МИР программирования

Н. СМАРТ

Криптография

ТЕХНОСФЕРА

МИР программирования

Н. СМАРТ

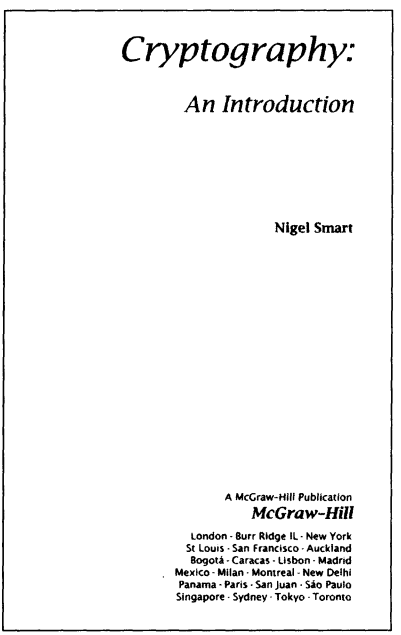
Криптография

Перевод с английского
С.А. Кулешова
под редакцией С.К. Ландо

ТЕХНОСФЕРА
Москва
2005

Н. Смарт
Криптография
Москва:
Техносфера, 2005. – 528 с. ISBN 5-94836-043-1

Один из лучших в мировой практике курсов. Предназначен специалистам, работающим в области защиты информации, и специалистам-разработчикам программного обеспечения. Чрезвычайно подробно изложены симметричные шифры, криптосистемы с открытым ключом, стандарты цифровых подписей, отражение атак на криптосистемы. Даны примеры на языке Java, многочисленные оригинальные задачи, отражающие новейшее развитие теории и практики криптографии.



ISBN I 5-94836-043-1
ISBN 0077099877 (англ.)

Содержание

Часть II. Симметричное шифрование

Предисловие

Можно спросить: зачем нужен еще один учебник по криптографии? Уже написано множество книг, одни из которых дают общее представление обо всех областях криптографии, например Шнайер (Schneier), в то время как другие посвящены энциклопедическому изложению предмета, подобно «Справочнику прикладной криптографии» (*Handbook of Applied Cryptography* или *HAC*). Однако ни одна из них не подходит для начального курса. Кроме того, технический подход к алгоритмам с открытым ключом заметно изменился за последние несколько лет, в частности, появилось понятие «доказуемой стойкости». Теперь криптографы не пускаются в длинные интуитивные обоснования безопасности своих шифров. Сейчас есть система понятий и методов, которая позволяет свести вопрос о безопасности криптосистемы к другим хорошо изученным задачам.

Курсы по криптографии в настоящее время читаются во всех ведущих университетах. Иногда их включают в программу математических факультетов, иногда — в программу факультетов вычислительной техники или электронного машиностроения. Действительно, как правило, отдельный курс по этой науке необходим студентам всех трех упомянутых специальностей, плюс некоторым учащимся, слушающим этот раздел науки в качестве дополнительного спецкурса. Подготовка студентов на этих факультетах, как и цели, стоящие перед ними, различны. Некоторым из них достаточно краткого обзора существующих алгоритмов и умения их применять, в то время как других необходимо подвести к переднему краю многообразных современных исследований. Следовательно, возникает насущная потребность в учебнике, начинающемся с самых основ и ведущего студентов сквозь дебри понятий и методов до тех пор, пока они не смогут свободно пользоваться специальной литературой.

Настоящий учебник предназначен студентам третьего – четвертого года обучения по специальности «вычислительная техника и информатика». Предполагается, что читатель знаком с основами дискретной математики (в частности, с арифметикой остатков) и имеет представление о задачах и методах теории вероятностей. Кроме того, автор надеется, что читатель прошел (хотя и плохо помнит) начальный курс математического анализа. Не то, чтобы ма-

тематический анализ был так уж необходим для изучения криптографии, но умение свободно оперировать формулами и символами *безусловно полезно. Тем не менее, вся необходимая предварительная* информация будет здесь представлена, так что учебник может читать даже тот, кто все основательно забыл или не знал вовсе. Студентам, желающим углубиться в необходимую для криптографии математику или нуждающимся в дополнительной информации, предназначено приложение А, в котором содержится выжимка из основ алгебры и приведены все понятия, необходимые для изучения современных криптосистем с отрытым ключом.

Это довольно стандартный курс для будущих специалистов в вычислительной технике и информатике, не включающий в себя многого из теории сложности и формальных методов. Большинство аналогичных курсов сконцентрировано на разработке программного обеспечения и приложениях вычислительной техники к таким областям, как графика, распознавание образов или искусственный интеллект. Главная цель этих курсов состоит скорее в подготовке специалистов-прикладников, чем в воспитании любителей покопаться в теоретических аспектах предмета. Поэтому я рассказываю о некоторых разделах теории вычислительных систем и информатики там, где они требуются. В результате одна глава посвящена приложениям теории сложности в криптографии, а другая имеет дело с формальными подходами к разработке протоколов. Обе эти главы можно читать без какой-либо предварительной подготовки в теории сложности и формальных методах.

Многие из подходов этой книги к алгоритмам с открытым ключом редукционны по своей природе, что является обычным для современной разработки протоколов и составляет главное отличие моего учебника от остальных курсов криптографии. Редукционный подход — следствие техники, применяемой в теории сложности, где принято доказывать сводимость одной сложной задачи к другой. Это делается в предположении, что трудноразрешимость второй задачи — непререкаемая истина, и показывается, как это свойство можно использовать при решении первой. В настоящей книге стойкость криптографических схем неоднократно исследуется именно таким способом, а в конце ее приводится краткий обзор доказуемой стойкости.

Учитывая интересы потенциального читателя, многие места в книге лишены абсолютной математической строгости. Часто, например, вместо полных доказательств приводятся лишь их схемы.

Тем не менее я стремился показать привлекательность используемой математики. Я пытался доступно объяснить, почему протокол был разработан именно таким, а не другим способом. Читатели, предпочитающие более глубокое и математически обоснованное изучение всех пропущенных или вскользь упомянутых моментов, могут обратиться к учебникам, подходящий к теме список которых приводится в разделе «Дополнительная литература» в конце каждой главы.

С другой стороны, терминология групп и конечных полей употребляется практически с самого начала книги. Делается это по двум причинам. Во-первых, терминология снабдит читателя необходимым словарем, чтобы он мог самостоятельно изучать отчеты о последних достижениях в области криптологии и, как следствие, приступил к самостоятельным изысканиям. Во-вторых, студенты, даже не стремящиеся продвинуться в изучении предмета до аспирантского уровня, при столкновении с «реальным миром», например при описании программных интерфейсов (API) и стандартов документов, найдут эту терминологию очень полезной. Предлагаемый курс опробован в Бристоле на студентах, у которых не было никакой подготовки в этой области математики. Практика показала, что соседство абстрактных понятий с реальными конкретными примерами и мотивациями дает очень хорошие результаты.

Я всегда полагал, что трудности при первом знакомстве с протоколами и криптосистемами заключаются в отделении открытой части алгоритма от той, которую пользователи стремятся хранить в тайне. Особенно явно убеждаешься в этом, когда впервые сталкиваешься с шифрующим алгоритмом с открытым ключом или предпринимаешь попытки расколоть шифр замены. Чтобы помочь читателю преодолеть указанные трудности, открытые чисти шифра или шифрограммы, как правило, набираются прописными буквами, а секретные (исходное сообщение, ключ) — строчными. Такое выделение будет применяться всюду, где оно может помочь разобраться в объяснениях. В других ситуациях, когда смысл написанного прозрачен или все упоминающиеся данные секретны, набор текста имеет стандартный вид.

Каждая глава организована таким образом, чтобы любой желающий мог изучать книгу самостоятельно, а именно:

- краткий список тем, раскрываемых в главе, с самого начала дающий представление о том, с чем Вам предстоит столкнуться,
- текст главы,
- краткое содержание главы, которое представлено в виде те-

зисов. Если Вы хотите что-то прочно закрепить в памяти, то это должны быть именно утверждения, приведенные в этом разделе.

- Дополнительная литература. Каждая глава содержит список нескольких книг или статей, из которых можно получить дальнейшую полезную информацию. Эти ссылки относятся, главным образом, к тому материалу, изучению которого подготовила Вас данная глава. Поскольку практически любая тема криптографии освещена в *HAC*, я не включаю этот справочник в каждый список дополнительной литературы. Он оставлен как общая книга, которой есть что сказать по каждой теме, представленной в этом учебнике.
- Упражнения. В книге они разделены на три типа:
  - Контрольные вопросы. На них следует отвечать по памяти сразу после прочтения. Фактически, они предназначены для проверки: удержалось ли у Вас в голове хоть что-нибудь в результате подробного знакомства с главой.
  - Лабораторные работы. Поскольку большинство студентов прикладных дисциплин сейчас требуют практических заданий, я включил в некоторые главы примеры возможных упражнений по программированию, рассчитанные на уровень студентов.
  - Упражнения. Они не потребуют от Вас много времени. Их цель — заставить Вас немного напрячься и глубже понять материал. В некоторых из задач требуется доказать несложные математические факты, используемые, но непроверенные в основном тексте. Поэтому многие вопросы из этого раздела можно использовать преподавателям в качестве домашнего задания или для составления экзамена.

В тексте отсутствуют ссылки на предшествующие работы. Это — учебник, и мне не хотелось ломать плавный поток изложения перекрестными ссылками. Однако Вы не должны считать отсутствие ссылок свидетельством тому, что ЛЮБОЙ из результатов книги — мой собственный. Фактически, здесь нет НИ ОДНОГО моего результата. Тот, кто хочет получить сведения о первоисточниках, должен обратиться к одной из книг, упомянутых в разделе «Дополнительная литература», или заглянуть в *HAC*.

Разные студенты в разнообразных институтах изучают различные языки программирования. В то время как студенты, обучающиеся на факультетах вычислительной математики и кибернетики,

специализируются в языке *C* или *Java*, студенты-математики вполне счастливы, овладев алгебраическим пакетом программ на подобие *Maple* или *Mathematica*. Некоторые предпочитают языки функционального программирования аналогичные *Haskell*, а другим более удобен объектно-ориентированный язык похожий на *C++*. В Бристоле мы пытаемся воспитать «многоязычных» специалистов, с первого курса преподавая множество языков программирования одновременно. Аналогичный прием использован и здесь, где применяются различные приемы (и языки) для наилучшего объяснения алгоритмов.

Задания по программированию формулируются на нейтральном языке и вполне могут быть выполнены любым студентом, не испытывающим патологического отвращения к компьютеру. Иногда при рассказе о программах я использую конкретный язык, но только тогда, когда от этого выигрывает понимаемость текста. Например, задание рекурсивной функции наиболее естественно выглядит в командах языка *Haskell*. За исключением таких ситуаций для описания алгоритмов я использую естественный язык или *C*-подобный псевдокод.

Желающим реализовать на компьютере какие-то из алгоритмов, приведенных в книге, необходимо освоить описание и оперирование с достаточно большими целыми числами и элементами конечных полей. В такие языки как *Java, Haskell* или алгебраический пакет *Maple*, достаточно большие целые числа и операции над ними уже встроены. А вот «пишущим» на языках *C* и *C++* придется организовать доступ к соответствующим библиотекам или самостоятельно написать соответствующие процедуры. Этого будет вполне достаточно для оперирования с большими целыми числами в образовательных или экспериментальных целях, но каждый должен отдавать себе отчет в том, что на практике в криптографических системах используются специальные средства повышения эффективности.

В некоторые из языков, например, *Java*, встроены доступные криптографические алгоритмы. Например, библиотека стандартных программ языка *Java* содержит возможности для вычисления хэш-функций и передачи электронной цифровой подписи. Кроме того, в нем имеются средства для работы с протоколом *X.509* (X.509 certificates). В нем также есть различные криптографические библиотеки, называемые *Java Cryptographic Engines* или *JCEs*.

Я не ставлю перед собой цель детально описать классы стандартных библиотек. Любой компетентный студент может в них ра-

зобраться в течение нескольких часов. Вместо этого в качестве помощи читателю в приложении Б приведено несколько примеров, использующих обе упомянутые библиотеки стандартных программ и программу *API* для машин *SUN JCE*. Точнее говоря, я использовал версию *Cryptix* для *SUN JCE*.

Работа со стандартными криптографическими программами и *JCE API* аналогична работе с большинством процедур языков *C* и *C++*, так что я ограничился примерами на *Java*, поскольку на сегодняшний день он является самым изучаемым языком в университетах. Доступные криптографические библиотеки дают возможность преподавателям заниматься практическими исследованиями или создавать учебные проекты для студентов.

Книга состоит из четырех частей. В первой даны краткие математические сведения, необходимые для изучения криптографии. Поэтому при первом чтении ее можно пропустить, возвращаясь к началу по мере необходимости. Часть II посвящена обсуждению симметричных алгоритмов шифрования и проблеме распределения ключей, возникающей в связи с их применением. В части III рассказано о криптосистемах с открытым ключом и электронных подписях вместе с некоторыми дополнительными темами, касающимися ключей, такими, например, как сертификаты (certificates), схемы передач и доказательство с нулевым разглашением. Часть IV — наиболее сложная и охватывает множество проблем, относящихся к более теоретической области криптографии, включая современное понятие доказуемой стойкости. Обсуждение алгоритмов с открытым ключом в третьей части было построено таким образом, чтобы подготовить читателя к восприятию основных концепций части IV. Последнюю часть следует рассматривать как щадящее, а не абсолютно строгое, введение в теоретические аспекты современной криптографии.

Тем преподавателям, кто планирует дать быстрый обзор (за 20–30 лекций) современной криптографии, я советую включить в курс главы 3, 5, 7, 10 и 12, посвятив первую лекцию главе 1, чтобы вооружить студентов информацией, необходимой для понимания последующего материала. Лекторам, читающим курс криптографии с акцентом на ее математическом фундаменте, необходимый для современных криптосистем с открытым ключом, (например для студентов, специализирующихся в математике), я предлагаю ориентироваться на главы 3, 7, 8, 9 и 11. Тем же, кто обучает будущих специалистов в вычислительной технике, вероятно, следует выбросить

из курса главы 2, 8 и 9, поскольку они слишком математизированы.

Если позволяет время, то в курс лекций можно включить практически весь материал учебника. Например, в Бристоле курс, посвященный только шифрованию, мы читаем беря за основу главы 3, 4, 5, 7 и 15. А в другом мы рассказываем о проблемах, относящихся к управлению ключами, цифровой подписи и *PKI*, т. е. проблемах, отраженных в главах 6, 10, 12 и 13. Кроме того, мы включаем во второй курс темы, касающиеся теоретико-информационной стойкости, не вошедшие в эту книгу. В нем шифрование представлено просто как «нажимание на кнопки», отвечающие за шифрование и расшифрование.

Множество тем, приобретающих все более важное значение, так и не вошло в книгу. Один из основных пробелов — подробное обсуждение линейного криптоанализа блочных шифров. Мне показалось, что эта тема больше подходит для курсовой работы или дополнительного чтения. Второе упущение — то, что я не упоминаю временной анализ, мощный дифференциальный анализ и другие стороны исследования информационных каналов. Это становится все более и более важной областью, связанной с реализацией криптографических алгоритмов, поскольку наиболее распространенные из них находятся во враждебной среде, как, например, изящная кредитная карта в Вашем бумажнике.

Я хотел бы поблагодарить студентов Бристольского университета, давших полезные комментарии к обоим нашим курсам и черновому варианту этой книги. Кроме того, следующие люди помогли мне, обеспечивая взаимосвязь между многообразными главами и темами: Ян Блэк (Ian Blake), Флориан Хесс (Florian Hess), Ник Хоугрэйв-Грэхам (Nick Howgrave-Graham), Джон Мэлон-Ли (John Malone-Lee), Венбо Мао (Wenbo Mao), Джон Мерриман (John Merriman), Фонг Нгуен (Phong Nguyen), Дэн Пэйдж (Dan Page), Винсент Риймен (Vincent Rijmen), Эдлин Теск (Edlyn Teske) и Фридерик Веркаутерен (Frederik Vercauteren). Я хотел бы выразить свою признательность сотрудникам издательства Мак-Гроу Хилл, которые мне очень помогали на всем протяжении работы над книгой, в особенности Конору Грэхем (Conor Graham) и Максу Элвей (Max Elvey).

Найджел Смарт
Университет Бристоля

Дополнительная литература

A. J. Menezes, P. van Oorschot and S. A. Vanstone. *The Handbook of Applied Cryptography*. CRC Press, 1997.

Дополнительные благодарности

Автор благодарен своим коллегам, способствовавшим в написании этой книги:

- Д-р Х. Эшман (H. Ashman) — университет Нотингема.
- Проф. Д. Эспиналл (D. Aspinall) — университет Эдинбурга.
- Д-р М. Хас (M. Huth) — Имперский колледж Лондонского университета.
- Д-р С. Джассим (S. Jassim) — университет Бэкингема.
- Д-р С. Мэрфи (S. Murphy) — Королевский колледж Лондонского университета (Holloway).
- Проф. Ф. Мэртаф (F. Murtagh) — Королевский университет, Белфаст.
- Д-р Р. Пэуит (R. Poet) — университет Глазго.
- Д-р Н. Жэан (N. Zhang) — университет Манчестера.

ЧАСТЬ I

ПРЕДВАРИТЕЛЬНЫЕ МАТЕМАТИЧЕСКИЕ СВЕДЕНИЯ

Прежде чем вплотную заняться криптографией, нам необходимо осветить некоторые основные математические факты. Многое из того, что изложено в этой части, можно найти в университетском курсе «Дискретная математика» или учебниках, посвященных программированию. Поэтому Вам далеко не все параграфы покажутся незнакомыми. Читателю, желающему узнать более формализованные определения или понятия, предназначено приложение А, расположенное в конце книги.

Эта часть — по существу краткий обзор необходимых сведений, позволяющий приступить к изучению основного материала книги. Поэтому Вы можете при желании сразу начать чтение со второй части, возвращаясь к первой по мере необходимости, тогда, когда встретите незнакомое Вам понятие.

ГЛАВА 1

АРИФМЕТИКА ОСТАТКОВ, ГРУППЫ, КОНЕЧНЫЕ ПОЛЯ И ВЕРОЯТНОСТЬ

Цели главы

- Понять арифметику остатков.
- Познакомиться с группами и конечными полями.
- Выучить основную технику — алгоритм Евклида, китайская теорема об остатках и символ Лежандра.
- Освоить основные идеи теории вероятностей.

1.1. Арифметика остатков

Эта книга во многом посвящена приложениям арифметики остатков, поскольку именно она является основой современной криптографии и криптосистем с открытым ключом в частности. В связи с этим в данной главе вводятся необходимые основные понятия и техника арифметики остатков.

Идея арифметики остатков по существу очень проста и похожа на «арифметику часов», с которой Вы знакомитесь в детстве. «Арифметика часов» связана с пересчетом времени из 24-часовой системы в 12-часовую и обратно, ведь в сутках 24 часа, а на циферблате их всего 12. Делается это довольно просто. Чтобы перевести значение в 24-часовой системе в 12-часовую, достаточно разделить его на 12. Остаток от деления и будет искомым значением в 12-часовой системе. Например, 13:00 в 24-часовой системе — то же самое, что и один час в 12-часовой, поскольку остаток от деления 13 на 12 равен 1.

Объясним более четко, что такое арифметика остатков. Прежде всего мы фиксируем положительное натуральное число N, которое называется *модулем*. Если разность двух целых чисел $b-a$ делится на N нацело, то пишут $a \equiv b \pmod N$ и говорят, что числа a и b *сравнимы* по модулю N. Очевидно, что в этом случае числа a и b имеют один и тот же остаток[1] от деления на N. Часто мы будем лениться и писать просто $a \equiv b$, если ясно по какому модулю N происходит сравнение.

Кроме того, мы будем использовать $\pmod N$ как обозначение *оператора модуля* на множестве целых чисел, который вычисляет наименьшее натуральное число, сравнимое с данным по модулю N. Например,

$$18 \pmod 7 = 4, \qquad -18 \pmod 7 = 3.$$

Оператор модуля похож на оператор % в языке программирования C, за тем исключением, что в нашей книге значения этого оператора обычно неотрицательны. Например, в языках C или *Java* оператор модуля действует так:

```
(-3)%2=-1,
```

а мы будем полагать, что $(-3) \pmod 2 = 1$.

Все возможные остатки от деления чисел на N образуют множество

$$\mathbb{Z}/N\mathbb{Z} = \{0, \ldots, N-1\}.$$

Очевидно, $\mathbb{Z}/N\mathbb{Z}$ — множество значений оператора модуля $\pmod N$. Заметим, что некоторые авторы обозначают это множество через $\mathbb{Z}_N$. Тем не менее в этой книге мы будем придерживаться обозначения $\mathbb{Z}/N\mathbb{Z}$. Еще раз отметим, что любой элемент множества $\mathbb{Z}/N\mathbb{Z}$ — это остаток от деления некоторого числа на N. Поскольку все сравнимые между собой по модулю N целые числа имеют один и тот же остаток, мы можем считать, что элемент $x \in \mathbb{Z}/N\mathbb{Z}$ изображает целый класс чисел вида $x+kN$, где $k \in \mathbb{Z}$. Таким образом, оперируя с целыми числами по модулю N, мы будем считать все сравнимые между собой числа равными друг другу и использовать традиционный знак «$=$» вместо «$\equiv$».

На множестве $\mathbb{Z}/N\mathbb{Z}$ есть две основных операции — сложение и умножение. Они определяются обычным путем, например,

$$(11+13) \pmod{16} = 24 \pmod{16} = 8,$$

[1] Отсюда название «арифметика остатков». — *Прим. перев.*

так как $24 = 1 \cdot 16 + 8$, и

$$(11 \cdot 13) \pmod{16} = 143 \pmod{16} = 15,$$

поскольку $143 = 8 \cdot 16 + 15$.

1.1.1. Группы и кольца

Сложение и умножение по модулю N работают почти так же, как арифметические операции над вещественными и целыми числами. В частности, они обладают следующими свойствами.

1. Замкнутость сложения:

$$\forall a, b \in \mathbb{Z}/N\mathbb{Z}: \quad a + b \in \mathbb{Z}/N\mathbb{Z}.$$

2. Ассоциативность сложения:

$$\forall a, b, c \in \mathbb{Z}/N\mathbb{Z}: \quad (a + b) + c = a + (b + c).$$

3. Нуль является единичным элементом (или *единицей*) по сложению:

$$\forall a \in \mathbb{Z}/N\mathbb{Z}: \quad a + 0 = 0 + a = a.$$

4. Всегда существует обратный элемент по сложению (*противоположный*):

$$\forall a \in \mathbb{Z}/N\mathbb{Z}: \quad a + (N - a) = (N - a) + a = 0.$$

5. Коммутативность сложения:

$$\forall a, b \in \mathbb{Z}/N\mathbb{Z}: \quad a + b = b + a.$$

6. Замкнутость умножения:

$$\forall a, b \in \mathbb{Z}/N\mathbb{Z}: \quad a \cdot b \in \mathbb{Z}/N\mathbb{Z}.$$

7. Ассоциативность умножения:

$$\forall a, b, c \in \mathbb{Z}/N\mathbb{Z}: \quad (a \cdot b) \cdot c = a \cdot (b \cdot c).$$

8. Число 1 является единичным элементом (единицей) по умножению:

$$\forall a \in \mathbb{Z}/N\mathbb{Z}: \quad a \cdot 1 = 1 \cdot a = a.$$

9. Умножение и сложение связаны законом дистрибутивности:

$$\forall a, b, c \in \mathbb{Z}/N\mathbb{Z}: \quad (a + b) \cdot c = a \cdot c + b \cdot c.$$

10. Коммутативность умножения:

$$\forall a, b \in \mathbb{Z}/N\mathbb{Z}: \quad a \cdot b = b \cdot a.$$

Многие множества, с которыми мы будем встречаться, наделены некоторыми из этих свойств. Введем для таких множеств специальные термины.

Определение 1.1. *Группой* называется множество с операцией, которая

- замкнута,
- обладает единицей,
- ассоциативна;
- относительно нее каждый элемент обладает обратным.

Группу с коммутативной операцией называют *коммутативной*, или *абелевой*. Почти все группы, встречающиеся в криптографии, — абелевы, поскольку именно свойство коммутативности придает им криптографический интерес. Итак, любое множество с операцией, удовлетворяющее свойствам 1, 2, 3 и 4 из перечисленных выше, называется группой, а если операция удовлетворяет еще и свойству 5, то — абелевой группой.

Стандартные примеры групп, с которыми Вы встречаетесь еще в школе, это:

- Целые, вещественные или комплексные числа по сложению. Единицей здесь является обычный 0, а обратным к x служит $-x$, т. к. $x + (-x) = 0$.
- Ненулевые рациональные, вещественные или комплексные числа. Единица в них — это 1, и обратным к x будет x^{-1}, поскольку $x \cdot x^{-1} = 1$.

Группа называется *мультипликативной*, если мы записываем ее операцию как умножение, т. е. точкой:

$$f = g \cdot h \qquad \text{и} \qquad g^5 = g \cdot g \cdot g \cdot g \cdot g.$$

Такую группу мы обозначаем $(G, \cdot)$ в том случае, когда возможны сомнения относительно используемой в ней операции. Группу называют *аддитивной*, если ее операция записывается как сложение:

$$f = g + h \qquad \text{и} \qquad 5 \cdot g = g + g + g + g + g.$$

Для аддитивных групп мы используем обозначение $(G, +)$. Абелева группа называется *циклической*, если в ней есть такой специальный элемент (*образующая*), что любой другой элемент группы получается многократным применением к нему[2] групповой операции. На-

[2] И обратному к образующей. — *Прим. перев.*

пример, в группе целых чисел по сложению любой элемент можно получить в результате многократного сложения 1 (или -1) с собой. Действительно,

$$7 = 1+1+1+1+1+1+1, \qquad -2 = (-1)+(-1).$$

Таким образом, число 1 — образующая группы целых чисел по сложению.

Если g — образующая циклической группы G, мы пишем $G = \langle g \rangle$. В мультипликативном случае каждый элемент h группы G можно записать в виде

$$h = g^x,$$

а в аддитивном —

$$h = x \cdot g,$$

где x в обоих случаях — некоторое целое число, называемое дискретным логарифмом элемента h по основанию g.

Введем понятие кольца.

Определение 1.2. *Кольцо* — это множество R с двумя операциями, сложением и умножением, традиционно обозначаемыми «+» и «·», которые обладают свойствами 1–9 со стр. 24. Мы можем обозначать кольцо вместе с его операциями тройкой $(R, \cdot, +)$.

Если окажется, что умножение в данном кольце коммутативно, то мы будем называть его *коммутативным* кольцом.

Такое определение на первый взгляд может показаться сложным, но оно просто суммирует многообразные свойства множеств, с которыми мы постоянно имеем дело. Например, Вы знакомы с бесконечными коммутативными кольцами целых, вещественных и комплексных чисел. Множество остатков $\mathbb{Z}/N\mathbb{Z}$ с операциями сложения и умножения тоже является коммутативным кольцом, которое часто называют *кольцом вычетов по модулю* N. Кольцо вычетов — один из центральных объектов криптографии, на свойствах которого основаны многие методы шифрования.

1.1.2. Функция Эйлера

Одна из центральных задач арифметики остатков — это решение уравнения

$$a \cdot x = b \pmod{N},$$

т. е. поиск элемента $x \in \mathbb{Z}/N\mathbb{Z}$, удовлетворяющего этому равенству. Линейное уравнение $ax = b$ с вещественными коэффициентами при

$a \neq 0$ всегда разрешимо. Если же рассматривать его над кольцом целых чисел, то ответ найдется не всегда. Например, не существует целого числа x, обращающего уравнение $2x = 5$ в верное равенство. Случай же кольца вычетов еще более сложный, а потому и интересный с любой точки зрения. Действительно, существует ровно одно решение уравнения

$$7 \cdot x = 3 \pmod{143},$$

уравнение

$$11 \cdot x = 3 \pmod{143}$$

решений не имеет, в то время как множество решений уравнения

$$11 \cdot x = 22 \pmod{143}$$

насчитывает 11 элементов. Таким образом, встает вопрос: при каких условиях на a, b и N уравнение $ax = b \pmod{N}$ имеет решения и как их все найти?

К счастью, существует очень простой критерий, позволяющий определить, имеет ли данное уравнение ни одного, одно или много решений. Мы просто вычисляем наибольший общий делитель чисел a и N: НОД(a, N).

- Если НОД$(a, N) = 1$, то существует ровно одно решение. Оно может быть найдено с помощью промежуточного числа c, удовлетворяющего уравнению $a \cdot c = 1 \pmod{N}$, после чего искомое неизвестное x вычисляется по формуле $x = b \cdot c \pmod{N}$.
- Если НОД$(a, N) \neq 1$ и $g =$ НОД(a, N) делит b, то уравнение будет иметь g решений. Чтобы их найти, нужно разделить исходное уравнение на g:

  $$a' \cdot x' = b' \pmod{N'},$$

  где $a' = \frac{a}{g}$, $b' = \frac{b}{g}$ и $N' = \frac{N}{g}$. Если x' — решение полученного уравнения, то решение исходного — любое число вида

  $$x = x' + i \cdot N',$$

  где $i = 0, 1, \ldots, g - 1$.
- В других случаях решений нет.

Ситуация, когда НОД$(a, N) = 1$, настолько важна, что для нее вводится специальное название. Говорят, что числа a и N *взаимно просты*.

Число элементов кольца $\mathbb{Z}/N\mathbb{Z}$, взаимно простых с N, дается функцией Эйлера φ и равно $\varphi(N)$. Значение $\varphi(N)$ можно найти по

разложению числа N на простые множители. Если

$$N = \prod_{i=1}^{n} p_i^{e_i},$$

где p_i — различные простые числа, то

$$\varphi(N) = \prod_{i=1}^{n} p_i^{e_i - 1}(p_i - 1).$$

Заметим, что последнее утверждение очень важно с точки зрения криптографии: *по разложению числа N на простые множители легко вычислить значение $\varphi(N)$*. Особый интерес представляют следующие частные случаи:

1. Если p простое, то

$$\varphi(p) = p - 1.$$

2. Если p и q — простые числа и $p \neq q$, то

$$\varphi(p \cdot q) = (p - 1)(q - 1).$$

1.1.3. Мультипликативные обратные по модулю N

Решая уравнение вида

$$ax = b \pmod{N},$$

мы приходим к вопросу о существовании мультипликативного обратного у числа a по модулю N. Иначе говоря, необходимо выяснить, существует ли число c, удовлетворяющее равенству

$$ac \equiv ca \equiv 1 \pmod{N}?$$

Такое число c естественно обозначить через a^{-1}. Как уже было отмечено, обратный к a существует только тогда, когда a и N взаимно просты, т. е. НОД $(a, N) = 1$. Особенно интересен случай простого модуля $N = p$, поскольку при этом для любого ненулевого элемента $a \in \mathbb{Z}/N\mathbb{Z}$ найдется единственное решение уравнения

$$ax = 1 \pmod{p}.$$

Итак, если p — простое число, то любой ненулевой элемент в $\mathbb{Z}/p\mathbb{Z}$ обладает мультипликативным обратным[3]. Кольца с таким свойством называются полями.

[3] То есть обратим. — *Прим. перев.*

Определение 1.3. *Полем* называется множество $(F, \cdot, +)$ с двумя операциями, обладающее дополнительными свойствами:

- $(F, +)$ — абелева группа с единичным элементом 0,
- $(F \setminus \{0\}, \cdot)$ — абелева группа с единичным элементом 1,
- $(F, \cdot, +)$ удовлетворяет закону дистрибутивности[4].

Следовательно, поле — это коммутативное кольцо, в котором каждый ненулевой элемент обратим. С полями Вы уже тоже встречались. Например, таковыми являются множества рациональных, вещественных и комплексных чисел.

Определим множество обратимых элементов в $\mathbb{Z}/N\mathbb{Z}$ как

$$(\mathbb{Z}/N\mathbb{Z})^* = \{x \in \mathbb{Z}/N\mathbb{Z} |\ \text{НОД}\,(x, N) = 1\}.$$

Для общего кольца A обозначение A^* закреплено для наибольшего его подмножества элементов, которые образуют группу по умножению. В нашей ситуации легко проверить, что множество $(\mathbb{Z}/N\mathbb{Z})^*$ — группа по умножению с числом элементов $\varphi(N)$.

В специальном случае, когда $N = p$ — простое число, получаем:

$$(\mathbb{Z}/p\mathbb{Z})^* = \{1, \ldots, p-1\},$$

поскольку каждый ненулевой элемент кольца $\mathbb{Z}/p\mathbb{Z}$ взаимно прост с p и поэтому обратим. Иными словами, $\mathbb{Z}/p\mathbb{Z}$ является конечным полем, которое обычно называется *полем вычетов по модулю* p и обозначается символом $\mathbb{F}_p$. Как следует из определения, мультипликативная группа F^* поля F совпадает с множеством $F \setminus \{0\}$. В частном случае поля вычетов получаем

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{0, \ldots, p-1\} \quad \text{и} \quad \mathbb{F}_p^* = (\mathbb{Z}/p\mathbb{Z})^* = \{1, \ldots, p-1\}.$$

Далее мы обсудим конечные поля более общего типа, а сейчас отметим лишь тот существенный момент, что целые числа по модулю N образуют поле тогда и только тогда, когда N — простое число.

Этот параграф заканчивается самой важной теоремой из элементарной теории конечных групп.

Теорема 1.4. (Теорема Лагранжа.) Если $(G, \cdot)$ — группа порядка (с числом элементов) $n = \#G$, то каждый ее элемент $a \in G$ удовлетворяет соотношению $a^n = 1$.

Таким образом, для $x \in (\mathbb{Z}/N\mathbb{Z})^*$ выполнено равенство

$$x^{\varphi(N)} = 1 \pmod{N},$$

[4]Нужно также потребовать, чтобы $0 \neq 1$. — *Прим. перев.*

поскольку $\#(\mathbb{Z}/N\mathbb{Z})^* = \varphi(N)$. Это следствие теоремы Лагранжа подводит нас к малой теореме Ферма (не путать с великой теоремой Ферма, в корне отличной от малой).

Теорема 1.5. (Малая теорема Ферма.) Предположим, что число p простое и $a \in \mathbb{F}_p^*$; тогда

$$a^p = 1 \pmod{p}.$$

Малая теорема Ферма является частным случаем теоремы Лагранжа и служит основой одного из тестов на простоту, который будет рассматриваться в следующих главах.

1.2. Конечные поля

Целые числа по простому модулю — не единственный пример конечных полей. Здесь мы разберем еще один, более общий, тип конечных полей, представляющий интерес для криптографии. Упомянутый общий тип полей будет использоваться лишь при обсуждении блочного шифра *Rijndael*, поточного шифра, основывающегося на регистре сдвига с обратными связями, и при изучении криптосистем, связанных с эллиптической кривой. Поэтому при первом чтении этот параграф можно пропустить.

Закрепим букву p за простым числом и рассмотрим множество многочленов от переменной x с коэффициентами из $\mathbb{F}_p$. Это множество обозначается через $\mathbb{F}_p[x]$ и образует кольцо относительно естественных операций суммы и умножения многочленов.

Особый интерес представляет случай $p = 2$, откуда мы и будем черпать наши примеры в данном параграфе. Например, в кольце $\mathbb{F}_2[x]$ выполнены равенства:

$$(1 + x + x^2) + (x + x^3) = 1 + x^2 + x^3,$$
$$(1 + x + x^2) \cdot (x + x^3) = x + x^2 + x^4 + x^5.$$

Можно зафиксировать многочлен $f(x)$ и рассматривать остальные элементы кольца $\mathbb{F}_p[x]$ по модулю[1] $f(x)$. Как и натуральные числа по модулю N, возможные остатки будут образовывать кольцо. Оно обозначается через

$$\mathbb{F}_p[x]/f(x)\mathbb{F}_p[x] \qquad \text{или} \qquad \mathbb{F}_p[x]/(f(x)).$$

[1] То есть оперировать с остатками от деления многочленов на $f(x)$. — *Прим. перев.*

Пусть, например, $f(x) = x^4 + 1$ и $p = 2$. Тогда

$$(1 + x + x^2) \cdot (x + x^3) \pmod{x^4 + 1} = 1 + x^2,$$

так как

$$x + x^2 + x^4 + x^5 = (x + 1) \cdot (x^4 + 1) + (1 + x^2).$$

Проверяя выписанные равенства, следует помнить, что все коэффициенты рассматриваются по модулю 2.

Напомним, что при знакомстве с целыми числами по модулю N нас интересовало уравнение

$$ax = b \pmod{N}.$$

Можно поставить аналогичный вопрос и для многочленов. Пусть a, b и f — многочлены из $\mathbb{F}_p[x]$. Существует ли решение уравнения

$$a\alpha = b \pmod{f}$$

относительно неизвестной α? Здесь, как и в целых числах по модулю N, ответ зависит от наибольшего общего делителя многочленов a и f и возможны три ситуации. Самая интригующая из них возникает при взаимно простых a и f, т. е. когда НОД $(a, f) = 1$.

Многочлен называется *неприводимым*, если у него нет делителей, отличных от него самого и констант. Таким образом, неприводимость многочленов — это то же самое, что простота целых чисел. Напомним, что целые числа по модулю N образуют поле, только если N — простое число. Точно так же, кольцо $\mathbb{F}_p[x]/(f(x))$ является конечным полем тогда и только тогда, когда многочлен $f(x)$ неприводим.

Предположим, что $p = 2$ и рассмотрим два неприводимых многочлена

$$f_1(x) = x^7 + x + 1 \quad \text{и} \quad f_2 = x^7 + x^3 + 1.$$

У нас возникают два конечных поля

$$F_1 = \mathbb{F}_2[x]/(f_1(x)) \quad \text{и} \quad F_2 = \mathbb{F}_2[x]/(f_2(x)),$$

каждое из которых состоит из 2^7 двоичных многочленов[2], степень которых не превосходит 6. Сложение в обоих полях выглядит одинаково, поскольку при вычислении суммы складываются коэффициенты многочленов по модулю 2. А вот умножаются элементы этих

[2] Действительно, любой такой многочлен имеет ровно 7 коэффициентов, равных 0 или 1. Поэтому число всех многочленов — 2^7. Легко понять, что в общей ситуации поле $\mathbb{F}_p(x)/(f(x))$ насчитывает p^n элементов, где n — степень неприводимого многочлена $f(x)$. — *Прим. перев.*

полей по-разному:

$$(x^3+1)\cdot(x^4+1) \pmod{f_1(x)} = x^4+x^3+x,$$
$$(x^3+1)\cdot(x^4+1) \pmod{f_2(x)} = x^4.$$

Возникает любопытный вопрос: действительно ли различны поля F_1 и F_2, или это только кажущееся различие? На математическом языке вопрос формулируется так: *изоморфны* ли поля F_1 и F_2? Говорят, что поля F_1 и F_2 изоморфны, если существует отображение $\varphi : F_1 \longrightarrow F_2$, называемое *изоморфизмом*, которое удовлетворяет двум требованиям:

$$\varphi(\alpha+\beta) = \varphi(\alpha)+\varphi(\beta), \qquad \varphi(\alpha\cdot\beta) = \varphi(\alpha)\cdot\varphi(\beta).$$

Оказывается, изоморфизм существует между любыми двумя конечными полями с одинаковым числом элементов. В частности, он существует и между нашими полями F_1 и F_2. Мы не будем доказывать здесь этого факта, но отметим, что для построения изоморфизма между F_1 и F_2 достаточно показать, как выражается корень $f_2(x)$ в виде многочлена от корня $f_1(x)$.

Приведенные выше конструкции по существу одинаковы и дают единственный способ построения конечных полей. Следовательно, все конечные поля фактически совпадают либо с целыми числами по простому модулю, либо с многочленами по модулю неприводимого многочлена (который тоже можно называть простым). Конечно, имеет место равенство $\mathbb{F}_p = \mathbb{F}_p[x]/(x)$. Так что можно утверждать единственность конструкции конечных полей. Итак, мы приходим к фундаментальной теореме.

Теорема 1.6. Существует единственное (с точностью до изоморфизма) конечное поле с числом элементов, равным степени простого числа.

Будем обозначать поле из $q = p^d$ элементов символом $\mathbb{F}_q$ или $GF(q)$. Обозначение $GF(q)$ означает поле Галуа из q элементов. Так (после XIX века) иногда называют конечные поля по имени французского математика Галуа. У него была интересная жизнь. Галуа написал свои блестящие работы в раннем возрасте и погиб на дуэли.

Нам необходимо объяснить несколько технических терминов, связанных с конечными полями. Любое конечное поле K содержит в себе экземпляр поля целых чисел по некоторому простому модулю p. Это поле называется *простым подполем* поля K. Число p элементов простого подполя называется *характеристикой* поля и обозначается через $\operatorname{char} K$. В частности, $\operatorname{char} \mathbb{F}_p = p$.

На конечном поле характеристики p можно определить так называемое *отображение Фробениуса*:

$$\Phi : \mathbb{F}_q \longrightarrow \mathbb{F}_q, \quad \Phi(\alpha) = (\alpha^p).$$

Легко проверить, что отображение Фробениуса — изоморфизм поля $\mathbb{F}_q$ с самим собой. Такого сорта изоморфизмы принято называть *автоморфизмами*. Интересное свойство автоморфизма Фробениуса заключается в том, что множество элементов из $\mathbb{F}_q$, остающихся неподвижными при действии Φ, совпадает с его простым подполем, т. е.

$$\{\alpha \in \mathbb{F}_q |\ \alpha^p = \alpha\} = \mathbb{F}_p.$$

Заметим, что это утверждение — обобщение малой теоремы Ферма на случай любых конечных полей. Если χ — произвольный автоморфизм конечного поля, то множество неподвижных относительно него элементов тоже образует подполе, которое принято называть *неподвижным полем* автоморфизма χ. Таким образом, предыдущее утверждение говорит о том, что неподвижное поле автоморфизма Фробениуса совпадает с простым подполем $\mathbb{F}_p$.

К тому, что поле $\mathbb{F}_q$ содержит копию $\mathbb{F}_p$, можно добавить, что $\mathbb{F}_{p^d}$ содержит подполе $\mathbb{F}_{p^e}$ для любого числа e, делящего d. Это подполе может быть определено как неподвижное поле автоморфизма Φ^e, т. е.

$$\{\alpha \in \mathbb{F}_{p^d} |\ \alpha^{p^e} = \alpha\} = \mathbb{F}_{p^e}.$$

Другое интересное свойство характеристики p поля $\mathbb{F}_q$ заключается в том, что взяв произвольный элемент $\alpha \in \mathbb{F}_q$ и сложив его с собой p раз, мы получим 0. Например в поле $\mathbb{F}_{49}$ имеет место равенство

$$X + X + X + X + X + X + X = 7X = 0 \pmod 7.$$

Ненулевые элементы конечного поля, множество которых обычно обозначается через $\mathbb{F}_q^*$, составляют конечную абелеву циклическую группу. Образующая группы $\mathbb{F}_q^*$ называется *примитивным элементом*[3] конечного поля. Подчеркнем, что примитивный элемент есть в любом конечном поле, поскольку группа его ненулевых элементов всегда циклическая. Иначе говоря, всегда можно найти такой элемент $g \in \mathbb{F}_q$, что любой ненулевой элемент $\alpha \in \mathbb{F}_q$ будет представляться в виде

$$\alpha = q^x$$

при некотором целом показателе x.

[3]Примитивных элементов, как и образующих циклических групп, может быть несколько. — *Прим. перев.*

В качестве примера рассмотрим поле из восьми многочленов

$$\mathbb{F}_{2^3} = \mathbb{F}_2[x]/(x^3 + x + 1).$$

В нем существует семь ненулевых элементов, а именно

$$1,\ \alpha,\ \alpha+1,\ \alpha^2,\ \alpha^2+1,\ \alpha^2+\alpha,\ \alpha^2+\alpha+1,$$

где α — корень многочлена[4] $x^3 + x + 1$. Убедимся, что α — примитивный элемент поля $\mathbb{F}_{2^3}$:

$$\begin{aligned} \alpha^1 &= \alpha, & \alpha^4 &= \alpha^2 + \alpha, & \alpha^7 &= 1. \\ \alpha^2 &= \alpha^2, & \alpha^5 &= \alpha^2 + \alpha + 1, & & \\ \alpha^3 &= \alpha + 1, & \alpha^6 &= \alpha^2 + 1, & & \end{aligned}$$

Отметим, что целые числа по модулю p также имеют примитивный элемент, т. к. $\mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$ — конечное поле.

1.3. Основные алгоритмы

Существует несколько численных алгоритмов или технических приемов, которые следует знать каждому читателю, т. к. они часто используются в данной книге. Здесь мы сконцентрируем внимание на

- алгоритме Евклида,
- китайской теореме об остатках,
- вычислении символов Якоби и Лежандра.

1.3.1. Наибольший общий делитель

В предыдущем параграфе мы отмечали, что для решения уравнений

$$a \cdot x = b \pmod{N}$$

в целых числах или

$$a\alpha = b \pmod{f}$$

в многочленах по простому модулю нам необходимо найти наибольший общий делитель. В частности, надо выяснить, имеет ли данный элемент $a \in \mathbb{Z}/N\mathbb{Z}$ (или $a \in \mathbb{F}_p[x]/(f)$) обратный элемент по умножению, т. е. выполнено ли равенство НОД $(a, N) = 1$ (соответственно НОД $(a, f) = 1$)? Мы не объяснили как вычислить этот наибольший

[4] Более точно, формальный корень, т. е. некий искусственно введенный элемент, удовлетворяющий соотношению $\alpha^3 + \alpha + 1 = 0$, в котором все действия выполняются по модулю 2. — *Прим. перев.*

общий делитель и не рассказали о методе построения обратного элемента при условии его существования. Исправим наше упущение описанием старейшего алгоритма, известного человечеству, а именно, алгоритма Евклида.

Если бы мы могли разложить числа a и N на простые делители, а многочлены a и f на неприводимые множители, то вычислить наибольший общий делитель не составило бы труда. Пусть, например,

$$a = 230\,895\,588\,646\,864 = 2^4 \cdot 157 \cdot 4\,513^3,$$
$$b = 33\,107\,658\,350\,407\,876 = 2^2 \cdot 157 \cdot 2\,269^3 \cdot 4\,513.$$

Из данного разложения мгновенно получается НОД:

$$\text{НОД}\,(a, b) = 2^2 \cdot 157 \cdot 4\,513 = 2\,834\,164.$$

Однако разложение на простые множители — очень трудоемкая операция, в то время как вычисление наибольшего общего делителя сравнительно несложно, что мы сейчас и продемонстрируем.

1.3.1.1. Алгоритм Евклида. Здесь мы рассмотрим только случай целых чисел. Распространить его на многочлены несложно, поскольку как целые числа, так и многочлены обладают свойством евклидовости: их можно делить с остатком. Разделить целое число a на число b с остатком — это значит найти такие числа q и r с $0 \leqslant r < |b|$, при которых выполняется равенство

$$a = q \cdot b + r.$$

Если мы хотим разделить многочлен f на многочлен g с остатком, то нам нужно найти многочлены q и r, такие что[1] $0 \leqslant \deg r < \deg g$ и

$$f = q \cdot g + r.$$

Для вычисления наибольшего общего делителя чисел $r_0 = a$ и $r_1 = b$ мы последовательно вычисляем r_2, r_3, r_4,..., производя деление с остатком по следующей схеме:

$$\begin{aligned} r_2 &= q_1 r_1 + r_0, \\ r_3 &= q_2 r_2 + r_1, \\ &\dots\ \dots \\ r_m &= q_{m-1} r_{m-1} + r_{m-2}, \\ r_{m+1} &= q_m r_m. \end{aligned}$$

Очевидно, что если число d делит как a, так и b, то оно делит и все

[1] Напомним, что символ $\deg g$ обозначает степень многочлена g, которая равна наибольшей степени входящих в него мономов. — *Прим. перев.*

r_i, начиная с $i = 0$ и заканчивая $i = m$. Следовательно,

$$\text{НОД}(a, b) = \text{НОД}(r_0, r_1) = \cdots = \text{НОД}(r_{m-1}, r_m) = r_m.$$

Продемонстрируем работу алгоритма на вычислении НОД (21, 12), равного, без сомнения, 3. Используя описанную схему, мы находим искомый делитель за несколько шагов:

$$\begin{aligned}\text{НОД}(21, 12) &= \text{НОД}(21 \pmod{12}, 12) = \\ &= \text{НОД}(9, 12) = \\ &= \text{НОД}(12 \pmod{9}, 9) = \\ &= \text{НОД}(3, 9) = \\ &= \text{НОД}(9 \pmod{3}, 3) = \\ &= \text{НОД}(0, 3) = 3.\end{aligned}$$

Приведем пример и с большими числами.

$$\begin{aligned}&\text{НОД}(1\,426\,668\,559\,730, 810\,653\,094\,756) = \\ &= \text{НОД}(810\,653\,094\,756, 616\,015\,464\,974) = \\ &= \text{НОД}(616\,015\,464\,974, 194\,637\,629\,782) = \\ &= \text{НОД}(194\,637\,629\,782, 32\,102\,575\,628) = \\ &= \text{НОД}(32\,102\,575\,628, 2\,022\,176\,014) = \\ &= \text{НОД}(2\,022\,176\,014, 1\,769\,935\,418) = \\ &= \text{НОД}(1\,769\,935\,418, 252\,240\,596) = \\ &= \text{НОД}(252\,240\,596, 4\,251\,246) = \\ &= \text{НОД}(4\,251\,246, 1\,417\,082) = \\ &= \text{НОД}(1\,417\,082, 0) = 1\,417\,082.\end{aligned}$$

Используя функциональный язык программирования, такой, например, как *Haskell*, алгоритм Евклида легко реализовать на практике, поскольку он естественно записывается в рекуррентном виде:

```
myGcd :: Int -> Int -> Int
myGcd a b = gcdstep (abc a) (abc b)

gcdstep :: Int -> Int -> Int
gcdstep a b
   | a == 0     = b
   |otherwise   = gcdstep (b `mod` a) a
```

Во многих языках программирования, таких как *Java*, *C* или пакетах компьютерной алгебры типа *Maple* из-за больших затрат

ресурсов рекуррентное определение функции может оказаться не самым лучшим способом действия. К счастью, мы не обязаны реализовывать алгоритм Евклида именно как рекурсию. Например, на *Java* мы можем написать:

```
static public long gcd(long a, long b)
{
   a=Math.abs(a);
   b=Math.abs(b);
   while (a != 0)
     { long temp=b%a;
       b=a;
       a=temp;
     }
   return b;
}
```

Работа алгоритма Евклида основана на том, что отображение

$$(a, b) \mapsto (a \pmod b, b)$$

сохраняет наибольший общий делитель. Неприятность состоит в том, что компьютерам намного легче складывать и умножать числа, чем вычислять остатки и частные. Поэтому реализация алгоритма с приведенным выше отображением обычно не столь эффективна, как хотелось бы. Однако найден ряд других подходящих отображений, которые также сохраняют НОД, но более экономичны с точки зрения компьютера, например,

$$(a, b) \mapsto \begin{cases} ((a-b)/2, b), & \text{если } a \text{ и } b \text{ нечетные;} \\ (a/2, b), & \text{если } a \text{ четное, а } b \text{ нечетное;} \\ (a, b/2), & \text{если } a \text{ нечетное и } b \text{ четное.} \end{cases}$$

Напомним, что компьютерам делить на 2 довольно легко, поскольку в двоичной системе счисления эта операция равносильна простому сдвигу разрядов[2]. Последнее отображение дает основу для *двоичного* алгоритма Евклида, который обычно и реализуется в компьютерных программах. По существу, этот алгоритм использует последнее отображение после того, как выделит из НОД максимально возможную степень двойки. Следующий пример на псевдокоде объясняет, как работает этот алгоритм с данными натуральными числами a и b.

[2] Аналогично делению на 10 в десятичной системе счисления. — *Прим. перев.*

```
   g=1 ;
/* Выделение степени двойки из НОД */
   while ((a%2==0) and (b%2==0))
     { a=a/2;
       b=b/2;
       g=2*g;
     }
/* По крайней мере одно из a и b сейчас нечетно */
   while (a != 0)
     { while ((a%2==0) {a=a/2;}
       while (b%2==0) {b=b/2;}
/* Сейчас как a так и b нечетны */
       if (a>=b) then a=(a-b)/2;
       else           b=(b-a)/2;
     }
   Return g*b;
```

1.3.1.2. Расширенный алгоритм Евклида. С помощью алгоритма Евклида, вычисляя НОД (a, N), мы можем выяснить, обратимо ли число a по модулю N. Однако мы до сих пор не знаем, как же найти обратный к a элемент, даже если он существует. Напомним, что алгоритм Евклида — это последовательное деление с остатком

$$r_{i-2} = q_{i-1}r_{i-1} + r_i, \quad i = 0, 1, \ldots, m,$$

где $r_0 = a$, $r_1 = b$ и $r_m = \text{НОД}\,(a, b)$. Сейчас мы преобразуем эти формулы, выразив все остатки r_i через a и b.

$$\begin{aligned}
r_2 &= r_0 - q_1 r_1 = a - q_1 b,\\
r_3 &= r_1 - q_2 r_2 = b - q_2(a - q_1 b) = -q_2 a + (1 + q_1 q_2)b,\\
&\ldots\ \ldots\\
r_{i-2} &= s_{i-2}a + t_{i-2}b,\\
r_{i-1} &= s_{i-1}a + t_{i-1}b,\\
r_i &= r_{i-2} - q_{i-1}r_{i-1} =\\
&= a(s_{i-2} - q_{i-1}s_{i-1}) + b(t_{i-2} - q_{i-1}t_{i-1}),\\
&\ldots\ \ldots\\
r_m &= s_m a + t_m b.
\end{aligned}$$

Расширенный алгоритм Евклида по данным целым числам a и b выдает r_m, s_m и t_m, так что

$$r_m = \text{НОД}\,(a, b) = s_m a + t_m b.$$

Теперь мы готовы решить исходную задачу по определению обратного элемента для a по модулю N, если это в принципе возможно сделать. Сначала мы применяем расширенный алгоритм Евклида к числам a и N и получаем такие числа d, s и t, что

$$d = \text{НОД}(a, N) = sa + tN.$$

Значит, $d = sa + tN = sa \pmod N$. Теперь видно, что уравнение $ax = 1 \pmod N$ имеет решение только тогда, когда $d = 1$. При этом решение имеет вид $x = a^{-1} = s$.

В качестве примера вычислим обратный элемент к 7 по модулю 19. Положим $r_0 = 7$, $r_1 = 19$ и проведем описанную процедуру.

$$r_2 = 5 = 19 - 2 \cdot 7,$$
$$r_3 = 2 = 7 - 5 = 7 - (19 - 2 \cdot 7) = -19 + 3 \cdot 7$$
$$r_4 = 1 = 5 - 2 \cdot 2 = (19 - 2 \cdot 7) - 2 \cdot (-19 + 3 \cdot 7) = 3 \cdot 19 - 8 \cdot 7.$$

Отсюда

$$1 = -8 \cdot 7 \pmod{19},$$

так что

$$7^{-1} = -8 = 11 \pmod{19}.$$

1.3.2. Китайская теорема об остатках

Китайская теорема об остатках или КТО также является очень древней находкой математики, возраст которой не менее 2000 лет. Мы будем использовать КТО неоднократно, например, для улучшения процедуры расшифрования в криптосистеме RSA и в некоторых других протоколах. Говоря неформально, КТО утверждает, что система уравнений

$$\begin{cases} x = a \pmod N, \\ x = b \pmod M \end{cases}$$

имеет единственное решение по модулю $M \cdot N$ тогда и только тогда, когда M и N взаимно просты. Кроме того, она дает простой метод поиска этого решения. Например система

$$\begin{cases} x = 4 \pmod 7, \\ x = 3 \pmod 5 \end{cases}$$

будет иметь решением $x = 18 \pmod{35}$. Легко убедиться в истинности найденного ответа, поскольку $18 \pmod 7 = 4$ и $18 \pmod 5 = 3$. Но как он был найден?

Сначала мы покажем «на пальцах», как решить эту конкретную задачу, а затем расскажем об общем методе. Итак, у нас есть уравнения

$$x = 4 \pmod 7 \quad \text{и} \quad x = 3 \pmod 5.$$

Если x удовлетворяет каждому из них, то для некоторого целого числа u должны быть выполнены равенства:

$$x = 4 + 7u \quad \text{и} \quad x = 3 \pmod 5.$$

Подставляя первое из соотношений во второе, получаем

$$4 + 7u = 3 \pmod 5.$$

Сделав преобразования, придем к уравнению

$$2u = 7u = 3 - 4 = 4 \pmod 5.$$

Поскольку НОД $(2, 5) =$ НОД $(7, 5) = 1$, мы можем решить последнее уравнение относительно u. Прежде всего вычислим $2^{-1} \pmod 5 = 3$, поскольку $2 \cdot 3 = 6 = 1 \pmod 5$. Затем найдем значение

$$u = 2^{-1} \cdot 4 \pmod 5 = 3 \cdot 4 \pmod 5 = 2 \pmod 5.$$

Теперь осталось подставить найденное значение u в выражение для x и получить решение

$$x = 4 + 7u = 4 + 7 \cdot 2 = 18.$$

Система двух уравнений имеет настолько важное значение, что мы приведем общую формулу ее решения. Предположим, что числа M и N взаимно просты и дана система

$$\begin{cases} x = a \pmod N, \\ x = b \pmod M. \end{cases}$$

Сначала определим

$$T = M^{-1} \pmod N,$$

что возможно сделать ввиду предположения о взаимной простоте чисел M и N. Затем вычисляем

$$u = (b - a)T \pmod N.$$

Решение по модулю $M \cdot N$ задается формулой

$$x = a + uM.$$

Чтобы убедиться в верности метода, сделаем проверку.

$$x \pmod M = a + uM \pmod M = a,$$
$$x \pmod N = a + uM \pmod N = a + (b-a)TM \pmod N =$$
$$= a + (b-a)M^{-1}M \pmod N = b.$$

Вернемся к общему случаю КТО, посвященному системе из более чем двух уравнений. Пусть $m_1, \ldots, m_r$ и $a_1, \ldots, a_r$ — целые числа, причем все m_i попарно взаимно просты. Нужно найти такой элемент x по модулю $M = m_1 m_2 \cdots m_r$, что

$$x = a_i \pmod{m_i} \text{ для всех } i.$$

Китайская теорема об остатках гарантирует существование и единственность решения и дает ответ:

$$x = \sum_{i=1}^{r} a_i M_i y_i \pmod M,$$

где

$$M_i = M/m_i, \qquad y_i = M_i^{-1} \pmod{m_i}.$$

В качестве примера найдем число x по модулю $M = 1001 = 7 \cdot 11 \cdot 13$, такое что

$$\begin{cases} x = 5 \pmod 7, \\ x = 3 \pmod{11}, \\ x = 10 \pmod{13}. \end{cases}$$

Руководствуясь методом, вычисляем

$$M_1 = 143, \qquad y_1 = 5,$$
$$M_2 = 91, \qquad y_2 = 4,$$
$$M_3 = 77, \qquad y_3 = 12.$$

Пользуясь формулой, получаем решение

$$x = \sum_{i=1}^{r} a_i M_i y_i \pmod M =$$
$$= 715 \cdot 5 + 364 \cdot 3 + 924 \cdot 10 \pmod{1001} = 894.$$

1.3.3. Символы Лежандра и Якоби

Пусть p — простое число, большее 2. Рассмотрим отображение

$$\mathbb{F}_p \longrightarrow \mathbb{F}_p, \qquad \alpha \mapsto \alpha^2,$$

сопоставляющее каждому элементу поля его квадрат. На множестве ненулевых элементов поля $\mathbb{F}_p$ это отображение в точности «два-в-один», т. е. если из ненулевого элемента $x \in \mathbb{F}_p$ можно извлечь квадратный корень, то таких корней у него ровно 2 и, кроме того, ровно половина элементов из $\mathbb{F}_p^*$ являются полными квадратами. Полные квадраты в $\mathbb{F}_p^*$ называются *квадратичными вычетами* по модулю p. Множество всех квадратичных вычетов по модулю p является подгруппой порядка $(p-1)/2$ в мультипликативной группе $\mathbb{F}_p^*$. Элементы группы $\mathbb{F}_p^*$, из которых нельзя извлечь квадратный корень, называются *квадратичными невычетами*.

Для выявления полных квадратов по простому модулю p вводится *символ Лежандра*

$$\left(\frac{a}{p}\right).$$

Он равен 0, если a делится на p, $+1$, если a — квадратичный вычет по модулю p, и -1, если a — квадратичный невычет.

Символ Лежандра легко вычисляется, например, по формуле

$$\left(\frac{a}{b}\right) = a^{(p-1)/2} \pmod p.$$

Однако использование этой формулы сопряжено с вычислениями больших степеней и на практике предпочитают пользоваться *законом квадратичной взаимности*

$$\left(\frac{q}{p}\right) = \left(\frac{p}{q}\right)(-1)^{(p-1)(q-1)/2}. \tag{1.1}$$

Иначе говоря,

$$\left(\frac{q}{p}\right) = \begin{cases} -\left(\frac{p}{q}\right), & \text{если } p = q = 3 \pmod 4, \\ \left(\frac{p}{q}\right) & \text{в других случаях.} \end{cases}$$

При вычислении символа Лежандра большую помощь оказывают следующие дополнительные формулы:

$$\left(\frac{q}{p}\right) = \left(\frac{q \pmod p}{p}\right), \tag{1.2}$$

$$\left(\frac{q \cdot r}{p}\right) = \left(\frac{q}{p}\right) \cdot \left(\frac{r}{p}\right), \tag{1.3}$$

$$\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}. \tag{1.4}$$

Используя разложение на множители, вычислим символ Лежандра.

$$\left(\frac{15}{17}\right) = \left(\frac{3}{17}\right)\cdot\left(\frac{5}{17}\right) = \left(\frac{17}{3}\right)\cdot\left(\frac{17}{5}\right) = \quad \text{(по форм. (1.3) и (1.1))}$$

$$= \left(\frac{2}{3}\right)\cdot\left(\frac{2}{5}\right) = (-1)\cdot(-1)^3 = 1 \qquad \text{(по форм. (1.2) и (1.4))}$$

Через некоторое время мы изучим более эффективный алгоритм вычисления символа Лежандра, не зависящий от возможности разложения числа на множители.

Извлечение квадратного корня из квадратичного вычета a по модулю p — тоже несложная задача. Сейчас мы приведем метод ее решения, называемый алгоритмом Шэнкса.

1. Выберем наугад такое n, что
$$\left(\frac{n}{p}\right) = -1.$$
2. Пусть e, q — целые числа с нечетным q, удовлетворяющие соотношению $p-1 = 2^e q$.
3. Положим $y = n^q \pmod p$, $r = e$, $x = a^{(q-1)/2} \pmod p$.
4. Положим $b = ax^2 \pmod p$, $x = ax \pmod p$.
5. Пока $b \neq 1 \pmod p$ делать:
   - найти наименьшее число m, такое что $b^{2^m} = 1 \pmod p$,
   - положить $t = y^{2^{r-m-1}} \pmod p$, $y = t^2 \pmod p$, $r = m$,
   - положить $x = xt \pmod p$, $b = by \pmod p$.
6. Вывести x.

Если $p = 3 \pmod 4$, то для извлечения корня из a можно использовать формулу

$$x = a^{(p+1)/4} \pmod p,$$

которая имеет неоспоримое преимущество перед общим алгоритмом Шэнкса ввиду ее явности и эффективности. Формула дает правильный ответ потому, что

$$x^2 = a^{(p+1)/2} = a^{(p-1)/2}\cdot a = \left(\frac{a}{p}\right)\cdot a = a.$$

Последнее равенство здесь верно ввиду предположения о том, что a — квадратичный вычет, а значит, соответствующий символ Лежандра равен 1.

Символ Лежандра определен только в случае простого знаменателя. Если же знаменатель составной, то вводится *символ Якоби*,

обобщающий символ Лежандра. Пусть n — нечетное число, большее 2 и

$$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}.$$

Символ Якоби определяется через символы Лежандра простых делителей числа n следующим образом

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \left(\frac{a}{p_2}\right)^{e_2} \cdots \left(\frac{a}{p_k}\right)^{e_k}.$$

Символ Якоби можно вычислять так же, как и символ Лежандра, опираясь на тождество, выведенное из закона квадратичной взаимности:

$$\left(\frac{a}{n}\right) = \left(\frac{2}{n}\right)^{e} \left(\frac{n \pmod{a_1}}{a_1}\right) (-1)^{(a_1-1)(n-1)/4},$$

где $a = 2^e a_1$ и a_1 нечетно. Полезно помнить еще несколько формул, справедливых при нечетном n:

$$\left(\frac{1}{n}\right) = 1,$$

$$\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8},$$

$$\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}.$$

Все это дает нам быстрый алгоритм вычисления символа Якоби и, соответственно, символа Лежандра, как его частный случай, без утомительного разложения на множители. Единственно, что нужно сделать, это выделить максимальную степень двойки:

$$\left(\frac{15}{17}\right) = (-1)^{56} \left(\frac{17}{15}\right) = \left(\frac{2}{15}\right) = (-1)^{28} = 1.$$

Напомним, что символ Лежандра $\left(\frac{a}{p}\right)$ сообщает нам, является ли a полным квадратом по модулю простого числа p. Увы, символ Якоби $\left(\frac{a}{n}\right)$ ничего не утверждает о возможности извлечения квадратного корня из a по модулю составного числа n. Если a в действительности квадрат по модулю n, то символ Якоби будет равен $+1$. Однако из равенства $\left(\frac{a}{n}\right) = +1$ нельзя сделать вывод о том, что a — полный квадрат. Несмотря на благоприятное равенство, квадратный корень из a может не извлекаться!

Пусть $n \geqslant 3$ — нечетное число. Обозначим через Q_n подмножество полных квадратов в $(\mathbb{Z}/n\mathbb{Z})^*$:

$$Q_n = \{x^2 \pmod{n} \mid x \in (\mathbb{Z}/n\mathbb{Z})^*\}.$$

С другой стороны, пусть J_n обозначает подмножество элементов в $(\mathbb{Z}/n\mathbb{Z})^*$, чей символ Якоби равен 1, т. е.

$$J_n = \left\{ x \in (\mathbb{Z}/n\mathbb{Z})^* \,\middle|\, \left(\frac{a}{n}\right) = 1 \right\}.$$

Таким образом, множество всех псевдо-квадратов[3] — это разность $J_n \backslash Q_n$. Криптографов интересуют два относительно простых частных случая: когда n простое число или произведение двух простых чисел.

- Если n — простое число, то
  - $Q_n = J_n$,
  - $\#Q_n = (n-1)/2$.
- Если $n = pq$ — произведение двух простых, то
  - $Q_n \subset J_n$,
  - $\#Q_n = \#(J_n - \mathbb{Q}_n) = (p-1)(q-1)/4$.

Как мы еще увидим, множества J_n и Q_n сыграют заметную роль в ряде криптографических алгоритмов и протоколов, особенно когда n — произведение двух простых чисел.

В заключение параграфа выясним, как извлечь корень из числа по модулю составного $n = p \cdot q$. Предположим, нам надо найти квадратный корень из a по модулю n. Вообще говоря, разложение числа n на простые множители может быть неизвестно, но, допустим, мы его знаем. Будем также считать, что a действительно полный квадрат по модулю n, т. е.

$$\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = 1.$$

Сначала мы извлечем корень из a по модулю p и обозначим его через s_p. Затем извлечем корень из a по модулю q и назовем его s_q. Наконец, для вычисления искомого корня мы применяем китайскую теорему об остатках к системе

$$\begin{cases} x = s_p \pmod{p}, \\ x = s_q \pmod{q}. \end{cases} \tag{1.5}$$

Вычислим, например, корень из $a = 217$ по модулю $n = 221 = 13 \cdot 17$. Квадратные корни из a по модулям 13 и 17 соответственно равны $s_{13} = 3$ и $s_{17} = 8$. Опираясь на китайскую теорему об остатках,

[3] То есть квадратичных невычетов, чей символ Якоби равен 1. — *Прим. перев.*

получаем $s = 42$. Проверим правильность найденного решения прямыми вычислениями:

$$s^2 = 42^2 = 217 \pmod{221}.$$

На самом деле, существуют еще три разных квадратных корня из числа 217 по модулю $n = 221$, поскольку n имеет два простых делителя. Чтобы их отыскать, достаточно применить КТО к трем системам вида (1.5) с

$$\begin{aligned} s_{13} &= 10, & s_{17} &= 8; \\ s_{13} &= 3, & s_{17} &= 9; \\ s_{13} &= 10, & s_{17} &= 9 \end{aligned}$$

и получить полный ответ:

$$42,\ 94,\ 127,\ 179.$$

1.4. Вероятность

В какой-то момент нам потребуется понимание основ элементарной теории вероятностей. Сейчас мы кратко дадим теоретические сведения и разберем несколько примеров. Для большинства читателей изложенный здесь материал знаком по учебе в ВУЗе.

Случайной величиной называется переменная X, принимающая свои значения с некоторой вероятностью. Если переменная X принимает значение s с вероятностью 0,01, мы будем писать

$$p(X = s) = 0,01.$$

Предположим, например, что T — случайная величина, представляющая результат подбрасывания симметричной монеты. Так как появление орла или решки равновероятны, то

$$p(T = \textit{орел}) = \frac{1}{2}, \qquad p(T = \textit{решка}) = \frac{1}{2}.$$

В качестве другого примера рассмотрим английский текст и обозначим через E случайную величину, принимающую буквенные значения. Статистический анализ большого количества информации позволяет оценить относительные вероятности появления той или иной буквы в тексте:

$$\begin{aligned} &p(E = a) = 0,082, \\ &\ldots \\ &p(E = e) = 0,127, \\ &p(E = z) = 0,001. \end{aligned}$$

Если X — дискретная случайная величина[1], то множество вероятностей всех ее значений называют *распределением вероятностей*, а функцию $p(X = x)$, сопоставляющую значению переменной соответствующую вероятность — *плотностью распределения вероятностей*. В общем случае имеют место следующие свойства:

$$p(X = x) \geqslant 0, \qquad \sum_x p(X = x) = 1.$$

Классический пример, иллюстрирующий теорию вероятностей, связан со стандартной колодой в 52 карты. Следуя традиции, расскажем о нем и мы. Обозначим через V случайную величину появления карты определенного достоинства при сдаче, через S — масть карты, а через C — ее цвет. Тогда

$$p(C = \textit{красный}) = \frac{1}{2},$$
$$p(V = \textit{туз треф}) = \frac{1}{52},$$
$$p(S = \textit{трефа}) = \frac{1}{4}.$$

Пусть X и Y — две случайные величины с распределением вероятностей $p(X = x)$ и $p(Y = y)$. Вероятность одновременного равенства $X = x$ и $Y = y$ называется *совместной вероятностью* $p(X = x, Y = y)$. Так, если $X = C$ и $Y = S$, то

$$p(C = \textit{красный}, S = \textit{трефа}) = 0, \quad p(C = \textit{красный}, S = \textit{бубна}) = \frac{1}{4},$$
$$p(C = \textit{красный}, S = \textit{черва}) = \frac{1}{4}, \quad p(C = \textit{красный}, S = \textit{пика}) = 0,$$
$$p(C = \textit{черный}, S = \textit{трефа}) = \frac{1}{4}, \quad p(C = \textit{черный}, S = \textit{бубна}) = 0,$$
$$p(C = \textit{черный}, S = \textit{черва}) = 0, \quad p(C = \textit{черный}, S = \textit{пика}) = \frac{1}{4}.$$

Две случайные величины X и Y называются *независимыми*, если для всех возможных значений x и y имеет место равенство

$$p(X = x, Y = y) = p(X = x) \cdot p(Y = y).$$

Таким образом, случайные величины C и S не являются *независимыми*. В качестве примера независимых величин рассмотрим эксперимент, во время которого подбрасываются две правильные монеты последовательно друг за другом. Пусть T_1 — итог подбрасывания первой монеты, а T_2 — второй. Поскольку в силу физических

[1] То есть ее множество значений конечно или счетно. — *Прим. перев.*

законов результат подбрасывания первой монеты никак не может сказаться на подбрасывании второй, можно предположить, что T_1 и T_2 — *независимые случайные величины*. Это подтверждается совместным распределением вероятностей:

$$p(T_1 = \textit{орел}, T_2 = \textit{орел}) = \frac{1}{4}, \qquad p(T_1 = \textit{орел}, T_2 = \textit{решка}) = \frac{1}{4},$$
$$p(T_1 = \textit{решка}, T_2 = \textit{орел}) = \frac{1}{4}, \qquad p(T_1 = \textit{решка}, T_2 = \textit{решка}) = \frac{1}{4}.$$

1.4.1. Теорема Байеса

Условной вероятностью $p(X = x|Y = y)$ случайных величин X и Y называется вероятность того, что переменная X принимает значение x, если известно, что $Y = y$.

Возвращаясь к примеру с колодой карт, имеем

$$p(S = \textit{пика}|C = \textit{красный}) = 0$$

и

$$p(V = \textit{туз пик}|C = \textit{черный}) = \frac{1}{26}.$$

Первое равенство следует из того, что пика имеет черный цвет и указанные события не могут произойти одновременно. Во втором случае условие $C = \textit{черный}$ ограничивает выбор карты ровно наполовину, и поскольку у нас есть только один туз пик среди 26 возможных карт, соответствующая вероятность равна $\frac{1}{26}$.

Сформулируем одну из самых значимых теорем элементарной теории вероятностей

Теорема 1.7. (Теорема Байеса.) Если $p(Y = y) > 0$, то

$$p(X = x|Y = y) = \frac{p(X = x) \cdot p(Y = y|X = x)}{p(Y = y)} = \\ = \frac{p(X = x, Y = y)}{p(Y = y)}.$$

Применим теорему Байеса к нашему примеру с картами.

$$p(S = \textit{пика}|C = \textit{красный}) = \frac{p(S = \textit{пика}, C = \textit{красный})}{p(C = \textit{красный})} = \\ = 0 \cdot \left(\frac{1}{4}\right)^{-1} = 0.$$

$$p(V = \textit{туз пик}|C = \textit{черный}) = \frac{p(V = \textit{туз пик}, C = \textit{черный})}{p(C = \textit{черный})} = \\ = \frac{1}{52} \cdot \left(\frac{1}{2}\right)^{-1} = \frac{2}{52} = \frac{1}{26}.$$

Для независимых случайных величин имеем

$$p(X = x|Y = y) = p(X = x),$$

т. е. значение величины X не зависит от того, чему равна случайная величина Y.

1.4.2. Парадокс дней рождения

Еще один полезный результат из элементарной теории вероятностей, который нам потребуется в дальнейшем, называется парадоксом дней рождения. Допустим, в ящике находятся m шариков разного цвета. Из ящика не глядя достают один шарик, записывают его цвет, возвращают его в ящик и тянут снова. Вероятность того, что после n вытаскиваний нам попадется хотя бы два шарика одного цвета, равна

$$1 - \frac{m^{(n)}}{m^n},$$

где

$$m^{(n)} = m(m-1)(m-2)\cdots(m-n+1).$$

Если m достаточно большое, то ожидаемое число шариков, которые нам придется просмотреть до первого совпадения, равно $\sqrt{\frac{\pi m}{2}}$.

Чтобы понять причину названия парадокса, оценим вероятность совпадения дней рождений двух людей из некоторой группы. Многие наивно полагают, что такая вероятность крайне мала, считая практически невозможным совпадение чьей-то даты рождения с его собственной. Однако, воспользовавшись выписанной формулой, легко убедиться, что вероятность совпадения дат рождений у двух людей из 23 равна

$$1 - \frac{365^{(23)}}{365^{23}} = 0,507.$$

На самом деле вероятность совпадения быстро возрастает с ростом количества людей, из которых производится выборка. Так она равна 0,706, если выбираем среди 30 человек, и 0,999 999 6, если среди 100.

Краткое содержание главы

- Группа — это множество с ассоциативной операцией и единицей, все элементы которого обратимы. Арифметика остатков

вместе с операцией сложения по определенному модулю доставляет примеры групп. Хотя с умножением необходимо быть осторожным, мы можем определить группу и относительно умножения по модулю.

- Кольцо — это множество с двумя операциями, обладающими теми же свойствами, что умножение и сложение натуральных чисел. Остатки от деления на целое число — пример кольца.
- Поле — это коммутативное кольцо, в котором каждый ненулевой элемент обладает мультипликативным обратным. Кольцо вычетов по модулю простого числа — пример поля.
- Обратить элемент в кольце вычетов можно на основе расширенного алгоритма Евклида.
- Система линейных уравнений в арифметике остатков может быть решена с помощью китайской теоремы об остатках.
- Полные квадраты по модулю простого числа выявляются символом Лежандра. Квадратные корни можно эффективно извлекать по алгоритму Шэнкса.
- Полные квадраты и квадратные корни по модулю составного числа n могут быть эффективно вычислены, если известны все делители числа n.
- Теорема Байеса позволяет вычислять условные вероятности.
- Парадокс дней рождения позволяет нам оценить, насколько быстро появится повторение, если производить выборку из конечного множества.

Дополнительная литература

Бэч и Шаллит — наилучшее известное мне введение в конечные поля и алгоритм Евклида. Эта книга содержит довольно много исторической информации и превосходный указатель важной исследовательской литературы. Предназначенная для специалистов в области информатики, некоторым она может показаться слишком математезированной. Более традиционное введение в основы дискретной математики, сложность которого можно оценить первым курсом информатики — это учебники Биггса и Розена.

E. Bach and J. Shallit. *Algorithmic Number Theory. Volume 1: Efficient Algorithms*. MIT Press, 1996.

N. L. Biggs. *Discrete Mathematics*. Oxford University Press, 1989.

K. H. Rosen. *Discrete Mathematics and its Applications*. McGraw-Hill, 1999.

Контрольные вопросы

1.1.1. Сформулируйте определения группы, кольца и поля.

1.1.2. Почему ненулевые целые числа не образуют группу по умножению?

1.1.3. Почему мы учитываем только ненулевые числа, говоря о вещественных или рациональных числах как о группе по умножению?

1.1.4. Почему множество целых чисел не является полем?

1.1.5. Почему $\mathbb{F}_p[x]/(f(x))$ всегда кольцо?

1.1.6. Чему равно значение функции Эйлера $\varphi(N)$ при $N = p$ и $N = p \cdot q$, где p и q — простые числа?

1.1.7. Дайте определение символа Лежандра и объясните, как его использовать для выяснения того, является ли данное число полным квадратом по модулю простого числа p.

1.1.8. Верно ли, что число a по модулю составного n является полным квадратом, если его символ Якоби относительно n равен 1?

1.1.9. Сколько раз нужно вытаскивать шарик из ящика с шестьюдесятью шариками разного увета (и возвращать его обратно), пока не попадется цвет, который уже вытаскивали?

Лабораторные работы

1.2.1. Напишите программу (на знакомом Вам языке) для стандартного и двоичного алгоритмов Евклида вычисления НОД. Сравните продолжительность их работы, постепенно увеличивая их входные данные.

1.2.2. Разработайте программу, реализующую расширенный алгоритм Евклида. Сначала используйте стандартное отображение, сохраняющее НОД, а затем двоичное.

1.2.3. Реализуйте на компьютере алгоритм вычисления символа Лежандра на основе квадратичного закона взаимности.

Упражнения

1.3.1. Покажите, что стандартный и расширенный алгоритмы Евклида могут быть приспособлены для работы с многочленами.

1.3.2. Напишите аналог двоичного расширенного алгоритма Евклида для многочленов с двоичными коэффициентами. Используя его, вычислите обратный элемент к многочлену $x^7 + x + 1$ в конечном поле $\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$.

1.3.3. Числа Фибоначчи определяются по рекуррентному правилу:

$$F_0 = 0,\ F_1 = 1,\ F_{n+1} = F_n + F_{n-1}.$$

Покажите, что если все неполные частные q_i, которые появляются в алгоритме Евклида, вычисляющем НОД (a, b), равны 1, то a и b — соседние числа Фибоначчи.

1.3.4. Пусть n — нечетное число, обладающее k разными простыми делителями. Покажите, что число решений уравнения

$$x^2 = 1 \pmod n$$

равно 2^k.

1.3.5. Докажите, что элемент $g \in \mathbb{F}_p$ является образующей циклической группы $\mathbb{F}_p^*$ тогда и только тогда, когда $g^{p-1} = 1 \pmod p$ и $g^q \neq 1 \pmod p$ для всех простых делителей q числа $p - 1$.

1.3.6. Дано составное число N и числа a, b, $a^{1/3}b^{1/5} \pmod N$. Вычислите $a^{1/3} \pmod N$ и $b^{1/5} \pmod N$.

1.3.7. Найдите примитивный элемент в каждом из конечных полей $\mathbb{F}_{2^n}$ при $1 \leqslant n \leqslant 8$.

1.3.8. Вычислите символы Якоби $\left(\frac{311}{653}\right)$ и $\left(\frac{666}{777}\right)$, используя лишь карманный калькулятор.

