа.

Рассеивание (diffusion). Желательное свойство блочного шифра, которое помогает скрыть зависимости между зашифрованным текстом и исходным текстом. Оно позволяет расстроить планы противника, который использует статистику зашифрованного текста, чтобы найти исходный текст.

Расширение ключей (key expansion). В шифре раунда — процесс создания ключей раунда из ключа шифра.

Расширенный евклидов алгоритм (extended Euclidean algorithm). Алгоритм, который получает два целых числа, *a* и *b*, и может найти значения двух переменных, *s* и *t*, которые удовлетворяют уравнение $s \times a + t \times b = \text{НОД}(a, b)$. Алгоритм может также найти мультипликативную инверсию целого числа в модульной арифметике.

Раунд (round). Каждая итеративная секция в итерационном блочном шифре.

Регистр сдвига (shift register). Последовательность ячеек, где каждая ячейка содержит единственный бит. Смещение значений битов может создать псевдослучайную последовательность битов.

Регистр сдвига с нелинейной обратной связью (NLFSR — NonLinear Feedback Shift Register) — регистр сдвига с обратной связью, в котором функция обратной связи является нелинейной.

Регистр сдвига с обратной связью (FSR — Feedback Shift Register). Регистр сдвига с функцией информации обратной связи.

Режимы работы (modes of operation). Набор режимов, разработанных для того, чтобы зашифровать текст любого размера, и использующих блочные шифры фиксированных размеров.

Режим счетчика (counter(CTR) mode). Режим работы, в котором нет информации обратной связи. Он похож на режим OFB (Output FeedBack), но счетчик используется вместо регистра сдвига.

Режим шифрования с обратной связью (CFB — Cipher FeedBack). Режим работы, в котором каждый r-битовый блок складывается по модулю два с r-битовым ключом, который является частью шифрующего регистра.

Режим электронной кодовой книги (ECB mode — Electronic CodeBook mode). Режим работы, в котором каждый блок зашифрован независимо тем же самым ключом шифра.

Решето Эратосфена (sieve of Eratosthenes). Метод, изобретенный греческим математиком Эратосфеном, чтобы найти все простые числа, меньшие, чем *n*.

Роторный шифр (rotor cipher) — моноалфавитная подстановка, которая изменяет отображение между исходным текстом и символами зашифрованного текста для каждого символа исходного текста.

С

Сверхвозрастающий кортеж (superincreasing tuple). Кортеж, в котором каждый элемент больше или равен сумме всех предыдущих элементов.

Сеанс (session). В SSL — связь между клиентом и сервером. После того как сеанс установлен, эти две стороны имеют общую информацию, такую как идентификатор сеанса, сертификат, подтверждающий подлинность каждого из них (в случае необходимости), метод сжатия (если необходимо), набор шифра и главный секретный код, который используется, чтобы создать ключи для шифрования и аутентификации сообщения.

Секретный ключ (private key). В криптосистеме с асимметричными ключами — ключ, используемый для дешифрования. В цифровой подписи такой ключ используется для подписания.

Сервер аутентификации (AS — Authentication Server) — сервер, который играет роль KDC (центра распределения ключей) в протоколе Цербера.

Сеть доверия (web of trust). В PGP — кольца ключей, совместно используемые группой людей.

Сингулярная эллиптическая кривая (singular elliptic curve). Эллиптическая кривая, уравнение которой $x^3 + ax + b = 0$ не имеет трех различных корней.

Синхронный шифр потока (synchronous stream cipher). Шифр потока, в котором ключевой поток является независимым от потока зашифрованного текста или исходного текста.

Система вычетов (residue class) — набор наименьших вычетов.

Слабые ключи (weak keys). Набор из четырех ключей в DES, где каждый ключ, после того как отброшены биты четности, состоит либо из всех нулей либо из всех единиц , либо половины нулей и половины единиц.