ГЛАВА 2

ЭЛЛИПТИЧЕСКИЕ КРИВЫЕ

Цели главы

- Рассказать, что такое эллиптическая кривая.
- Выяснить, почему эллиптическая кривая представляет интерес для криптографии.
- Показать, как использование проективных координат повышает скорость вычислений.
- Объяснить, как сжатие точек используется для повышения эффективности вычислений.

2.1. Введение

Эта глава посвящена введению в теорию эллиптических кривых над конечными полями. Некоторые из наиболее современных криптографических систем с открытым ключом основаны на использовании эллиптической кривой, в результате чего они обеспечивают высокую эффективность и большую пропускную способность. Настоящую главу можно пропустить при первом чтении, поскольку многое из этой книги можно читать, понимая лишь, что эллиптическая кривая над конечным полем — конечная абелева группа, в которой можно ставить задачу о вычислении дискретных логарифмов.

Проективная плоскость $\mathbb{P}^2(K)$ над полем K определяется как множество троек (X, Y, Z) не равных одновременно нулю элементов X, Y, $Z \in K$, на котором введено отношение эквивалентности:

$$(X, Y, Z) \sim (\lambda X, \lambda Y, \lambda Z) \qquad \text{для любых } \lambda \in K^*.$$

Так, например, две точки $(4, 1, 1)$ и $(5, 3, 3)$ эквивалентны в $\mathbb{P}^2(\mathbb{F}_7)$. Класс эквивалентности троек называется *проективной точкой*.

Эллиптической кривой E называется множество точек проективной плоскости, удовлетворяющих *однородному уравнению Вей-*

ерштрасса

$$E: \quad F(X,Y,Z) = \\ = -X^3 + Y^2Z + a_1XYZ - a_2X^2Z + a_3YZ^2 - a_4XZ^2 - a_6Z^3 = 0,$$

где a_1, a_2, a_3, a_4, $a_6 \in K$. Это уравнение также называют *длинной формой Вейерштрасса*. Кривая должна быть неособой в том смысле, что частные производные

$$\frac{\partial F}{\partial X}, \quad \frac{\partial F}{\partial Y}, \quad \frac{\partial F}{\partial Z}$$

не должны обращаться в нуль одновременно ни в одной ее точке.

Множество K-рациональных точек кривой E, т. е. точек из $\mathbb{P}^2(K)$, удовлетворяющих уравнению кривой, обозначается через $E(K)$. Отметим, что кривая имеет ровно одну точку, чья координата Z равна нулю, а именно $(0,1,0)$. Ее принято называть *бесконечно удаленной точкой* (или просто *точкой на бесконечности*) и обозначать символом $\mathcal{O}$.

Для удобства мы будем часто пользоваться аффинной версией уравнения Вейерштрасса:

$$E: \quad Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, \qquad (2.1)$$

с $a_i \in K$.

K-рациональные точки в аффинном случае — это решения уравнения в K^2 и бесконечно удаленная точка $\mathcal{O}$. Хотя большинство протоколов в криптографии используют эллиптическую кривую в аффинном виде, с точки зрения вычислений бывает удобно перейти к проективным координатам. К счастью, такой переход легко осуществить:

- точка на бесконечности всегда переходит в бесконечно удаленную точку, как при переходе от аффинных координат к проективным, так и наоборот;
- проективная точка (X,Y,Z) кривой, отличная от бесконечно удаленной ($Z \neq 0$), переходит в аффинную точку с координатами $(\frac{X}{Z}, \frac{Y}{Z})$;
- чтобы найти проективные координаты аффинной точки (X,Y), не лежащей на бесконечности, достаточно выбрать произвольное значение $Z \in K^*$ и вычислить $(X \cdot Z, Y \cdot Z, Z)$.

Как мы увидим, иногда будет удобнее пользоваться слегка модифицированной формой проективной плоскости, когда проективные координаты (X,Y,Z) представляют аффинную точку $(X/Z^2, Y/Z^3)$.

Для эллиптической кривой, заданной уравнением (2.1), вводятся следующие константы, которые будут использованы в дальнейших формулах:

$$\begin{aligned} b_2 &= a_1^2 + 4a_2, \\ b_4 &= a_1a_3 + 2a_4, \\ b_6 &= a_3^2 + 4a_6, \\ b_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2, \\ c_4 &= b_2^2 - 24b_4, \\ c_6 &= -b_2^3 + 36b_2b_4 - 216b_6. \end{aligned}$$

Дискриминант кривой E определяется по формуле

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6.$$

Если $\operatorname{char} K \neq 2, 3$, то дискриминант можно вычислить и так:

$$\Delta = \frac{c_4^3 - c_6^2}{1728}.$$

Заметим, что $1728 = 2^6 3^3$, так что деление на это число имеет смысл только в тех полях, чья характеристика отлична от 2 и от 3. Известно, что кривая E неособа тогда и только тогда, когда $\Delta \neq 0$. С этого момента будем предполагать, что $\Delta \neq 0$.

Для неособых кривых вводится j-инвариант

$$j(E) = \frac{c_4^3}{\Delta}.$$

Рассмотрим пример эллиптической кривой, определенной над полем $\mathbb{F}_7$, к которому мы часто будем возвращаться:

$$E: \quad Y^2 = X^3 + X + 3. \tag{2.2}$$

Вычисляя выписанные ранее константы, найдем $\Delta = 3$ и $j(E) = 5$. Инвариант j тесно связан с понятием *изоморфизма эллиптических кривых*. Говорят, что кривая E с координатами X и Y изоморфна над полем K кривой E' с координатами X' и Y' (обе заданы уравнением Вейерштрасса), если найдутся такие константы r, s, $t \in K$ и $u \in K^*$, что при замене переменных

$$X = u^2X' + r, \quad Y = u^3Y' + su^2X' + t$$

кривая E перейдет в кривую E'. Отметим, что изоморфизм кривых определен относительно поля K.

Вернемся к нашему примеру, т. е. кривой E из (2.2) над полем $\mathbb{F}_7$. Сделаем замену переменных с набором констант $[u, r, s, t] =$

$= [2, 3, 4, 5]$, т. е. положим

$$X = 4X' + 3 \quad \text{и} \quad Y = Y' + 2X' + 5.$$

Получим изоморфную кривую, задаваемую уравнением

$$E' : \quad Y'^2 + 4X'Y' + 3Y' = X'^3 + X' + 1.$$

Легко убедиться, что $j(E) = j(E') = 5$.

Изоморфизм эллиптических кривых является отношением эквивалентности. Следующая лемма показывает, что j-инвариант разделяет классы эквивалентности этого отношения над алгебраическим замыканием $\overline{K}$ поля K.

Лемма 2.1. Изоморфные над полем K кривые имеют один и тот же j-инвариант. С другой стороны, любые кривые с совпадающими j-инвариантами изоморфны над алгебраическим замыканием $\overline{K}$.

Однако кривые с одним и тем же j-инвариантом не обязательно изоморфны над основным полем. Например, j-инвариант кривой над полем $\mathbb{F}_7$, заданной уравнением

$$E'' : \quad Y''^2 = X''^3 + 4X'' + 4,$$

равен 5, как и у нашей кривой E (см. (2.2)). Но эти кривые не изоморфны над $\mathbb{F}_7$, поскольку замена переменных, задающая изоморфизм, выглядит как

$$X = 3X'' \quad \text{и} \quad Y = \sqrt{6}\,Y''.$$

Но $\sqrt{6} \notin \mathbb{F}_7$. Итак, E и E'' определены над полем $\mathbb{F}_7$, но не изоморфны над ним. Эти кривые будут изоморфны над любым алгебраическим расширением поля $\mathbb{F}_7$, содержащим элемент $\sqrt{6}$, например, над полем $\mathbb{F}_{7^2} = \mathbb{F}_7[\sqrt{6}]$.

2.2. Групповой закон

Допустим, что $\operatorname{char} K \neq 2, 3$ и рассмотрим замену переменных

$$X = X' - \frac{b_2}{12}, \quad Y = Y' - \frac{a_1}{2}\left(X' - \frac{b_2}{12}\right) - \frac{a_3}{2},$$

переводящую кривую, заданную длинной формой Вейерштрасса (2.1), в изоморфную ей кривую, определяемую *короткой формой Вейерштрасса*

$$E : \quad Y^2 = X^3 + aX + b$$

при некоторых a, $b \in K$. На таких представителях классов изоморфных эллиптических кривых можно наглядно ввести групповой закон *методом хорд и касательных*.

Сложение точек определяется с помощью хорд (рис. 2.1). Пусть P и Q — две точки кривой. Соединим их прямой линией. Она обязательно пересечет кривую в какой-то третьей точке R, поскольку мы пересекаем кубическую кривую прямой. Точка R будет определена над тем же полем, что сама кривая и исходные точки P и Q. Отразим затем точку R относительно горизонтальной оси координат и получим точку, определенную над основным полем. Последняя точка и будет суммой $P + Q$.

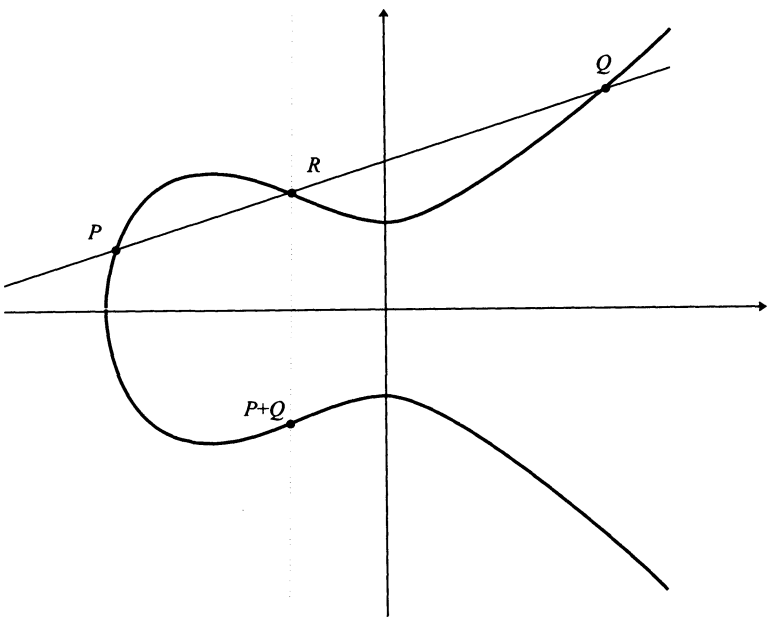


Рис. 2.1. Сложение точек на эллиптической кривой

Касательные служат для удвоения точек (используя хорду, нельзя сложить точку с собой). Пусть P — произвольная точка эллиптической кривой (рис. 2.2). Проведем касательную к кривой в точке P. Она пересечет кривую еще в какой-то одной точке R (кубическая кривая пересекается с прямой по трем точкам с учетом кратности пересечения). Отразив R относительно горизонтальной оси, мы получим точку $[2]P = P + P$. Вертикальная касательная в точке P «пересекает» кривую в бесконечно удаленной точке. В этой ситуации $P + P = \mathcal{O}$ и говорят, что P — *точка порядка* 2.

Можно показать, что метод хорд и касательных наделяет эллиптическую кривую структурой абелевой группы с бесконечно удаленной точкой в качестве единичного элемента, т. е. нуля. Определение операций можно легко перенести на случай общей эллиптической кривой, заданной длинной формой Вейерштрасса (в частности, характеристика поля может быть любой). Необходимо только заменить отражение относительно оси абсцисс на симметрию относительно прямой

$$Y = a_1 X + a_3.$$

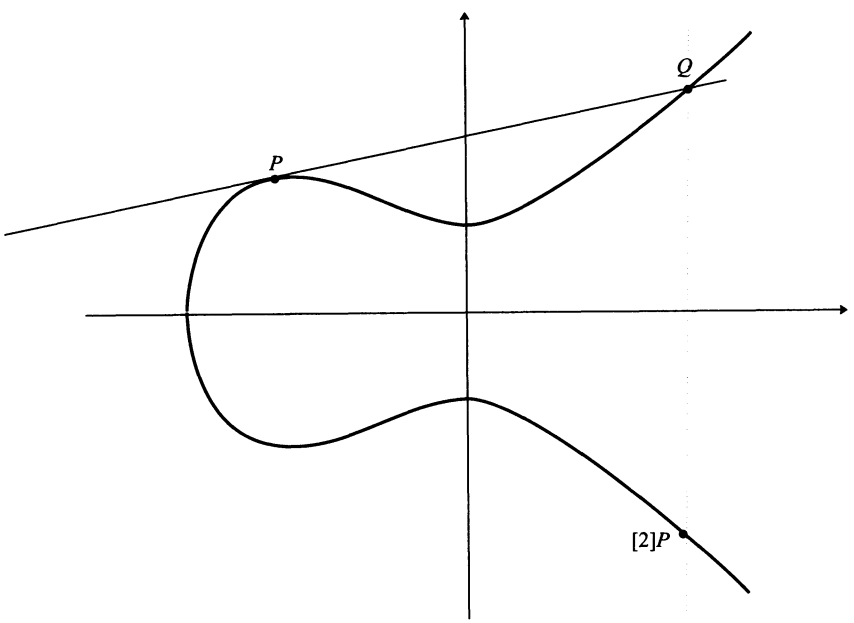


Рис. 2.2. Удвоение точек на эллиптической кривой

В дополнение к сказанному приведем алгебраические формулы, реализующие сложение точек по методу хорд и касательных. Это необходимо сделать, поскольку вычерчивание диаграмм в поле конечной характеристики — дело безнадежное.

Лемма 2.2. Пусть E — эллиптическая кривая, определяемая уравнением

$$E: \quad Y^2 + a_1 XY + a_3 Y = X^3 + a_2 X_2 + a_4 X + a_6,$$

на которой выбраны точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$. Точка $-P_1$ имеет

координаты

$$-P_1(x_1 - y_1 - a_1x_1 - a_3).$$

Введем коэффициенты

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \qquad \mu = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}$$

при $x_1 \neq x_2$ и

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}, \qquad \mu = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3},$$

если $x_1 = x_2$, но $P_2 \neq -P_1$. Если

$$P_3(x_3, y_3) = P_1 + P_2 \neq \mathcal{O},$$

то x_3 и y_3 вычисляются по формулам:

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \quad y_3 = -(\lambda + a_1)x_3 - \mu - a_3.$$

Описанный ранее изоморфизм эллиптических кривых сохраняет структуру группы. Так что на изоморфных кривых указанные формулы определяют структуры изоморфных абелевых групп.

Фиксируем натуральное число m и обозначим через $[m]$ отображение кривой на себя, сопоставляющее каждой точке P ее кратное $[m]P$, т. е.

$$[m] : P \mapsto \underbrace{P + P + \cdots P}_{m}.$$

Это отображение — основа криптографических систем, опирающихся на эллиптическую кривую, поскольку его можно легко вычислить, но крайне сложно обратить, т. е. по данным координатам $P(x, y)$ и $[m]P(x', y')$ найти m очень трудно. Конечно, наше высказывание о сложности обращения предполагает специальный выбор эллиптической кривой и соблюдение нескольких других условий, к чему мы еще вернемся.

Закончим этот параграф иллюстрацией группового закона на эллиптической кривой. Вновь рассмотрим кривую E над полем $\mathbb{F}_7$, заданную уравнением (2.2). Оказывается, на этой кривой всего лишь шесть точек, одна из которых — $\mathcal{O}$, а координаты других представлены списком:

$$(4, 1),\ (6, 6),\ (5, 0),\ (6, 1),\ (4, 6).$$

Результаты сложения несложно получить по формулам леммы 2.2. Сведем их в таблицу 2.1. Найдем координаты образов точки $P(4, 1)$

при отображении $[m]$ для различных m.

$$[2]P(6,6), \qquad [5]P(4,6),$$
$$[3]P(5,0), \qquad [6]P = \mathcal{O}.$$
$$[4]P(6,1),$$

Из вычислений видно, что в нашем примере $E(\mathbb{F}_7)$ — конечная циклическая группа порядка 6, а точка $P(4,1)$ — ее образующая. Эллиптическая кривая над любым конечным полем будет конечной абелевой группой, и, что очень удачно, циклической (или близкой к циклической).

Таблица 2.1. Сложение точек на кривой (2.4) над полем $\mathbb{F}_7$

+	$\mathcal{O}$	(4,1)	(6,6)	(5,0)	(6,1)	(4,6)
$\mathcal{O}$	$\mathcal{O}$	(4,1)	(6,6)	(5,0)	(6,1)	(4,6)
(4,1)	(4,1)	(6,6)	(5,0)	(6,1)	(4,6)	$\mathcal{O}$
(6,6)	(6,6)	(5,0)	(6,1)	(4,6)	$\mathcal{O}$	(4,1)
(5,0)	(5,0)	(6,1)	(4,6)	$\mathcal{O}$	(4,1)	(6,6)
(6,1)	(6,1)	(4,6)	$\mathcal{O}$	(4,1)	(6,6)	(5,0)
(4,6)	(4,6)	$\mathcal{O}$	(4,1)	(6,6)	(5,0)	(6,1)

2.3. Эллиптические кривые над конечными полями

Как мы уже отмечали, количество $\mathbb{F}_q$-рациональных точек эллиптической кривой конечно. Обозначим его символом $\#E(\mathbb{F}_q)$. Ожидаемое число точек кривой близко к $q+1$ и можно положить

$$\#E(\mathbb{F}_q) = q + 1 - t,$$

где «дефект» t называется *следом отображения Фробениуса* в q. В хорошо известной теореме Хассе дана оценка порядка группы $E(\mathbb{F}_q)$.

Теорема 2.3. (Хассе, 1933.) След отображения Фробениуса удовлетворяет неравенству

$$|t| \leqslant 2\sqrt{q}.$$

В примере (2.2) кривая над полем $\mathbb{F}_7$ имеет 6 точек. Так что след отображения Фробениуса здесь равен 2, что, конечно, меньше $2\sqrt{q} = 2\sqrt{7} \approx 5,29$.

На эллиптической кривой E над полем $\mathbb{F}_q$ определено отображение Фробениуса:

$$\varphi : E(\overline{\mathbb{F}}_q) \longrightarrow E(\overline{\mathbb{F}}_q), \qquad \varphi(x,y) = (x^q, y^q), \quad \varphi(\mathcal{O}) = \mathcal{O}.$$

Оно сопоставляет точке кривой E точку на той же кривой, вне зависимости от поля, над которым эта точка определена. Кроме того, отображение Фробениуса сохраняет групповую операцию, т. е.

$$\varphi(P+Q) = \varphi(P) + \varphi(Q).$$

Другими словами, φ — эндоморфизм группы E над алгебраическим замыканием $\overline{\mathbb{F}}_q$, который обычно называют *эндоморфизмом Фробениуса*.

Сед отображения Фробениуса и эндоморфизм Фробениуса связаны уравнением:

$$\varphi^2 - [t]\varphi + [q] = [0],$$

т. е. для произвольной точки $P(x,y)$ кривой выполнено соотношение

$$(x^{q^2}, y^{q^2}) - [t](x^q, y^q) + [q](x,y) = \mathcal{O},$$

в котором сложение и вычитание — суть групповые операции на эллиптической кривой. Есть два частных случая криптографически непригодных эллиптических кривых:

- Кривая $E(\mathbb{F}_q)$ называется *аномальной*, если ее след Фробениуса равен 1, т. е. $\#E(F_q) = q$. Эта кривая особенно неудобна, когда q — простое число.
- Кривая $E(\mathbb{F}_q)$ называется *суперсингулярной*, если характеристика p поля $\mathbb{F}_q$ делит след отображения Фробениуса t. Таких кривых также стараются избегать в криптографии. При $q = p$ суперсингулярная кривая насчитывает $p+1$ точку, поскольку $t = 0$ в этом случае. Если же $q = p^f$, то t у суперсингулярных кривых может принимать значения

  при нечетном f: $t = 0$, $t^2 = 2q$ и $t^2 = 3q$;

  при четном f: $t^2 = 4q$, $t^2 = q$, если $p = 1 \pmod 3$; и $t = 0$, если $p \neq 1 \pmod 4$.

Выбирая кривую для шифрования, нужно стремиться к тому, чтобы число ее точек делилось на достаточно большое простое число. В связи с этим необходимо научиться вычислять порядок группы $E(\mathbb{F}_q)$.

Известно, что порядок произвольной группы $E(\mathbb{F}_q)$ над любым полем вычисляется за полиномиальное время. Это делается с помощью сложного алгоритма, который мы не можем объяснить в этой книге. Достаточно запомнить, что вычисление порядка группы возможно как в теоретическом, так и в практических планах. В следующих главах, рассматривая алгоритмы решения задачи о дискрет-

ных логарифмах, мы увидим, что информация о порядке группы очень существенна для оценки стойкости протокола, основанного на соответствующей кривой.

Одним из достоинств эллиптических кривых является то, что они доставляют большое число возможных групп. Можно менять как основное поле, так и коэффициенты уравнения кривой. Наконец, отыскать эллиптическую кривую с хорошими криптографическими свойствами для создания безопасного протокола, относительно легко.

Как было отмечено ранее, случаи $\operatorname{char} K = 2, 3$ требуют дополнительных усилий. Реализация криптографических систем, основанных на эллиптической кривой, базируется на поле $\mathbb{F}_{2^n}$, чья характеристика равна 2, или на поле $\mathbb{F}_p$ с большим простым числом p. Поэтому в конце текущей главы мы сконцентрируем внимание на полях характеристики 2 и $p > 3$, опустив случай $\operatorname{char} K = 3$. Большинство общих рассуждений с некоторой модификацией переносится на случай характеристики три, что хорошо освещено в специальной литературе.

2.3.1. Кривые над полем характеристики $p > 3$

Пусть основное поле $K = \mathbb{F}_q$ с $q = p^n$, где $p > 3$ — простое число и $n \geqslant 1$. Как уже отмечалось, уравнение кривой над таким полем можно представить в виде короткой формы Вейерштрасса

$$E: \quad Y^2 = X^3 + aX + b. \tag{2.3}$$

Ее дискриминант равен $\Delta = -16(4a^3 + 27b^2)$, а j-инвариант — $j(E) = -1728(4a)^3/\Delta$. Формулы из леммы 2.2, описывающие групповой закон, также упрощаются: $-P_1 = (x_1, -y_1)$ и, если $P_3(x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$, то координаты x_3, y_3 вычисляются так:

$$x_3 = \lambda^2 - x_1 - x_2, \qquad y_3 = (x_1 - x_3)\lambda - y_1,$$

где при $x_1 \neq x_2$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},$$

а при $x_1 = x_2$, $y_1 \neq 0$

$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

2.3.2. Кривые над полем характеристики 2

Здесь мы ограничимся конечными полями $\mathbb{F}_q$ с $q = 2^n$ при $n \geqslant 1$. В этом случае j-инвариант кривой E вычисляется по формуле

$j(E) = a_1^{12}/\Delta$. Условие $j(E) = 0$, т.е. $a_1 = 0$, в характеристике 2 равносильно суперсингулярности кривой E. Мы уже отмечали, что суперсингулярные кривые — очень специфический случай, который не используется в криптографии. Поэтому будем предполагать, что $j(E) \neq 0$.

В этих предположениях представитель любого класса изоморфизма эллиптических кривых над $\mathbb{F}_q$ записывается уравнением

$$E: \quad Y^2 + XY = X^3 + a_2X^2 + a_6, \tag{2.4}$$

где $a_6 \in \mathbb{F}_q^*$ и $a_2 \in \{0, \gamma\}$. Здесь γ — фиксированный элемент поля $\mathbb{F}_q$, удовлетворяющий соотношению: $\mathrm{Tr}_{q|2}(\gamma) = 1$, где $\mathrm{Tr}_{q|2}$ — абсолютный след, вычисляющийся по формуле

$$\mathrm{Tr}_{2^n|2}(\alpha) = \sum_{i=0}^{n-1} \alpha^{2^i}.$$

Когда характеристика поля равна 2, формула для противоположного элемента эллиптической кривой из леммы 2.2 преобразуется к виду $-P_1 = (x_1, y_1 + x_1)$ и, если $P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$, то

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a_2 + x_1 + x_2, \\ y_3 &= (\lambda + 1)x_3 + \mu = (x_1 + x_3)\lambda + x_3 + y_1, \end{aligned}$$

где при $x_1 \neq x_2$

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1}, \quad \mu = \frac{y_1x_2 + y_2x_1}{x_2 + x_1};$$

а если $x_1 = x_2 \neq 0$, то

$$\lambda = \frac{x_1^2 + y_1}{x_1}, \quad \mu = x_1^2.$$

2.4. Проективные координаты

Одна из проблем, возникающих при использовании формул группового закона как при большой, так и при четной характеристике поля, связана с необходимостью деления. Деление в конечном поле считается дорогой операцией, т.к. включает в себя некий вариант расширенного алгоритма Евклида, который хотя и имеет приблизительно ту же сложность, что и умножение, однако обычно не может быть реализован достаточно эффективно.

Во избежание операции деления применяют проективные координаты. При этом уравнение эллиптической кривой записывается

через три координаты (X, Y, Z) вместо двух (X, Y). Однако вместо стандартного варианта уравнения кривой, который был приведен в начале главы, используется уравнение вида

$$E: \quad Y^2 + a_1XYZ + a_2YZ^4 = X^3 + a_2X^2Z^2 + a_4XZ^4 + a_6Z^6.$$

Точка на бесконечности здесь также имеет координаты $(0, 1, 0)$, но переход от проективных координат к аффинным осуществляется по правилу

$$(X, Y, Z) \mapsto (X/Z^2, Y/Z^3).$$

Выбор таких проективных координат обусловлен стремлением сделать арифметические операции более эффективными.

2.4.1. Большая характеристика

Формулы сложения точек эллиптической кривой, заданной уравнением

$$Y^2 = X^3 + aXZ^2 + bZ^6,$$

над полем большой характеристики выглядят теперь как

$$(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, Z_2),$$

где тройка координат (X_3, Y_3, Z_3) вычисляется последовательно по правилу:

$$\begin{aligned}
\lambda_1 &= X_1Z_2^2, & \lambda_2 &= X_2Z_1^2,\\
\lambda_3 &= \lambda_1 - \lambda_2, & \lambda_4 &= Y_1Z_2^3,\\
\lambda_5 &= Y_2Z_1^3, & \lambda_6 &= \lambda_4 - \lambda_5,\\
\lambda_7 &= \lambda_1 + \lambda_2, & \lambda_8 &= \lambda_4 + \lambda_5,\\
Z_3 &= Z_1Z_2\lambda_3, & X_3 &= \lambda_6^2 - \lambda_7\lambda_3^2,\\
\lambda_9 &= \lambda_7\lambda_3^2 - 2X_3, & Y_3 &= (\lambda_9\lambda_6 - \lambda_8\lambda_3^3)/2.
\end{aligned}$$

Обратите внимание, что здесь нет ни одной операции деления, кроме деления на 2, которая легко заменяется умножением на заранее вычисленное число $2^{-1} \pmod p$.

Удвоение точек $(X_3, Y_3, Z_3) = [2](X_1, Y_1, Z_1)$ упрощается с помощью формул

$$\begin{aligned}
\lambda_1 &= 3X_1^2 + aZ_1^4, & Z_3 &= 2Y_1Z_1,\\
\lambda_2 &= 4X_1Y_1^2, & X_3 &= \lambda_1^2 - 2\lambda_2,\\
\lambda_3 &= 8Y_1^4, & Y_3 &= \lambda_1(\lambda_2 - X_3) - \lambda_3.
\end{aligned}$$

2.4.2. Четная характеристика

Уравнение эллиптической кривой над полем четной характеристики записывается в виде

$$Y^2 + XYZ = X^3 + a_2X^2Z^4 + a_6Z^6.$$

Сложение точек

$$(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, Z_2)$$

в этом случае осуществляется по рецепту:

$$\begin{aligned}
&\lambda_1 = X_1Z_2^2, && \lambda_2 = X_2Z_1^2,\\
&\lambda_3 = \lambda_1 + \lambda_2, && \lambda_4 = Y_1Z_2^3,\\
&\lambda_5 = Y_2Z_1^3, && \lambda_6 = \lambda_4 + \lambda_5,\\
&\lambda_7 = Z_1\lambda_3, && \lambda_8 = \lambda_6X_2 + \lambda_7Y_2,\\
&Z_3 = \lambda_7Z_2, && \lambda_9 = \lambda_6 + Z_3,\\
&X_3 = a_2Z_3^2 + \lambda_6\lambda_9 + \lambda_3^4, && Y_3 = \lambda_9X_3 + \lambda_8\lambda_7^2.
\end{aligned}$$

И, наконец, координаты удвоенной точки определяются по правилу:

$$\begin{aligned}
&Z_3 = X_1Z_1^2, && X_3 = (X_1 + a_6Z_1^2)^4,\\
&\lambda = Z_3 + X_1^2 + Y_1Z_1, && Y_3 = X_1^4Z_3 + \lambda X_3.
\end{aligned}$$

Итак, в обоих интересных случаях — при большой и четной характеристиках поля — мы исключили дорогостоящую операцию деления.

2.5. Сжатие точек

Во многих криптографических протоколах возникает необходимость хранить в памяти или передавать по сети отдельные точки эллиптической кривой. В аффинных координатах это можно сделать с помощью двух элементов поля: координат x и y. Однако экономнее применять так называемую технику сжатия точек.

Метод сжатия точек работает благодаря тому, что уравнение кривой в аффинных координатах при фиксированном значении x превращается в квадратное уравнение относительно координаты y. Поэтому каждому возможному значению координаты x точки кривой соответствует не более двух значений координаты y. Значит, вместо двух координат для идентификации точки кривой можно хранить в памяти компьютера только координату x и еще некий

двоичный параметр[1] b, сообщающий о том, какое именно значение координаты y нужно брать. Остается только решить, как вычислять параметр b и как по нему и данному x восстановить y-координату нужной точки.

2.5.1. Случай большой характеристики поля

Заметим, что если $p > 2$, то квадратные корни $\pm\beta$ из элемента $\alpha \in \mathbb{F}_p$ представляются натуральными числами разной четности из промежутка $1, \ldots, p-1$, поскольку

$$-\beta = p - \beta \pmod{p}.$$

Таким образом, в качестве параметра b можно выбрать четность y-координаты соответствующей точки. Покажем теперь, как восстановить полную информацию о координатах точки по паре (x, b). Сначала вычисляется[2]

$$\beta = \sqrt{x^3 + ax + b} \pmod{p},$$

а затем переменной y присваивают значение β, если четность β совпадает с четностью b, и $p - \beta$, когда четности разные. Если же оказалось, что $\beta = 0$, то, не обращая внимания на параметр b, можно положить $y = 0$.

В качестве примера рассмотрим кривую

$$E: \quad Y^2 = X^3 + X + 3$$

над полем $\mathbb{F}_7$. Для представления каждой из ее точек $(4, 1)$ и $(4, 6)$ в двоичной системе счисления нам потребуется шесть разрядов:

$$(0b\,100, 0b\,001) \quad \text{и} \quad (0b\,100, 0b\,110),$$

в то время как при использовании метода сжатия точек всего четыре:

$$(0b\,100, 0b\,1) \quad \text{и} \quad (0b\,100, 0b\,0).$$

В реальных криптографических протоколах получается более существенная экономия компьютерной памяти. Рассмотрим, например, ту же кривую, но над полем $\mathbb{F}_p$ с

$$p = 1\,125\,899\,906\,842\,679 = 2^{50} + 55.$$

[1] Не путать с коэффициентом b в уравнении (2.3). — *Прим. перев.*

[2] Напомним, что кривая в случае большой характеристики поля определяется уравнением (2.3). — *Прим. перев.*

На ней есть точка с координатами

$$(1\,125\,899\,906\,842\,675, 245\,132\,605\,757\,739),$$

для записи которой нам потребовалось 102 разряда. При сжатии информации эту точку можно представить в виде

$$(1\,125\,899\,906\,842\,675, 1),$$

где использовано только 52 разряда.

2.5.2. Четная характеристика

В случае четной характеристики необходимо проявить больше изобретательности. Предположим, что нам дана точка $P(x, y)$ на эллиптической кривой

$$Y^2 + XY = X^3 + a_2X^2 + a_6. \tag{2.5}$$

Если $y = 0$, то можно положить $b = 0$. В противном случае вычисляют $z = y/x$ и присваивают переменной b самый младший двоичный разряд числа z. Для восстановления y по данной паре (x, b) в случае $x \neq 0$ вычисляют

$$\alpha = x + a_2 + \frac{a_6}{x^2}$$

и обозначают через β одно из решений уравнения

$$z^2 + z = \alpha.$$

Если наименьший двоичный разряд числа β совпадает с b, то $y = x\beta$. В противоположной ситуации $y = x(\beta + 1)$.

Для объяснения того, почему этот метод правильно восстанавливает y-координату точки, напомним, что координаты (x, y) точки P — решение уравнения (2.5). Поэтому пары $(x, y/x)$ и $(x, 1 + y/x)$ удовлетворяют соотношению

$$Z^2 + Z = X + a_2 + \frac{a_6}{X^2}.$$

Краткое содержание главы

- Эллиптическая кривая над конечным полем — еще один пример конечной абелевой группы. Существует очень много таких групп, поскольку мы вольны в выборе как коэффициентов кривой, так и поля, над которым она определена.
- В криптографии необходимо уметь вычислять порядок группы. Хотя это делается с помощью сложного алгоритма, о ко-

тором здесь не рассказывалось, вычисления занимают полиномиальное время.
- Как правило, суперсингулярные и аномальные кривые не используются в криптографических приложениях.
- Скорость алгоритмов, реализующих групповой закон на кривой, можно увеличить, если использовать проективные координаты.
- Для увеличения пропускной способности и экономии памяти координаты точки эллиптической кривой (x, y) успешно сжимаются до x и отдельного бита b. Полная информация восстанавливается также довольно эффективно.

Дополнительная литература

Тем, кто хочет аккуратно выучить общую теорию эллиптических кривых, можно порекомендовать учебник Сильвермана (предназначенный студентам математических факультетов). Интересующиеся лишь криптографическими приложениями эллиптических кривых и соответствующими алгоритмами могут просмотреть книгу Блэка, Сироуси и Смарта.

I. F. Blake, G. Seroussi and N. P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

J. H. Silverman. *The Arithmetic of Elliptic Curves*. Springer-Verlag, 1985.

Контрольные вопросы

2.1.1. Когда эллиптическая кривая особа?

2.1.2. Что означает совпадение j-инвариантов у двух эллиптических кривых?

2.1.3. Опишите метод хорд и касательных и объясните, как с его помощью реализуется групповой закон на эллиптической кривой.

2.1.4. Что такое след отображения Фробениуса, и какому неравенству он удовлетворяет?

2.1.5. В каком случае эллиптическую кривую называют аномальной?

2.1.6. Для чего нужны проективные координаты?

2.1.7. Объясните, как сжатие точек экономит память и увеличивает скорость при передаче точек эллиптической кривой.

Лабораторные работы

2.2.1. Создайте электронную библиотеку программ, складывающих и удваивающих точки эллиптической кривой по модулю p.

2.2.2. Расширьте Вашу библиотеку, включив в нее программы работы с проективными координатами, сжимающие и восстанавливающие точки.

Упражнения

2.3.1. Выведите формулы леммы 2.2 из геометрического описания группового закона.

2.3.2. Пусть E — эллиптическая кривая, заданная уравнением

$$Y^2 = X^3 + aX + b$$

над полем характеристики $p > 3$. Опишите возможные точки порядка три[3] на кривой E.

2.3.3. Покажите, что кривые

$$E: \quad Y^2 = X^3 + aX + b \quad \text{и} \quad E': \quad Y^2 = X^3 + ad^2X + bd^3,$$

определенные над полем K, изоморфны над алгебраическим замыканием $\overline{K}$. Когда они будут изоморфны над исходным полем K?

[3]То есть точки P, удовлетворяющие условию $[3]P = \mathcal{O}$. — *Прим. перев.*

ЧАСТЬ II

СИММЕТРИЧНОЕ ШИФРОВАНИЕ

Кодирование в большинстве случаев опирается на блочные и групповые шифры, которые служат типичными примерами симметричных алгоритмов шифрования. Кроме того, исторические (т. е. использовавшиеся до 1960 г.) шифры, симметричные по своей природе, служат основой многих современных криптосистем.

Главный недостаток симметричного шифрования связан с проблемой распределения секретных ключей между пользователями. В следующих главах будет рассказано о теории и практике симметричных криптосистем, но сначала мы познакомимся с докомпьютерными шифрами.

ГЛАВА 3

ИСТОРИЧЕСКИЕ ШИФРЫ

Цели главы

- Рассказать о некоторых докомпьютерных шифрах, таких, как шифр Цезаря, шифр замены и машине «Энигма».
- Показать ненадежность этих шифров ввиду наследования шифротекстом статистики подлежащего языка.
- Ввести понятия замены и перестановки как основных компонентов шифра.
- Описать несколько атак на шифры, например, атаку с выбором открытого текста.

3.1. Введение

Алгоритм шифрования (или шифр) — это перевод открытого текста в текст зашифрованный (или шифротекст, шифрограмму, криптограмму) с помощью секретного ключа. Этот процесс называют шифрованием. Мы будем писать

$$C = E_k(m),$$

где m — *открытый текст*, E — *шифрующая* функция, k — *секретный ключ* и C — *шифротекст*.

Обратный процесс называют *расшифрованием* и пишут

$$m = D_k(C).$$

Заметим, что алгоритмы шифрования и расшифрования E и D открыты, и секретность исходного текста m в данном шифротексте C зависит от секретности ключа k. Обе части этого процесса используют один и тот же ключ, в связи с чем такие алгоритмы принято называть *симметричными криптосистемами*, или *криптосистемами с секретным ключом*. Существуют алгоритмы шифрования, зависящие от двух разных ключей. Первый из них открыт и нужен для шифрования, в то время как второй — секретный и употребляется

при восстановлении текста из шифровки. Эти последние криптосистемы называются *асимметричными*, или *криптосистемами с открытым ключом*. Мы их будем изучать в следующих главах.

Стороны, обменивающиеся шифрованной информацией, обычно обозначаются A и B. Довольно часто, однако, употребляют более дружественные имена: Алиса и Боб. Но не следует думать, что стороны, участвующие в процессе, обязательно люди. Используя эти имена, мы вполне можем описывать обмен секретной информацией между двумя автономными механизмами. Подслушивающей стороне, «плохой девочке», которая взламывает шифротекст, обычно дают имя Ева.

В этой главе мы познакомимся с несколькими ранними шифрами, применявшимися для защиты информации в докомпьютерную эпоху. Мы покажем, что эти шифры легко взламываются с помощью статистических исследований языка, на котором они написаны (такой язык принято называть подлежащим), в нашем случае английского[1]. В главе 4 мы обсудим связь между стойкостью шифра и знанием статистического распределения букв подлежащего открытого текста.

Таблица 3.1. Среднестатистические частоты употребления английских букв

Буква	Процентное содержание	Буква	Процентное содержание
a	8,2	n	6,7
b	1,5	o	7,5
c	2,8	p	1,9
d	4,2	q	0,1
e	12,7	r	6,0
f	2,2	s	6,3
g	2,0	t	9,0
h	6,1	u	2,8
i	7,0	v	1,0
j	0,1	w	2,4
k	0,8	x	0,1
l	4,0	y	2,0
m	2,4	z	0,1

Распределение (встречаемость) букв в английском среднестатистическом тексте представлено в табл. 3.1. Соответствующая гисто-

[1]Редакцией было принято решение не переводить примеры дешифрования на русский язык, поскольку в реальной жизни перехваченная шифровка может быть написана на любом языке. — *Прим. перев.*

грамма изображена на рис. 3.1. Как видно из этих данных, наиболее часто встречающиеся буквы — это «e» и «t».

При взломе шифра полезно знать статистическую информацию и второго порядка, а именно, частоту встречаемости групп из двух и трех букв, называемых биграммами и триграммами. Наиболее часто встречающиеся биграммы представлены в табл. 3.2, а самые популярные триграммы выписаны в строку в порядке убывания частоты их использования:

the, ing, and, her, ere, ent, tha, nth, was, eth, for.

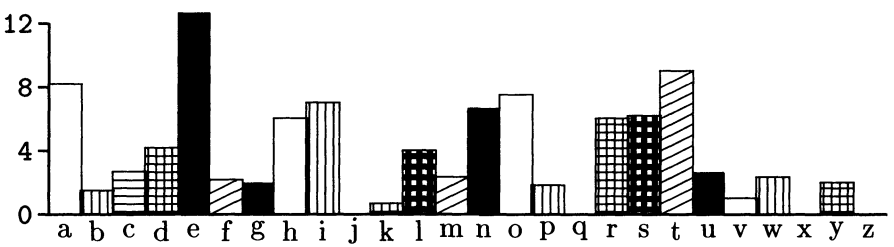


Рис. 3.1. Относительная встречаемость английских букв

Вооруженные такой статистической информацией, мы готовы сейчас исследовать и взломать несколько классических шифров.

Таблица 3.2. Частота встречаемости английских биграмм

Биграмма	Процентное содержание	Биграмма	Процентное содержание
th	3,15	he	2,51
an	1,72	in	1,69
er	1.54	re	1,48
es	1.45	on	1.45
ea	1,31	ti	1,28
at	1,24	st	1,21
en	1,20	nd	1,18

3.2. Шифр сдвига

Коллекцию классических криптосистем открывает один из самых первых известных типов шифра, называемый шифром сдвига. Процесс шифрования заключается в замене каждой буквы на другую, отстоящую от исходной на определенное число позиций в алфавите в зависимости от значения ключа. Так, например, если ключ равен 3, то буква «a» исходного текста в шифровке изображается «D», вместо буквы «b» появится «E» и т. д. Слово «hello» будет представлено

шифровкой «KHOOR». Когда этот шифр употребляется с ключом, равным трем, его часто называют шифром Цезаря, хотя во многих книгах это название относится к шифру сдвига с любым ключом. Строго говоря, такое обобщение не совсем допустимо, поскольку Юлий Цезарь пользовался шифром сдвига именно с ключом 3.

Существует более математизированное описание шифра сдвига, которым мы будем руководствоваться при дальнейшем обсуждении. Сначала нужно пронумеровать все буквы алфавита, начиная с 0, т. е. букве «a» присваивается номер 0, «b» — 1, и т. д. до буквы «z» под номером 25. Затем мы переписываем исходный текст, заменяя каждую букву соответствующим номером, и получаем последовательность чисел. Шифротекст возникает после того, как к каждому числу в этой последовательности прибавляется значение ключа k по модулю 26, k, очевидно, — целое число между 0 и 26. Таким образом, шифр сдвига можно интерпретировать как поточный шифр с потоком ключей в виде повторяющейся последовательности

$$k,\ k,\ k,\ k,\ k,\ k,\ \ldots$$

Этот поток, естественно, далек от случайного, вследствие чего криптостойкость шифра сдвига крайне низка. Наивный путь атаки на шифр сдвига состоит в простом переборе возможных значений ключа до тех пор, пока не получится осмысленный текст. Поскольку существует ровно 25 вариантов (ключ 0 не меняет текста), то для их перебора потребуется не очень много времени, особенно в случае короткой шифрограммы.

Покажем сейчас, как можно взломать шифр сдвига, опираясь на статистику подлежащего языка. В случае шифра сдвига такой способ не является острой необходимостью. Однако позже мы познакомимся с шифрами, составленными из нескольких шифров сдвига, которые применяются поочередно, и тут уж без статистического подхода не обойтись. Кроме того, приложение этого метода к шифру сдвига иллюстрирует проявление статистики исходного открытого текста в соответствующем шифротексте.

Рассмотрим пример шифрограммы.

GB OR, BE ABG GB OR: GUNG VF GUR DHRFGVBA:
JURGURE 'GVF ABOYRE VA GUR ZVAQ GB FHSSRE
GUR FYVATF NAQ NEEBJF BS BHGENTRBHF SBEGHAR,
BE GB GNXR NEZF NTNVAFG N FRN BS GEBHOYRF,
NAQ OL BCCBFVAT RAQ GURZ? GB OVR: GB FYRRC;
AB ZBER; NAQ OL N FYRRC GB FNL JR RAQ
GUR URNEG-NPUR NAQ GUR GUBHFNAQ ANGHENY FUBPXF

GUNG SYRFU VF URVE GB, 'GVF N PBAFHZZNGVBA
QRIBHGYL GB OR JVFU'Q, GB QVR, GB FYRRC;
GB FYRRC: CREPUNAPR GB QERNZ: NL, GURER'F GUR EHO;
SBE VA GUNG FYRRC BS QRNGU JUNG QERNZF ZNL PRZR
JURA JR UNIR FUHSSYRQ BSS GUVF ZBEGNY PBVY,
ZHFG TVIR HF CNHFR: GURER'F GUR ERFCRPG
GUNG ZNXRF PNYNZVGL BS FB YBAT YVSR;

Один из приемов взлома этого образца шифротекста основывается на том, что шифровка все еще сохраняет относительные длины слов исходного текста. Например, «N» появляется в нем как однобуквенное слово. Поскольку в английском языке таковыми словами могут быть лишь «a» и «i», легко предположить, что ключ равен либо 13 (т. к. «N» — тринадцатая буква в алфавите после «A»), либо 5 («N» — пятая буква после «I»). Отсюда мораль — пробелы между словами в исходном тексте перед его шифрованием с помощью шифра сдвига следует убирать. Но даже если игнорировать информацию о длине слов, мы можем без особого труда вскрыть шифр сдвига, применив *частотный анализ.*

Вычислим частоты появления букв в шифротексте и сравним их со среднестатистическими из табл. 3.1. Полученная информация представлена на двух гистограммах (рис. 3.2), расположенных друг над другом. Посмотрев на них, легко убедиться, что гистограмма (б), отражающая статистику букв в шифротексте, очень похожа на сдвиг гистограммы (а) со среднестатистической информацией. Заметим, что мы не вычисляли частоты для вычерчивания первой диаграммы по исходному открытому тексту, поскольку он не был нам известен. Мы просто воспользовались легко доступной статистической информацией о подлежащем языке.

Сравнивая гистограммы, можно предположить, насколько вторая сдвинута относительно первой. Наиболее употребляемая буква в английском тексте — это «e». Отмечая большие столбцы во второй гистограмме, замечаем, что буква «e» исходного текста может быть заменена на «G», «N», «R» или «B», т. е. ключом может служить одно из чисел:

$$2,\ 9,\ 13 \quad \text{или} \quad 23.$$

Аналогично, исследуя кандидаттуры на букву «a», получаем, что возможный ключ равен одному из

$$1,\ 6,\ 13 \quad \text{или} \quad 17.$$

Среди двух серий гипотетических ключей есть только одно совпа-

дающее значение, а именно 13. Поэтому естественно предположить, что именно 13 и является истинным ключом. Приняв это за рабочую гипотезу, сделаем попытку расшифровать сообщение и обнаружим осмысленный текст.

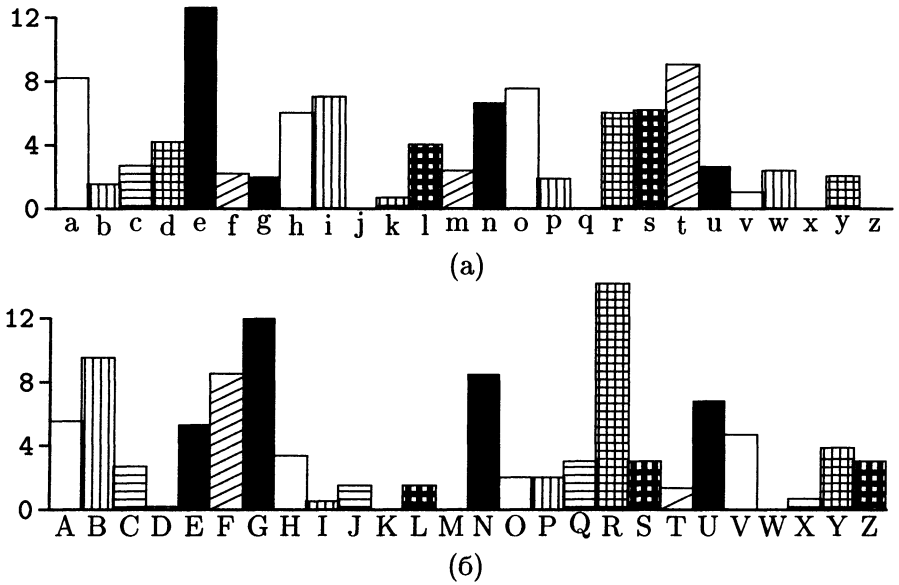


Рис. 3.2. Сравнение частот появления букв в шифротексте со среднестатистической

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep: perchance to dream: ay, there's the rub;
For in that sleep of death what dreams may come
When we have shuffled off this mortal coil,
Must give us pause: there's the respect
That makes calamity of so long life;

Вы, конечно, узнали знаменитый монолог Гамлета из трагедии Вильяма Шекспира.

3.3. Шифр замены

Основной недостаток шифра сдвига заключается в том, что существует слишком мало возможных ключей, всего 25. В целях устранения указанного недостатка был изобретен *шифр замены*. Чтобы описать ключ такого шифра, сначала выписывается алфавит, а непосредственно под ним — тот же алфавит, но с переставленными буквами[1]. Это дает нам правило, по которому буквы открытого текста замещаются символами шифровки. Например,

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

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

Шифрование состоит в замене каждой буквы в открытом тексте на соответствующую ей нижнюю букву. Чтобы расшифровать шифротекст, нужно каждую его букву найти в нижней строке таблицы и заменить ее соответствующей верхней. Таким образом, криптограмма слова «hello» будет выглядеть как ESVVJ, если пользоваться приведенным соответствием.

Количество всех возможных ключей такого шифра совпадает с числом всевозможных перестановок 26 элементов, т. е. порядком симметрической группы S_{26}, равным

$$26! \approx 4,03 \cdot 10^{26} \approx 2^{88}.$$

Поэтому, перебирая все возможные ключи с помощью самого современного и быстрого компьютера, мы потратим столько времени, что задача о дешифровании конкретного сообщения перестанет быть актуальной. Тем не менее, мы можем взломать шифр замены, опираясь на статистику подлежащего языка, аналогично тому, как мы вскрыли шифр сдвига.

Если шифр сдвига можно считать поточным шифром, когда шифротекст получается комбинацией открытого текста с потоком ключей, шифр замены похож на более современный блочный шифр, блок в котором состоит из одной английской буквы. Блок шифротекста получается из блока открытого текста в результате применения некоторого ключа (предположительно, простого), зависящего от используемого алгоритма.

Шифры замены имеют богатую и интересную историю, и как правило именно они фигурируют во всевозможных детективных историях. Один из таких шифров описан в рассказе Артура Конан Дойля

[1] Или любой другой набор различных знаков, например, азбука Морзе. Однако в этой книге автор ограничивается перестановками стандартного алфавита. — *Прим. перев.*

«Пляшущие человечки». Интрига детектива замешана на шифре замены, в котором буквы замещались схематично изображенными человечками в различных положениях. Метод, которым Холмс и Ватсон взломали шифр, и есть тот действенный способ, который мы и возьмем на вооружение, проводя атаку на шифротекст, приведенный ниже.

Мы разберем в деталях пример атаки на криптограмму, которая предварительно была упрощена по сравнению с оригинальной: в ней оставлены промежутки между словами подлежащего открытого текста. Сделано это в целях более наглядной демонстрации метода. В какой-то момент мы воспользуемся этой информацией, хотя стоит признаться, что это сильно облегчит нам задачу.

Рассмотрим шифротекст.

XSO MJIWXVL JODIVA STW VAO VY OZJVCO'W LTJDOWX KVAKOAXJTXIVAW VY SIDS XOKSAVLVDQ IAGZWXJQ. KVUCZXOJW, KVUUZAIKTXIVAW TAG UIKJVOLOKXJVAIKW TJO HOLL JOCJOWOAXOG, TLVADWIGO GIDIXTL UOGIT, KVUCZXOJ DTUOW TAG OLOKXJVAIK KVUUOJKO. TW HOLL TW SVWXIAD UTAQ JOWOTJKS TAG CJVGZKX GONOLVCUOAX KOAXJOW VY UTPVJ DLVMTL KVUCTAIOW, XSO JODIVA STW T JTCIGLQ DJVHIAD AZUMOJ VY IAAVNTXINO AOH KVUCTAIOW. XSO KVUCZXOJ WKIOAKO GOCTJXUOAX STW KLVWO JOLTXIVAWSICW HIXS UTAQ VY XSOWO VJDTAIWTXIVAW NIT KVLLTMVJTXINO CJVPOKXW, WXTYY WOKVAGUOAXW TAG NIWIXIAD IAGZWXJITL WXTYY. IX STW JOKOAXLQ IAXJVJZKOG WONOJTL UOKSTAIWUW YVJ GONOLVCIAD TAG WZCCVJXIAD OAXJOCJOAOZJITL WXZGOAXW TAG WXTYY, TAG TIUW XV CLTQ T WIDAIYIKTAX JVLO IA XSO GONOLVCUOAX VY SIDS-XOKSAVLVDQ IAGZWXJQ IA XSO JODIVA.

XSO GOCTJXUOAX STW T LTJDO CJVDJTUUO VY JOWOTJKS WZCCVJXOG MQ IAGZWXJQ, XSO OZJVCOTA ZAIVA, TAG ZE DVNOJAUOAX JOWOTJKS OWXTMLIWSUOAXW TAG CZMLIK KVJCVJTXIVAW. T EOQ OLOUOAX VY XSIW IW XSO WXJVAD LIAEW XSTX XSO GOCTJXUOAX STW HIXS XSO KVUCZXOJ, KVUUZAIKTXIVAW, UIKJVOLOKXJVAIKW TAG UOGIT IAGZWXJIOW IA XSO MJIWXVL JODIVA. XSO TKTGOUIK JOWOTJKS CJVDJTUUO IW VJDTAIWOG IAXV WONOA DJVZCW, LTADZTDOW TAG TJKSIXOKXZJO, GIDIXTL UOGIT, UVMILO TAG HOTJTMLO KVUCZXIAD, UTKSIAO LOTJAIAD, RZTAXZU KVUCZXIAD, WQWXOU NOJIYIKTXIVA, TAG KJQCXVDJTCSQ TAG IAYVJUTXIVA WOKZJIXQ.

Вычислим частоту встречаемости отдельных букв в этом шифротексте (см. табл. 3.3).

Таблица 3.3. Частоты встречаемости букв в шифротексте примера

Буква	Частота	Буква	Частота	Буква	Частота
A	8,6995	B	0,0000	C	3,0493
D	3,1390	E	0,2690	F	0,0000
G	3,6771	H	0,6278	I	7,8923
J	7,0852	K	4,6636	L	3,5874
M	0,8968	N	1,0762	O	11,479
P	0,1793	Q	1,3452	R	0,0896
S	3,5874	T	8,0717	U	4,1255
V	7,2645	W	6,6367	X	8,0717
Y	1,6143	Z	2,7802		

Кроме того, наиболее употребительные биграммы в шифровке — это

$$\text{TA, AX, IA, VA, WX, XS, AG, OA, JO, JV,}$$

а

$$\text{OAX, TAG, IVA, XSO, KVU, TXI, UOA, AXS —}$$

чаще всего встречающиеся триграммы.

Поскольку буква «O» в нашем образце имеет самую высокую частоту, а именно 11,479, можно предположить, что она соответствует букве «e» открытого текста. Посмотрим, что это может означать для наиболее общих триграмм шифротекста.

- Триграмма OAX шифротекста соответствует «e**» исходного сообщения.
- Триграмма XSO шифротекста соответствует «**e» исходного сообщения.

Вспомним теперь часто употребляемые триграммы в английском языке (стр. 73) и выберем из них те, которые начинаются или оканчиваются на букву «e»: *ent*, *eth* и *the*. Заметим, что первая из популярнейших триграмм шифротекста оканчивается буквой «X», а вторая с нее начинается. Аналогичным свойством обладают выбранные триграммы открытого текста: *ent* и *the*. У них есть общая буква «*t*». В связи с этим можно с большой долей вероятности заключить, что имеет место соответствие

$$\text{X} = \text{t}, \quad \text{S} = \text{h}, \quad \text{A} = \text{n}.$$

Даже после столь небольшого анализа мы намного облегчили понимание открытого текста, скрытого в шифровке. Ограничившись

двумя первыми предложениями, произведем в них замены найденных соответствий, считая, что нашли их правильно.

the MJIWtVL JeDIVn hTW Vne VY eZJVCe'W LTJDeWt
KVnKentJTtIV nW VY hIDh teKhnVLVDQ InGZWtJQ.
KVUCZteJW, KVUUZnIKTtIVnW TnG UIKJVeLeKtJVnIKW
TJe HeLL JeCJeWenteG, TLVnDWIGe GIDItTL UeGIT,
KVUCZteJ DTUeW TnG eLeKtJVnIK KVUUeJKe.

Напомним, что такое продвижение в дешифровании произошло после замен:

$$\mathrm{O} = \mathrm{e}, \quad \mathrm{X} = \mathrm{t}, \quad \mathrm{S} = \mathrm{h}, \quad \mathrm{A} = \mathrm{n}.$$

Теперь мы сжульничаем и воспользуемся тем, что в криптограмме оставлены промежутки между словами. Поскольку буква «T» появляется в шифровке как отдельное слово, она может замещать лишь одну из двух букв открытого текста: «i» или «a». Частота буквы «T» в шифротексте — 8,0717, а среднестатистические частоты букв «i» и «a» равны, соответственно, 7,0 и 8,2 (см. табл. 3.1). Следовательно, скорее всего

$$\mathrm{T} = \mathrm{a}.$$

Мы уже рассмотрели самую встречаемую триграмму в шифровке, так что перенесем наше внимание к следующей по популярности триграмме. Таковой является триграмма TAG. Произведя известные замены, увидим, что она означает триграмму *an** открытого текста. Отсюда вполне обоснованно можно сделать вывод: G = d, поскольку триграмма *and* — одно из наиболее употребительных буквосочетаний английского языка.

При всех сделанных предположениях о соответствии букв частично дешифрованный кусок шифровки имеет вид:

the MJIWtVL JeDIVn haW Vne VY eZJVCe'W LaJDeWt
KVnKentJatIV nW VY hIDh teKhnVLVDQ IndZWtJQ.
KVUCZteJW, KVUUZnIKatIVnW and UIKJVeLeKtJVnIKW
aJe HeLL JeCJeWented, aLVnDWIde dIDItaL UedIa,
KVUCZteJ, DaUeW and eLeKtJVnIK KVUUeJKe.

Такой результат получился после шести замен:

$$\mathrm{O} = \mathrm{e}, \quad \mathrm{X} = \mathrm{t}, \quad \mathrm{S} = \mathrm{h}, \quad \mathrm{A} = \mathrm{n}, \quad \mathrm{T} = \mathrm{a}, \quad \mathrm{G} = \mathrm{d}.$$

На этом этапе исследуем двубуквенные слова, попадающиеся в криптограмме.

IX. Это слово, как мы знаем, означает *t. Значит, буква шифра «I» может замещать либо «a», либо «i», т. к. только два двубуквенных слова английского языка оканчиваются на «t»: «at» и «it». Однако мы уже убедились, что буква «a» открытого текста замещается буквой «T», так что остается одна возможность: I = i.

XV соответствует сочетанию «t*» открытого текста, откуда V = o.

VY можно заменить на «o*». Поэтому буква «Y» шифровки может замещать лишь «f», «n» или «r». Но мы уже знаем букву шифротекста, подменяющую собой «n», и у нас остается только две возможности для выбора. Частота встречаемости символа «Y» в криптограмме — 1,6, в то время как вероятность встретить букву «f» в английском тексте равна 2,2, а букву «r»— 6,0. Так что возможно, имеет место соответствие Y = f.

IW должно означать «i*». Таким образом, «W» замещает одну из четырех букв: «f», «n», «s» или «t». Так как пары для символов «f», «n» и «t» нам известны, то W = s.

Итак, после вычисленных замен:

$$O = e, \quad X = t, \quad S = h, \quad A = n, \quad T = a,$$
$$G = d, \quad I = i, \quad V = o, \quad Y = f, \quad W = s$$

первые два предложения шифротекста выглядят так:

the MJistoL JeDion has one of eZJoCe's LaJDest
KonKentJations of hiDh teKhnoLoDQ indZstJQ.
KoUCZteJs, KoUUZniKations and UiKJoeLeKtJoniKs
aJe HeLL JeCJesented, aLonDside diDitaL Uedia,
KoUCZteJ DaUes and eLeKtJoniK KoUUeJKe.

Даже с половиной определенных букв теперь не очень сложно понять подлежащий открытый текст, взятый с веб-сайта факультета вычислительной математики Бристольского университета. Мы оставляем читателю выявить все оставшиеся буквы и восстановить текст полностью.

3.4. Шифр Виженера

Основной недостаток шифров сдвига и замены заключается в том, что каждая буква открытого текста при шифровании заменяется раз и навсегда фиксированным символом. Поэтому при взломе шифра эффективно работает статистика подлежащего языка. Напри-

мер, не составляет труда определить, за каким знаком в шифровке скрывается буква «E». С начала XIX века разработчики шифров пытались преодолеть такую связь между открытым текстом и его шифрованным вариантом.

Шифр замены, описанный в предыдущем параграфе, относится к так называемым *моноалфавитным* шифрам замены, в которых используется только один упорядоченный набор символов, подменяющий собой стандартный алфавит. Один из путей решения указанной проблемы состоит в том, чтобы брать несколько наборов символов вместо стандартного алфавита и шифровать буквы открытого текста, выбирая соответствующие знаки из разных наборов в определенной последовательности. Шифры такого типа носят название *полиалфавитных* шифров замены.

Например, можно рассмотреть такое соответствие:

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

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

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

в котором первая строка — английский алфавит, а вторая и третья — первый и второй алфавиты шифротекста. В этой ситуации буквы открытого текста, стоящие на нечетных позициях, замещаются соответствующими буквами второй строки, а стоящие на четных — третьей. Таким образом, исходное слово *hello* в шифротексте будет выглядеть как SHLJV. При этом буква «l», встречающаяся два раза, замещается разными символами. Итак, мы существенно усложнили применение статистических методов при атаке на шифр. Если теперь применить наивный частотный анализ, то мы не сможем найти символ шифра, подменяющий собой самую популярную английскую букву «e».

В этом примере мы, по существу, за один шаг шифруем две буквы. Следовательно, мы имеем дело с блочным шифром, блок которого равен двум английским буквам. На практике можно использовать не два, а вплоть до пяти различных алфавитов шифротекста, многократно увеличивая пространство ключей. Действительно, легко подсчитать, что если мы берем символы из пяти замещающих наборов, то число возможных ключей равно $(26!)^5 \approx 2^{441}$. Однако пользователю необходимо помнить, что в этом случае ключ — последовательность из $26 \cdot 5 = 130$ букв. Естественно, чтобы усложнить жизнь Еве, вскрывающей шифр, необходимо скрыть количество используемых алфавитов, считая его частью ключа. Но для среднего пользователя начала XIX века такая система шифрования казалась

слишком громоздкой, поскольку ключ был слишком большим, чтобы запомнить его.

Несмотря на указанный недостаток, самые известные шифры XIX столетия основывались именно на описанном принципе. *Шифр Виженера* был одним из вариантов полиалфавитного шифра замены, но имел несложный для запоминания ключ. С одной стороны, шифр Виженера является полиалфавитным блочным шифром, но его можно также отнести и к поточным шифрам, естественно обобщающим шифр сдвига.

Как блочный шифр алгоритм Виженера относится к полиалфавитным шифрам, множество замещающих наборов которого ограничивается 26 циклическими сдвигами стандартного алфавита. Если, например, в нем применяются 5 замещающих алфавитов, то пространство ключей сводится к $26^5 \approx 2^{23}$ возможностям, а в качестве ключа можно запомнить 5 чисел между 0 и 25.

Однако описание шифра Виженера как поточного шифра более естественно. Как и в случае шифра сдвига, мы снова перенумеруем буквы, начиная с 0. Секретный ключ здесь — это короткая последовательность букв (т. е. слово, часто называемое *лозунгом*), которое повторяется снова и снова, формируя поток ключей. Кодирование заключается в сложении букв открытого текста с буквами потока ключей (воспринимаемых как числа). Например, если ключом является слово *sesame*, то шифрование выглядит так:

```
  t h i s i s a t e s t m e s s a g e
+ s e s a m e s e s a m e s e s a m e
= L L A S U W S X W S F Q W W K A S I.
```

Заметим, что и в этом примере буква «а» замещается различными символами в зависимости от того, на каком месте открытого текста она стоит.

Шифр Виженера все же не очень сложно взломать, опираясь на статистику подлежащего языка. Как только мы узнаем длину ключевого слова, нам останется несколько раз применить тактику взлома шифра сдвига.

В качестве примера рассмотрим следующую криптограмму:

UTPDHUG NYH USVKCG MVCE FXL KQIB. WX RKU GI TZN, RLS BHZLXMSNP KDKS; CEB IH HKEW IBA, YYM SRB PFR SBS, JV UPL O UVADGR HRRWXF. JV ZTVOOV YH ZCQU Y UKWGEB, PL UQFB P FOUKCG, TBF RQ VHCF R KPG, OU KFT ZCQU MAW QKKW ZGSY, FP PGM QKFTK UQFB DER EZRN, MCYE, MG UCTFSVA, WP KFT ZCQU MAW KOIJS. LCOV

NTHDNV JPNUJVB IH GGV RWX ONKCGTHKFL XG VKD, ZJM VG CCI MVGD JPNUJ, RLS EWVKJT ASGUCS MVGD; DDK VG NYH PWUV CCHIIY RD DBQN RWTH PFRWBBI VTTK VCGNTGSF FL IAWU XJDUS, HFP VHSF, RR LAWEY QDFS RVMEES FZB CHH JRTT MVGZP UBZN FD ATIIYRTK WP KFT HIVJCI; TBF BLDPWPX RWTH ULAW TG VYCHX KQLJS US DCGCW OPPUPR, VG KFDNUJK GI JIKKC PL KGCJ IAOV KFTR GJFSAW KTZLZES WG RWXWT VWTL WP XPXGG, CJ FPOS VYC BTZCUW XG ZGJQ PMHTRAIBJG WMGFG. JZQ DPB JVYGM ZCLEWXR:CEB IAOV NYH JIKKC TGCWXF UHF JZK.

WX VCU LD YTTKFTK WPKCGVCWIQT PWVY QEBFKKQ, QNH NZTTW IRFL IAS VFRPE ODJRXGSPTC EKWPTGEES, GMCG TTVVPLTFFJ; YCW WV NYH TZYRWH LOKU MU AWO, KFPM VG BLTP VQN RD DSGG AWKWUKKPL KGCJ, XY OPP KPG ONZTT ICUJCHLSF KFT DBQHJTWUG. DYN MVCK ZT MFWCW HTWF FD JL, OPU YAE CH LQ! PGR UF, YH MWPP RXF CDJCGOSE, XMS UZGJQJL, SXVPN HBG!

Существует способ, с помощью которого можно определить длину лозунга, генерирующего поток ключей. Этот способ называют *тестом Казисского*. Его идея основана на периодичности потока ключей. Кроме того, в естественном языке существуют часто встречаемые буквосочетания: биграммы и триграммы. Учитывая это, возникает надежда, что повторяющиеся наборы символов в шифротексте — след повторений популярных биграмм и триграмм открытого текста. Расстояние между повторениями в таком случае должно быть кратно длине лозунга. Слодовательно, вычислив наибольший общий делитель всех таких расстояний, мы получим рабочую гипотезу о длине ключа.

Исследуем расстояния между повторениями биграммы WX в приведенном выше шифротексте. Некоторые из расстояний между повторениями этого буквосочетания равны 9, 21, 66 и 30. Отдельные совпадения могут возникнуть случайно, в то время как остальные содержат информацию о длине ключевого слова. Вычислим попарные наибольшие общие делители этих чисел.

$$\text{НОД}\,(30, 66) = 6,$$

$$\text{НОД}\,(3, 9) = \text{НОД}\,(9, 66) = \text{НОД}\,(9, 30) = \text{НОД}\,(21, 66) = 3.$$

Маловероятно, что ключевое слово состоит из трех букв, так что будем считать, что расстояния 9 и 21 попали случайно, а длина ключа равна 6.

Возьмем теперь каждую шестую букву шифротекста и применим частотный анализ, как мы уже делали, взламывая шифр сдвига,

и определим первую букву ключа. Преимущество гистограмм в частотном анализе здесь очевидно, поскольку наивный перебор всех возможных 26 ключей не сможет дать убедительного результата. Действительно, даже если каждая шестая буква шифротекста будет угадана, получить осмысленный текст мы не сможем. Так что метод гистограмм более продуктивен в этой ситуации.

На рисунке 3.3 дано сравнение частот повторяемости каждой шестой буквы, начиная с первой, со среднестатистической. Изучая эти гистограммы, мы отмечаем возможные замены букв «a», «e» и «t» естественного языка, и делаем вывод, что они сдвинуты на 2. Следовательно, первая буква ключа — «c», поскольку именно она обеспечивает такой сдвиг.

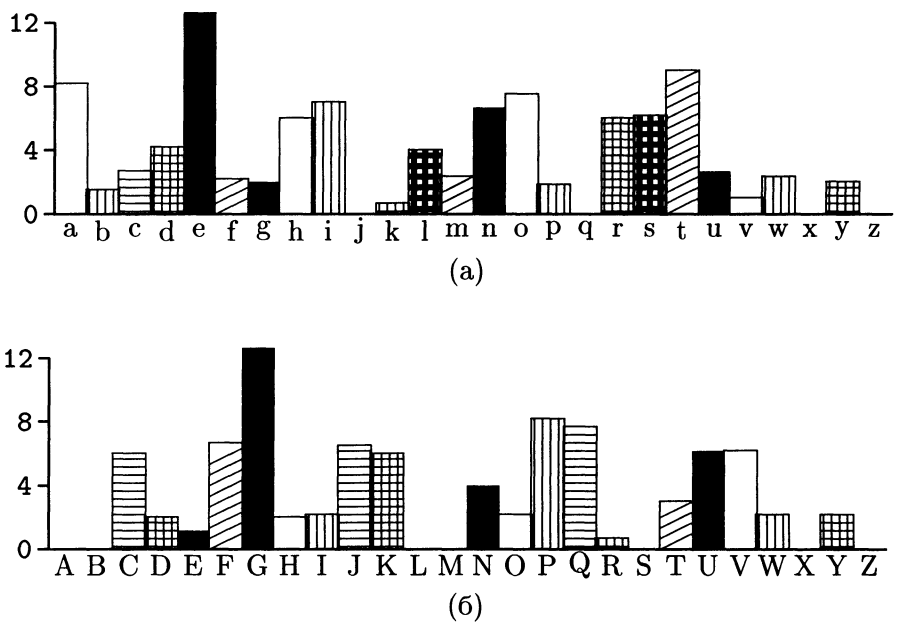


Рис. 3.3. Сравнение повторяемости каждой шестой буквы, начиная с первой, в примере шифра Виженера со среднестатистической

Применим аналогичный анализ для каждой шестой буквы, начиная со второй. Соответствующие гистограммы изображены на рис. 3.4. Используя ту же технику, мы видим, что гистограмма (б) «сдвинута» относительно диаграммы (а) на 17 позиций, что соответствует букве «r», стоящей на втором месте ключевого слова.

Продолжая в том же духе, найдем оставшиеся четыре буквы ключа и обнаружим, что лозунг — это «*crypto*». Теперь можно прочесть исходный текст:

Scrooge was better than his word. He did it all, and infinitely more; and to Tiny Tim, who did not die, he was a second father. He became as good a friend, as good a master, and as good a man, as the good old city knew, or any other good old city, town, or borough, in the good old world. Some people laughed to see the alteration in him, but he let them laugh, and little heeded them; for he was wise enough to know that nothing ever happened on this globe, for good, at which some people did not have their fill of laughter in the outset; and knowing that such as these would be blind anyway, he thought it quite as well that they should wrinkle up their eyes in grins, as have the malady in less attractive forms. His own heart laughed: and that was quite enough for him.

He had no further intercourse with Spirits, but lived upon the Total Abstinence Principle, ever afterwards; and it was always said of him, that he knew how to keep Christmas well, if any man alive possessed the knowledge. May that be truly said of us, and all of us! And so, as Tiny Tim observed, God bless Us, Every One!

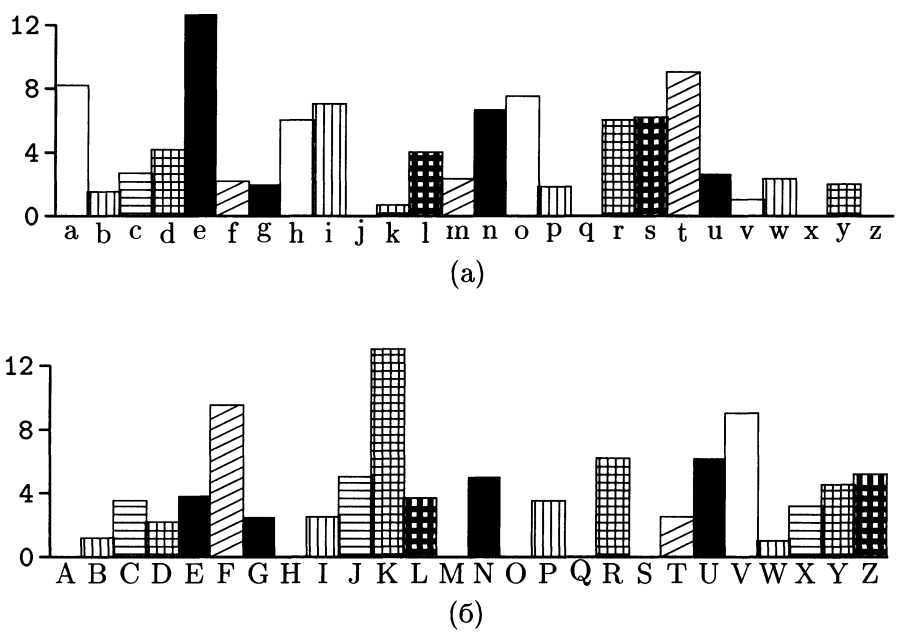


Рис. 3.4. Сравнение повторяемости каждой шестой буквы, начиная со второй, в примере шифра Виженера со среднестатистической

Отрывок взят из произведения Чарльза Диккенса «Рождественская песнь в прозе. Святочный рассказ с привидениями».

3.5. Перестановочные шифры

Идеи, лежащие в основе шифра замены, движут и современными криптографами, разрабатывающими симметричные алгоритмы. Позже мы увидим, что в шифрах *DES* и *Rijndael* присутствуют компоненты, называемые S-блоками, которые являются простыми подстановками. Другие составляющие современных симметричных шифров основываются на перестановках.

Перестановочные шифры активно применялись в течение нескольких столетий. Здесь мы опишем один из самых простейших, который легко поддается взламыванию.

Фиксируется симметрическая группа S_n и какой-то ее элемент $\sigma \in S_n$. Именно перестановка σ является секретным ключом. Предположим, что

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 1 & 3 & 5 \end{pmatrix} = (1243) \in S_5$$

и зашифруем с ее помощью открытый текст:

Once upon a time there was a little girl called Snow White.

Разобьем текст на части по 5 букв:

onceu ponat imeth erewa salit egirl calle dsnow white.

Затем переставим буквы в них в соответствии с нашей перестановкой:

coenu npaot eitmh eewra lsiat iergl lclae ndosw iwthe.

Убрав теперь промежутки между группами, чтобы скрыть значение n, получим шифротекст

coenunpaoteitmheewralsiatiergllclaendoswiwthe.

Перестановочный шифр поддается взлому атакой с выбором открытого текста в предположении, что участвующая в шифровании использованная симметрическая группа (т. е. параметр n) не слишком велика. Для этого нужно навязать отправителю шифрованных сообщений нужный нам открытый текст и получить его в зашифрованном виде. Предположим, например, что нам удалось подсунуть алфавит

abcdefghijklmnopqrstuvwxyz

и получить его шифрованную версию

CADBEHFIGJMKNLORPSQTWUXVYZ.

Сопоставляя открытый текст и криптограмму, получаем перестановку

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & \dots \\ 2 & 4 & 1 & 3 & 5 & 7 & 9 & 6 & 8 & 10 & 12 & 14 & 11 & 13 & 15 & \dots \end{pmatrix}.$$

После короткого анализа полученной информации мы обнаруживаем повторяющиеся серии перестановок: каждые пять чисел переставляются одинаково. Таким образом, можно сделать вывод, что $n = 5$, и восстановить ключ, взяв первые пять столбцов этой таблицы.

3.6. Одноразовый шифр–блокнот

Во время Первой Мировой Войны криптологи активно использовали так называемый одноразовый шифр-блокнот, который часто называют шифром Вернама. Этот алгоритм подробно изучается в четвертой главе, а сейчас мы перейдем к другой, не менее интересной криптосистеме.

3.7. Роторные машины и «Энигма»

С наступлением 1920 года назрела необходимость механизировать процесс шифрования. Наиболее подходящим для этого типом шифра казался подстановочный. Для механизации процесса шифрования брался полый диск с нанесенными с двух сторон контактами, соответствующими алфавитам открытого и шифрованного текста, причем контакты соединялись между собой по некоторой подстановке, называемой коммутацией диска. Эта коммутация определяла замену букв в начальном угловом положении. При изменении углового положения диска изменялась и соответствующая замена на сопряженную подстановку. Отсюда название механического устройства — *ротор*, или *роторная машина*.

Предположим, что коммутация диска задает подстановку

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

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

которая реализуется в начальном угловом положении. Тогда первая буква сообщения замещается согласно этому соответствию. Перед шифрованием второй буквы открытого текста коммутационный диск поворачивается на одну позицию и получается другое правило замещения:

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

Для третьей буквы используется очередная подстановка:

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

и т. д. Все это дает нам полиалфавитный шифр замены с 26 алфавитами.

Наиболее знаменитой из роторных машин была машина «Энигма», стоявшая на вооружении Германии во время второй мировой войны. Мы опишем самую простую версию «Энигмы», содержащую лишь три коммутационных диска, которые реализовывали три из следующих перестановок:

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

Машины, применяемые в конце войны, имели большее число дисков, выбираемых из большего набора перестановок. Обратите внимание, что порядок компоновки дисков в машине существенен. Поэтому число возможных компоновок дисков в нашем примере равно

$$5 \cdot 4 \cdot 3 = 60.$$

При каждом повороте первого ротора соединенное с ним кольцо попадает в паз второго диска и толкает его. Аналогично пошаговые итерации третьего ротора контролируются вторым ротором. Оба кольца подвижны, и их положения тоже формируют часть пространства ключей. Число всех положений двух колец составляет $26^2 = 676$. За счет вращения дисков конкретная буква текста замещалась разными символами при каждом нажатии на клавишу.

Наконец, для двойной замены букв при каждом шифровании использовалась штекерная панель, что увеличивало сложность и добавляло еще 10^{14} возможных ключей.

Таким образом, в формировании пространства ключей принимали участия коммутации дисков, порядок их компоновки, начальные угловые положения дисков и положения колец. В результате общее число секретных ключей доходило до 2^{75}.

Для контроля соответствия операций шифрования и расшифрования использовался так называемый рефлектор, в качестве которого выступала фиксированная открытая подстановка, заданная таблицей

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

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

На рис. 3.5 приведено схематическое изображение упрощенной машины «Энигма». Жирными линиями выделен путь, по которому буква «a» открытого текста заменяется на знак «D» шифротекста. Заметим, что шифрование и расшифрование должно осуществляться машинами, находящимися в одном и том же положении. Предположим теперь, что первый диск провернули на одну позицию, так что «a» перейдет в «D» под воздействием первого диска, «b» в «A», «c» в «C» и «d» в «B». Вам стоит продумать, куда перейдет «a» открытого текста при втором шаге шифрования (см. рис. 3.5).

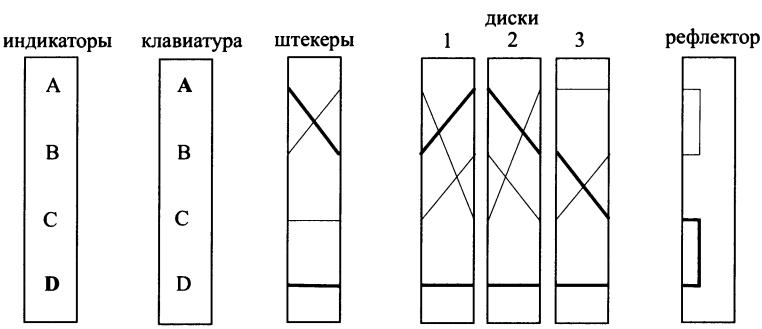


Рис. 3.5. Упрощенная машина «Энигма»

При эксплуатации «Энигмы» немцы ежедневно меняли следующие установки:

- расположение штекеров,
- коммутационные диски и их компоновку,
- позиции колец,
- начальные угловые положения дисков.

Однако правильнее было бы менять ключ с передачей каждого нового сообщения, поскольку шифрование слишком большого количества информации с одними и теми же установками — неудачная практика. Именно этот недосмотр в режиме работы «Энигмы» позволил сотрудникам Блетчлей Парка раскрыть график передвижения германских войск. Подчеркнем, что утечка информации про-

изошла ввиду неверной эксплуатации машины, а не из-за слабой криптостойкости алгоритма.

Режим эксплуатации «Энигмы» во время военных действий выглядел следующим образом. Ежедневно перед отправкой первого шифрованного сообщения менялись и фиксировались на весь день настройки машины. Выбиралась короткая последовательность букв, скажем «a, f, g» (на современном языке это называется *сеансовым ключом*), которая дважды шифровалась. Полученный шифротекст, т. е. шестибуквенная последовательность (в нашем примере «G, H, K, L, P, T»), передавалась в самом начале сообщения. Адресат, получив комбинированный шифротекст, дешифровал сеансовый ключ и в соответствии с результатом менял настройки машины. После этого расшифровывался текст первого сообщения. Именно эта схема использования позволила людям из Блетчлей Парка взломать «Энигму». О подробностях, касающихся взлома этого шифра, можно прочесть в книгах, указанных в разделе «Дополнительная литература» этой главы.

У «Энигмы» было и другое слабое место, известное сотрудникам разведки, которое они так никогда и не использовали. Наличие этого дополнительного недостатка показывает, что система шифрования «Энигма» нестойка даже в том случае, если эксплуатируется правильно. По существу, штекерная панель и композиционные диски — ортогональные механизмы шифрования, что допускает успешную атаку на отдельный достаточно большой шифротекст. Расскажем о схеме такой атаки.

Допустим, у нас есть большое шифрованное сообщение, которое мы намерены взломать. Сначала мы игнорируем участие штекерной панели в алгоритме и пытаемся нащупать положения всех коммутационных дисков и колец. Критерием корректности подбираемого расположения служит близость статистики частично расшифрованного текста к естественной. Как только мы получили искомую близость, можно с достаточной долей вероятности утверждать, что найденные установки коммутационных дисков и колец действительно верные. После этого мы поочередно помещаем штекеры на панель до полной расшифровки текста. Такой подход к взлому не был использован в военных действиях, поскольку для его успешной реализации необходим пространный шифротекст, а большинство реальных перехваченных немецких сообщений были очень короткими.

Для осуществления описанной выше атаки нам необходимо понять, близок ли шифр криптограммы по своей природе к подстановочным шифрам. Другими словами, произошел ли данный шифротекст из чего-либо похожего на естественный язык.

Один из статистических методов, который мы хотим предложить Вашему вниманию, впервые был применен Фридманом в 1920 г. и носит название *индекс совпадения*.

Определение 3.1. (**Индекс совпадения.**) Пусть $x_1, \ldots, x_n$ — строка букв, а $f_0, \ldots, f_{25}$ — числа появлений букв в строке. Величина

$$IC(x) = \frac{\sum\limits_{i=0}^{25} f_i(f_i - 1)}{n(n-1)}$$

называется *индексом совпадений*.

Заметим, что для случайного набора букв этот индекс равен

$$IC(x) \approx 0,038,$$

в то время как для осмысленного текста, написанного на английском или немецком языках он равен

$$IC(x) \approx 0,065.$$

Эта замечательная находка используется при атаке на шифр «Энигма» следующим образом.

1. Найдем порядок коммутационных дисков. Освободим панель от штекеров и установим диски в позицию «*a, a, a*». Перебирая все угловые положения дисков и их взаимные расположения, определим ту комбинацию, которая даст самый высокий индекс совпадения IC для преобразованного текста. На это потребуется

$$60 \cdot 26^3 \approx 2^{20}$$

операций расшифрования.

2. Аппроксимируем начальные угловые положения дисков. Тот факт, что начальные угловые положения дисков на предыдущем шаге были точно определены, не вызывает доверия. Однако мы верим в то, что их взаимное расположение найдено верно. Теперь выясним стартовые угловые позиции. На этом этапе штекерная панель все еще остается пустой, а диски вновь устанавливаются в позицию «*a, a, a*», с соблюдением порядка, найденного на предыдущем шаге. Проходим через все положения каждого из трех коммутационных дисков и первого кольца, расшифровывая сообщения при каждой комбинации. Вновь стремимся приблизить коэффициент IC к максимальному значению. При этом происходит

$$26^4 \approx 2^{19}$$

расшифрований. После указанной процедуры мы определим наибо-

лее вероятное приближение к начальным позициям коммутационных дисков и нащупаем положение первого кольца.

3. Определение начальных угловых положений. К этому моменту мы знаем как порядок следования дисков, так и начальные положения первого кольца и первого ротора. Кроме того, мы уже обнаружили приблизительные начальные положения остальных дисков. Теперь мы перебираем все положения второго кольца и второго диска, повторяя предыдущие действия. Это потребует

$$26^2 \approx 2^9$$

операций. В результате мы найдем точные положения второго кольца и второго коммутационного диска. Похожая процедура проводится и для третьего диска. Теперь мы знаем порядок дисков, их начальные угловые положения и позиции колец. Единственное, что остается, — это выявить комбинацию штекеров.

4. Определение позиций штекеров. Мы подбираем штекеры поочередно до тех пор, пока не сможем прочитать шифротекст. Это можно сделать с помощью индекса совпадения IC (что требует очень большого шифротекста) или статистического теста, основанного на информации о распределении триграмм подлежащего языка.

Краткое содержание главы

- Многие шифры, применявшиеся на ранней стадии развития криптологии, были взломаны, поскольку не смогли скрыть статистики подлежащего языка.
- Важнейшие приемы, лежащие в основе первых шифров — это замены и перестановки.
- Шифры работали с блоками знаков посредством снабженного ключом алгоритма или просто прибавляли очередное число из потока ключей к каждой букве открытого текста.
- Шифры, предназначенные устранить недостатки, связанные со статистикой языка, оказались менее стойкими, чем ожидалось, либо из-за конструкционных недочетов, либо в связи с неграмотной политикой использования, адаптированной в интересах операторов.

Дополнительная литература

Лучшая книга по истории шифров написана Каном. Однако она слишком большая, и тому, кто хочет получить быстрое представление о истории криптографии, можно рекомендовать книгу Сингха. Монография Черчхауса тоже содержит краткий обзор множества

исторических шифров. Особенно хорошо в ней рассказано о машине «Энигма» и истории ее взлома.

R. Churchhouse. *Codes and Ciphers. Julius Caesar, the Enigma and the Internet*. Cambridge University Press, 2001.

D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

S. Singh. *The Codebook: The Evolution of Secrecy from Mary Queen of Scots to Quantum Cryptography*. Doubleday, 2000.

Контрольные вопросы

3.1.1. Назовите три наиболее популярные буквы английского алфавита.

3.1.2. Какие из биграмм и триграмм английского языка наиболее употребительны?

3.1.3. Объясните, какое отношение имеет шифр Виженера к шифру сдвига.

3.1.4. Расскажите, как шифр замены связан с машиной «Энигма».

3.1.5. Покажите, чем похожи и в чем разнятся перестановки и замены как компоненты шифров.

3.1.6. Приведенный ниже шифротекст был получен с помощью шифра сдвига. Расшифруйте его.

PELCG BTENC ULVFN AVAGR ERFGV ATFHO WRPG.

Лабораторные работы

3.2.1. Напишите программы, реализующие шифрование и расшифрование (при условии известного ключа), для следующих шифров:

(а) сдвига,
(б) Виженера,
(в) замены,
(г) «Энигма».

Для компьютерной реализации последнего Вам стоит чуть больше узнать о шифре «Энигма» из интернета или книги Черчхауса.

3.2.2. Создайте программу, которая дешифрует шифротекст, полученный с помощью шифра сдвига (предполагая, что соответствующий открытый текст написан по-английски).

3.2.3. Разработайте программу, имеющую в качестве входных данных шифротекст, полученный при помощи шифра Виженера, и выдающую набор символов, наиболее близкий к открытому тексту. Считайте, что используется английский язык.

3.2.4*. Сделайте аналогичную программу для «Энигмы».

3.2.5. Насколько большой шифротекст Вы должны обработать Вашими программами для получения достаточно разумного результата?

Упражнения

3.3.1. *Книжный шифр* похож на шифр Виженера, но для формирования потока ключей вместо повторяющегося лозунга берется часть текста из книги, начинающегося с заранее оговоренного места. Следующее сообщение зашифровано с помощью книжного шифра:

BAACG WLTIV SLSKH ZFSVI RESSM HPACW LPCHB BUIK.

Попытайтесь его расшифровать.

3.3.2. Криптоаналитик при атаке с выбором открытого текста пытается подсунуть противнику для шифрования открытый текст по своему выбору. Покажите, что шифры Цезаря, Виженера и замены могут быть мгновенно взломаны таким способом. Для каждого из этих шифров определите наименьшую длину открытого текста, достаточную для взлома.

ГЛАВА 4

ТЕОРЕТИКО-ИНФОРМАЦИОННАЯ СТОЙКОСТЬ

Цели главы

- Ввести концепцию абсолютной стойкости.
- Обсудить стойкость одноразового шифр-блокнота.
- Ввести понятие энтропии.
- Объяснить понятия ложного ключа, фиктивных ключей и расстояния единственности.
- Использовать эти понятия для объяснения слабой криптостойкости ранних алгоритмов шифрования.

4.1. Введение

Теория информации — один из краеугольных камней вычислительной техники. В данной главе изучается ее отношение к криптографии. Мы не будем предполагать, что читатель знаком с этой теорией.

На первом этапе нам потребуется общее представление о разнице между теоретико-информационной стойкостью (или теоретической стойкостью) и вычислительной защищенностью (или практической стойкостью). Говоря неформально, криптографическая система называется *вычислительно защищенной* (или вычислительно стойкой), если наилучший из возможных алгоритмов, взламывающих ее, требует неоправданно высоких затрат вычислительных ресурсов. Принимая во внимание мощность современных компьютеров, можно считать, что 2^{80} операций, необходимых для взлома шифра, это тот предел, выходя за который алгоритмы взлома становятся слишком дорогостоящими. Таким образом, если минимальное число N операций, необходимых алгоритму, атакующему данную криптосистему, больше 2^{80}, то говорят, что она вычислительно защищена. Заметим, что никакую реальную систему нельзя

обоснованно считать вычислительно защищенной, поскольку мы не сможем доказать оптимальность найденного метода взлома. Поэтому на практике мы утверждаем ее защищенность в вычислительном отношении в том случае, если лучший из *известных* алгоритмов для ее взлома требует недопустимо большого количества вычислений.

Другой практический подход, связанный с вычислительной защищенностью, состоит в том, чтобы свести взлом системы к решению хорошо изученных трудных проблем. Например, мы можем попытаться показать, что вскрытие конкретной криптосистемы равносильно разложению данного большого целого числа на множители. Такие системы часто называют *доказуемо стойкими*. Однако при этом стойкость системы обосновывается сведением к трудной задачи, что нельзя считать корректным доказательством.

По существу, вычислительно, или доказуемо, стойкая криптосистема является стойкой по отношению к противнику, чьи вычислительные ресурсы ограничены. Даже в том случае, когда противник обладает большими, но ограниченными ресурсами, он все еще не сможет взломать систему.

При изучении вычислительно защищенных схем необходимо иметь четкие представления о некоторых проблемах:

- Нужно позаботиться о длине ключа. Если размер ключа мал, то у противника вполне может хватить ресурсов для взлома криптосистемы.
- Важно следить за последними алгоритмическими достижениями и развитием компьютерной техники.
- Мы должны быть готовы к тому, что наша криптосистема в какой-то момент будет взломана либо из-за усовершенствования аппаратных средств, либо в результате крупного алгоритмического прорыва.

Большинство криптосистем, активно эксплуатирующихся в настоящее время, вычислительно защищены. Поэтому в каждой из последующих глав мы будем уделять много внимания вычислительной стойкости изучаемых криптосистем.

С другой стороны, система называется *абсолютно стойкой* или *совершенной*, если мы не ограничиваем вычислительной мощности противника. Иначе говоря, криптосистема совершенна, если ее нельзя взломать даже с помощью бесконечного числа операций. Следовательно, независимо от алгоритмических достижений и совершенства вычислительной техники, абсолютно стойкую схему взломать невозможно. В литературе можно встретить и другой тер-

мин, закрепленный за абсолютной стойкостью, а именно, теоретико-информационная стойкость.

Мы уже видели, что следующие системы не являются вычислительно защищенными, поскольку их можно полностью раскрыть, имея довольно скромные компьютерные мощности:

- шифр сдвига,
- шифр замены,
- шифр Виженера.

К вычислительно стойким системам можно отнести криптосистемы

- DES и Rijndael,
- RSA,
- ЭльГамаля.

С ними мы познакомимся в следующих разделах. Однако упомянутые системы не являются абсолютно стойкими.

Чуть позже в этой главе мы познакомимся с одноразовым шифр-блокнотом. Вот эту систему можно назвать совершенной, но только в том случае, если она используется корректно.

4.2. Вероятность и шифры

Прежде чем дать формальное определение абсолютной стойкости, нам необходимо в деталях осознать роль вероятности в понимании простых шифров. Зафиксируем следующие обозначения:

$\mathbb{P}$ — множество возможных открытых текстов, т. е. пространство сообщений;

$\mathbb{K}$ — совокупность возможных ключей;

$\mathbb{C}$ — множество шифротекстов.

Каждое из этих множеств можно понимать как пространство событий, в котором вероятности обозначаются как $p(P = m)$, $p(K = k)$, $p(C = c)$.

Например, если наше пространство сообщений имеет вид $\mathbb{P} = \{a, b, c\}$ и сообщение a встречается с вероятностью $1/4$, то мы пишем $p(P = a) = \frac{1}{4}$.

Будем считать, что любые события $P \in \mathbb{P}$ и $K \in \mathbb{K}$ независимы, т. е. пользователь не меняет ключ шифра в зависимости от содержания открытого текста, подлежащего шифрованию. Символом $\mathbb{C}(k)$

обозначим множество всех криптограмм, полученных из пространства сообщений шифрованием посредством ключа k:

$$\mathbb{C}(k) = \{E_k(x) |\ x \in \mathbb{P}\},$$

где E_k — шифрующая функция. Имеет место соотношение

$$p(C = c) = \sum_{k:\ c \in \mathbb{C}(k)} p(K = k) \cdot p(P = D_k(c)), \qquad (4.1)$$

где D_k — расшифровывающая функция.

В качестве базисного примера этого параграфа рассмотрим пространство сообщений $\mathbb{P} = \{a, b, c, d\}$ с распределением вероятностей

$$p(P = a) = \tfrac{1}{4}, \qquad p(P = b) = \tfrac{3}{10},$$
$$p(P = c) = \tfrac{3}{20}, \qquad p(P = d) = \tfrac{3}{10}.$$

Предположим, что пространство ключей имеет вид $\mathbb{K} = \{k_1, k_2, k_3\}$, а вероятности их выбора равны

$$p(K = k_1) = \frac{1}{4}, \quad p(K = k_2) = \frac{1}{2}, \quad p(K = k_3) = \frac{1}{4}.$$

Пусть, наконец, $\mathbb{C} = \{1, 2, 3, 4\}$, а функция шифрования представлена таблицей:

	a	b	c	d
k_1	3	4	2	1
k_2	3	1	4	2
k_3	4	3	1	2

Используя формулу (4.1), вычислим распределение вероятностей на множестве $\mathbb{C}$.

$$p(C = 1) = p(K = k_1)p(P = d) + p(K = k_2)p(P = b) + p(K = k_3)p(P = c) = 0,2625,$$

$$p(C = 2) = p(K = k_1)p(P = c) + p(K = k_2)p(P = d) + p(K = k_3)p(P = d) = 0,2625,$$

$$p(C = 3) = p(K = k_1)p(P = a) + p(K = k_2)p(P = a) + p(K = k_3)p(P = b) = 0,2625,$$

$$p(C = 4) = p(K = k_1)p(P = b) + p(K = k_2)p(P = c) + p(K = k_3)p(P = a) = 0,2125.$$

Отсюда видно, что шифротексты распределены почти однородно.

Для пары $c \in \mathbb{C}$ и $m \in \mathbb{P}$ можно вычислить условную вероятность $p(C = c|P = m)$, т. е. вероятность выбора шифротекста c, если известно, что открытый текст — это m. Справедлива формула

$$p(C = c|P = m) = \sum_{k:\ m=d_k(c)} p(K = k). \tag{4.2}$$

Здесь сумма берется по всем ключам k, для которых значение d_k на шифрограмме c совпадает с данным открытым текстом m. Вычислим условные вероятности для базисного примера.

$$\begin{aligned}
&p(C = 1|P = a) = 0, && p(C = 2|P = a) = 0,\\
&p(C = 3|P = a) = 0,75, && p(C = 4|P = a) = 0,25,\\
&p(C = 1|P = b) = 0,5, && p(C = 2|P = b) = 0,\\
&p(C = 3|P = b) = 0,25, && p(C = 4|P = b) = 0,25,\\
&p(C = 1|P = c) = 0,25, && p(C = 2|P = c) = 0,25,\\
&p(C = 3|P = c) = 0, && p(C = 4|P = c) = 0,5,\\
&p(C = 1|P = d) = 0,25, && p(C = 2|P = d) = 0,75,\\
&p(C = 3|P = d) = 0, && p(C = 4|P = d) = 0.
\end{aligned}$$

Однако при взломе шифра нам нужна условная вероятность другого сорта. То есть мы хотим знать вероятность открытого текста при данном шифротексте. Вероятность того, что m — действительно исходное сообщение для данной криптограммы, вычисляется по формуле:

$$p(P = m|C = c) = \frac{p(P = m)p(C = c|P = m)}{p(C = c)}.$$

Таким образом, требуемые условные вероятности способен определить любой, кому известна функция шифрования и распределение вероятностей на множествах $\mathbb{K}$ и $\mathbb{P}$. Опираясь на эти условные вероятности, можно сделать определенные выводы относительно открытого текста, как только Вы увидите криптограмму.

Вооружившись этой полезной формулой, вычислим для нашего примера следующие вероятности:

$$\begin{aligned}
&p(P = a|C = 1) = 0, && p(P = b|C = 1) = 0,571,\\
&p(P = c|C = 1) = 0,143, && p(P = d|C = 1) = 0,286,
\end{aligned}$$

$$p(P = a|C = 2) = 0, \qquad p(P = b|C = 2) = 0,$$
$$p(P = c|C = 2) = 0,143, \qquad p(P = d|C = 2) = 0,857,$$

$$p(P = a|C = 3) = 0,714 \qquad p(P = b|C = 3) = 0,286,$$
$$p(P = c|C = 3) = 0, \qquad p(P = d|C = 3) = 0,$$

$$p(P = a|C = 4) = 0,294, \qquad p(P = b|C = 4) = 0,352,$$
$$p(P = c|C = 4) = 0,352, \qquad p(P = d|C = 4) = 0.$$

Анализируя полученные результаты, можно сделать следующие выводы:

- Если мы получили шифротекст 1, то точно знаем, что исходное сообщение не может быть a. Более того, предположение о том, что соответствующий открытый текст — b, более вероятно, чем c или d.
- При получении шифротекста 2 мы уверены, что исходное сообщение — не a и не b, а скорее всего — d.
- Увидев третью криптограмму, мы заключаем, что отправленное сообщение не может быть ни c и ни d, но имеет большие шансы оказаться текстом a.
- Если у нас оказалась последняя шифрограмма, то отправленное сообщение отлично от d. Однако у нас очень мало оснований предпочесть какое-либо из оставшихся.

Таким образом, шифротекст в нашем примере дает немало информации относительно оригинального сообщения. Но именно этого мы и хотим избежать. Мы стремимся к тому, чтобы по криптограмме невозможно было получить какую-либо информацию об открытом тексте.

Криптосистема, шифротекст в которой не дает никакой информации о соответствующем открытом тексте, называется *абсолютно стойкой*, или *совершенной*.

Определение 4.1. Криптосистема обладает *абсолютной стойкостью*, если равенство

$$p(P = m|C = c) = p(P = m)$$

имеет место для всех открытых текстов $m \in \mathbb{P}$ и всех криптограмм $c \in \mathbb{C}$.

Иначе говоря, наличие шифрограммы никак не сказывается на вероятности того, что открытый текст совпадает с m. Вновь под-

черкнем: абсолютная стойкость шифра означает, что шифротекст не несет в себе сведений об открытом тексте[1]. Второй способ определения абсолютной стойкости дает следующая лемма.

Лемма 4.2. Криптосистема является абсолютно стойкой, если

$$p(C = c|P = m) = p(C = c)$$

для всех m и c.

Доказательство. Утверждение леммы немедленно следует из определения абсолютной стойкости и формулы

$$p(P = m|C = c) = \frac{p(P = m)p(C = c|P = m)}{p(C = c)}.$$

■

Первое свойство абсолютно стойких криптосистем сформулировано в лемме 4.3.

Лемма 4.3. Для абсолютно стойкой криптосистемы имеет место неравенство:

$$\#\mathbb{K} \geqslant \#\mathbb{C} \geqslant \#\mathbb{P},$$

где $\#\mathbb{K}$ — число возможных ключей, $\#\mathbb{C}$ — количество возможных шифрограмм и $\#\mathbb{P}$ — размер пространства сообщений.

Доказательство. Прежде всего заметим, что в любой криптосистеме выполнено равенство:

$$\#\mathbb{C} \geqslant \#\mathbb{P},$$

поскольку шифрующая функция должна быть инъективной.

Предположим, что при шифровании может получиться любой шифротекст $c \in \mathbb{C}$, т. е. $\forall c \in \mathbb{C}\ p(C = c) > 0$. Если это не так, то можно подправить определение множества $\mathbb{C}$, выбросив из него лишние элементы. Тогда любое сообщение m и шифротекст c абсолютно стойкой криптосистемы удовлетворяют соотношению

$$p(C = c|P = m) = p(C = c) > 0.$$

Из него следует, что для любой пары m и c найдется такой ключ k, что $p(K = k) > 0$ и $m = d_k(c)$ (см. (4.2)). Следовательно, зафиксировав открытый текст $m = m_0$, мы можем каждой шифрограмме c поставить в соответствие такой ключ $k(c)$, что $d_{k(c)}(c) = m_0$. При этом, естественно, разным шифротекстам будут отвечать разные ключи, откуда $\#\mathbb{K} \geqslant \#\mathbb{C}$. ■

[1] Если, конечно, не знать ключа. — *Прим. перев.*

Это подводит нас к основной теореме Шеннона, которая дает критерий абсолютной стойкости шифра.

Теорема 4.4. (Шеннон.) Пусть набор

$$(\mathbb{P}, \mathbb{C}, \mathbb{K}, e_k(\cdot), d_k(\cdot))$$

обозначает симметричную криптосистему, в которой $\#\mathbb{P} = \#\mathbb{C} = \#\mathbb{K}$. Она обладает абсолютной стойкостью тогда и только тогда, когда

- использование всех ключей равновероятно, т. е. $p(K = k) = \frac{1}{\#\mathbb{K}}$ $\forall k \in \mathbb{K}$;
- для каждой пары $m \in \mathbb{P}$ и $c \in \mathbb{C}$ существует единственный ключ k, такой что $e_k(m) = c$.

Доказательство. Предположим, что криптосистема наделена абсолютной стойкостью. Тогда, как мы уже убедились при доказательстве предыдущей леммы, для каждой пары $m \in \mathbb{P}$ и $c \in \mathbb{C}$ найдется такой ключ $k \in \mathbb{K}$, при котором $d_k(c) = m$. Ввиду симметричности криптосистемы это равенство влечет соотношение: $e_k(m) = c$. Отсюда, благодаря предположению $\#\mathbb{C} = \#\mathbb{K}$, получаем

$$\#\{e_k(m) |\ k \in \mathbb{K}\} = \#\mathbb{K},$$

т. е. не существует таких двух ключей k_1 и k_2, что $e_{k_1}(m) = e_{k_2}(m) = c$. Так что для всех $m \in \mathbb{P}$ и $c \in \mathbb{C}$ существует ровно один ключ $k \in \mathbb{K}$ со свойством $e_k(m) = c$. В частности, тройка (m, c, k) удовлетворяет соотношению

$$P(C = c | P = m) = P(K = k).$$

Покажем, что применение всех ключей равновероятно, т. е.

$$p(K = k) = \frac{1}{\#\mathbb{K}} \quad \forall k \in \mathbb{K}.$$

Пусть $n = \#\mathbb{K}$ и $\mathbb{P} = \{m_i,\ 1 \leqslant i \leqslant n\}$. Фиксируем $c \in \mathbb{C}$ и пронумеруем ключи $k_1, \ldots, k_n$ так, что

$$e_{k_i}(m_i) = c \quad \text{при} \quad 1 \leqslant i \leqslant n.$$

Опираясь на абсолютную стойкость криптосистемы, т. е. на соотношение

$$p(P = m_i | C = c) = p(P = m_i),$$

получаем $p(P = m_i) = p(P = m_i | C = c) =$

$$= \frac{p(C = c|P = m_i)p(P = m_i)}{p(C = c)} =$$
$$= \frac{p(K = k_i)p(P = m_i)}{p(C = c)}.$$

Следовательно, для всех $1 \leqslant i \leqslant n$

$$p(C = c) = p(K = k_i).$$

Из этого равенства, ввиду фиксированности c, вытекает, что все ключи используются с одинаковой вероятностью

$$p(K = k) = p(C = c) = \frac{1}{\#\mathbb{K}} \quad \text{для всех } k \in \mathbb{K}.$$

Докажем вторую часть критерия, а именно, предположив, что

- $\#\mathbb{K} = \#\mathbb{C} = \#\mathbb{P}$,
- использование ключей равновероятно,
- для любой пары $m \in \mathbb{P}$ и $c \in \mathbb{C}$ существует единственный ключ k, для которого $e_k(m) = c$,

покажем справедливость равенства

$$p(P = m|C = c) = p(P = m).$$

Из равновероятности применения ключей получаем

$$p(C = c) = \sum_k p(K = k)p(P = d_k(c)) =$$
$$= \frac{1}{\#\mathbb{K}} \sum_k p(P = d_k(c)).$$

Кроме того, поскольку для каждой пары m и c существует единственный ключ k, переводящий m в c, имеем

$$\sum_k p(P = d_k(c)) = \sum_m p(P = m) = 1.$$

Значит, $p(C = c) = 1/\#\mathbb{K}$. Если $c = e_k(m)$, то $p(C = c|P = m) = p(K = k) = 1/\#\mathbb{K}$. Теперь из теоремы Байеса вытекает

$$p(P = m|C = c) = \frac{p(P = m)p(C = c|P = m)}{p(C = c)} =$$
$$= \frac{p(P = m)\frac{1}{\#\mathbb{K}}}{\frac{1}{\#\mathbb{K}}} = p(P = m).$$

Теорема доказана. ■

Закончим параграф обсуждением двух систем с абсолютной стойкостью.

4.2.1. Модифицированный шифр сдвига

Напомним, что шифром сдвига считается тот шифр, при котором шифрование заключается в «прибавлении» фиксированной буквы к каждому знаку открытого текста. Модифицируем его, выбирая свой ключ для каждой буквы сообщения. Например, для шифрования слова «hello» мы возьмем 5 случайных ключей, скажем fuiat. Затем прибавим по модулю 26 ключевое слово к открытому тексту и получим шифрограмму MYTLH. Обратите внимание, что буква «l» открытого текста переходит в разные символы шифровки.

При использовании в шифре сдвига разных равномерно распределенных случайных ключей для каждой буквы текста получается абсолютно стойкая криптосистема. Чтобы проверить это, рассмотрим шифрование сообщения из n букв. Тогда количество всех ключей, криптограмм и открытых текстов равно

$$\#\mathbb{K} = \#\mathbb{C} = \#\mathbb{P} = 26^n.$$

К тому же каждый ключ будет применяться с одной и той же вероятностью $p(K = k) = \frac{1}{26^n}$, а для любой пары m и c выделен единственный ключ $k = c - m \pmod{26}$ со свойством $e_k(m) = c$. В результате, благодаря теореме Шеннона, можно заключить, что описанная криптосистема обладает абсолютной стойкостью.

4.2.2. Шифр Вернама

Главная операция описанного в предыдущем подпараграфе модифицированного шифра сдвига — сложение по модулю 26, что с точки зрения вычислительной техники является слишком дорогостоящим, в то время как двоичная арифметика сравнительно легка. В первую очередь нас интересует сложение по модулю 2, которое равносильно логической операции «исключающее ИЛИ», обозначаемой обычно «XOR»:

$\oplus$	0	1
0	0	1
1	1	0

В 1917 г. Гильберт Вернам запатентовал шифр, основанный на этой операции, который теперь называют *шифром Вернама* или *одноразовым шифр-блокнотом*. Для пересылки строки битов необхо-

дим ключ, состоящий из того же количества двоичных знаков, что и сообщение. Каждый бит посылаемой строки складывается по модулю 2 с соответствующим знаком ключа и получается криптограмма.

Любой ключ разрешается использовать для шифрования только один раз, отсюда и название: *одноразовый шифр-блокнот*. Таким образом, распределение ключей здесь — основная проблема, к которой мы будем возвращаться снова и снова. Чтобы убедиться в неизбежности неприятностей при шифровании нескольких текстов одним ключом, рассмотрим атаку с выбором открытого текста. Предположим, что Алиса всегда пользуется одним ключом для шифрования данных. Ева пытается определить этот ключ и предпринимает следующую атаку:

- генерирует сообщение m и подсовывает его Алисе для шифрования,
- получает $c = m \oplus k$,
- вычисляет $k = c \oplus m$.

Вы можете возразить, что Алиса не настолько глупа, чтобы шифровать сообщение Евы. Но при разработке криптосистемы мы обязаны учесть любые ситуации, в том числе и глупого пользователя, т. е. создать «защиту от дураков».

Другая проблема, возникающая при двойном употреблении ключа, состоит в следующем. Допустим, Ева перехватила два сообщения, зашифрованные одним ключом

$$c_1 = m_1 \oplus k, \qquad c_2 = m_2 \oplus k.$$

Тогда она может получить частичную информацию о сообщениях, сложив шифрограммы:

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2.$$

Несмотря на проблемы, связанные с распределением ключей, одноразовый шифр-блокнот применялся как в минувших войнах, так и в дипломатической почте.

4.3. Энтропия

Криптосистема, требующая ключа той же длины, что и сообщение, вместе с условием однократного использования каждого ключа, крайне неудобна при интенсивной эксплуатации, например для осуществления сделок через Интернет. Происходит это потому, что

передача секретного ключа в рамках одной криптосистемы от одного пользователя к другому становится неразрешимой задачей. Действительно, скрытая передача необходимого ключа требует очередного ключа, тот – следующего, и т. д. Возникающую проблему называют проблемой распределения ключей.

В целях преодоления указанной трудности абсолютно стойкие криптографические алгоритмы заменяются системами, будем надеяться, вычислительно защищенными. В этом состоят задачи современных криптографов, одна из которых — изобретение системы, удовлетворяющей следующим требованиям:

- один ключ используется несколько раз,
- небольшим ключом шифруются длинные сообщения.

Такие схемы не будут совершенными ввиду теоремы Шеннона. В лучшем случае они будут вычислительно защищены.

Нам потребуются некоторые разделы теории информации, касающиеся вычислительно стойких криптосистем. Основные результаты здесь вновь принадлежат Шеннону и относятся к концу 40-ых годов прошлого века. В частности, мы будем эксплуатировать идею Шеннона об *энтропии*[1] как о способе измерения количества информации.

«Энтропия»— синоним слова «неопределенность». Основной принцип теории информации состоит в таком наблюдении: информация, по существу, то же самое, что и неопределенность. На первый взгляд, такое отождествление довольно странно, но специалисты приводят в его пользу следующий аргумент: если Вы сомневаетесь в значении чего-либо, то его уточнение дает Вам информацию. В применении к криптографии основной принцип можно пояснить так. Предположим, что Вы хотите извлечь информацию из шифротекста, другими словами, хотите знать, каково его истинное значение. При этом:

- Вам неясно, что означает данная криптограмма.
- Вы могли бы что-то предположить относительно соответствующего открытого текста.
- Уровень неизвестности, содержащийся в Вашем предположении, соответствует количеству информации, находящейся в шифротексте.

[1]В русскоязычных книгах по криптографии вместо термина «энтропия» употребляется термин «мера неопределенности». — *Прим. перев.*

Количество энтропии, ассоциированной со случайной переменной X, обозначается через $H(X)$. Отложив формальное определение этой величины на будущее, обратимся к простому примеру, проясняющему основные идеи этого понятия.

Предположим, что X — ответ на какой-то вопрос, т. е. *нет* или *да*. Если Вам известно, что я на любой вопрос всегда говорю *да*, то мой ответ Вам ничего не дает. Стало быть, количество информации, содержащейся в X, равно 0, т. е. $H(X) = 0$. Поскольку ответ известен заранее, он не несет в себе информации, поэтому нет и энтропии.

Если Вам заранее неизвестна моя реакция на вопрос, а я говорю *да* с той же вероятностью, что и *нет*, то своим ответом даю Вам один бит информации. В терминах энтропии это можно записать как $H(X) = 1$.

Заметим, что энтропия не зависит от длины фактического сообщения. В предыдущем случае сообщение состояло либо из двух, либо из трех букв (*да*, *нет*), а количество информации не превышало одного бита.

Теперь дадим строгое определение.

Определение 4.5. Пусть X — случайная величина, принимающая значения x_i $(1 \leqslant i \leqslant n)$ с вероятностями $p(X = x_i) = p_i$. Ее *энтропией*[2] называется величина

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

При этом считается, что $p_i \log_2 p_i = 0$ как только $p_i = 0$.

Вернемся к нашему примеру с вопросами и покажем, что интуитивное понятие об энтропии, которое там присутствует, соответствует формальному определению. Напомним, X — мой ответ *да* или *нет* на некий вопрос. Если известно, что я всегда тупо повторяю *да*, то $p_1 = 1$, $p_2 = 0$ и, согласно определению,

$$H(X) = -1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0.$$

Это означает, что мой ответ не дает Вам ни малейшей информации. Во второй ситуации, когда мои ответы *да* или *нет* равновероятны, т. е. $p_1 = p_2 = 1/2$, получаем

$$H(x) = -\frac{\log_2 \frac{1}{2}}{2} - \frac{\log_2 \frac{1}{2}}{2} = 1.$$

[2] Априорной мерой неопределенности. — *Прим. перев.*

Здесь мой ответ сообщает Вам один бит информации.

Из определения энтропии следуют несколько ее элементарных свойств.

- Энтропия всегда неотрицательна: $H(X) \geqslant 0$.
- Энтропия $H(X)$ равна нулю тогда и только тогда, когда $p_i = 1$ для некоторого i, а при $j \neq i$ имеем $p_j = 0$.
- Если $p_i = \frac{1}{n}$ для всех i, то $H(X) = \log_2 n$.

С другой точки зрения энтропия измеряет степень сжатия информации. Посылая ответы *да* или *нет* с помощью одного символа в коде ASCII, например, «Y» вместо *да* и «N» вместо *нет*, я передаю 8 битов данных, однако, фактически, только один бит информации. Таким образом, при желании посылаемые данные можно было бы сжать до 1/8 их первоначального размера. Следовательно, говоря наивно, сообщение длины n, сжатое с коэффициентом ε от исходного размера, содержит $\varepsilon \cdot n$ битов информации.

Обратимся к нашей «детской» криптосистеме из предыдущего параграфа. Напомним, там были пространства исходов

$$\mathbb{P} = \{a, b, c, d\}, \quad \mathbb{K} = \{k_1, k_2, k_3\} \quad \text{и} \quad \mathbb{C} = \{1, 2, 3, 4\}$$

с распределением вероятностей

$$p(p = a) = 0,25,\ p(P = b) = p(P = d) = 0,3,\ p(P = c) = 0,15,$$
$$p(K = k_1) = p(K = k_3) = 0,25,\ p(K = k_2) = 0,5,$$
$$p(C = 1) = p(C = 2) = p(C = 3) = 0,2625,\ p(C = 4) = 0,2125.$$

Соответствующие энтропии имеют значения

$$H(P) \approx 1,9527, \quad H(K) \approx 1,9944, \quad H(C) \approx 1,5. \tag{4.3}$$

Следовательно, шифротекст в нашем примере «дает утечку» в 1,5 бита информации о ключе и соответствующем открытом тексте, поскольку ровно столько неопределенности содержится в отдельной шифрограмме. Позже мы выясним, какая доля из этой «утечки» приходится на ключ, а какая — на открытый текст.

Хотелось бы получить какую-нибудь оценку сверху для энтропии случайной величины, поскольку нижняя ее граница нам известна: $H(X) \geqslant 0$. Для этого нам потребуется следующий частный случай неравенства Йенсена

Теорема 4.6. (Неравенство Йенсена.) Предположим, что

$$\sum_{i=1}^{n} a_i = 1,$$

где $a_i > 0$ при $1 \leqslant i \leqslant n$. Тогда для положительных чисел $x_i > 0$ имеет место неравенство

$$\sum_{i=1}^{n} a_i \log_2 x_i \leqslant \log_2 \left(\sum_{i=1}^{n} a_i x_i \right),$$

равенство в котором достигается тогда и только тогда, когда $x_1 = x_2 = \cdots = x_n$.

Опираясь на этот факт, докажем следующую теорему.

Теорема 4.7. Энтропия случайной величины X, принимающей n различных значений, удовлетворяет неравенству

$$0 \leqslant H(X) \leqslant \log_2 n.$$

Нижняя граница неравенства достигается на случайной величине, принимающей одно из своих значений с вероятностью 1, а верхняя — на равномерно распределенной, т. е. принимающей все свои значения равновероятно.

Доказательство. Мы уже обсудили все, касающееся нижней границы энтропии, так что сконцентрируем свои усилия на верхней. Пусть X — случайная величина с распределением вероятностей p_1, $\ldots$, p_n, причем все $p_i > 0$. Тогда

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} \leqslant$$

$$\leqslant \log_2 \left(\sum_{i=1}^{n} \left(p_i \cdot \frac{1}{p_i} \right) \right) = \log_2 n.$$

Неравенство в этих преобразованиях следует из теоремы 4.6. В силу неравенства Йенсена, знак «$\leqslant$» меняется на «=» тогда и только тогда, когда все $p_i = \frac{1}{n}$, т. е. случайная величина X распределена равномерно. ■

Основы теории энтропии близки к основам теории вероятностей. Например, для двух случайных величин в теории вероятностей определяется совместное распределение:

$$r_{i,j} = p(X = x_i \text{ и } Y = y_i)$$

при $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$. По нему можно вычислить их совместную энтропию

$$H(X, Y) = -\sum_{i=1}^{n} \sum_{j=1}^{m} r_{i,j} \log_2 r_{i,j}.$$

Совместную энтропию $H(X,Y)$ можно понимать как количество информации, которое получается из наблюдения пары значений (x,y) этих случайных величин.

Имеет место неравенство

$$H(X,Y) \leqslant H(X) + H(Y),$$

превращающееся в равенство только при независимых X и Y. Оставим его доказательство читателю в качестве упражнения.

Точно так же, как совместное распределение в теории вероятностей ассоциировано с условной вероятностью, понятие совместной энтропии связано с понятием энтропии условной[3]. Последнее — важнейший инструмент исследования неидеальных шифров, описанных в конце главы.

Пусть X и Y — случайные величины. Напомним, что $p(X=x|Y=y)$ показывает вероятность равенства $X = x$ при условии $Y = y$. Предположим, нам известно, что значение случайной величины Y равно y. В этом случае энтропия случайной величины X определяется формулой:

$$H(X|y) = -\sum_x p(X = x|Y = y) \log_2 p(X = x|Y = y).$$

Используя это, определим условную энтропию случайной величины X при данной величине Y как

$$\begin{aligned} H(X|Y) = & -\sum_y p(Y = y) H(X|y) = \\ & = -\sum_x \sum_y p(Y = y) p(X = x|Y = y) \log_2 p(X = x|Y = y). \end{aligned}$$

Это та неопределенность в отношении величины X, которая остается после раскрытия значения Y. Условная и совместная энтропии связаны соотношением:

$$H(X,Y) = H(Y) + H(X|Y),$$

откуда получается оценка сверху

$$H(X|Y) \leqslant H(X),$$

где равенство достигается только в случае независимых случайных величин X и Y. Доказательство этих формул мы также оставляем читателю в качестве упражнения.

[3] Апостериорная мера неопределенности. — *Прим. перев.*

Возвращаясь к криптографии, выпишем несколько очевидных утверждений, касающихся энтропии величин P, K и C.

- $H(P|K,C) = 0$: при известных шифротексте и ключе мы полностью знаем соответствующий открытый текст. Этот факт, очевидно, верен, поскольку в противном случае расшифрование будет работать неправильно.
- $H(C|P,K) = 0$: если мы знаем открытый текст и ключ, то можем получить шифротекст. Это высказывание справедливо для всех шифров, с которыми мы уже знакомы, и остается в силе для симметричных криптосистем, описанных в следующих главах. Однако в современных схемах шифрования с открытым ключом, при условии правильного их применения, указанное свойство нарушается.

Кроме того, имеют место следующие тождества:

$$\begin{aligned} & H(K,P,C) = \\ &= H(P,K) + H(C|P,K) = && (\text{т. к. } H(X,Y) = H(Y) + H(X|Y)) \\ &= H(P,K) = && (\text{т. к. } H(C|P,K) = 0) \\ &= H(K) + H(P) && (\text{т. к. } K \text{ и } P \text{ независимы}); \end{aligned}$$

$$\begin{aligned} & H(K,P,C) = \\ &= H(K,C) + H(P|K,C) = && (\text{т. к. } H(X,Y) = H(Y) + H(X|Y)) \\ &= H(K,C) && (\text{т. к. } H(P|K,C) = 0). \end{aligned}$$

Следовательно,

$$H(K,C) = H(K) + H(P).$$

Последняя формула имеет особое значение, поскольку она относится к условной энтропии $H(K|C)$, называемой *неопределенностью ключа*. Величина $H(K|C)$ — это то количество неопределенности относительно истинного ключа, которое остается после прочтения шифротекста. Напомним, что наша цель — определение ключа по данной шифрограмме. Совмещая два предыдущих равенства, находим

$$H(K|C) = H(K,C) - H(C) = H(K) + H(P) - H(C). \qquad (4.4)$$

Иначе говоря, неопределенность в знании ключа при известном шифротексте равна сумме неопределенностей относительно открытого текста и ключа за вычетом неопределенности в отношении шифротекста.

На стр. 109 мы уже вычислили энтропию величин P, K и C для «игрушечной» криптосистемы (соотношение (4.3)). Подставив эти

числа в формулу (4.4), получим

$$H(K|C) \approx 1,9527 + 1,9944 - 1,5 \approx 1,4583.$$

Таким образом, после просмотра отдельного шифротекста нам остается найти около полутора бит информации об истинном ключе шифровки. Этот пример объясняет, как криптосистема допускает утечку информации, и показывает, почему ее нельзя считать стойкой. Как-никак, вначале у нас есть 1.9944 бита неопределенности относительно ключа, а знание одной шифровки уменьшает неопределенность до 1.4593 бита. То есть отдельный шифротекст сообщает о ключе $1.9944 - 1.4593 \approx 0.535$ бита информации.

4.4. Ложные ключи и расстояние единственности

Отдельный шифротекст выдает ненулевое количество информации об истинном ключе криптосистемы, поскольку он исключает некоторое подмножество неподходящих ключей. Среди всех оставшихся лишь один правилен. Для возможных, но отличных от истинного ключей вводится термин *ложный ключ*.

Рассмотрим (немодифицированный) шифр сдвига, т. е. шифр, в котором один ключ используется для замены любой буквы. Возьмем шифрограмму WNAJW и предположим, что язык открытого текста — английский. Тогда есть только два разумных кандидата на открытый текст: *river* и *arena*, соответствующие двум возможным ключам: «*f*» и «*w*». Один из них истинный, а второй — ложный.

Обоснуем в очередной раз слабую криптостойкость шифра замены, но на сей раз опираясь на понятие *расстояния единственности*. Мы подробно расскажем об этом термине несколько позже, а сейчас более четко представим себе, что такое открытый текст. Во многих компьютерных сетях открытый текст можно рассматривать как случайную строку битов. Но довольно часто бывают исключения. В одних случаях шифруется обычный текст, а в других — некий его образ. В наших обсуждениях мы рассматриваем ситуацию, когда исходный открытый текст взят из английского языка, как в шифре замены. Такой язык обычно называют *естественным*, дабы отличать его от строк битов, используемых в компьютерных сетях.

Сначала найдем энтропию (количество информации) H_L, которую несет одна буква естественного языка (в нашей ситуации — английского). Отметим, энтропия случайной последовательности английских букв равна

$$\log_2 26 \approx 4,70.$$

Поэтому справедлива оценка $H_L \leqslant 4,70$. Случайная величина P, принимающая значения букв в английском тексте, имеет распределение

$$p(P = a) = 0,082, \ldots, p(P = e) = 0,127, \ldots, p(P = z) = 0,001$$

(сравните с табл. 3.1 на стр. 72). По стандартной формуле получаем

$$H_L \leqslant H(P) \approx 4,14.$$

Следовательно, учитывая среднестатистическую частоту букв в английском тексте, можно утверждать, что на каждую букву приходится 4,14 бит информации, а не 4,7, как мы предполагали после грубой оценки.

Однако и это приближение далеко от истинного значения, поскольку буквы нельзя считать независимыми. Так, например, за «q» всегда следует «u», а биграмма «th» наиболее популярна в английском языке. Возникает гипотеза, что для лучшего приближения к истинному значению энтропии H_L стоит изучить распределение биграмм. Итак, пусть P^2 — случайная биграмма. Обозначив через $p(P = i, P' = j)$ вероятность появления биграммы «ij», найдем

$$H(P^2) = -\sum_{i,j} p(P = i, P' = j) \log_2 p(P = i, P' = j).$$

Эта величина вычислялась неоднократно разными авторами. Обычно ее считают равной

$$H(P^2) \approx 7.12$$

Отсюда энтропия отдельной буквы оценивается как

$$H_L \leqslant \frac{H(P^2)}{2} \approx 3,56.$$

И опять нас не может удовлетворить такое ограничение, т. к. при его вычислении мы не учитывали наиболее употребительных триграмм. Значит, для ее улучшения надо исследовать совместное распределение трех букв P^3 и подсчитать $H(P^3)/3$, что увеличит качество оценки, но не до конца. Этот процесс последовательного совершенствования подводит нас к следующему определению.

Определение 4.8. Энтропия естественного языка L определяется по формуле

$$H_L = \lim_{n \to \infty} \frac{H(P^n)}{n}.$$

Точное значение этого предела найти крайне сложно, но его можно аппроксимировать. Фактически пользуются экспериментальной

оценкой, которая в случае английского языка имеет вид

$$1,0 \leqslant H_L \leqslant 1,5.$$

Так что каждая буква

- требует 5 битов для представления в компьютере,
- но содержит в себе не более 1,5 битов информации.

Это свидетельствует о том, что естественный язык имеет высокую степень избыточности, в чем легко убедиться на следующем предложении, из которого все еще можно извлечь заложенный в нем смысл, хотя там и удалены по две из каждых четырех букв[1].

*On** ur** a t**e t**re **s a **rl **al**d S**w W**te.*

Формально *избыточность* языка вычисляется по правилу

$$R_L = 1 - \frac{H_L}{\log_2 \#\mathbb{P}}.$$

Положив $H_L \approx 1,25$, получим, что избыточность английского языка равна

$$\mathbb{R}_L \approx -\frac{1,25}{\log_2 26} = 0,75.$$

Таким образом, теоретически мы можем сжать десятимегабайтный файл с английским текстом до 2,5 MB

Вернемся к общему шифру и предположим, что $c \in \mathbb{C}^n$, т. е. c — шифротекст, состоящий из n знаков. Определим $\mathbb{K}(c)$ как множество ключей, с помощью которых из шифрограммы c получаются поддающиеся интерпретации тексты. Тогда $\#\mathbb{K}(c) - 1$ — число ложных ключей для данного c. Среднее число ложных ключей обозначается символом $\overline{s}_n$. Оно равно

$$\begin{aligned}
\overline{s}_n &= \sum_{c \in \mathbb{C}^n} p(C = c)(\#\mathbb{K}(c) - 1) = \\
&= \sum_{c \in \mathbb{C}^n} p(C = c)\#\mathbb{K}(c) - \sum_{c \in \mathbb{C}^n} p(C = c) = \\
&= \left(\sum_{c \in \mathbb{C}^n} p(C = c)\#\mathbb{K}(c) \right) - 1.
\end{aligned}$$

[1] Поскольку наивно предполагать совершенное знание английского у русскоязычного читателя, его вниманию предлагается следующая известная строчка: «Бу** мг**ю **бо к**ет, **хри **ежн** кру**». — *Прим. перев.*

Для достаточно большого n при условии $\#\mathbb{P} = \#\mathbb{C}$ получаем

$$\begin{aligned}
\log_2(\overline{s}_n + 1) = \log_2 \sum_{c\in\mathbb{C}^n} p(C=c)\#\mathbb{K}(c) \geqslant \\
\geqslant \sum_{c\in\mathbb{C}^n} p(C=c)\log_2 \#\mathbb{K}(c) \geqslant && \text{(неравенство Йенсена)} \\
\geqslant \sum_{c\in\mathbb{C}^n} p(C=c)H(K|c) = \\
= H(K|C^n) = && \text{(по определению)} \\
= H(K) + H(P^n) - H(C^n) \approx && \text{(формула (4.4))} \\
\approx H(K) + nH_L - H(C^n) = && \text{(т. к. } n \gg 1\text{)} \\
= H(K) - H(C^n) + \\
\quad + n(1-R_L)\log_2 \#\mathbb{P} \geqslant && \text{(по определению } R_L\text{)} \\
\geqslant H(K) - n\log_2 \#\mathbb{C} + \\
\quad + n(1-R_L)\log_2 \#\mathbb{P} = && \text{(т. к. } H(C^n) \leqslant n\log_2 \#\mathbb{C}\text{)} \\
= H(K) - nR_L \log_2 \#\mathbb{P} && \text{(т. к. } \#\mathbb{P} = \#\mathbb{C}\text{).}
\end{aligned}$$

Итак, при больших n и $\#\mathbb{P} = \#\mathbb{C}$ имеет место оценка:

$$\overline{s}_n \geqslant \frac{\#\mathbb{K}}{\#\mathbb{P}^{nR_L}} - 1.$$

При атаке на шифр нам хотелось бы понизить число ложных ключей до нуля. Ясно, что с ростом длины шифротекста число ложных ключей уменьшается. *Расстоянием единственности* шифра называют такую длину n_0 шифротекста, начиная с которой число ложных ключей становится равным нулю. Другими словами, это средний объем шифротекста, необходимый для точного вскрытия ключа атакующим, имеющим неограниченные вычислительные ресурсы. Для абсолютно стойких криптосистем $n_0 = \infty$, но для других — расстояние единственности может оказаться опасно малым. Оценим n_0, положив в предыдущей формуле $\overline{s}_n = 0$:

$$n_0 \approx \frac{\log_2 \#\mathbb{K}}{R_L \log_2 \#\mathbb{P}}.$$

Для шифра замены имеем

$$\#\mathbb{P} = 26, \qquad \#\mathbb{K} = 26! \approx 4 \cdot 10^{26},$$

и используя значение $R_L = 0,75$ для английского языка, получаем, что расстояние единственности приблизительно равно

$$n_0 \approx \frac{88,4}{0,75 \cdot 4,7} \approx 25.$$

Таким образом, теоретически нам достаточно шифротекста из 25 букв для взлома шифра замены, если у нас неограниченные вычислительные возможности. Во всяком случае мы ожидаем, что шифротекст, содержащий свыше 25 букв, расшифровывается однозначно.

Предположим теперь, что у нас есть современный шифр, который шифрует строку битов с помощью ключа, состоящего из l битов, тогда

$$\#\mathbb{P} = 2, \qquad \#\mathbb{K} = 2^l.$$

Вновь считая, что $R_L = 0,75$ (что не совсем верно, поскольку теперь нам нужно какое-нибудь электронное представление языка, например, код ASCII), находим расстояние единственности

$$n_0 \approx \frac{l}{0,75} = \frac{4l}{3}.$$

Теперь рассмотрим ситуацию, когда перед передачей текст в формате ASCII сжимается. Если при этом пользоваться идеально архивирующим алгоритмом, то сжатый текст уже не будет обладать избыточностью, т. е. значение $\mathbb{R}_l$ станет очень близким к 0. В этом случае расстояние единственности будет выглядеть как

$$n_0 \approx \frac{l}{0} = \infty.$$

Возникает естественный вопрос: могут ли современные криптосистемы зашифровать текст с нулевой избыточностью? Ответ — нет: даже если современный шифр и сжимает данные, для увеличения криптостойкости он добавляет некоторую избыточную информацию к открытому тексту перед шифрованием.

Мы только что исследовали возможности так называемой пассивной атаки, т. е. атаки на шифр, при которой атакующий может исследовать только шифротекст и по нему должен восстановить секретный ключ. Существуют атаки и другого типа. Их называют активными. Следуя тактике активной атаки, нападающий генерирует открытый текст или шифрограмму по своему выбору и подсовывает свой продукт законным пользователям для шифрования или расшифрования, в зависимости от ситуации. Первая разновидность активной атаки называется атакой с выбором открытого текста, а вторая — с выбором шифротекста.

В криптосистемах с открытым ключом, с которыми мы познакомимся дальше, невозможно препятствовать атаке с выбором открытого текста, поскольку там любому позволено шифровать все что угодно. Поэтому там следует ставить преграды перед атаками

с выбором шифротекста. Накопленный опыт эксплуатации криптосистем с открытым ключом говорит о пользе добавления некоторой избыточной информации в открытый текст перед его шифрованием. Это сильно затрудняет нападающему задачу по созданию шифротекста, который можно было бы удовлетворительно расшифровать. Мы обсудим это в следующих главах.

Краткое содержание главы

- Криптосистемы, в которых знание шифротекста ничего не добавляет к той информации, которой атакующий обладал до его просмотра, называются абсолютно стойкими.
- Абсолютно стойкие системы существуют, но требуют ключей той же длины, что и само сообщение, причем новых для каждого следующего послания. Поэтому абсолютно стойкие системы ценны только с теоретической точки зрения и малоприменимы на практике.
- «Информация» и «неопределенность», по существу, родственные понятия. Нападающий фактически стремится к извлечению информации об открытом тексте по его шифрованному варианту. Количество неопределенности в случайной величине измеряется ее энтропией.
- Соотношение $H(K|C) = H(K) + H(P) - H(C)$ позволяет нам оценить величину неопределенности ключа после прочтения шифротекста.
- Избыточность естественного языка приводит к тому, что для взлома наивных шифров не требуется слишком длинного шифротекста.

Дополнительная литература

Наше обсуждение теории Шеннона близко следовало книге Стинсона. Другой возможный источник информации об этой теории — книга Уэлша. Общее введение в теорию информации, включая ее приложения к теории шифрования, изложено в книге ван дер Люббе.

J.C.A. van der Lubbe. *Information Theory*. Cambridge University Press, 1997.

D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

D. Welsh. *Codes and Cryptography*. Oxford University Press, 1988.

Контрольные вопросы

4.1.1. Найдите ошибку в таблице

	a	b	c	d
k_1	W	X	Y	Z
k_2	Z	Y	Z	X
k_3	X	Z	W	Y
k_4	Y	W	X	Z

,

задающей шифрующую функцию, определенную на множестве открытых текстов $\mathbb{P} = \{a, b, c, d\}$ со значениями в шифрограммах $\mathbb{C} = \{W, X, Y, Z\}$, которая использует ключи $\mathbb{K} = \{k_1, k_2, k_3, k_4\}$.

4.1.2. Какая из величин более интересна для атакующего:

$$p(C = c|P = m) \quad \text{или} \quad p(P = m|C = c),$$

где m — открытый текст, а c — его шифрованная версия? Объясните свой ответ.

4.1.3. Какими свойствами должен обладать шифротекст, который обладает теоретико-информационной стойкостью?

4.1.4. Вспомните определение энтропии случайной величины X с распределением вероятностей $p_i = p(X = x_i)$.

4.1.5. Пусть случайная величина X принимает не более t значений. Назовите минимальное и максимальное значение энтропии $H(X)$.

4.1.6. Определите, какое из соотношений справедливы для любых шифров:

а) $H(P|K, C) = 0$,

б) $H(K, P) = H(K) + H(P)$.

Аргументируйте свой ответ.

4.1.7. Дайте определение терминов «ложный ключ» и «расстояние единственности».

4.1.8. Известно, что расстояния единственности шифров C_1 и C_2 равны, соответственно, n_1 и n_2. Какой из них предпочтительней использовать, если $n_1 > n_2$?

Лабораторные работы

4.2.1. Напишите программу, вычисляющую значения $H(P^n)$ при разных n для части введенного текста. Воспользуйтесь Вашей программой для оценки энтропии H_L и избыточности R_L естественного языка.

Упражнения

Несколько первых задач относятся к «игрушечной» криптосистеме, в которой $\mathbb{P} = \{a, b, c, d\}$, $\mathbb{C} = \{W, X, Y, Z\}$ и $\mathbb{K} = \{k_1, k_2, k_3, k_4\}$, а функция шифрования задана таблицей

	a	b	c	d
k_1	W	X	Y	Z
k_2	X	Z	W	Y
k_3	Z	W	X	Y
k_4	Y	W	Z	X

Считайте, что все сообщения и ключи равновероятны.

4.3.1. Вычислите вероятности $p(C = W)$, $p(C = X)$, $p(C = y)$ и $p(C = Z)$.

4.3.2. Вычислите условные вероятности $p(P = m|C = c)$ для всех пар (m, c).

4.3.3. Оцените с помощью подсчитанных условных вероятностей, насколько хороша эта криптосистема.

4.3.4. Вычислите энтропии $H(K)$, $H(P)$, $H(C)$ и $H(K|C)$. Подтверждают ли найденные значения Ваши оценки?

4.3.5. Исследуйте докомпьютерные шифры из главы 3 с помощью техники главы 4. Вычислите приближенное значение расстояния единственности для шифров Цезаря, Виженера и шифра замены и примените полученные результаты к объяснению их незащищенности от взлома.

4.3.6. Докажите неравенство Йенсена.

4.3.7. Докажите, что неравенство

$$H(X, Y) \leqslant H(X) + H(Y)$$

справедливо для любой пары случайных величин X и Y. Кроме того, покажите, что равенство достигается тогда и только тогда, когда они независимы.

4.3.8. Покажите, что энтропии случайных величин удовлетворяют соотношению

$$H(X,Y) = H(Y) + H(X|Y).$$

4.3.9. Проверьте истинность оценки

$$H(X|Y) \leqslant H(X).$$

Покажите, неравенство превращается в равенство тогда и только тогда, когда X и Y независимы.

4.3.10. Покажите, что если шифр с каким-то распределением вероятностей открытых текстов является абсолютно стойким, то он остается таковым и при любом их распределении.

4.3.11. Докажите, что шифротексты абсолютно стойкой криптосистемы с условием $\#\mathbb{P} = \#\mathbb{K} = \#\mathbb{C}$ распределены равномерно.

4.3.12. Пусть X и Y — случайные величины, отражающие результат бросания двух костей. Обозначим через $Z = X + Y$ их сумму. Проверьте, что

$$H(Z) < H(X,Y).$$

4.3.13. После удаления из высказывания всех гласных букв и промежутков между словами получилось следующее:

WHTWLDYLKFRLNCHTDY.

Восстановите сообщение.

ГЛАВА 5

СИММЕТРИЧНЫЕ ШИФРЫ

Цели главы

- Понять основные принципы современных симметричных алгоритмов.
- Разобраться в работе алгоритма *DES*.
- Понять, как работает алгоритм *Rijndael*.
- Узнать о стандартных режимах работы блочных шифров.
- Очертить область поточных шифров и регистров сдвига с линейной обратной связью (*РСЛОС*).

5.1. Введение

Работа симметричных шифров включает в себя два преобразования:

$$C = E_k(m) \quad \text{и} \quad m = D_k(C),$$

где m — открытый текст, E — шифрующая функция, D — расшифровывающая функция, k — секретный ключ, C — шифротекст. Следует отметить, что как шифрующая, так и расшифровывающая функции общеизвестны, и тайна сообщения при известном шифротексте зависит только от секретности ключа k. Хотя этот хорошо обоснованный принцип, называемый *принципом Керкхоффа*, был известен еще начиная с середины 1800-ых годов, множество компаний все еще игнорирует его. Существует много отдельных фирм, разрабатывающих собственные секретные схемы шифрования, которые теряют свою стойкость, как только кто-то из персонала допускает утечку информации о деталях алгоритмов. Опыт показывает, что наилучшими схемами шифрования являются те, которые изучались в течение длительного времени большой армией исследователей и рекомендованы к применению как наиболее безопасные. Схема, из которой создают коммерческую тайну, не может изучаться никем, кто не работает на соответствующую компанию.

Алгоритм, символическая запись которого приведена в начале главы, называется криптосистемой с симметричным ключом, поскольку обе стороны, обменивающиеся шифрованной информацией, применяют один и тот же секретный ключ. Иногда симметричные криптосистемы используют два ключа: один для шифрования, а другой для обратного процесса. В этом случае мы будем предполагать, что шифрующий ключ легко восстанавливается по расшифровывающему и наоборот.

Позже мы встретимся с криптосистемами с открытым ключом. В них только один ключ хранится в тайне и называется секретным, в то время как второй, открытый ключ, доступен всем желающим. При этом считается невозможным для кого бы то ни было вычислить секретный ключ, опираясь на информацию об открытом.

Возвращаясь к симметричной криптографии, отметим, что число возможных ключей должно быть очень велико. Это требование возникает в связи с тем, что при проектировании криптоалгоритма мы обязаны учитывать самый плохой сценарий развития событий, ставя гипотетического противника в максимально выгодное положение, т. е. мы считаем, что атакующий

- обладает полнотой информации о шифрующем (расшифровывающем) алгоритме,
- имеет в своем распоряжении некоторое количество пар (открытый текст, шифротекст) ассоциированных с истинным ключом k.

Если количество возможных ключей мало́, то атакующий имеет возможность взломать шифр простым перебором вариантов. Он может шифровать один из данных открытых текстов, последовательно используя разные ключи, до тех пор, пока не получит соответствующий известный шифротекст. В результате искомый ключ будет найден. Именно для исключения такого сорта атаки необходимо предусмотреть достаточно большое пространство ключей. Принято считать, что вычисления, состоящие из 2^{80} шагов, в ближайшие несколько лет будут неосуществимы. Поэтому ключ, исключающий взлом простым перебором, должен насчитывать по крайней мере 80 битов.

Хороший разработчик шифра в совершенстве владеет двумя специальностями: он должен взламывать криптосистемы так же блестяще, как и изобретать их. В наши дни, несмотря на многочисленные попытки, все еще не найден метод аналитического доказательства стойкости симметричных криптосистем, в отличие от алгоритмов с открытым ключом, где криптостойкость шифров доказывается цепочкой логических рассуждений, начинающейся достаточно

естественными предположениями. Однако симметричные алгоритмы все еще активно эксплуатируются, поскольку лучшие методы атак на эти шифры, разработанные ведущими криптоаналитиками, пока не достигают цели.

5.1.1. Упрощенная модель

На рис. 5.1 изображена упрощенная модель шифрования битовой строки, которая, несмотря на свою простоту, вполне подходит для практического применения. Идея модели состоит в применении к открытому тексту обратимой операции для получения шифротекста, а именно, побитовое сложение по модулю 2 открытого текста со «случайным потоком» битов. Получатель может восстановить текст с помощью обратной операции, сложив шифротекст с тем же самым случайным потоком.

Рис. 5.1. Упрощенная модель, шифрующая строку битов

Такую модель легко реализовать на практике, поскольку для ее реализации необходима одна из простейших компьютерных операций — исключающее ИЛИ, т. е. сложение по модулю 2, которое обозначается знаком «⊕». В главе 4 мы убедились, что шифруя каждое новое сообщение своим ключом, длина которого совпадает с длиной открытого текста, мы получим абсолютно стойкую симметричную криптосистему. Напомним, что такая криптосистема называется одноразовым шифр-блокнотом. Однако, несмотря на совершенство этого алгоритма, он не применяется на практике, поскольку порождает почти неразрешимую проблему распределения ключей. В связи с этим разрабатываются симметричные криптосистемы, в которых длинное сообщение шифруется коротким ключом, причем этот ключ можно использовать несколько раз. Естественно, такие системы далеки от абсолютно стойких, но, с другой стороны, распределение ключей для них — хотя и трудная, но вполне решаемая задача.

Существует несколько типов атак на основную массу шифров, некоторые из которых мы приведем ниже, разделив их на пассив-

ные и активные. Вообще говоря, пассивное нападение организовать проще, чем активное.

- *Пассивная атака.* Здесь противник лишь читает перехваченные шифрованные сообщения и пытается взломать криптосистему либо раскрыв ключ, либо узнав ту секретную информацию, утечки которой и хотели избежать законные пользователи криптосистемы. Один из стандартных приемов пассивного нападения состоит в анализе обмена сообщениями. Эта техника восходит своими корнями к первой мировой войне, когда внезапное усиление радиопереговоров на отдельном участке западного фронта свидетельствовало о неизбежном наступлении.
- *Активная атака.* В этой ситуации противник может вставлять, удалять или повторять сообщения, вклиниваясь между переговаривающимися партнерами. Обычно требуется, чтобы для проведения атаки вставки необходимо было предварительно взломать шифр, а сам шифр должен обеспечивать возможность как обнаружения атаки удалением или повторением, так и восстановления текста.

Большинство симметричных шифров можно разделить на две больших группы. Первая — поточные шифры, где за один раз обрабатывается один элемент данных (бит или буква), а вторая — блочные шифры, в которых за один шаг обрабатывается группа элементов данных (например, 64 бита).

5.1.2. Поточные шифры

Рисунок 5.2 дает простую, но яркую иллюстрацию поточного шифра. Обратите внимание, как она похожа на предыдущую упрощенную модель. Тем не менее, случайный поток битов теперь генерируется по короткому секретному ключу с помощью открытого алгоритма, называемого *генератором ключевого потока*. Здесь биты шифротекста получаются по правилу: $C_i = m_i \oplus k_i$, где m_0, m_1, ... — биты открытого текста, а k_0, k_1, ... — биты ключевого потока.

Поскольку процесс шифрования — это сложение по модулю 2, расшифрование является, по существу, той же самой операцией:

$$m_i = C_i \oplus k_i.$$

Поточные шифры, похожие на описанные выше, просты и удобны для реализации. Они позволяют очень быстро шифровать боль-

шой объем данных. Поэтому они подходят для передачи аудио- и видео-сигналов в реальном времени. Кроме того, в этом процессе не происходит накопления ошибки. Если отдельный бит шифротекста исказился в процессе передачи вследствие слабого радиосигнала или из-за вмешательства противника, то в расшифрованном открытом тексте только один бит окажется неверным. Однако повторное использование того же ключа дает тот же ключевой поток, что влечет за собой зависимость между соответствующими сообщениями. Предположим, например, что сообщения m_1 и m_2 были зашифрованы одним ключом k. Тогда противник, перехватив шифровки, легко найдет сумму по модулю 2 открытых текстов:

$$C_1 \oplus C_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2.$$

Следовательно, необходимо менять ключи либо с каждым новым сообщением, либо с очередным сеансом связи. В результате мы сталкиваемся с трудностями управления ключами и их распределения, которые преодолеваются, как мы позже увидим, с помощью криптосистем с открытым ключом. Обычно алгоритмы с открытым ключом применяются для передачи ключа, закрепленного за отдельным сообщением или за целым сеансом связи, а фактические данные шифруются после этого с помощью поточного или блочного шифров.

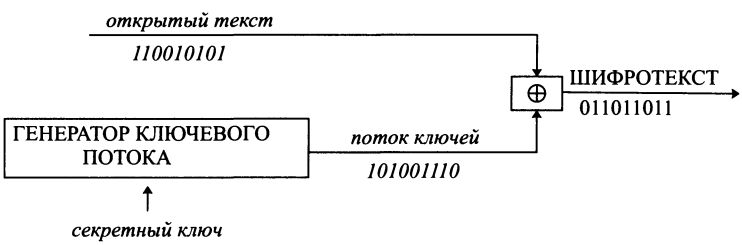


Рис. 5.2. Поточные шифры

Чтобы придать необходимую стойкость шифру, генератор ключевого потока производит строку битов с определенными свойствами. Как минимум, ключевой поток должен:

- Иметь большой период. Поскольку ключевой поток получается в результате детерминированного процесса из основного ключа, найдется такое число n, что $k_i = k_i + n$ для всех значений i. Число n называется периодом последовательности, и для обеспечения стойкости шифра выбирается достаточно большим.

- Иметь псевдо-случайные свойства. Генератор должен производить последовательность, которая кажется случайной. Другими словами, генерируемая последовательность должна выдержать определенное число статистических тестов на случайность.
- Обладать линейной сложностью. Далее в этой главе мы объясним значение этого термина.

Однако перечисленных условий не хватает, поскольку восстановление значительной части этой последовательности должно быть неосуществимым в вычислительном отношении. В идеале, даже если кто-то знает первый миллиард битов ключевой последовательности, вероятность угадать следующий бит не должна превышать 50%.

В §5.6 мы обсудим, как создаются поточные шифры с использованием комбинации простых микросхем, называемых «регистром сдвига с линейной обратной связью».

5.1.3. Блочные шифры

На рис. 5.3 изображена схема блочного алгоритма шифрования. Блочный шифр за один прием обрабатывает блок открытого текста. Основное отличие блочного шифра от поточного состоит в том, что поточным шифрам необходимо постоянно помнить о том, какое место битовой строки они в данный момент обрабатывают, чтобы определить, какую часть ключевого потока нужно сейчас генерировать; блочные же шифры избавлены от этой необходимости. Как и в случае поточных шифров, мы пишем

$$C = E_k(m) \quad \text{и} \quad m = D_k(C),$$

где m — блок открытого текста, k — секретный ключ, E — шифрующая функция, D — расшифровывающая функция, C — блок шифротекста.

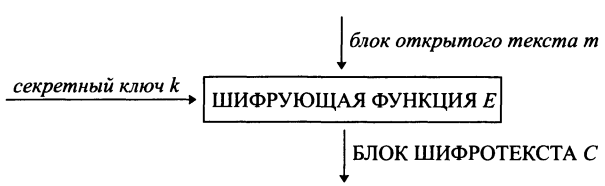


Рис. 5.3. Схема работы блочного шифра

Размер блока для шифрования обычно выбирают разумно большим. В системе *DES* (стандарт шифрования данных), например,

он состоит из 64 битов, а в современных блочных криптосистемах он достигает 128 битов и более. Часто зашифрованный первый блок сообщения используют для шифрования следующего. Такой прием обычно называют *режимом шифрования*. Режимы используются, чтобы избежать некоторых атак, основанных на стирании или вставке, придавая каждому блоку шифротекста контекст, присущий всему сообщению. Каждый режим шифрования предполагает свою защиту от накопления ошибок из-за сбоев передачи шифротекста. Кроме того, в зависимости от режима работы (и приложений) выбирается ключ сообщения или сеанса связи. Например, многие режимы шифрования требуют некоего начального значения, вводимого перед операциями шифрования и расшифрования. Позже в этой главе мы обсудим режимы блочных шифров более детально.

Сегодня принято на вооружение довольно много разновидностей блочных шифров, некоторые из которых с большой долей вероятности используются Вашим web-браузером: *RC5, RC6, DES* или *3DES*. Наиболее знаменитый из них — *DES*, т. е. стандарт шифрования данных. Впервые он был опубликован в середине семидесятых годов XX века как федеральный стандарт США и вскоре оказался, дефакто, международным стандартом в банковских операциях. *DES* успешно выдержал испытание временем, но к началу 90-ых годов назрела необходимость в разработке новых стандартов. Произошло это потому, что как длина блока (64 бита), так и размер ключа (56 битов) оригинального алгоритма *DES* оказались недостаточными для обеспечения секретности сообщений. В настоящее время можно восстановить 56-битовый ключ системы *DES*, используя либо сеть компьютеров, либо специализированные аппаратные средства ЭВМ. В ответ на эту проблему национальный институт стандартов и технологий США (NIST) положил начало своего рода соревнованию по поиску нового блочного шифра, достойного названия «новый стандарт шифрования» (*AES*).

В отличие от фактически засекреченных работ над проектированием *DES*, проект *AES* осуществлялся публично. Множество исследовательских групп всего мира представили свои варианты *AES* на конкурс. В финал вышли пять алгоритмов, которые изучались глубже с целью выбора победителя. Это были криптосистемы:

- *MARS* от группы при компании *IBM*;
- *RC6*, представленная компанией *RSA Security*;
- *Twofish* от группы базирующейся в Коунтерпэйне, Беркли и других местах;

- *Serpent* от группы трех ученых, работающих в Израиле, Норвегии и Британии;
- *Rijndael* от двух бельгийских криптографов.

Наконец, в конце 2000 г. NIST объявил, что победителем конкурса был выбран шифр *Rijndael*.

Криптосистема *DES* и все финалисты проекта *AES* — примеры *итерированного* блочного шифра. В таких шифрах стойкость обеспечивается повторяющимся использованием простой *раундовой функции*, преобразующей n-битовые блоки в n-битовые блоки, где n — размер блока шифра. Число раундов S может меняться или быть фиксированным. Как правило, с увеличением числа раундов уровень стойкости блочного шифра повышается.

При каждом применении раундовой функции используется подключ

$$k_i \quad \text{при } 1 \leqslant i \leqslant S,$$

выводящийся из основного секретного ключа k с помощью алгоритма *разворачивания ключа*. Чтобы шифротекст можно было успешно расшифровать, функция, генерирующая подключи, должна быть обратимой. При расшифровании подключи применяются в порядке, обратном тому, в котором они использовались при шифровании. Требование обратимости каждого раунда не подразумевает обратимости функций, в нем участвующих. На первый взгляд это кажется странным, но станет совершенно очевидным после подробного обсуждения криптосистемы *DES*. Функции, которые в ней используются, необратимы, но, тем не менее, каждый раунд обратим. В то же время, в схеме *Rijndael* обратимы не только раунды, но и все функции.

Существует несколько общих методик, которые можно применять при взломе блочного шифра, например, полный перебор, опирающийся на предварительно вычисленные таблицы промежуточных параметров, или прием, условно называемый «разделяй и властвуй». Некоторые (неудачно спроектированные) блочные шифры могут оказаться беззащитными перед атакой с выбором открытого текста, где шифрование специально выбранного сообщения может выявить важные свойства секретного ключа. Криптоаналитики, как правило, сочетают математические приемы взлома с навыками разгадывания головоломок, да и толика удачи не помешает. Разработано небольшое число довольно успешных методов нападений, некоторые из которых применимы вообще к любому шифру (а не только к блочному).

- *Дифференциальный криптоанализ.* В дифференциальном криптоанализе изучаются пары шифротекстов, исходные сообщения в которых имеют специфические различия. Результат применения логической операции исключающего ИЛИ к таким парам называется дифференциалом. Определенные дифференциалы обладают характерными свойствами, зависящими от использованного ключа. Исследуя вероятности дифференциалов, вычисленных при атаке с выбором открытого текста, можно надеяться на выявление основной структуры ключа.
- *Линейный криптоанализ.* Несмотря на то, что хороший блочный шифр содержит нелинейные компоненты, идея, на которой основан линейный анализ, состоит в аппроксимации нелинейных компонент линейными функциями. А цель его все та же — опираясь на распределение вероятностей, выудить полезную информацию о ключе.

Удивительно, но эти методы весьма эффективно взламывают некоторые шифры. Однако они пасуют перед *DES* и *Rijndael*, двумя самыми важными блочными шифрами современности.

Поскольку *DES* и *Rijndael*, вероятно, не потеряют свою значимость в течение нескольких ближайших лет, мы изучим их более подробно, чем остальные. Их важность с точки зрения обучения обусловлена еще и тем, что на их примере можно четко увидеть основные принципы конструирования шифров: замены и перестановки. Напомним, что докомпьютерные шифры тоже включают в себя эти операции, так что с тех пор мало что изменилось. Сейчас, однако, применяют более запутанные перестановки и замены. Сами по себе они не обеспечивают стойкости шифра, но при их использовании на протяжении нескольких раундов можно получить достаточную для практических целей криптостойкость.

Закончим этот параграф обсуждением вопроса, какой из шифров лучше — блочный или поточный? Увы, корректного ответа не существует. Они оба используются и обладают разными свойствами.

- Блочный шифр является более общим, и мы увидим, что его легко трансформировать в поточный.
- Поточный шифр имеет более математизированную структуру, что с одной стороны дает больше возможностей для его взлома, но с другой — позволяет легче изучать и строго оценивать его стойкость.
- Общие поточные шифры не очень удобны с точки зрения программного обеспечения, так как они обычно шифруют один

бит за прием. Однако они высоко эффективны с точки зрения аппаратной реализации.

- Блочные шифры удобны как для программных, так и для аппаратных средств, но они не допускают такой высокой скорости обработки информации, как поточные.
- Аппаратные средства функционируют быстрее, чем программное обеспечение, но этот выигрыш происходит за счет снижения гибкости.

5.2. Шифр Фейстеля и DES

Шифр *DES* — вариант базисного шифра Фейстеля (см. рис. 5.4), названного по имени Г. Фейстеля, работавшего в фирме *IBM* и выполнившего некоторые из самых ранних невоенных исследований в области алгоритмов шифрования. Интересная особенность шифра Фейстеля заключается в том, что функция раунда обратима вне зависимости от свойств функции F (см. рис. 5.4). Чтобы в этом убедиться, обратимся к формулам. Каждый раунд шифрования осуществляется по правилу

$$l_i = r_{i-1}, \qquad r_i = l_{i-1} \oplus F(k_i, r_{i-1}).$$

Следовательно, расшифрование происходит за счет преобразований:

$$r_{i-1} = l_i, \qquad l_{i-1} = r_i \oplus F(k_i, l_i).$$

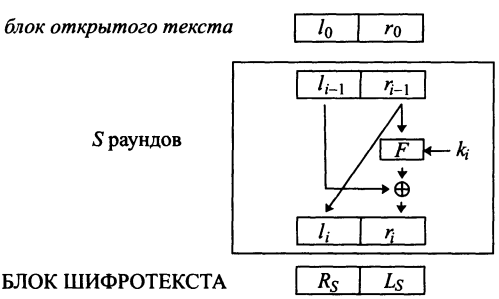


Рис. 5.4. Основные операции шифра Фейстеля

Выписанные формулы показывают, что мы несколько упростили первоначальный проект, поскольку

- в качестве F можно выбрать любую функцию и получить шифрующую функцию, которая будет обращаться при помощи секретного ключа;

– одну и ту же микросхему можно использовать как для шифрования, так и для расшифрования. Нам необходимо лишь проследить за порядками подключей, которые в этих процессах обратны друг другу.

Конечно, для создания криптостойкого шифра нам все еще надо позаботиться о том,

– как генерировать подключи,
– сколько должно быть раундов,
– как определить функцию F.

Работа над *DES* была начата в начале 1970-ых годов группой сотрудников *IBM*, в которую входил и Фейстель. Отправной точкой проекта послужил более ранний шифр — «Люцифер», как называли его в *IBM*. Было известно, что управление национальной безопасности (*NSA*) внесло изменения в проект. Долгое время специалисты в области секретности считали, что изменения состояли в использовании ловушки в функции F. Однако теперь считается, что они были направлены на повышение безопасности шифра. В частности, эти модификации укрепили сопротивляемость шифра дифференциальному криптоанализу — технике, с которой гражданские исследователи не были знакомы до 1980-ых годов.

В документах национального института стандартизации США (*ANSI*) криптосистема *DES* называется *алгоритмом шифрования данных* (*DEA*), а международная организация по стандартизации, ссылаясь на шифр *DES*, пользуется аббревиатурой *DEA-1*. Этот алгоритм являлся мировым стандартом на протяжении более чем двадцати лет и утвердился как первый доступный всем желающим официальный алгоритм. Поэтому его стоит отметить как важнейшую веху на пути криптографии от чисто военного использования к широкомасштабному применению.

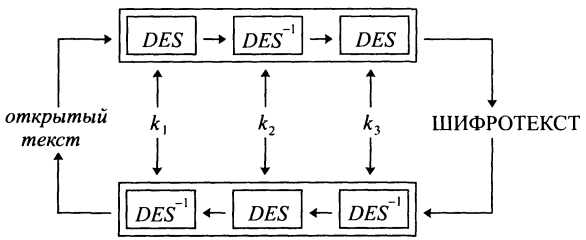
Основные черты шифра *DES* определяются прежде всего тем, что он — обобщение шифра Фейстеля и, кроме того, в нем

– число раундов S равно 16,
– длина блока n — 64 бита,
– размер ключа — 56 битов,
– каждый из подключей k_1, k_2, ..., k_{16} насчитывает 48 битов.

Заметим, что для многих современных алгоритмов длина ключа в 56 битов недостаточна. Поэтому в *DES* зачастую используют три ключа вместо одного, проводя три итерации стандартного процесса. При этом, как легко подсчитать, длина ключа становится равной

168 битам. Такая версия классического шифра называется *тройным DES* или *3DES* (рис. 5.5). Есть и другой способ модификации *DES*, в которой берут два ключа, увеличивая длину основного ключа до 112 битов. Упражнение 5.3.3 посвящено подробностям этого метода.