Слепые подписи (blind signatures). Патентованная схема, разработанная Дэвидом Чомом, которая позволяет подписывать документ, не показывая содержание документа подписывающему лицу.

Слово (word). В AES — группа из 32 битов, которая может быть обработана как единственный объект: матрица-строка из четырех байтов или матрица-столбец из четырех байтов.

Случайная модель Оракл (Random Oracle Model) — идеальная математическая модель, введенная для хеш-функции.

Смеситель (mixer). В шифре Файстеля — самоконвертируемый компонент, который состоит из неконвертируемой функции и операции ИСКЛЮЧАЮЩЕЕ ИЛИ.

Современный блочный шифр (modern block cipher). Шифр с симметричным ключом, в котором каждый n-битовый блок исходного текста зашифрован как n-битовый блок зашифрованного текста, с использованием того же самого ключа.

Современный шифр потока (modern stream cipher). Шифр с симметричным ключом, в котором шифрование и дешифрование выполняются одновременно по r бит и используют ключи потока.

«Соление» пароля (salting). Метод улучшения аутентификации на основе пароля, в котором случайная строка, называемая «соль», привязывается к паролю.

Составной объект (composite). Положительное целое число, которое имеет больше чем два делителя.

Составной шифр (product cipher). Сложный шифр — это шифр, который комбинирует подстановку, перестановку и другие компоненты, чтобы обеспечить перемешивание и рассеивание. Введен Шенноном.

Список Виженера (Vigenere's tableau). Таблица шифрования и расшифровки текста в шифре Виженера.

Список ключей (key schedule). См. *расширение ключа*.

Сравнение (congruence). Если n — положительное целое число, и даны два целых числа a и b, то говорят, что они сравнимы по модулю n, $a \equiv b \pmod{n}$, если $a - b = kn$, для некоторого целого числа k.

Стандарт цифровой подписи (DSS — Digital Signature Standard). Стандарт цифровой подписи, принятый NIST под номером FIPS 186.

Стандарт шифрования данных (DES — Data Encryption Standard). Блочный шифр с симметричными ключами, использующий раунды шифров Файстеля и стандартизированный NIST.

Статистическая атака (statistical attack). Атака, основанная на знании статистики употребления символов языка.

Стратегия безопасности (SP — Security Policy). В IPSec — набор заранее заданных требований безопасности, применяемых к пакету, когда его нужно передать или принять.

Стеганография (steganography). Методика безопасности, в которой сообщение скрыто в чем-либо (например, в тексте обычного рассказа или в нотах).

Существование инверсии (existence of inverse). В алгебраической структуре: если a — элемент основного набора и • определяет одну из операций, это свойство гарантирует, что существует элемент a', называемый обратным элементом, такой, что $a \bullet a' = a' \bullet a = e$, где e — нейтральный элемент.

Существование тождественности (existence of identity). В алгебраической структуре: если a — элемент основного набора и • определяет одну из операций, это свойство гарантирует, что существует элемент e, называемый нейтральным элементом, таким, что $a \bullet e = e \bullet a = a$.

Схема Дэвиса-Мейера (Davies-Meyer's scheme). Схема хэш-функции в основном та же, что и схема Рабина (Rabin), за исключением того, что она использует прямую связь для защиты против атаки «сведения к середине».

Схема Матиса-Мейера-Оseaca (Matyas-Meyer-Oseas's scheme). Версия схемы Девиса-Мейера, в которой блок сообщения используется как ключ к криптографической системе.

Схема Меркеля-Дамгарда (Merkle-Damgard's scheme). Итеративная хэш-функция, которая устойчива к коллизиям, если функция сжатия тоже устойчива к коллизиям.

Схема Миагучи-Пренеля (Miyaguchi-Preneel's scheme). Расширенная версия схемы Матиса-Мейера-Оseaca, в которой исходный текст, ключ шифра и зашифрованный текст складывают по модулю два, чтобы создать новый дайджест.