Рис. 5.5. Тройной DES



5.2.1. Обзор действия шифра *DES*

Как уже не раз отмечалось, в первом приближении *DES* — это шифр Фейстеля с 16 раундами (рис. 5.6), за исключением того, что как перед, так и после основных итераций алгоритма Фейстеля осуществляются некоторые перестановки. Обратите внимание (рис. 5.6) на то, как два блока меняются местами перед последней перестановкой алгоритма. Эта замена никак не влияет на стойкость шифра, и пользователи часто задавались вопросом: зачем ее вообще делать? Один из членов творческого коллектива, разработавшего *DES*, утверждал, что она облегчает микросхемную реализацию процедуры шифрования.

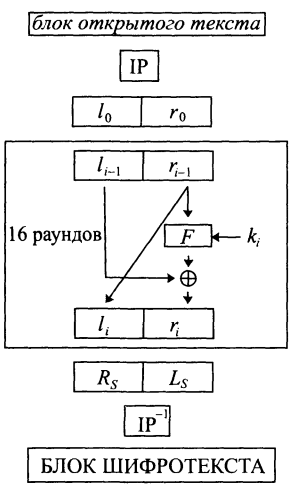


Рис. 5.6. Алгоритмы *DES* и Фейстеля

Шифр *DES* преобразует открытый текст из 64 битов следующим образом:

- производит начальную перестановку (IP);
- расщепляет блок на левую и правую половины;
- осуществляет 16 раундов с одним и тем же набором операций;
- соединяет половины блока;
- производит конечную перестановку.

Конечная перестановка обратна начальной. Это позволяет использовать одно и то же программное обеспечение и «железо» для двух сторон процесса: шифрования и расшифрования. Разворачива-

ние ключа дает 16 подключей по 48 битов каждый, выделяя их из 56-битного основного ключа.

Сейчас мы опишем действие функции F. В каждом раунде алгоритма *DES* оно состоит из шести шагов:

- *Перестановка с расширением.* Правая половина из 32 битов растягивается до 48 битов и перемешивается. Это помогает рассеиванию связи между входными битами и выходными. Перестановка с расширением (отличная от начальной) выбирается так, чтобы один входной бит воздействовал на две замены через S-блоки, о которых речь пойдет ниже. Это помогает распространять зависимости и создает *лавинный эффект* (малое различие между двумя наборами входных данных превращается в большое на выходе).
- *Сложение с подключом.* К строке из 48 битов, полученной после перестановки с расширением, и подключу (его длина тоже 48 битов) применяется операция исключающего ИЛИ, т. е. каждая пара соответствующих битов складывается по модулю 2. Заметим, что подключ используется только в этом месте алгоритма.
- *Расщепление.* Результат предыдущего шага расщепляется на 6 частей по 8 битов в каждом.
- *S-блок.* Каждый 6-битовый кусок передается в один из восьми S-блоков (блоков подстановки), где он превращается в набор из 4 битов. S-блоки — нелинейные компоненты алгоритма *DES* и именно они дают основной вклад в криптостойкость шифра. Каждый S-блок представляет собой поисковую таблицу из четырех строк и шестнадцати столбцов. Шесть входящих в S-блок битов определяют, какую строку и какой столбец необходимо использовать для замены. Первый и шестой бит задают номер строки, а остальные — номер столбца. Выход S-блока — значение соответствующей ячейки таблицы.
- *P-блок.* На этот момент у нас есть восемь групп 4-битовых элементов, которые комбинируются здесь в 32-битовую строку и перемешиваются, формируя выход функции F.

Структура функции F алгоритма *DES* схематически изображена на рис. 5.7.

Сейчас мы подробно расскажем о тех шагах, которые мы пока полностью не раскрыли.

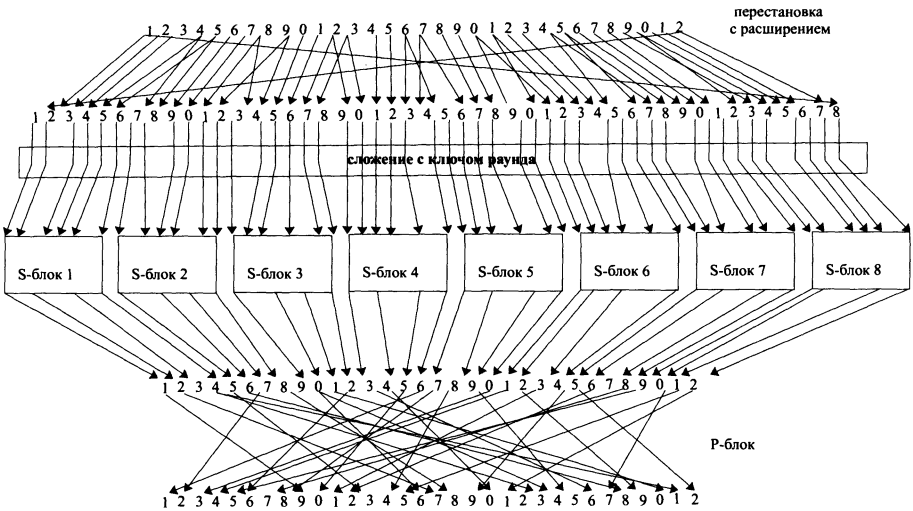


Рис. 5.7. Структура функции F алгоритма *DES*

5.2.1.1. Начальная перестановка, IP. Начальная перестановка алгоритма *DES* определяется таблицей 5.1. Эту и все другие таблицы, изображающие перестановки, следует читать слева направо и сверху вниз. Так, число 58, расположенное в первой строке и первом столбце таблицы, означает, что IP перемещает пятьдесят восьмой бит входных данных на первое место. Аналогично, согласно этой таблице, второй бит перемещается в позицию 50, и т. д.

Таблица 5.1. Начальная перестановка

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Обратная перестановка задается таблицей 5.2.

5.2.1.2. Перестановка с расширением E. Перестановка E также представляется таблицей (таб. 5.3). Каждая строка в ней соответствует битам, входящим в соответствующий S-блок на следующем шаге. Обратите внимание, как биты, нумерующие строку одного S-блока (первый и последний бит каждой строки), влияют на выбор столбца другого S-блока.

Таблица 5.2. Перестановка, обратная к начальной

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Таблица 5.3. Перестановка с расширением

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

5.2.1.3. Перестановка в P-блоке, P. Эта перестановка превращает 8 групп 4-битовых элементов на выходе из S-блоков, в 32-битовую строку, соединяя и перемешивая их, как показано в табл. 5.4.

Таблица 5.4. Перестановка в P-блоке

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

5.2.1.4. S-блок. Содержимое восьми S-блоков алгоритма представлено в табл. 5.5. Напомним, что каждый из них представляет собой таблицу из 4 строк и 16 столбцов.

5.2.2. Разворачивание ключа в *DES*

Разворачивание ключа работает с 56-битовым ключом, который представлен строкой из 64 знаков, включающей в себя контрольные би-

ты, следящие за четностью. Каждый восьмой бит этой строки, т. е. стоящий на позициях 8, 16, ..., 64, отвечает за то, чтобы каждый байт ключа состоял из нечетного числа битов.

Таблица 5.5. S-блоки

S-блок 1.	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S-блок 2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S-блок 3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S-блок 4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S-блок 5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S-блок 6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
cS-блок 7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S-блок 8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

При разворачивании ключа сначала, согласно перестановке из табл. 5.6, перемешиваются его биты. Эта перестановка имеет 64 вхо-

да и 56 выходов. Таким образом, после ее применения, в частности, отбрасываются контрольные биты.

Таблица 5.6. Перестановка *PC-1*

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Результат этой перестановки, называемой в литературе *PC-1*, делится на две половины (по 28 битов в каждой). Левая часть обозначается через C_0, а правая — через D_0. Теперь для каждого раунда с номером i вычисляется

$$C_i = C_{i-1} \lll p_i, \quad D_i = D_{i-1} \lll p_i,$$

где $x \lll p_i$ обозначает циклический сдвиг битовой строки x влево на p_i позиций. Для раундов с номером $i = 1, 2, 3, 9$ и 16 имеем $p_i = 1$, а для остальных — $p_i = 2$.

Наконец, две части C_i и D_i соединяются вместе и подаются на вход следующей перестановки, называемой *PC-2* (табл. 5.7), выходом которой и будет 48-битовый подключ i-го раунда.

Таблица 5.7. Перестановка *PC-2*

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

5.3. Rijndael

Победитель конкурса *AES*, объявленный в конце 2000 года, алгоритм *Rijndael*, был разработан двумя бельгийскими криптографами: Дименом (Daemen) и Рийменом (Rijmen). Эта криптосистема, относясь к блочным шифрам, имеет много общего с *DES*, хотя и

не является непосредственным обобщением шифра Фейстеля. Для обеспечения криптостойкости алгоритм *Rijndael* включает в себя повторяющиеся раунды, каждый из которых состоит из замен, перестановок и прибавления ключа. Кроме того, *Rijndael* использует сильную математическую структуру: большинство его операций основаны на арифметике поля $\mathbb{F}_{2^8}$. Однако, в отличие от *DES*, шифрование и расшифрование в этом алгоритме — процедуры разные.

Напомним, что элементы поля $\mathbb{F}_{2^8}$ хранятся в памяти компьютера в виде 8-битовых векторов (или байтов), представляющих двоичные многочлены. Например, байт $'83h'$ в шестнадцатиричной системе[1] соответствует двоичному числу

$$'1000\,0011b', \quad \text{т. к.} \quad '83h' = 8 \cdot 16 + 3 = 131$$

в десятичной системе. Этот набор двоичных разрядов можно получить непосредственно из байта $'83h' = '80h' + '03h'$, заметив, что цифра 8 шестнадцатиричной системы, стоящая во втором разряде представляет число $8 \cdot 16 = 8 \cdot 2^4$. Само число $8 = 2^3$ в двоичной системе записывается в виде последовательности 1000 двоичных знаков. Значит, число $'80h' = 2^3 \cdot 2^4$ в двоичной системе счисления выглядит как $'1000\,0000b'$. Осталось к этой строке битов прибавить запись числа 3 в двоичной системе, т. е. 0011. Заметьте, что при этом достаточно к числу 8 (1000) в двоичной системе приписать число 3 (0011). Указанная последовательность битов соответствует многочлену $X^7 + X + 1$ над полем $\mathbb{F}_2$. Таким образом, можно сказать, что шестнадцатиричное число $'83h'$ представляет тот же многочлен.

Арифметические операции в поле $\mathbb{F}_{2^8}$ соответствуют операциям над двоичными многочленами из $\mathbb{F}_2[X]$ по модулю неприводимого полинома

$$m(X) = X^8 + X^4 + X^3 + X + 1.$$

В алгоритме *Rijndael* 32-битовые слова отождествляются с многочленами степени 3 из $\mathbb{F}_{2^8}[X]$. Отождествление делается в формате «перевертыш», т. е. старший (наиболее значимый) бит соответствует младшему коэффициенту многочлена. Так, например, слово

$$a_0 || a_1 || a_2 || a_3$$

[1] Российское представление чисел в шестнадцатиричной системе. Цифры: первые десять цифр от 0 до 9 представляются стандартно, цифры от 10 до 15 замещаются буквами A,B,C,D,E,F. Для перевода десятичного числа в шестнадцатиричную систему, переведем его сначала в двоичную, затем разделим на блоки по 4 цифры и каждый блок заменим шестнадцатиричной цифрой. Например, $'0110\,1100\,1111\,b' = '6CFh'$

соответствует многочлену

$$a_3X^3 + a_2X^2 + a_1X + a_0.$$

Арифметика в алгоритме совпадает с арифметическими действиями в кольце многочленов $\mathbb{F}_{2^8}[X]$ по модулю многочлена $M(X) = X^4 + 1$. Заметим, что многочлен $M(X) = (X+1)^4$ приводим, и, следовательно, арифметические действия в алгоритме отличны от операций поля, в частности, бывают пары ненулевых элементов, произведение которых равно 0.

Rijndael — *настраиваемый* блочный алгоритм, который может работать с блоками из 128, 192 или 256 битов. Для каждой комбинации блока и размера ключа определено свое количество раундов. В целях упрощения обсуждения мы рассмотрим самый простой и, вероятно, наиболее часто используемый вариант алгоритма, при котором блоки, как и ключ, состоят из 128 битов. В этом случае в алгоритме выполняется 10 раундов. С данного момента мы будем иметь дело лишь с этой простой версией.

Rijndael оперирует с внутренней байтовой матрицей размера 4×4, называемой матрицей состояний:

$$S = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix},$$

которую обычно записывают как вектор 32-битовых слов. Каждое слово в векторе представляет столбец матрицы. Подключи также хранятся в виде матрицы 4×4:

$$K_i = \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix}.$$

5.3.1. Операции алгоритма *Rijndael*

Раундовая функция в *Rijndael* действует с использованием четырех операций, которые мы сейчас опишем.

5.3.1.1. SubBytes. В алгоритме есть два типа S-блоков. Один тип применяется при шифровании, а другой — при расшифровании.

Каждый из них обратен другому. S-блоки в алгоритме *DES* отбирались из большого числа себе подобных так, чтобы предотвратить взлом шифра с помощью дифференциального криптоанализа. В *Rijndael* S-блоки имеют прозрачную математическую структуру, что позволяет формально анализировать устойчивость шифра к дифференциальному и линейному анализам. Эта математическая структура не только повышает сопротивляемость дифференциальному анализу, но и убеждает пользователя, что в алгоритм не закрался недосмотр.

S-блоки поочередно обрабатывают строки матрицы состояний $s = [s_7, \ldots, s_0]$, воспринимая их как элементы поля $\mathbb{F}_{2^8}$. Их работа состоит из двух шагов.

1. Вычисляется мультипликативный обратный к элементу $s \in \mathbb{F}_{2^8}$ и записывается как новый байт $x = [x_7, \ldots, x_0]$. По соглашению, элемент $[0, \ldots, 0]$, не имеющий обратного, остается неизменным.

2. Битовый вектор x с помощью линейного преобразования над полем $\mathbb{F}_2$ переводится в вектор y:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix},$$

служащий выходом S-блока. Действия S-блока на стадии расшифрования состоят в обратном линейном преобразовании и вычислении мультипликативного обратного. Эти преобразования байтов можно осуществить, используя табличный поиск или микросхему, реализующую вычисление обратных элементов в $\mathbb{F}_{2^8}$ и линейные преобразования.

5.3.1.2. ShiftRows. Операция `ShiftRows` в *Rijndael* осуществляет раундический сдвиг матрицы состояний. Каждая из ее строк сдвигается на свое число позиций. В рассматриваемой версии ши-

фра это преобразование имеет вид:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}.$$

Обратная операция — тоже простой циклический сдвиг, но в противоположном направлении. Операция `ShiftRows` гарантирует, что столбцы матрицы состояний будут «взаимодействовать» друг с другом на протяжении нескольких раундов.

5.3.1.3. MixColumns. Операция `MixColumns` задумана с тем, чтобы строки матрицы состояний «взаимодействовали» друг с другом на протяжении всех раундов. В комбинации с предыдущей операцией она наделяет каждый байт выходных данных зависимостью от каждого байта на входе.

Мы представляем каждый столбец матрицы состояний как многочлен степени 3 с коэффициентами из $\mathbb{F}_{2^8}$:

$$a(X) = a_0 + a_1 X + a_2 X^2 + a_3 X^3.$$

Новый столбец получается умножением многочлена $a(X)$ на фиксированный многочлен[2]

$$c(x) = '02h' + '01h' \cdot X + '01h' \cdot X^2 + '03h' \cdot X^3$$

по модулю многочлена $M(X) = X^4 + 1$. Так как умножение на многочлен — линейная операция, ее можно представить в виде действия матрицы:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} '02h' & '03h' & '01h' & '01h' \\ '01h' & '02h' & '03h' & '01h' \\ '01h' & '01h' & '02h' & '03h' \\ '03h' & '01h' & '01h' & '02h' \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}.$$

Матрица коэффициентов невырождена над $\mathbb{F}_{2^8}$, поэтому операция `MixColumns` обратима, а обратное к ней действие реализуется матрицей, обратной к выписанной.

[2] Элементы поля $\mathbb{F}_{2^8}$ мы записываем либо как битовый вектор, либо как байт. В частности, $'03h'$ — это элемент поля $\mathbb{F}_{2^8}$, представляемый многочленом $X+1$ по модулю $M(x)$. — *Прим. перев.*

5.3.1.4. AddRoundKey. Сложение с подключом осуществляется просто. Нужно сложить по модулю 2 (применить операцию исключающего ИЛИ) все байты матрицы состояний (которую мы постоянно меняли) с соответствующими элементами матрицы подключа. Обратная операция, очевидно, совпадает с исходной.

5.3.2. Структура раундов

Запишем алгоритм *Rijndael* на псевдокоде.

```
AddRoundKey(S,K[0]);
for (i=1; i<=9; i++)
   { SubBytes(S);
     ShiftRows(S);
     MixColumns(S);
     AddRoundKey(S,K[i]);
   }
SubBytes(S);
ShiftRows(S);
AddRoundKey(S,K[10])
```

Блок открытого текста, предназначенный для шифрования, записывается в виде матрицы состояний S. Полученный в результате алгоритма шифротекст представляется той же матрицей. Обратите внимание, что в последнем раунде операция `MixColumns` не осуществляется.

Процедура расшифрования представлена следующей программой на псевдокоде.

```
AddRoundKey(S,K[10]);
InverseShiftRows(S);
InverseSubBytes(S);
for (i=9; i>=1; i--)
   { AddRoundKey(S,K[i]);
     InverseMixColumns(S);
     InverseShiftRows(S);
     InverseSubBytes(S)
   }
AddRoundKey(S,K[0]);
```

5.3.3. Разворачивание ключа

Единственное, что осталось осветить, это вычисление подключей. Напомним, что основной ключ алгоритма состоит из 128 битов, а нам нужно произвести 10 подключей $K_1, \ldots, K_{10}$, каждый из которых включает в себя четыре 32-битовых слова. Эти слова соответствуют столбцам матрицы, описанной на стр. 140. Здесь используется константа раунда RC_i, вычисляющаяся по правилу

$$RC_i = X^i \pmod{X^8 + X^4 + X^3 + X + 1}.$$

Обозначим i-ый подключ через $(W_{4i}, W_{4i+1}, W_{4i+2}, W_{4i+3})$. Основной ключ алгоритма делится на четыре 32-битовых слова (k_0, k_1, k_2, k_3), после чего подключи получаются в результате выполнения приведенного ниже алгоритма. В нем через `RotBytes` обозначена процедура циклического сдвига слова влево на один байт, а через `SubBytes` — применение S-блока (из этапа шифрования) к каждому байту слова.

```
W[0]=K[0]; W[1]=K[1]; W[2]=K[2]; W[3]=K[3];
for (i=1; i<=10; i++)
   { T=RotBytes(W[4*i-1]);
     T=SubBytes(T);
     T=T∧RC[i];
     W[4*i]=W[4*i-4]∧T;
     W[4*i+1]=W[4*i-3]∧W[4*i];
     W[4*i+2]=W[4*i-2]∧W[4*i+1];
     W[4*i+3]=W[4*i-1]∧W[4*i+2]; }
```

5.4. Режимы работы DES

Блочный шифр, подобный *DES* или *Rijndael*, можно по-разному использовать для шифрования строк данных. Вскоре после *DES* в США был принят еще один федеральный стандарт, *рекомендующий* четыре способа эксплуатации алгоритма *DES* для шифрования данных. С тех пор эти режимы стали общепринятыми и применяются с любыми блочными шифрами. Перечислим их.

- ***ECB.*** Этот режим прост в обращении, но слабо защищен от возможных атак с удалениями и вставками. Ошибка, допущенная в одном из битов шифротекста, влияет на целый блок в расшифрованном тексте.
- ***CBC*** — наилучший способ эксплуатации блочного шифра, поскольку предназначен для предотвращения потерь в результа-

те атаки с использованием удалений и вставок. Здесь ошибочный бит шифротекста при расшифровании не только превращает в ошибочный блок, в котором содержится, но и портит один бит в следующем блоке открытого текста, что можно легко определить и интерпретировать как сигнал о предпринятой атаке.

- ***OFB***. При таком методе блочный шифр превращается в поточный. Режим обладает тем свойством, что ошибка в один бит, просочившаяся в шифротекст, дает только один ошибочный бит в расшифрованном тексте.
- ***CFB***. Как и в предыдущем случае, здесь блочный шифр трансформируется в поточный. Отдельная ошибка в криптограмме при этом влияет как на блок, в котором она была допущена, так и на следующий блок, как при режиме *CBC*.

Разберем каждый из стандартных режимов шифрования подробно.

5.4.1. Режим *ECB*

Режим *ECB* (Electronic Code Book — электронная кодовая книга) является простейшим среди стандартных способов использования блочного шифра. Данные m, которые предстоит зашифровать, делятся на блоки по n битов:

$$m_1, m_2, \ldots, m_q.$$

Последний из них, при необходимости, дополняют до длины n. По ним определяются блоки $C_1, \ldots, C_q$ как результат воздействия шифрующей функции

$$C_i = E_k(m_i),$$

как показано на рис. 5.8. Расшифрование здесь — простое обращение предыдущей операции (см. рис. 5.9).

С режимом *ECB* связан ряд проблем. Первая возникает из-за того, что при равенстве $m_i = m_j$ мы получим одинаковые блоки шифротекста $C_i = C_j$, т. е. одинаковые блоки на входе индуцируют совпадающие блоки на выходе. Это, действительно, проблема, поскольку шаблонные начало и конец сообщений совпадают. Вторая проблема связана с тем, что удаление из сообщения какого-либо блока не оставляет следов, и атакующий может таким образом исказить передаваемую

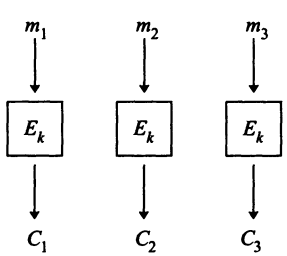


Рис. 5.8. Шифрование в режиме *ECB*

информацию. Третья очень близка ко второй, но связана со вставкой блоков из других сообщений.

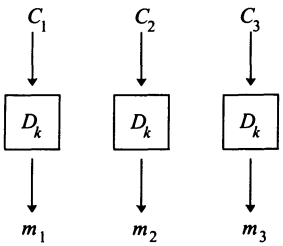


Рис. 5.9. Расшифровывание в режиме *ECB*

Чтобы лучше представить себе эти проблемы, возьмем простейшую модель шифра, в которой блок соответствует слову, и предположим, что открытое сообщение

Плати Алисе сто фунтов,
Не плати Бобу двести фунтов

в зашифрованном виде выглядит так:

У КОШКИ ЧЕТЫРЕ НОГИ,
А У ЧЕЛОВЕКА ДВЕ НОГИ.

Мы можем теперь заставить получателя оплатить Алисе две сотни фунтов, вместо одной, отправив ему сообщение

У КОШКИ ДВЕ НОГИ,

которое получено из первого заменой одного из блоков на блок из второго сообщения. Кроме того, мы можем приостановить выплаты Алисе, поставив блок «А» шифротекста в начало первого сообщения. Надеюсь, Вы заметили, что «А»— шифрованный вариант частицы «не» открытого текста. Сообразите, как мы можем побудить получателя криптограммы выплатить Бобу двести фунтов.

Таким атакам можно противостоять, добавляя контрольные суммы нескольких блоков открытого текста или используя режим, при котором к каждому блоку шифротекста добавляется «контекстный идентификатор».

5.4.2. Режим *CBC*

Один из путей обхода проблем, возникающих при использовании режима *ECB*, состоит в «зацеплении» шифра, т. е. в добавлении к каждому блоку шифротекста контекстного идентификатора. Самый простой способ сделать это — применить режим «сцепления блоков шифра» или *CBC* (сокращение от «Cipher Block Chaining»).

В этом режиме открытый текст, как обычно, разбивается на серию блоков:

$$m_1, \ldots, m_q.$$

Как и в предыдущем режиме, последний блок может потребовать дополнения, чтобы длина открытого текста стала кратной длине

блока. Шифрование осуществляется согласно формулам

$$C_1 = E_k(m_1 \oplus IV), \qquad C_i = E_k(m_i \oplus C_{i-1}) \text{ при } i > 1,$$

(см. рис. 5.10).

Обратите внимание на то, что в вычислении первого блока шифротекста участвует величина IV (начальное значение), которую следует отнести к заданию шифрующей функции. Величину IV привлекают к шифрованию с тем, чтобы шифрованные версии одинаковых частей открытого текста выглядели поразному. Нет необходимости скрывать значение IV, и на практике ее передают в открытом виде как часть сообщения.

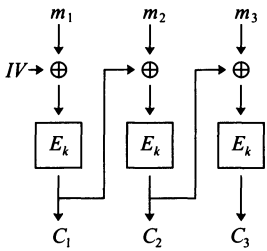


Рис. 5.10. Шифрование в режиме *CBC*

Естественно, величина IV участвует и в расшифровании. Этот процесс выглядит следующим образом (рис. 5.11):

$$m_1 = D_k(C_1) \oplus IV, \qquad m_i = D_k(C_i) \oplus C_{i-1} \text{ при } i > 1.$$

Напомним, что при шифровании в режиме *ECB* ошибка в одном знаке шифротекста, появляющаяся на стадии передачи сообщения, повлияет на весь блок, в котором она допущена, и он, естественно, будет расшифрован неверно. В случае режима *CBC*, как видно из формул, ошибочный знак повлияет не только на свой блок, но и на соответствующий бит в следующем блоке.

5.4.3. Режим *OFB*

Режим, называемый «обратной связью по выходу», или *OFB* (Output Feedback), адаптирует блочный шифр к его поточному использованию. Для этого выбирается переменная j ($1 \leqslant j \leqslant n$), обозначающая число битов на выходе генератора потока ключей при каждой итерации. С помощью блочного шифра создается поток ключей, j битов за один раз. Рекомендуется брать j равное n, поскольку при этом ожидаемая длина периода потока ключей получается большей, нежели при других значениях.

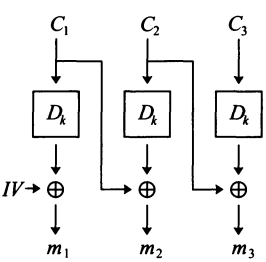


Рис. 5.11. Расшифрование в режиме *CBC*

Как и ранее, мы разбиваем открытый текст на серию блоков:

$$m_1, \ldots, m_q.$$

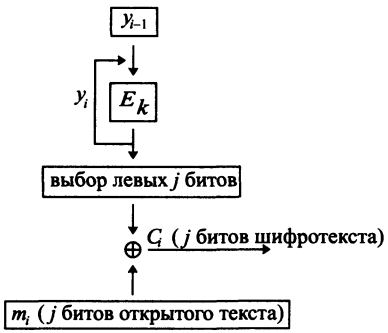


Рис. 5.12. Шифрование в режиме *OFB*

Но на этот раз, в отличие от предыдущих случаев, блоки состоят из j битов. Процесс шифрования происходит по следующей схеме (рис. 5.12). Прежде всего, переменной x_1 присваивается начальное значение IV. Затем, при $i = 1, 2, \ldots, q$, делаются преобразования:

$y_i = E_k(x_i)$,

$e_i = j$ крайних слева битов блока y_i,

$C_i = m_i \oplus e_i$,

$x_{i+1} = y_i$.

Расшифрование происходит аналогично (см. рис. 5.13).

5.4.4. Режим *CFB*

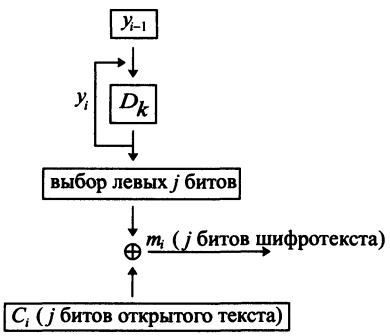


Рис. 5.13. Расшифрование в режиме *OFB*

Последний режим, который мы рассмотрим, носит название «обратной связи по шифротексту» или *CFB* (Cipher FeedBack). Он похож на режим *OFB*, но блочный шифр в нем трансформируется в поточный. Напомним, что в предыдущем режиме начало потока ключей получается из значения IV, а остальной поток формируется пошагово, в результате шифрования значения шифрующей функции, вычисленного на предыдущей стадии. В случае *CFB* поток ключей возникает в результате еще одного шифрования блоков криптограммы (рис. 5.14):

$y_0 = IV$, $\qquad z_i = E_k(y_{i-1})$

$e_i = j$ крайних слева битов блока z_i, $\qquad y_i = m_i \oplus e_i$.

Мы не будем разбирать процедуру расшифрования, оставив ее в качестве упражнения.

5.5. Подлинность сообщений

Блочные шифры могут не только обеспечивать конфиденциальность сообщений, но и контролировать целостность данных. Предположим, что стороны, обменивающиеся секретной информацией, хотят быть уверенными в том, что информация при передаче не была искажена. Они могут использовать общий (shared) секретный ключ и алгоритм с ключом (например, блочный шифр) для генерирования контрольного значения, посылаемого вместе с данными. Такие значения называют *кодами аутентификации сообщений*, или *MAC* (Message Authentication Codes):

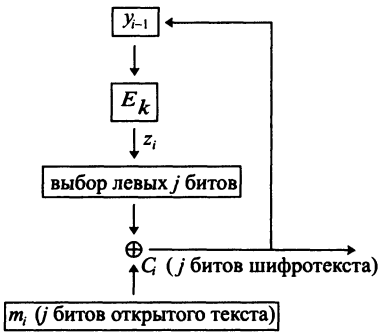


Рис. 5.14. Шифрование в режиме *CFB*

$$MAC = F_k(M),$$

где F — контрольная функция (check function), k — секретный ключ, а M — сообщение. При этом мы не предполагаем секретность сообщения[1], поскольку пытаемся на этом этапе сохранить целостность данных, а не их конфиденциальность. Если нам нужно засекретить сообщение, то его следует зашифровать после вставки MAC. В результате такой комбинации пользователь передает

$$E_k(m||MAC(m)).$$

Существует несколько различных типов MAC, но наиболее известная и широко используемая — это *CBC-MAC*. Этот код обобщает режим *CBC* блочного шифра. Коды *CBC-MAC* упоминаются в различных международных стандартах, изданных в начале 80-ых годов прошлого столетия. Точности ради стоит отметить, что в стандартах говорится об использовании алгоритма *DES* в режиме *CBC* для создания системы аутентификации сообщений, хотя, очевидно, вместо *DES* можно брать любой блочный шифр.

Опишем процедуру, в результате которой из n-битового блочного шифра получается m-битовый ($m \leqslant n$) MAC.

- Сначала данные дополняются так, чтобы их можно было разбить на серию n-битовых блоков.

[1] Поэтому оно и обозначено заглавной буквой. — *Прим. перев.*

- Полученные блоки шифруются блочным шифром в режиме CBC.
- Последний из полученных блоков после необязательной заключительной обработки и отбрасывания лишних битов (при $m < n$) является требуемым кодом аутентификации MAC.

Таким образом, если $M_1, \ldots, M_q$ — n-битовые блоки данных, то этапы вычисления MAC можно описать так:

$$o_1 = E_k(M_1), \quad l_i = M_i \oplus o_{i-1} \text{ для } i = 2, \ldots, q, \quad o_i = E_k(l_i).$$

Последнее значение o_q подается на вход необязательной заключительной процедуры, результат которой обрезается до нужного размера и считается искомым MAC (рис. 5.15).

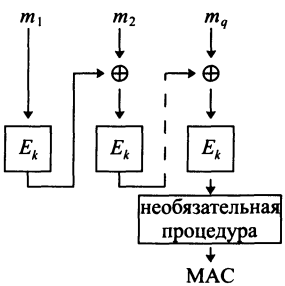


Рис. 5.15. Схема работы *CBC-MAC*

При пользовании схемой вычисления CBC-MAC необходимо осторожно обращаться с начальными дополнениями данных. Международные стандарты предлагают три способа дополнения:

1. Все дополняющие символы — нули. Этот метод имеет множество недостатков, связанных с тем, что при нем невозможно обнаружить добавление, стирание или перемещение нулей, если не известна длина оригинального сообщения.
2. В конце всех добавляемых по необходимости нулей вставляется одна единица, сигнализирующая о конце сообщения, если оно оканчивается строкой нулей.
3. Опять все дополняющие символы — нули. Но на этот раз к сообщению добавляется специальный блок, содержащий информацию о длине исходного текста.

Стандарты предписывают два возможных необязательных заключительных шага, разработанных для усложнения атакующим поиска ключа методом полного перебора:

1) выбрать ключ k_1 и вычислить $o_q = E_k(D_{k_1}(o_q))$;
2) выбрать ключ k_2 и вычислить $o_q = E_{k_2}(o_q)$.

Другой способ аутентификации основан на криптографических хэш-функциях (см. §10.3). При этом секретный ключ присоединяется к сообщению, и вся связка подается на вход хэш-функции. Выход хэш-функции и есть код MAC.

5.6. Современные поточные шифры

При шифровании большого объема данных (таких, например, как речь, или «живое» видео) в реальном времени, требуется что-то более быстрое, нежели один из блочных шифров, рассмотренных ранее. Для этих целей подходят поточные шифры. Они, как правило, похожи на одноразовый шифр-блокнот, с которым мы уже встречались. Их суть заключается в сложении по модулю 2 битов потока ключей с битами сообщения. Однако вместо фиксированного потока ключей, который в ранних поточных шифрах и являлся секретным ключом, в современных системах поток ключей генерируется из короткого основного ключа с помощью однозначно определенных детерминированных алгоритмов.

5.6.1. *РСЛОС*

Стандартный способ генерирования потока битов основан на использовании *регистра сдвига с обратной связью*. Это микросхема с ячейками памяти, в каждой из которых записан один бит информации. Множество таких ячеек и образует регистр. На каждом шаге содержимое нескольких заранее определенных ячеек, которые называются *отводами*, пропускается через *функцию обратной связи*. А ее значение записывается в самую левую ячейку регистра, сдвигая все остальные его биты на одну позицию вправо. Самый крайний справа, «вытолкнутый» из регистра, бит — выход регистра сдвига на данном шаге (рис. 5.16).

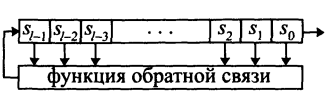


Рис. 5.16. Регистр сдвига с обратной связью

По причинам, о которых мы будем говорить позже, в качестве функций обратной связи желательно брать нелинейные функции. Однако это сложно осуществить на практике, и поэтому пользуются *регистром сдвига с линейной обратной связью* или, сокращенно, *РСЛОС*, функция обратной связи в котором линейна. Этот регистр устроен так же, как и общий, описанный выше. Но в качестве функции обратной связи берется логическая операция XOR исключающего ИЛИ.

На языке математики работу регистра длины l можно описать следующим образом. Задается последовательность битов $[c_1, \ldots, c_l]$, в которой на отводах стоят единицы, а в остальных ячейках — нули. Допустим, что в начальном положении в регистре записана последовательность $[s_{l-1}, \ldots, s_1, s_0]$. На выходе регистра получается

последовательность $s_0, s_1, s_2, \ldots, s_{l-1}, s_l, s_{l+1}, \ldots$, где при $j \geqslant l$

$$s_j = c_1 \cdot s_{j-1} \oplus c_2 \cdot s_{j-2} \oplus \cdots \oplus c_l \cdots s_{j-l}.$$

Заметим, что если в начальном состоянии регистра во всех его ячейках стояли нули, то и на выходе мы будем получать сплошные нули. Если же стартовая последовательность была нетривиальной, то генерируемая последовательность, в конечном счете, окажется периодической. Действительно, число возможных последовательностей длины l конечно, и меняя их на каждом шагу, мы волей-неволей получим одну из уже встречавшихся. После этого, ввиду детерминированности процесса, мы получим периодичность. Напомним, что периодом последовательности называется наименьшее натуральное число N, при котором для всех достаточно больших значений параметра i выполняется равенство

$$s_{N+i} = s_i.$$

Нетрудно подсчитать, что существует $2^l - 1$ разных ненулевых начальных состояний регистра. Поэтому максимум, на что можно рассчитывать, это на то, что регистр сдвига с линейной обратной связью при любом ненулевом начальном положении будет генерировать последовательность битов периода $2^l - 1$.

Свойства выдаваемой *РСЛОС* последовательности тесно связаны со свойствами двоичного многочлена

$$C(X) = 1 + c_1 X + c_2 X^2 + \cdots + c_l X^l \in \mathbb{F}_2[X],$$

ассоциированного с этим регистром. Его ненулевые коэффициенты называются отводами, как и соответствующие ячейки регистра, поставляющие значения аргументов функции обратной связи. На рис. 5.17 изображен *РСЛОС*, ассоциированный с многочленом X^3+X+1, а на рис. 5.18 — с многочленом $X^{32} + X^3 + 1$. На практике применяют *РСЛОС* с примитивным ассоциированным многочленом.

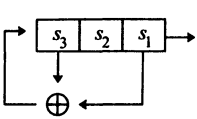


Рис. 5.17. Регистр сдвига с линейной обратной связью, ассоциированный с многочленом $X^3 + X + 1$

Определение 5.1. Двоичный многочлен $C(X)$ степени l называется *примитивным*, если он неприводим, а его корень θ является образующей мультипликативной группы поля $\mathbb{F}_{2^l}$. Иначе говоря, примитивный многочлен степени l удовлетворяет двум условиям:

- $\mathbb{F}_2[X]/(C(X)) = \mathbb{F}_2(\theta) = \mathbb{F}_{2^l}$,
- $\mathbb{F}_{2^l}^* = \langle\theta\rangle$.

Свойства последовательности битов, генерируемой *РСЛОС*, зависят от ассоциированного многочлена:

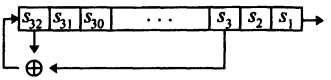


Рис. 5.18. Регистр сдвига с линейной обратной связью, ассоциированный с многочленом $X^{32} + X^3 + 1$

- Если старший коэффициент ассоциированного многочлена $c_l = 0$, то периодичность генерируемой последовательности может проявляться не сразу.
- Если $c_l = 1$, то соответствующая последовательность называется *неособой*. Она будет периодичной с самого начала, т. е. равенство $s_{N+i} = s_i$ выполнено для всех i, а не только для достаточно больших. Наиболее интересны неособые последовательности, соответствующие многочленам со следующими дополнительными свойствами:
  - Когда $C(X)$ неприводим, при любом ненулевом начальном состоянии регистра период генерируемой последовательности равен наименьшему числу N, при котором многочлен $C(X)$ делит $1 + X^N$. Как следствие, период последовательности будет делить число $2^l - 1$.
  - Если $C(X)$ примитивен, то любое ненулевое начальное состояние регистра дает последовательность с максимально возможным периодом $2^l - 1$.

Опустим доказательство этих утверждений. Его можно найти в любом хорошем учебнике, посвященном приложениям конечных полей к теории шифрования, криптографии или системам передачи данных. В качестве примера рассмотрим *РСЛОС* с ассоциированным многочленом $1 + X + X^3$. Генерируемая последовательность в этом случае имеет вид

$$s_j = s_{j-1} \oplus s_{j-3}.$$

Допустим, что перед началом процесса в регистре записана последовательность $[0, 0, 1]$, тогда период генерируемого потока битов будет равен 7, и мы получаем последовательность, представленную в табл. 5.8.

Поскольку внутреннее состояние на седьмом шаге вернулось к исходному, то начиная со следующего шага мы будем получать повторения. Иными словами, период последовательности оказался равен 7, что произошло ввиду примитивности многочлена $X^3 + X + 1$.

Мы не будем здесь обсуждать существующие алгоритмы, генерирующие примитивные многочлены. Ограничимся лишь неболь-

шим списком таких многочленов с малым числом отводов. Очевидно, что чем меньше отводов в многочлене, тем быстрее работает соответствующий регистр сдвига.

$$X^{31}+X^3+1, \quad X^{31}+X^6+1, \quad X^{31}+X^7+1,$$
$$X^{39}+X^4+1, \quad X^{60}+X+1, \quad X^{63}+X+1,$$
$$X^{71}+X^6+1, \quad X^{93}+X^2+1, \quad X^{137}+X^{21}+1,$$
$$X^{145}+X^{52}+1, \quad X^{161}+X^{18}+1, \quad X^{521}+X^{32}+1.$$

Хотя *РСЛОС* быстро генерирует поток битов, исходя из ключа малого размера, особенно при аппаратной реализации, они плохо подходят для целей криптографии ввиду их линейности, собственно, именно того свойства, которое придает им высокую эффективность в аппаратных средствах.

Таблица 5.8. Пример генерируемой последовательности бит

Номер шага	состояние	генерируемый бит
0	[0,0,1]	-
1	[1,0,0]	1
2	[1,1,0]	0
3	[1,1,1]	0
4	[0,1,1]	1
5	[1,0,1]	1
6	[0,1,0]	1
7	[0,0,1]	0

Обосновывая слабую криптостойкость линейных регистров, покажем, что зная длину l регистра и $2l$ последовательных битов, вышедших из *РСЛОС*, можно вычислить весь генерируемый поток. Заметим, что для этого нам достаточно определить значения отводов, т. е. набор c_i коэффициентов ассоциированного многочлена, поскольку стартовое состояние регистра $S_0, \ldots, S_{l-1}$ можно взять из известной нам последовательности сгенерированных битов. Такие данные можно получить с помощью атаки с известным открытым текстом, когда мы получаем шифрограмму, соответствующую известной нам части исходного сообщения. Напомним, что функция обратной связи в этой ситуации — просто сложение значений отводов по модулю 2, действующая по формуле

$$S_j = \sum_{j-1}^{l} c_i \cdot S_{j-1} \pmod 2.$$

Используя это соотношение для $j = l, \ldots, 2l-1$, получим систему l линейных уравнений с неизвестными c_i, которые нам и нужно определить. В матричной форме эта система выглядит следующим образом:

$$\begin{pmatrix} S_{l-1} & S_{l-2} & \ldots & S_1 & S_0 \\ S_l & S_{l-1} & \ldots & S_2 & S_1 \\ \vdots & \vdots & & \vdots & \vdots \\ S_{2l-3} & S_{2l-4} & \ldots & S_{l-1} & S_{l-2} \\ S_{2l-2} & S_{2l-3} & \ldots & S_l & S_{l-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{l-1} \\ c_l \end{pmatrix} = \begin{pmatrix} S_l \\ S_{l+1} \\ \vdots \\ S_{2l-2} \\ S_{2l-1} \end{pmatrix}.$$

Предположим, что мы перехватили последовательность битов

$$1,1,1,1,0,1,0,1,1,0,0,1,0,0,0,\ldots,$$

созданную 4-битовым *РСЛОС*. Подставляя в предыдущую систему уравнений элементы перехваченной последовательности и решая ее по модулю 2, найдем, что многочлен, ассоциированный с *РСЛОС*, равен $X^4 + X + 1$. Можно сделать вывод, что поточный шифр, основанный на единственном *РСЛОС*, беззащитен перед атаками с известным открытым текстом.

Мерой криптографического качества последовательности бит служит понятие линейной сложности последовательности.

Определение 5.2. *Линейной сложностью* бесконечной последовательности битов

$$s = s_0, s_1, s_2, s_3, \ldots,$$

называется величина $L(s)$, равная

- 0, если s — последовательность нулей,
- ∞, если s нельзя получить с помощью какого-нибудь *РСЛОС*,
- длине наименьшего *РСЛОС*, выдающего последовательность s в остальных случаях.

Поскольку бесконечную последовательность битов просмотреть невозможно, ограничиваются ее первыми n элементами, обозначая эту часть как s^n. Имеют место следующие свойства.

- Для любого $n \geqslant 1$ выполнено неравенство $0 \leqslant L(s^n) \leqslant n$.
- Если последовательность s периодична и ее период равен N, то $L(s) \leqslant N$.
- $L(s \oplus t) \leqslant L(s) + L(t)$.

Ожидаемая линейная сложность *случайной* двоичной последовательности s^n (именно ее мы и хотели бы видеть в качестве потока ключей) должна быть больше, чем $n/2$. С другой стороны, *РСЛОС* длины l выдает последовательность, чья линейная сложность удовлетворяет соотношению: $L(s^n) = l$ для всех $n \geqslant l$. Таким образом, линейный регистр генерирует потоки битов слишком далекие от случайных.

Как мы видели, зная длину *РСЛОС* и генерируемую им последовательность битов, можно вычислить ассоциированный с регистром многочлен. Для определения длины регистра используется ряд линейных сложностей, т. е. последовательность $L(s^1)$, $L(s^2)$, $L(s^3)$, ... Кроме того, существует эффективный алгоритм Берлекэмпа – Мэсси, вычисляющий ряд линейных сложностей $L(s^1)$, $L(s^2)$, ..., $L(s^n)$ для данной конечной последовательности s^n.

Следовательно, при генерировании потока ключей регистром сдвига с линейной обратной связью длины l, атакующий, получив не более $2l$ последовательных битов этого потока, может раскрыть регистр полностью и самостоятельно производить бегущий ключ любой нужной ему длины. Поэтому возникает необходимость найти *нелинейный* способ эксплуатации *РСЛОС*, при котором генерируемая последовательность обладает высокой линейной сложностью.

5.6.2. Комбинирование *РСЛОС*

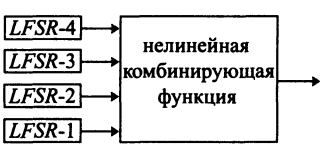


Рис. 5.19. Комбинирование *РСЛОС*

На практике используют несколько регистров с линейной обратной связью одновременно, генерируя совокупность последовательностей $x_1^{(i)}, \ldots, x_n^{(i)}$. Ключом такого способа эксплуатации служит начальное положение всех *РСЛОС*, а генерируемый поток битов получается в результате применения нелинейной комбинирующей функции к их выходным последовательностям, как показано на рис. 5.19.

В качестве комбинирующей выбирают булеву функцию, равную сумме различных произведений переменных, например,

$$f(x_1, x_2, x_3, x_4, x_5) = 1 \oplus x_2 \oplus x_3 \oplus x_4 \cdot x_5 \oplus x_1 \cdot x_2 \cdot x_3 \cdot x_5.$$

Предположим, что у нас есть n *РСЛОС* с максимальным периодом (т. е. ассоциированные многочлены примитивны) с попарно различными периодами $l_1, \ldots, l_n$, каждый из которых больше двух. Тог-

да линейная сложность потока ключей, генерируемого функцией $f(x_1, \ldots, x_n)$, вычисляется с помощью функции f:

$$f(l_1, \ldots, l_n).$$

Здесь сложение и умножение по модулю 2 заменено на обычные сложение и умножение целых чисел, так что f записывается в привычной алгебраической форме. Над развитием способов комбинирования *РСЛОС* с целью создания поточных шифров работают многочисленные творческие коллективы. Мы оставляем читателю исследование этого вопроса по дополнительной литературе, поскольку объем книги ограничен и мы не можем позволить себе разрабатывать здесь эту тему.

5.6.3. *RC4*

RC — сокращение от английского *Ron's Cipher* (шифр Рона). Этот шифр разработал Рон Ривест (Ron Rivest) из Массачуссетского технологического института (MIT). Не следует думать, что шифр *RC4* — более ранняя версия блочных шифров *RC5* и *RC6*. На самом деле, *RC4* — очень и очень быстрый поточный шифр, на удивление простой для запоминания.

Массив $S[0 \ldots 255]$, заполненный целыми числами от 0 до 255, переставленными некоторым, зависящим от ключа, способом, подается на вход *RC4*. В результате работы алгоритма получается поток ключей k, который складывается по модулю 2 с открытым текстом по одному байту за один раз. Поскольку алгоритм оперирует с байтами, а не с битами, и использует наиболее простую операцию, он является быстрым и в программной реализации. Опишем его подробно. Сначала обнуляются переменные i и j, а затем повторяется следующая процедура:

```
i=(i+1)%256;
j=(j+S[i])%256;
swap(S[i], S[j]);
t=(S[i]+S[j])%256;
K=s[t];
```

Напомним, что %256 обозначает оператор вычисления остатка по модулю 256 (см. стр. 23), а функция `swap` меняет местами свои аргументы. Стойкость шифра основана на следующем наблюдении: даже если нападающий узнал ключ K и номер шага i, он может вычислить всего лишь значение $S[t]$, но не все внутреннее состояние массива.

Это следует из того, что нападающий не в состоянии определить значение переменной t, не зная j, $S[i]$ или $S[j]$

Это очень сильный алгоритм, поскольку каждый его шаг увеличивает криптостойкость.

- **i=(i+1)%256** обеспечивает использование каждого элемента массива, причем однократное.
- **j=(j+S[i])%256** обеспечивает нелинейную зависимость выходных данных от массива.
- **swap(S[i],S[j])** изменяет массив в процессе итераций.
- **t=(S[i]+S[j])%256**. Благодаря этому этапу генерируемая последовательность мало говорит о внутреннем состоянии массива.

Начальная перестановка массива S определяется ключом по следующему правилу. На старте алгоритма значения ячеек весьма незамысловаты: $S[i] = i$. В другой массив длиной 256 байтов заносится необходимое число копий ключа. После этого присваивают параметру j нулевое значение и выполняют следующие шаги, последовательно увеличивая параметр i от 0 до 255:

```
j=(j+S[i]+K[i])%256;
swap(S[i],S[j]);
```

Краткое содержание главы

- Современные шифры с симметричным ключом относятся к блочным или поточным шифрам.
- Наиболее известный и широко используемый блочный шифр — это *DES*, обобщающий шифр Фейстеля.
- Относительно новый блочный шифр — это *Rijndael*.
- Как *DES*, так и *Rijndael* добиваются высокой криптостойкости благодаря повторяющимся простым раундам, состоящим из замен, перестановок и сложений с ключом.
- При эксплуатации блочного шифра необходимо уточнять режим его использования. Простейший режим — это *ECB*, который, к сожалению, имеет недостатки. Поэтому, как правило, используется более подходящий режим, а именно, *CBC*.
- Поточные шифры могут быть получены нелинейным комбинированием простых генераторов битовых строк, называемых

РСЛОС. При этом получаются довольно быстрые шифры, удобные для реализации с помощью аппаратных средств, которые шифруют информацию (например, изображение или звук) в реальном времени.

Дополнительная литература

Описание алгоритма *Rijndael*, представление о *AES* и подробное обсуждение атаки на блочный шифр (в частности и на *Rijndael*) можно найти в книге Димена и Риймена. В учебнике Стинсона, предназначенном студентам, наилучшим образом объясняется дифференциальный анализ. В книге Шнайера содержится очень много сведений о блочных и поточных шифрах.

J. Daemen and V. Rijmen. *The Design of Rijndael: AES — The Advanced Encryption Standard*. Springer-Verlag, 2002.

D. Stinson. *Cryptography Theory and Practice*. CRC Press, 1995.

B. Schneier. *Applied Cryptography*. Wiley, 1996.

Контрольные вопросы

5.1.1. В чем состоит принцип Керкхоффа?

5.1.2. Опишите операции шифра Фейстеля.

5.1.3. В чем главное отличие криптосистемы *DES* от оригинального шифра Фейстеля?

5.1.4. Что меняется в шифре Фейстеля при переходе от шифрующей функции к расшифровывающей?

5.1.5. Опишите операции тройного *DES*.

5.1.6. В чем состоит роль замен и перестановок в шифре *DES*?

5.1.7. В какой части алгоритма *DES* перестановки усиливают криптостойкость?

5.1.8. В чем состоит роль замен и перестановок в шифре *Rijndael*?

5.1.9. Назовите слабые стороны режима шифрования *ECB* и скажите, как они исправлены в режиме *CBC*.

5.1.10. Что такое *MAC*? Приведите три примера, в которых могли бы использоваться эти коды.

5.1.11. *РСЛОС* часто используется для создания поточных шифров. Что такое *РСЛОС* и почему их нельзя непосредственно применять для поточного шифрования?

Лабораторные работы

5.2.1. Найдите доступную реализацию алгоритма *DES* в Интернете. Сравните описание этого алгоритма в тексте книги с фактической. Объясните, почему изменения, внесенные программистами в алгоритм, корректны (т. е. и не меняют выходных данных) и для чего они были сделаны.

5.2.2. Выясните, как вычислить линейную сложность последовательности, и разработайте соответствующий алгоритм. С помощью *РСЛОС* произведите несколько последовательностей и вычислите их линейную сложность.

Упражнения

5.3.1. Пусть $\overline{a}$ обозначает поразрядное дополнение битовой строки[1] a. Покажите, что если E_k — шифрующая функция алгоритма *DES* с ключом k и $C = E_k(m)$, то $\overline{C} = E_{\overline{k}}(\overline{m})$.

Указание. Используйте тождество $\overline{a \oplus b} = \overline{a} \oplus b$.

5.3.2. Рассмотрим композицию шифрующих функций блочного шифра с n-битовым ключом

$$C = E_{k_1}(E_{k_2}(m))$$

как шифрующую функцию нового шифра с ключом из $2n$ битов. Покажите, что существует успешная атака с выбором открытого текста на этот шифр, для которой нужно $O(2^n)$ памяти и $O(2^n)$ операций шифрования и расшифрования с использованием функции E.

5.3.3. Рассмотрим модификацию алгоритма *DES* с двумя ключами вместо трех, а именно,

$$C = DES_{k_1}(DES_{k_2}^{-1}(DES_{k_1}(m))).$$

Найдите успешную атаку с выбором открытого текста на этот двухключевой *DES*, которой требуется около 2^{56} шагов

[1] Результат поразрядного сложения по модулю 2 строки a со строкой единиц. Если, например, $a = 100110$, то $\overline{a} = 011001$. — *Прим. перев.*

и хранения в памяти 2^{56} шифротекстов, полученных с помощью однократного *DES*.

5.3.4. Опишите операцию расшифрования в режиме *CFB*.

5.3.5. Пусть многочлен $C(X)$, ассоциированный с *РСЛОС*, неприводим, и старший его коэффициент равен 1. Покажите, что в этом случае последовательность, генерируемая *РСЛОС*, периодична с самого начала. Более того, при любом его начальном состоянии период получаемой последовательности равен наименьшему целому числу N, при котором $C(X)$ делит многочлен $1 + X^N$.

ГЛАВА 6

РАСПРЕДЕЛЕНИЕ СИММЕТРИЧНЫХ КЛЮЧЕЙ

Цели главы

- Понять проблемы, связанные с управлением секретными ключами и их распределением.
- Узнать о технике распределения ключей, основанной на различных протоколах.
- Ввести понятие формального анализа протоколов.

6.1. Управление ключами

Для успешного использования симметричных криптосистем подобных *DES* или *Rijndael* партнерам необходимо как-то договориться о секретном ключе, т. е. найти путь управления ключами. В этом параграфе мы обсудим некоторые способы решения этой задачи, в частности здесь будет рассказано

- о распределении ключей,
- о выборе ключа,
- о времени жизни ключа,
- о разделении секрета.

Но сначала нам необходимо разделить все многообразие симметричных ключей на отдельные группы и понять, чем они друг от друга отличаются. Введем термины.

Статичный (долговременный) ключ. Так называют ключ, который используется в течение большого периода времени. Точное значение слова «большого» зависит от приложений, где ключ используется, и период, о котором идет речь, может варьироваться от нескольких часов до нескольких лет. Компрометация (раскрытие) статичного ключа обычно считается главной проблемой с потенциально катастрофическими последствиями.

Эфемерный или сеансовый (кратковременный) ключ применяется лишь малое время, от нескольких секунд до одного дня. Его обычно берут на вооружение для обеспечения конфиденциальности в одном сеансе связи. Раскрытие сеансового ключа может повлечь за собой лишь нарушение секретности сеанса и ни коим образом не должно влиять на криптостойкость всей системы.

6.1.1. Распределение ключей

Распределение ключей — одна из фундаментальных задач криптографии. Существует несколько ее решений, подходящее из которых выбирается в зависимости от ситуации.

Физическое распределение. С помощью доверенных курьеров или вооруженной охраны ключи могут рассылаться традиционным физическим путем. До семидесятых годов двадцатого века это действительно был единственный безопасный путь распределения ключей при установке системы. Ему сопутствовал ряд трудностей, в особенности при расширении, масштабировании (модульном наращивании системы в рамках унифицированной архитектуры) криптосистемы, но основной недостаток, связанный с таким способом распределения, состоит в том, что криптостойкость системы зависит не столько от ключа, сколько от курьера. Если подкупить, похитить или просто убить курьера, то система будет скомпрометирована.

Распределение с помощью протоколов с секретным ключом. Если долговременные секретные ключи распределены между пользователями и неким центром, который обычно называют *центром доверия*, то его можно использовать для генерирования ключей и обмена между любыми двумя пользователями всякий раз, когда в этом возникает необходимость. Протоколы, предназначенные для этой цели, — предмет обсуждения настоящей главы. Обычно они достаточно эффективны, но не лишены и недостатков. В частности, этот способ распределения предусматривает, что как оба пользователя, так и центр работают в режиме онлайн. Кроме того, статичные ключи при этом должны распределяться физическим путем.

Распределение с помощью протоколов с открытым ключом. Используя криптосистемы с открытым ключом, партнеры, не доверяющие посредникам и лишенные возможности встретиться, могут договориться об общем секретном ключе в режиме онлайн в соответствии с протоколом об обмене ключей. Это наиболее распространенное приложение техники шифрования с открытым ключом. Вместо

того, чтобы шифровать большой объем данных непосредственно с помощью открытого ключа, стороны предварительно согласовывают секретный ключ. Затем для шифрования фактической информации применяется симметричный шифр с согласованным ключом.

Чтобы понять масштабность проблемы, отметим, что при обслуживании n пользователей, обменивающихся закрытой информацией друг с другом, необходимо

$$\frac{n(n-1)}{2}$$

разных секретных ключей. С ростом n возникает проблема управления огромным числом ключей. Например, для небольшого университета с 10 000 студентов нужно около пятидесяти миллионов отдельных секретных ключей. С большим количеством уже существующих ключей связано много проблем. Например, к чему приведет компрометация Вашего ключа? Другими словами, кто-то посторонний нашел Ваш ключ. Что Вам следует предпринять в связи с этим?... Итак, большое число ключей порождает сложную проблему управления.

Одно из ее решений заключается в том, что за каждым пользователем закрепляется единственный ключ, используя который он может связываться с центром доверия. В этом случае система с n пользователями требует только n ключей. Когда двое пользователей хотят обменяться секретными сведениями, они генерируют ключ, который будет использован только для передачи этого сообщения. Его называют *сеансовым ключом*. Сеансовый ключ генерируется с участием центра доверия при помощи одного из протоколов, о которых будет рассказано позже в этой главе.

6.1.2. Выбор ключа

Секретный ключ должен быть случайным в полном смысле этого слова, поскольку иначе, как мы убедились в главе 4, нападающий может получить информацию о ключе, зная вероятностные распределения ключей и сообщений. Все ключи должны быть равновероятны и производиться с помощью настоящего генератора случайных чисел. Однако источник абсолютно случайных чисел очень трудно создать. Стоит заметить, что хотя по-настоящему случайный ключ очень хорош для применения, его крайне трудно удержать в человеческой памяти. Поэтому многие системы используют пароль или подходящие фразы для генерирования секретного ключа. Но теперь лобовая атака даже более опасна. Как видно из табл. 6.1, пароль на-

подобие PIN-кода, т. е. простой номер, лежащий в пределах от 0 до 9999, легко установить с помощью лобовой атаки. Даже при использовании пароля из 8 символов число возможностей не дотягивает до 2^{80}, чего нам хотелось бы для обеспечения безопасности.

Таблица 6.1. Зависимость числа ключей от длины и типа используемых знаков

Размер ключа	Цифры	Буквы
4	$10^4 \approx 2^{13}$	$10^7 \approx 2^{23}$
8	$10^8 \approx 2^{26}$	$10^{15} \approx 2^{50}$

Можно было бы использовать длинные фразы, состоящие из 20–30 знаков, однако это тоже не выход, поскольку, как мы уже смогли убедиться, последовательность букв в естественном языке далеко не случайна.

Короткие пароли, основывающиеся на именах или словах — общая проблема большинства крупных организаций. Многие из них требуют соблюдения, чтобы в пароле присутствовали

- по крайней мере одна прописная буква,
- по крайней мере одна заглавная буква,
- по крайней мере одна цифра,
- по крайней мере один знак, отличный от цифры и буквы,
- длина пароля — не менее восьми знаков.

Но перечисленные правила, исключая атаку по словарю, все еще не обеспечивают максимально возможное число паролей, которое достигается при действительно случайном выборе восьми знаков.

6.1.3. Время жизни ключа

Один из важных моментов, который следует принимать во внимание при генерировании и хранении ключей, — это продолжительность их жизни. Общее правило состоит в том, что чем дольше используемый ключ находится в обращении, тем легче нападающему его вскрыть, и тем большую ценность он для него представляет. Кроме того, важно правильно уничтожить ключ по истечении его жизни. Перекладывание проблемы на плечи операционной системы командой `del` или `rem` не гарантирует того, что нападающий не сможет восстановить информацию, исследуя жесткий диск. Обычно удаление файла не стирает его содержания, а всего лишь сообщает системе о том, что ячейки памяти, отведенные под него, теперь свободны для записи новых данных.

6.1.4. Разделение секрета

Как мы уже говорили, главная проблема — безопасное управление распределением секретных ключей. Даже при использовании центра доверия необходим какой-нибудь способ получения ключа каждому его пользователю.

Один из возможных путей решения состоит в расщеплении ключа (более формально — *разделении секрета*), при котором ключ делится на несколько частей

$$k = k_1 \oplus k_2 \oplus \cdots \oplus k_r.$$

Каждая его часть передается по своему каналу. Красота этого решения бросается в глаза: для определения ключа нападающий должен суметь подключиться ко всем каналам сразу. С другой стороны, если противнику удалось проникнуть в один из каналов, передающих части ключа, он может воспрепятствовать законному восстановлению ключа.

Более сложный метод, лишенный последнего недостатка, состоит в использовании одной из форм *схемы порогового разделения секрета*. Ключ, как и прежде, разделяется на несколько частей, например, W. Легальный пользователь сможет восстановить ключ полностью, получив некоторое количество этих частей, превышающее определенное пороговое значение T. Однако противник, выведав только $(T-1)$ часть, не сможет вскрыть ключ. Эта схема на данный момент не часто применяется на практике. Мы ввели эту концепцию здесь, дабы подготовить читателя к лучшему восприятию различных схем разделения секрета в последующих главах, когда будем рассматривать протокол голосования.

Схема Шамира разделения секрета является типичным примером пороговой схемы. Предположим, что ключ k разделяется на W частей таким образом, что по T из них, собранных вместе, ключ однозначно восстанавливается. Схема с такими значениями называется (T, W)-пороговой схемой.

Берем простое число P, большее чем $W+1$. Ключ k — элемент поля $\mathbb{F}_P$. Доверенное лицо выбирает значения $X_i \in \mathbb{F}_P$ для $i = 1, \ldots, W$ по одному для каждой части ключа. Каждый участник разделения секрета получает свое значение X_i, которое будет известно и всем остальным участникам. Для разделения ключа k между пользователями ответственное лицо выбирает $T-1$ элемен-

тов поля $a_1, \ldots, a_{T-1}$ и строит многочлен

$$F(X) = k + \sum_{j=1}^{T-1} a_j X^j.$$

После этого вычисляются его значения

$$y_i = F(X_i) \quad \text{при } 1 \leqslant i \leqslant W$$

и раздаются участникам разделения ключа (они держатся в секрете).

Чтобы восстановить ключ, пользователи применяют процедуру интерполяции многочлена. Предположим, что L хранителей секрета собрались вместе и обменялись значениями y_i $(i = 1, \ldots, L)$. В этом случае они могут попытаться решить систему уравнений:

$$\begin{cases} y_1 = k + a_1 X_1 + \cdots a_{T-1} X_1^{T-1}, \\ \ldots \\ y_L = k + a_1 X_L + \cdots a_{T-1} X_L^{T-1}. \end{cases}$$

Если $L \geqslant T$, то система будет иметь единственное решение, которое позволит восстановить $F(X)$, а значит и ключ. Если же $L < T$, то система получится *неопределенной* и никак не поможет восстановить нужный многочлен. Таким образом, никакой информации о ключе k извлечь не удастся.

На практике применяется короткий способ решения этой системы с помощью *интерполяционного многочлена Лагранжа*. Не вдаваясь в подробное описание этого многочлена, приведем лишь схему восстановления ключа. Вычисляются коэффициенты

$$B_j = \prod_{1 \leqslant \alpha \leqslant T,\, \alpha \neq j} \frac{X_\alpha}{X_\alpha - X_j},$$

и по ним восстанавливается ключ:

$$k = \sum_{j=1}^{T} B_j y_j.$$

6.2. Распределение секретных ключей

Напомним, что n пользователям, желающим обмениваться закрытой информацией друг с другом, необходимо

$$\frac{n(n-1)}{2}$$

разных долговременных криптографических пар[1]. Как было отмечено ранее, это порождает проблемы управления огромным числом ключей и их распределения. Мы уже говорили, что лучше использовать сеансовые ключи и несколько статичных, но не объяснили как развертывается сеансовый ключ.

Для решения этой задачи разработано множество протоколов, в которых используется криптография с симметричным ключом для распределения сеансовых ключей. Некоторые из них будут представлены в этом параграфе. Позже мы познакомимся с решением проблемы распределения, которое предлагают криптосистемы с открытым ключом и которое часто бывает весьма элегантным.

6.2.1. Обозначения

При описании протоколов нам потребуются некоторые стандартные обозначения. Сначала договоримся о символах, отвечающих за участников протокола и встречающиеся величины.

Участники/администратор: A, B, S. Будем предполагать, что в обмене закрытой информацией участвуют двое: A — Алиса и B — Боб. Кроме того, предполагается, что они прибегают к услугам доверенного лица (ДЛ), которое будем обозначать буквой S.

Долговременные секретные ключи: k_{ab}, k_{bs}, k_{as}. Символ k_{ab} закреплен за ключом, известным только A и B.

Числовые вставки: n_a, n_b. Это случайные одноразовые числа, уникальные для каждого сообщения протокола. Число n_a обозначает числовую вставку, произведенную участником A. Заметим, что эти числа могут обозначаться и по-другому.

Временна́я метка: t_a, t_b, t_s. Величина t_a — временна́я метка, созданная участником A. При использовании временно́й метки мы предполагаем, что участники пытаются соблюдать синхронизацию часов, используя какой-то другой протокол.

Символьная запись

$$A \longrightarrow B : M,\ A,\ B,\ \{n_a, m, a, b\}_{k_{as}}$$

означает, что A посылает участнику B сообщение, состоящее из

- числовой вставки M,
- имени посылающего сообщение A,
- имени адресата B,

[1] Используемые в шифровании открытым ключом общедоступный ключ и личный ключ. — *Прим. перев.*

– текста послания $\{n_a, m, a, b\}$, зашифрованного с помощью ключа k_{as}, используемого A совместно с S. Таким образом, получатель B не в состоянии прочесть зашифрованную часть этого письма.

Перед описанием первого из протоколов нам следует определить его цели. Предположим, что заинтересованные стороны A и B пользуются ключами k_{as} и k_{bs} для связи с центром доверия S, а в результате работы протокола они хотели бы договориться о ключе k_{ab} и получить его в свое распоряжение для обмена закрытой информацией друг с другом.

Кроме этого, нам необходимо предусмотреть возможные атаки на протокол. Как всегда, будем учитывать самую плохую ситуацию, при которой нападающий может перехватывать любое сообщение, переданное по сети, приостанавливать, вносить свою правку или переадресовывать его. Будем считать также, что атакующий способен передать по сети и свое собственное сообщение. Противника, наделенного такими большими возможностями, принято отождествлять с самой сетью.

Сеансовый ключ, о котором договариваются A и B, должен быть новым, т. е. только что созданным, никогда не применявшимся ни одной из договаривающихся сторон. Новизна ключа помогает переиграть противника, поскольку получив сообщение, зашифрованное старым ключом, легко понять, что оно подложное. Новизна ключа также подтверждает, что сторона, с которой Вы ведете закрытую переписку, все еще дееспособна.

6.2.2. Протокол широкоротой лягушки

Первый протокол, с которым мы познакомимся — это протокол широкоротой лягушки, предложенный Барроузом. Протокол передает ключ k_{ab} от A к B через посредника S, используя лишь 2 сообщения, но имеет множество недостатков. В частности, для его реализации необходима синхронизация часов, что создает дополнительные проблемы. В нем предполагается, что A выбирает сеансовый ключ k_{ab} и пересылает его пользователю B. Это означает, что пользователь B верит в компетентность A, в его способность создать стойкий ключ и хранить его в секрете. Такое сильное требование служит основной причиной слабого применения на практике протокола широкоротой лягушки. С другой стороны, это довольно простой и хороший пример, на котором можно продемонстрировать формальный анализ протоколов, чему, в частности, посвящена глава.

Протокол состоит из обмена двумя сообщениями (рис. 6.1):

$$A \longrightarrow S : A, \{t_a, b, k_{ab}\}_{k_{as}},$$
$$S \longrightarrow B : \{t_s, a, k_{ab}\}_{k_{bs}}.$$

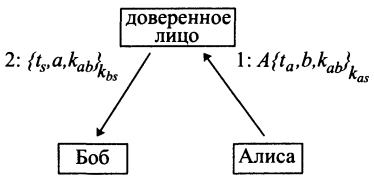


Рис. 6.1. Протокол широкоротой лягушки

Получив первое послание, центр доверия S расшифровывает последнюю его часть и проверяет, что временна́я метка близка к текущему моменту времени. Расшифрованное сообщение говорит S о том, что тому следует переслать ключ k_{ab} пользователю B. Если временная метка соответствует недавнему времени, то S шифрует требуемый ключ вместе со своей временно́й меткой и пересылает полученный шифротекст пользователю B. При получении сообщения от S участник переписки B расшифровывает его и проверяет свежесть временно́й метки. После этого он может прочесть ключ k_{ab} и имя человека A, который хочет переслать ему зашифрованную информацию.

Корректная временна́я метка означает, что сеансовый ключ был создан недавно. Однако пользователь A мог сгенерировать этот ключ годы назад и хранить его на своем жестком диске, куда Ева имела возможность забраться несколько раз и снять копию ключа.

Мы уже говорили, что протокол широкоротой лягушки корректно работает только при синхронизированных часах всех его участников. Однако это не создает больших сложностей, поскольку доверенный центр S проверяет или генерирует все временны́е метки, используемые в протоколе. Поэтому остальные стороны должны лишь записать разницу в показаниях своих часов и часов центра. Тем самым протокол будет работать некорректно лишь тогда, когда какие-то из трех часов идут медленнее или быстрее остальных, или же показания часов были изменены принудительно.

Этот протокол действительно очень прост, что обусловливается синхронизацией часов и предположением о том, что участнику A можно доверить генерирование сеансового ключа.

6.2.3. Протокол Нидхейма–Шредера

Рассмотрим более сложные протоколы, начав с одного из самых знаменитых, а именно, с протокола Нидхейма–Шредера. Этот протокол был разработан в 1978 году и является самым изучаемым на сего-

дняшний день. Он получил известность благодаря тому, что даже самый простой протокол может долгое время скрывать свои пробелы в обеспечении безопасности. Обмен письмами идет по следующей схеме (рис. 6.2):

$$
\begin{aligned}
&A \longrightarrow S : A,\ B,\ n_a,\\
&S \longrightarrow A : \{n_a, b, k_{ab}, \{k_{ab}, a\}_{k_{bs}}\}_{k_{as}},\\
&A \longrightarrow B : \{k_{ab}, a\}_{k_{bs}},\\
&B \longrightarrow A : \{n_b\}_{k_{ab}},\\
&A \longrightarrow B : \{n_b - 1\}_{k_{ab}}.
\end{aligned}
$$

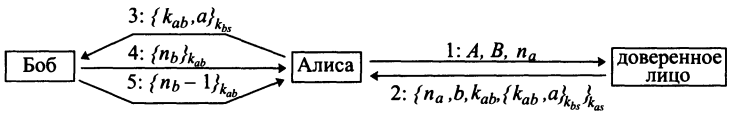


Рис. 6.2. Протокол Нидхейма–Шредера

Теперь мы разберем каждое из сообщений протокола подробно и объясним их предназначение.

- В первом пользователь A сообщает центру доверия S, что он намерен получить ключ для переписки с B. Обратите внимание на то, что к сообщению прикреплена уникальная числовая вставка, созданная A.
- S генерирует ключ k_{ab} и посылает его A вторым письмом. В него включается числовая вставка n_a, по которой клиент A узнает, что полученное сообщение было послано в ответ на его запрос. Сеансовый ключ зашифровывается с помощью k_{bs} и прикрепляется к этому сообщению.
- В третьем письме сеансовый ключ пересылается пользователю B.
- Участник B должен проверить, что отправителем этого послания действительно является A, т. е. он должен удостовериться, что A все еще действует, и в четвертом сообщении протокола он пересылает свою числовую вставку участнику A в зашифрованном виде.
- В последнем сообщении, чтобы убедить партнера B в своей дееспособности, инициатор переговоров шифрует простое выражение, зависящее от n_b, и отсылает его B.

Основной недостаток протокола Нидхейма–Шредера заключается в том, что в результате его работы у пользователя B нет оснований считать, что полученный ключ является новым, — факт, который

был замечен только спустя некоторое время после опубликования протокола. Противник, найдя сообщения и ключ предыдущих сеансов, может использовать старые письма вместо последних трех сообщений, в которых упоминается B. Таким образом нападающий может обмануть B, вынуждая его принять свой ключ, в то время, как B предполагает, что ведет переговоры с A.

Заметим, что A и B принимают секретный сеансовый ключ, созданный центром доверия, в связи с чем ни у одной из сторон нет необходимости полагаться на другую в выборе хорошего ключа, поскольку вся ответственность здесь ложится на плечи S. В некоторых протоколах обходятся без посредников, а привлекаются другие алгоритмы, например, с открытым ключом. В этой главе мы будем считать, что все участники протоколов доверяют S во всех функциях, которые на него возложены.

6.2.4. Протокол Отвэй–Риса

Протокол Отвэй–Риса практически не используется начиная с 1987 года, но он важен с исторической точки зрения. Аналогично протоколу Нидхейма–Шредера, в нем не требуют синхронизации часов, но и он не лишен недостатков.

Как и ранее, два пользователя пытаются достигнуть договоренности о ключе через посредничество центра доверия S. Здесь участвуют числовые вставки n_a и n_b, свидетельствуя о свежести всех шифрованных компонентах сообщений. Кроме того, числовая вставка M связывает сообщения одного сеанса между собой. Протокол Отвэй-Риса короче протокола Нидхейма–Шредера, поскольку он состоит только из четырех сообщений. Однако эти сообщения выглядят совсем по-другому. Как и прежде, центр доверия генерирует ключ k_{ab} для двух пользователей.

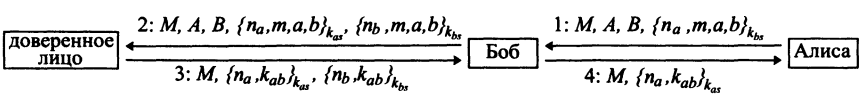


Рис. 6.3. Протокол Отвэй-Риса

Протокол Отвэй-Риса состоит из следующих этапов (рис. 6.3):

$$A \longrightarrow B:\ M,\ A,\ B,\ \{n_a, m, a, b\}_{k_{as}},$$
$$B \longrightarrow S:\ M,\ A,\ B,\ \{n_a, m, a, b\}_{k_{as}},\ \{n_b, m, a, b\}_{k_{bs}},$$
$$S \longrightarrow B:\ M,\ \{n_a, k_{ab}\}_{k_{as}},\ \{n_b, k_{ab}\}_{k_{bs}},$$
$$B \longrightarrow A:\ M,\ \{n_a, k_{ab}\}_{k_{as}}.$$

Поскольку протокол не использует ключ k_{ab} для шифрования сообщений, ни одна из сторон не представляет, известен ли этот ключ другому участнику. Подчеркивая эту особенность, говорят, что протокол Отвэй–Риса не содержит подтверждения ключа. Посмотрим, что знают договаривающиеся стороны. A понимает, что B послал сообщение, содержащее числовую вставку n_a, в новизне которой пользователь A уверен, поскольку именно он был ее создателем. Следовательно, свое письмо B тоже должен был послать недавно. С другой стороны, сервер уведомляет участника B о числовой вставке, включенной в сообщение клиентом A, но у B нет никаких оснований полагать, что полученное им послание не было повторением старого сообщения.

6.2.5. Цербер

Закончим параграф рассмотрением Цербера, системы аутентификации, основанной на симметричном шифровании с помощью центра аутентификации. В его основу легли идеи протокола Нидхейма–Шредера. Цербер разработан в Массачуссетском технологическом институте (MIT) в 1987 г. как часть проекта «Афина». Модифицированная версия этого протокола используется в операционной системе *Windows 2000*.

Предполагается, что компьютерная сеть состоит из клиентов и сервера, причем клиентами могут быть пользователи, программы или специальные службы. Цербер хранит центральную базу данных, включающую как клиентов, так и их секретные ключи. Таким образом, если в систему входит n клиентов, размер пространства ключей должен иметь порядок $O(n)$. Цель Цербера состоит в идентификации клиентов и генерировании для них сеансовых ключей.

Кроме того, Цербер может служить системой предоставления доступа к различного вида услугам и ресурсам. Разделение функций идентификации и предоставления доступа — хорошая идея, с которой мы будем разбираться позже, знакомясь с *SPKI*. Такое разделение отражает обычное положение дел в реальных фирмах. Действительно, персонал одного отдела, например, устанавливает Вашу личность, в то время как другой отдел проверяет Ваш уровень доступа к ресурсам компании. Такое же разделение функций присутствует и в системе Цербер, где имеется центр идентификации и сервер генерирования сертификатов (*TGS* — ticket generation server), который выдает сертификаты (разрешения) на доступ к ресурсам: файлам, принтерам, и т. д.

Предположим, клиент A хочет воспользоваться ресурсами клиента B. Тогда он заходит на сервер аутентификации, используя свой пароль. Сервер выдает ему сертификат, зашифрованный с помощью этого пароля. Сертификат, в частности, содержит сеансовый ключ k_{as}. Клиент A применяет ключ k_{as} для получения следующего сертификата, разрешающего доступ к ресурсам клиента B. Последний сертификат состоит из ключа k_{ab}, времени его жизни l и временно́й метки t_s. Выданный сертификат используется для «удостоверения личности» A в последующем обращении к B. Обмен информацией в системе Цербер представлен ниже (рис. 6.4):

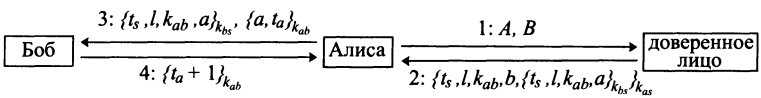


Рис. 6.4. Цербер

$$
\begin{aligned}
&A \longrightarrow S:\ A, B,\\
&S \longrightarrow A:\ \{t_s, l, k_{ab}, b,\ \{t_s, l, k_{ab}, a\}_{k_{bs}}\}_{k_{as}},\\
&A \longrightarrow B:\ \{t_s, l, k_{ab}, a\}_{k_{bs}},\ \{a, t_a\}_{k_{ab}},\\
&B \longrightarrow A:\ \{t_a + 1\}_{k_{ab}}.
\end{aligned}
$$

- В первом обращении клиент A сообщает S, что он хотел бы связаться с B.
- Если S разрешает эту связь, то создает сертификат $\{t_s, l, k_{ab}, a\}$, зашифрованный ключом k_{bs}, и отсылает его A для передачи B. Пользователь A получает копию этого ключа в той форме, которую он может прочесть.
- Клиент A, желая проверить действенность сертификата и свою возможность воспользоваться ресурсами B, посылает зашифрованную временну́ю метку t_a участнику B.
- Клиент B отсылает назад зашифрованную величину $t_a + 1$, проверив, что временна́я метка является свежей, показывая тем самым, что он знает сеансовый ключ и готов к связи.

В этом протоколе устранены недостатки, присущие протоколу Нидхейма – Шредера, но за счет обязательной синхронизации часов.

6.3. Формальные методы проверки протоколов

Из обсуждения протоколов в предыдущем параграфе можно сделать вывод, что они весьма запутанны, и выявление их недостатков ока-

зывается довольно тонким делом. Чтобы подойти к этой задаче с научной точки зрения, необходимо разработать соответствующий аппарат. Наиболее сильный из них — *BAN*-логика, изобретенная Барроузом, Абади и Нидхеймом.

BAN-логика, как и любая формальная логика, имеет ряд недостатков, но она была в свое время очень полезна при проектировании и анализе протоколов распределения симметричных ключей типа Цербер и Нидхейма–Шредера. В настоящее время она вытеснена более сложной техникой и формальными методами, но сохраняет свое значение как с исторической, так и с учебной точек зрения.

Основная идея *BAN*-логики состоит в том, что при анализе протоколов в первую очередь стоит обратить внимание на восприятие сторонами поступающей информации — что они принимают на веру, а что им доподлинно известно или может быть выведено логическим путем из достоверных для них фактов. Даже при современных подходах к моделированию инфрасруктуры открытых ключей (*PKI*) берут на вооружение эту идею, в связи с чем мы сейчас исследуем *BAN*-логику более подробно.

Введем обозначения.

- $P| \equiv X$ означает, что участник P *верит* в (или имеет право брать на веру) высказывание X, т. е. P станет действовать так, как будто X — истинная информация.
- $P \lhd X$ подразумевает, что P *видит* X, т. е. кто-то послал сообщение пользователю P, в котором содержится X. Так что P может прочесть и воспроизвести X.
- $P| \sim X$ означает, что P *однажды высказал* X и в тот момент верил в его истинность. Заметим, что здесь мы не уточняем время этого события.
- $P| \Rightarrow X$ означает, что X входит в *юрисдикцию* P, т. е. P обладает властью над X и по этому вопросу P можно доверять.
- $\#X$ означает, что высказывание X получено недавно. Этот символ используется обычно для числовых вставок.
- $P \overset{k}{\longleftrightarrow} Q$ означает, что P и Q используют для общения *разделенный ключ* k. Предполагается, что ключ достаточно стоек и не может быть раскрыт никем из посторонних, если это не предусмотрено протоколом.
- Обозначение $\{X\}_k$ обычно подразумевает, что данные X зашифрованы ключом k. Шифрование считают совершенным, в частности X останется засекреченным до тех пор, пока од-

на из сторон сознательно не раскроет его в какой-то другой части протокола.

Кроме того, вместо обычной логической операции «И» употребляется запятая, а запись

$$\frac{A, B}{C}$$

означает, что из истинности утверждений A и B следует истинность высказывания[1] C. Такой способ символьного изображения сложных конструкций принят во многих формальных логиках.

BAN-логика опирается на множество постулатов или правил вывода. Мы разберем лишь главные из них.

Правило о значении сообщения:

$$\frac{A| \equiv A \xleftrightarrow{K} B,\ A \lhd \{X\}_K}{A| \equiv B| \sim X}$$

Эквивалентная словесная формулировка: из предположений о том, что A верит в совместное использования ключа K с B, и A видит сообщение X, зашифрованное ключом K, мы делаем вывод: A верит, что B в какой-то момент высказал X. Заметим, что здесь неявно предполагается, что сам A никогда не высказывал X.

Правило проверки уникальности числовых вставок:

$$\frac{A| \equiv \#X,\ A| \equiv B| \sim X}{A| \equiv B| \equiv X},$$

т. е. если A верит в новизну X и в то, что B когда-то высказал X, то A верит, что B по-прежнему доверяет X.

Правило юрисдикции

$$\frac{A| \equiv B| \Rightarrow X,\ A| \equiv B| \equiv X}{A| \equiv X}$$

говорит, что если A верит в полномочия B относительно X, т. е. A полагается на B в деле с X, и A верит в то, что и B верит X, то A должен верить в X.

Другие правила. Оператор доверия («≡») и запятая, разделяющая высказывания, рассматриваемые совместно, подчиняются следующим соотношениям:

$$\frac{P| \equiv X,\ P| \equiv Y}{P| \equiv (X,Y)}, \qquad \frac{P| \equiv (X,Y)}{P| \equiv X}, \qquad \frac{P| \equiv Q| \equiv (X,Y)}{P| \equiv Q| \equiv X}.$$

[1] Часто в литературе можно встретить эквивалентное обозначение: A И $B \Rightarrow C$, которое читается так: высказывания A и B влекут C. — *Прим. перев.*

Оператор «однажды высказал» удовлетворяет аналогичному соотношению:

$$\frac{P| \equiv Q| \sim (X,Y)}{P| \equiv Q| \sim X}.$$

Заметим, что предположения $P| \equiv Q \sim X$ и $P| \equiv Q \sim Y$ не влекут утверждения $P| \equiv Q \sim (X,Y)$, поскольку последнее означает, что Q произнес X и Y в одно и то же время. Наконец, если какая-то часть высказывания получена недавно, то это же можно утверждать и про всю конструкцию:

$$\frac{P| \equiv \#X}{P| \equiv \#(X,Y)}.$$

Попытаемся теперь проанализировать протокол выбора ключа для A и B, опираясь на BAN-логику. Прежде всего сформулируем цели протокола, которые как минимум состоят в следующем:

$$A| \equiv A \xleftrightarrow{K} B \quad \text{и} \quad B| \equiv A \xleftrightarrow{K} B,$$

т. е. оба участника должны поверить в то, что они нашли секретный ключ для обмена друг с другом закрытой информацией.

Однако можно было бы потребовать и большего, например,

$$A| \equiv B| \equiv A \xleftrightarrow{K} B \quad \text{и} \quad B| \equiv A| \equiv A \xleftrightarrow{K} B,$$

что обычно называют *подтверждением приема* ключа. Иначе говоря, можно считать, что в результате работы протокола A будет уверен в знании B о том, что он разделяет секретный ключ с A, а B верит в то, что и A знает об их общем ключе.

Перепишем протокол в символьной форме согласно правилам BAN-логики. Этот процесс называется *идеализацией* и наиболее уязвим для ошибок, поскольку его невозможно автоматизировать. Кроме того, нам необходимо сделать предположения или сформулировать постулаты, которые считаются истинными в начале работы протокола.

Для наглядности, чтобы понять, как это все работает «в реальной жизни», подвергнем анализу протокол широкоротой лягушки, использующий синхронизацию часов.

6.3.1. Анализ протокола широкоротой лягушки

Напомним, что он состоит из обмена двумя сообщениями:

$$A \longrightarrow S:\ A, \{t_a, b, k_{ab}\}_{k_{as}},$$
$$S \longrightarrow B:\ \{t_s, a, k_{ab}\}_{k_{bs}}.$$

При идеализации это выглядит так:

$$A \longrightarrow S:\ \{t_a, A \overset{k_{ab}}{\longleftrightarrow} B\}_{k_{as}},$$

$$S \longrightarrow B:\ \{t_s, A| \equiv A \overset{k_{ab}}{\longleftrightarrow} B\}_{k_{bs}}.$$

Идеализированное первое сообщение говорит S, что

- t_a — временна́я или числовая вставка,
- k_{ab} — ключ, который предлагается использовать для коммуникации с B.

Итак, какие предположения сделаны в начале работы протокола? Ясно, что A, B и S используют секретные ключи для обмена шифрованными сообщениями друг с другом, что на языке BAN-логики может быть выражено как

$$A| \equiv A \overset{k_{as}}{\longleftrightarrow} S, \qquad S| \equiv A \overset{k_{as}}{\longleftrightarrow} S$$
$$B| \equiv B \overset{k_{bs}}{\longleftrightarrow} S, \qquad B| \equiv A \overset{k_{bs}}{\longleftrightarrow} S.$$

Есть пара предположений о временны́х вставках:

$$S| \equiv \#t_a \quad \text{и} \quad B| \equiv \#t_s$$

и еще три предположения:

- B полагается на A в выборе хорошего ключа:

$$B| \equiv (A| \Rightarrow A \overset{k_{ab}}{\longleftrightarrow} B),$$

- B доверяет S передать ключ от A:

$$B| \equiv (S| \Rightarrow A| \equiv A \overset{k_{ab}}{\longleftrightarrow} B),$$

- A верит, что сеансовый ключ принят:

$$A| \equiv A \overset{k_{ab}}{\longleftrightarrow} B.$$

Обратите внимание на то, как последние предположения обнажают проблемы, связанные с протоколом, о которых мы говорили ранее.

Опираясь на сделанные предположения, мы можем проанализировать протокол. Посмотрим, какой вывод можно сделать из первого сообщения

$$A \longrightarrow S:\ \{t_a, A \overset{k_{ab}}{\longleftrightarrow} B\}_{k_{as}}.$$

- S, видя сообщение, зашифрованное ключом k_{as}, может сделать вывод о том, что оно было послано клиентом A.
- Наличие свежей временной вставки t_a позволяет участнику S заключить, что и все сообщение написано недавно.

- Из свежести всего сообщения S выводит, что клиент A верил в то, что послал.
- Следовательно,

$$S| \equiv A| \equiv A \overset{k_{ab}}{\longleftrightarrow} B,$$

т. е. S подготовлен к посылке второго сообщения протокола.

Обратимся к следующему этапу:

$$S \longrightarrow B : \{t_s, a| \equiv A \overset{k_{ab}}{\longleftrightarrow} B\}_{k_{bs}}.$$

- Увидев послание, зашифрованное ключом k_{bs}, клиент B понимает, что оно было отправлено S.
- Временна́я вставка t_s доказывает B, что все сообщение было послано только что.
- Ввиду свежести сообщения, B заключает, что S доверяет всему посланному.
- В частности, B верит в то, что S доверяет второй части сообщения.
- Но B верит и в то, что в юрисдикцию S входит выяснить, знает ли его партнер A секретный ключ, и поэтому B вверяет A полномочия по генерированию ключа.

Из этих рассуждений можно сделать вывод:

$$B| \equiv A \overset{k_{ab}}{\longleftrightarrow} B \quad \text{и} \quad B| \equiv A| \equiv A \overset{k_{ab}}{\longleftrightarrow} B.$$

Комбинируя это с исходным предположением: $A| \equiv A \overset{k_{ab}}{\longleftrightarrow} B$, получаем, что анализируемый протокол распределения ключей обоснован. Единственное, чего мы не встретили в нем, это

$$A| \equiv B| \equiv A \overset{k_{ab}}{\longleftrightarrow} B,$$

т. е. A не добился подтверждения тому, что B получил нужный ключ.

Обратите внимание на то, что применение BAN-логики к анализу формализовало все этапы протокола, так что их стало легче сравнивать с предположениями, необходимыми любому протоколу для работы. Кроме того, этот формализм проясняет точки зрения на работу протокола всех его участников.

Краткое содержание главы

- Распределение секретных ключей — основная проблема, связанная с симметричными шифрами.
- Для распределения ключей по нескольким каналам можно использовать схему порогового разделения секрета.

- Многие существующие протоколы распределения ключей прибегают к посредничеству третьего лица (центра доверия) и симметричным алгоритмам шифрования. Эти протоколы требуют предварительной договоренности о долговременных ключах для обмена информацией клиентов с центром доверия и, иногда, синхронизации часов.
- Для анализа протоколов разработаны всевозможные аппараты. Наиболее важным из них считается BAN-логика. Она помогает вычленить четкие предположения и проблемы, связанные с протоколом.

Дополнительная литература

Удачное введение в современные схемы разделения секрета можно найти в одиннадцатой главе книги Стинсона. Статья Барроуза, Абади и Нидхейма — легко читаемое введение в BAN-логику с примерами анализа протоколов распределения ключей. Многое из рассказанного в этой главе основывается на этой работе.

M. Burrows, M. Abadi and R. Needham. *A Logic of Authentication*. Digital Equipment Corporation, SRC Research Report 39, 1990.

D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

Контрольные вопросы

6.1.1. Что собой представляет проблема распределения ключей?

6.1.2. Что такое схема порогового разделения секрета с порогом t?

6.1.3. Как числовые вставки используются в протоколах этой главы? Какие стороны безопасности протоколов они обеспечивают?

6.1.4. Объясните значения следующих символов BAN-логики:

$$P| \equiv X, \quad P \lhd X \quad \text{и} \quad P| \sim X.$$

6.1.5. Расскажите, почему перечисленные ниже правила вывода имеют силу постулатов.
Правило о значении сообщения:

$$\frac{A| \equiv A \overset{K}{\longleftrightarrow} B,\ A \lhd \{X\}_K}{A| \equiv B| \sim X}$$

Правило проверки уникальных числовых вставок:

$$\frac{A| \equiv \#X,\ A| \equiv B| \sim X}{A| \equiv B| \equiv X},$$

Правило юрисдикции:

$$\frac{A| \equiv B| \Rightarrow X,\ A| \equiv B| \equiv X}{A| \equiv X}$$

6.1.6. Почему в протоколе Нидхейма–Шредера мы не посылаем в качестве пятого сообщения сообщение вида:

$$A \longrightarrow B:\ \{n_b\}_{k_{ab}}?$$

Упражнения

6.2.1. Просмотрите еще раз наш анализ протокола широкоротой лягушки и осознайте, какие постулаты BAN-логики использовались на каждом из этапов анализа.

6.2.2. Докажите, что интерполяционный метод Лагранжа действительно раскрывает секретный ключ схемы Шамира разделения секрета.

6.2.3. Алиса и Боб знают об абсолютной стойкости одноразового шифр-блокнота, но не хотят мучиться со всеми перепитиями процедуры распределения ключей. Поэтому они разрабатывают свой протокол, основанный на одноразовом шифр-блокноте. При этом Алиса шифрует сообщение m и пересылает его Бобу. Как Алиса, так и Боб вычисляют свой поток ключей, т. е. Алиса знает только свой ключ a, а Боб — только свой b. Обмен сообщениями происходит по схеме:

$$A \longrightarrow B:\ A_1 = m \oplus a,$$
$$B \longrightarrow A:\ B_1 = A_1 \oplus b,$$
$$A \longrightarrow B:\ A_2 = B_1 \oplus a.$$

Покажите, что после такого обмена информацией, Боб сможет расшифровать сообщение m, вычислив

$$A_2 \oplus b.$$

Докажите, также, что Ева, т. е. противник, тоже в состоянии прочесть оригинальное сообщение m.

ЧАСТЬ III

КРИПТОСИСТЕМЫ С ОТКРЫТЫМ КЛЮЧОМ И ПОДПИСИ

Техника шифрования с открытым ключом, разработанная для решения проблемы распределения ключей и аутентификации, имеет много преимуществ перед симметричным шифрованием. Главное из них состоит в том, что стороны, даже не подозревавшие о существовании друг друга до первого письма, могут успешно обмениваться информацией, зашифрованной с помощью криптосистемы с открытым ключом. Кроме того, она дает возможность сторонам подписывать сообщения, такие как электронные заказы или финансовые поручения. В результате шифрование данных с открытым ключом обеспечивает существование технологий электронной торговли.

ГЛАВА 7

ОСНОВНЫЕ АЛГОРИТМЫ ШИФРОВАНИЯ С ОТКРЫТЫМ КЛЮЧОМ

Цели главы

- Узнать о шифровании с открытым ключом и трудных математических задачах, на которых оно основывается.
- Понять алгоритм *RSA* и условия его криптостойкости.
- Разобраться в криптосистеме Эль-Гамаль.
- Узнать о шифрующем алгоритме Рабина.

7.1. Криптография с открытым ключом

Напомним, что в симметричной криптографии каждая из переписывающихся сторон должна иметь копию общего секретного ключа, что создает сложнейшую проблему управления ключами. В криптосистемах, о которых пойдет речь в этой главе, используются два ключа: открытый и секретный.

Открытый ключ может быть опубликован в справочнике наряду с именем пользователя. В результате любой желающий может зашифровать с его помощью свое письмо и послать закрытую информацию владельцу соответствующего секретного ключа. Расшифровать посланное сообщение сможет только тот, у кого есть секретный ключ. Более точно, имеют место преобразования:

сообщение + *ОТКРЫТЫЙ КЛЮЧ АЛИСЫ* = *ШИФРОТЕКСТ*
ШИФРОТЕКСТ + *секретный ключ Алисы* = *сообщение*.

Таким образом, каждый может послать Алисе секретную информацию, воспользовавшись ее открытым ключом. Но только Алиса

в состоянии расшифровать сообщение, поскольку лишь у нее есть соответствующий секретный ключ.

Причина работоспособности таких криптосистем заключается в односторонней математической связи, существующей между двумя ключами, при которой информация об открытом ключе никак не помогает восстановить секретный, но владение секретным ключом обеспечивает возможность расшифровывать сообщения, зашифрованные открытым. На первый взгляд такая связь кажется странной, и для ее понимания требуется определенное время и умственные усилия. Идея криптографии с открытым ключом впервые появилась в 1976 г. в революционной работе Диффи и Хеллмана «Новые направления в криптографии». Но только год спустя была опубликована первая (и наиболее успешная) криптосистема с открытым ключом, а именно, *RSA*.

В предыдущем абзаце описана «официальная» история возникновения криптографии с открытым ключом. Однако в конце 1990-ых годов появилась неофициальная версия освещения событий. Оказалось, что в 1969 году, более чем за пять лет до публикации основополагающей работы Диффи и Хеллмана, Джеймс Эллис, работающий на центр связи Британского правительства (GCHQ), открыл концепцию криптографии с открытым ключом (или несекретное шифрование, как он ее называл) как средство решения проблемы распределения ключей. Впрочем, Эллис, как и Диффи с Хеллманом, не смог разработать в деталях соответствующую криптосистему.

Проблема создания работающего алгоритма шифрования с открытым ключом была решена новым сотрудником GCHQ по имени Клиффорд Кокс в 1973 году. В течение одного дня Кокс разработал систему, которая по существу, является алгоритмом *RSA*, за целых четыре года до Ривеста, Шамира и Адлемана. В 1974 году другой служащий GCHQ, Малькольм Уильямсон, изобрел концепцию алгоритма (обмена ключом) Диффи–Хеллмана, к которому мы обратимся в главе 10. Таким образом, к 1974 году Британская служба безопасности уже имела основную технику криптографии с открытым ключом.

В основе алгоритма шифрования с открытым ключом лежит на удивление мало идей. Этим и объясняется близость алгоритмов, найденных как Эллисом, так и Диффи с Хеллманом, к алгоритму *RSA*. Малое количество идей объясняется необходимостью найти какое-нибудь легко осуществимое на стадии шифрования математическое преобразование, которое сложно было бы обратить (без знания специальной секретной информации) для реализации вто-

рой стадии алгоритма, т. е. расшифрования. Преобразование, обладающее указанным свойством, называется *односторонней функцией* или *функцией-ловушкой*., поскольку в ее дверь войти легко (зашифровать данные), а вот выйти без ключа довольно проблематично.

Придумать такую функцию на пустом месте довольно непросто, но, к счастью, есть некоторый набор широко известных и всесторонне изученных односторонних функций. К ним относятся задача разложения целых чисел на множители, проблема вычисления дискретных логарифмов или вычисление квадратных корней по модулю составного числа. В следующем параграфе, предваряя рассказ об алгоритмах шифрования с открытым ключом, мы познакомимся с примерами односторонних функций. Однако они являются односторонними только в вычислительном отношении, т. е. имея достаточно большие компьютерные мощности, их вполне можно обратить, причем быстрее, чем найти секретный ключ в результате полного перебора.

7.2. Односторонние функции

Наиболее важная односторонняя функция, используемая в криптографии с открытым ключом, — это *разложение на множители* или *факторизация* целых чисел. Под разложением на множители целого числа понимают его представление в виде произведения простых делителей, например,

$$10 = 2 \cdot 5,$$

$$60 = 2^2 \cdot 3 \cdot 5,$$

$$2^{113} - 1 = 3391 \cdot 23\,279 \cdot 65\,993 \cdot 1\,868\,569 \cdot 1\,066\,818\,132\,868\,207.$$

Определение множителей является очень трудоемкой вычислительной операцией. Для оценки сложности алгоритма, раскладывающего целое число N на простые множители, часто используют функцию

$$L_N(\alpha, \beta) = \exp\Big(\big(\beta + o(1)\big)\big(\ln N\big)^{\alpha}\big(\ln\ln N\big)^{1-\alpha}\Big).$$

Обратите внимание на то, что если алгоритм, раскладывающий на множители целое число, имеет сложность $O(L_N(0, \beta))$, то ему требуется полиномиальное время на работу (напомним, что размер задачи на входе — $\ln N$). Однако при сложности алгоритма $O(L_N(1, \beta))$ ему для работы потребуется уже экспоненциальное время. Таким образом, скорость роста функции $L_N(\alpha, \beta)$ при $0 < \alpha < 1$ лежит между полиномиальной и экспоненциальной. Поэтому про алгоритм

со сложностью $O(L_N(\alpha, \beta))$ при $0 < \alpha < 1$ говорят, что он требует суб-экспоненциального времени. Заметим, что умножение, т. е. процесс обратный к разложению на множители, — очень простая операция, требующая времени меньше, чем $O(L_N(0, 2))$.

Существует несколько методов факторизации числа $N = p \cdot q$ с простыми p и q. Некоторые из них мы будем обсуждать в следующей главе, а сейчас лишь назовем наиболее известные.

- *Пробное деление*. В этом алгоритме для всех простых чисел p, не превосходящих $\sqrt{N}$, проверяется условие $N/p \in \mathbb{Z}$. Ясно, что такой подход близок к полному перебору и имеет экспоненциальную сложность $L_N(1, 1)$.
- *Метод эллиптической кривой* хорошо работает, если один из простых множителей p не превосходит 2^{50}. Его сложность оценивается как $L_p(1/2, c)$, т. е. она суб-экспоненциальна. Заметим, что оценка выражена через неизвестный простой делитель. Так что, если N — произведение двух сильно отличающихся друг от друга простых чисел, то метод эллиптической кривой может оказаться весьма эффективным.
- *Квадратичное решето*, вероятно, наиболее быстрый способ разложения чисел, лежащих между 10^{80} и 10^{100}. Его сложность — $L_N(1/2, 1)$.
- *Квадратичное решето в числовом поле*. В настоящее время это наиболее успешный метод для чисел, насчитывающих 100 и более десятичных знаков. С его помощью можно разлагать на множители числа до $10^{155} \approx 2^{512}$, а его сложность оценивается как $L_N(1/2, 1)$.

Существует ряд других сложных задач, связанных с факторизацией, которые используются для разработки криптосистем с открытым ключом. Предположим, дано число $N = p \cdot q$, но не известен ни один из его простых делителей. Возникают четыре основных задачи:

- *Факторизация*: найти делители p и q числа $N = p \cdot q$.
- *Задача RSA*: даны числа C и E, последнее из которых удовлетворяет соотношению:

$$\text{НОД}\,(E, (p-1)(q-1)) = 1.$$

Требуется найти такое число m, что

$$m^E = C \pmod N.$$

- *Тест на квадратичный вычет*: определить, является ли данное число A полным квадратом по модулю N.
- *Извлечение квадратных корней*: дано такое число A, что

$$A = x^2 \pmod{N};$$

нужно вычислить x.

Другой важный класс задач связан с дискретным логарифмированием. Пусть $(G, \cdot)$ — конечная абелева группа, например, мультипликативная группа конечного поля или эллиптическая кривая над конечным полем. Проблема вычисления дискретных логарифмов (*ПДЛ*) состоит в определении целого числа x, которое при данных A, $B \in G$ удовлетворяет соотношению

$$A^x = B.$$

Говоря нестрого, $x = \log_A B$, отсюда и название. Для некоторых групп G такая задача довольно проста. Например, если G — группа целых чисел по модулю N по сложению, то в *ПДЛ* нам надо по известным A, $B \in \mathbb{Z}/N\mathbb{Z}$ найти решение уравнения $x \cdot A = B$. Как мы уже видели в первой главе, существуют простые методы, основанные на расширенном алгоритме Евклида, проверяющие разрешимость такого уравнения и вычисляющие его корень.

Для других групп, однако, эта задача более сложная. Например, в мультипликативной группе конечного поля наилучший из известных алгоритмов, решающий проблему дискретного логарифмирования, — это метод квадратичного решета в числовом поле. Сложность вычисления дискретных логарифмов в этом случае оценивается как $L_N(1/3, c)$, где c — некоторая константа, зависящая от типа поля, например, от его характеристики.

Для групп, подобных эллиптической кривой, задача дискретного логарифмирования еще более трудна. Наилучший из доступных на сегодняшний день методов, вычисляющих дискретные логарифмы над общей эллиптической кривой над полем $\mathbb{F}_q$, называется ρ-методом Полларда. Его сложность — $\sqrt{q} = L_q(1, 1/2)$. Следовательно, это в точности экспоненциальный алгоритм. Поскольку задача дискретного логарифмирования в случае эллиптической кривой сложнее аналогичной задачи для мультипликативной группы конечного поля, мы можем использовать группы не очень большого порядка, что дает преимущества в размере ключа. Поэтому криптосистемы, основанные на эллиптической кривой, как правило, работают с малым ключом, около 160 бит, в то время как системы, от-

талкивающиеся от разложения на множители или вычисления дискретных логарифмов в конечных полях, используют ключи в 1024 бита.

С проблемой дискретного логарифмирования тоже связано несколько близких задач. Для их формулировки предположим, что дана конечная абелева группа $(G, \cdot)$ и ее элемент $A \in G$.

- *ПДЛ* — задача дискретного логарифмирования, о которой мы говорили выше. А именно, по данным A, $B \in G$ найти такой x, что $B = A^x$.
- *ЗДХ* — задача Диффи-Хеллмана, которая состоит в следующем: даны элементы $A \in G$, $B = A^x$ и $C = A^y$; требуется вычислить $D = A^{xy}$.
- *ПВДХ* — проблема выбора Диффи–Хеллмана. Дано:

$$A \in G, \quad B = A^x, \quad C = A^y \quad \text{и} \quad D = A^z;$$

  требуется определить, является ли z произведением $z = x \cdot y$.

Важно знать взаимосвязь всех этих сложных задач, в частности нужно уметь оценивать сложность одной задачи по сравнению с другой. Допустим, например, что у нас есть две задачи: A и B, причем предполагается наличие эффективного алгоритма (*оракула*), решающего задачу B. Если эти задачи действительно близки по сложности, то мы могли бы свести задачу A к задаче B и, опираясь на оракул, решающий вторую задачу, получить ответ к первой. Если сведение задачи A к задаче B занимает не более чем полиномиальное время, то говорят, что задача A не сложнее задачи B.

Кроме того, имеет смысл ввести понятие *полиномиальной эквивалентности* задач, которое означает, что первая не сложнее второй, а вторая — не сложнее первой, если мы покажем, как свести одну задачу к другой и наоборот.

В качестве примера покажем, как свести задачу Диффи–Хеллмана к проблеме дискретного логарифмирования.

Лемма 7.1. В любой конечной абелевой группе G задача Диффи–Хеллмана не сложнее проблемы дискретного логарифмирования.

Доказательство. Предположим, что существует оракул $\mathcal{O}_{лог}$, решающий задачу дискретного логарифмирования, т. е. по элементам A и $B = A^x$ он находит показатель x. Для поиска ответа в задаче Диффи–Хеллмана с данными $B = A^x$ и $C = A^y$ вычисляем

$$Z = \mathcal{O}_{лог}(B) \quad \text{и} \quad D = C^Z.$$

Найденное значение D — искомый ответ к задаче Диффи–Хеллмана. Очевидно, такое сведение имеет полиномиальную сложность и выдает решение задачи Диффи–Хеллмана в предположении, что алгоритм $\mathcal{O}_{лог}$ корректно вычисляет логарифмы. Следовательно, задача Диффи–Хеллмана не сложнее дискретного логарифмирования. ■

Для некоторых групп можно доказать, что эти две задачи полиномиально эквивалентны. Соответствующее рассуждение довольно сложно, и мы не будем его здесь приводить.

Покажем, как проблема выбора Диффи–Хеллмана может быть сведена к задаче Диффи–Хеллмана и, тем самым, к дискретному логарифмированию.

Лемма 7.2. Проблема выбора Диффи–Хеллмана в любой конечной абелевой группе G не сложнее задачи Диффи–Хеллмана.

Доказательство. Пусть $\mathcal{O}_{ДХ}$ — алгоритм, решающий задачу Диффи – Хеллмана, который по элементам A^x и A^y вычисляет элемент A^{xy}. Рассмотрим элементы $B = A^x$, $C = A^y$ и $D = A^z$ и вычислим $E = \mathcal{O}_{ДХ}(B, C)$. Если $E = D$, то решение проблемы выбора положительно, в противном случае — отрицательно.

Как и в предыдущей лемме, сведение одной задачи к другой происходит за полиномиальное время. Поэтому, считая, что алгоритм $\mathcal{O}_{ДХ}$ работает корректно, получаем решение проблемы выбора Диффи – Хеллмана. ■

Мы показали, что проблема выбора Диффи – Хеллмана не труднее прямой задачи Диффи – Хеллмана. Однако существуют группы, в которых проблема выбора Диффи – Хеллмана решается за полиномиальное время, а наилучшие алгоритмы для прямой задачи находят ответ лишь за суб-экспоненциальное время.

Итак, три разновидности проблемы дискретного логарифмирования соотносятся между собой следующим образом: проблема выбора Диффи – Хеллмана — самая легкая, затем идет задача Диффи–Хеллмана, и замыкает ряд задача вычисления дискретных логарифмов.

Вернемся к сравнению сложностей задач, связанных с разложением на множители. Наиболее важный результат здесь сформулирован в лемме 7.3.

Лемма 7.3. Задачи разложения на множители и вычисления квадратного корня полиномиально эквивалентны.

Доказательство. Сначала сведем задачу извлечения квадратного корня к разложению на множители. Предположим, у нас есть алго-

ритм, разлагающий число на множители, и мы хотим с его помощью извлечь квадратный корень по модулю составного числа N. Рассмотрим уравнение

$$Z = x^2 \pmod{N}$$

относительно переменной x. Найдем простые делители P_i числа N, опираясь на известный оракул, и вычислим

$$S_i = \sqrt{Z} \pmod{P_i},$$

что можно сделать за полиномиальное время с помощью алгоритма Шэнкса. После этого, используя китайскую теорему об остатках, найдем x как решение системы уравнений:

$$\{x = S_i \pmod{P_i}\}.$$

При этом необходимо соблюдать осторожность в случае, когда N делится на более высокую, чем первая, степень числа P_i. Однако и с этим случаем можно работать, но мы не будем его подробно разбирать. Итак, вычисление квадратного корня по модулю N не сложнее задачи факторизации.

Покажем, как разложение на множители может быть сведено к задаче об извлечении квадратного корня. Предположим, что у нас есть алгоритм, извлекающий квадратные корни по модулю составного числа N, равного произведению двух простых чисел, что несомненно является самым сложным случаем. Случай общего составного N с точки зрения математики чуть более хитрый, но в вычислительном отношении разложить на множители число, состоящее из произведения трех и более простых делителей, существенно проще, чем отыскать делители числа, если их всего два.

Мы намерены использовать алгоритм извлечения корней для определения делителей числа N. Иными словами, дано число $N = pq$ и требуется найти p. Для этого выберем случайное число $x \in (\mathbb{Z}/N\mathbb{Z})^*$ и вычислим

$$z = x^2 \pmod{N}.$$

Опираясь на алгоритм извлечения корней, найдем

$$y = \sqrt{z} \pmod{N}.$$

Заметим, что существует ровно четыре квадратных корня, поскольку N — произведение двух простых чисел. С пятидесятьюпроцентной вероятностью можно утверждать, что

$$y \neq \pm x \pmod{N}.$$

Если это не так, то можно выбрать другой корень, т. е. в среднем

после двух проходов алгоритма мы наверняка получим требуемое неравенство.

Теперь, так как $x^2 = y^2 \pmod N$, то N должно делить число

$$x^2 - y^2 = (x - y)(x + y).$$

Но N не может делить ни $x-y$, ни $x+y$, поскольку $y \neq \pm x \pmod N$. Таким образом, делители числа N должны делить как сумму $(x+y)$, так и разность $(x - y)$. Поэтому один из нетривиальных делителей числа N может быть найден как НОД $(x - y, N)$.

Очевидно, сведения каждой из задач к другой можно осуществить за полиномиальное время, что доказывает полиномиальную эквивалентность задач факторизации и извлечения корней. ■

В этом доказательстве был применен стандартный прием алгоритмов факторизации, использующий разложение на множители разности квадратов. Мы еще вернемся к нему в главе 8.

Стоит отметить, что задача о квадратичных вычетах безусловно слабее проблемы извлечения корней, поскольку алгоритм, вычисляющий квадратные корни, очевидно способен определить, является ли данное число квадратичным вычетом по определенному модулю, или нет.

Закончим параграф доказательством сводимости задачи *RSA* к проблеме факторизации. Напомним, что задача *RSA* состоит в определении числа m по данным N, E и $C = m^E \pmod N$.

Лемма 7.4. Задача *RSA* не сложнее проблемы факторизации.

Доказательство. Применяя алгоритм факторизации, разложим число N на простые множители, вычислим значение функции Эйлера $\Phi = \varphi(N)$ и найдем

$$D = 1/E \pmod \Phi.$$

Теперь, зная число D, легко восстановить m, поскольку

$$C^D = m^{ED} = m^{1 \pmod \Phi} = m \pmod N.$$

Отсюда следует, что задача *RSA* не сложнее проблемы факторизации. ■

Существует гипотеза, подтвержденная некоторыми косвенными соображениями, что задача *RSA* на самом деле легче проблемы факторизации, т. е. эти задачи не эквивалентны. В настоящее время проверка этой гипотезы — один из главных открытых вопросов криптологии.

7.3. RSA

Алгоритм *RSA*, первый из алгоритмов шифрования с открытым ключом, достойно выдержал испытание временем. Этот алгоритм основывается на задаче *RSA*, с которой мы познакомились в предыдущем параграфе. Как Вы помните, она сводится к поиску простых делителей больших натуральных чисел. Так что можно утверждать, что криптостойкость алгоритма *RSA* базируется на сложности проблемы факторизации, хотя и не в полной мере, поскольку задачу *RSA* можно решать не прибегая к разложению на множители.

Предположим, Алиса считает нужным разрешить всем желающим отправлять ей секретные сообщения, расшифровать которые способна только она. Тогда Алиса подбирает два больших простых числа p и q. Держа их в секрете, Алиса публикует их произведение

$$N = p \cdot q,$$

которое называют *модулем* алгоритма. Кроме того, Алиса выбирает шифрующую экспоненту E, удовлетворяющую условию

$$\text{НОД}\left(E, (p-1), (q-1)\right) = 1.$$

Как правило E берут равным 3, 17 или 65 537. Пара, доступная всем желающим, — это (N, E). Для выбора секретного ключа Алиса применяет расширенный алгоритм Евклида к паре чисел E и $(p-1)(q-1)$, получая при этом расшифровывающую экспоненту d. Найденная экспонента удовлетворяет соотношению

$$E \cdot d = 1 \pmod{(p-1)(q-1)}.$$

Секретным ключом является тройка (d, p, q). Фактически, можно было бы выбросить простые делители p и q из ключа и помнить лишь о d и всем числе N. Но, как мы позже убедимся, это снизит скорость алгоритма (см. стр. 293).

Допустим теперь, что Боб намерен зашифровать сообщение, адресованное Алисе. Он сверяется с открытым ключом и представляет сообщение в виде числа m, строго меньшего модуля N алгоритма. Шифротекст C получается из m по следующему правилу:

$$C = m^E \pmod{N}.$$

Алиса, получив шифрограмму, расшифровывает ее, возводя число C в степень d:

$$m = C^d \pmod{N}.$$

Равенство имеет место в связи с тем, что порядок группы $(\mathbb{Z}/N\mathbb{Z})^*$

равен $\varphi(N) = (p-1)(q-1)$. Поэтому, по теореме Лагранжа,

$$x^{(p-1)(q-1)} = 1 \pmod{N}$$

для любого числа $x \in (\mathbb{Z}/N\mathbb{Z})^*$. Поскольку E и d взаимно обратны по модулю $(p-1)(q-1)$, при некотором целом числе s получается равенство

$$Ed - s(p-1)(q-1) = 1.$$

Следовательно,

$$C^d = (m^E)^d = m^{Ed} = m^{1+s(p-1)(q-1)} =$$
$$= m \cdot m^{s(p-1)(q-1)} = m \pmod{N}.$$

Для прояснения ситуации рассмотрим детский пример. Пусть $p = 7$ и $q = 11$. Тогда $N = 77$, а $(p-1)(q-1) = 6 \cdot 10 = 60$. В качестве открытой шифрующей экспоненты возьмем число $E = 37$, поскольку НОД$(37, 60) = 1$. Применяя расширенный алгоритм Евклида, найдем $d = 13$, т. к.

$$37 \cdot 13 = 481 = 1 \pmod{60}.$$

Предположим, нужно зашифровать сообщение, численное представление которого имеет вид: $m = 2$. Тогда мы вычисляем

$$C = m^E \pmod{N} = 2^{37} \pmod{77} = 51.$$

Процесс расшифровывания происходит аналогично:

$$m = C^d \pmod{N} = 51^{13} \pmod{77} = 2.$$

7.3.1. Шифрование *RSA* и одноименная задача

При первом знакомстве с алгоритмом *RSA* можно увидеть, что его криптостойкость обеспечивается сложностью вычисления расшифровывающей экспоненты d по открытому ключу, т. е. по известным числу N (модулю системы) и шифрующей экспоненте.

Мы показали, что задача *RSA* не сложнее проблемы факторизации. Поэтому, если разложим на множители число N, т. е. найдем p и q, то сможем вычислить d. Следовательно, если проблема факторизации модуля системы окажется легкой, *RSA* будет взломана. Самые большие числа, которые к настоящему времени удается разложить на множители за разумное время, имеют 500 двоичных знаков. В связи с этим, для обеспечения стойкости систем среднего срока действия, рекомендуют брать модули шифрования порядка 1024 битов. Для систем большого срока действия следует выбирать модули, состоящие из 2048 битов.

В этой главе под криптостойкостью системы будем понимать невозможность дешифрования сообщения без знания секретного ключа. Позже мы покажем, что такой подход к безопасности слишком наивен во многих приложениях. Кроме того, в следующих главах мы продемонстрируем, что алгоритм *RSA* в том виде, в котором мы его представили, не может устоять против атак с выбором шифротекста.

В криптосистемах с открытым ключом атакующий всегда имеет возможность шифровать свои сообщения. Значит, он способен применять атаку с выбором открытого текста. *RSA* криптостоек против таких атак, если принять во внимание наше слабое определение стойкости и верить в трудность решения задачи *RSA*. Для аргументации этого заявления нужно использовать прием сведения одной задачи к другой, изложенный в предыдущей главе. В данной ситуации доказательство безопасности алгоритма *RSA* довольно тривиально, но мы рассмотрим этот вопрос подробно, т. к. соответствующие аргументы будут применяться снова и снова в последующих главах.

Лемма 7.5. Если задача *RSA* является трудноразрешимой, то криптосистема *RSA* вычислительно защищена от атак с выбором открытого текста в том смысле, что атакующий не в состоянии корректно восстановить открытый текст, имея лишь шифротекст.

Доказательство. Разработаем алгоритм решения задачи *RSA*, основываясь на алгоритме взлома одноименной криптосистемы. Сделав это, мы покажем, что взлом криптосистемы не сложнее задачи *RSA*.

Напомним, что в задаче *RSA* дано число N, имеющее два неизвестных простых делителя p и q, элементы $E, Y \in (\mathbb{Z}/N\mathbb{Z})^*$ и требуется найти такой элемент x, что $x^E \pmod N = Y$. С помощью оракула дешифруем сообщение $C = Y$ и получим соответствующий открытый текст m, удовлетворяющий, по определению, соотношению

$$m^E \pmod N = C = Y.$$

Отсюда видно, что взломав криптосистему, мы сможем решить задачу *RSA*. ■

7.3.2. Секретная экспонента и проблема факторизации

Пока нам не ясно, является ли взлом криптосистемы *RSA*, в смысле обращения функции *RSA*, эквивалентным проблеме факторизации,

решив которую, можно найти секретный ключ d по открытой информации о N и E. Продемонстрируем взаимосвязь этих задач.

Лемма 7.6. Если известна расшифровывающая экспонента d алгоритма *RSA*, соответствующая открытому ключу (N, E), то число N можно эффективно разложить на множители.

Доказательство. Напомним, что при некотором целом s имеет место равенство

$$Ed - 1 = s(p-1)(q-1).$$

Возьмем произвольное целое число $X \neq 0$. Тогда

$$X^{Ed-1} = 1 \pmod{N}.$$

Вычисляем квадратный корень Y_1 по модулю N:

$$Y_1 = \sqrt{X^{Ed-1}} = X^{\frac{Ed-1}{2}} \pmod{N},$$

что можно сделать, поскольку $Ed - 1$ известно и четно. Приходим к тождеству

$$Y_1^2 - 1 = 0 \pmod{N},$$

которое можно использовать для определения делителей числа N с помощью вычисления НОД$(Y_1 - 1, N)$. Однако это будет работать, только если $Y_1 \neq \pm 1 \pmod{N}$.

Предположим, что нам не повезло и $Y_1 = \pm 1 \pmod{N}$. Если $Y_1 = -1 \pmod{N}$, вернемся к началу и выберем другое число X. Если $Y_1 = 1 \pmod{N}$, то можно взять еще один квадратный корень

$$Y_2 = \sqrt{Y_1} = X^{\frac{ED-1}{4}} \pmod{N}.$$

Опять получаем

$$Y_2^2 - 1 = Y_1 - 1 = 0 \pmod{N},$$

откуда НОД$(Y_2 - 1, N)$ — делитель числа N. К сожалению, здесь тоже может оказаться, что $Y_2 = \pm 1 \pmod{N}$. Тогда придется повторить все снова.

Алгоритм необходимо повторять до тех пор, пока мы не разложим N на множители или не придем к числу $(ED - 1)/2^t$, которое уже не будет делиться на 2. В последнем случае нам придется вернуться к началу алгоритма, выбрать новое случайное значение X и все повторить. ■

Алгоритм, приведенный в этом доказательстве, — типичный пример алгоритма Лас-Вегаса, вероятностного по своей природе,

которая проявляется в процессе работы. Но если уж ответ отыщется, то он обязательно будет верным.

Разберем небольшой пример, иллюстрирующий изложенный метод. Рассмотрим следующие входные данные задачи *RSA*:

$$N = 1\,441\,499, \quad E = 17 \quad \text{и} \quad d = 507\,905.$$

Напомним, что мы предполагаем известной расшифровывающую экспоненту d, которая обычно хранится в секрете. Опираясь на предыдущий алгоритм, найдем делители числа N. Положим

$$T_1 = (Ed - 1)/2 = 4\,317\,192,$$
$$X = 2.$$

Для вычисления Y_1, сделаем преобразования:

$$Y_1 = X^{(Ed-1)/2} = 2^{T_1} = 1 \pmod{N}.$$

Поскольку Y_1 оказался равным 1, нам нужно взять

$$T_2 = T_1/2 = (Ed - 1)/4 = 2\,158\,596 \quad \text{и} \quad Y_2 = 2^{T_2}.$$

Теперь

$$Y_2 = X^{(Ed-1)/4} = 2^{T_2} = 1 \pmod{N}.$$

Снова нужно повторять предыдущие шаги, что приведет нас к

$$T_3 = (Ed - 1)/8 = 1\,079\,298$$

и

$$Y_3 = X^{(Ed-1)/8} = 2^{T_3} = 119\,533 \pmod{N}.$$

Отсюда,

$$Y_3^2 - 1 = (Y_3 - 1)(Y_3 + 1) = 0 \pmod{N},$$

и мы можем найти простой делитель числа N, вычислив

$$\text{НОД}\,(Y_3 - 1, N) = 1\,423.$$

7.3.3. Значение функции Эйлера $\varphi(N)$ и проблема факторизации

Как мы убедились, информация о секретной экспоненте d позволяет найти простой делитель числа N. Здесь мы покажем, как знание значения функции Эйлера $\Phi = \varphi(N)$ помогает решить ту же задачу.

Лемма 7.7. Значение $\Phi = \varphi(N)$ позволяет эффективно разложить число N на множители.

Доказательство. Имеем

$$\Phi = (p-1)(q-1) = N - (p+q) + 1.$$

Следовательно, положив $S = N + 1 - \Phi$, мы получим

$$S = p + q.$$

Нам нужно определить числа p или q, опираясь на их сумму S и произведение N. Рассмотрим многочлен

$$f(X) = (X-p)(X-q) = X^2 - SX + N.$$

Теперь можно найти как p, так и q, решая квадратное уравнение $f(X) = 0$ стандартным образом:

$$p = \frac{S + \sqrt{S^2 - 4N}}{2}, \qquad q = \frac{S - \sqrt{S^2 - 4N}}{2}.$$

■

В качестве примера рассмотрим открытый модуль $N = 18\,923$ криптосистемы *RSA* и предположим, что нам известно $\Phi = \varphi(N) = 18\,648$. Вычисляем

$$S = p + q = N + 1 - \Phi = 276.$$

Соответствующий многочлен имеет вид:

$$f(X) = X^2 - SX + N = X^2 - 276X + 18\,923,$$

а его корни — $p = 149$, $q = 127$.

7.3.4. Разделенный модуль

Поскольку арифметика остатков — дорогое удовольствие с точки зрения компьютера, весьма заманчиво разработать систему шифрования, в которой пользователи разделяют общий модуль N, но применяют различные шифрующие и расшифровывающие экспоненты (E_i, d_i). Одна из причин, побуждающая это делать, — ускорить алгоритмы шифрования и расшифровывания в аппаратных средствах, специально настроенных на определенный модуль N. Однако это неудачная идея, поскольку пасует перед двумя типами нападающих: внутреннего, т. е. законного пользователя системы, и внешнего.

Предположим, что нападающим является один из законных клиентов криптосистемы, скажем пользователь номер 1. Он может найти значение расшифровывающей экспоненты, которую хранит в секрете пользователь номер 2, а именно d_2. Сначала он вычисляет p и q с помощью алгоритма из доказательства леммы 7.6, что он в

состоянии сделать, зная свою расшифровывающая экспоненту d_1. Затем злоумышленник находит $\varphi(N) = (p-1)(q-1)$ и, наконец, раскрывает значение d_2 по формуле

$$d_2 = \frac{1}{E_2} \pmod{\varphi(N)}.$$

Предположим теперь, что атакующий не принадлежит к пользователям криптосистемы, использующим общий модуль. Допустим, Алиса посылает одно и то же сообщение m двум клиентам криптосистемы, открытые ключи которых

$$(N, E_1) \quad \text{и} \quad (N, E_2).$$

Ева, нападающая извне, видит зашифрованные сообщения C_1 и C_2, где

$$C_1 = m^{E_1} \pmod{N}, \quad C_2 = m^{E_2} \pmod{N}.$$

Она может вычислить

$$T_1 = E_1^{-1} \pmod{E_2}, \quad T_2 = (T_1 E_1 - 1)/E_2$$

и восстановить сообщение m по следующей схеме:

$$\begin{aligned} C_1^{T_1} C_2^{-T_2} &= m^{E_1 T_1} m^{-E_2 T_2} = \\ &= m^{1+E_2 T_2} m^{-E_2 T_2} = \\ &= m^{1+E_2 T_2 - E_2 T_2} = \\ &= m^1 = m. \end{aligned}$$

Рассмотрим пример внешней атаки в случае

$$N = N_1 = N_2 = 18\,923, \quad E_1 = 11 \quad \text{и} \quad E_2 = 5.$$

Предположим, что перехвачены шифротексты

$$C_1 = 1\,514 \quad \text{и} \quad C_2 = 8\,189,$$

соответствующие одному открытому тексту m. Тогда Ева вычисляет $T_1 = 1$ и $T_2 = 2$, после чего раскрывает исходную информацию:

$$m = C_1^{T_1} C_2^{-T_2} = 100 \pmod{N}.$$

7.3.5. Использование малых шифрующих экспонент

Иногда в криптосистемах *RSA* с целью экономии затрат на шифрование используются небольшие шифрующие экспоненты E. Покажем, что это тоже создает дополнительные проблемы, связанные с

криптостойкостью. Предположим, что у нас есть три пользователя с различными модулями шифрования

$$N_1, \quad N_2 \quad \text{и} \quad N_3$$

и одной шифрующей экспонентой $E = 3$. Пусть некто посылает им одно сообщение m, зашифрованное тремя разными открытыми ключами. Нападающий видит три криптограммы:

$$C_1 = m^3 \pmod{N_1},$$
$$C_2 = m^3 \pmod{N_2},$$
$$C_3 = m^3 \pmod{N_3}$$

и с помощью китайской теоремы об остатках находит решение системы

$$\{X = C_i \pmod{N_i} | \ i = 1, 2, 3\}$$

в виде

$$X = m^3 \pmod{N_1 N_2 N_3}.$$

Но поскольку $m^3 < N_1 N_2 N_3$, целые числа X и m^3 должны совпадать. Поэтому, вычисляя кубический корень из X, мы раскрываем сообщение.

Пусть, например,

$$N_1 = 323, \quad N_2 = 299, \quad N_3 = 341$$

и Ева перехватывает сообщения

$$C_1 = 50, \quad C_2 = 299 \quad \text{и} \quad C_3 = 1,$$

Чтобы восстановить исходное сообщение m, Ева находит решение системы

$$X = 300\,763 \pmod{N_1 N_2 N_3},$$

после чего извлекает обычный кубический корень

$$m = X^{1/3} = 67.$$

Обе атаки, о которых мы сейчас рассказали, представляют несомненный интерес, поскольку позволяют раскрыть текст, не решая сложную задачу разложения на множители. Возможность таких атак служит свидетельством (хотя и слабым) в пользу того, что вскрытие криптосистемы *RSA* более легкая задача, чем проблема факторизации. Однако наиболее важный вывод, который следует извлечь из этих атак, заключается в необходимости дополнять оригинальные сообщения случайными цифрами перед шифрованием.

Такая предосторожность сильно снизит вероятность шифрования одного и того же сообщения разными пользователями. Кроме того, следует избегать небольших шифрующих экспонент. Сейчас обычно выбирают $E = 65\,537$. Однако при использовании *RSA* для электронной цифровой подписи (см. ниже) можно беспроблемно брать и небольшие шифрующие экспоненты.

7.4. Криптосистема Эль-Гамаль

Простейший алгоритм шифрования, основывающийся на дискретном логарифмировании, — это криптосистема Эль-Гамаль. Сейчас мы опишем шифрование в системе Эль-Гамаль, использующее конечные поля. Существует аналогичная система эллиптических кривых. Ее продумывание мы оставим в качестве упражнения.

В отличие от *RSA*, в алгоритме Эль-Гамаль существуют некоторые открытые параметры, которые могут быть использованы большим числом пользователей. Они называются *параметрами домена* и выглядят следующим образом:

- P — «большое простое число», т. е. число, насчитывающее около 1024 битов, такое, что $P - 1$ делится на другое, «среднее простое число» Q, лежащее неподалеку от 2^{160}.
- G — элемент мультипликативной группы поля $\mathbb{F}_P^*$, порядок которой, как мы знаем, делится на Q, причем

$$G^{(P-1)/Q} \pmod{P} \neq 1.$$

Все параметры домена, т. е. P, Q и G, выбираются таким образом, чтобы элемент $G^{(P-1)/Q}$ был образующей абелевой группы A порядка Q. Информация об этой группе открыта и используется большим числом пользователей.

После выбора параметров домена определяют открытый и секретный ключи. Секретным ключом может априори быть любое натуральное число x, а открытый ключ получается по следующей формуле:

$$H = G^x \pmod{P}.$$

Обратите внимание на то, что каждый из пользователей *RSA* должен генерировать два больших простых числа для определения ключевой пары, что является довольно громоздкой задачей, а в системе Эль-Гамаль для построения ключевой пары достаточно найти какое-нибудь случайное число и сделать сравнительно несложные вычисления в арифметике остатков.

Сообщение в этой системе представляется ненулевым элементом поля $m \in \mathbb{F}_P^*$. Для его шифрования поступают следующим образом:

- генерируют случайный эфемерный ключ k,
- вычисляют $C_1 = G^k$,
- находят $C_2 = m \cdot H^k$,
- выдают получившийся шифротекст в виде пары $C = (C_1, C_2)$.

Заметим, что при каждом шифровании применяется свой кратковременный ключ. Поэтому, шифруя одно сообщение дважды, мы получаем разные шифротексты.

Чтобы расшифровать пару данных $C = (C_1, C_2)$, производят следующие преобразования:

$$\begin{aligned}\frac{C_2}{C_1^x} &= \frac{m \cdot H^k}{G^{xk}} = \\ &= \frac{m \cdot G^{xk}}{G^{xk}} = \\ &= m.\end{aligned}$$

Разберем небольшой пример, выбрав сначала параметры домена. Пусть

$$Q = 101, \quad P = 809 \quad \text{и} \quad G = 3.$$

Легко проверить, что Q действительно делит число $P-1$, а порядок элемента G в группе $\mathbb{F}_P^*$ делится на Q. Порядок элемента G равен 808, поскольку

$$3^{808} = 1 \pmod P,$$

и ни при каких меньших степенях такого равенства не получается. В качестве пары открытого и секретного ключа выберем

$$x = 68 \quad \text{и} \quad H = G^x = 3^{68} = 65 \pmod P.$$

Допустим, нам нужно зашифровать сообщение, численное представление которого равно $m = 100$. Поступаем следующим образом.

- Генерируем случайный эфемерный ключ $k = 89$.
- Находим $C_1 = G^k = 3^{89} = 345 \pmod P$.
- Получаем $C_2 = m \cdot H^k = 100 \cdot 65^{89} = 517 \pmod P$.
- Отправляем шифротекст $C = (345, 517)$.

Партнер сможет восстановить текст, делая также вычисления:

$$\frac{C_2}{C_1^x} = \frac{517}{345^{68}} = 100.$$

Последнее равенство получается чуть более сложно, чем в вещественных числах: сначала число 345 возводится в степень 68 по модулю 809, вычисляется мультипликативный обратный к результату по этому же модулю, а затем найденный обратный умножается на 517.

Позже мы увидим, что система Эль-Гамаль в том виде, о котором только что было рассказано, беззащитна против атак с выбором шифротекста. Поэтому обычно применяют модифицированные схемы шифрования. Тем не менее, Эль-Гамаль сможет выстоять против атаки с выбором открытого текста, если считать, что задача Диффи – Хеллмана трудна для решения. Опять-таки, здесь мы используем наивное понятие криптостойкости алгоритма, считая, что система защищена, если противник не сможет обратить шифрующую функцию.

Лемма 7.8. Если задача Диффи – Хеллмана трудноразрешима, то система Эль-Гамаль защищена против атак с выбором открытого текста, где защищенность означает, что нападающий не может восстановить открытый текст по перехваченной шифрограмме за разумное время.

Доказательство. Чтобы показать защищенность системы Эль-Гамаль от атак с выбором открытого текста в предположении о сложности задачи Диффи – Хеллмана, будем считать, что у нас есть оракул $\mathcal{O}$, вскрывающий шифр Эль-Гамаль. На вход оракула подается открытый ключ H и шифротекст (C_1, C_2), а выходными данными служит дешифрованный открытый текст. Покажем теперь, как с помощью оракула решается задача Диффи – Хеллмана, т. е. даны

$$G^x \quad \text{и} \quad G^y,$$

и требуется вычислить G^{xy}.

Выберем открытый ключ в системе Эль-Гамаль в соответствии с поставленной задачей, т. е. положим

$$H = G^x.$$

Заметим, что мы не знаем секретного ключа x. Теперь выписываем «шифротекст»:

$$C = (C_1, C_2),$$

где $C_1 = G^y$, а C_2 случайный элемент поля $\mathbb{F}_P^*$. Вводим этот шифротекст в оракул, взламывающий Эль-Гамаль, и получаем соответствующий открытый текст $M = \mathcal{O}(H, (C_1, C_2))$, который по расшифровывающей процедуре Эль-Гамаль должен получаться в виде отношения $M = \frac{C_2}{C_1^x}$. Используя полученные данные, можно решить

исходную задачу Диффи-Хеллмана, вычисляя

$$\frac{C_2}{M} = \frac{M \cdot C_1^x}{M} = (G^y)^x = G^{xy}.$$

■

7.5. Криптосистема Рабина

Есть еще одна криптосистема, принадлежащая Рабину, которая основывается на трудной проблеме факторизации больших целых чисел. Более точно, она связана с трудностью извлечения квадратного корня по модулю составного числа $N = p \cdot q$. Напомним, что эти задачи эквивалентны, т. е.

- зная простые делители числа N, мы можем извлекать квадратные корни по модулю N,
- умея извлекать квадратные корни по модулю N, мы в состоянии разложить N на простые множители.

Потому такую систему можно считать в некотором отношении более криптостойкой, чем *RSA*. Процесс шифрования в алгоритме Рабина происходит намного быстрее, чем практически в любой другой криптосистеме с открытым ключом. Однако, несмотря на эти преимущества, система Рабина используется все же реже, чем *RSA*. Тем не менее, система Рабина важна как с исторической точки зрения, так и в качестве наглядного примера криптосистемы, основные идеи которой используются в протоколах более высокого уровня.

Выберем разные простые числа, удовлетворяющие условию:

$$p = q = 3 \pmod 4.$$

Такой специальный вид простых чисел сильно ускоряет процедуру извлечения корней по модулю p и q. Секретным ключом системы является пара (p, q). Для определения соответствующего открытого ключа берут произведение $N = p \cdot q$ и генерируют случайное целое число $B \in \{0, \ldots, N-1\}$. Открытый ключ — это пара

$$(N, B).$$

Для шифрования сообщения m в алгоритме Рабина вычисляют

$$C = m(m + B) \pmod N.$$

Таким образом, шифрование состоит из операций сложения и умножения по модулю N, что обеспечивает более высокую скорость ши-

фрования, чем в *RSA*, даже если в последней выбирают небольшую шифрующую экспоненту.

Расшифрование в этом алгоритме гораздо более сложное. По существу, нам нужно вычислить

$$m = \sqrt{\frac{B^2}{4} + C} - \frac{B}{2} \pmod{N}.$$

На первый взгляд здесь не требуется никакой секретной информации, но, очевидно, для извлечения корней по модулю N очень и очень полезно знать разложение последнего на простые делители. Поскольку N — произведение двух простых чисел, существует четыре возможных квадратных корня из числа по модулю N. Поэтому при расшифровании получается четыре возможных открытых текста. Чтобы выбор был более определенным, стоит к открытому тексту добавлять некую избыточную информацию.

Объясним, почему при расшифровании мы действительно получаем m. Напомним, что шифротекст имеет вид

$$C = m(m + B) \pmod{N},$$

поэтому

$$\begin{aligned}\sqrt{\frac{B^2}{4} + C} - \frac{B}{2} &= \sqrt{\frac{B^2 + 4m(m+B)}{4}} - \frac{B}{2} = \\ &= \sqrt{\frac{4m^2 + 4Bm + B^2}{4}} - \frac{B}{2} = \\ &= \sqrt{\frac{(2m+B)^2}{4}} - \frac{B}{2} = \\ &= \frac{2m+B}{2} - \frac{B}{2} = m.\end{aligned}$$

Конечно, здесь мы предполагаем, что выбрали правильный корень из четырех возможных.

Закончим главу примером шифрования в алгоритме Рабина. Пусть открытые и секретные ключи имеют такой вид:

- $p = 127$ и $q = 131$,
- $N = 16\,637$ и $B = 12\,345$.

Для шифрования сообщения $m = 4410$ вычисляем

$$C = m(m + B) \pmod{N} = 4633.$$

При обратной операции сначала находим

$$T = B^2/4 + C \pmod{N} = 1500,$$

а затем извлекаем квадратные корни из T по модулю p и q:

$$\sqrt{T} \pmod{p} = \pm 22, \quad \sqrt{T} \pmod{q} = \pm 37.$$

После этого, применяя Китайскую теорему об остатках к паре

$$\pm 22 \pmod{p} \quad \text{и} \quad \pm 37 \pmod{q},$$

находим квадратный корень из T по модулю N:

$$s = \sqrt{T} \pmod{N} = \pm 3705 \text{ или } \pm 14\,373.$$

Четыре варианта расшифрования

$$4410, \quad 5851, \quad 15\,078, \text{ или } 16\,519.$$

получаются из формулы

$$s - \frac{B}{2} = s - \frac{12\,345}{2}.$$

Краткое содержание главы

- Шифрование с открытым ключом требует односторонних функций, примерами которых служат факторизация чисел, извлечение корней, дискретное логарифмирование и задача Диффи–Хеллмана.
- Между этими задачами существуют связи, которые можно выявить сведением одной задачи к другой.
- *RSA* — наиболее популярная криптосистема с открытым ключом. Ее защищенность по вычислениям основывается на трудности решения задачи *RSA*, которая близка проблеме факторизации, но не совпадает с ней.
- Криптостойкость системы Эль-Гамаль зависит от трудности решения задачи Диффи–Хеллмана.
- Криптостойкость алгоритма Рабина гарантируется сложностью извлечения корней по составному модулю. Поскольку эта задача полиномиально эквивалентна проблеме факторизации, можно считать, что безопасность системы Рабина опирается на проблему разложения на множители.

Дополнительная литература

Оригинальная статья Диффи и Хеллмана все еще остается самым лучшим и быстрым введением в концепцию криптографии с открытым ключом. Кроме того, можно посмотреть работы, касающиеся Эль-Гамаль, *RSA* и системы Рабина.

W. Diffie and M. Hellman. *New directions in cryptography*. IEEE Trans, on Info. Theory, **22**, 644–654, 1976.

T. ElGamal. *A public key cryptosystem and a signature scheme based on discrete logarithms*. IEEE Trans. Info. Theory, **31**, 469–472, 1985.

R.L. Rivest, A. Shamir and L.M. Adleman. *A method for obtaining digital signatures and public-key cryptosystems*. Comm. ACM, **21**, 120–126, 1978.

M. Rabin. *Digitized signatures and public key functions as intractable as factorization*. MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

Контрольные вопросы

7.1.1. Что представляет собой самый быстрый на сегодняшний день алгоритм разложения на множители?

7.1.2. В чем заключаются задачи факторизации, *RSA* и определения квадратичного вычета? Расположите эти проблемы в порядке увеличения сложности.

7.1.3. Расскажите о проблеме дискретного логарифмирования и задачах Диффи – Хеллмана. Расположите их в порядке увеличения сложности

7.1.4. Опишите алгоритм шифрования *RSA*.

7.1.5. Может ли шифрующая экспонента в системе *RSA* быть четной?

7.1.6. Обсудите утверждения:
(а) Информация о расшифровывающей экспоненте в алгоритме *RSA* равносильна разложению модуля шифрования на множители.
(б) Взлом алгоритма шифрования в *RSA* эквивалентен разложению на множители модуля шифрования.

7.1.7. В шифровании по алгоритму Эль-Гамаль присутствует элемент случайности, что обеспечивает разные шифротексты при шифровании одного сообщения дважды. Хорошо это или плохо?

7.1.8. Расскажите о двух преимуществах, которые система Рабина имеет перед *RSA*. Имеет ли алгоритм Рабина какие-либо недостатки?

Лабораторные работы

7.2.1. Разработайте программы шифрования и расшифровывания сообщений для алгоритмов Эль-Гамаль и Рабина. Какая из них более эффективна при шифровании, а какая при обратной процедуре?

Упражнения

7.3.1. Пусть N — модуль шифрования алгоритма *RSA* и $\lambda(N) = \text{НОК}(p-1, q-1)$. Докажите, что если E — шифрующая экспонента алгоритма, то расшифровывающую экспоненту d можно выбрать так, чтобы

$$E \cdot d = 1 \pmod{\lambda(N)}.$$

Указание. Покажите, что $\lambda(N)$ — максимальный порядок элементов группы $(\mathbb{Z}/N\mathbb{Z})^*$.

7.3.2. Пусть N — модуль в *RSA* и $\lambda(N) = \text{НОК}(p-1, q-1)$. Предположим, что порядок шифрующей экспоненты E в группе $(\mathbb{Z}/\lambda(N)\mathbb{Z})^*$ равен K. Покажите, что

$$m^{E^k} = m \pmod{N}.$$

Выведите отсюда, что порядок элемента E по модулю $p-1$ или $q-1$ должен быть большим.

7.3.3. Опишите схему шифрования Эль-Гамаль, основанную на эллиптических кривых.

7.3.4. Покажите, что схему шифрования *RSA* с *простым* модулем $N = P$ очень легко взломать.

7.3.5. Объясните, почему при передаче большого числа данных наиболее приемлемый способ заключается в том, что сначала с помощью криптосистемы с открытым ключом передается секретный ключ, а потом с его помощью шифруются основные данные симметричным алгоритмом.

7.3.6. Формальным рассуждением покажите, что вскрытие шифрующего алгоритма Рабина эквивалентно разложению на множители натуральных чисел.

7.3.7. Исследуйте различные леммы в этой главе и предположите, что оракулы дают верный ответ лишь с некоторой вероятно-

стью, например, верный ответ — один из десяти. Измените формулировки лемм и докажите получившиеся утверждения в соответствии с новыми предположениями. В каких леммах вы можете «увеличить» вероятность и получить редукцию, которая работает с высокой вероятностью, даже если оракул выдает верные ответы с низкой вероятностью?

ГЛАВА 8

ТЕСТЫ НА ПРОСТОТУ И ФАКТОРИЗАЦИЯ

Цели главы

- Объяснить основные тесты на простоту.
- Подробно описать часто используемый тест на простоту, а именно, тест Миллера – Рабина.
- Рассказать о различных алгоритмах факторизации.
- Показать, как работает самый удачный алгоритм разложения на множители: метод решета в числовом поле.

8.1. Простые числа

Простые числа необходимы для почти всех алгоритмов шифрования с открытым ключом, например,

- В *RSA* или криптосистеме Рабина нам нужны простые числа p и q для определения открытого ключа $N = p \cdot q$.
- В Эль-Гамаль необходимо простое число p и простое q, делящее $p - 1$.
- В варианте Эль-Гамаль, использующем эллиптические кривые, требуется эллиптическая кривая над конечным полем, порядок которой делится на большое простое число q.

Мы увидим, что тестирование чисел на простоту можно выполнить с помощью очень простого и эффективного алгоритма, допускающего, однако, вероятностную ошибку. Итерируя этот алгоритм, можно уменьшить вероятность ошибки до любой требуемой величины.

Некоторые из более сложных тестов выдают сертификат, проверив который третья сторона может убедиться в том, что проверяемое число действительно простое. Одно из очевидных требований, предъявляемых к такому сертификату, состоит в том, что его должно быть легче проверить, чем получить Проверку чисел

на простоту с помощью сертификатов называют *алгоритмом доказательства простоты*, а соответствующий сертификат — *доказательством простоты*. Надо сказать, что основной алгоритм, использующийся в криптографии, выдает сертификаты того, что данное число является составным, но не сертификаты простоты.

Перед обсуждением алгоритмов обратим внимание на некоторые эвристические свойства простых чисел. Знаменитый результат, сформулированный Гауссом в начале XIX века в виде гипотезы после долгих численных экспериментов, сейчас носит название *закона распределения простых чисел*:

Теорема 8.1. (**Закон распределения простых чисел.**) Функция $\pi(X)$, вычисляющая количество простых чисел, не превосходящих X, имеет следующую аппроксимацию:

$$\pi(X) \approx \frac{X}{\ln X}.$$

Этот факт говорит о том, что простые числа встречаются довольно часто, например, количество простых чисел, меньших 2^{512}, приблизительно равно 2^{503}. Кроме того, с помощью закона распределения можно вычислять вероятность простоты выбранного наугад числа. Вероятность случайным образом выбранного числа p оказаться простым равна $\frac{1}{\ln p}$. Отсюда вытекает, что взятое наугад число, состоящее из 512 двоичных знаков, будет простым с вероятностью $\approx \frac{1}{\ln p} \approx \frac{1}{177}$. Это означает, что в среднем нам нужно просмотреть 177 случайных чисел порядка 2^{512}, чтобы найти одно простое.

8.1.1. Пробное деление

Наивный способ проверки числа p на простоту состоит в пробном делении. По существу, мы пытаемся разделить число p на все числа, начиная с 2 и кончая $\sqrt{p}$. Если результат деления — целое число, то p — составное, и мы в качестве бесплатного приложения получим один из делителей числа p, что тоже неплохо. Если же не удастся разделить нацело p ни на какое из упомянутых чисел, то оно является простым. Таким образом, пробное деление обладает тем преимуществом (в сравнении с более сложными тестами на простоту), что в результате работы такого алгоритма мы либо докажем простоту числа p, либо найдем его нетривиальный делитель.

Однако пробное деление — очень громоздкая стратегия! В самом плохом случае, когда p действительно простое, нам потребуется

$\sqrt{p}$ шагов, т. е. метод имеет экспоненциальную сложность. Другой недостаток алгоритма состоит в том, что он не дает никакого легкого способа проверки простоты найденного числа p, оставляя единственную возможность убедить кого-либо в верности результата — повторение всего алгоритма. Простой способ проверки отсутствия простоты, конечно, есть: мы нашли нетривиальный делитель, поэтому для доказательства того, что p — составное число, достаточно проверить его делимость на указанный множитель.

Несмотря на свои недостатки, метод пробного деления хорош для поиска небольших простых чисел. Существует способ и частичного перебора, когда не проверяются на простоту заведомо составные числа. Например, никакое четное число, большее 2, не может быть простым, любое число с суммой цифр, кратной 3, делится на 3 и т. д. Кроме того, в качестве гипотетических делителей достаточно брать лишь простые числа.

8.1.2. Тест Ферма

Наиболее развитые вероятностные алгоритмы проверки чисел на простоту основаны на теореме, обратной малой теореме Ферма. Напомним, что если G — мультипликативная группа порядка $\#G$, то ввиду теоремы Лагранжа $a^{\#G} = 1$ для любого элемента $a \in G$. Следовательно, если $G = (\mathbb{Z}/N\mathbb{Z})^*$ — мультипликативная группа кольца вычетов по модулю N, т. е. группа порядка $\varphi(N)$, то для любого ее элемента a имеет место равенство

$$a^{\varphi(N)} = 1 \pmod{N}.$$

Значит, если $n = p$ — простое (случай малой теоремы Ферма), то

$$a^{p-1} = 1 \pmod{p}.$$

Итак, при простом N равенство

$$a^{N-1} = 1 \pmod{N}$$

выполнено всегда, а вот для составных N его истинность маловероятна.

Поскольку вычисление $a^{N-1} \pmod{N}$ довольно быстрая операция, у нас возникает быстрый тест, проверяющий, является ли данное число составным. Его называют тестом Ферма по основанию a. Заметим, что тест Ферма может лишь убедить нас в том, что число N имеет делители, но не в состоянии доказать его простоту. Действительно, рассмотрим составное число $N = 11 \cdot 13 = 341$, а основание в тесте Ферма выберем равным 2. Тогда $2^{340} = 1 \mod 341$, в

то время как число 341 не простое. В этом случае говорят, что N — псевдопростое число (в смысле Ферма) по основанию 2. Существует бесконечно много псевдопростых чисел по фиксированному основанию, хотя они встречаются реже, чем простые. Можно показать, что для составного N вероятность неравенства

$$a^{N-1} \neq 1 \pmod{N}.$$

больше $1/2$. Подводя итог сказанному, выпишем алгоритм теста Ферма.

```
for (i=0; i<k; i++)
   { выбрать a из [2,...,n-1];
     b = a^{n-1} mod n;
     if (b!=1)
       { напечатать (составное, a);
         exit;
       }
   }
напечатать «правдоподобно простое»
```

Если на выходе этого алгоритма напечатано «`составное, a`», то мы с определенностью можем заявить, что число n не является простым, и что a свидетельствует об этом факте, т. е. чтобы убедиться в истинности высказывания, нам достаточно провести тест Ферма по основанию a. Такое a называют *свидетелем* того, что n составное.

Если же в результате работы теста мы получим сообщение: «правдоподобно простое», то можно заключить: n — составное с вероятностью $\leqslant 1/2^k$.

Возьмем, например, число $n = 43\,040\,357$. Оно является составным, что можно проверить тестом Ферма по основанию $a = 2$. Действительно,

$$2^{43\,040\,356} \pmod{43\,040\,357} = 9\,888\,212 \neq 1.$$

В качестве следующего примера рассмотрим число $n = 2^{192} - 2^{64} - 1$. Алгоритм выдаст на печать «правдоподобно простое», поскольку мы не сможем найти свидетеля обратному. Фактически, это число является простым, так что нет ничего удивительного в том, что мы не смогли найти свидетеля его делимости.

Тем не менее, существуют составные числа, относительно которых тест Ферма выдаст ответ

`правдоподобно простое`,

для всех взаимно простых с n оснований a. Такие числа называются

числами Кармайкла. К сожалению, их бесконечно много. Первые из них — это 561, 1105 и 1729. Числа Кармайкла обладают следующими свойствами:

- они нечетны,
- имеют по крайней мере три простых делителя,
- свободны от квадратов[1],
- если p делит число Кармайкла N, то $p-1$ делит $N-1$.

Чтобы получить представление об их распределении, посмотрим на числа, не превосходящие 10^{16}. Среди них окажется примерно $2,7 \cdot 10^{14}$ простых чисел, но только $246\,683 \approx 2,4 \cdot 10^5$ чисел Кармайкла. Следовательно, они встречаются не очень часто, но и не настолько редко, чтобы их можно было игнорировать.

8.1.3. Тест Миллера – Рабина

Из-за существования чисел Кармайкла тест Ферма применяют крайне неохотно. Существует модификация теста Ферма, называемая тестом Миллера – Рабина, которая обходит проблему составных чисел, для которых не находится свидетеля. Это не означает, что легко найти свидетеля для каждого составного числа, это значит, что такой свидетель в принципе должен быть. Кроме того, тест Миллера – Рабина принимает составное число за простое с вероятностью 1/4 для любого случайного основания a. Поэтому многократное повторение теста позволяет уменьшить вероятность ошибки до любой наперед заданной величины. Приведем тест Миллера – Рабина на псевдокоде.

```
Записать n-1 = 2^s m, с нечетным m;
for (j=0; j<k; j++)
   { выбрать a из [2,..., n-1];
     b = a^m mod n;
     if (b!=1)
       { флажок=истина;
         for (i=1; i<s; i++)
           { if (b==(n-1))
             { флажок=ложь;
               остановка алгоритма;
             }
             b=b^2 mod n;
           }
```

[1] Не делятся на квадраты простых чисел. — *Прим. перев.*

```
        if (флажок=истина)
          { напечатать (составное, a);
            exit;
          }
      }
  }
напечатать «правдоподобно простое»;
```

Мы не будем объяснять, почему тест Миллера – Рабина работает. Если Вам это интересно, то можете посмотреть любую книгу по алгоритмической теории чисел, например, книги Коэна или Бача и Шаллита, на которые есть точные ссылки в конце этой главы. Так же как и тест Ферма, тест Миллера – Рабина повторяют k раз с разными основаниями и получают результат с вероятностью ошибки $1/4^k$, если алгоритм выдает ответ «правдоподобно простое». Таким образом, можно ожидать, что если тест Миллера – Рабина после $k > 20$ проходов напечатает «правдоподобно простое», то тестируемое число окажется действительно простым.

Если тест Миллера – Рабина по основанию a нашел, что n — составное, то основание a называют свидетелем Миллера – Рабина делимости n, и согласно обобщенной гипотезе Римана (в справедливость которой верит большинство математиков) для составного числа n найдется свидетель Миллера Рабина, не превосходящий $O((\ln n)^2)$.

8.1.4. Доказательство простоты

До сих пор мы говорили только о свидетелях делимости чисел и интерпретировали наличие свидетелей как доказательство того, что тестируемое число является составным. Кроме того, у нас были методы, говорящие о том, что данное число правдоподобно простое, но стопроцентной уверенности в простоте чисел у нас не было (за исключением алгоритма пробного деления). Вероятностные ответы вполне приемлемы, поскольку вероятность того, что составное число выдержит тест Миллера – Рабина с двадцатью основаниями примерно равна 2^{-40}, чего никогда не встречается на практике. Но с теоретической точки зрения (и может быть с практической, если Вы одержимы паранойей) этого может оказаться недостаточным. Точнее говоря, нам может потребоваться по-настоящему простое число, а не только правдоподобно простое.

Есть алгоритмы, выдающие свидетельства простоты чисел. Такие свидетельства называют доказательством простоты. Фактиче-

ски, соответствующие алгоритмы применяются тогда, когда уже есть уверенность в простоте числа, но нет строгого доказательства этого факта. Другими словами, данное число уже выдержало тест Миллера – Рабина по нескольким основаниям и все, что остается — это *доказать* его простоту.

Наиболее успешный из алгоритмов, доказывающих простоту, основывается на эллиптических кривых и называется *ECPP* (от англ. Elliptic Curve Primality Prover). В свою очередь, этот алгоритм основывается на более старом алгоритме доказательства простоты, принадлежащем Поклингтону и Лемеру. Его вариант, опирающийся на эллиптические кривые, был разработан Гольдвассером и Килианом. Алгоритм *ECPP* случаен, т. е. нельзя гарантировать с математической точностью, что он всегда выдаст ответ — свидетеля, доказывающего простоту введенного в него простого числа. Если число, подаваемое на вход алгоритма, составное, то вообще нет уверенности в том, что алгоритм когда-либо остановится. Хотя алгоритм *ECPP* работает занимает полиномиальное время, т. е. он довольно эффективен, проверку получаемого им свидетеля простоты данного числа можно произвести еще быстрее.

Есть алгоритм, авторы которого — Адлеман и Хуанг. В отличие от алгоритма *ECPP* он гарантирует окончание работы и доказательство простоты введенного простого числа. Метод основывается на гиперэллиптических кривых, обобщающих эллиптические кривые и, насколько мне известно, никогда не был реализован. Алгоритм Адлемана и Хуанга никогда не осуществлялся не только из-за своей сложной математической структуры, но и потому, что несмотря на отсутствие уверенности в получении ответа, метод *ECPP* на практике дает тот же результат, что и алгоритм, основанный на гиперэллиптических кривых, но с гораздо меньшими усилиями.

8.2. Алгоритмы факторизации

Методы разложения на множители (или факторизации) можно разделить на средневековые методы, такие как

- пробное деление,
- $(P-1)$-метод,
- $(P+1)$-метод,
- ρ-метод Полларда;

и современные:

- метод непрерывных дробей ($CFRAC$),
- квадратичного решета,
- квадратичного решета в числовом поле.

Ни время, ни объем книги не позволяют нам обсуждать все эти методы подробно. Поэтому мы ограничимся парой средневековых методов и объясним основные идеи, питающие некоторые из современных алгоритмов.

Время работы современных алгоритмов лежит где-то между полиномиальным и экспоненциальным, т. е. в суб-экспоненциальной области. Их сложность измеряется функцией

$$L_N(\alpha,\beta) = \exp\Big((\beta + o(1))(\ln N)^{\alpha}(\ln\ln N)^{1-\alpha}\Big).$$

Заметим, что

- $L_N(0,\beta) = (\ln N)^{\beta+o(1)}$ — полиномиальная сложность,
- $L_N(1,\beta) = N^{\beta+o(1)}$ — экспоненциальная сложность.

Так что, как было замечено в главе 7, при $0 < \alpha < 1$ функция $L_N(\alpha,\beta)$ описывает сложность, лежащую между полиномиальной и экспоненциальной. Приведем оценки сложности некоторых алгоритмов факторизации в терминах этой функции.

- Наиболее медленный алгоритм, алгоритм пробного деления, имеет сложность $L_N(1,1/2)$.
- До начала 1990-ых годов самым быстрым из общих методов факторизации был метод квадратичного решета со сложностью $L_N(1/2,c)$ с некоторой константой c.
- Сложность наиболее приемлемого алгоритма на сегодняшний день, метода квадратичного решета в числовом поле, равна $L_N(1/3,c)$, где c — константа, не зависящая от разлагаемого числа N.

8.2.1. Пробное деление

Пробное деление — это самый элементарный алгоритм факторизации. Для разложения числа N на множители поступают следующим образом:

```
for (p=1; p<sqrt(N); p++)
   { e=0;
     if ((N mod p)==0) then
       { while ((N mod p)==0)
```

```
        { e=e+1;
          N=N/p;
        }
        вывести (p,e);
      }
  }
```

Даже поверхностное знакомство с алгоритмом показывает, что в самом плохом случае, — случае простого N, — он требует $O(\sqrt{N})$ операций. Размер входных данных пропорционален $\log_2 N$, т. е. этот алгоритм имеет экспоненциальную сложность. Но совсем пренебрегать этим методом не стоит. Он вполне приемлем для разложения чисел, не превосходящих 10^{12}.

8.2.2. Гладкие числа

Для факторизации больших чисел хотелось бы иметь на вооружении что-то более быстрое, чем пробное деление. Практически все остальные алгоритмы используют вспомогательные числа, называемые гладкими. По существу, к гладким относят те числа, которые легко раскладываются на множители методом пробного деления. Уточним этот термин.

Определение 8.2. (**Гладкие числа.**) Пусть B — целое число. Число N называется B-гладким, если любой его простой делитель меньше, чем B.

Например, число

$$N = 2^{78} \cdot 3^{89} \cdot 11^3$$

является 12-гладким. Иногда мы будем называть число *просто гладким*, если граница B его гладкости мала по сравнению с самим числом.

Количество y-гладких чисел, не превосходящих x, обозначается через $\psi(x, y)$. Это довольно сложная функция, которая аппроксимируется как

$$\psi(x, y) \approx x\rho(u),$$

где ρ — функция Дикмана–Де Брюина и

$$u = \frac{\ln x}{\ln y}.$$

Функция Дикмана–Де Брюина определяется как решение диффе-

ренциального уравнения

$$u\rho'(u) + \rho(u-1) = 0$$

при $u > 1$. В частности, $\rho(u)$ можно аппроксимировать выражением

$$\rho(u) \approx u^{-u},$$

которое имеет место при $u \to \infty$. Это подводит нас к результату, важному при анализе современных алгоритмов факторизации.