Схема подписи Шнорра (Schnorr's signature). Схема цифровой подписи, основанная на схеме Эль-Гамаля, но с уменьшенным размером подписи.

Схема подписи Эль-Гамаля (ElGamal's signature scheme). Схема цифровой подписи в системе Эль-Гамаля, с использованием тех же самых ключей.

Схема подписи RSA (RSA signature scheme). Схема цифровой подписи, которая базируется на криптографической системе RSA, но изменяет роли секретных и открытых ключей: передатчик использует свой собственный секретный ключ, чтобы подписать документ, а приемник использует открытый ключ передатчика, чтобы проверить подпись.

Схема Рабина (Rabin's scheme). Итеративная схема хэш-функции, предложенная Рабиным на основе схемы Меркеля-Дамгарда.

Схема цифровой подписи (digital signature scheme). Метод систематического создания безопасной цифровой подписи.

Схема цифровой подписи эллиптических кривых (ECDSA — Elliptic Curves Digital Signature Scheme). Алгоритм цифровой подписи, основанный на стандарте DSA, но использующий эллиптические кривые.

Сцепление блоков шифрованного текста (CBC — Cipher Block Chaining). Режим работы, подобный электронной кодовой книге (Electronic Codebook), но каждый блок сначала складывается по модулю два с предыдущим зашифрованным текстом.

Т

Тест Казиского (Kasiski's test). Испытание для того, чтобы найти длину ключа в многоалфавитном шифре.

Тест Миллера-Рабина для определения простого числа (Miller-Rabin's primality test). Тест есть комбинация *тестов Ферма* и *квадратного корня* для нахождения сильного псевдопростого числа.

Теорема Эйлера (Euler's theorem). Обобщение Малой теоремы Ферма, в которой модуль является целым числом.

Транспортный режим (transport mode). Режим в IPSec, который защищает информацию, доставляемую от транспортного уровня до сетевого уровня.

Трехкратный DES (3DES triple DES). Шифр, которые использует три экземпляра шифров DES для шифрования и три экземпляра обратных шифров DES для дешифрования.

Трехкратный DES с тремя ключами (triple DES with three keys). Реализация трехкратного DES, где есть три ключа: K_1 , K_2 и K_3.

Трехкратный DES с двумя ключами (triple DES with two keys). Реализация трехкратного DES, где есть только два ключа: K_1 и K_2 . Первый и третьи каскады используют K_1; второй каскад использует K_2.

Триграмма (trigram). Строка с тремя буквами.

Туннельный режим (tunnel mode). Режим в IPSec, который защищает полный пакет IP. Он принимает IP пакет, включая заголовок, применяет методы безопасности IPSec к полному пакету и затем добавляет новый заголовок IP.

У

Услуги безопасности (security services). Пять услуг, связанных с целями безопасности и атаками: конфиденциальность данных, целостность данных, аутентификация, исключение отказа от сообщения и управление доступом.

Управление доступом (access control). Служба безопасности, которая защищает против неправомочного доступа к данным. Также механизм безопасности, который проверяет право пользователя обратиться к определенным данным.

Управление маршрутизацией (routing control). Механизм безопасности, который непрерывно изменяет доступные маршруты между передатчиком и приемником, чтобы не дать противнику подслушивать конкретный маршрут.

Уровень безопасных розеток (SSL — Secure Sockets Layer). Протокол, созданный для обеспечения безопасности и сжатия данных, поступивших от прикладного уровня.

Услуги обеспечения безопасности (SA — Security Association). В IPSec — логические отношения между двумя хостами.

Услуги безопасности (security services). Пять услуг, связанных с целями безопасности и атаками: конфиденциальность данных, целостность данных, аутентификация, исключение отказа от сообщения и управление доступом.

Услуги обеспечения безопасности Интернет и протокол управления ключами (ISAKMP — Internet Security Association and Key Management Protocol) — протокол, разработанный Агентством Национальной Безопасности (NASA), который усовершенствует свойства, определенные в IKE.