Теорема 8.3. Доля $x^{1/u}$-гладких целых чисел, не превосходящих x, асимптотически равна u^{-u}.

Положив теперь $y = L_N(\alpha, \beta)$, мы получим

$$u = \frac{\ln N}{\ln y} = \frac{1}{\beta}\left(\frac{\ln N}{\ln\ln N}\right)^{1-\alpha}.$$

Отсюда можно вывести, что

$$\frac{1}{N}\psi(N,y) \approx u^{-u} = \exp(-u\ln u) = \frac{1}{L_N(1-\alpha,\gamma)},$$

где γ — какая-то константа. Предположим, что нас интересуют числа, меньшие N, которые являются $L_N(\alpha,\beta)$-гладкими. Вероятность того, что любое число, меньшее N, действительно $L_N(\alpha,\beta)$-гладкое, равна $1/L_N(1-\alpha,\gamma)$. Это, будем надеяться, объяснит на интуитивном уровне, почему сложность некоторых современных методов разложения на множители приблизительно равна $L_N(1/2, c)$. В квадратичном решете в числовом поле получается более высокая скорость только благодаря усложненному математическому алгоритму.

При дальнейших обсуждениях нам потребуется понятие *показательно B-гладких* чисел.

Определение 8.4. (Показательная гладкость.) Число N называют показательно B-гладким, если любая степень простого числа, делящая N, меньше B.

Например, число $N = 2^5 \cdot 3^3$ является показательно 33-гладким, т. к. максимальная степень простого числа, делящая N, — это $2^5 = 32$.

8.2.3. $(P-1)$-метод Полларда

Самое известное имя конца двадцатого века, связанное с алгоритмами факторизации, — это имя Джона Полларда. Практически все важные продвижения в факторизации были сделаны им, например,

- $(P-1)$-метод,

– ρ-метод,

– метод квадратичного решета в числовом поле.

В этом параграфе обсуждается $(P-1)$-метод, а в следующем мы рассмотрим метод квадратичного решета в числовом поле, оставив остальные методы в качестве упражнений.

Пусть число N, которое мы планируем разложить на множители, равно произведению двух простых $N = p \cdot q$. Кроме того, будем считать, что нам известно такое целое число B, что $p-1$ является показательно B-гладким, а $q-1$ таковым не является. Тогда можно надеяться, что $p-1$ делит $B!$, в то время как $q-1$, скорее всего, этого числа не делит.

Предположим, что мы нашли

$$A = 2^{B!} \pmod{N}.$$

Представив себе, что мы можем подсчитать то же самое по модулям p и q, мы получим

$$A = 1 \pmod{p},$$

поскольку $p-1$ делит $B!$ и, по малой теореме Ферма, $A^{p-1} = 1 \pmod{p}$.

С другой стороны, равенство

$$A = 1 \pmod{q}$$

маловероятно. Следовательно, p делит $A-1$, а q этого числа не делит. Поэтому можно восстановить p, вычисляя НОД $(A-1, N)$.

На псевдокоде этот алгоритм выглядит следующим образом:

```
A=2;
for (j=2; j<=B; j++)
   { A=A^j mod N; }
p=НОД(a-1,N);
if (p!=1 и p!=N) then
     вывести: «p --- делитель N»;
else
     вывести: «результат не получен»
```

Попытаемся разложить на множители число $N = 15\,770\,708\,441$. Выберем $B = 180$ и прогоним этот алгоритм. В результате получим

$$A = 2^{B!} \pmod{N} = 1\,162\,022\,425.$$

Применяя алгоритм Евклида, найдем

$$p = \text{НОД}\,(A-1, N) = 135\,979.$$

Обратите внимание на то, что в нашем примере $N = 135\,979 \cdot 115\,979$. Поэтому

$$p - 1 = 135\,979 - 1 = 2 \cdot 3 \cdot 131 \cdot 173,$$
$$q - 1 = 115\,979 - 1 = 2 \cdot 103 \cdot 563.$$

Следовательно, $p-1$ действительно показательно 180-гладкое и даже 174-гладкое, в то время как $q - 1$ таковым не является.

Можно показать, что сложность $(P-1)$-метода оценивается как

$$O(B \ln B (\ln N)^2 + (\ln N)^3).$$

Значит, выбор $B = O((\ln N)^i)$ при некотором натуральном i обеспечивает полиномиальную сложность алгоритма, дающего, правда, результат только для чисел специального вида.

Ввиду существования $(P-1)$-метода Полларда для алгоритма *RSA* рекомендуют выбирать простые числа вида

$$p = 2p_1 + 1 \quad \text{и} \quad q = 2q_1 + 1,$$

где p_1, q_1 — тоже простые. В этой ситуации p и q называются защищенными простыми. Однако такая рекомендация сегодня не является строго необходимой, если в приложениях *RSA* используется большой модуль. Дело в том, что слишком мала вероятность числу $p-1$ оказаться показательно B-гладким с малым B, если p выбрано случайным образом и насчитывает 512 двоичных знаков. Следовательно, выбор случайного 512-значного двоичного простого числа скорее всего сделает $(P-1)$-метод Полларда бесполезным.

8.2.4. Разность квадратов

Основной прием в алгоритмах факторизации, известный уже много веков, заключается в генерировании двух целых чисел x и y приблизительно той же величины, что и N, удовлетворяющих условию

$$x^2 = y^2 \pmod N.$$

Применяя формулу сокращенного умножения, получим

$$x^2 - y^2 = (x-y)(x+y) = 0 \pmod N.$$

Если $N = p \cdot q$, то имеют место четыре возможности:

1) p делит $x - y$, и q делит $x + y$;
2) p делит $x + y$, и q делит $x - y$;
3) как p, так и q делят $x - y$, но не делят $x + y$,
4) как p, так и q делят $x + y$, но не делят $x - y$.

Эти случаи встречаются с равной вероятностью, а именно 1/4. Если мы теперь вычислим

$$d = \text{НОД}\,(x - y, N),$$

то в зависимости от ситуации может получиться одно из четырех:

1) $d = p$, 2) $d = q$, 3) $d = N$, 4) $d = 1$.

Поскольку эти равенства равновероятны, с вероятностью 1/2 мы найдем нетривиальный делитель числа N. Остается только понять, как же подобрать подходящие x и y, разность квадратов которых делится на N.

8.3. Современные методы факторизации

Современные методы разложения чисел на множители построены на следующей стратегии, основанной на разности квадратов, о чем мы с Вами только что говорили.

- Выбирают границу гладкости B.
- Вычисляют *базу простых чисел*, не превосходящих B (или *факторбазу* F).
- Находят большое количество пар B-гладких чисел x и y, удовлетворяющих условию $x = y \pmod N$. Такие пары называют *соотношениями* на факторбазе.
- С помощью линейной алгебры по модулю 2 ищут комбинацию соотношений, которая дает X и Y с условием $X^2 = Y^2 \pmod N$.
- Пытаются найти делитель N, вычисляя НОД $(X - Y, N)$.

Самое трудное во всех алгоритмах, использующих такую стратегию, — это поиск соотношений. Указанная стратегия может применяться и для дискретного логарифмирования, о чем мы упоминали в предыдущей главе. Здесь мы расскажем о тех частях, которые являются общими для всех алгоритмов, разлагающих на множители, и объясним, почему они работают.

Один из подходов к факторизации основан на теории конечных групп. Мы уже говорили, что информация о порядке группы $(\mathbb{Z}/N\mathbb{Z})^*$ эквивалентна разложению числа N на множители. Поэтому можно сказать, что задача факторизации сводится к определению порядка группы. Факторбаза, по существу, является множеством образующих группы $(\mathbb{Z}/N\mathbb{Z})^*$, в то время как соотношения — суть соотношения между этими образующими. Как только достаточно большое количество соотношений между образующими будет найдено, с помощью стандартных алгоритмов теории групп можно бу-

дет выявить структуру группы, а значит и ее порядок. Дело еще упрощается тем, что группа абелева и поэтому не очень сложная. Алгоритмы, выясняющие структуру группы, могут, в частности, опираться на нормальную форму Смита ассоциированной матрицы. Поэтому в применении линейной алгебры к соотношениям для факторизации целого числа нет ничего неожиданного.

8.3.1. Комбинирование соотношений

Алгоритм приведения матрицы к нормальной форме Смита слишком сложен для тех алгоритмов факторизации, в которых используются более элементарные методы линейной алгебры, что мы сейчас и собираемся продемонстрировать. Предположим, что у нас есть такие соотношения:

$$p^2q^5r^2 = p^3q^4r^3 \pmod N,$$
$$pq^3r^5 = pqr^2 \pmod N,$$
$$p^3q^5r^3 = pq^3r^2 \pmod N,$$

где p, q и r — простые числа из нашей факторбазы F. Разделив левую часть каждого из равенств на правую, получим

$$\begin{aligned} p^{-1}qr^{-1} &= 1 \pmod N,\\ q^2r^3 &= 1 \pmod N,\\ p^2q^2r &= 1 \pmod N. \end{aligned} \tag{8.1}$$

Перемножая последние два соотношения, находим

$$p^2q^4r^4 = 1 \pmod N.$$

Следовательно, если $X = pq^2r^2$, а $Y = 1$, то

$$X^2 = Y^2 \pmod N,$$

и мы можем с вероятностью пятьдесят процентов найти делитель, вычисляя НОД $(X - Y, N)$.

В этом примере легко усмотреть необходимые преобразования, позволяющие получить разность квадратов, делящуюся на N. На практике факторбаза может состоять из сотен тысяч простых чисел, и мы можем найти примерно такое же количество соотношений. Поэтому необходимо как-то автоматизировать процесс комбинирования соотношений для получения требуемой разности квадратов. В этом нам помогает линейная алгебра.

Объясним применение методов линейной алгебры на предыдущем простом примере. Напомним, что соотношения в нем эквива-

лентны соотношениям (8.1). Чтобы определить, какие из них стоит перемножить и получить полный квадрат, выписывают матрицу A, столбцы которой соответствуют простым числам, участвующим в соотношениях (p, q, r в нашем случае), а строки — уравнениям. При этом от каждого уравнения берутся только показатели простых чисел, причем по модулю 2. В нашем примере матрица имеет вид:

$$A = \begin{pmatrix} -1 & 1 & -1 \\ 0 & 2 & 3 \\ 2 & 2 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \pmod 2.$$

Найдем теперь такой двоичный вектор Z, что $ZA = 0 \pmod 2$. В нашем примере искомый вектор — $Z = (0\ 1\ 1)$. Действительно,

$$(0\ 1\ 1) \begin{pmatrix} -1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} = (0\ 0\ 0) \pmod 2.$$

Вектор Z говорит нам, что умножая друг на друга два последних соотношения, мы получим полный квадрат по модулю N.

Поиск вектора Z можно осуществить с помощью некоторого варианта элементарных преобразований Гаусса. Как правило, в общей ситуации нам необходимо иметь больше уравнений (т. е. соотношений), чем элементов базы множителей. Комбинирование соотношений в целях поиска полного квадрата — обычно самая трудная часть алгоритмов факторизации, поскольку соответствующие матрицы имеют очень большие размеры. Например, при применении квадратичного решета в числовом поле к факторизации стозначного десятичного числа может потребоваться матрица с более чем 100 000 строками и столбцами. В результате создаются большие проблемы с распределением компьютерной памяти, которые пытаются обойти, создавая специальные программы обработки матриц или специализированные суперкомпьютеры.

Матрицы, появляющиеся при разложении интересных для криптографии чисел, имеют порядка 500 000 строк и столько же столбцов. Поскольку матрицы двоичные, для записи каждого элемента достаточно одного бита. Если бы у нас были плотно заполненные матрицы, то для ее записи в компьютер нам бы потребовалось около 29 гигабайт. К счастью, матрицы очень сильно разрежены, поэтому места для их хранения требуется не слишком много.

Как мы говорили выше, вектор Z, аннулирующий матрицу, можно искать, применяя разновидность элементарных преобразований

Гаусса над полем $\mathbb{Z}/2\mathbb{Z}$. Стандартный метод Гаусса преобразует матрицу к верхнетреугольному виду, который может оказаться плотным. Так что применяя его, мы опять приходим к проблеме памяти. Для преодоления возникающей трудности развиты довольно хорошие матричные алгоритмы, которые стараются не менять матрицу вообще. Мы не будем их здесь обсуждать, а отошлем интересующихся к книге Ленстры и Ленстры, на которую есть точная ссылка в разделе «Дополнительная литература».

Мы еще не рассказали о том, как найти соотношения. Этому как раз и посвящен следующий параграф.

8.4. Метод решета в числовом поле

Квадратичное решето в числовом поле — самый быстрый из известных алгоритмов разложения на множители. Основная его идея состоит в поиске целых чисел x и y, разность квадратов которых делится на N. Как уже было сказано, после этого можно надеяться, что НОД $(x-y, N)$ — нетривиальный делитель числа N.

Чтобы объяснить работу этого метода, мы начнем с линейного решета и покажем, как оно может быть обобщено до решета в числовом поле. Линейное решето — не очень хороший алгоритм, на котором, однако, демонстрируются основные идеи эффективных алгоритмов.

8.4.1. Линейное решето

Предположим, что мы собираемся разложить на множители B-гладкое число N с некоторой границей B. Для этого надо сформировать факторбазу «малых» простых чисел

$$F = \{p |\ p \leqslant B\}.$$

Идея линейного решета состоит в поиске таких чисел a и λ, для которых комбинация $b = a + N\lambda$ является B-гладкой. Причем a тоже нужно выбирать среди B-гладких чисел. Тогда будут иметь место разложения

$$a = \prod_{p \in F} p^{a_p} \quad \text{и} \quad b = a + \lambda N = \prod_{p \in F} p^{b_p},$$

из которых получаются следующие соотношения на факторбазе:

$$\prod_{p \in F} p^{a_p} = \prod_{p \in F} p^{b_p} \pmod{N}.$$

Таким образом, основной вопрос линейного решета звучит так: как найти нужные значения a и λ? Их поиск осуществляют по следующей схеме:

- фиксируют произвольное значение λ;
- заводят просеивающий массив с $A+1$ нулевыми элементами, пронумерованными от 0 до A;
- для каждого простого числа $p \in F$ прибавляют $\log_2 p$ к каждому значению ячейки массива, чей номер сравним с $-\lambda N$ по модулю p;
- выбирают a как номер ячейки, значение которой превышает определенную пороговую величину, например, является максимальным элементом массива.

Дело в том, что при удачном выборе пороговой величины взятый номер ячейки, будучи сложенным с λN, имеет хорошие шансы оказаться B-гладким и, одновременно, делиться на большое количество простых чисел из F.

Разберем пример. Пусть $N = 1159$ и $F = \{2, 3, 5, 7, 11\}$. Выберем $\lambda = -2$. Наша цель — найти гладкое число вида $a - 2N$. Вводим массив:

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

Берем первое простое число из F, а именно, $p = 2$, и вычисляем $-\lambda N \pmod p = 0$. Прибавляем теперь $\log_2 2 = 1$ к содержимому ячеек с четными номерами.

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

Переходим к $p = 3$ и находим $-\lambda N \pmod 3 = 2$. Добавляем $\log_2 3 = 1{,}6$ к ячейкам с номерами, равными 2 по модулю 3:

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

Продолжая процесс для простых $p = 5$, 7 и 11, заполняем массив полностью:

0	1	2	3	4	5	6	7	8	9
1,0	2,8	2,6	2,3	1,0	1,6	1,0	0,0	11,2	0,0

Содержимое восьмой ячейки максимально. Поэтому мы подозреваем, что значению $a = 8$ соответствует гладкое и подходящее для наших целей число, в чем несложно убедиться, поскольку

$$b = a - \lambda N = 8 - 2 \cdot 1159 = -2310 = -2 \cdot 3 \cdot 5 \cdot 7 \cdot 11.$$

Итак, с помощью линейного решета мы получим большую коллекцию чисел a и b, удовлетворяющих соотношениям вида

$$a_i = \prod_{p_j \in F} p_j^{a_{i,j}} = \prod_{p_j \in F} p_j^{b_{i,j}} = b_i \pmod N.$$

Имея по крайней мере $|B| + 1$ таких соотношений, можно сформировать матрицу со строками

$$(a_{i,1}, \ldots, a_{i,t}, b_{i,1}, \ldots, b_{i,t}) \pmod 2.$$

Затем найдем аннулирующий эту матрицу вектор Z, который подскажет, как перемножить соотношения, чтобы получить равенство вида

$$x^2 = y^2 \pmod N.$$

Используя его, можно попытаться найти делитель числа N. Если попытка окажется неудачной, нужно найти новый аннулирующий вектор и сформировать другую разность квадратов.

Основной вариант линейного решета дает слишком бедный урожай соотношений. Существует его разновидность, называемая *вариацией больших простых чисел*, которая ослабляет условия линейного решета, позволяя парам a и b быть почти B-гладкими, допуская по одному «большому» простому делителю этих чисел. При этом «большие» простые числа комбинируются таким образом, что аппарат линейной алгебры, привлекаемый к линейному решету, работает без изменения. Это делается с помощью построения графов и алгоритмов, вычисляющих базисы в множестве циклов графа. Вариация больших простых чисел появилась в квадратичном решете, но работает в любых просеивающих алгоритмах, раскладывающих числа на множители.

Ясно, что просеивание можно осуществлять параллельно. Поэтому такой алгоритм можно распределить между большим числом компьютеров по всему миру. Подчиненные компьютеры сообщают все найденные ими соотношения на главный сервер, который осуществляет заключительную стадию алгоритма, связанную с линейной алгеброй. Таким образом Интернет можно превратить в «один большой» вычислительный центр, занимающийся задачей факторизации. Заключительный этап с линейной алгеброй, как мы уже отмечали,

часто требует специального оснащения и большого объема памяти. Поэтому последний этап невозможно распределить между дочерними компьютерами.

8.4.2. Решето в числовом поле

Линейное решето — простейший, но не слишком хороший метод факторизации больших чисел. И действительно, линейное решето никогда не предлагалось как практический алгоритм разложения на множители. Однако его работа поучительна для восприятия других просеивающих алгоритмов. Решето в числовом поле использует для построения соотношений между элементами факторбазы арифметику конечных полей. Собственно, это единственная модификация по сравнению с линейным решетом. Этап линейной алгебры, вариация больших простых чисел и распределение задачи между разными компьютерами, — все остается практически без изменения в алгоритме решета в числовом поле. Расскажем теперь об этом алгоритме, но в существенно более простой его форме, чем используемая в реальной жизни. Читатель, незнакомый с алгебраической теорией чисел, может пропустить этот параграф.

Сначала строят нормированные неприводимые многочлены f_1 и f_2 степеней d_1 и d_2 с целыми коэффициентами, удовлетворяющие условию

$$f_1(m) = f_2(m) = 0 \pmod{N}$$

для некоторого $m \in \mathbb{Z}$. Алгоритм пользуется арифметикой числовых полей

$$K_1 = \mathbb{Q}(\theta_1) \quad \text{и} \quad K_2 = \mathbb{Q}(\theta_2),$$

где $f_1(\theta_1) = f_2(\theta_2) = 0$. Зафиксируем обозначения для двух гомоморфизмов

$$\varphi_i : \begin{cases} \mathbb{Z}[\theta_i] \longrightarrow \mathbb{Z}/N\mathbb{Z}, \\ \theta_i \mapsto m. \end{cases}$$

Как и в линейном решете, мы стремимся найти множество

$$S \subset \{(a,b) \in \mathbb{Z}^2 | \text{ НОД}(a,b) = 1\}$$

таких элементов, что

$$\prod_{(a,b)\in S} (a - b\theta_1) = \beta^2 \quad \text{и} \quad \prod_{(a,b)\in S} (a - b\theta_2) = \gamma^2,$$

где $\beta \in K_1$, а $\gamma \in K_2$. Вычислив β и γ, получим

$$\varphi_1(\beta)^2 = \varphi_2(\gamma)^2 \pmod N$$

и можно надеяться, что число

$$\text{НОД}\,(N; \varphi_1(\beta) - \varphi_2(\gamma))$$

является нетривиальным делителем N.

В связи с этим можно сформулировать три очевидные задачи.

- Как найти множество S?
- Как вычислить β по известному элементу $\beta^2 \in \mathbb{Q}(\theta_1)$?
- Как подобрать многочлены f_1 и f_2 из первой части алгоритма?

8.4.3. Как найти множество S?

Аналогично процессу в линейном решете, с помощью линейной алгебры мы строим множество S так, чтобы разности

$$a - b\theta_1 \quad \text{и} \quad a - b\theta_2$$

были «гладкими». Необходимо уточнить, что в данной ситуации называть гладкими объектами. Для этого придется привлечь теорию числовых полей. Обобщая старое определение гладких целых чисел на случай алгебраических целых, мы получаем следующее определение:

Определение 8.5. Алгебраическое целое число называют *«гладким»*, если и только если идеал, им порожденный, делится только на «маленькие» простые идеалы.

Положим по определению: $F_i(X, Y) = Y^{d_i} f_i(\frac{X}{Y})$. Тогда

$$N_{\mathbb{Q}(\theta_i)/\mathbb{Q}}(a - b\theta_i) = F_i(a, b).$$

Мы определим две факторбазы, по одной для каждого многочлена:

$$\mathcal{F}_i = \{(p, r) |\ p \text{ — простое, } r \in \mathbb{Z}, \text{ причем } f_i(r) = 0 \pmod p\}.$$

Каждый элемент факторбазы $\mathcal{F}_i$ соответствует степени простого идеала в подкольце $\mathbb{Z}[\theta_i]$ целых элементов $\mathcal{O}_{\mathbb{Q}(\theta_i)}$ поля $\mathbb{Q}(\theta_i)$. Идеал имеет вид

$$\langle p, \theta_i - r\rangle = p\mathbb{Z}[\theta_i] + (\theta_i - r)\mathbb{Z}[\theta_i].$$

Для данных значений a и b мы можем легко определить, «раскладывается» ли идеал $\langle a - \theta_i b\rangle$ над нашей факторбазой. Обратите внимание на то, что слово «раскладывается» взято в кавычки. Дело в том, что теорема о единственности разложения на простые множители неверна в кольце $\mathbb{Z}[\theta_i]$, а справедлива лишь в кольце целых,

т. е. в $\mathcal{O}_{\mathbb{Q}(\theta_i)}$. Тем не менее, отсутствие однозначности разложения не препятствует нашим целям. Чтобы убедиться в этом, достаточно обратиться к книге Ленстры и Ленстры.

Если $\mathbb{Z}[\theta_i] = \mathcal{O}_{\mathbb{Q}(\theta_i)}$, то сформулированный ниже порядок действий дает действительно единственное разложение идеала $\langle a - \theta_i b\rangle$ в произведение простых.

- Положим
$$F_i(a,b) = \prod_{(p_j,r)\in\mathcal{F}_i} p_j^{s_j^{(i)}}.$$
- Если идеал, соответствующий паре (p,r), нетривиальным образом входит в разложение элемента $a-\theta_i b$, то возникает равенство $(a:b) = (r:1) \pmod p$ элементов проективной прямой над полем $\mathbb{F}_p$.
- Получаем
$$\langle a-\theta_i b\rangle = \prod_{(p_j,r)\in\mathcal{F}_i} \langle p_j, \theta_i - r\rangle^{s_j^{(i)}}.$$

Эта процедура подводит нас к алгоритму, просеивающему элементы a и b, для которых идеал $\langle a-\theta_i b\rangle$ раскладывается над факторбазой. Как и в случае линейного решета, просеивание позволяет избегать большого количества дорогостоящих попыток деления в надежде определить гладкие идеалы. Нам остается только проверить разложимость элементов, вероятность разложения которых достаточно большая.

- Фиксируем a.
- Задаем просеивающий массив для b $(-B \leqslant b \leqslant B)$, заполняя его по следующему правилу:
$$S[b] = \log_2(F_1(a,b)\cdot F_2(a,b)).$$
- Для каждой пары $(p,r) \in \mathcal{F}_i$ вычитаем $\log_2 p$ из каждого элемента массива с номером b, удовлетворяющего условию:
$$a - rb = 0 \pmod p.$$
- Числа b, которые нам нужны, соответствуют элементам $S[b]$, лежащим ниже некоторого допустимого уровня.

Если уровень допуска установлен разумным образом, то с большой долей вероятности $F_1(a,b)$ и $F_2(a,b)$ разложатся на простые идеалы из факторбазы и, быть может, будут иметь в качестве множителей

по одному «большому» простому идеалу. Мы перепишем эти разложения в виде соотношений, как уже делали в линейном решете.

Затем, после применения линейной алгебры, мы найдем искомое подмножество S всех таких пар (a, b), что

$$\prod_{(a,b)\in S} \langle a - \theta_i b\rangle = \text{квадрат } \textbf{идеала} \text{ в } \mathbb{Z}[\theta_i].$$

Однако это еще не все. Напомним, что мы хотели получить произведение $\prod a - \theta_i b$, которое будет квадратом **элемента** из $\mathbb{Z}[\theta_i]$. Чтобы обойти эту проблему, нам нужно добавить информацию из «бесконечного числа» мест. Пусть q — такое рациональное простое число (не лежащее ни в $\mathcal{F}_1$, ни в $\mathcal{F}_2$), для которого найдется s_q с $f_i(s_q) = 0 \pmod q$ и $f_i'(s_q) \neq 0 \pmod q$ для какого-нибудь из значений $i = 1$ или 2. Потребуем тогда дополнительного условия, а именно,

$$\prod_{(a,b)\in S} \left(\frac{a - bs_q}{q}\right) = 1,$$

где $\left(\frac{\cdot}{q}\right)$ — символ Лежандра. Ввиду мультипликативности символа Лежандра, мы можем подставить это дополнительное условие в нашу матрицу. Сделаем это для нескольких простых q. Поэтому мы выберем некое множество простых рациональных q и добавим соответствующие символы в нашу матрицу как дополнительные столбцы нулей и единиц в соответствии с правилом

$$\text{если } \left(\frac{a - bs_q}{q}\right) = \begin{cases} 1, & \text{ставим } 0, \\ -1, & \text{ставим } 1. \end{cases}$$

Найдя достаточно много соотношений, мы рассчитываем найти и такое подмножество S, что

$$\prod_S (a - b\theta_1) = \beta^2 \quad \text{и} \quad \prod_S (a - b\theta_2) = \gamma^2.$$

8.4.4. Как извлекать квадратные корни?

Теперь нужно научиться извлекать квадратные корни из элементов для восстановления β и γ по известным β^2 и γ^2. Покажем, как это делается для β. Квадрат задан в виде

$$\beta^2 = \sum_{j=0}^{d_1-1} a_j \theta_1^j,$$

где a_j — очень большие целые числа. Нам необходимо найти целочисленные решения b_j уравнения

$$\left(\sum_{j=0}^{d_1-1} b_j\theta_1^j\right)^2 = \sum_{j=0}^{d_1-1} a_j\theta_1^j.$$

Способ его решения принадлежит Кувеню. Он сводится к вычислению квадратных корней по модулю большого количества очень и очень больших простых чисел p. После этого применяется китайская теорема об остатках с надеждой восстановить искомый корень. Это наиболее простой для изучения метод, хотя существуют и более продвинутые способы, такие, например, как способ Нгуена.

8.4.5. Выбор начальных многочленов

Эта часть метода решета в числовом поле на сегодняшний день больше смахивает на черную магию. Мы требуем выполнения только одного условия:

$$f_1(m) = f_2(m) = 0 \pmod N,$$

хотя существуют веские эвристические аргументы в пользу выбора многочленов с дополнительными свойствами:

- Многочлены обладают малыми коэффициентами.
- Многочлены имеют «много» вещественных корней. Заметим, что случайный многочлен четной степени вообще не имеет вещественных корней.
- f_1 и f_2 имеют «много» корней по модулю нескольких малых простых чисел.
- Группы Галуа многочленов f_i должны быть небольшими.

Иногда приходится тратить несколько недель на поиск хорошей пары многочленов перед тем, как запустить соответствующий алгоритм факторизации. Известно несколько стратегий, применяющихся для их поиска. Как только несколько кандидатов на требуемые многочлены найдено, проводится серия экспериментальных просеиваний, чтобы отобрать наиболее удачную пару, выдающую нам большее число соотношений. Только после такого отбора имеет смысл приступать к реальной задаче факторизации.

8.4.6. Пример

Я благодарен Ричарду Пинчу, позволившему мне включить в книгу пример, о котором идет речь в этом разделе. Пример взят из его

записок лекционного курса в Кембридже, прочитанного в середине 1990-ых годов.

Предположим, мы хотим разложить на множители число $N = 290^2 + 1 = 84\,101$. Возьмем $f_1(x) = x^2 + 1$ и $f_2(x) = x - 290$, а число $m = 290$. Тогда

$$f_1(m) = f_2(m) = 0 \pmod{N}.$$

С одной стороны, у нас есть кольцо $\mathbb{Z}[i]$ целых[1] элементов поля $\mathbb{Q}(i)$, а с другой — стандартное кольцо целых чисел $\mathbb{Z}$.

Мы получаем следующие разложения:

x	y	$N(x - iy)$	множители	$x - my$	множители
-38	-1	1445	$(5)(17^2)$	252	$(2^2)(3^2)(7)$
-22	-19	845	$(5)(13^2)$	(5488)	$(2^4)(7^3)$

Затем мы находим два разложения, которые являются настоящими разложениями элементов, поскольку $\mathbb{Z}[i]$ — область с единственным разложением на множители:

$$-38 + i = -(2 + i)(4 - i)^2, \qquad -22 + 19i = -(2 + i)(3 - 2i)^2.$$

Следовательно, после применения элементарных методов линейной алгебры мы обретаем следующие «квадраты»:

$$(-38 + i)(-22 + 19i) = (2 + i)^2(3 - 2i)^2(4 - i)^2 = (31 - 12i)^2$$

и

$$(-38 + m)(-22 + 19 \times m) = (2^6)(3^2)(7^4) = 1176^2.$$

Применяя отображение φ_1 к элементу $31 - 12i$, находим

$$\varphi_1(31 - 12i) = 31 - 12 \times m = -3449.$$

Теперь

$$\begin{aligned}(-3449)^2 &= \varphi_1(31 - 12i)^2 = \varphi_1((31 - 12i)^2) = \\ &= \varphi_1((-38 + i)(-22 + 19i)) = \varphi_1(-38 + i)\varphi_1(-22 + 19i) = \\ &= (-38 + m)(-22 + 19 \times m) = 1176^2 \pmod{N}.\end{aligned}$$

Далее вычисляем

$$\text{НОД}\,(N, -3449 + 1176) = 2273 \quad \text{и} \quad \text{НОД}\,(N, -3449 - 1176) = 37.$$

Следовательно, 37 и 2273 — делители числа $N = 84\,101$.

[1] Здесь i — мнимая единица, т. е. $i^2 = -1$. — *Прим. перев.*

Краткое содержание главы

- Простые числа встречаются довольно часто, а вероятность того, что случайное n-битовое число будет простым, равна $1/n$.
- Исследовать числа на простоту можно с помощью вероятностных тестов, таких как тест Ферма или алгоритм Миллера–Рабина. Недостаток теста Ферма заключается в том, что существуют составные числа, которые выдерживают этот тест по всем возможным основаниям.
- Для обоснования простоты какого-либо числа применяются алгоритмы доказательства простоты, требующие полиномиального времени.
- Алгоритмы факторизации часто основаны на вычислении разности квадратов.
- Современные алгоритмы факторизации разбиваются на две стадии. На первой из них подбирается коллекция соотношений на факторбазе с помощью процесса просеивания, который можно осуществить, разделив задачу между множеством компьютеров через Интернет. На второй — эти соотношения подвергаются преобразованиям методами линейной алгебры на большом центральном сервере. Для окончания факторизации вычисляется разность квадратов.

Дополнительная литература

Стандартная книга по вычислительной теории чисел, содержащая многие алгоритмы факторизации и доказательства простоты, — это книга Коэна. Произведение Бача и Шаллита — тоже хороший источник тестов на простоту. Основная работа, освещающая метод решета в числовом поле, принадлежит Ленстре и Ленстре.

E. Bach and J. Shallit. *Algorithmic Number Theory. Volume 1: Efficient Algorithms*. MIT Press, 1996.

H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.

A. Lenstra and H. Lenstra. *The Development of the Number Field Sieve*. Springer-Verlag, 1993.

Контрольные вопросы

8.1.1. В каких случаях стоит использовать метод пробного деления для тестирования чисел на простоту?

8.1.2. Что такое тест Ферма и в чем его главный недостаток?

8.1.3. В чем состоит основное различие между алгоритмами, тестирующими числа на простоту (например, алгоритм Миллера–Рабина) и доказывающими простоту (например, *ECCP*)?

8.1.4. Что подразумевают под гладкими числами и как их используют при разложении на множители?

8.1.5. Опишите $(P-1)$-метод факторизации.

8.1.6. Почему наличие пары чисел x и y с условием $x^2 = y^2 \pmod N$ позволяет найти нетривиальный делитель числа N с вероятностью $1/2$?

8.1.7. Что понимают под просеиванием и почему его используют в современных алгоритмах разложения?

Лабораторные работы

8.2.1. Реализуйте тесты Ферма и Миллера–Рабина, описанные в тексте, и найдите с их помощью свидетелей делимости числа $2^{1024} - 3$.

8.2.2. Напишите программу для $(P-1)$-метода факторизации. Насколько большие составные числа Ваша программа сможет разложить на множители?

Упражнения

8.3.1. Покажите, что если составное число N выдерживает тест Ферма по основанию a, но не проходит тест Миллера–Рабина по тому же основанию, то мы можем разложить N на множители.

В следующих трех задачах раскрывается еще один метод Полларда факторизации $N = p \cdot q$, называемый ρ-методом. Пусть

$$x_0 = 2, \quad x_{i+1} = x_i^2 + 1 \pmod N.$$

8.3.2. Покажите, что если найти два разных элемента последовательности, удовлетворяющих условию $x_i = x_j \pmod p$, то с высокой вероятностью мы можем разложить N на множители, вычисляя

$$\text{НОД}\,(x_i - x_j, N).$$

8.3.3. Найдите аргументы, подтверждающие, что мы обязательно найдем разные элементы последовательности с условием $x_i = x_j \pmod{p}$.

8.3.4. Объясните, как найти такие элементы, используя небольшое количество памяти.

8.3.5. $(P-1)$-метод Полларда основывается на арифметике поля $\mathbb{F}_p$ и позволяет раскладывать на множители произведения pq с гладким $p-1$ и не гладким $q-1$. Разработайте аналогичный метод, опирающийся на арифметику поля $\mathbb{F}_{p^2}$, позволяющий факторизовать числа с гладким $p+1$. Такой метод носит название $(P+1)$-метода Полларда.

8.3.6*. Обобщите $(P-1)$- и $(P+1)$-методы на эллиптические кривые следующим образом. Рассмотрите эллиптическую кривую E над кольцом $\mathbb{Z}/N\mathbb{Z}$ и покажите, что в случае гладкого числа $\#E(\mathbb{F}_p)$ число N можно разложить, опираясь на закон сложения точек эллиптической кривой. Это обобщение носит название метода факторизации Ленстры.

ГЛАВА 9

ДИСКРЕТНЫЕ ЛОГАРИФМЫ

Цели главы

- Исследовать алгоритмы, решающие проблему дискретного логарифмирования.
- Ввести алгоритм Полига–Хеллмана.
- Рассказать об алгоритме шаги младенца/шаги гиганта.
- Объяснить методы Полларда.
- Показать, как в конечных полях с помощью методов, похожих на алгоритмы факторизации, вычисляются дискретные логарифмы.
- Описать известные результаты о дискретном логарифмировании на эллиптических кривых.

9.1. Введение

Здесь мы познакомимся с известными методами дискретного логарифмирования в различных группах, т. е. способами решения уравнения

$$H = G^x$$

относительно неизвестной x, где H и G — элементы конечной абелевой группы. Эти методы можно разделить на две категории: общие алгоритмы, применяемые к любой конечной абелевой группе, или специфические алгоритмы, разработанные для специальных групп. Начав знакомство с общих, к концу главы мы подойдем к специальным способам.

9.2. Метод Полига–Хеллмана

Первое наблюдение, связанное с дискретным логарифмированием в абелевой группе A, состоит в том, что сложность этой задачи совпадает с ее сложностью в максимальной подгруппе простого порядка.

Такое наблюдение было сделано Полигом и Хеллманом и справедливо в любой конечной абелевой группе. Вот его обоснование.

Предположим, что у нас есть конечная циклическая абелева группа $A = \langle G \rangle$, чей порядок имеет следующее разложение на простые множители:

$$N = \#A = \prod_{i=1}^{t} p_i^{e_i}.$$

Допустим, нам дан элемент $H \in \langle G \rangle$, т. е. существует натуральное число x, для которого $H = G^x$. Наша цель — найти x. Можно сначала отыскать решения по всем модулям $p_i^{e_i}$, а затем применить китайскую теорему об остатках и восстановить решение по модулю N.

Из теории групп известно, что существует групповой изоморфизм

$$\Phi : A \longrightarrow C_{p_i^{e_i}} \times \cdots \times C_{p_t^{e_t}},$$

где C_{p^e} — циклическая группа порядка p^e. Проекция отображения Φ на компоненту C_{p^e} задается формулой

$$\Phi_p : \begin{cases} A \longrightarrow C_{p^e} \\ f \mapsto f^{N/p^e}. \end{cases}$$

Поскольку Φ_p — гомоморфизм групп, его применение к соотношению $H = G^x$ приводит к равенству $\Phi_p(H) = \Phi_p(G)^x$ в группе C_{p^e}. Проблема дискретного логарифмирования в группе C_{p^e} определена только по модулю p^e. Решая ее, мы найдем x по модулю p^e. Сделав это для всех простых делителей N, мы восстановим x по модулю N, используя китайскую теорему об остатках. Подводя итог, предположим, что мы владеем оракулом $\mathcal{O}(G, H, p, e)$, который по элементам G, $H \in C_{p^e}$ вычисляет дискретный логарифм элемента H по основанию G. Тогда мы отыщем нужный нам x в результате следующего алгоритма:

```
S={ };
для всех простых p, делящих N
  { вычислить наибольшее число e, для которого T=p^e делит N;
    G1=G^{M/T};
    H1=H^{N/T};
    z=O(G1,H1,p,e);
    S = S + { (z,T) }; }
x=КТО(S);
```

Осталось только показать, как отыскать дискретный логарифм в группе C_{p^e}. Сделаем это, сводя задачу к случаю $e = 1$. Пусть

$G, H \in C_{p^e}$, причем для некоторого натурального числа x имеет место равенство $H = G^x$. Очевидно, число x определено по модулю p^e, и мы можем представить его в виде

$$x = x_0 + x_1 p + \cdots + x_{e-1} p^{e-1}.$$

Будем искать коэффициенты $x_0, x_1, \ldots$ по очереди, используя индуктивную процедуру. Допустим, мы знаем x' — значение x по модулю p^t, т. е.

$$x' = x_0 + \cdots + x_{t-1} p^{t-1},$$

и хотим определить x_t (или найти x по модулю x^{t+1}). Запишем x как

$$x = x' + p^t x'',$$

где $x'' = x_t + p x_{t+1} + \cdots + x_{e-1} p^{e-t-1}$. Тогда

$$H = G^x = G^{x'} \left(G^{p^t}\right)^{x''}.$$

Следовательно, положив

$$H' = HG^{-x'} \quad \text{и} \quad G' = G^{p^t},$$

получим уравнение на x'':

$$H' = G'^{x''}.$$

Теперь G' — элемент порядка p^{e-t}. Поэтому, чтобы получить элемент порядка p, и, следовательно, свести задачу к группе C_p, нам нужно возвести предыдущее уравнение в степень $s = p^{e-t-1}$. Положив

$$H'' = H'^s \quad \text{и} \quad G'' = G'^s,$$

мы получим задачу о вычислении логарифмов в группе C_p:

$$\begin{aligned} H'' = G''^{x''} &= G''^{x_t + p x_{t+1} + \cdots + x_{e-1} p^{e-t-1}} = \\ &= G''^{x_t} \cdot G''^{p(x_{t+1} + \cdots x_{e-1} p^{e-t-2})} = \\ &= G''^{x_t}. \end{aligned}$$

Предполагая, что мы можем решить задачу о логарифмах в группе простого порядка, найдем x_t, а значит и требуемый x.

Способ решения задачи дискретного логарифмирования в группах C_p мы осветим в следующих двух параграфах, а сейчас проиллюстрируем изложенное, предполагая, что случай простого порядка группы нам известен.

В качестве примера рассмотрим мультипликативную группу конечного поля $\mathbb{F}_{397}$. Ее порядок равен $396 = 2^2 \cdot 3^2 \cdot 11$. Одна из обра-

зующих группы $\mathbb{F}^*_{397}$ — это $G = 5$. Решим показательное уравнение

$$H = 208 = 5^x \pmod{397}.$$

Разделим задачу между тремя подгруппами, порядок которых — степень простого числа:

$$334 = H^{396/4} = G^{396x_4/4} = 334^{x_4} \pmod{397},$$
$$286 = H^{396/9} = G^{396x_9/9} = 79^{x_9} \pmod{397},$$
$$273 = H^{396/11} = G^{396x_{11}/11} = 290^{x_{11}} \pmod{397}.$$

Обратите внимание на то, что x_a (где $a = 4, 9, 11$) — решение исходной задачи по модулю a. Поэтому, найдя все три x_a, мы восстановим x по модулю 396, что нам и нужно.

9.2.1. Определяем x_4

Легко увидеть, что $x_4 = 1$, но стоит потрудиться и понять, как это обнаруживает алгоритм. Представим x_4 в виде

$$x_4 = x_{4,0} + 2 \cdot x_{4,1},$$

где $x_{4,0}, x_{4,1} \in \{0, 1\}$. Напомним, что нам нужно решить уравнение

$$H' = 334 = 334^{x_4} = G'^{x_4}.$$

Вводим элементы $H'' = H'^2$ и $G'' = G'^2$ и решаем задачу дискретного логарифмирования

$$H'' = G''^{x_{4,0}}$$

в циклической группе порядка 2. Применяя метод, о котором будет рассказано позже (назовем его временно оракул), найдем $x_{4,0} = 1$. Теперь, идя по индукции, приходим к уравнению

$$\frac{H'}{G'} = G''^{x_{4,1}} \pmod{397}.$$

Подставляя численные значения, имеем

$$1 = 396^{x_{4,1}},$$

что является очередной задачей дискретного логарифмирования в группе простого порядка. Вновь прибегая к помощи оракула, находим $x_{4,1} = 0$. Значит, $x_4 = x_{4,0} + 2 \cdot x_{4,1} = 1 + 2 \cdot 0 = 1$.

9.2.2. Ищем x_9

Представляем его в виде многочлена:

$$x_9 = x_{9,0} + 3 \cdot x_{9,1}$$

с коэффициентами $x_{9,0}$, $x_{9,1} \in \{0, 1, 2\}$. Как Вы помните, мы решаем уравнение

$$H' = 286 = 79^{x_9} = G'^{x_9}.$$

Положим $H'' = H'^3$, $G'' = G'^3$ и получим задачу логарифмирования в группе порядка 3:

$$H'' = 34 = G''^{x_{9,0}} = 362^{x_{9,0}}.$$

Применяя оракул, определим $x_{9,0} = 2$. Это нам дает соотношение

$$\frac{H'}{G'^2} = G''^{x_{9,1}} \pmod{397}.$$

Следовательно, $1 = 362^{x_{9,1}}$ — очередная задача логарифмирования в группе порядка 3. Находим $x_{9,1} = 0$, откуда

$$x_9 = x_{9,0} + 3 \cdot x_{9,1} = 2 + 3 \cdot 0 = 2.$$

9.2.3. Определяем x_{11}

Здесь задача

$$273 = 290^{x_{11}} \pmod{397}$$

сформулирована в циклической группе простого порядка. Поэтому для ее решения можно воспользоваться оракулом, который нам найдет $x_{11} = 6$.

Финал заключается в восстановлении решения исходной задачи

$$208 = 5^x \pmod{397}$$

по установленным тождествам:

$$\begin{cases} x = 1 \pmod 4, \\ x = 2 \pmod 9, \\ x = 6 \pmod{11}. \end{cases}$$

Применяя китайскую теорему об остатках к этой системе, находим ответ исходной задачи: $x = 281$.

9.3. Шаги младенца/шаги гиганта

В предыдущем параграфе мы предполагали, что существует оракул, решающий задачу дискретного логарифмирования в циклических группах простого порядка. Теперь мы опишем общий метод решения этой проблемы, который называется «шаги младенца/шаги

гиганта» и применяется, вообще говоря, в любой конечной абелевой группе.

Поскольку предварительные этапы в методе Полига–Хеллмана довольно просты, основная трудность решения общей проблемы дискретных логарифмов падает на группу простого порядка. Таким образом, в общих группах сложность метода шаги младенца/шаги гиганта будет доминировать над сложностью любого алгоритма. Действительно, можно показать, что метод, о котором сейчас пойдет речь, — лучший из решающих задачу дискретного логарифмирования в произвольной группе. Конечно, в любой конкретной группе может найтись специальный алгоритм, который работает быстрее, но для общей группы, метод «шагов»— вероятно, самое лучшее, что можно придумать.

Зафиксируем обозначения. Пусть $A = \langle G \rangle$ — циклическая группа простого порядка p. В ней выделен элемент H и спрашивается: как найти число x по модулю p, удовлетворяющее уравнению $H = G^x$? Будем предполагать, что фиксирован некий способ записи элементов группы в компьютер, так что без труда можно хранить ее элементы, сортировать их и производить среди них поиск требуемого элемента.

Идея, легшая в основу метода «шагов», — стандартная стратегия «разделяй и властвуй», используемая во многих областях информатики. Решение определяется в виде

$$x = x_0 + x_1 \lceil \sqrt{p} \rceil,$$

где $\lceil y \rceil$ — целая часть числа y с недостатком, т. е. наибольшее целое n, удовлетворяющее неравенству $n \leqslant y$. Неравенство $x \leqslant p$ влечет: $0 \leqslant x_0, x_1 < \lceil \sqrt{p} \rceil$.

Сначала делаем шаги младенца:

$$G_i = G^i \text{ для } 0 \leqslant i < \lceil \sqrt{p} \rceil.$$

Пары (G_i, i) хранятся в таблице таким образом, что любую из них можно легко найти по первому элементу. Это делается либо с помощью упорядочивания таблицы по первому элементу, либо, что более эффективно, с помощью хэш-таблиц. На вычисления результатов алгоритму требуется

$$O(\lceil \sqrt{p} \rceil)$$

операций и примерно такой же объем компьютерной памяти для их хранения. Далее вычисляют шаги гиганта:

$$H_j = HG^{-j\lceil \sqrt{p} \rceil} \text{ для } 0 \leqslant j < \lceil \sqrt{p} \rceil.$$

После этого пытаемся найти такую пару из таблицы, сформированной на предыдущем этапе, чтобы $G_i = H_j$. Если такая пара найдется, то

$$x_0 = i \quad \text{и} \quad x_1 = j,$$

поскольку при $G_i = H_j$ имеет место равенство

$$G^i = HG^{-j\lceil\sqrt{p}\rceil}, \text{ т. е. } G^{i+j\lceil\sqrt{p}\rceil} = H.$$

Заметим, что время, требуемое на вычисление шагов гиганта не больше

$$O(\lceil\sqrt{p}\rceil).$$

Следовательно, сложность метода шаги младенца/шаги гиганта равна

$$O(\sqrt{p}).$$

Таким образом, если мы хотим, чтобы решение задачи дискретного логарифмирования в группе A с учетом алгоритма Полига–Хеллмана занимало около 2^{80} операций, нам нужно подобрать такую группу A, у которой существует подгруппа простого прядка p с $p > 2^{160}$.

Разберем пример, взяв в качестве группы подгруппу порядка 101 в мультипликативной группе поля $\mathbb{F}_{607}$, порожденную элементом $G = 64$. Допустим, нам нужно решить уравнение

$$H = 182 = 64^x \pmod{607}.$$

Сначала делаем шаги младенца:

$$G_i = 64^i \pmod{607} \text{ для } 0 \leqslant i < \lceil\sqrt{101}\rceil = 11.$$

Результаты вычислений занесем в таблицу:

i	$64^i \pmod{607}$	i	$64^i \pmod{607}$
0	1	6	330
1	64	7	482
2	454	8	498
3	527	9	308
4	343	10	288
5	100		

Затем вычисляем шаги гиганта:

$$H_j = 182 \cdot 64^{-11j} \pmod{607} \text{ для } 0 \leqslant j < 11$$

и смотрим, когда шаг гиганта совпадает с шагом младенца:

j	$182 \cdot 64^{-11j} \pmod{607}$	j	$182 \cdot 64^{-11j} \pmod{607}$
0	182	6	60
1	143	7	394
2	69	8	483
3	271	9	76
4	343	10	580
5	573		

Из сравнения таблиц видно, что совпадение имеет место при $i = 4$ и $j = 4$. Это означает, что

$$x = 4 + 11 \cdot 4 = 48,$$

что можно проверить, вычислив

$$64^{48} \pmod{607} = 182.$$

9.4. Методы Полларда

Недостаток метода шаги младенца/шаги гиганта заключается в том, что несмотря на сравнительно малое время работы ($O(\sqrt{p})$), он требует столько же места в памяти компьютера. Такие требования, предъявляемые алгоритмами к памяти, даже более обременительны на практике, чем несколько большее время работы. В связи с этим возникает вопрос: можно ли существенно снизить требования к памяти, не увеличивая существенно сложность алгоритма? Ответ положителен, но, к сожалению, мы при этом можем оценить лишь ожидаемое время работы, но не фактическое. Существует несколько алгоритмов, которые снижают требования к памяти. Все они своим существованием обязаны идеям Полларда.

9.4.1. ρ-метод Полларда

Пусть $f : S \longrightarrow S$ — случайная функция на множестве S и $\#S = n$. Выбираем произвольный элемент $x_0 \in S$ и последовательно вычисляем

$$x_{i+1} = f(x_i) \text{ при } i \geqslant 0.$$

Значения $x_0, x_1, x_2, \ldots$ будем рассматривать как *детерминированное случайное блуждание*. Последнее высказывание означает, что

каждый шаг $x_{i+1} = f(x_i)$ пути — детерминированная функция текущей позиции x_i. Но мы будем предполагать, что последовательность $x_0, x_1, x_2, \ldots$ *ведет себя* как случайная. Такие последовательности еще называют *псевдо-случайными*.

Поскольку S — конечное множество, мы должны получить равенство $x_i = x_j$ для некоторой пары индексов, и тогда

$$x_{i+1} = f(x_i) = f(x_j) = x_{j+1}.$$

Иначе говоря, последовательность $x_0, x_1, x_2, \ldots$ будет вести себя циклическим образом. Если мы попытаемся изобразить это блуждание на множестве S, то получится что-то вроде греческой буквы «ро», т. е. ρ. Другими словами, в нем есть циклическая часть и некоторый начальный «хвост», в цикл не входящий. Можно показать, что ожидаемая длина хвоста случайного отображения f (т. е. число элементов в хвосте), равна $\sqrt{\pi n/8}$, а ожидаемая длина цикла — $\sqrt{3\pi n/8}$. Цель большинства методов, восходящих к Полларду, — найти начало цикла в случайном блуждании т. е. такие элементы x_i и x_j последовательности, которые равны друг другу при несовпадающих индексах. Благодаря парадоксу дней рождений мы получаем, что ожидаемое повторение произойдет через $\sqrt{\pi n/2}$ итераций функции f. Наивно говоря, до ожидаемого повтора пройдет $O(\sqrt{n})$ времени, а чтобы засечь повторение, нам потребуется $O(\sqrt{n})$ памяти. Но это и есть та самая проблема, с которой мы сталкиваемся в методе шаги младенца/шаги гиганта.

Чтобы найти повтор и воспользоваться ρ-формой случайного блуждания, применяется алгоритм Флойда поиска циклов. Он заключается в следующем: по паре (x_1, x_2) мы вычисляем (x_2, x_4), затем (x_3, x_6) и т. д., т. е. вслед за парой (x_i, x_{2i}) вычисляется пара

$$(x_{i+1}, x_{2i+2}) = (f(x_i), f(f(x_{2i}))).$$

Процесс заканчивается при равенстве: $x_m = x_{2m}$. Если длина хвоста нашей последовательности равна λ, а длина ее цикла — μ, то такое совпадение можно ожидать при[1]

$$m = \mu(1 + \lfloor \lambda/\mu \rfloor).$$

Принимая во внимание неравенство $\lambda < m \leqslant \lambda + \mu$, мы получаем, что $m = O(\sqrt{n})$. Причем это будет точной оценкой сложности, если f ведет себя как средняя случайная функция. Следовательно, мы можем обнаружить повторение, практически ничего не храня в памяти компьютера.

[1] Символ $\lfloor y \rfloor$ означает целую часть y с избытком, т. е. наименьшее целое n, для которого $n \geqslant y$. — *Прим. перев.*

Все это замечательно, но мы не показали, как это блуждание связано с дискретным логарифмированием. Пусть A — группа порядка n и мы решаем в ней задачу дискретного логарифмирования:

$$H = G^x.$$

Разобьем группу на три части S_1, S_2 и S_3, причем $1 \notin S_2$, и определим случайное блуждание по группе A следующим образом:

$$x_{i+1} = f(x_i) = \begin{cases} H \cdot x_i, & x_i \in S_1, \\ x_i^2, & x_i \in S_2, \\ G \cdot x_i, & x_i \in S_3. \end{cases}$$

Фактически нам нужно хранить данные в виде вектора: (x_i, a_i, b_i), где

$$a_{i+1} = \begin{cases} a_i, & x_i \in S_1, \\ 2a_i \pmod n, & x_i \in S_2, \\ a_i + 1 \pmod n, & x_i \in S_3; \end{cases}$$

$$b_{i+1} = \begin{cases} b_i + 1 \pmod n, & x_i \in S_1, \\ 2b_i \pmod n, & x_i \in S_2, \\ bi, & x_i \in S_3. \end{cases}$$

Если мы начинаем с тройки $(x_0, a_0, b_0) = (1, 0, 0)$, то для всех i

$$\log_G(x_i) = a_i + b_i \log_G(H) = a_i + b_i x,$$

где $x = \log_G H$. Применяя алгоритм Флойда поиска циклов, мы получим повторение последовательности, т. е. такое число m, что $x_m = x_{2m}$. Это позволяет нам вывести следующее равенство:

$$a_m + b_m x = a_m + b_m \log_G(H) = \log_G(x_m) = \log_G(x_{2m}) = \\ = a_{2m} + b_{2m} \log_G(H) = a_{2m} + b_{2m} x.$$

После преобразований находим

$$(b_m - b_{2m})x = a_{2m} - a_m,$$

так что если $b_m \neq b_{2m}$, получаем

$$x = \frac{a_{2m} - a_m}{b_m - b_{2m}} \pmod n.$$

При больших n вероятность равенства $b_m = b_{2m}$ настолько мала, что ее можно игнорировать.

Итак, если случайное блуждание получается из случайной функции $f : A \longrightarrow A$, то вышеописанный алгоритм решит задачу дискретного логарифмирования за $O(\sqrt{n})$ операций.

В качестве примера рассмотрим подгруппу A поля $\mathbb{F}_{607}$ порядка $n = 101$, порожденную элементом 64. Поставим задачу о вычислении логарифма:

$$H = 122 = 64^x.$$

Определяем подмножества группы:

$$S_1 = \{x \in \mathbb{F}_{607} |\ x \leqslant 201\},$$
$$S_2 = \{x \in \mathbb{F}_{607} |\ 202 \leqslant x \leqslant 403\},$$
$$S_3 = \{x \in \mathbb{F}_{607} |\ 404 \leqslant x \leqslant 606\}.$$

В процессе работы алгоритма Флойда будут получаться следующие данные:

i	x_i	a_i	b_i	x_{2i}	a_{2i}	b_{2i}
0	1	0	0	1	0	0
1	122	0	1	316	0	2
2	316	0	2	172	0	8
3	308	0	4	137	0	18
4	172	0	8	7	0	38
5	346	0	9	309	0	78
6	137	0	18	352	0	56
7	325	0	19	167	0	12
8	7	0	38	498	0	26
9	247	0	39	172	2	52
10	309	0	78	137	4	5
11	182	0	55	7	8	12
12	352	0	56	309	16	26
13	76	0	11	352	32	53
14	167	0	12	167	64	6

Повторение произошло при $m = 14$. Значит,

$$G^0 H^{12} = G^{64} H^6,$$

откуда вытекает уравнение

$$12x = 64 + 6x \mod 101.$$

Другими словами,

$$x = \frac{64}{12-6} \pmod{101} = 78.$$

9.4.2. λ-метод Полларда

Этот алгоритм похож на ρ-метод тем, что в нем используется детерминированное случайное блуждание и задействовано мало памяти для хранения промежуточных шагов. Однако λ-метод предназначен, в первую очередь, для ситуации, когда известен отрезок, где может лежать значение логарифма:

$$x \in [a, \ldots, b].$$

В ρ-методе используется одно случайное блуждание, своей формой напоминающее греческую букву «ρ». А в λ-методе используются две случайные последовательности элементов группы, которые выглядят как буква «λ», откуда и происходит название алгоритма.

Расскажем о λ-методе. Пусть $w = b - a$ обозначает длину отрезка, в котором лежит значение искомого дискретного логарифма. Задают множество

$$S = \{s_0, \ldots, s_{k-1}\}$$

целых чисел, расположенных по неубыванию, средний элемент m которого приблизительно равен $N = \sqrt{w}$. Как правило, выбирают

$$s_i = 2^i \text{ для } 0 \leqslant i < k.$$

В этой ситуации средний член равен $\frac{2^k}{k}$. Поэтому берут $k \approx \frac{1}{2}\log_2(w)$.

Группу A, в которой ставится задача, разбивают на k подмножеств S_i $(i = 0, \ldots, k-1)$ и фиксируют функцию, определяющую псевдо-случайную последовательность:

$$x_{i+1} = x_i \cdot G^{s_j} \text{ если } x_i \in S_j.$$

После этого вычисляются элементы детерминированного случайного блуждания, начиная с $G_0 = G^b$, по правилу

$$G_i = G_{i-1} \cdot G^{s_j}$$

для $i = 1, \ldots, N$. Обозначают $c_0 = b$ и $c_{i+1} = c_i + s_j \pmod{q}$, где q — порядок группы A. В результате последовательных вычислений будет найден элемент G_N (где $N = \sqrt{w}$), который и записывается в память компьютера. При этом нам известен элемент $c_N = \log_G(G_N)$, который тоже следует запомнить.

Теперь вычисляют второе детерминированное случайное блуждание, начинающееся с неизвестной точки интервала x, а именно,

пусть $H_0 = H = G^x$, тогда $H_{i+1} = H_i \cdot G^{s_j}$. Кроме того, обозначают $d_0 = 0$ и $d_{i+1} = d_i + s_j \pmod q$. Заметим, что имеет место соотношение:

$$\log_G(H_i) = x + d_i.$$

Если путь H_i пересечется с последовательностью G_i, то H_i дальше пойдет по пути G_i и можно будет найти такое значение M, при котором H_M окажется равным сохраненному нами элементу G_N. В этой ситуации

$$c_N = \log_G(G_N) = \log_G(H_M) = x + d_M.$$

Следовательно, искомое решение задачи о дискретных логарифмах будет иметь вид:

$$x = c_N - d_M \pmod q.$$

Если пересечения путей не произойдет, нам придется увеличить N и продолжить оба блуждания до их пересечения.

Ожидаемое время работы алгоритма имеет порядок $\sqrt{w}$, и можно показать, что количество требуемой памяти постоянно. λ-метод возможно использовать и тогда, когда известно лишь, что искомый логарифм лежит в полном отрезке $[0, ..., q-1]$. Но тогда асимптотика сложности алгоритма совпадает со сложностью ρ-метода.

В качестве примера снова возьмем группу $G \subset \mathbb{F}^*_{607}$ порядка 101 с образующей 64, но задачу выберем другую, а именно, попытаемся найти x, при котором

$$H = 524 = 64^x.$$

Будем считать известным, что искомое решение лежит на отрезке $[60, \ldots, 80]$. Возьмем в качестве коэффициентов числа $s_i = 2^i$ при $i = 0, 1, 2, 3$. Подмножества $S_0, \ldots, S_3$ группы определим так:

$$S_i = \{a \in A |\ a \pmod 4 = i\}.$$

Сначала вычисляем детерминированную случайную последовательность G_i и дискретные логарифмы $c_i = \log_G(G_i)$ для $i = 0, \ldots, N = 4$:

i	G_i	c_i
0	151	80
1	537	88
2	391	90
3	478	98
4	64	1

Потом вычисляем вторую псевдо-случайную последовательность.

i	H_i	$d_i = \log_G(H_i) - x$
0	524	0
1	151	1
2	537	9
3	391	11
4	478	19
5	64	23

Получили встречу: $H_5 = G_4$. Поэтому

$$x = 1 - 23 \pmod{101} = 79.$$

Обратите внимание на то, что при исследовании этих таблиц можно заметить и более ранние пересечения наших блужданий. Однако мы их не могли использовать, поскольку не запоминали промежуточных значений последовательности G_i, т. е. элементов G_0, G_1, G_2 и G_3. Единственное, о чем мы помнили — это G_4.

9.4.3. Параллельный ρ-метод

При применении этих алгоритмов на практике для распределения вычислений между большим числом компьютеров через Интернет используют параллельную версию метода. Суть распараллеливания состоит в следующем. Допустим, перед нами стоит задача о решении уравнения

$$H = G^x$$

в группе A простого порядка q. Сначала зададим легко вычислимую функцию

$$f : A \longrightarrow \{1, \ldots, k\},$$

где k обычно берут около 20. Затем определим множество коэффициентов m_i по случайным числам a_i, $b_i \in [0, \ldots, q-1]$ с помощью формулы

$$m_i = G^{a_i} H^{b_i}.$$

Чтобы начать случайное блуждание, мы берем произвольную пару чисел $s_0, t_0 \in [0, \ldots, q-1]$ и вычисляем

$$G_0 = G^{s_0} H^{t_0}.$$

Следующие члены (G_i, s_i, t_i) псевдослучайной последовательности

получаются рекуррентно:

$$G_{i+1} = G_i \cdot m_{f(G_i)}, \quad s_{i+1} = s_i + a_{f(G_i)} \pmod q,$$
$$t_{i+1} = t_i + b_{f(G_i)} \pmod q.$$

Следовательно, для каждого G_i мы записываем значения s_i и t_i, удовлетворяющие соотношению

$$G_i = G^{s_i} H^{t_i}.$$

Допустим, что у нас есть m процессоров, каждый из которых вычисляет блуждающие последовательности, начинающиеся с разных элементов, по одному и тому же алгоритму. Когда два процессора или даже один и тот же найдет элемент последовательности, который уже встречался, мы получим уравнение:

$$G^{s_i} H^{t_i} = G^{s'_j} H^{t'_j},$$

из которого мы можем извлечь дискретный логарифм x. Ожидают, что после $O(\sqrt{\pi q/2}/m)$ итераций этих параллельных блужданий найдется повторение и удастся вычислить дискретный логарифм.

Однако при таком способе решения проблемы подразумевается, что каждый из участвующих в процессе компьютеров должен сообщать очередные члены последовательности на центральный сервер, который будет их хранить и сравнивать. Это весьма неэффективный метод, поскольку придется запоминать слишком много, а именно $O(\sqrt{\pi q/2})$ данных. Можно уменьшить требуемое количество памяти. Объясним, как это сделать.

Возьмем функцию $d : A \longrightarrow \{0,1\}$, которая принимает единичные значения примерно один раз из 2^t. Ее часто задают так, что $d(a) = 1$, если определенное подмножество бит, представляющих элемент a, состоит из нулей. Те элементы $a \in A$, для которых $d(a) = 1$, называют отмеченными. Теперь на центральный сервер поступают сведения лишь об отмеченных элементах группы. Это означает, что случайные блуждания скорее всего продлятся еще на 2^t шагов, прежде чем будет обнаружено пересечение их путей. Следовательно, время вычислений становится теперь равным

$$O\left(\frac{\sqrt{(\pi q)/2}}{m} + 2^t\right),$$

и требуется

$$O\left(\frac{\sqrt{(\pi q)/2}}{2^t}\right)$$

памяти.

Такой прием позволяет сократить расходы памяти до любой требуемой величины за счет сравнительно небольшого увеличения времени работы алгоритма. Мы не будем приводить примера, поскольку это усовершенствование полезно лишь для очень больших групп, порядок которых превышает 2^{20}. Но мы советуем читателю продумать свой собственный пример.

9.5. Суб-экспоненциальные методы в числовых полях

Существует тесная связь между суб-экспоненциальными методами разложения на множители и суб-экспоненциальными методами дискретного логарифмирования в конечных полях. Мы рассмотрим случай поля $\mathbb{F}_p$ с простым числом p элементов, отметив, что аналогичные методы применяются и к конечным полям характеристики 2. Суб-экспоненциальные алгоритмы в конечных полях часто называют *исчислением показателей* по причинам, которые станут понятны после знакомства с методами.

Как и прежде, мы считаем, что нам поставлена задача о решении уравнения

$$H = G^x$$

с $H, G \in \mathbb{F}_p^*$. Выберем факторбазу $\mathcal{F}$ элементов, состоящую из малых простых чисел, и применим одну из просеивающих стратегий, использовавшихся при факторизации. Получим большое число соотношений вида

$$\prod_{p_i \in \mathcal{F}} p_i^{e_i} = 1 \pmod{p}.$$

Эти соотношения переписываются в уравнения дискретного логарифмирования:

$$\sum_{p_i \in \mathcal{F}} e_i \log_G(p_i) = 0 \pmod{(p-1)}.$$

Как только достаточно уравнений, подобных этому, будет найдено, мы сможем решить задачу дискретного логарифмирования для каждого элемента из факторбазы, т. е. мы сможем найти число

$$x_i = \log_H p_i,$$

которое иногда называют *показателем* элемента p_i по отношению к G. Эти вычисления осуществляются с помощью линейной алгебры по модулю $p-1$, более сложной, чем линейная алгебра по модулю 2,

привлекавшаяся в алгоритмах факторизации. Однако приемы, снижавшие в алгоритмах факторизации требования к памяти компьютера до приемлемого уровня, работают и в этом случае. Линейная алгебра применяется один раз для каждого основания G, а ее результаты работают потом для многих элементов H.

Когда возникает необходимость в вычислении конкретного логарифма $H = G^x$, с помощью техники решета или метода пробного деления выписывают разложение

$$H = \prod_{p_i \in \mathcal{F}} p_i^{h_i} \pmod{p},$$

т. е. мы можем вычислить

$$T = H \prod_{p_i \in \mathcal{F}} p_i^{f_i} \pmod{p},$$

и получить разложение вида

$$T = \prod_{p_i \in \mathcal{F}} p_i^{g_i} \pmod{p}.$$

Сделав это, получим

$$H = \prod_{p_i \in \mathcal{F}} p_i^{g_i - f_i} \pmod{p}.$$

Искомый x, наконец, можно найти, сделав следующие вычисления:

$$\begin{aligned} x = \log_G(H) &= \log_G \left(\prod_{p_i \in \mathcal{F}} p_i^{g_i} \right) = \\ &= \sum_{p_i \in \mathcal{F}} h_i \log_G(p_i) \pmod{p-1} = \\ &= \sum_{p_i \in \mathcal{F}} h_i \cdot x_i \pmod{p-1}. \end{aligned}$$

Это означает, что как только один логарифм будет найден, остальные вычислить будет легче, поскольку мы уже будем знать все показатели x_i.

Наилучший способ поиска соотношений на факторбазе — решето в числовом поле. Его сложность оценивается функцией

$$O(L_p(1/3, c))$$

с некоторой константой c. Она примерно совпадает со сложностью алгоритма факторизации больших чисел, хотя при дискретном логарифмировании используется матрица по модулю $p - 1$, а не по модулю 2, как это было при факторизации.

Вывод из анализа суб-экспоненциальных методов говорит о том, что размер p конечного поля, в котором ставится задача дискретного логарифмирования, должен быть примерно таким же, как и модуль системы RSA, т. е. порядка $p > 2^{1024}$.

Даже если p будет очень большим, нам все еще нужно принять меры против общей атаки. Поэтому простые делители q числа $p-1$ должны быть больше 2^{160}, поскольку в случае конечных полей, где проводится логарифмирование, мы обычно работаем с подгруппами группы $\mathbb{F}_p^*$ порядка q.

9.6. Специальные методы для эллиптической кривой

В случае эллиптических кривых неизвестны суб-экспоненциальные методы, решающие задачу дискретного логарифмирования, за исключением некоторых особых случаев. Это означает, что единственный метод дискретного логарифмирования в общих группах эллиптических кривых — это параллельная версия ρ-метода Полларда.

Пусть эллиптическая кривая E определена над конечным полем $\mathbb{F}_q$. Положим

$$\#E(\mathbb{F}_q) = h \cdot r,$$

где r — простое число. По теореме Хассе 2.3 значение $\#E(\mathbb{F}_q)$ близко к q. Таким образом, обычно выбирают кривую E с параметром r, мало отличающимся от q, т. е. берут кривую, для которой $h = 1$, 2 или 4.

Наилучший из известных алгоритмов дискретного логарифмирования на эллиптической кривой — параллельный ρ-метод Полларда, чья сложность — $O(\sqrt{r})$, близкая к $O(\sqrt{q})$. Поэтому для обеспечения той же криптостойкости, которая есть у 80-битового блочного шифра, нам нужно брать $q \approx 2^{160}$, что несколько меньше, чем размер поля, рекомендованный к использованию в криптосистемах, основывающихся на дискретном логарифмировании в конечных полях. Это отражается на пропускной способности и времени вычислений в системах, базирующихся на эллиптической кривой.

Однако существует несколько специальных случаев, которых следует избегать. Сейчас мы о них расскажем, но не станем подробно объяснять причины их нежелательности, поскольку такое объяснение использует слишком сложную математику. Как обычно, будем предполагать, что q — либо большое простое число, либо степень 2.

- Для любого q нам нужно выбрать кривую, для которой нет маленьких чисел t, при которых $q^t - 1$ делится на r, где r —

большой простой делитель числа $\#E(\mathbb{F}_q)$. Такое требование исключает из рассмотрения суперсингулярные кривые и несколько других. В этом случае есть простое преобразование, сводящее задачу дискретного логарифмирования над эллиптической кривой к аналогичной задаче в конечном поле $\mathbb{F}_{q^t}$. Следовательно, в этой ситуации можно получить суб-экспоненциальный метод решения задачи дискретного логарифмирования над эллиптической кривой.

- Если $q = p$ — большое простое число, то нам следует избегать аномальных кривых, для которых $\#E(\mathbb{F}_p) = p$. В этом случае существует алгоритм решения задачи, требующий $O(\ln p)$ операций над точками эллиптической кривой.
- Если $q = 2^n$, то обычно выбирают простое n, чтобы избежать возможных атак, основанных на понятии «спуск Вейля».

Нужно быть предельно внимательным, имея дело с этими тремя исключениями, примерно так же, как и при генерировании больших целых чисел для алгоритма *RSA*. Учитывая $(P-1)$-метод разложения на множители, в системе *RSA*, как правило, выбирают модули $N = pq$ так, что p представляется в виде $2p_1 + 1$ с каким-то другим простым p_1. Другая особенность криптосистемы *RSA* состоит в том, что мы применяем модули с двумя простыми множителями, а не с тремя или четырьмя. Мотивируется это тем, что произведение двух простых чисел разложить на множители сложнее, чем произведение бóльшего числа множителей.

Краткое содержание главы

- Ввиду существования алгоритма Полига–Хеллмана трудную задачу о дискретном логарифмировании следует ставить в группах, чей порядок имеет большой простой делитель.
- Существование общих алгоритмов, таких, например, как шаги младенца/шаги гиганта, свидетельствует о том, что для обеспечения той же криптостойкости, как в 80-битовом блочном шифре, нужно выбирать группы с простым делителем порядка группы, насчитывающим не менее 160 бит.
- Шаги младенца/шаги гиганта — общий алгоритм, чье время работы ограничено сверху числом $\sqrt{q}$, где q — большой простой делитель порядка группы A, в которой ставится задача дискретного логарифмирования. Однако этому методу необходимо $O(\sqrt{q})$ памяти.

- Существует несколько техник, идеи которых восходят к Полларду, основывающихся на детерминированном случайном блуждании по группе. Это общие методы, не требующие много компьютерной памяти, и решающие задачу дискретного логарифмирования за среднее время $O(\sqrt{q})$.
- Для конечных полей существует алгоритмы вычисления показателей за суб-экспоненциальное время. Это означает, что для постановки трудноразрешимой задачи дискретного логарифмирования необходимо выбирать большие конечные поля $\mathbb{F}_{p^t}$ с $p^t \geqslant 2^{1024}$.
- Для эллиптических кривых не найдено суб-экспоненциальных алгоритмов, за исключением специальных кривых. Поэтому единственный из известных методов, решающий задачу дискретного логарифмирования на общей эллиптической кривой, — это параллельный ρ-метод Полларда.

Дополнительная литература

Написано множество хороших обзоров проблемы дискретных логарифмов. Я бы рекомендовал Мак-Карлея и Одлыжко. Однако эти статьи лишь мимоходом касаются проблемы логарифмирования на эллиптической кривой. Для пополнения своих знаний по последней проблеме Вы можете консультироваться с книгой Коблица.

N. Koblitz, A. Menezes and S. Vanstone. *The state of elliptic curve cryptography*. Designs Codes and Cryptography, **19,** 173-193, 2000.

K. McCurley. *The discrete logarithm problem.* In *Cryptology and Computational Number Theory*, Proc. Symposia in Applied Maths, Volume 42, 1990.

A. Odlyzko. *Discrete logarithms: The past and the future*. Designs Codes and Cryptography, 19, 129-145, 2000.

Контрольные вопросы

9.1.1. Какие выводы в отношении криптографии можно вывести из алгоритма Полига–Хеллмана?

9.1.2. Как работает алгоритм шаги младенца/шаги гиганта?

9.1.3. Какие проблемы возникают при реализации метода шаги младенца/шаги гиганта, и как они преодолеваются с помощью ρ-метода Полларда?

9.1.4. Какие уроки в отношении криптосистем можно извлечь из существования ρ-метода Полларда?

9.1.5. Какие выводы следуют из существования алгоритмов вычисления показателей в конечных полях с точки зрения криптографии?

9.1.6. Обсудите утверждение о том, что эллиптические кривые предлагают более высокую криптостойкость на один бит, нежели конечные поля.

Лабораторные работы

9.2.1. Реализуйте ρ-метод Полларда и поэкспериментируйте с разными определениями детерминированных случайных блужданий, которые встречались в тексте. Какой из них наиболее эффективен? (Эффективность здесь означает более быстрое в среднем решение задачи о дискретных логарифмах.)

9.2.2. Разработайте программу для параллельного метода Полларда решения задачи дискретного логарифмирования в конечных полях. Насколько сложную задачу дискретного логарифмирования Вы сможете решить с помощью этой программы в течение 24 часов?

Упражнения

9.3.1. В тексте главы были определены три случайных блуждания, по одной на каждый из методов: ρ, λ и параллельный методы Полларда. Можно ли использовать эти псевдо-случайные последовательности для разных алгоритмов? Какие преимущества (недостатки) возникают при использовании в указанных методах случайных блужданий, разработанных для других алгоритмов?

9.3.2. Используя лишь карманный калькулятор, найдите число x, удовлетворяющее уравнению $3^x = 5 \pmod{p}$, где $p = 2 \cdot 3 \cdot 101 \cdot 103 \cdot 107^2 + 1$.

ГЛАВА 10

РАСПРЕДЕЛЕНИЕ КЛЮЧЕЙ, СХЕМЫ ПОДПИСЕЙ И ХЭШ-ФУНКЦИИ

Цели главы

- Ввести протокол распределения ключей Диффи-Хеллмана.
- Объяснить все необходимые для цифровых подписей понятия.
- Рассказать о двух наиболее широко употребляемых алгоритмах подписей, а именно *RSA* и *DSA*.
- Ввести и объяснить все необходимое для криптографических хэш-функций.
- Рассказать о некоторых других алгоритмах подписей и технике распределения ключей, обладающих интересными свойствами.

10.1. Распределение ключей Диффи–Хеллмана

Напомним, что основное препятствие к быстрому шифрованию большого количества данных с использованием блочного или поточного шифра, — это сложная проблема распределения ключей. Мы уже познакомились с некоторыми методами ее решения: с помощью протоколов, базирующихся на симметричном шифровании или с использованием алгоритма с открытым ключом для генерирования и передачи сеансового ключа заинтересованным партнерам. Однако все эти методы не лишены недостатков. Например, протоколы с симметричным ключом слишком сложны для анализа и требуют постоянно работающих долговременных ключей для обмена закрытой информацией между пользователями и центром доверия.

Говорят, что криптосистема обладает *прогрессивной секретностью*, если компрометация долгосрочного секретного ключа в какой-

то момент времени не приводит к вскрытию тайны прошлой переписки, осуществлявшейся с использованием этого ключа. Распределение ключей посредством систем с открытым ключом не обладает прогрессивной секретностью. Чтобы понять, почему это важно, предположим, что Вы шифруете видеопередачу сеансовым ключом, который был получен с помощью открытого ключа *RSA* адресата. Допустим, что в не очень отдаленном будущем секретный ключ Вашего адресата будет вскрыт. В этот момент будет дешифрована и Ваша секретная видеопередача, если, конечно, атакующий озаботился ее вовремя записать.

Кроме того, при передаче ключей подразумевается, что получатель доверяет отправителю в его способности генерировать хороший сеансовый ключ. Иногда у получателя может возникнуть желание внести свои собственные случайные изменения в сеансовый ключ. Однако это можно сделать только если обе переписывающиеся стороны находятся на связи в один и тот же момент времени. Передавать же ключ, вообще говоря, удобнее, когда в сети находится только посылающий информацию, например, с помощью электронной почты.

Проблема распределения ключей была решена в той же основополагающей работе Диффи и Хеллмана, где разработана схема шифрования с открытым ключом. Их протокол распределения ключей, названный протоколом Диффи–Хеллмана обмена ключей, позволяет двум сторонам достигнуть соглашения о секретном ключе по открытому каналу связи без предварительной личной встречи. Его стойкость основывается на трудноразрешимой проблеме дискретного логарифмирования в конечной абелевой группе A.

В своей работе авторы предлагали использовать группу $A = \mathbb{F}_P^*$, но на сегодняшний день многие эффективные версии этого протокола берут за основу группу эллиптической кривой. Такие версии обозначают аббревиатурой *EC-DH*, возникшей от сокращений английских терминов: *Elliptic Curve* и *Diffie–Hellman*. Основные сообщения в протоколе Диффи–Хеллмана представлены следующей диаграммой[1]:

Алиса		Боб
a	$G^a \longrightarrow$	G^a
G^b	$\longleftarrow G^b$	b

[1]Напомню, что строчными знаками обозначаются секретные элементы протокола, а прописными — открытые. — *Прим. перев.*

Каждая из сторон обладает своим кратковременным секретным ключом a и b соответственно, с помощью которых они могут договориться об общем сеансовом ключе:

- Алиса может вычислить $k = (G^b)^a$, поскольку она знает a и посланное Бобом сообщение G^b.
- Боб тоже может определить $k = (G^a)^b$, поскольку ему известно b, а G^a он узнает от Алисы.

Атакующая Ева может перехватить сообщения G^a и G^b и попытаться восстановить секретный ключ, равный $k = G^{ab}$, что является в точности задачей Диффи – Хеллмана, рассмотренной в главе 7. Следовательно, криптостойкость этого протокола основывается не на дискретном логарифмировании, а на сложности решения задачи Диффи – Хеллмана. Напомним, что, возможно, задача Диффи – Хеллмана более легкая, чем дискретное логарифмирование, хотя в это с трудом верится, когда речь идет о группах, традиционно используемых в реальных протоколах.

Обратите внимание на то, что протокол Диффи – Хеллмана может нормально работать и в режиме онлайн (когда обе стороны вносят свои поправки в вырабатываемый ключ), и в режиме оффлайн (когда один из партнеров использует долговременный ключ вида G^a вместо кратковременного). Значит, с помощью протокола Диффи – Хеллмана можно как договариваться о ключах, так и менять их.

Приведем простой пример. Заметим, что для практических целей выбирают $\mathbb{F}_P$ с $P \approx 2^{1024}$, но мы в качестве параметров возьмем

$$P = 2\,147\,483\,659 \quad \text{и} \quad G = 2.$$

На диаграмме изображен возможный обмен сообщениями в протоколе Диффи – Хеллмана:

Алиса		Боб
$a = 12\,345$		$b = 654\,323$
$A = G^a = 428\,647\,416$	$\longrightarrow$	$A = 428\,647\,416$
$B = 450\,904\,856$	$\longleftarrow$	$B = G^b = 450\,904\,856$

Общий ключ вычисляется по формулам:

$$A^b = 428\,647\,416^{654\,323} \pmod{P} = 1\,333\,327\,162,$$
$$B^a = 450\,904\,856^{12\,345} \pmod{P} = 1\,333\,327\,162.$$

Обратите внимание на то, что в протоколе передаются элементы

выбранной абелевой группы. Следовательно, при использовании $\mathbb{F}_P^*$ затраты на передачи составят около 1024 бит в каждом направлении, поскольку $P \approx 2^{1024}$. Однако если используется группа эллиптической кривой $E(\mathbb{F}_Q)$, можно брать $Q \approx 2^{160}$ и снизить затраты на передачу сообщений до 160 бит на сообщение. Кроме того, возведение в степень на эллиптической кривой осуществляется более эффективно, чем в числовом поле.

Для «игрушечного» примера протокола *EC-DH* возьмем эллиптическую кривую, задаваемую уравнением

$$E:\ Y^2 = X^3 + X - 3$$

над полем $\mathbb{F}_{199}$. Пусть элемент G имеет координаты $(1, 76)$. Тогда возможные сообщения протокола имеют вид:

Алиса		Боб
$a = 23$		$b = 86$
$A = [a]G = (2, 150)$	$\longrightarrow$	$A = (2, 150)$
$B = (123, 187)$	$\longleftarrow$	$B = [b]G = (123, 187)$.

Общим ключом служит X-координата точки, вычисляемой следующим образом:

$$[b]A = [86](2, 150) = (156, 75),$$
$$[a]B = [23](123, 187) = (156, 75).$$

Общий ключ здесь — 156. Кроме того, вместо полной точки кривой можно передавать сжатую точку, что существенным образом скажется на времени передачи.

Итак, задача распределения ключей кажется решенной. Однако остается важная проблема: откуда Вы знаете *с кем* Вы договариваетесь о сеансовом ключе? У Алисы нет оснований для уверенности, что она переписывается именно с Бобом. Это может привести к следующей атаке, которая условно называется «человек посередине»:

Алиса		Ева		Боб
a	$\longrightarrow$	G^a		
G^m	$\longleftarrow$	m		
G^{am}		G^{am}		
		n	$\longrightarrow$	G^n
		G^b	$\longleftarrow$	b
		G^{bn}		G^{bn}

При этом

- Алиса договаривается о ключе с Евой, думая, что переписывается с Бобом;
- Боб ведет переговоры о ключе с Евой, считая, что его корреспондент — Алиса;
- Ева может изучать сообщения, т. к. они проходят через нее как через коммутатор. Поскольку она не вносит изменений в открытый текст, ее действия не могут быть обнаружены.

Итак, можно сделать вывод о том, что самого по себе протокола Диффи – Хеллмана не достаточно для обеспечения секретности распределения ключей.

10.2. Схемы цифровой подписи

Одна из возможностей пресечь атаку на протокол Диффи – Хеллмана, описанную в конце предыдущего параграфа, заключается в визировании сообщений. В этом случае обе стороны доподлинно знают, с кем ведут обмен сообщениями. Подписи — важнейший момент в криптографии с открытым ключом. Они были предложены Диффи и Хеллманом все в той же статье 1976 года, но первая практическая разработка подписи принадлежит Ривесту, Шамиру и Адлеману. Движущая пружина подписей в криптографии с открытым ключом изображена на следующей схеме:

СООБЩЕНИЕ + секретный ключ Алисы = ПОДПИСЬ,
СООБЩЕНИЕ + ПОДПИСЬ + ОТКРЫТЫЙ КЛЮЧ АЛИСЫ = ДА/НЕТ.

Эта схема называется *схемой подписи с приложением*, так как подпись добавляется в конец сообщения перед его передачей. Полученное сообщение подается на вход процедуры проверки подписи. Другой вариант — *схема подписи с восстановлением сообщения*, когда сообщение восстанавливается на выходе процедуры проверки подписи:

СООБЩЕНИЕ + секретный ключ Алисы = ПОДПИСЬ,
ПОДПИСЬ + ОТКРЫТЫЙ КЛЮЧ АЛИСЫ = ДА/НЕТ + СООБЩЕНИЕ.

Важным моментом здесь является то, что *только* Алиса может подписать свое сообщение, поскольку *только* она имеет доступ к своему секретному ключу. С другой стороны, ее подпись может быть

проверена любым желающим, поскольку для этого нужен лишь открытый ключ Алисы.

Главная проблема состоит в том, каким открытым ключам можно доверять. Как удостовериться, что тот или иной открытый ключ принадлежит конкретному лицу? Вы можете считать, что данный ключ использует Алиса, в то время как он имеет отношение к Еве. Поэтому Ева может подписывать чеки или нечто подобное, а Вы будете считать, что все это исходит от Алисы. Кажется мы столкнулись с той же проблемой управления ключами, что имела место в симметричных системах, хотя акцент теперь падает не на сохранение ключей в тайне, а на проверку их подлинности. Мы вернемся к этой проблеме позже.

Более формально, схема цифровой подписи состоит из таких двух преобразований:

- секретное преобразование подписи s,
- открытое преобразование проверки V.

Далее будем предполагать, что используется схема подписи с восстановлением сообщения, поскольку схема подписи с приложением несильно от нее отличается.

Алиса, посылая сообщение M, вычисляет $S = s(M)$ и передает результат S, где S — цифровая подпись на сообщении M. Заметим, что мы сейчас не заботимся о секретности сообщения, т. к. для нас сейчас важно лишь то, кто его отправил. Если существенна и конфиденциальность сообщения, то подпись S можно зашифровать, используя, например, открытый ключ адресата.

Получатель подписи S применяет открытое преобразование проверки V к S и получает на выходе процедуры сообщение M и некоторый бит v, который отвечает за результат проверки подписи. Если результат проверки положителен, то адресат получает уверенность в следующем:

- в целостности сообщения, т. е. в том, что оно не было изменено при передаче;
- в его оригинальности, т. е. в том, что сообщение было послано именно Алисой;
- в отсутствии *ренегатства*: Алиса не сможет утверждать, что не посылала сообщения.

Заметим, что первые два момента из упомянутых присущи также кодам аутентификации сообщений MAC. Однако последнего свойства, уверенности в отсутствии ренегатства, в схемах MAC нет.

А оно имеет важное значение в электронной коммерции. Чтобы убедиться в этом, представьте себе, что случится, если Вы начнете отрицать факт подписания чека, в действительности подписанного Вами.

Алгоритм шифрования *RSA* представляет особый интерес, поскольку его можно непосредственно использовать в качестве алгоритма подписи с восстановлением сообщения.

- Отправитель применяет расшифровывающее преобразование *RSA*, чтобы поставить подпись, беря сообщение и возводя его в секретную степень d: $S = m^d \pmod N$.
- Получатель использует шифрующую функцию *RSA* и восстанавливает оригинальное сообщение: $M = S^E \pmod N$.

При этом встает вопрос о проверке законности подписи. Если исходное сообщение было написано на естественном языке, то мы можем проверить, что восстановленное сообщение написано на том же языке. Но это не очень удачное решение. Лучшим выходом из ситуации служит добавление к сообщению некоторой избыточной информации.

Один из способов сделать это заключается в следующем. Предположим, сообщение D состоит из t битов, а модуль N алгоритма *RSA* насчитывает k битов, причем $t < k - 32$. Тогда мы дополняем сообщение D справа нулями до длины, кратной 8. Затем мы прибавляем к D $(k - t)/8$ байтов слева и получаем строку байтов вида

$$M = 00||01||FF||FF\cdots||FF||00||D,$$

после чего подпись вычисляется посредством формулы:

$$M^d \pmod N.$$

При проверке подписи правильность восстановления сообщения M подтверждается корректностью дополнения.

К сожалению, далеко не все сообщения имеют малую длину, позволяющую применить вышеупомянутый метод. Следовательно, с наивной точки зрения для применения алгоритма подписи *RSA* к длинному сообщению нам нужно разбить его на блоки и подписывать каждый из них. Такой способ съедает слишком много времени, если сообщение действительно длинное. Более того, при разбиении сообщения на блоки нам необходимо добавлять еще некоторые избыточные данные, страхуясь от атак, при которых можно было бы удалить часть блоков незаметно для получателя, как это было отмечено при рассмотрении режима шифрования *ECB*. Такая проблема возникает в связи с тем, что наша схема подписи основана на вос-

становлении сообщения, т. е. сообщение восстанавливается из подписи и процедуры проверки. При использовании схемы подписи с приложением мы могли бы перемешать (хэшировать) сообщение и подписать результат. Однако здесь не годится любая старая хэш-функция. Нам необходимы хэш-функции со специальными свойствами, о которых мы сейчас будем говорить.

10.3. Хэш-функции

Криптографическая *хэш-функция* h — это функция, определенная на битовых строках произвольной длины со значениями в строках битов фиксированной длины. Ее значение часто называют *хэш-кодом* или *хэш-значением*. В информатике тоже используются своего рода хэш-функции, но важное отличие криптографических хэш-функций от стандартных состоит в том, что первые должны быть односторонними. Другими словами, должно быть невозможно в вычислительном отношении по элементу Y из множества значений хэш-функции подобрать такой x из области определения, при котором $h(x) = Y$. Другая характеризация односторонних хэш-функций — сказать о них, что они защищены от восстановления прообразов. Применение криптографических хэш-функций позволяет создать схему подписи *RSA* без восстановления сообщения, что намного эффективней для длинных сообщений.

Предположим, нам дано длинное сообщение M для визирования. Сначала вычисляется $h(M)$ и потом применяется преобразование подписи *RSA* к хэш-значению $h(M)$, т. е. подпись получается как

$$S = h(M)^d \pmod{N}.$$

Наконец, подпись и само сообщение передаются вместе в виде пары (M, S). Проверка пары (M, S) состоит из трех этапов:

- «Шифрование» S с помощью шифрующей экспоненты *RSA* для получения H':
$$H' = S^E \pmod{N}.$$
- Вычисление $h(M)$ по M.
- Проверка равенства $H' = h(M)$. Если оно верно, то подпись законна. В противном случае — незаконна.

На самом деле при практическом использовании тут тоже нужно пополнение, но его можно делать например так, как в схеме подписи с восстановлением сообщения.

Итак, почему нам нужна хэш-функция со свойством односторонности? Ответ заключается в том, что свойство односторонности препятствует криптоаналитику фабриковать сообщение с данной подписью. Предположите, например, что мы используем только что описанную схему подписи *RSA* с приложением, но хэш-функцию берем стандартную, а не одностороннюю. Тогда возможна следующая атака.

- Ева вычисляет $H' = R^E \pmod N$ с некоторым выбранным наугад целом числом R.
- Кроме того, она находит прообраз значения H' при отображении h (напомним, что h сейчас стандартная хэш-функция и вычисление прообраза вполне возможно), т. е. Ева определяет $M = h^{-1}(H')$.

Теперь Ева обладает Вашей подписью (M, R) сообщения M. Такая подделка называется *экзистенциальной*. В ней нападающий не имеет контроля над содержанием сообщения, на которое он поставил Вашу подпись.

На практике требуется нечто большее, чем свойство односторонности. Хэш-функцию h называют *защищенной от повторений*, если вычислительно невозможно найти два таких различных значения x и x', при которых $h(x) = h(x')$.

Это требование помогает избежать следующих атак со стороны подписывающего лица.

- Отправитель выбирает два сообщения M и M', удовлетворяющие соотношению $h(M) = h(M')$.
- Он подписывает M и получает подпись (M, S).
- Потом он отказывается от своего сообщения, утверждая, что посылал сообщение M'.

Например, можно предположить, что M — электронный чек на сумму в 1000 евро, в то время как M' — чек на сумму в 10 евро. Ясно, что получится неприятность.

Ввиду парадокса дней рождений защищенные от повторений хэш-функции строятся сложнее, чем односторонние. Чтобы найти повторы в значениях хэш-функции f, нам необходимо запоминать результаты вычисления $f(x_1)$, $f(x_2)$, $f(x_3)$, и т. д., пока не встретится повторение. Если значения этой функции имеют длину n битов, то ожидаемое совпадение появится через $O(2^{n/2})$ итераций. Эту оценку стоит сравнить с числом шагов, необходимых для вычисления прообраза, которое имеет порядок $O(2^n)$ для корректно заданной

хэш-функции. Следовательно, для достижения необходимого уровня стойкости устойчивая от повторений функция, определенная на 80-битовых строках должна принимать значения в 160-битовых строках.

Но и этого еще недостаточно. Криптографическая хэш-функция должна обладать свойством *защищенности от вторых прообразов*, т. е. по данному M должно быть практически невозможно отыскать такой $M' \neq M$, для которого $h(M) = h(M')$. Это требование обеспечивает стойкость против следующих атак:

- нападающий получает Вашу подпись (M, S) на сообщении M;
- находит другое сообщение M', для которого $h(M) = h(M')$;
- ставит Вашу подпись (M', S) на своем сообщении M'.

В итоге криптографические хэш-функции должны обладать следующими тремя свойствами:

1) **защищенностью от восстановления прообразов**: должно быть невозможно в вычислительном отношении найти сообщение с данным значением хэш-функции;
2) **защищенностью от повторений**: вычислительно невозможно найти два сообщения с одним и тем же значением хэш-функции;
3) **защищенностью от вторых прообразов**: по данному сообщению нереально найти другое сообщение с тем же значением хэш-функции.

Как же соотносятся эти свойства? Как и в случае анализа криптосистем с открытым ключом, мы можем проследить связи свойств, сводя одно из них к другому.

Лемма 10.1. Свойство защищенности от восстановления прообразов сильнее защищенности от повторений и вторых прообразов.

Доказательство. Пусть h — функция, а $\mathcal{O}$ обозначает оракул, который по данному y находит такой x, что $h(x) = y$, т. е. $\mathcal{O}$ — оракул, взламывающий защищенность от восстановления прообразов функции h.

Используя $\mathcal{O}$, мы можем найти повторения в значениях h, выбрав x наугад и вычислив $y = h(x)$. Подставив найденное значение y в оракул, мы найдем x', удовлетворяющий равенству $y = h(x')$. Поскольку область определения функции h бесконечна, вероятность совпадения $x = x'$ крайне мала. Следовательно, мы нашли повторение значений функции h.

Аналогичным рассуждением можно показать, как используя алгоритм, взламывающий защищенность от восстановления прообразов, можно найти второй прообраз. ■

Лемма 10.2. Защищенность от вторых прообразов сильнее защищенности от повторений.

Доказательство. Пусть существует оракул $\mathcal{O}$, который по данному x может отыскать $x' \neq x$, для которого $h(x') = h(x)$. Ясно, что с его помощью мы легко найдем повторения, взяв произвольное x и применив оракул. ■

Хотелось бы основывать криптографические схемы на самых сильных свойствах. Но в этой главе мы будем считать, что все используемые хэш-функции защищены от повторений, т. е. выберем самое слабое свойство. В следующих главах мы познакомимся с криптографическими схемами, в которых требуются функции, защищенные от вторых прообразов.

Заметим, что криптостойкость любой схемы цифровой подписи, использующей криптографические хэш-функции, зависит как от сложности подлежащей математической задачи, такой, например, как задачи факторизации или дискретного логарифмирования, так и от стойкости хэш-функции.

Криптографические хэш-функции без повторений можно применять как основу MAC. Одна из возможностей построения кода MAC, опирающегося на хэш-функции, состоит в присоединении к сообщению ключа и применении к такой связке хэш-функции, т. е.

$$MAC = h(M||k),$$

хотя этот способ не считают особенно удачным. Во многих работах коды аутентификации сообщений, работающие по этому же, но чуть более сложному (в целях повышения стойкости) принципу, называют $HMAC$:

$$HMAC = h(k||P_1||h(k||P_2||M)),$$

где P_1 и P_2 — последовательности символов, используемые для пополнения входных данных до полных блоков.

Хэш-функции можно также рассматривать как специальный тип кодов обнаружения вмешательства, MDC (от англ. manipulation detection code). Например, хэш-функцию можно использовать для защиты целостности больших файлов так, как это делается в некоторых программах защиты от вирусов. Вычисляется значение хэш-

функции от содержания файла и либо хранится в каком-то безопасном месте (например, на дискете, спрятанной в сейф), или помещается в файл с подобными данными, который затем снабжается цифровой подпись, препятствующей дальнейшему вмешательству.

Основной принцип проектирования хэш-функции заключается в том, что ее значения должны производить лавинный эффект. Другими словами, небольшое изменение в аргументе хэш-функции должно очень сильно повлиять на ее значение. Это обеспечивает дополнительную стойкость, например, чтобы подпись на чеке в 30 фунтов нельзя было переделать в подпись на чеке в 30 000 фунтов, и наоборот.

Чтобы значения хэш-функции, применяемой в системах с низким уровнем стойкости, были свободны от повторений, их длина должна быть приблизительно равна 128 битам, но предпочтительнее значения в 160 битов.

10.3.1. Семейство *MD4*

Несколько хэш-функций находят широкое применение. Все они по своей природе итерационны. Назовем три из них: *MD5*, *RIPEMD-160* и *SHA-1*. Алгоритм *MD5* производит 128-битовые строки, в то время как значения *RIPEMD-160* и *SHA-1* имеют длину в 160 битов. Недавно национальный институт стандартов и технологий США предложил новые хэш-функции: *SHA-256*, *SHA-384* и *SHA-512*, значения которых — 256-, 384- и 512-битовые строки соответственно. Все они — обобщение более раннего и простого алгоритма, который называется *MD4*. Семь основных алгоритмов семейства *MD4* представлены следующим списком:

- *MD4*, состоящий из 3 раундов по 16 шагов в каждом со 128-битовой выходной строкой.
- *MD5*, насчитывающий 4 раунда по 16 шагов в каждом со 128-битовой выходной строкой.
- *SHA-1* включает в себя 4 раунда по 20 шагов в каждом. Длина его выходных данных — 160 битов.
- *RIPEMD-160* состоит из 5 раундов по 16 шагов в каждом и имеет длину выходной строки в 160 битов.
- *SHA-256* имеет 64 раунда по одному шагу, а длина его значения — 256 битов.
- *SHA-512* состоит из 80 одношаговых раундов и выдает строки в 512 битов.
- *SHA-384* практически идентичен *SHA-512* за исключением того, что его выход урезан до 384 битов.

Обсудим подробно *MD4*, поскольку остальные — более сложные версии этого алгоритма. Оставим их изучение по доступной литературе на усмотрение любознательного читателя. В *MD4* участвуют три поразрядные функции от трех 32-битовых переменных:

$$f(u,v,w) = (u \wedge v) \vee (\overline{u} \wedge w),$$
$$g(u,v,w) = (u \wedge v) \vee (u \wedge w) \vee (v \wedge w),$$
$$h(u,v,w) = u \oplus v \oplus w.$$

На протяжении всего алгоритма мы следим за текущим хэш-состоянием

$$(H_1, H_2, H_3, H_4)$$

32-битовых переменных, начальные значения которых равны

$$H_1 = '67452301h', \qquad H_3 = '98BADCFEh',$$
$$H_2 = 'EFCDAB89h', \qquad H_4 = '10325476h'.$$

Существуют различные фиксированные константы (y_i, z_i, s_i), свои для каждого раунда. Имеем

$$y_j = \begin{cases} 0, & 0 \leqslant j \leqslant 15, \\ '5A827999h', & 16 \leqslant j \leqslant 31, \\ '6ED9EBA1h', & 32 \leqslant j \leqslant 47. \end{cases}$$

Значения z_i и s_i приведены в следующих массивах:

$$z_{0\ldots15} = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],$$
$$z_{16\ldots31} = [0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15],$$
$$z_{32\ldots47} = [0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15],$$
$$s_{0\ldots15} = [3,7,11,19,3,7,11,19,3,7,11,19,3,7,11,19],$$
$$s_{16\ldots31} = [3,5,9,13,3,5,9,13,3,5,9,13,3,5,9,13],$$
$$s_{32\ldots47} = [3,9,11,15,3,9,11,15,3,9,11,15,3,9,11,15].$$

Поток данных состоит из 16 одновременно загружаемых слов в массив $X[j]$ $(0 \leqslant j < 16)$. Затем мы выполняем следующие преобразования над каждым из 16 слов, внесенных в поток данных:

```
(A,B,C,D) = (H[1],H[2,H[3],H[4]).
Выполнить первый раунд.
Выполнить второй раунд.
Выполнить третий раунд.
(H[1],H[2],H[3],H[4]) = (H[1]+A, H[2]+B, H[3]+C, H[4]+D).
```

После считывания всех данных выходные данные представляют собой конкатенацию окончательных значений переменных:

$$H_1, H_2, H_3, H_4.$$

Подробности раундов приведены в следующем описании, где $\lll$ обозначает побитывый циклический сдвиг влево:

- *Раунд 1*
  For $j = 0$ to 15 do
  1. $t = A + f(B, C, D) + X[z_j] + y_j$.
  2. $(A, B, C, D) = (D, t \lll s_j, B, C)$.
- *Раунд 2*
  For $j = 16$ to 31 do
  1. $t = A + g(B, C, D) + X[z_j] + y_j$.
  2. $(A, B, C, D) = (D, t \lll s_j, B, C)$.
- *Раунд 3*
  For $j = 32$ to 47 do
  1. $t = A + h(B, C, D) + X[z_j] + y_j$.
  2. $(A, B, C, D) = (D, t \lll s_j, B, C)$.

10.3.2. Хэш-функции и блочные шифры

Хэш-функции можно строить с помощью n-битового блочного шифра E_k. Есть несколько способов такого конструирования. Все они используют постоянное открытое начальное значение IV. Некоторые схемы привлекают еще функцию g, переводящую n-битовые строки в ключи.

Сначала сообщение пополняется до нужной длины и разбивается на группы x_0, x_1, ..., x_t, длина которых совпадает с размером блоков или длиной ключа используемого блочного шифра. Выбор размера групп зависит от точного определения хэш-функции, которую мы собираемся строить. Значение конструируемой хэш-функции совпадает с окончательным значением H_t, полученным итерацией следующего процесса:

$$H_0 = IV, \qquad H_i = f(x_i, H_{i-1}).$$

Определение функции f, участвующей в итерациях, зависит от используемой схемы. Мы предложим Вашему вниманию три возможности, хотя есть и другие.

- *Хэш-функция Матиаса–Мейера–Осиса*

$$f(x_i, H_{i-1}) = E_{g(H_{i-1})}(x_i) \oplus x_i.$$

- *Хэш-функция Давиэса–Мейера*

$$f(x_i, H_{i-1}) = E_{x_i}(H_{i-1}) \oplus H_{i-1}.$$

- *Хэш-функция Миягучи–Пренила*

$$f(x_i, H_{i-1}) = E_{g(H_{i-1})}(x_i) \oplus x_i \oplus H_{i-1}.$$

10.4. Алгоритмы цифровой подписи

Мы уже рассказывали об одной из схем цифровой подписи, а именно о схеме *RSA*. Можно спросить: зачем нужны еще какие-то схемы подписей? Ответ разобьем на несколько утверждений:

- Что если кто-то вскроет алгоритм *RSA* или решит задачу разложения на множители?
- Алгоритм *RSA* неприменим в некоторых приложениях, поскольку генерирование подписи с его помощью — слишком дорогостоящая операция.
- Сама *RSA*-подпись очень велика, а в некоторых практических приложениях требуется, чтобы подпись занимала мало места.

Один из алгоритмов, в котором учтены все эти возражения, называют алгоритмом цифровой подписи, или *DSA* (от английского эквивалента: Digital Signature Algorithm). Иногда его именуют стандартом цифровой подписи или *DSS* (Digital Signature Standard). Оригинальная версия алгоритма использовала мультипликативную группу $\mathbb{F}_P^*$ с большим простым числом P, сейчас же повсеместно пользуются группой эллиптической кривой. Такие методы обозначают *EC-DSA*, т. е. алгоритм цифровой подписи на эллиптических кривых. Этот вариант *DSA* работает быстрее, и подпись, полученная с его помощью, занимает места меньше, чем практически во всех других алгоритмах подписи. Более того, и размер ключа в нем сравнительно невелик.

Сначала мы опишем основной *DSA*-алгоритм, который работает с конечными полями. Здесь криптостойкость зиждется на сложности дискретного логарифмирования в поле $\mathbb{F}_P$.

DSA — алгоритм подписи с дополнением, в котором собственно подпись состоит из двух 160-битовых целых чисел R и S. Число R является функцией 160-разрядного случайного числа k, которое называют *эфемерным* ключом, изменяемым с каждым новым сообщением. Число S — функция от сообщения, секретного ключа x,

принадлежащего подписывающему сообщение лицу, числа R и эфемерного ключа k.

Аналогично алгоритму Эль-Гамаль, здесь есть несколько параметров домена, которые являются общими для ряда пользователей. Параметры домена в *DSA* открыты и очень похожи на параметры домена в криптосистеме Эль-Гамаль. Для их задания сначала фиксируется 160-битовое простое число Q, а затем выбирается такое большое простое число P, лежащее между 2^{512} и 2^{2048}, что $P-1$ делится на Q. Наконец, генерируется случайное число $H < P$ и вычисляется

$$G = H^{\frac{P-1}{Q}}.$$

Если $G = 1$, то переходят к другому случайному H до тех пор, пока не получат $G \neq 1$. Этим обеспечивается выполнение следующего условия: G — элемент группы $\mathbb{F}_P^*$ порядка Q, т. е.

$$G^Q = 1 \pmod{P}.$$

После выбора параметров домена (P, Q, G) каждый пользователь генерирует свой собственный секретный подписывающий ключ x, удовлетворяющий неравенству $0 < x < Q$. Соответствующим открытым ключом служит число Y, вычисляемое по правилу

$$Y = G^x \pmod{P}.$$

Обратите внимание на то, что процедура выбора пользовательской ключевой пары существенно проще, чем в *RSA*, поскольку она требует лишь одного возведения в степень в числовом поле.

Чтобы подписать сообщение M, пользователь осуществляет следующие шаги:

- вычисляет значение хэш-функции $H = h(M)$,
- выбирает случайный эфемерный ключ $0 < k < Q$,
- определяет

$$R = (G^k \pmod{P}) \pmod{Q},$$

- находит

$$S = \frac{H + xR}{k} \pmod{Q}.$$

Подписью сообщения M служит пара (R, S), имеющая в общей сложности 320 двоичных знаков.

Для проверки подписи (R, S) на сообщении M делают так:

- вычисляют хэш-значение $H = h(M)$,
- определяют $A = H/S \pmod{Q}$ и $B = R/S \pmod{Q}$,

- находят
$$V = (G^A Y^B \pmod P) \pmod Q,$$
где Y — открытый ключ автора сообщения.
- Подпись считается корректной тогда и только тогда, когда $V = R$.

Приведем пример работы *DSA*, выбрав малые параметры домена:

$$Q = 13, \quad P = 4Q + 1 = 53 \quad \text{и} \quad G = 16.$$

Предположим, что ключевая пара пользователя имеет вид $x = 3$ и $Y = G^3 \pmod P = 15$. Если мы хотим подписать сообщение с хэш-значением $H = 5$, то сначала нам нужно выбрать эфемерный ключ $k = 2$ и найти

$$R = (G^k \pmod P) \pmod Q = 5,$$
$$S = (H + xR)/k \pmod Q = 10.$$

Для проверки нашей подписи получатель определяет

$$A = \frac{H}{S} \pmod Q = 7,$$
$$B = \frac{R}{S} \pmod Q = 7,$$
$$V = (G^A Y^B \pmod P) \pmod Q = 5.$$

Ввиду равенства $V = R$, делаем вывод о корректности подписи.

Алгоритм *DSA* использует подгруппу в $\mathbb{F}_P^*$ порядка Q, порожденную элементом G. Следовательно, задача дискретного логарифмирования должна решаться в циклической группе $\langle G \rangle$ порядка Q. Для обеспечения криптостойкости нам необходимо потребовать, чтобы

- $P > 2^{512}$, хотя, в целях предупреждения атаки, опирающиеся на решето в числовом поле, более благоразумно выбирать $P > 2^{1024}$;
- $Q > 2^{160}$; это предотвратит атаки с использованием алгоритма шаги младенца/шаги гиганта.

Следовательно, для достижения грубого эквивалента стойкости 80-битового алгоритма *DES*, нам нужно оперировать с целыми числами, насчитывающими около 1024 двоичных знаков. В результате *DSA* работает даже медленнее, чем *RSA*, поскольку арифметические операции в нем сложнее. Процедура проверки в *RSA* привлекает лишь одно возведение в степень по модулю 1024-разрядного числа, а в алгоритме *DSA* процедура проверки предусматривает два

возведения в степень по модулю 1024-разрядного числа. Кроме того, процедура подписания в *DSA* более сложная, поскольку при этом нам нужно вычислять значение S.

Основная причина этих проблем кроется в том, что теоретически *DSA*-алгоритм должен работать в абелевых группах порядка 2^{160}, но так как целые числа по модулю P поддаются атакам с помощью решета в числовом поле, на практике нам нужно работать с группой элементов по 1024 знаков каждый, что очень сильно замедляет работу.

К счастью, можно обобщить *DSA* на произвольную конечную абелеву группу, в которой сложно решается задача о дискретном логарифмировании. Мы можем использовать ту, где эта задача решается труднее всего, например, группу точек эллиптической кривой над конечным полем.

Пусть A — циклическая группа, порожденная элементом G. Будем считать, что

- элемент G имеет простой порядок $Q > 2^{160}$;
- дискретное логарифмирование по основанию G — трудноразрешимая задача;
- существует открытая функция $F : A \longrightarrow \mathbb{Z}/Q\mathbb{Z}$.

Сведем различия в средах оперирования *DSA* и *EC-DSA* в табл. 10.1

Таблица 10.1.

Параметр	*DSA*	*EC-DSA*
A	$\langle G\rangle \subset \mathbb{F}_P^*$	$\langle P\rangle \subset E(\mathbb{F}_P)$
G	$G \in \mathbb{F}_P^*$	$P \in E(\mathbb{F}_P)$
Y	G^x	$[x]P$
F	$(\mathrm{mod}\, Q)$	x-координата P по модулю Q

В обобщенной версии *DSA* каждый пользователь тоже генерирует секретный подписывающий ключ k и вычисляет открытый по формуле: $Y = G^x$. Подпись получается в результате следующего процесса:

- Вычисляется хэш-значение $H = h(M)$.
- Выбирается эфемерный ключ $0 < k < Q$.
- Определяется $R = F(G^k)$ и $S = (H + xR)/k \ (\mathrm{mod}\, Q)$.

Подписью сообщения M служит пара (R, S).

Для проверки подписи (R, S) на сообщении M, осуществляют следующую процедуру:

- Вычисляют хэш-значение $H = h(M)$.
- Определяют $A = \frac{H}{S} \pmod{Q}$
- и $B = \frac{R}{S} \pmod{Q}$.
- Находят $V = F(G^a Y^b)$, где Y — открытый ключ подписавшего сообщение.
- Подпись считается корректной, если $V = R$.

Стоит сравнить эти процедуры генерирования и проверки подписи с теми, что осуществляются в алгоритме *DSA*, и найти, в чем они отличаются. Заметим, что обычно все операции в алгоритмах *EC-DSA* записываются аддитивно, т. е. через знак «+», а мы оставили мультипликативную запись, чтобы удобнее было сравнивать две версии схемы.

Разберем «игрушечный» пример работы алгоритма *EC-DSA*, взяв за основу эллиптическую кривую над полем $\mathbb{F}_{199}$ вида

$$E: \quad Y^2 = X^3 + X + 3.$$

Число ее элементов равно $Q = 197$, т. е. является простым. Поэтому соответствующая группа — циклическая, а координаты ее образующей P — $(1, 76)$. Выберем $x = 29$ в качестве секретного ключа. Тогда соответствующий открытый ключ будет равен

$$Y = [x]P = [29](1, 76) = (113, 191).$$

Предположим, что владелец секретного ключа намерен подписать сообщение с хэш-значением $H(M) = 68$. Тогда ему нужно произвести эфемерный ключ, который мы выберем равным $k = 153$, и вычислить

$$\begin{aligned} R &= x\text{-коорд.}([k]P) = x\text{-коорд.}([153](1, 76)) = \\ &= x\text{-коорд.}((185, 35)) = 185. \end{aligned}$$

Далее он находит

$$\begin{aligned} S &= (h(M) + x \cdot R)/k \pmod{Q} = \\ &= (68 + 29 \cdot 185)/153 \pmod{197} = 78. \end{aligned}$$

Подпись, которую он посылает вместе с сообщением, — это пара $(R, S) = (185, 78)$.

Чтобы проверить подпись, мы вычисляем

$$\begin{aligned} A &= h(M)/S \pmod{Q} = 68/78 \pmod{197} = 112, \\ B &= R/S \pmod{Q} = 185/78 \pmod{197} = 15. \end{aligned}$$

После этого определяем

$$Z = [A]P + B[Y] = [112](1,76) + [15](113,191) = \\ = (111,60) + (122,140) = (185,35).$$

Подпись оказалась верной, поскольку

$$R = 185 = x\text{-коорд.}(Z).$$

10.5. Подпись Шнорра

Множество вариантов схем подписей основывается на дискретных логарифмах. С практической точки зрения интересен алгоритм, носящий название подписи Шнорра. Расскажем об этом алгоритме в его оригинальном виде, оставив читателю разбираться самостоятельно с его обобщением на эллиптические кривые.

Пусть A — открытая конечная абелева группа, порожденная элементом G простого порядка Q. Ключевая пара в алгоритме подписи Шнорра совпадает с такой же парой в *DSA*, а именно, секретным ключом служит целое число x из интервала $(0, Q)$, а открытый ключ определяется формулой $Y = G^x$. Чтобы подписать сообщение, в алгоритме Шнорра поступают следующим образом:

- Выбирают эфемерный ключ k из промежутка $(0, Q)$.
- Вычисляют соответствующий открытый ключ $R = G^k$.
- Находят $E = h(M||R)$. Обратите внимание на то, что значение хэш-функции зависит как от сообщения, так и от эфемерного открытого ключа.
- Вычисляют $S = k + x \cdot E \pmod{Q}$.

Полученная таким образом пара (E, S) является искомой подписью.

Проверка подписи довольно проста: вычисляют $R = G^S Y^{-E}$ и $h(M||R)$. Подпись корректна, если верно равенство $E = h(M||R)$.

Для примера выберем следующие параметры домена:

$$Q = 101, \quad P = 607 \quad \text{и} \quad G = 601.$$

Чтобы зафиксировать ключевую пару, положим $x = 3$ и

$$Y = G^x \pmod{P} = 391.$$

Затем генерируем эфемерный ключ $k = 65$ и вычисляем

$$R = G^k \pmod{p} = 223.$$

Теперь находим хэш-значение $E = h(M||R) \pmod{Q}$. Допустим, что при этом получилось $E = 93$. Тогда вторая компонента подписи

выглядит как

$$S = k + x \cdot E \pmod{Q} = 65 + 3 \cdot 93 \pmod{101} = 41.$$

В следующих главах мы увидим, что можно доказать криптостойкость алгоритма подписи Шнорра в предположении о трудноразрешимости задачи дискретного логарифмирования, в то время как никакого доказательства криптостойкости *DSA* пока не найдено.

Алгоритм подписи Шнорра был предложен к эксплуатации в процедурах запрос–ответ идентификации кредитных карточек, поскольку «отвечающая» часть подписи (значение S) легко проверяется, так как для этого нужно всего лишь одно умножение и одно сложение в конечном поле. Независимо от того, в какой группе мы работаем, эта заключительная стадия требует только арифметики по модулю относительно небольшого простого числа.

Чтобы увидеть, как работает алгоритм подписи при процедуре аутентификации, приведем следующий сценарий. Допустим, Вы хотите с помощью смарт-карты[1] подтвердить свою подлинность при проходе в какое-то закрытое помещение или при обращении к торговому автомату. Считыватель карты обладает копией Вашего открытого ключа Y, в то время как в карточке записан Ваш секретный ключ x. Пока Вы носите карточку в кармане, она генерирует некие обращения, которые фактически являются эфемерным открытым ключом вида $R = G^k$. Когда Вы помещаете смарт-карту в считывающее устройство, она передает тому одно из этих заранее вычисленных обращений. Считыватель выдает запрашивающее сообщение E. После чего Вашей карточке остается лишь найти

$$S = k + xE \pmod{Q}$$

и передать результат считывателю, который проверит «подпись» вычислением $G^S = RY^E$. Обратите внимание, что все требуемые вычисления при этом крайне просты для реализации.

Более подробно, если обозначить через K смарт-карту, а через A — читающий ее автомат, то

$$\begin{aligned} K &\longrightarrow A : R = G^k, \\ A &\longrightarrow K : E, \\ K &\longrightarrow A : S = k + xE \pmod{Q}. \end{aligned}$$

Таким образом, этот протокол состоит из трех фаз

$$\text{обращение} \longrightarrow \text{запрос} \longrightarrow \text{ответ},$$

[1] Кредитная карточка с микропроцессором. — *Прим. перев.*

которые присущи всем идентифицирующим протоколам. Мы познакомимся с большим числом таких протоколов, когда будем изучать в главе 13 доказательства с нулевым разглашением.

10.6. Подпись Ниберга–Руппеля

Может оказаться, что подпись на коротком сообщении будет более длинной, чем его основное содержание. Напомним, что *RSA* можно использовать и как схему подписи с приложением, и как схему подписи с восстановлением сообщения. Пока ни один из описанных нами алгоритмов подписи, основанный на дискретном логарифмировании, нельзя использовать в виде схемы с восстановлением сообщения. Сейчас мы приведем пример схемы подписи с восстановлением сообщения, называемый алгоритмом подписи Ниберга–Руппеля, основанный на вычислении логарифмов в открытой конечной абелевой группе A.

Все схемы подписи с восстановлением сообщения используют открытую функцию избыточности f. Функция f преобразует фактическое сообщение в данные, которые затем подписываются. Это действие напоминает то, что делает хэш-функция в схемах подписи с приложением. Однако в отличие от хэш-функции функция избыточности должна быть легко обратимой. В качестве простого примера можно взять

$$f = \begin{cases} \{0,1\}^{n/2} & \longrightarrow & \{0,1\}^n, \\ m & \mapsto & m||m. \end{cases}$$

Мы предполагаем, что множество значений функции f может быть вложено в группу A. В нашем описании мы будем использовать целые числа по модулю P, т. е. $A = \mathbb{F}_P^*$ и, как обычно, будем считать, что большое простое число Q делит $P-1$, а G — образующая подгруппы в A порядка Q. Ключевая пара выбирается стандартным способом для систем, основанных на дискретном логарифмировании:

$$(Y = G^x, x).$$

Алгоритм подписи Ниберга–Руппеля состоит в следующем:

1. Берут случайное число $k \in \mathbb{Z}/Q\mathbb{Z}$ и вычисляют $R = G^k \pmod P$.
2. Находят $E = f(M) \cdot R \pmod P$.
3. Определяют $S = x \cdot E + k \pmod Q$.

Подпись представляет собой пару (E, S). Исходя из этой пары и целого числа по модулю Q, нам нужно

- убедиться в том, что подпись принадлежит пользователю с открытым ключом Y;
- восстановить сообщение M.

В процедуре проверки подписи Ниберга–Руппеля по паре (E, S) и открытому ключу отправителя $Y = G^x$ вычисляют

$$U_1 = G^S Y^{-E} = G^{S-Ex} = G^k \pmod P,$$

$$U_2 = \frac{E}{U_1} \pmod P.$$

После этого убеждаются, что U_2 является значением функции избыточности, т. е. в справедливости равенства $U_2 = f(M) = M||M$. Если это равенство ложно, то подпись отклоняют. В противном случае восстанавливают сообщение по правилу $M = f^{-1}(U_2)$ и принимают подпись.

Пример. Выберем параметры домена:

$$Q = 101, \quad P = 607, \quad \text{и} \quad G = 601.$$

Пусть ключевая пара имеет вид: $x = 3$, $Y = G^x \pmod P = 391$.

Чтобы подписать сообщение $M = 12$ (в нашем маленьком примере M должно лежать на отрезке $[0, 15]$), вычисляем эфемерный ключ $k = 45$ и

$$R = G^k \pmod P = 143.$$

Допустим, $f(M) = M + 2^4 \cdot M$. Тогда $f(M) = 204$ и

$$E = f(M) \cdot R \pmod P = 36,$$

$$S = x \cdot E + k \pmod Q = 52.$$

Итак, наша подпись — это пара $(E, S) = (36, 52)$. Покажем теперь, как проверяется подпись и восстанавливается сообщение. Все начинается с вычисления

$$U_1 = G^S Y^{-E} = 143.$$

Обратите внимание на то, что полученное проверяющим значение U_1, совпадает со значением R, найденным лицом, подписывающим сообщение. Далее проверяющий находит

$$U_2 = E/U_1 \pmod P = 204.$$

Теперь необходимо убедиться в том, что найденное число U_2 представимо в виде $M + 2^4 M$ для некоторого целого числа $M \in [0, 15]$. Мы видим, что U_2 действительно так представляется, поэтому подпись корректна. Сообщение $M = 12$ восстанавливается как решение уравнения

$$M + 2^4 M = 204.$$

10.7. Соглашение об аутентифицированном ключе

Теперь мы знаем, как реализовать цифровые подписи, с помощью которых можно решить проблему, возникавшую в протоколе Диффи–Хеллмана распределения ключей. Напомним, что атака «человек посередине» достигала успеха потому, что каждая из сторон не знала, кто с ней обменивался сообщениями. Теперь мы в состоянии идентифицировать любую из сторон, обязав их подписывать свои послания.

Мы получим прогрессивную секретность, поскольку долговременный ключ для подписи используется лишь для установления подлинности сообщений и никак не задействован в передаче ключей.

Кроме того, мы можем выбрать одну из версий протокола Диффи–Хеллмана. Первая из них основывается на дискретном логарифмировании в конечных полях (*DH*), а другая берет за основу эллиптическую кривую (*EC-DH*). Существует по крайней мере три возможных варианта алгоритма подписания документов: *RSA*, *DSA* и *EC-DSA*. Считая, что для обеспечения криптостойкости необходимо выбирать в *RSA* модуль шифрования порядка 1024 битов, простое число в *DSA* около 2^{1024}, а порядок группы как в *DSE*, так и в *EC-DSA* приблизительно равен 2^{160}, мы получим следующий размер подписанных сообщений в протоколе Диффи–Хеллмана:

Таблица 10.2. Размер сообщений в протоколе Диффи–Хеллмана

Алгоритм	Размер сообщения	Размер подписи	Общий размер
DH+DSA	1024	320	1344
DH+RSA	1024	1024	2048
EC-DH+RSA	160	1024	1184
EC-DH+EC-DSA	160	320	480

Видно, что при этом получаются слишком большие накладные расходы, а нам нужно всего лишь договориться об использовании сеансового ключа, размер которого может быть всего 128 битов.

Для уменьшения объема сообщения Менезис, Кью и Ванстоун разработали протокол, названный протоколом *MQV*. Он основывается на задаче дискретного логарифмирования в группе A, порожденной элементом G. Его можно использовать как в конечном поле, так и на эллиптических кривых для обмена аутентифицированным ключом, причем размер сообщений дается таблицей 10.3.

Таким образом, *MQV*-протокол дает нам значительную экономию в размере передаваемых данных. Протокол работает в предпо-

ложении, что Алиса и Боб имеют долговременную ключевую пару, состоящую из открытого и секретного ключей, которые мы обозначим

$$(A = G^{\alpha}, \alpha) \quad \text{и} \quad (B = G^{\beta}, \beta).$$

Будем предполагать следующее: Боб знает, что A — аутентификационный открытый ключ Алисы, а Алиса уверена, что B — ключ с теми же свойствами, принадлежащий Бобу. Установление подлинности открытых ключей может быть обеспечено применением некоторой формы сертификатов открытых ключей, о которых пойдет речь в следующих главах.

Таблица 10.3. Размер сообщений в протоколе *MQV*

Протокол	Размер сообщения
DL-MQV	1024
EC-MQV	160

Предположим, что Боб с Алисой намерены достигнуть соглашения о секретном сеансовом ключе. При этом они генерируют уникальные случайные вставки, обеспечивающие стойкость протокола и означающие, что ни одна из договаривающихся сторон не обязана доверять другой в выборе сеансового ключа. Алиса с Бобом производят свою эфемерную ключевую пару

$$C = (G^{\gamma}, \gamma) \quad \text{и} \quad (D = G^{\delta}, \delta)$$

и обмениваются информацией

$$\text{Алиса} \longrightarrow \text{Боб: } G^{\gamma},$$
$$\text{Боб} \longrightarrow \text{Алиса: } G^{\delta}.$$

До некоторой степени это напоминает стандартный протокол Диффи–Хеллмана без всяких подписей. Однако дело в том, что здесь окончательный сеансовый ключ будет зависеть также от долговременных ключей A и B.

Допустим, что Вы — Алиса. Тогда Вы знаете

$$A,\ B,\ C,\ D,\ \alpha \text{ и } \gamma.$$

Пусть l обозначает половину битового размера порядка группы A, например, если порядок группы $Q \approx 2^{160}$, то $l = 160/2 = 80$. Чтобы определить сеансовый ключ, Алиса производит следующие операции:

1) преобразует C в целое число I,
2) полагает $S_A = (i \pmod{2^l}) + 2^l$,
3) преобразует D в целое число j,

4) вычисляет $T_A = (j \pmod{2^l}) + 2^l$,
5) находит $h_A = \gamma + S_A \cdot \alpha$,
6) определяет $p_A = (DB^{T_A})^{h_A}$.

Боб делает то же, но со своей ключевой парой:

1) преобразует D в целое число I,
2) полагает $S_B = (i \pmod{2^l}) + 2^l$,
3) преобразует C в целое число j,
4) вычисляет $T_B = (j \pmod{2^l}) + 2^l$,
5) находит $h_B = \delta + S_B \cdot \beta$,
6) определяет $p_B = (CA^{T_B})^{h_B}$.

В результате, $p_A = p_B$ — общий секрет. Дабы убедиться, что величина p_A, найденная Алисой, совпадает с p_B, вычисленной Бобом, заметим, что $S_A = T_B$ и $S_B = T_A$. Поэтому

$$\begin{aligned}\log_G(p_A) &= \log_G\left((DB^{T_A})^{h_A}\right) = (\delta + \beta T_A)h_A = \\ &= \delta(\gamma + S_A\alpha) + \beta T_A(\gamma + S_A\alpha) = \\ &= \delta(\gamma + T_B\alpha) + \beta S_B(\gamma + T_B\alpha) = \\ &= \gamma(\delta + S_B\beta) + \alpha T_B(\delta + S_B\beta) = \\ &= (\gamma + \alpha T_B)h_B = \log_G\left((CA^{T_B})^{h_B}\right) = \log_G(p_B).\end{aligned}$$

Краткое содержание главы

- Протокол Диффи–Хеллмана можно использовать для достижения соглашения о сеансовом секретном ключе по открытому каналу. Однако этот протокол восприимчив к атаке «человек посередине», поэтому нуждается в какой-нибудь форме удостоверения подлинности переговаривающихся сторон.
- Цифровая подпись обеспечивает аутентификацию как для долговременных, так и для краткосрочных целей. Ее применяют в виде двух схем: либо подпись с приложением, либо подпись с восстановлением сообщения.
- Используя алгоритм шифрования *RSA* в обратном направлении, можно получить схему подписи с открытым ключом, но необходимо комбинировать этот алгоритм с хэш-функциями, чтобы добиться стойкости как для коротких, так и для длинных сообщений.
- Для целей криптографии нужны односторонние, защищенные от повторений хэш-функции. Такие функции препятствуют

атакам, направленным на фальсификацию цифровой подписи. Благодаря парадоксу дней рождений, размер значений хэш-функции должен по крайней мере вдвое превышать размер первоначальной оценки вычислительных возможностей атакующего.

- *DSA* — алгоритм цифровой подписи, основанный на дискретных логарифмах. Он уменьшает размер подписи по сравнению с *RSA*, но работает медленнее. *EC-DSA* — вариант *DSA*, основанный на эллиптических кривых, являющийся более эффективным, чем оригинал.
- Существуют и другие алгоритмы подписи с различными свойствами, использующие дискретное логарифмирование. Мы рассмотрели алгоритмы Шнорра и Нуберга–Руппеля.
- Другой путь распределения ключей, не требующий цифровой подписи, состоит в применении системы *MQV*, которая предъявляет слабые требования к пропускной способности канала. При этом происходит неявное удостоверение подлинности ключа, о котором ведутся переговоры, с помощью комбинации эфемерного ключа и долговременного ключа каждого из пользователей.

Дополнительная литература

Информацию о более криптостойких схемах подписи, таких, как, например, одноразовые подписи, подписи с завершением процесса при ошибке, неоспоримые подписи, можно найти в книгах Стинсона и Шнайера. В них также хорошо изложены хэш-функции и коды аутентификации сообщений, хотя безусловно, самый лучший источник — *HAC*.

B. Schneier. *Applied Cryptography*. Wiley, 1996.

D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

Контрольные вопросы

10.1.1. Расскажите об атаке «человек посередине», направленной на протокол Диффи–Хеллмана.

10.1.2. Почему алгоритмы цифровой подписи решают проблему, связанную с атакой «человек посередине»?

10.1.3. Почему *MAC* не обеспечивает защиту от ренегатства?

10.1.4. Чем хэш-функции отличаются от блочных шифров?

10.1.5. Что означает каждое из следующих свойств хэш-функций:
- защищенность от восстановления прообразов;
- защищенность от повторений;
- защищенность от вторых прообразов ?

10.1.6. Опишите алгоритм подписи *DSA* и объясните, как в нем обеспечивается стойкость.

10.1.7. В чем преимущества алгоритма Шнорра перед *DSA*?

10.1.8. Объясните различия и совпадения в протоколах Диффи – Хеллмана и *MQV*.

Лабораторные работы

10.2.1. Напишите программы, реализующие алгоритмы *DSA*, Шнорра и Ниберга – Руппеля.

10.2.2. Реализуйте протокол *MQV* и сравните его со стандартной реализацией схемы подписи Диффи – Хеллмана. С какой точки зрения протокол *MQV* более эффективен: пропускной способности или времени работы?

10.2.3. Разработайте на программном уровне основанные на эллиптических кривых варианты алгоритмов *DSA*, Шнорра, Ниберга – Руппеля, Диффи – Хеллмана и *MQV*. Что более эффективно — Ваши варианты, или оригинальные, базирующиеся на проблеме дискретного логарифмирования в числовом поле?

Упражнения

10.3.1. Обсудите утверждение: «криптография с открытым ключом решает проблему распределения ключей с помощью аутентификационных методов и распределения открытых ключей.»

10.3.2. Покажите, что если пользователь применяет один и тот же эфемерный ключ в алгоритме *DSA* для подписи двух разных сообщений, то нападающий может раскрыть его долговременный секретный ключ.

10.3.3. Предположим, что $h_1 : \{0,1\}^{2n} \longrightarrow \{0,1\}^n$ — хэш-функция, защищенная от повторений. Определим

$$h_2 = \begin{cases} \{0,1\}^{4n} \longrightarrow \{0,1\}^n, \\ x_1||x_2 \mapsto h_1(h_1(x_1)||h_1(x_2)), \end{cases}$$

где $x_1, x_2 \in \{0,1\}^{2n}$. Покажите, что h_2 тоже защищена от повторений.

10.3.4. Фиксируем открытый ключ (N, E) в алгоритме RSA и определяем хэш-функцию h от сообщения M, состоящего из k блоков: $M = M_1 \dots M_k$, сопоставляющую ему элемент H_k, где $H_1 = M_1$ и

$$H_i = \left(H_{i-1}^E \pmod N\right) \oplus M_i, \quad i = 2, \dots, k.$$

Покажите, как найти повторяющиеся значения функции h.

10.3.5. Атака на функцию MAC, размер блоков и ключа в которой равен n, в идеале требует 2^n операций. Рассмотрите следующие хэш-функции, основанные на функциях MAC:

$$MAC = h(k||M), \qquad MAC = h(M||k).$$

Разработайте атаку, которая осуществляется менее чем за 2^n операций.

10.3.6. Говорят, что схема распределения ключей обладает свойством подтверждения, если каждая из сторон имеет гарантию, что ее партнер пользуется тем же ключом, что и она. Продумайте, как придать это свойство протоколу MQV.

ГЛАВА 11

РЕАЛИЗАЦИЯ ОПЕРАЦИЙ

Цели главы

- Показать, как работают алгоритмы возведения в степень.
- Объяснить, как эффективно реализовать арифметические операции в кольцах вычетов.
- Перечислить приемы, ускоряющие операции в алгоритмах *RSA* и *DSA*.
- Объяснить, как эффективно реализовать конечные поля характеристики 2.

11.1. Введение

В настоящей главе разбирается практическое воплощение криптографических операций. Разговор будет идти, главным образом, об операциях в криптосистемах с открытым ключом, поскольку на их примере можно рассказать почти обо всех необходимых операциях. Например, в *RSA* и *DSA* нам нужно возводить в степень по модулю чисел, двузначное представление которых насчитывает более тысячи знаков. Это означает, что нам нужно понять способы реализации как арифметики остатков, так и алгоритмов возведения в степень.

Существует и другая причина фокусировки внимания на алгоритмах с открытым ключом: они работают существенно медленнее, чем алгоритмы с симметричным ключом. Фактически эти алгоритмы могут выполняться настолько медленно, что их использование может серьезно тормозить работу сети и серверов Интернета. Следовательно, эффективная реализация операций весьма важна, если заботиться об экономии средств.

Поскольку *RSA* наиболее доступна для понимания, мы сконцентрируемся именно на ней, хотя будем упоминать и о других схемах, где работает специализированная техника вычислений. Будем говорить в основном об алгоритмах, используемых в программном обес-

печении. Хотя в аппаратных средствах применяется другая техника, она несомненно связана с методами, применяемыми в программах. Поэтому изучение способов программных реализаций имеет большое значение.

11.2. Алгоритмы возведения в степень

До сих пор в этой книге мы полагали, что вычисление $a^b \pmod c$ — простая операция. Она используется как в *RSA*, так и в системах, основанных на дискретном логарифмировании, таких как Эль-Гамаль и *DSA*. В этом параграфе мы будем заниматься алгоритмами возведения в степень, предполагая, что арифметические операции в кольце вычетов $\mathbb{Z}/N\mathbb{Z}$ реализуются достаточно эффективно. В следующих разделах мы рассмотрим и эти операции.

Как уже отмечалось, основная операция в *RSA* и *DSA* — возведение в степень по модулю целого числа, т. е.

$$M = C^d \pmod N.$$

Прежде всего отметим, что не имеет смысла сначала находить $R = C^d$, а потом определять его остаток от деления на N. Действительно, поступая таким образом при вычислении

$$123^5 \pmod{511} = 28\,153\,056\,843 \pmod{511} = 359,$$

мы получаем большой промежуточный результат — 28 153 056 843. В реальной ситуации, когда возводятся в степень числа с 1024 двоичными знаками, промежуточный результат будет иметь $2^{1024} \cdot 1024$ знаков. Только для записи таких чисел потребуется около 10^{301} гигабайт на винчестере.

Чтобы промежуточные результаты не были столь огромны, необходимо вспомнить, что мы работаем по модулю N. Но даже при этом стоит быть внимательным. Наивный подход к решению нашей задачи привел бы к следующим вычислениям:

$$\begin{aligned}
x &= 123,\\
x^2 &= x \cdot x \pmod{511} = 310,\\
x^3 &= x \cdot x^2 \pmod{511} = 316,\\
x^4 &= x \cdot x^3 \pmod{511} = 32,\\
x^5 &= x \cdot x^4 \pmod{511} = 359.
\end{aligned}$$

На это уходит 4 умножения в кольце вычетов, что кажется приемлемым для нашего игрушечного примера. Но в общем случае, при

возведении в 1024-разрядную степень потребуется около 2^{1024} таких умножений. Если каждое произведение при этом осуществляется за 1 миллионную долю секунды, нам потребуется 10^{294} лет на осуществление операции расшифрования в алгоритме *RSA*.

Однако даже на нашем небольшом примерчике легко увидеть, как можно было бы сократить количество умножений:

$$x = 123,$$
$$x^2 = x \cdot x \pmod{511} = 310,$$
$$x^4 = x^2 \cdot x^2 \pmod{511} = 32,$$
$$x^5 = x \cdot x^4 \pmod{511} = 359.$$

Здесь нам потребовалось только три произведения, а не 4, как раньше. Обратите внимание на то, что двоичное представление числа 5 имеет вид: $'101b'$, т. е.

- это трехзначное число,
- его вес Хемминга[1] равен $h = 2$.

Поэтому мы произвели $1 = h - 1$ умножение общего вида и $2 = t - 1$ возведения в квадрат. Такой подсчет остается справедливым и в общем случае: возведение в степень по модулю целого числа может быть осуществлено с помощью

- $h - 1$ умножений и
- $t - 1$ возведения в квадрат,

где t — количество знаков в двоичном представлении показателя, а h — его вес Хемминга. Усредненный вес Хемминга целого числа равен половине количества его двоичных знаков, т. е. $t/2$. Поэтому среднее число умножений и возведений в квадрат при вычислении экспоненты в кольце вычетов равно $t + t/2 - 1$. Это означает, что при возведении в 1024-битовую степень потребуется не более 2048 умножений, а среднее число операций и того меньше — 1535.

Метод, с помощью которого достигается такая экономия операций носит специальное название: *метод двоичного потенцирования*. Работает он, поочередно считывая знаки двоичного представления показателя, начиная с младшего разряда. Чтобы увидеть, как это происходит, запишем на псевдокоде алгоритм двоичного потенцирования, вычисляющего $y = x^d \pmod{n}$, где x, y, d и n — большие целые числа, представленные с помощью средств языков *C* или *Java*.

[1] Вес Хемминга числа равен количеству единиц в его двоичном представлении. — *Прим. перев.*

```
y=i;
while (d<>0)
   { if ((d%2)!=0)
       { y=(y*x)%n;
         d=d-1;
       }
     d/=2;
     x=(x*x)%n;
   }
```

Этот метод имеет много разных названий. Некоторые авторы называют его алгоритмом *квадратов и умножений*, поскольку он действительно осуществляется с помощью этих операций. Другие именуют его алгоритмом *индийского потенцирования*. Тот, что мы привели на псевдокоде, иногда величают алгоритмом потенцирования *справа-налево*, поскольку он перебирает знаки показателя, начиная от самого младшего, т. е. стоящего справа.

Можно реализовать этот метод и с помощью вызова рекурсивной функции. Приведенная ниже программа, написанная на языке *Haskell*, реализует двоичное потенцирование по модулю m. Для вычисления $a^n \pmod m$ используется процедура `bin_pow a m n`, определяемая следующим образом:

```
bin_pow :: Int -> Int -> Int -> Int
bin_pow a m n
   | n == 0          = 1
   | n `mod` 2 == 0 = bin_pow ((a*a) `mod` m) m (n `div` 2)
   | otherwise       = (a*t) `mod` m
      где t = bin_pow ((a*a) `mod` m) m ((n-1) `div` 2)
```

В большинстве случаев возведение в квадрат осуществляется быстрее, чем общее умножение. Следовательно, чтобы еще больше сократить время вычислений, стоит попытаться уменьшить количество общих умножений. Это можно сделать, применяя технику окон, которая использует заранее вычисленные значения.

Чтобы лучше понять этот новый метод, вновь обратимся к стандартному двоичному потенцированию. Но на этот раз вместо варианта *справа-налево*, мы начнем со старшего разряда показателя, осуществляя вариант *слева-направо* двоичного потенцирования. Предположим, что мы находим $y = x^d \pmod n$, и введем обозначе-

ния для двоичного представления показателя:

$$d = \sum_{i=0}^{t} d_i 2^i,$$

где $d_i \in \{0, 1\}$. Алгоритм слева–направо двоичного потенцирования выполняется следующим образом:

```
y=i;
for (i=t; i>=0; i--)
   { y = (y*y)%n;
     if (d[i]==1)
       { y = (y*x)%n; } }
```

Здесь обрабатывается один бит показателя за каждый проход цикла. При этом число возведений в квадрат равно t, а ожидаемое число общих умножений — $t/2$.

В оконном методе за один раз учитывается w знаков показателя. Сначала заполняется таблица заранее вычисленных значений

$$x_i = x^i \pmod n \quad \text{для } i = 0, \ldots, 2^w - 1.$$

Потом показатель представляется в виде

$$d = \sum_{i=0}^{t/w} d_i 2^{iw},$$

где $d_i \in \{0, 1, \ldots, 2^w - 1\}$. Реализация оконного метода выглядит так:

```
y=i;
for (i=t/w; i>=0; i--)
     { for (j=0; j<w; j++)
         { y = (y*y)%n; }
       j=d[i];
       y = (y*x[j])%n; }
```

Проследим за действиями алгоритма с шириной окна $w = 3$ на примере вычисления $y = x^{215} \pmod n$. Разложим показатель по степеням 2^3:

$$215 = 3 \cdot 2^6 + 2 \cdot 2^3 + 7.$$

Тогда

$$y = 1, \qquad y = y^8 = x^{24}, \qquad y = y^8 = x^{208},$$
$$y = y \cdot x^3 = x^3, \qquad y = y \cdot x^2 = x^{26}, \qquad y = y \cdot x^7 = x^{215}.$$

В оконном методе по-прежнему требуется t возведений в квадрат, но число общих умножений уменьшается в среднем до t/w. Можно добиться еще больших успехов, если применить метод скользящего окна, в котором показатель представляется в виде

$$d = \sum_{i=0}^{l} d_i 2^{e_i},$$

где $d_i \in \{1, 3, 5, \cdots 2^w - 1\}$, а $e_{i+1} - e_i \geqslant w$. Выбирая лишь нечетные значения для коэффициентов d_i, и меняя ширину окна, мы достигаем экономии памяти на хранение заранее найденных значений и увеличиваем эффективность вычислений. Считая, что уже определены степени $x[i] = x^i$ для $i = 1, 3, 5, \ldots, 2^w - 1$, представим метод скользящего окна на псевдокоде:

```
y=i;
for (i=l ; i>=0; i--)
   { for (j=0; j<e[i+1]-e[i]; j++)
       { y = (y*y)%n; }
     j=d[i];
     y = (y*x[j])%n;
   }
for (j=0; j<e[0]; j++)
   { y = (y*y)%n; }
```

Тут число возведений во вторую степень по-прежнему остается равным t, а число общих умножений сокращается в среднем до l до $t/(w+1)$. В нашем примере вычисления $y = x^{215} \pmod n$ происходит следующее:

$$215 = 3 \cdot 2^6 + 2^4 + 7;$$

$$y = 1, \qquad y = y^4 = x^{12}, \qquad y = y^{16} = x^{208},$$
$$y = y \cdot x^3 = x^3, \qquad y = y \cdot x = x^{13}, \qquad y = y \cdot x^7 = x^{215}.$$

Обратите внимание на то, что все описанные нами оконные методы применимы для возведения в степень в любой абелевой группе, а не только в кольце вычетов. Следовательно, их можно использовать для вычисления α^d в конечном поле или $[d]P$ на эллиптической

кривой, хотя в последнем случае происходит умножение точки на число.

В случае эллиптических кривых легко вычисляются обратные элементы, т. е. по данной точке P достаточно просто можно определить точку $-P$. Это позволяет разработать *знаковые* двоичный и оконный методы. Ограничимся рассказом о знаковом оконном методе. Сначала вычисляют

$$P_i = [i]P \text{ для } i = 1, 3, 5, \ldots 2^{w-1} - 1,$$

т. е. лишь половину значений, требуемых для скользящего оконного метода, или четверть необходимых данных в стандартном оконном методе. Теперь представляем показатель в виде

$$d = \sum_{i=0}^{l} d_i 2^{e_i},$$

где $d_i \in \{\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$. Знаковый оконный метод вычисления степеней на эллиптических кривых представлен следующим алгоритмом на псевдокоде:

```
Q=0;
for (i=l ; i>=0; i--)
   { for (j=0; j<e[i+l]-e[i]; j++)
       { Q = [2] Q; }
     j=d[i];
     if (j>0)  { Q = Q + P[j]; }
     else      { Q = Q - P[-j]; }
   }
for (j=0; j<e[0]; j++)
   { Q = [2] Q; }
```

11.3. Потенцирование в RSA

Для ускорения процесса возведения в степень в алгоритме *RSA* применяется множество приемов, специально разработанных для этой криптосистемы. Выбор ускоряющего метода зависит от того, выполняем ли мы процедуру шифрования (проверки подлинности подписи) с открытым ключом или обращаемся к процессу расшифрования (подписывания сообщения) с секретным ключом.

11.3.1. Шифрование (проверка подписи) в *RSA*

Как уже было отмечено в предыдущих главах, часто применяются достаточно небольшие открытые (шифрующие) экспоненты, например, $E = 3$, 17 или 65 537. Причина, по которой выбираются такие значения, состоит в том, что эти числа имеют небольшой вес Хемминга, фактически наименьший из возможных для открытых *RSA*-ключей, а именно, 2. Поэтому двоичный метод или любой другой разумный алгоритм потенцирования будет использовать лишь одно общее умножение и k возведений в квадрат, где k — число двоичных знаков открытой экспоненты. Например,

$$M^3 = M^2 \cdot M, \qquad M^{17} = M^{16} \cdot M = (((M^2)^2)^2)^2 \cdot M.$$

11.3.2. Расшифровывание (подписывание) в *RSA*

В случае расшифрования или подписывания сообщения в алгоритме *RSA* показатель вычисляемой степени будет общим 1000-битовым числом. Следовательно, нам нужен какой-нибудь путь сокращения числа операций. К счастью, мы знаем секретный ключ,а значит и разложение модуля алгоритма на множители: $N = pq$. При расшифровывании сообщения приходится вычислить

$$m = C^d \pmod N.$$

Для ускорения процесса сначала найдем m по модулям p и q:

$$m_p = C^d \pmod p = C^{d \pmod{p-1}} \pmod p,$$
$$m_q = C^d \pmod q = C^{d \pmod{q-1}} \pmod q.$$

Поскольку p и q — числа порядка 2^{512}, на эти действия потребуется два потенцирования с показателем в 512 знаков по модулю 512-битового числа. Это существенно быстрее, чем одно потенцирование с 1024-битовым показателем по модулю числа с 1024 двоичными знаками.

Теперь предстоит восстановить m по m_p и m_q, что можно сделать с помощью китайской теоремы об остатках. Вычисляем $t = p^{-1} \pmod q$ и храним его вместе с секретным ключом, а сообщение m восстанавливается по схеме:

$$u = (m_q - m_p)t \pmod q,$$
$$m = m_p + up.$$

Как раз для ускорения вычислений и надо хранить p и q вместе с секретным ключом, как было замечено в главе 7, хотя с точки зрения математики в этом нет никакой необходимости.

11.4. Потенцирование в DSA

Напомним, что при проверке подлинности подписи в алгоритме *DSA* нам необходимо вычислить

$$R = G^A Y^B.$$

Это можно сделать, поочередно возводя в степень сначала G, потом Y, а затем перемножая полученные значения. Однако зачастую проще возводить в степень эти элементы одновременно. Существует множество методов, реализующих эту идею, которые используют различные варианты оконной техники или что-либо похожее. Но все они по сути эксплуатируют один трюк, называемый *приемом Шамира*.

Сначала заполняется поисковая таблица

$$G_i = G^{i_0} Y^{i_1},$$

где $i = (i_0, i_1)$ — двоичное представление числа $i = 0, 1, 2, 3$. Затем заполняется массив показателей, исходя из данных A и B. Его размерность — $2 \times t$, где t — максимум из числа знаков A и B. Строки массива — двоичное представление показателей A и B. Через I_j ($j = 1, \ldots, t$) обозначаются числа, чье двоичное представление задается столбцами этого массива. Наконец, параметру r присваивают единичное значение и вычисляют

$$r = r^2 \cdot G_{I_j} \quad \text{для } j \text{ от } 1 \text{ до } t.$$

В качестве примера найдем $r = G^{11} Y^7$. В этом случае $t = 4$. Сначала мы вычисляем

$$G_0 = 1,\ G_1 = G,\ G_2 = Y,\ G_3 = G \cdot Y.$$

Поскольку в двоичной системе счисления числа 11 и 7 имеют вид $'1011b'$ и $'111b'$, массив показателей будет таким:

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

Отсюда находим I_j:

$$I_1 = 1, \quad I_2 = 2, \quad I_3 = 3, \quad I_4 = 3.$$

Таким образом, четыре шага алгоритма выглядят следующим образом:

$$r = G_1 = G,$$
$$r = r^2 \cdot G_2 = G^2 \cdot Y,$$

$$r = r^2 \cdot G_3 = (G^4 \cdot Y^2) \cdot (G \cdot Y) = G^5 \cdot Y^3,$$
$$r = r^2 \cdot G_3 = (G^{10} \cdot Y^6) \cdot (G \cdot Y) = G^{11} \cdot Y^7.$$

Заметим, что уловку, аналогичную приему Шамира, в случае эллиптических кривых можно разработать, используя знаковое представление показателя. Мы не станем объяснять здесь модификацию метода для кривых, а оставим это пытливым читателям как возможность для приложения своих способностей.

11.5. Арифметика многократной точности

Здесь объясняется, как осуществляются арифметические действия в кольце вычетов по 1024-значным модулям. Мы покажем, как эти действия реализуются на современных процессорах и почему наивные алгоритмы заменяются специализированной техникой, предложенной Монтгомери.

В криптографических приложениях арифметики остатков обычно фиксируется количество знаков чисел, над которыми совершаются действия, например, 1024 битов для алгоритмов *RSA* и *DSA*, и 200 битов для алгоритмов, основанных на эллиптических кривых. Это приводит к возможности выбора различных программ из библиотеки «арифметика многократной точности». В результате вопросы динамического распределения памяти снимаются и можно сосредоточиться на ускорении процесса.

Целые числа в компьютере представляются в так называемом «малословном» формате, т. е. если большое целое число x хранится в компьютере в виде набора $x_0, x_1, \ldots, x_n$, то x_0 — наименьшее значащее слово, а x_n — наибольшее. Например, в 32-разрядных машинах 64-битовое число x хранится в виде двух 32-битовых слов: $[x_0, x_1]$, где $x = x_1 \cdot 2^{32} + x_0$.

11.5.1. Сложение

Большинство современных процессоров имеют специальный признак переноса, учитывающий добавочное слагаемое к следующему разряду при побитовом сложении. Кроме того, существуют специальные функции, обычно называемые чем-то вроде **addc**, которые складывают два целых числа с учетом признака переноса. Так, если мы хотим сложить два 64-битовых числа $x = x_1 \cdot 2^{32} + x_0$ и $y = y_1 \cdot 2^{32} + y_0$, нам нужно вычислить:

$$z = x + y = z_2 \cdot 2^{64} + z_1 \cdot 2^{32} + z_0.$$

Значения z_i определяются следующим образом:

```
z0 <- add x0,y0
z1 <- addc x1,y1
z2 <- addc 0,0
```

Заметим, что значение z_2 не превышает 1 и z может оказаться 65-битовым числом. Описанная техника сложения двух 64-битовых целых чисел может быть без труда обобщена на сложение чисел любой фиксированной длины. Ее можно использовать и при вычитании.

11.5.2. Умножение в столбик

Теперь обратимся к следующей простейшей арифметической операции, идущей после сложения и вычитания, а именно, к умножению. Заметим, что при умножении двух 32-битовых чисел получится 64-битовое, и, естественно, большинство современных процессоров оснащено операцией, осуществляющей это действие.

$$w_1 \cdot w_2 = (\textit{Высший}, \textit{Низший}) = (H(w_1 \cdot w_2), L(w_1 \cdot w_2)).$$

При стандартном умножении в столбик мы получаем что-то вроде таких вычислений:

$$\begin{array}{rrrr}
 & & x_1 & x_0 \\
 & \times & y_1 & y_0 \\
\hline
 & & H(x_0 \cdot y_0) & L(x_0 \cdot y_0) \\
 & H(x_0 \cdot y_1) & L(x_0 \cdot y_1) & \\
 & H(x_1 \cdot y_0) & L(x_1 \cdot y_0) & \\
H(x_1 \cdot y_1) & L(x_1 \cdot y_1) & &
\end{array}$$

Затем все четыре строки складываются и получается искомый ответ $z = x \cdot y$. При этом, естественно, необходимо следить за переносами разрядов. На псевдокоде процесс умножения выглядит следующим образом.

```
(z1,z0)<-mul x0,y0
(z3,z2)<-mul x1,y1
(h,l)  <-mul x1,y0
z1     <-add z1,l
z2     <-addc z2,h
```

```
z3      <-addc z3,0
(h,l)   <-mul x0,y1
z1      <-add z1,l
z2      <-addc z2,h
z3      <-addc z3,0
```

Если n обозначает битовый размер целых чисел, над которыми производятся действия, то описанный выше способ перемножения требует $O(n^2)$ операций над битами, а сложение и вычитание — $O(n)$. Возникает естественный вопрос: можно ли умножить числа быстрее, чем за $O(n^2)$ операций?

11.5.3. Умножение Карацубы

Один из методов ускорения умножения носит название *умножения Карацубы*. Предположим, нам нужно перемножить два n-битовых числа x и y. Запишем их в виде

$$x = x_0 + 2^{n/2}x_1, \qquad y = y_0 + 2^{n/2}y_1,$$

где $0 \leqslant x_0, x_1, y_0, y_1 < 2^{n/2}$. Умножение происходит с помощью вычислений:

$$\begin{aligned} A &= x_0 \cdot y_0, \\ B &= (x_0 + x_1) \cdot (y_0 + y_1), \\ C &= x_1 \cdot y_1. \end{aligned}$$

Ответ получается в результате преобразований:

$$\begin{aligned} C2^n + (B - A - C)2^{n/2} + A = z_1y_1 2^n + (x_1y_0 + x_0y_1)2^{n/2} + x_0y_0 = \\ = (x_0 + 2^{n/2}x_1) \cdot (y_0 + 2^{n/2}y_1) = x \cdot y. \end{aligned}$$

Легко заметить, что при этом было использовано три умножения и два сложения $(n/2)$-битовых чисел, и три операции сложения–вычитания над n-битовыми числами. Обозначив через $M(n)$ количество операций, необходимых для умножения n-битовых чисел, а через $A(n)$ — для их сложения или вычитания, получим следующее соотношение:

$$M(n) = 3M(n/2) + 2A(n/2) + 3A(n).$$

Используя аппроксимацию $A(n) \approx n$, замечаем, что

$$M(n) \approx 3M(n/2) + 4n.$$

Если умножение $n/2$-битовых чисел происходит по той же схеме, то решая полученное рекуррентное соотношение, найдем, что при $n \to \infty$

$$M(n) \approx 9n^{\frac{\ln 3}{\ln 2}} \approx 9n^{1{,}58}.$$

Итак, у нас есть алгоритм с асимптотической сложностью $O(n^{1{,}58})$. Умножение Карацубы выполняется заметно быстрее, чем умножение в столбик, если умножаются числа, количество знаков в которых насчитывает несколько сотен. Однако можно умножать еще быстрее. Сложность наиболее эффективного из известных методов оценивается как

$$O(n \ln n \ln n \ln n).$$

Но ни этот последний метод, ни умножение Карацубы не используются в криптографических приложениях. Причины станут ясны после обсуждения деления.

11.5.4. Деление

После знакомства с умножением перейдем к самой сложной арифметической операции — делению. Оно необходимо, в частности, для определения остатков от деления двух целых чисел — основной задачи арифметики остатков, а значит и *RSA*.

Итак, для данных двух целых x и y чисел мы хотим найти такие q и r, что $x = qy + r$, причем, как обычно, $0 \leqslant r < y$. Напомним, что такое деление называется делением с остатком или евклидовым делением.

Представим делимые числа в малословном формате

$$x = (x_0, \ldots, x_n) \quad \text{и} \quad y = (y_0, \ldots, y_n),$$

где основание разложения равно $b = 2^w$, и обозначим через `t<<v` сдвиг целого числа t влево на v слов, т. е. результат умножения t на b^v. При этих соглашениях алгоритм на псевдокоде выглядит так:

```
r=x;
/* Тривиальный случай */
if (t>n)
   { q=0; return; }
q=0; s=0;
/* Нормализуем делитель */
while (y[t]<b/2)
   { y=y*2; r=r*2; s++; }
if (r[n+1]!=0) { n=n+1; }
```

```
/* Получаем частное */
while (r>=(y<<(n-t)))
   { q[n-t]=q[n-t]+1;
     r=r-(y<<n-t);
   }
/* Разбираемся с остатком */
for (i=n; i>=t+1; i --)
   { if (r[i]==v[t])
       { q[i-t-1]=b-1; }
     else
       { q[i-t-1]=целая_часть( (x[i]*b+r[i-1])/y[t] ); }

     if (t!=0) { rhs_m = y[t]*b+y[t-1]; }
     else      { rhs_m =y[t]*b;         }
     rhs=q[i-t-1]*rhs_m;

     if (i!=1) { lhs = x[i]*b*b+r[i-1]*b+r[i-2]; }
     else      { lhs = x[i]*b*b+r[i-1]*b;        }

     while (rhs>lhs)
       { q[i-t-1]=q[i-t-1]-1;
         rhs=rhs-rhs_m;
       }
     r=r-(q[i-t-1]*y)<<(i-t-1);
     if (r<0)
       { r=r+(y<<i-t-1);
         q[i-t-1]=q[i-t-1]-1;
       }
   }
/* Перенормировка */
for (i=0; i<s; i++) { r=r/2; }
```

Как видите, это очень сложная операция. Поэтому ее следует избегать по мере возможности.

11.5.5. Арифметика Монтгомери

Поскольку деление больших целых чисел — операция сложная, все наши криптографические действия будут идти очень медленно, если мы будем использовать стандартный метод деления, представленный в предыдущем параграфе. Напомним, что практически все кри-

птосистемы с открытым ключом работают с арифметикой остатков. Поэтому нам хотелось бы уметь вычислять остатки от деления, не прибегая к дорогостоящей операции деления. На первый взгляд это кажется невозможным, но если прибегнуть к специальной арифметике Монтгомери, то все окажется не так страшно.

Эта арифметика работает с альтернативным представлением целых чисел — представлением Монтгомери. Обозначим через b степень двойки, показатель которой — длина слов в нашем компьютере, т. е. $b = 2^{32}$ или $b = 2^{64}$. Выберем целое число R, удовлетворяющее неравенству $R = b^t > N$. Теперь вместо числа x будем хранить в памяти компьютера величину $x \cdot R \pmod N$ в малословном формате. Значение $x \cdot R \pmod N$ называется представлением Монтгомери числа $x \pmod N$.

Сложение двух целых чисел в этом представлении осуществляется просто. Нам даны $x \cdot R \pmod N$ и $y \cdot R \pmod N$, а нам нужно вычислить $z \cdot R \mod N$, где $z = x + y$. Представим нужное вычисление на псевдокоде:

```
zR = xR + yR
if (zR>=N) { zR-=N; }
```

Для прояснения его работы разберем пример.

$$N = 1\,073\,741\,827, \quad b = R = 2^{32} = 4\,294\,967\,296.$$

Запишем числа 1, 2 и 3 в представлении Монтгомери:

$$1 \longrightarrow 1 \cdot R \pmod N = 1\,073\,741\,815,$$
$$2 \longrightarrow 2 \cdot R \pmod N = 1\,073\,741\,803,$$
$$3 \longrightarrow 3 \cdot R \pmod N = 1\,073\,741\,791.$$

Теперь мы можем проверить работу сложения, поскольку в стандартном представлении результат нам известен: $1 + 2 = 3$. В представлении Монтгомери это действие отражается следующим образом:

$$1\,073\,741\,815 + 1\,073\,741\,803 = 1\,073\,741\,791 \pmod N.$$

Посмотрим, как происходит умножение в арифметике Монтгомери. Если просто перемножить два элемента в представлении Монтгомери, то получим:

$$(xR) \cdot (yR) = xyR^2 \pmod N.$$

Но нам нужно вычислить $xyR \pmod N$. Следовательно, нам придется еще разделить результат стандартного умножения на R. Поскольку R — степень двойки, мы надеемся, что этот процесс окажется

не очень сложным. Пусть, в общей ситуации, нам нужно вычислить $z = y/R \pmod N$. Сделаем это с помощью редукции Монтгомери. Сначала вычисляется целое число $q = 1/N \pmod R$, которое можно найти не делением, а с помощью двоичного алгоритма Евклида. Затем осуществляем шаги, записанные на псевдокоде на следующей странице.

```
u=(-y*q)%R;
z=(y+u*N)/R;
if (z>=N) { z-=N; }
```

Заметим, что редукция по модулю R, которая происходит в первой строке алгоритма, является легкой операцией: мы вычисляем произведение традиционным способом, а редукцию осуществляем простым обрезанием результата[1]. Действительно, ведь R — степень числа b. Деление на R, происходящее во второй строке, тоже вполне доступно: поскольку $y + u \cdot N = 0 \pmod R$, мы просто сдвигаем наше представление числа на t слов вправо (потому что $R = b^t$).