Усовершенствованный стандарт шифрования (AES — Advanced Encryption Standard) — блочный шифр не-Файстеля с симметричными ключами, изданный NIST.

Установление подлинности с нулевым разглашением (zero-knowledge authentication). Метод аутентификации объекта, в котором претендент не раскрывает ничего, что могло бы создать опасность конфиденциальности и безопасности. Претендент доказывает верификатору, что он знает секретную информацию, не раскрывая ее.

Устойчивость к коллизиям (collision resistance). Свойство функции криптографического хэширования. Гарантирует, что злоумышленник не может найти два сообщения, такие, которые при хэшировании приводят к тому же самому дайджесту.

Устойчивость к прообразу (preimage resistance). Желательное свойство функции криптографического хэширования, при котором даны h и y = h (M) и для противника должно быть чрезвычайно трудно найти любое сообщение M', такое, что y = h (M.').

Устойчивость ко второму прообразу (second preimage resistance). Желательное свойство в функции криптографического хэширования, в которой даны M. и h (M), и злоумышленник не может найти другое сообщение M', такое, что h (M') = h (M).

Ф

Федеральный Стандарт Обработки Информации (FIPS — FEDERAL INFORMATION PROCESSING STANDARD). Американский документ, определяющий стандарт обработки данных.

Фиксированный метод Диффи-Хеллмана (fixed Diffie-Hellman). В SSL или TLS — версия протокола Диффи-Хеллмана, в котором каждый объект может создать фиксированный полуключ и передать полуключи, внедренные в сертификат.

Фиксированный пароль (fixed-password). Пароль, который используется неоднократно для каждого доступа.

Фи-функция Эйлера (Euler's phi-function). Функция, которая находит число целых чисел, которые являются и меньшими, чем *n*, и взаимно простыми с *n*.

Функция (function). Отображение, которое связывает один элемент в наборе A, называемомоласть определения, с одним элементом в наборе B, называемом диапазон.

Функции криптографического хэширования (cryptographic hash function). Функция, которая создает намного более короткий выход информации от заданной информации на входе. Чтобы быть полезной, функция должна быть стойкой, к атакам прообраза и коллизии.

Функция обратной связи (feedback function). Функция, используемая в регистре сдвига с обратной связью. Вход к функции — значения всех ячеек; выход — значение первой ячейки.

Функция расширения данных (data expansion function). В протоколе TLS — функция, которая использует заранее заданный HAMAC (HASH — Based Message Authentication Code), чтобы увеличить длину секретных данных.

Функция сжатия (compression function). Функция, которая создает дайджест фиксированного размера из сообщения переменного размера.

Х

Характеристический полином (characteristic polynomial). Полином, представляющий функцию информации обратной связи в линейном регистре сдвига (см. линейный регистр LFSR).

Хэширование (hashing) — криптографическая методика, в которой дайджест сообщения фиксированной длины создается из сообщения переменной длины.

Хэшированное сообщение подтверждения подлинности (hashed message authentication). Подтверждение подлинности (аутентификация), использующее дайджест сообщения.

Хэш-функция Whirlpool (Whirlpool hash function). Итеративная функция криптографического хэширования, одобренная NESSIE. Основана на криптографической системе Whirlpool.

Ц

Целостность (integrity). См. *целостность данных.*

Центр сертификации (CA — Certification Authority) — организация, которая связывает открытый ключ с объектом и вырабатывает сертификат.

Циклическая подгруппа (cyclic subgroup). Подгруппа, которая может быть сгенерирована, используя возведение в степень элемента в группе.

Цели безопасности (security goals). Три цели информационной безопасности: конфиденциальность, целостность и готовность.

Целостность данных (data integrity). Услуга безопасности, разработанная для защиты данных от модификации, вставки, удаления и повторной имитации. Также механизм безопасности, который добавляет к данным (в конец) короткую проверочную комбинацию, созданную определенным процессом самостоятельно от данных. Проверка может быть использована, чтобы защитить целостность данных.

Центр распределения ключей (KDC — Key-Distribution Center). Третье лицо, которому доверяют распределение секретных ключей между двумя сторонами.

Цербер (Kerberos). Протокол аутентификации и также Центр распределения ключей (KDC), разработанный в MIT.

Цифровой Алгоритм Подписи (DSA — Digital Signature Algorithm) — цифровой алгоритм подписи, используемый Цифровым Стандартом Подписи (DSS).

Цифровая подпись (digital signature) — механизм безопасности, при котором отправитель может с помощью электроники подписать сообщение, а приемник может проверить сообщение, чтобы доказать, что сообщение действительно подписано отправителем.

Ч

Числа Ферма (Fermat's number). Множество целых чисел в форме $F_n = 2^{2^n} + 1$, где n — целое число.

Число Мерсенны (Mersenne's number). Множество целых чисел в формуле $M_p = 2^p - 1$, где p — простое число.

Ш

Шифр (cipher) — алгоритм дешифрования и/или шифрования.

Шифр Виженера (Vigenere's cipher). Многоалфавитный шифр, разработанный Блезом Виженером, в котором поток ключей является повторением начального потока ключа засекречивания.

Шифр транспозиции (transposition cipher). Шифр который перемещает символы в исходном тексте, чтобы создать зашифрованный текст.

Шифр не-Файстеля (non-Feistel's cipher). Составной шифр, который использует только обратимые компоненты.

Шифрование (encryption). Процедура получения зашифрованного текста из исходного текста при помощи криптографической системы.

Шифрование симметричными ключами (symmetric-key encipherment). Шифрование, которое использует криптографическую систему с симметричными ключами.

Шифр, ориентированный на биты (bit-oriented cipher). Шифр, в котором символы в исходном тексте, зашифрованном тексте и ключе являются битами.

Шифр, ориентированный на символ (character-oriented cipher). Шифр, в котором символы в исходном тексте, зашифрованном тексте и ключе являются буквами или цифрами.

Шифр подстановки (substitution cipher). Шифр, который заменяет один символ другим.

Шифр с автоматическим ключом. Шифр потока, в котором каждый подключ в потоке является тем же самым, что и предыдущий символ исходного текста. Первый подключ — секретный по соглашению между двумя сторонами.

Шифрование с асимметричными ключами (asymmetric-key encipherment). Шифрование, использующее криптосистему с асимметричными ключами.

Шифр с двойной перестановкой (double transposition cipher). Шифр транспозиции, в котором одни и те же алгоритмы шифрования и дешифрования повторяются с двумя ключами или одним и тем же ключом.

Шифр плейфеера (playfair's cipher). Многоалфавитный шифр, в котором ключ засекречивания состоит из 25 букв алфавита, размещенных в матрице 5 × 5.

Шифр потока (stream cipher). Тип шифра, в котором шифрование и дешифрование выполняются одновременно только с одним символом (таким как символ или бит).

Шифр Файстеля (Feistel's cipher). Класс составных шифров, состоящих и из обратимых, и из необратимых компонентов. Шифр Файстеля комбинирует все необратимые элементы в модуле (в этой книге называемом «смеситель») и использует тот же самый модуль в алгоритмах дешифрования и шифрования.

Шифр Хилла (Hill's cipher). Многоалфавитный шифр, в котором исходный текст разделен на блоки равного размера, и блоки зашифрованы по одному таким способом, что каждый символ в блоке вносит вклад в шифрование других символов в блоке.

Шифр Цезаря (Caesar's cipher). Аддитивный шифр с ключом, имеющим фиксированное значение. Использовался Юлием Цезарем.

Э

Экзистенциальная подделка (existential forgery). Тип подделки подписи, в которой подделывающий может и способен создать правильную пару подпись/сообщение, но не может реально ее использовать.

Электронная почта (electronic mail — e-mail) — электронная версия почты.

Эллиптические кривые (elliptic curves). Кубические уравнения с двумя переменными следующей формы: $y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$.

Энергичный режим (aggressive mode) — режим в IKE, который является сжатой версией соответствующего основного режима и использует обмен тремя сообщениями вместо шести.

Литература

[Bar02] Barr, T. Invitation to Cryptology. Upper Saddle River, NJ: Prentice Hall, 2002.

[Bis03] Bishop, D. Cryptography with Java Applets. Sudbury, MA: Jones and Bartlett, 2003.

[Bis05] Bishop, M. Computer Security. Reading, MA: Addison-Wesley, 2005.

[Bla03] Blahut, U. Algebraic Codes for Data Transmission. Cambridge: Cambridge University Press, 2003.

[BW00] Brassoud, D., and Wagon, S. Computational Number Theory. Emerville, CA: Key College, 2000.

[Cou99] Coutinho, S. The Mathematics of Ciphers. Natick, MA: A. K. Peters, 1999.

[DF04] Dummit, D., and Foote, R. Abstract Algebra. Hoboken, NJ: John Wiley & Sons, 2004.

[DH03] Doraswamy, H., and Harkins, D. IPSec. Upper Saddle River, NJ: Prentice Hall, 2003.

[Dur05] Durbin, J. Modern Algebra. Hoboken, NJ: John Wiley & Sons, 2005.

[Eng99] Enge, A. Elliptic Curves and Their Applications to Cryptography. Norwell, MA: Kluver Academic, 1999.

[For06] Forouzan, B. TCP/IP Protocol Suite. New York: McGraw-Hill, 2006.

[For07] Forouzan, B. Data Communication and Networking. New York: McGraw-Hill, 2007.

[FraOl] Frankkel, S. Demystifying the IPSec Puzzle. Norwood, MA: Artech House, 2001.

[GarOl] Garret, P. Making, Breaking Codes. Upper Saddle River, NJ: Prentice Hall, 2001.

[Kah96] Kahn, D. The Code breakers: The Story of Secret Writing. New York: Scribner,,1996.

[KPS02] Kaufman, C., Perlman, R., and Speciner, M. Network Security. Upper Saddle River, NJ: Prentice Hall, 2001.

[LEF04] Larson, R., Edwards, B., and Falvo, D. Elementary Linear Algebra. Boston: Houghton Mifflin, 2004.

[Mao04] Mao, W. Modern Cryptography. Upper Saddle River, NJ: Prentice Hall, 2004.

[MOV97] Menezes, A., Oorschot, P., and Vanstone, S. Handbook of Applied Cryptography. New York: CRC Press, 1997.

[PHS03] Pieprzyk, J., Hardjono, T., and Seberry, J. Fundamentals of Computer Security. Berlin: Springer, 2003.

[Res0l] Rescoria, E. SSL and TLS. Reading, MA: Addison-Wesley, 2001.
[Rhe03] Rhee, M. Internet Security. Hoboken, NJ: John Wiley & Sons, 2003.
[Ros06] Rosen, K. Elementary Number Theory. Reading, MA: Addison-Wesley, 2006.
[Sal03] Solomon, D. Data Privacy and Security. Berlin: Springer, 2003.
[Sch99] Schneier, B. Applied Cryptography. Reading, MA: Addison-Wesley, 1996.
[Sta06] Stallings, W. Cryptography and Network Security. Upper Saddle River, NJ: Prentice Hall, 2006.
[Sti06] Stinson, D. Cryptography: Theory and Practice. New York: Chapman & Hall/CRC, 2006.
[Tho00] Thomas, S. SSL and TLS Essentials. New York: John Wiley & Sons, 2000.
[TW06] Trappe, W., and Washington, L. Introduction to Cryptography and Coding Theory. Upper Saddle River, NJ: Prentice Hall, 2006.
[Vau06] Vaudenay, S. A Classical Introduction to Cryptography. New York: Springer, 2006.

Предметный указатель

Сокращения

2DES	double DES	двукратный DES
3DES	triple DES	трехкратный DES
AES	Advanced Encryption Standard	усовершенствованный стандарт шифрования
AH	Authentication Header	заголовок аутентификации
AS	authentication server	сервер аутентификации
CA	certification authority	центр сертификации
CBC	cipher-block chaining	сцепление блоков шифрованного текста
CFB	cipher feedback	кодированная обратная связь
CMAC	cipher-based message authentication code	аутентификация сообщения на основе шифра
CMS	Cryptographic Message Syntax	криптографический синтаксис сообщения
CRT	Chinese remainder theorem	китайская теорема об остатках
CTR	counter	счетчик
DES	Data Encryption Standard	стандарт шифрования данных
DSA	Digital Signature Algorithm	алгоритм цифровой подписи
DSS	Digital Signature Standard	стандарт цифровой подписи
ECB	electronic codebook	электронная кодовая книга
ECDSA	elliptic curve digital signature algorithm	алгоритм цифровой подписи, использующий метод эллиптической кривой
ESP	Encapsulating Security Payload	полезная нагрузка со встроенной защитой
FAR	false acceptance rate	коэффициент ложной идентификации
FCC	Federal Communications Commission	Федеральная Комиссия Связи
FIPS	Federal Information Processing Standard	Федеральный стандарт обработки информации
FRR	false rejection rate	коэффициент ложной тревоги
FSR	feedback shift register	регистр сдвига с обратной связью
GCD (НОД)	greatest common divisor	наибольший общий делитель
GMS	Global System for Mobile Communication	Глобальная система подвижной связи
HMAC	hashed message authentication code	код аутентификации сообщения на основе хэширования
HTTP	Hypertext Transfer Protocol	протокол передачи гипертекста
IAB	Internet Architecture Board	Совет по архитектуре Интернет

IANA	Internet Assigned Numbers Authority	Совет по регистрации доменов Интернет
ICANN	Internet Corporation for Assigned Names and Numbers	Корпорация Интернет по регистрации имен и номеров доменов
IEEE	Institute of Electrical and Electronics Engineers	Институт инженеров по электротехнике и радиоэлектронике ИИЭР
IEGS	Internet Engineering Steering Group	Группа Регулирования Разработки Интернет
IETF	Internet Engineering Task Force	Группа инженерной поддержки сети Интернет
IKE	Internet Key Exchange	протокол обмена ключами
IP	Internet Protocol	межсетевой протокол
ipad	input pad	входной блокнот
IPSec	IP Security	протокол безопасного IP
ISAKMP	Internet Security Association and Key Management Protocol	услуги обеспечения безопасности и протокол управления ключами
ISO	International Organization for Standardization	Международная Организация по Стандартизации
ISOC	Internet Society	Интернет-сообщество
ITU	International Telecommunication Union	Международный Союз Электросвязи
ITU-T	International Telecommunication Union—Telecommunication Standardization Sector	Международный Союз Электросвязи — Сектор Стандартизации
IV	initial vector	начальный вектор
KDC	key distribution center	центр распределения ключей
LFSR	linear feedback shift register	регистр сдвига с линейной обратной связью
MAA	message access agent	агент доступа к сообщению
MAC	media access control	управление доступом к среде
MAC	message authentication code	код аутентификации сообщения
MD	Message Digest	дайджест сообщения
MIC	message integrity code	код целостности сообщения
MIME	Multipurpose Internet Mail Extension	многоцелевое расширение почты
MTA	message transfer agent	почтовый агент
MTU	Maximum Transmission Unit	максимальный размер блока передаваемой информации
NESSIE	New European Schemes for Signature, Integrity, and Encryption	новые европейские схемы подписи, целостность и шифрование
NIST	National Institute of Standards and Technology	Национальный Институт Стандартов и Технологии

NLFSR	Non linear feedback shift register	регистр сдвига с нелинейной обратной связью
NSA	National Security Agency	Агентство Национальной безопасности
OAFP	optimal asymmetric encryption padding	оптимальное ассимитричное дополнение шифрования
OFB	output feedback	внешняя обратная связь
opad	outpad	Внешний блокнот
OWF	one way function	односторонняя функция
PFS	Perfect Forward Security	идеальная прямая безопасность
PGP	Pretty Good Privacy	очень хорошая конфиденциальность
PKI	public-key infrastructure	инфраструктура открытого ключа
PRF	pseudorandom function	псевдослучайная функция
PRNG	pseudorandom number generator	генератор псевдослучайных чисел
GNR	quadratic nonresidue	квадратный невычет
QR	quadratic residue	квадратичный вычет
RACE	Research and development for Advanced Communications in Europe	Программа исследований и разработок в области передовых технологий связи
RC	round constant	константа раунда
RC4	Ron's Code	код Рона
RFC	Request for Comment	запрос на комментарий
RIPMED	RACE Integrity Primitives Evaluation Message Digest	дайджест сообщения оценки примитивов целостности RACE
RNG	random number generator	генератор случайных чисел
RSA	Rivest, Shamir, Adelman	Ривест, Шамир, Адельман
SA	Security Association	услуги обеспечения безопасности
SAD	Security Association Database	база данных безопасности
SCTP	Stream Control Transmission Protocol	транспортный протокол управления потоком
SHA	Secure Hash Algorithm	алгоритм безопасного хэширования
SKEME	Secure Key Exchange Mechanism	безопасный механизм смены ключей
SHS	Secure Hash Standard	стандарт безопасного хэширования
S/MIME	Secure/Multipurpose Internet Mail Extension	безопасное/многоцелевое расширение почты
SP	Security Policy	стратегия безопасности
SPD	Security Policy Database	база данных стратегии обеспечения безопасности

SSL	Secure Sockets Layer	уровень безопасных розеток
TCP	Transmission Control Protocol	протокол управления передачей
TCP/IP	Transmission Control Protocol/ Internet Protocol	протокол управления передачей / межсетевой протокол
TELNET	TERMINAL NETWORK	телекоммуникационная сеть
TGS	ticket-granting server	предоставляющий билет сервер
TLS	Transport Layer Security	безопасность транспортного уровня
TOWF	trapdoor one-way function	«лазейка» в односторонней функции
UA	user agent	агент пользователя
UDP	User Datagram Protocol	дейтаграммный протокол пользователя
URL	uniform resource locator	унифицированный указатель (информационного) ресурса
WWW	World Wide Web	Всемирная Паутина

Учебное издание

Бехроуз А. Форузан

КРИПТОГРАФИЯ И БЕЗОПАСНОСТЬ СЕТЕЙ

Учебное пособие

Пер. с англ. под ред. А.Н. Берлина

Литературный редактор *С. Перепелкина*
Корректор *Ю. Голомазова*
Компьютерная верстка *Н. Овчинникова*
Дизайн обложки *М. Автономова*

Подписано в печать 25.10.2009. Формат 60x90 $^{1}/_{16}$.
Гарнитура «Таймс». Бумага офсетная. Печать офсетная.
Усл. печ. л. 49. Тираж 2000 экз. Заказ №

ООО «ЭКОМ Паблишерз»
Москва, ул. Бутлерова, 17А, http://www..ecom.ru

ООО «ИНТУИТ.ру»
Интернет-Университет Информационных Технологий,
Москва, Электрический пер., 8, стр.3.
E-mail: admin@intuit.ru, http://www.intuit.ru

ООО «БИНОМ. Лаборатория знаний»
Москва, проезд Аэропорта, д. 3
Телефон: (499) 157-1902, (499) 157-5272
E-mail: binom@lbz.ru, http://www.lbz.ru