В качестве примера опять возьмем

$$N = 1\,073\,741\,827 \quad \text{и} \quad b = R = 2^{32} = 4\,294\,967\,296.$$

Вычислим произведение $2 \cdot 3$ с помощью арифметики Монтгомери. Напомним, что

$$2 \longrightarrow 2 \cdot R \pmod N = 1\,073\,741\,803,$$
$$3 \longrightarrow 3 \cdot R \pmod N = 1\,073\,741\,791.$$

Прибегая к стандартному алгоритму умножения, получаем

$$w = x \cdot y = 1\,152\,921\,446\,624\,789\,173 = 2 \cdot 3 \cdot R^2.$$

Теперь нам нужно перевести значение w в представление Монтгомери. Находим

$$w = 1\,152\,921\,446\,624\,789\,173,$$
$$q = (1/N) \pmod R = 1\,789\,569\,707,$$
$$u = -w \cdot q \pmod R = 3\,221\,225\,241,$$
$$z = (w + u \cdot N)/R = 1\,073\,741\,755.$$

Таким образом, результатом произведения x и y в арифметике Монтгомери служит число

$$1\,073\,741\,755.$$

[1] Обратите внимание, что $123456 \pmod{100} = 56$. — *Прим. перев.*

Можно убедиться в корректности ответа, подсчитав,

$$6 \cdot R \pmod N = 1\,073\,741\,755.$$

Итак, мы увидели, что арифметика Монтгомери позволяет нам складывать и умножать остатки от деления на N, не привлекая дорогостоящую операцию деления.

Описанный метод редукции Монтгомери использует два полных умножения. Поэтому умножая два числа в арифметике Монтгомери, мы затратим три умножения. Если сомножители насчитывают по 1024 бит, то результат будет состоять из 2048 знаков. Нам бы хотелось чуть сэкономить и мы можем сделать это.

Предположим, y задан в малословном формате

$$y = (y_0, y_1, \ldots, y_{2t-2}, y_{2t-1}).$$

Тогда наилучший способ осуществления редукции Монтгомери заключается в том, что используется заранее вычисленная величина

$$N' = -1/N \pmod b,$$

которая определяется сравнительно легко. Делаем следующее:

```
z=y;
for (i=0; i<t; i++)
   { u=(zi*N')%b;
     z+=u*N*b;
   }
z/=R;
if (z>=N) { z-=N; }
```

Заметим, что ввиду редукции по модулю b в первой строке цикла, мы можем выполнить начальное умножение, используя алгоритм умножения слов. Второй шаг цикла требует сдвига на одно слово (умножение на b) и одного умножения сло́ва на большое целое число. Следовательно, мы сокращаем число больших промежуточных результатов в редукции Монтгомери.

Можно также чередовать умножение с редукцией, ограничиваясь одним проходом цикла и получая

$$Z = XY/R \pmod N.$$

Так что если $X = xR$ и $Y = yR$, то

$$Z = (xy)R \pmod N.$$

Эта процедура называется умножением Монтгомери и позволяет вычислять произведения в арифметике Монтгомери, не прибегая к большим целыми числам. Перемежающаяся версия представлена «программой»

```
Z=0;
for (i=0; i<t; i++)
   { u=((z0+Xi*Y0)*N')%b;
     Z=(Z+Xi*Y+u*N)/b;
   }
if (Z>=N) { Z-=N; }
```

Хотя сложность умножения Монтгомери оценивается как $O(n^2)$, в отличие от $O(n^{1,58})$ — сложности умножения Карацубы, — предпочтительнее использовать арифметику Монтгомери, поскольку она более эффективна в арифметике остатков.

11.6. Арифметика в конечных полях

Кроме полей вычетов по простому модулю p в криптографии используются поля четной характеристики. Такие поля встречаются, например, в алгоритме *Rijndael* и в некоторых системах, основанных на эллиптических кривых. В *Rijndael* поле настолько мало, что можно использовать поисковые таблицы или специальные микросхемы для реализации основных арифметических действий, поэтому в этом параграфе мы будем разбираться с полями над $\mathbb{F}_2$ большой степени, такими, которые, например, используются в эллиптических кривых. Кроме того, нас будут интересовать только программные реализации операций. На аппаратных средствах операции в полях характеристики 2 могут реализовываться с помощью так называемых оптимальных нормальных базисов, но мы не будем здесь этого касаться.

Напомним, что для задания конечного поля характеристики 2 мы выбираем неприводимый многочлен $f(x)$ с коэффициентами из $\mathbb{F}_2$ степени n и рассматриваем факторкольцо:

$$\mathbb{F}_{2^n} = \mathbb{F}_2[x]/(f(x)),$$

т. е. работаем с двоичными многочленами по модулю $f(x)$. Элементы такого поля обычно представляются строкой бит, изображающей двоичный многочлен. Например, строка битов

101010111

представляет многочлен

$$x^8 + x^6 + x^4 + x^2 + x + 1.$$

Сложение и вычитание элементов поля $\mathbb{F}_{2^n}$ осуществляется простым поразрядным сложением битов строки по модулю 2. Поэтому трудность могут представлять только действия умножения и деления.

Оказывается, деление, хотя и более медленная операция, чем умножение, описывается проще. Так что начнем с него. Чтобы вычислить α/β, где $\alpha, \beta \in \mathbb{F}_{2^n}$, сначала находят β^{-1}, после чего ищут произведение $\alpha \cdot \beta^{-1}$. Таким образом, деление сводится к умножению и вычислению обратного элемента. Один из способов обращения элемента состоит в применении теоремы Лагранжа, которая говорит, что для любого $\beta \neq 0$

$$\beta^{2^n-1} = 1.$$

Следовательно,

$$\beta \cdot \beta^{2^n-2} = 1, \text{ т. е. } \beta^{-1} = \beta^{2^n-2} = \beta^{2(2^{n-1}-1)}.$$

Второй способ вычисления β^{-1} связан с двоичным алгоритмом Евклида. Берем многочлен f, определяющий поле, b, представляющий элемент β, и применяем следующую версию двоичного алгоритма Евклида, где `lsb(b)` — наименьший значащий бит строки b (другими словами, коэффициент при x^0).

```
a=f;
B=0;
D=1;
/* По крайней мере один из a и b теперь имеет свободный член
   на каждом проходе цикла.
*/
   while (a != 0) do
     { while (lsb(a)=0)
       { a=a>>1;
         if (lsb(B)!=0) { B=B^f; }
         B=B>>1; }
       }
     while (lsb(b)=0)
       { b=b>>1;
         if (lsb(D)!=0) { D=D^f; }
         D=D>>1;
       }
/* Сейчас как a так и b имеют свободный член */
```

```
    if (deg(a)>=deg(b))
      { a=(a^b); B=B^D; }
    else
      { b=(a^b); D=D^B; }
    }
вывести D;
```

Обратимся к операции умножения. В отличие от целых чисел по модулю N, где мы применяли специальные методы арифметики Монтгомери, в полях характеристики 2 существует возможность выбора многочлена $f(x)$, обладающего «хорошими свойствами». Любой неприводимый многочлен степени n можно использовать для построения конечного поля $\mathbb{F}_{2^n}$. Так что нам нужно лишь взять наиболее удобный.

Почти всегда многочлен $f(x)$ выбирают среди трехчленов

$$f(x) = x^n + x^k + 1$$

или пятичленов

$$f(x) = x^n + x^{k_3} + x^{k_2} + x^{k_1} + 1.$$

Замечено, что для всех полей, степень которых не превышает 10 000, всегда можно подобрать трехчлен или пятичлен, сильно облегчающий операцию умножения. В таблице 11.1, размещенной в конце этой главы, приводится список всех значений n, лежащих между 2 и 500, с примерами трехчленов или пятичленов, определяющих поле $\mathbb{F}_{2^n}$. Во всех случаях, где можно подобрать трехчлен, мы его вписываем в таблицу. В противном случае представляем соответствующий пятичлен.

Теперь, чтобы перемножить элементы α и β, мы сначала перемножаем представляющие их многочлены и получаем многочлен $\gamma(x)$, степень которого не превышает $2n-2$. После этого нам необходимо вычислить остаток от деления $\gamma(x)$ на многочлен $f(x)$.

Покажем, как это делать, если $f(x)$ — трехчлен, оставив случай пятичлена в качестве упражнения. Записываем

$$\gamma(x) = \gamma_1(x)x^n + \gamma_0(x),$$

где $\deg\gamma_1(x)$, $\deg\gamma_0(x) \leqslant n-1$. Тогда

$$\gamma(x) \pmod{f(x)} = \gamma_0(x) + (x^k + 1)\gamma_1(x).$$

Правую часть этого уравнения можно вычислить с помощью побитовых операций:

$$\delta = \gamma_0 \oplus \gamma_1 \oplus (\gamma_1 \ll k).$$

Степень многочлена δ не может превышать $n-1+k$. Повторим эту процедуру, представив δ в виде

$$\delta(x) = \delta_1(x)x^n + \delta_0(x),$$

где $\deg \delta_0(x) \leqslant n-1$ и $\deg \delta_1(x) \leqslant k-1$. Значит $\gamma(x) \pmod{f(x)}$ равен битовой строке, полученной с помощью формулы:

$$\gamma' = \delta_0 \oplus \delta_1 \oplus (\delta_1 \ll k).$$

Степень γ' — $\max(n-1, 2k-1)$. Поэтому если наш трехчлен $f(x)$ выбран так, что $k < n/2$, то действия, описанные в следующем абзаце, завершат деление с остатком.

Пусть g — многочлен степени $2n-2$, который нам нужно привести по модулю f (т. е. найти остаток от деления g на f), причем оба многочлена представлены строками битов. Тогда делаем такие шаги:

```
g1=g>>n;
g0=g[n-1..0];
g=g0^g1^(g1<<k);
g1=g>>n;
g0=g[n-1..0];
g=g0^g1^(g1<<k);
```

Для завершения рассказа об умножении элементов поля $\mathbb{F}_{2^n}$ нам осталось объяснить, как реализовать умножение двоичных многочленов степени $\leqslant n-1$.

Здесь тоже можно применить наивный алгоритм умножения. Довольно часто для умножения многочленов, степень которых меньше восьми (они представляются одним байтом), применяются поисковые таблицы, т. е. что-то вроде таблицы умножения, а многочлены бóльшей степени умножаются в столбик, как в школьном алгоритме. Сложность такого умножения описывается функцией $O(n^2)$, где n — степень перемножаемых многочленов.

Предположим, что у нас есть таблица умножения всех двоичных многочленов степени < 8. Тогда степень произведений будет меньше 15. Функцию, сопоставляющую паре 8-битовых многочленов a и b их произведение, обозначим через `Mult_Tab(a,b)`. Приведем алгоритм умножения общих n-битовых многочленов, обозначив через `y>>8` сдвиг строки вправо на 8 бит.

```
z=0; i =0;
while (x!=0)
```

```
{ u=y; j=0;
  while (u!=0)
    { w=Mult_Tab(x&255,u&255);
      w=w<<(8*(i+j));
      ans=ans^w;
      u=u>>8; j=j+1;
    }
  x=x>>8; i=i+1;
}
```

Как и при умножении целых чисел, здесь применима техника «разделяй и властвуй», основанная на умножении Карацубы. Ее сложность будет оцениваться как $O(n^{1{,}58})$. Допустим, нам нужно перемножить два многочлена

$$a = a_0 + x^{n/2}a_1 \quad \text{и} \quad b = b_0 + x^{n/2}b_1,$$

где a_0, a_1, b_0, b_1 — многочлены степени меньшей чем $n/2$. Поступаем следующим образом:

$$\begin{aligned} A &= a_0 \cdot b_0, \\ B &= (a_0 + a_1) \cdot (b_0 + b_1), \\ C &= a_1 \cdot b_1. \end{aligned}$$

Произведение $a \cdot b$ получается по формуле:

$$\begin{aligned} Cx^n + (B - A - C)x^{n/2} + A = a_1b_1x^n + (a_1b_0 + a_0b_1)x^{n/2} + a_0b_0 = \\ = (a_0 + x^{n/2}a_1) \cdot (b_0 + x^{n/2}b_1) = a \cdot b. \end{aligned}$$

Умножая a_0 на b_0 и т. д., мы вновь применим умножение Карацубы рекуррентным образом. И как только сведем умножение к вычислению произведений многочленов степени 7 и ниже, воспользуемся таблицей умножения. В отличие от случая целых чисел, здесь умножение Карацубы эффективнее алгоритма умножения в столбик даже для вычисления произведения многочленов достаточно малой степени $n \approx 100$.

Следует отметить, что возведение в квадрат многочленов над полем характеристики 2 вообще практически ничего не стоит. Пусть дан многочлен

$$a = a_0 + a_1x + a_2x^2 + a_3x^3,$$

где a_i равно 0 или 1. Его квадрат получается простым «прореживанием» коэффициентов:

$$a = a_0 + a_1x^2 + a_2x^4 + a_3x^6.$$

Происходит это благодаря формуле, справедливой над любым полем характеристики 2: $(A+B)^2 = A^2 + B^2$. Таким образом, возведение в квадрат над полем характеристики 2 — очень быстрая операция в сравнении с общим умножением.

Краткое содержание главы

- Возведение в степень в арифметике остатков или в любой другой конечной абелевой группе можно осуществить с помощью метода двоичного потенцирования. Часто оконный метод оказывается более эффективным, а для эллиптических кривых можно применять знаковое потенцирование.
- Алгоритм *RSA* можно оптимизировать. В качестве открытой экспоненты нужно выбирать число не только малое, но и обладающее наименьшим весом Хемминга. В процедуре расшифрования нужно использовать информацию о разложении модуля алгоритма N на простые сомножители и китайскую теорему об остатках.
- В процедуре проверки подписи алгоритма *DSA* существует метод одновременного возведения в степень, который оказывается эффективнее способа, при котором каждая из степеней вычисляется отдельно, а потом перемножаются результаты.
- Основные действия в арифметике остатков реализуются с помощью представления Монтгомери. Это позволяет избежать дорогостоящей операции деления, заменяя ее простым сдвигом разрядов. Правда, эффективность здесь достигается благодаря нестандартному представлению чисел.
- Операции в полях характеристики 2 также эффективно реализованы. В этом случае редукция по модулю неприводимого многочлена $f(x)$ может быть упрощена выбором специального трехчлена или пятичлена. Обращение элементов поля на практике осуществляется с помощью адаптированного варианта двоичного алгоритма Евклида. Однако вычисление обратного элемента происходит как правило в 3–10 раз медленнее, чем умножение.

Дополнительная литература

Стандартная ссылка на описание алгоритмов, разобранных в этой главе — второй том книги Кнута. Более легкое введение в них можно найти в книге Бэча и Шаллита, а алгоритмы, не вошедшие в наш

учебник, представлены у Коэна. Первая глава произведения Коэна дает много полезных сведений по разработке стандартного калькулятора, в то время как книга Бэча и Шаллита содержит обширную библиографию и соответствующие комментарии.

E. Bach and S. Shallit. *Algorithmic Number Theory, Volume 1: Efficient Algorithms*. MIT Press, 1996.

H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.

D. Knuth. *The Art of Computing Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 1975.

Контрольные вопросы

11.1.1. Чему равны максимальное и среднее количество умножений в основном алгоритме двоичного потенцирования?

11.1.2. Почему наименьший возможный вес Хемминга шифрующей экспоненты алгоритма *RSA* равен 2?

11.1.3. Что мешает применять алгоритм знакового потенцирования в системе *RSA*?

11.1.4. Почему умножение Карацубы редко используется в системе *RSA*?

11.1.5. Что такое представление Монтгомери и почему его активно используют в алгоритмах типа *RSA*?

11.1.6. Почему в полях характеристики 2 предпочтительнее использовать аналог умножения Карацубы, а не Монтгомери?

Лабораторные работы

11.2.1. (Предполагается знакомство с языком *C* или *Java*.) Сравните алгоритм двоичного потенцирования с тем, что приведен ниже:

```
y=i;
while (d<>0)
  { if ((d&1)!=0)
      { y=(y*x)%n; }
    d>>=1;
    x=(x*x)%n;
  }
```

Почему этот алгоритм более эффективен? Можете ли Вы придумать еще более хороший метод?

11.2.2. Разработайте процедуры, осуществляющие сложение, умножение и деление в полях характеристики 2. Ваши алгоритмы должны работать с полями, чья степень над полем $\mathbb{F}_2$ может быть любой, вплоть до 500.

11.2.3. Напишите программы, умножающие точки эллиптической кривой, определенной либо над полем характеристики 2, либо над полем вычетов по модулю простого числа, предполагая, что элементы полей представляются битовыми строками длиною около 190 знаков. Примеры полей и кривых можно найти в одном из многочисленных криптографических стандартов, доступных в Интернет.

Упражнения

11.3.1. Предположим, что умножение k-битовых элементов кольца вычетов требует k^2 операций. Что быстрее: два возведения в степень 512-разрядного числа с 512-разрядным показателем или одно потенцирование 1024-битового числа с показателем в 1024 бита? Заключите из ответа, что метод расшифрования или генерирования подписи в *RSA*, опирающийся на китайскую теорему об остатках, более эффективен.

11.3.2. Предположим, что для ускорения процесса расшифрования (генерирования подписи) в *RSA* применяется китайская теорема об остатках, т. е. сначала производят вычисления

$$m_p = c^{d \pmod{p-1}} \pmod{p},$$
$$m_q = c^{d \pmod{q-1}} \pmod{q},$$

а потом находят m, исходя из m_p и m_q. Допустим, что атакующий сумел заставить эту программу вычислять m_p неправильно, не вмешиваясь в процесс вычисления m_q, т. е. последнее значение найдено верно. Покажите, что в этом случае атакующий может раскрыть секретный ключ.

11.3.3. Докажите, что две версии редукции Монтгомери, представленные на псевдокоде в тексте главы, действительно приводят к ожидаемому результату. Покажите, что перемежающаяся версия умножения тоже работает корректно.

11.3.4. Покажите, что шифрование и расшифровывание в алгоритме *RSA* можно реализовать за $O(n^3)$ операций, где n — битовая длина модуля N алгоритма.

11.3.5. Покажите, что шифрование в системе Эль-Гамаль требует $2\ln p$ умножений по модулю p. Выведите отсюда, что его битовая сложность равна $O((\ln p)^3)$.

11.3.6. Решив рекуррентные соотношения, приведенные в тексте, покажите, что сложность умножения Карацубы действительно равна $O(n^{1{,}58})$.

11.3.7. Умножение Карацубы многочленов основано на разбиении многочленов на две половины. Разработайте вариант такого умножения, при котором многочлены разбиваются на три части.

Таблица 11.1. Неприводимые трехчлены и пятичлены

n	$k/k_1, k_2, k_3$	n	$k/k_1, k_2, k_3$	n	$k/k_1, k_2, k_3$
2	1	3	1	4	1
5	2	6	1	7	1
8	7,3,2	9	1	10	3
11	2	12	3	13	4,3,1
14	5	15	1	16	5,3,1
17	3	18	3	19	5,2,1
20	3	21	2	22	1
23	5	24	8,3,2	25	3
26	4,3,1	27	5,2,1	28	1
29	2	30	1	31	3
32	7,3,2	33	10	34	7
35	2	36	9	37	6,4,1
38	6,5,1	39	4	40	5,4,3
41	3	42	7	43	6,4,3
44	5	45	4,3,1	46	1
47	5	48	11,5,1	49	9
50	4,3,2	51	6,3,1	52	3
53	6,2,1	54	9	55	7
56	7,4,2	57	4	58	19
59	7,4,2	60	1	61	5,2,1
62	29	63	1	64	11,2,1
65	32	66	3	67	5,2,1
68	33	69	6,5,2	70	37,34,33

Продолжение таб. 11.1.

71	35	72	36,35,33	73	42
74	35	75	35,34,32	76	38,33,32
77	38,33,32	78	41,37,32	79	40,36,32
80	45,39,32	81	35	82	43,35,32
83	39,33,32	84	35	85	35,34,32
86	49,39,32	87	46,34,32	88	45,35,32
89	38	90	35,34,32	91	41,33,32
92	37,33,32	93	35,34,32	94	43,33,32
95	41,33,32	96	57,38,32	97	33
98	63,35,32	99	42,33,32	100	37
101	40,34,32	102	37	103	72
104	43,33,32	105	37	106	73,33,32
107	54,33,32	108	33	109	34,33,32
110	33	111	49	112	73,51,32
113	37,33,32	114	69,33,32	115	53,33,32
116	48,33,32	117	78,33,32	118	33
119	38	120	41,35,32	121	35,34,32
122	39,34,32	123	42,33,32	124	37
125	79,33,32	126	49	127	63
128	55,33,32	129	46	130	61,33,32
131	43,33,32	132	44,33,32	133	46,33,32
134	57	135	39,33,32	136	35,33,32
137	35	138	57,33,32	139	38,33,32
140	45	141	85,35,32	142	71,33,32
143	36,33,32	144	59,33,32	145	52
146	71	147	49	148	61,33,32
149	64,34,32	150	53	151	39
152	35,33,32	153	71,33,32	154	109,33,32
155	62	156	57	157	47,33,32
158	76,33,32	159	34	160	79,33,32
161	39	162	63	163	48,34,32
164	42,33,32	165	35,33,32	166	37
167	35	168	134,33,32	169	34
170	105,35,32	171	125,34,32	172	81
173	71,33,32	174	57	175	57
176	79,37,32	177	88	178	87
179	80,33,32	180	33	181	46,33,32
182	81	183	56	184	121,39,32
185	41	186	79	187	37,33,32
188	46,33,32	189	37,34,32	190	47,33,32

Продолжение таб. 11.1.

191	51	192	147,33,32	193	73
194	87	195	50,34,32	196	33
197	38,33,32	198	65	199	34
200	57,35,32	201	59	202	55
203	68,33,32	204	99	205	94,33,32
206	37,33,32	207	43	208	119,34,32
209	45	210	49,35,32	211	175,33,32
212	105	213	75,33,32	214	73
215	51	216	115,34,32	217	45
218	71	219	54,33,32	220	33
221	63,33,32	222	102,33,32	223	33
224	39,33,32	225	32	226	59,34,32
227	81,33,32	228	113	229	64,35,32
230	50,33,32	231	34	232	191,33,32
233	74	234	103	235	34,33,32
236	50,33,32	237	80,34,32	238	73
239	36	240	177,35,32	241	70
242	95	243	143,34,32	244	111
245	87,33,32	246	62,33,32	247	82
248	155,33,32	249	35	250	103
251	130,33,32	252	33	253	46
254	85,33,32	255	52	256	91,33,32
257	41	258	71	259	113,33,32
260	35	261	89,34,32	262	86,33,32
263	93	264	179,33,32	265	42
266	47	267	42,33,32	268	61
269	207,33,32	270	53	271	58
272	165,35,32	273	53	274	67
275	81,33,32	276	63	277	91,33,32
278	70,33,32	279	38	280	242,33,32
281	93	282	35	283	53,33,32
284	53	285	50,33,32	286	69
287	71	288	111,33,32	289	36
290	81,33,32	291	168,33,32	292	37
293	94,33,32	294	33	295	48
296	87,33,32	297	83	298	61,33,32
299	147,33,32	300	45	301	83,33,32
302	41	303	36,33,32	304	203,33,32
305	102	306	66,33,32	307	46,33,32
308	40,33,32	309	107,33,32	310	93

Продолжение таб. 11.1.

311	78,33,32	312	87,33,32	313	79
314	79,33,32	315	132,33,32	316	63
317	36,34,32	318	45	319	36
320	135,34,32	321	41	322	67
323	56,33,32	324	51	325	46,33,32
326	65,33,32	327	34	328	195,37,32
329	50	330	99	331	172,33,32
332	89	333	43,34,32	334	43,33,32
335	113,33,32	336	267,33,32	337	55
338	86,35,32	339	72,33,32	340	45
341	126,33,32	342	125	343	75
344	135,34,32	345	37	346	63
347	56,33,32	348	103	349	182,34,32
350	53	351	34	352	147,34,32
353	69	354	99	355	43,33,32
356	112,33,32	357	76,34,32	358	57
359	68	360	323,33,32	361	56,33,32
362	63	363	74,33,32	364	67
365	303,33,32	366	38,33,32	367	171
368	283,34,32	369	91	370	139
371	116,33,32	372	111	373	299,33,32
374	42,33,32	375	64	376	227,33,32
377	41	378	43	379	44,33,32
380	47	381	107,34,32	382	81
383	90	384	295,34,32	385	51
386	83	387	162,33,32	388	159
389	275,33,32	390	49	391	37,33,32
392	71,33,32	393	62	394	135
395	301,33,32	396	51	397	161,34,32
398	122,33,32	399	49	400	191,33,32
401	152	402	171	403	79,33,32
404	65	405	182,33,32	406	141
407	71	408	267,33,32	409	87
410	87,33,32	411	122,33,32	412	147
413	199,33,32	414	53	415	102
416	287,38,32	417	107	418	199
419	200,33,32	420	45	421	191,33,32
422	149	423	104,33,32	424	213,34,32
425	42	426	63	427	62,33,32
428	105	429	83,33,32	430	62,33,32

Окончание таб. 11.1.

431	120	432	287,34,32	433	33
434	55,33,32	435	236,33,32	436	165
437	40,34,32	438	65	439	49
440	63,33,32	441	35	442	119,33,32
443	221,33,32	444	81	445	146,33,32
446	105	447	73	448	83,33,32
449	134	450	47	451	406,33,32
452	97,33,32	453	87,33,32	454	128,33,32
455	38	456	67,34,32	457	61
458	203	459	68,33,32	460	61
461	194,35,32	462	73	463	93
464	143,33,32	465	59	466	143,33,32
467	156,33,32	468	33	469	116,34,32
470	149	471	119	472	47,33,32
473	200	474	191	475	134,33,32
476	129	477	150,33,32	478	121
479	104	480	169,35,32	481	138
482	48,35,32	483	288,33,32	484	105
485	267,33,32	486	81	487	94
488	79,33,32	489	83	490	219
491	61,33,32	492	50,33,32	493	266,33,32
494	137	495	76	496	43,33,32
497	78	498	155	499	40,33,32
500	75				

ГЛАВА 12

ПОЛУЧЕНИЕ АУТЕНТИЧНОГО ОТКРЫТОГО КЛЮЧА

Цели главы

- Ввести понятие цифрового сертификата.
- Объяснить понятие *PKI*.
- Исследовать различные подходы, такие как *X509*, *PGP* и *SPKI*.
- Показать, как может работать схема неявных сертификатов.
- Объяснить, как работают криптографические схемы идентификационной информации.

12.1. Общие сведения о цифровых подписях

Применение цифровых подписей гораздо шире тех, которые обычно ставят ручкой на бумаге. Например, они используются для

- контроля доступа к данным,
- подтверждения личности пользователя,
- аутентификации данных,
- подписывания «реальных» документов.

Приложения цифровых подписей имеют дело с различными типами данных, обладают разным сроком службы, могут иметь многообразные формы проставления и проверки.

Например, при межбанковских расчетах подпись ставится на документе, содержащем только номера двух счетов и сумму перевода. При этом подпись должен ставить отправитель платежа, а проверять ее будет компьютер, осуществляющий перечисление. Срок

службы такой подписи ограничен подведением баланса на соответствующих счетах. В частности, подпись утратит свою силу после окончания срока, в течение которого клиент имеет право обратиться в банк с требованием исправить какие-либо допущенные при перечислении ошибки.

Другой пример — механизм идентификации типа запрос–ответ. Здесь для удостоверения своей подлинности пользователь подписывает запрос, выданный устройством. При этом срок службы подписи может быть ограничен несколькими секундами. Пользователь конечно принимает, что вызов происходит случайным образом и не является финансовым требованием. Поэтому для банковских операций и опознавательных паролей для доступа к устройствам разумно использовать ключи разной степени сложности.

В качестве последнего примера рассмотрим подпись на контракте. Отрезок времени, в течение которого она должна иметь силу, может простираться до нескольких лет. Поэтому требования к криптостойкости подписи на долгосрочных юридических документах должны быть несравненно выше, чем к паролям, открывающим доступ к устройству.

Необходимо помнить, что цифровая подпись, в отличие от рукописной,

- *не обязательно* ставится на документах: любой цифровой материал может быть подписан;
- *не переносится на другие документы*: цифровая подпись, в отличие от ручной, меняется от документа к документу;
- *тесно связана с документом*: нельзя изменить документ, не меняя подписи;
- *не ставится человеком*: цифровая подпись никогда не генерируется человеком, за исключением случаев, когда схема подписи очень проста (и поэтому не стойка) или этот человек — математический гений.

Все, для чего предназначены цифровые подписи, — это связывать хранимый в тайне секретный ключ и какие-то цифровые данные.

12.2. Цифровые сертификаты и PKI

Используя криптосистемы с симметричным ключом, мы не заботимся о том, какой ключ кому принадлежит. Неявно предполагается (см., например, главу 6 о протоколах с симметричным ключом и *BAN*-логике), что если Алиса обладает долговременным секретным

ключом k_{ab}, который (по ее мнению) она разделяет с Бобом, то Боб фактически имеет копию того же самого ключа. Такая уверенность достигается распределением долговременных ключей физическими методами, например, с помощью вооруженных курьеров.

В криптографии с открытым ключом проблема в другом. Алиса может обладать открытым ключом, который по ее мнению связан с Бобом, но мы обычно не предполагаем, что Алиса на 100% убеждена в этом. Так происходит потому, что в системах с открытым ключом не прибегают к тайному физическому распределению ключей. В конце концов, криптография с открытым ключом и была разработана для упрощения проблемы управления ключами. Алиса могла получить открытый ключ Боба из его веб-страницы в сети Интернет, но полной уверенности в том, что информация, содержащаяся там, была подлинной, у нее не будет.

Процесс сопоставления открытого ключа физическому лицу или уполномоченному агенту, будь это человек, машина или процесс, называется *привязкой*. Один из способов привязки, общий для многих ситуаций, в которых владелец ключа должен присутствовать лично, заключается в передаче физического объекта, например, интеллектуальной карточки. Владение таким символом и знание любого PIN-кода (пароля), необходимого для отпирания символа, считается достаточным для удостоверения личности. С таким методом связано много проблем, поскольку карточка может сломаться или быть украденной, именно поэтому их защищают PIN-кодами (а в более важных случаях биометрическими данными). Но основная проблема в том, что большинство владельцев ключей не является людьми: это компьютеры, а компьютеры не носят с собой карт. Кроме того, многие протоколы с открытым ключом выполняются через сети, где физическое присутствие владельца (если он — человек) невозможно проверить.

Таким образом, необходимо иметь несколько форм привязки, которые можно было бы использовать в разных ситуациях. Основной инструмент привязки, который сегодня эксплуатируется, называется *цифровым сертификатом*. При этом прибегают к помощи специального посредника *TTP* (от англ. trusted third party), именуемого центром сертификатов (ЦС), который отвечает за *аутентичность* (подлинность) открытого ключа.

Принцип работы ЦС заключается в следующем:

- Каждый из пользователей обладает надежной копией открытого ключа ЦС. Например, он может быть заложен в Ваш ком-

пьютер при покупке, а Вы, «естественно», доверяете продавцу и изготовителю программного обеспечения.

- ЦС ставит цифровую подпись на строке данных

  (Алиса, открытый ключ Алисы)

  Эта строка данных вместе с цифровой подписью и называется цифровым сертификатом. Центр Сертификатов подпишет строку данных только в том случае, если искренне верит, что данный открытый ключ действительно принадлежит Алисе.
- Если теперь Алиса вышлет Вам свой открытый ключ, содержащийся в цифровом сертификате, то у Вас не будет причин сомневаться в его подлинности, поскольку Вы уверены в корректности работы ЦС.

Такие сертификаты связывают имя «Алиса» с открытым ключом. Поэтому их часто называют сертификатами личности. Существуют и другие привязки. С некоторыми из них мы познакомимся позже.

Сертификаты открытых ключей обычно (хотя и не всегда) хранятся в долговременной памяти компьютера и выдаются по мере надобности. Например, большинство браузеров хранит список сертификатов, с которыми они когда-либо встречались. При этом можно не заботиться о защите информации, поскольку сертификаты невозможно изменить ввиду наличия на них цифровой подписи.

Чтобы более четко осознать преимущества сертификатов и ЦС, рассмотрим пример сообщества, в котором нет ЦС. Если ЦС нет, то любой открытый ключ от каждого из партнеров нам необходимо получать каким-то достоверным образом. Например[1],

6a5def....a21 открытый ключ Джима Беана,

7f341a....bff открытый ключ Жана Доуи,

b5f34a....e6d ключ для обновления программ фирмы Майкрософт.

Если же существует какой-то ЦС (назовем его условно Тед), то достаточно удостовериться в подлинности только его открытого ключа. После этого можно получать сколько угодно разных открытых ключей, подписанных центром сертификатов, не заботясь о способе их получения. Например, они могут быть вложены в электронные письма или взяты из Интернета[2]:

---

[1] Строчными буквами записаны ключи, подлинность которых необходимо проверять.

[2] Здесь заглавными буквами обозначены ключи, подлинность которых подтверждена ЦС.

a45efb....c45 абсолютно надежный ключ Теда,

6A5DEF....A21 Тед утверждает, что это «открытый ключ Джима Беана»,

7F341A....BFF Тед утверждает, что это «открытый ключ Жана Доуи»,

B5F34A....E6D Тед утверждает, что это «ключ для обновления программ фирмы Майкрософт».

Если Вы доверяете ключу Теда и уверены в его корректной работе, то можете доверять и всем ключам, полученным с его помощью.

В общей ситуации цифровой сертификат не ограничивается цифровой подписью на отдельной паре

(Алиса, открытый ключ Алисы).

К сертификатам можно добавлять и другую, специфическую в каждом приложении, информацию. Как правило, сертификаты содержат следующие реквизиты:

- имя пользователя;
- открытый ключ пользователя;
- тип ключа: шифрующий или подписывающий;
- название ЦС;
- серийный номер сертификата;
- срок действия сертификата.
- .....

Коммерческие центры сертификатов зачастую подписывают удостоверения Вашего открытого ключа после соответствующей оплаты и проверки сообщенной Вами информации о Вашей личности. Удостоверения, произведенные коммерческими ЦС, как правило обнародуются. Поэтому их часто называют открытыми сертификатами ключей и используют в незащищенных сетях, доступных всем желающим.

ЦС также находят применение в частных фирмах, например в системе дебет/кредит или больших корпорациях. В этих ситуациях пользователи могут предпочесть сохранить конфиденциальность своих сертификатов открытых ключей. Такие удостоверения принято называть частными сертификатами открытых ключей. Но надо иметь в виду, что вид сертификата — открытый он или частный — никак не влияет на безопасность секретного ключа, связанного с открытым ключом из сертификата. Решение о неразглашении сер-

тификата обычно бывает продиктовано соображениями бизнеса, а не безопасности.

Как правило, существует несколько ЦС. Беглое исследование Вашего веб-браузера позволит обнаружить большое число центров сертификатов, которым он доверяет. Поэтому обычной практикой является подписывание одним центром сертификатов открытого ключа другого ЦС и наоборот — процесс, известный как перекрестная (взаимная) сертификация.

Необходимость в перекрестной сертификации возникает, когда функционирует более одного ЦС, т. к. пользователь может не иметь достоверной копии открытого ключа определенного ЦС, нужной, чтобы проверить подлинность открытого ключа, содержащегося в выданном сертификате. Эта проблема решается перекрестной сертификацией, когда открытый ключ одного центра сертификатов подписывается другим ЦС. Пользователь сначала проверяет открытый ключ соответствующего ЦС, а затем ключ клиента, подписанный этим ЦС.

При существовании достаточного количества ЦС можно получать довольно длинные цепочки, с помощью которых проверяется подлинность того или иного открытого ключа, как показано на рис. 12.1. Предположим, что Алиса верит в подлинность открытого ключа некоторого центра сертификатов ЦС (назовем его для определенности корневым), а открытый ключ Боба она получила с подписью другого центра, ЦС2. Открытый ключ ЦС2 Алиса может получить либо вместе с цифровым сертификатом Боба, либо каким-то другим путем. Если открытый ключ ЦС2 будет содержаться в сертификате, подписанном корневым центром сертификатов, то Алиса сможет доверять всем свидетельствам, подписанным ЦС2 и, в частности, открытому ключу Боба.

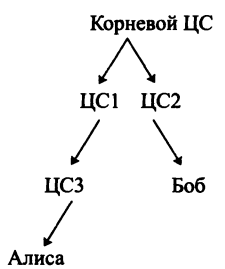


Рис. 12.1. Пример иерархии сертификатов

Обычно функции центра сертификатов состоят из двух частей: первая из них связана с удостоверением личности пользователя, а вторая — с подписыванием открытых ключей. Вторая функция осуществляется собственно ЦС, в то время как первая передается центру регистрации (ЦР). Такое разделение полномочий — удачная идея, поскольку при этом ЦС работает в более безопасной среде, что усиливает защищенность его долгосрочного секретного ключа.

Основная проблема системы центров сертификатов возникает, когда открытый ключ пользователя компрометируется или стано-

вится недостоверным по каким-то другим причинам. Например,

- третья сторона получила информацию о частном ключе пользователя;
- служащий с таким ключом увольняется из компании.

Поскольку скомпрометированному открытому ключу больше нельзя доверять, все подписанные с его помощью сертификаты становятся недействительными и должны быть аннулированы. Но они могут быть распределены между большим числом слабо доступных пользователей, а каждому из них необходимо сообщить об истечении срока действия сертификата. Таким образом, ЦС должен как-то информировать пользователей о том, что все сертификаты, содержащие данный открытый ключ, более не действительны. Этот процесс называется *отзывом сертификатов*.

Один из возможных путей распространения информации о недействительных сертификатах — *список отозванных сертификатов* (СОС). Это сообщение, подписанное ЦС, содержащее серийные номера всех сертификатов, которые были отозваны этим центром, но чей период действия еще не истек. Очевидно, что сертификаты с истекшим сроком действия в этот список включать не обязательно. Читая список, пользователь должен быть уверен, что он достаточно свежий. Поэтому СОС обновляется через регулярные промежутки времени, даже если никаких реальных изменений в списке не произошло. Такая система хорошо работает в больших корпорациях.

В других ситуациях абсолютно непонятно, как СОС можно довести до сведения каждого из пользователей, особенно если существует большое число ЦС, которым доверяет каждый из пользователей.

Подводя итог, скажем, что основными проблемами в криптографии с секретным ключом были распределение ключей и управление ими. Эти проблемы могли решаться передачей ключей по закрытым каналам. В криптографии с открытым ключом названные проблемы замещаются проблемой аутентификации ключей. Другими словами, возникает вопрос: какой ключ кому принадлежит? Здесь ключи должны передаваться по достоверным (или аутентичным) каналам. Цифровые сертификаты как раз и предназначены для обеспечения таких надежных каналов распределения открытых ключей.

Система, состоящая из центров сертификатов и самих сертификатов, обычно называется *инфраструктурой открытых ключей*. Она по существу позволяет «перераспределять доверие»: необходимость веры в подлинность каждого открытого ключа, которым Вы обладаете, заменяется на доверие ЦС и его корректной работе.

Будучи уверенным в корректной работе ЦС, Вы можете полагаться на легальную систему, поддерживаемую ЦС, или на коммерческую структуру, в которой ЦС не имеет деловой заинтересованности. Например, если при подписании Вашего ключа была допущена ошибка, Вы могли бы публично требовать возмещения ущерба.

Закончим этот параграф таким замечанием: на данный момент проблема распределения ключей полностью решена. Чтобы два пользователя могли договориться об общем секретном ключе, они сначала получают достоверный открытый ключ от ЦС. Затем секретный сеансовый ключ вырабатывается, например, с помощью протокола Диффи–Хеллмана с подписями:

$$\begin{array}{lr} \text{Алиса} & \text{Боб} \\ (G^a, Sign_{\text{Алиса}}(G^a)) \longrightarrow & \\ & \longleftarrow (G^b, Sign_{\text{Боб}}(G^b)) \end{array}$$

где $Sign_{\text{Боб}}$ — процедура подписывания, принадлежащая Бобу.

12.3. Пример приложения инфраструктуры открытых ключей

В этом параграфе мы познакомимся с некоторыми действующими системами «перераспределения доверия», опирающимися на цифровые сертификаты

- *PGP*,
- *SSL*,
- *X509* (или *PKIX*),
- *SPKI*.

12.3.1. *PGP*

Программа шифрования электронной почты *PGP* (сокр. от Pretty Good Privacy) использует подход *снизу–вверх* к «перераспределению доверия». *PGP* была разработана с целью обеспечить всех пользователей недорогой системой шифрования и подписывания. Поэтому громоздкая, идущая сверху вниз, глобальная инфраструктура открытых ключей здесь не годилась. Вместо этого система использует то, что обычно называют «сетью доверия».

Управление открытыми ключами осуществляется самими пользователями по принципу снизу–вверх. Каждый из них берет на себя

функции ЦС и подписывает открытые ключи других пользователей. Так Алиса может подписать открытый ключ Боба, а затем Боб может передать этот подписанный «сертификат» Чарли. В этом случае Алиса выступает как ЦС для Боба. Если Чарли доверяет мнению Алисы о достоверности ключей отдельных людей, то он будет уверен, что подписанный ею ключ действительно принадлежит Бобу. Поскольку пользователи постоянно делают такие перекрестные свидетельства, то сеть достоверных открытых ключей растет снизу вверх (отсюда и название).

Сама *PGP* как программа использует шифрование *RSA* для данных небольшого объема, таких, например, как сеансовый ключ. Для передачи больших объемов закрытой информации применяется блочный шифр *IDEA*. Размер блока в этом шифре — 64 бита, а ключа — 128. Шифр эксплуатируется в режиме *CFB*. Цифровые подписи в *PGP* можно ставить с помощью алгоритма *RSA* или *DSA*. При приеме сообщений используется *MD5* или *SHA-1*.

Ключи, которым доверяет отдельный пользователь, объединяются в так называемое кольцо ключей. Это означает, что пользователи самостоятельно контролируют свой локальный запас открытых ключей. Такой способ не исключает централизованного хранения открытых ключей, но говорит о его необязательности.

Отзыв ключей все еще остается нерешенной проблемой программы *PGP*, как впрочем и других подобных систем. Метод «решения» этой проблемы, применяемый в *PGP*, заключается в том, что при компрометации Вашего ключа необходимо сообщить всем друзьям, что следует выбросить это ключ из их колец ключей. Оповещенные Вами должны передать информацию дальше и т. д.

12.3.2. Протокол защищенных сокетов

В то время как проект *PGP* был движим альтруистической идеей обеспечить защиту информации для широких масс, протокол защищенных сокетов или *SSL* (Secure Socket Layer) был разработан в коммерческих целях, а именно, для обеспечения безопасности Интернет-магазинов и продаж по сети. По существу *SSL* усиливает надежность *TCP*[1]. Он обеспечивает безопасность данных и позволяет нормально работать различным протоколам, таким как, например, *HTTP*, *FTP*, *TELNET* и т. д.

[1] Сокр. от Transmission Control Protocol протокол управления передачей (основной протокол транспортного и сеансового уровней в наборе протоколов Internet, обеспечивающий надежные ориентированные на соединения полнодуплексные потоки) — *Прим. перев.*

Первоначальная его цель состояла в обеспечении безопасности канала, по которому передаются подробности кредитных карточек или паролей. Весь обмен сообщениями, кроме начальных, необходимых для установления связи, зашифрован.

Сервер, обеспечивающий передачу информации, а именно, вебсайт или основной компьютер в сеансе *Telnet*, всегда аутентифицирован для удобства клиента. Иногда может быть аутентифицирован и клиент, но это редко делается при основных коммерческих сделках по сети Интернет.

Как и в *PGP*, шифрование большого потока данных в *SSL* осуществляется с помощью блочного или поточного шифра (обычно *DES*, или алгоритма семейства *RC*). Выбор шифра происходит на начальной стадии установления связи. Сеансовый ключ, используемый в этом обмене информацией, выбирается с помощью стандартных протоколов, таких как протокол Диффи–Хеллмана или основной протокол *RSA* передачи ключей.

Сервер аутентифицирутся, снабжая клиента сертификатом *X509* своего открытого ключа. Этот сертификат (для сделок по Интернету) подписывается глобальным ЦС, чей открытый ключ находится на веб-браузере клиента. Для безопасности сеанса *Telnet* (который часто называют *SSH* по имени программы, его осуществляющей) сертификат сервера обычно подписывает сам основной компьютер.

Далее приведем упрощенную версию операций *SSL*.

- Клиент устанавливает связь с сервером через специальный порт, что является сигналом о защищенном сеансе.
- Сервер высылает клиенту сертифицированный открытый ключ.
- Клиент проверяет сертификат и решает, верить ли этому открытому ключу.
- Клиент выбирает случайный секретный ключ.
- Клиент шифрует выбранный секретный ключ с помощью открытого ключа сервера и пересылает результат серверу.
- К этому моменту клиент и сервер обладают общим сеансовым ключом.
- Сервер подтверждает клиенту свою подлинность, отвечая на его запрос с помощью общего с ним сеансового ключа.

Определение сеансового ключа может быть дорогостоящей операцией как для сервера, так и для клиента, особенно когда данные входят в какие-нибудь крупные пакеты, например, при оформлении сделок через Интернет или при использовании удаленного доступа

к компьютеру. Поэтому создано некоторое усовершенствование, позволяющее повторное использование сеансовых ключей. Клиент может предложить использовать ключ предыдущего сеанса, а сервер волен либо согласиться на это, либо потребовать создания нового. Во избежание проблем, это усовершенствование ограничено двумя правилами. Во-первых, сеансовый ключ имеет строго ограниченное время жизни, а во-вторых, любая фатальная ошибка в любой части протокола приводит к немедленной дискредитации сеансового ключа и к созданию нового. При установлении связи в *SSL* достигается соглашение между клиентом и сервером о шифрующем алгоритме, который будет использован при передаче основного потока данных. Обычно он выбирается из *RC4*, *RS5*, *DES* или утроенного *DES*.

12.3.3. Сертификаты *X509*

При обсуждении *SSL* мы упоминали об использовании сервером сертификата *X509*. Это стандарт, определяющий структуру сертификатов открытых ключей. В настоящий момент *X509* — наиболее широко эксплуатируемый стандарт сертификатов. ЦС присваивает уникальное имя каждому пользователю и выдает подписанный сертификат. Именем часто служит *URL* или электронный адрес (email). При использовании последнего могут происходить различные казусы. Дело в том, что большинство пользователей имеют различные версии одного и того же email'а. Например, Вы посылаете письмо с Вашим сертификатом на свой email:

N.P.Smart@some.where.com,

а Ваша почтовая программа реально посылает эту информацию по такому:

Nigel.Smart@some.where.com.

Поэтому, даже если с Вашей точки зрения оба адреса эквивалентны, почтовая программа получателя может счесть подпись не действительной.

Различные ЦС объединены друг с другом в виде дерева, в котором каждый ЦС выдает сертификат для стоящего ниже (в этом дереве). Кроме того, возможны и перекрестные свидетельства между ветвями дерева. Сертификаты *X509* определены в стандартах на языке *ASN.1*[2]. При первом знакомстве они могут показаться доволь-

[2] *ASN.1* — сокр. от Abstract Syntax Notation One — абстрактная синтаксическая нотация версии 1. Это язык, используемый в рамках протоколов взаимодействия открытых систем для описания абстрактных синтаксических структур. — *Прим. перев.*

но сложными, а обработка всех возможных опций часто заканчивается невероятным сообщением: «раздувание кодов» ('code bloat').

Структура базисного сертификата *X509* очень проста, но сильно усложняется в любом разумном приложении. Происходит так потому, что в современных приложениях *X509* возникает необходимость вставлять внутрь сертификатов дополнительную информацию, обеспечивающую авторизацию и другие возможности. Тем не менее, следующие записи всегда присутствуют в сертификате:

- Номер версии стандарта *X509*, которому соответствует данный сертификат.
- Серийный номер сертификата.
- Идентификатор подписывающего алгоритма ЦС, что дает информацию об алгоритме и его параметрах домена, если соответствующий ЦС имеет возможность использовать разные алгоритмы и параметры домена.
- Имя изготовителя сертификата, т. е. название ЦС.
- Срок действия сертификата в форме «не раньше – не позже».
- Имя субъекта, т. е. того, чей открытый ключ подписывается. Именем может служить как email-адрес, так и имя, используемое в домене.
- Открытый ключ субъекта, что включает в себя название используемого алгоритма, все необходимые параметры домена и фактическое значение открытого ключа.
- Подпись ЦС на открытом ключе субъекта и всех сопутствующих данных, например, имени субъекта.

12.3.4. *SPKI*

Чтобы устранить некоторые недостатки, присущие стандарту *X509*, был предложен другой тип сертификатов, *SPKI* (сокр. от Simple Public Key Infrastructure — простая инфраструктура открытых ключей). Эта система призвана решать вопросы авторизации так же хорошо, как и проблемы идентификации. Кроме того, в ней предусмотрена возможность делегирования полномочий. Такие возможности могут оказаться более подходящими для сделок по Интернету. Например, когда отдельные менеджеры уходят в отпуск, они могут делегировать свои полномочия относительно определенных действий своим подчиненным.

SPKI не предполагает наличие глобальной иерархической структуры центров сертификатов, что имеет место в случае стандарта

X509. Здесь используется более детализированный подход, похожий на *PGP*. Однако в настоящее время *SPKI* не находит широкого применения, поскольку продавцы инфраструктуры открытых ключей интенсивно инвестируют стандарт *X509* и, вероятно, не спешат переключаться на новую систему (кроме того, программное обеспечение персональных компьютеров, например, Ваш веб-браузер, тоже должно было бы претерпеть существенные изменения при переходе к новому стандарту).

Для написания сертификатов вместо языка *ASN.1 SPKI* употребляет S-выражения. Это *LISP*-подобные структуры[3], очень простые в использовании и описании. Кроме того, S-выражения легки для понимания, в отличие от сертификатов *X509*, которые могут разобрать только компьютеры. Предусмотрен также удобный интерфейс для облегченного восприятия S-выражений.

Каждый *SPKI*-сертификат содержит открытые ключи (или их хэш-значения) создающего сертификат центра и объекта сертифицирования. Никакие имена в него не входят, поскольку авторы *SPKI* полагают, что только ключ имеет значение, но не имя. Кроме всего прочего, именно ключ используется для подписания документов и т. п. Перенос акцента на ключи означает, что в этой системе большое внимание уделяется функциональности. Разработаны два типа *SPKI*-сертификатов: один для аутентификации ключей, а другой — для их авторизации. Они представлены как кортеж из 4 или 5 объектов, о котором сейчас пойдет речь.

12.3.4.1. Кортеж SPKI из четырех объектов. Для аутентифицирующего сертификата и привязки ключа к имени, как это делает *X509*, *SPKI* использует 4-объектную структуру. Это рабочее абстрактное название, отражающее четыре составные части сертификата:

(Создатель, Имя, Объект, Период действия).

На практике каждый сертификат состоит фактически из следующих пяти полей:

- открытый ключ создателя;
- имя субъекта;
- открытый ключ субъекта;
- период действия;

[3]LISP — сокр. от list processing language — язык обработки списков Лисп (язык программирования). — *Прим. перев.*

- подпись создателя сертификата на тройке (Имя, Объект, Период действия).

Любой способен сделать такой сертификат и, следовательно, стать ЦС.

12.3.4.2. Кортеж SPKI из пяти объектов. Такая система применяется для авторизации ключей. Опять-таки, это условное название, обозначающее 5 составных разделов сертификата:

(Создатель, Объект, Делегирование, Авторизация, Период действия).

В реальной жизни сертификат насчитывает шесть полей:

- открытый ключ создателя;
- открытый ключ субъекта;
- делегирование, т. е. знак «Да» или «Нет», означающий, может ли субъект делегировать полномочия или нет;
- авторизация: что, собственно, может делегировать субъект;
- период действия авторизации;
- подпись создателя сертификата на четверке (Объект, Делегирование, Авторизация, Период действия).

Можно комбинировать сертификат авторизации с сертификатом аутентифицирования для получения контрольного следа. В этом возникает потребность, поскольку сертификат авторизации лишь позволяет совершать какие-либо действия с ключом, но не сообщает, кто ключом владеет. Для привязки ключа к имени надо использовать сертификат аутентифицирования.

После проверки комбинации сертификатов процедура 5-объектного *SPKI* считается выполненной. Комбинирование сертификатов осуществляется по следующему правилу редукции:

$$(I_1, S_1, D_1, A_1, V_1) + (I_2, S_2, D_2, A_2, V_2) = (I_1, S_2, D_2, A_1 \cap A_2, V_1 \cap V_2),$$

где I_i — создатель, S_i — объект, D_2 — делегирование, A_i — авторизация, V_i — срок действия. Равенство верно, только если

$$S_1 = D_2 \quad \text{и} \quad D_1 = \text{Да}.$$

Это означает, что два сертификата, сцепленные вместе, можно интерпретировать как третий. Последний кортеж пяти объектов фактически не является сертификатом, а лишь представляет сцепку первых двух.

В качестве примера покажем, что комбинация двух кортежей эквивалентна делегированию полномочий. Предположим, первый сертификат состоит из реквизитов, выписанных на следующей странице.

- I_1 = Алиса;
- S_1 = Боб;
- D_1 = Да;
- A_1 = снять £100 со счета Алисы;
- V_1 = неограничено.

Этим сертификатом Алиса разрешает Бобу снять со своего счета £100 и позволяет перепоручить это действие любому по его выбору.

Теперь рассмотрим другой 5-объектный кортеж:

- I_2 = Боб;
- S_2 = Чарли;
- D_2 = Нет;
- A_2 = снять сумму от £50 до £200 со счета Алисы;
- V_2 = до завтрашнего утра.

Этим Боб поручает Чарли снять сумму между £50 и £200 со счета Алисы, причем это должно произойти до завтрашнего утра.

Сцепим эти два сертификата, руководствуясь правилом редукции, и получим новый 5-объектный набор:

- I_3 = Алиса;
- S_3 = Чарли;
- D_3 = Нет;
- A_3 = снять сумму от £50 до £200 со счета Алисы;
- V_3 = до завтрашнего утра.

Так как Алиса разрешила Бобу делегировать полномочия, то получилось, что она фактически поручает Чарли снять деньги с ее счета до завтрашнего утра.

12.4. Другие приложения третьей доверенной стороны

В некоторых приложениях возникает необходимость в большом сроке действия подписей. Аннулирование открытого ключа лишает законной силы все цифровые подписи, созданные с помощью этого ключа, даже в том случае, если подписи были созданы достаточно давно. В этом состоит главная проблема подписей, которые стоят на долгосрочных документах: завещаниях, страховых полисах, закладных и т. д. По существу нам нужны способы доказательств того факта, что подпись была сделана еще до аннулирования ключа и все

еще имеет силу. Такая задача подводит нас к понятию *временно́й маркировки*.

Временна́я маркировка означает, что доверенное лицо добавляет к подписанному сообщению вставку даты и времени, и подписывает всю совокупность с помощью своего собственного секретного ключа. Это свидетельствует о времени проставления основной подписи (подобно нотариальной службе при стандартном страховании жизни). Однако при этом возникает условие: открытый ключ, служащий для временно́й маркировки, никогда не может быть аннулирован. Альтернативой временно́й маркировке служит использование безопасного архива подписанных документов.

Другое приложение доверенной третьей стороны связано с проблемой хранения действительно секретных ключей, предназначенных для шифрования.

- Если секретный ключ будет потерян или забыт, то Вы потеряете все зашифрованные данные.
- Если хранитель ключа увольняется из компании или его устраняют, то у компании возникает необходимость получить доступ к зашифрованным данным без участия хранителя.
- Если пользователь совершил преступление, то органы правопорядка должны иметь возможность изучить всю его зашифрованную информацию.

Одно из решений описанных проблем состоит в том, чтобы отдать кому-то на хранение копию Вашего секретного ключа на случай его потери или другого неблагоприятного стечения обстоятельств. С другой стороны, передача ключа какому-то постороннему лицу (в том числе и государственным органам) — крайне опасное мероприятие.

Выход из положения — *условное депонирование ключей*, осуществляющееся посредством схемы разделения секрета. Здесь секретный ключ разбивается на части, каждую из которых можно проверить и убедиться в ее корректности. Части раздаются доверенным лицам. Как только возникает необходимость в восстановлении ключа, некоторые из доверенных лиц могут собраться вместе и, используя свои части, сделать это. Привлечение третьих лиц для процедуры условного депонирования ключей — пример использования третьей доверенный стороны, поскольку Вы действительно должны им доверять. Фактически, требование доверия настолько высоко, что такое решение было главным предметом споров в криптографии и правительственных кругах в прошлом. Распределение

частей ключа между доверенными лицами, привлеченными к условному депонированию, можно сделать, используя прием разделения секрета, который был описан в главе 6.

12.5. Неявные сертификаты

Одна из проблем, связанных с сертификатами, состоит в том, что они могут быть довольно большими. Каждый сертификат должен содержать по крайней мере как открытый ключ пользователя, так и подпись центра сертификатов на этом ключе. Это может приводить к большим размерам сертификатов, как показано на следующей таблице.

Таблица 12.1. Размеры ключей и подписей в сертификатах

	RSA	*DSA*	*EC-DSA*
Ключ пользователя	1024	1024	160
Подпись ЦС	1024	320	320

Данные таблицы означают, что в *RSA* используется 1024-битовый модуль, в *DSA* применяется 1024-битовое простое число p, а в *EC-DSA* 160-битовая кривая. Следовательно, например, если ЦС для подписи ключа в 1024-битовом *DSA* прибегает к 1024-битовому алгоритму *RSA*, то общий размер сертификата должен состоять по крайней мере из 2048 знаков. Возникает интересный вопрос: можно ли уменьшить размер сертификатов?

Неявные сертификаты позволяют этого добиться. Они выглядят как $X|Y$, где X — данные, связанные с открытым ключом, а Y — неявный сертификат для X. Опираясь на Y, нам необходимо уметь восстанавливать открытый ключ, ассоциированный с данными X, и доказывать производство сертификата определенным ЦС. В системе, основанной на *DSA* или *EC-DSA*, которую мы сейчас представим, размер Y будет равен 1024 или 160 битов, соответственно. Таким образом, размер сертификата уменьшается до размера ключа, который необходимо сертифицировать.

12.5.1. Описание системы

ЦС выбирает открытую группу A известного порядка N и элемент $P \in A$. После этого он берет долговременный секретный ключ c и вычисляет открытый ключ $Q = P^c$. Открытый ключ должен быть известен всем пользователям.

12.5.2. Запрос сертификата

Предположим, Алиса хочет получить сертификат и открытый ключ, связанный с информацией ID, которая может быть, например, ее именем. Алиса вычисляет эфемерный секретный ключ t, соответствующий открытый ключ $R = P^t$ и посылает R вместе с ID центру сертификатов.

12.5.3. Обработка запроса

ЦС проверяет, что он получил ID именно от Алисы, выбирает другое случайное число k и вычисляет

$$G = P^k R = P^k P^t = P^{k+t}.$$

Затем он находит

$$S = c \cdot h(ID||G) + k \pmod N$$

и отсылает Алисе пару (G, S). Неявным сертификатом служит пара (ID, G). Теперь нужно убедиться, что

- Алиса сможет восстановить законную ключевую пару;
- любой другой пользователь сможет извлечь открытый ключ Алисы из этого сертификата.

12.5.4. Действия Алисы

Алиса обладает следующей информацией:

$$t,\ S,\ R = P^t.$$

Отсюда она может узнать свой секретный ключ

$$a = t + S \pmod N.$$

Заметим, что секретный ключ Алисы известен только ей, но не ЦС. Кроме того, у Алисы есть еще случайно выбранное число t, а у центра сертификатов — k. Открытый ключ Алисы:

$$P^a = P^{t+S} = P^t P^S = R \cdot P^S.$$

12.5.5. Действия пользователя

Поскольку информация о S и R открыта, пользователь, скажем Боб, может узнать открытый ключ Алисы, вычисляя $R \cdot P^S$. Но это действие ничего не скажет о связи между ЦС, открытым ключом Алисы и информацией ID. Вместо этого Боб восстанавливает открытый

ключ из неявного сертификата (ID, G) и открытого ключа Q центра сертификатов:

$$P^a = Q^{h(ID||G)}G.$$

Как только Боб видит применение ключа Алисы, например, при проверке ее подписи, он абсолютно уверен, что этот ключ должен быть создан ЦС, поскольку в противном случае подпись Алисы не выдержала бы проверки.

С такой системой связано много проблем. Именно поэтому она не находит широкого применения в жизни. Например,

- Что делать, если ключ ЦС будет скомпрометирован? Обычно Вы выбираете новый ключ ЦС и создаете новые сертификаты ключей пользователей. Но здесь это невозможно сделать, поскольку открытые ключи пользователей выбираются в диалоговом режиме в процессе создания сертификата.
- Неявные сертификаты требуют, чтобы как ЦС, так и пользователь работали на одном уровне секретности. Это обычно считается не лучшим требованием. Как правило, ЦС работает на более высоком уровне засекреченности (скажем 2048-битовом *DSA*), чем пользователь (1024-битовый *DSA*).

Однако для устройств с ограниченной пропускной способностью неявные сертификаты могут быть подходящей альтернативой традиционных сертификатов, если последние не доступны.

12.6. Криптография идентификационной информации

Другой способ удостоверения открытых ключей, не прибегающий к сертификатам, состоит в использовании системы, посредством которой ключ пользователя получается из его идентификатора, например, имени. Система называется *схемой шифрования с идентификационной информацией* или схемой подписи с идентификационной информацией. Такие системы все еще нуждаются в доверенном третьем лице для первоначального установления подлинности пользователя, но здесь уже не нужно хранить и передавать сертификаты.

Первая схема такого типа подписи разработана Шамиром в 1984 г, хотя вплоть до 2001 г. считалось, что авторами схемы подписи с идентификационной информацией являются Бонех и Франклин. Мы опишем лишь оригинальную схему подписи Шамира, которая основывается на проблеме *RSA*.

Сначала третья доверенная сторона (TTP) вычисляет модуль RSA, т. е. число N, храня в секрете два его простых делителя. Она публикует открытую экспоненту E, оставляя в тайне соответствующую расшифровывающую экспоненту d. Кроме того, фиксируется определенное отображение

$$I : \{0,1\}^* \longrightarrow (\mathbb{Z}/N\mathbb{Z})^*,$$

которое переводит строку битов в элемент $(\mathbb{Z}/N\mathbb{Z})^*$. Отображение I может быть реализовано хэш-функцией.

Предположим, что Алиса намерена получить секретный ключ g, соответствующий ее имени «Алиса». Тогда TTP вычисляет этот ключ, используя уравнение

$$g = I(\text{Алиса})^d \pmod N.$$

Чтобы подписать сообщение M, Алиса генерирует пару (T, S) с помощью равенств

$$T = r^E \pmod N, \qquad S = g \cdot r^{h(M||T)} \pmod N,$$

где r — случайное целое число, а h — хэш-функция.

Для проверки подписи (T, S) на сообщении M другому пользователю необходима открытая информация о TTP и идентификатор Алисы. Если он обладает такими данными, то убеждается в корректности следующих равенств по модулю N:

$$I(\text{Алиса}) \cdot T^{h(M||T)} = g^E \cdot r^{E \cdot h(M||T)} = \left(g \cdot r^{h(M||T)}\right)^E = S^E.$$

Краткое содержание главы

- Цифровые сертификаты дают возможность связать открытый ключ с другой информацией, например, с идентификатором пользователя.
- Эта связка, в свою очередь, позволяет решить проблему распределения аутентичных открытых ключей.
- Для решения задачи распределения аутентичных ключей были предложены различные системы инфраструктуры открытых ключей. Все они имеют свои достоинства и недостатки.
- *PGP* и *SPKI* работают по принципу снизу–вверх, в то время как *X509* — сверху–вниз. В системе *SPKI* предусмотрена возможность делегирования полномочий.

- Третья доверенная сторона привлекается как для временнóй маркировки подписи, так и для условного депонирования ключей.
- Неявные сертификаты предназначены для сокращения объема удостоверений по сравнению со стандартным сертифицированием, однако они обладают большим числом недостатков.
- Криптография идентификационной информации помогает установить подлинность открытого ключа пользователя, используя его идентификатор как некий открытый ключ, но при этом не исключается участие третьей доверенной стороны.

Дополнительная литература

Хороший обзор инфраструктуры открытых ключей можно найти у Адамса и Ллойда. За дальнейшей информацией о *PGP* и *SSL* стоит обратиться к книгам Гарфинкеля и Рескорла.

C. Adams and S. Lloyd. *Understanding Public-Key Infrastructure: Concepts, Standards and Deployment Considerations*. New Riders Publishing, 1999.

S. Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1994.

E. Rescorla. *SSL and TLS: Design and Building Secure Systems*. Addison-Wesley, 2000.

Контрольные вопросы

12.1.1. Что предлагает центр сертификатов?

12.1.2. Почему в сертификат обычно включают срок его действия?

12.1.3. Что такое перекрестное сертифицирование и зачем оно нужно?

12.1.4. Объясните, чем отличаются *PGP*, *X509* и *SPKI*.

12.1.5. Для чего обычно применяется *SSL*?

ГЛАВА 13

ПРОТОКОЛЫ

Цели главы

- Представить схемы обязательств.
- Рассказать о доказательствах с нулевым разглашением.
- Разобрать протокол голосования.

13.1. Введение

В этой главе мы исследуем криптографические протоколы, позволяющие обеспечить высокий уровень практического обслуживания. Фактически, мы сфокусируем внимание на протоколах для

- схем обязательств,
- доказательств с нулевым разглашением.

Поскольку существует большое количество литературы, посвященной такого сорта протоколам, мы не будем углубляться в теорию, а займемся ее приложением к реальным задачам. Для иллюстрации работы обсуждаемых протоколов написан последний параграф этой главы, в котором рассказано о системе электронного голосования.

13.2. Схемы обязательств

Предположим, Алиса хочет поиграть с Бобом по телефону в игру «камень, ножницы, бумага». Идея игры состоит в том, что Алиса с Бобом одновременно выбирают по одному предмету из множества {камень, ножницы, бумага}. Исход выбора, т. е. кто при этом побеждает, определяется по следующему правилу:

- Бумага может обернуть камень. Поэтому, если Алиса выбирает бумагу, а Боб — камень, то побеждает Алиса.
- Камень тупит ножницы. Значит, Алиса, выбрав камень, опять одержит победу, если Боб при этом возьмет ножницы.
- Ножницы режут бумагу. Стало быть, как только Алиса назовет ножницы, а Боб — бумагу, победа опять достанется ей.

Если же оба игрока выберут одинаковые предметы, раунд закончится с ничейным результатом. Таким образом, если игра проводится по телефону, то невозможно добиться одновременного оглашения предмета. Кто-то должен первым озвучить свой выбор, предоставляя тем самым полную победу второму игроку.

Один из способов преодоления возникающей проблемы состоит в том, что начинающий игру, скажем Алиса, передает свой предмет в виде «затемненного обязательства», которое соперник не сможет прочесть до определенного момента. После того, как Боб назовет свой предмет, Алиса открывает обязательство. При этом важно, чтобы второй из играющих смог убедиться в неизменности обязательства в период между передачей и открытием. Такая система называется *схемой обязательств*. Наиболее простой способ ее реализации состоит в использовании хэш-функции:

$$A \longrightarrow B: H_A = H(r_A || \text{бумага}),$$
$$B \longrightarrow A: \text{НОЖНИЦЫ},$$
$$A \longrightarrow B: r_A, \text{бумага}$$

Бобу нужно проверить, что хэш-значение H_A, переданное Алисой, совпадает с $H(r_A||\text{бумага})$. При положительном результате проверки, Боб будет уверен, что Алиса не жульничала. В этом раунде, конечно, Алиса проиграла, поскольку ножницы режут бумагу.

Посмотрим на перспективы Алисы в этом обмене сообщениями. Сначала Алиса передает свой выбор «бумага», посылая Бобу хэш-значение H_A. При этом Боб не в состоянии определить скрытое содержание обязательства, поскольку он не знает случайного значения r_A и не может обратить хэш-функцию. Свойство схемы обязательств, обеспечивающее невозможность прочтения содержащейся в сообщении информации, называется *сокрытием*.

Далее Боб посылает Алисе слово «ножницы». Та, получив его послание, уже знает о своем поражении, но не может сжульничать, поскольку у нее нет возможности заменить r_A на какое-нибудь другое r'_A, удовлетворяющее условию:

$$H(r_A||\text{бумага}) = H(r'_A||\text{камень}).$$

Действительно, если ей удастся найти такое r'_A, то у соответствующей хэш-функции будут повторяющиеся значения. А мы верим в то, что функция, привлеченная к схеме, защищена от повторений. Фактически, здесь достаточно требовать, чтобы хэш-функция была защищена от вторых прообразов. Свойство схемы обязательств,

благодаря которому Алиса не способна изменить содержание переданного послания, называется *связывающим*.

Изучим скрывающее и связывающее свойства более внимательно. Напомним, что шифрующая функция обладает теоретико-информационной стойкостью, если нападающий, обладая неограниченными вычислительными возможностями, не сможет ее взломать. С другой стороны, криптосистема называется вычислительно защищенной, если ее нельзя вскрыть за полиномиальное время. Аналогично и схемы обязательств можно разделить по степени защищенности, хотя здесь нужно учитывать два свойства: скрывающее и связывающее.

Определение 13.1. Говорят, что схема обязательств

- *теоретико-информационно связывающая*, если передающий обязательство не может изменить его значение, даже обладая неограниченными вычислительными возможностями;
- *вычислительно связывающая*, если отправитель может изменить значение обязательства, но только в результате огромного количества вычислений;
- *теоретико-информационно скрывающая*, если получатель не способен определить содержание обязательства до стадии раскрытия вне зависимости от его вычислительных возможностей;
- *вычислительно скрывающая*, если получатель может узнать содержимое обязательства до его раскрытия, но в результате привлечения неоправданно большого, хоть и конечного, количества вычислительных ресурсов.

Непосредственно из определения вытекает следующая лемма.

Лемма 13.2. Схема обязательств $H(r||c)$ со случайным значением r, обязательством c и криптографической хэш-функцией H в лучшем случае является

- вычислительно связывающей,
- вычислительно скрывающей.

Доказательство. Все криптографические хэш-функции, с которыми мы встречались, только лишь вычислительно защищены от восстановления прообразов и вторых прообразов.

Связывающее свойство гарантировано только защищенностью подлежащей хэш-функции от вторых прообразов. Следовательно, это свойство защищено лишь в вычислительном отношении.

Аналогично, поскольку скрывающее свойство обеспечивается защищенностью хэш-функции от восстановления прообразов, то и оно защищено только по вычислениям. ■

Возникает вопрос: можно ли разработать схему обязательств, оба свойства в которой: и скрывающее, и связывающее, — будут обладать теоретико-информационной стойкостью? Это представляется трудной, если не невозможной задачей. Однако можно предложить две изящных схемы, скрывающее или связывающее свойство в которых будет обладать теоретико-информационной стойкостью.

Пусть A — конечная абелева группа простого порядка Q и образующей G, B — элемент этой группы, чей дискретный логарифм по основанию G не известен ни одному из пользователей системы. Последнее свойство достаточно просто обеспечить. Рассмотрим, например, мультипликативную группу конечного поля $\mathbb{F}_P^*$, выберем Q среди делителей числа $P-1$ и построим G методом, аналогичным которому можно отыскать и B:

- возьмем произвольное $R \in \mathbb{Z}$,
- вычислим $F = H(R) \in \mathbb{F}_P^*$ для некоторой криптографической хэш-функции H,
- положим $G = F^{\frac{P-1}{Q}} \pmod P$. Если $G = 1$, то вернемся к первому шагу. Если $G \neq 1$, то выведем пару (R, G).

Таким способом мы получим случайный элемент группы $\mathbb{F}_P^*$ порядка Q, причем эта случайность гарантируется случайным выбором R.

По данной паре (G, B) мы построим две схемы: $D(x)$ и $D_a(x)$, передающие целое число x по модулю Q.

$$D(x) = G^x, \qquad D_a(x) = B^x G^a,$$

где a — случайное целое число по модулю Q. Величина a называется *затемняющей*, поскольку она скрывает переданное число x даже от нападающего с неограниченными вычислительными возможностями. Чтобы раскрыть обязательство, пользователь открывает значение x в первой схеме и пару (a, x) — во второй.

Лемма 13.3. Схема обязательств $D(x)$ самое большее теоретико-информационно связывающая и вычислительно скрывающая.

Доказательство. Пусть Алиса опубликовала $C = D(x) = G^x$, и намерена изменить элемент из $\mathbb{Z}/Q\mathbb{Z}$, который передала этим обязательством. Увы! вне зависимости от вычислительных возможностей Алисы существует только один элемент в $\mathbb{Z}/Q\mathbb{Z}$, являющийся дискретным логарифмом от C по основанию G, а именно x. Таким

образом, эта система несомненно теоретико-информационно связывающая.

Теперь допустим, что получатель обязательства желает определить скрытый элемент. Все, что нужно для этого сделать, — это найти дискретный логарифм по основанию G. Поэтому схема всего лишь вычислительно скрывающая. ■

Лемма 13.4. Схема обязательств $D_a(x)$ — вычислительно связывающая (если отправитель не знает дискретного логарифма B по основанию G), но теоретико-информационно скрывающая.

Доказательство. Предположим, что Алиса, передав обязательство $M = D_a(x) = B^x G^a$, хочет изменить x на y. Для этого ей достаточно найти

$$F = \frac{M}{B^y},$$

после чего она может вычислить дискретный логарифм $a' = \log_G F$ и утверждать, что она посылала пару (a', y), а не (a, x). Отсюда следует, что схема вычислительно связывающая.

Допустим, получатель собирается восстановить переданный ему x, не дожидаясь стадии раскрытия. Поскольку для данного значения M и каждого x существует лишь одно a, при котором $M = B^x G^a$, то даже обладая неограниченными вычислительными возможностями, получатель не в состоянии восстановить x из M. ■

Закончим параграф замечанием, что обе представленные выше схемы обязательств, основанные на дискретном логарифмировании, обладают свойством гомоморфности:

$$\begin{aligned} D(x_1) \cdot D(x_2) &= G^{x_1} \cdot G^{x_2} = G^{x_1+x_2} = D(x_1 + x_2), \\ D_{a_1}(x_1) \cdot D_{a_2}(x_2) &= B^{x_1} \cdot G^{a_1} \cdot B^{x_2} \cdot G^{a_2} = \\ &= B^{x_1+x_2} \cdot G^{a_1+a_2} = D_{a_1+a_2}(x_1 + x_2). \end{aligned}$$

Это свойство нам потребуется при обсуждении протокола голосования в конце главы.

13.3. Доказательства с нулевым разглашением

Допустим, что Алиса пытается убедить Боба в обладании какой-то информацией, не сообщая в точности, какими конкретно сведениями она располагает. Именно с таким очевидным противоречием имеют дело доказательства с нулевым разглашением. В литературе по доказательствам с нулевым разглашением Алису называют

доказывающей стороной, т. к. именно она хочет что-то доказать, а Боба — проверяющей стороной (или проверяющим), поскольку в его намерения входит проверить, действительно ли доказывающий что-то знает. Мы тоже будем следовать установившейся традиции, именуя доказывающего Пегги (от англ. prover), а проверяющего — Виктором (от англ. verifier).

Классический пример доказательств с нулевым разглашением связан с проблемой изоморфизма графов. Два графа G_1 и G_2 с одним и тем же количеством вершин называются изоморфными, если вершины одного из них можно перенумеровать так, чтобы получился второй. Такая перенумерация φ называется изоморфизмом графов и обозначается

$$\varphi : G_1 \longrightarrow G_2.$$

Поиск изоморфизма между двумя графами — трудная вычислительная проблема.

Предположим, что Пегги знает какой-то изоморфизм φ между двумя общеизвестными графами G_1 и G_2. Назовем φ секретными исходными данными доказывающей стороны, а графы G_1 и G_2 — открытыми, или общими исходными данными. Пегги хочет доказать Виктору, что ей известен изоморфизм указанных графов, ничего не сообщая ему о точной природе изоморфизма. Это делается с помощью следующего доказательства с нулевым разглашением.

Пегги применяет секретную случайную перестановку ψ к вершинам графа G_2, получает другой граф H, изоморфный обоим исходным, и публикует H как **обязательство**. При этом она, естественно, знает в чем состоит секрет:

$$\begin{aligned} \varphi &: G_1 \longrightarrow G_2, \\ \psi &: G_2 \longrightarrow H, \\ \psi \circ \varphi &: G_1 \longrightarrow H. \end{aligned}$$

Виктор делает **запрос**, выбирая число $B \in \{1, 2\}$ и спрашивая об изоморфизме графов H и G_B. Пегги выдает свой **ответ** на запрос, пересылая Виктору либо $\chi = \psi$, либо $\chi = \psi \circ \varphi$. Протокол выглядит как

$$\begin{aligned} P &\longrightarrow V : H, \\ V &\longrightarrow P : B, \\ P &\longrightarrow V : \chi. \end{aligned}$$

Исследуем вопрос о возможности жульничества со стороны Пегги. Если Пегги реально не знает изоморфизма φ, то для корректного

ответа ей необходимо заранее узнать, какой именно граф G_b Виктор собирается ей послать. Следовательно, если Пегги пытается обмануть проверяющего, она способна дать правильный ответ на запрос лишь в 50% случаев. Поэтому повторяя протокол несколько раз, честная Пегги сможет убедить Виктора в том, что она действительно знает упомянутый изоморфизм с малой вероятностью ошибки.

Теперь стоит выяснить: извлекает ли Виктор какую-либо полезную информацию из протокола, т. е. действительно ли доказательство Пегги ничего не разглашает? Прежде всего заметим, что Пегги следует в каждом раунде обмена сообщениями в протоколе генерировать новый граф H. В противном случае у Виктора появляется возможность тривиального жульничества. Будем считать, что такого не происходит.

Чтобы увидеть, получил ли Виктор скрываемую информацию в результате работы протокола, нужно посмотреть на его запись и понять, можно ли что-либо полезное из нее извлечь. Чтобы убедиться, что Виктор реально ничем не пополнил свои знания, достаточно заметить, что он не смог бы выступать в этом протоколе в качестве Пегги. Следовательно, Виктор не сможет убедить кого-то другого в своем знании изоморфизма. Тем самым можно сделать вывод: Пегги своим доказательством не открыла тайны изоморфизма.

Виктор может сделать правдоподобные записи протокола без участия Пегги, используя следующую имитацию:

- выбрать $B \in \{0,1\}$,
- переставив произвольным образом вершины графа G_B, получить изоморфный граф H и соответствующий изоморфизм χ,
- сфабриковать запись протокола:

$$P \longrightarrow V : H,$$
$$V \longrightarrow P : B,$$
$$P \longrightarrow V : \chi.$$

Таким образом, диалоговая природа протокола показывает, что он осуществляет доказательство с нулевым разглашением. Отметим, что три фазы

$$\text{обязательство} \longrightarrow \text{запрос} \longrightarrow \text{ответ}$$

являются характерной чертой таких протоколов.

Протоколы доказательств тоже можно разделить на типы по степени криптостойкости. К одному типу отнести доказательства,

стойкие относительно противника с ограниченными вычислительными возможностями, а к другому — с неограниченными. Ясно, что к основным свойствам систем интерактивных доказательств относятся

- *Полнота*: если Пегги действительно знает то, что собирается доказать, то Виктор должен принять ее доказательство с вероятностью 1.
- *Корректность*: если Пегги не знает, что доказывает, то вероятность убеждения Виктора должна быть крайне мала.

Будем считать, что Виктор имеет вычислительные средства, ограниченные полиномиальным временем, а Пегги обладает неограниченными вычислительными возможностями.

Как мы уже отметили, свойство нулевого разглашения связано с концепцией имитации. Предположим, что справедливые записи протокола (которые действительно получились в результате работы протокола, осуществившего доказательство) обозначаются через $\mathcal{V}$, а возможные имитации — через S. Надежность протокола, таким образом, связана с тем, насколько S похоже на $\mathcal{V}$.

Говорят, что доказательство обладает *абсолютно* нулевым разглашением, если нападающий, обладая неограниченными вычислительными возможностями, не может отличить $\mathcal{V}$ от S. Если же отличить их друг от друга не в состоянии лишь противник с ограниченными вычислительными возможностями, будем говорить, что доказательство имеет *вычислительно* нулевое разглашение.

Обладая неким секретом, можно использовать доказательство с нулевым разглашением как схему идентификации. Недостаток предыдущего примера с изоморфизмом графов заключается в его непрактичности. Данные, которые при этом передаются, занимают большой объем, причем протокол необходимо повторять несколько раз, чтобы абсолютно убедить Виктора в знании секрета.

К счастью, при обсуждении подписи Шнорра в главе 10 мы уже видели протокол, имеющий более высокую пропускную способность. Предположим, что Пегги знает $x = \log_G Y$ в конечной абелевой группе A простого порядка Q. Протокол, доказывающий это знание, может выполняться следующим образом:

$$P \longrightarrow V : R = G^k \text{ для случайного } k,$$
$$V \longrightarrow P : E,$$
$$P \longrightarrow V : S = k + xE \pmod{Q}.$$

Виктор убеждается в том, что Пегги знает дискретный логарифм

x, проверяя, что

$$R = G^S Y^{-E}.$$

Рассмотрим протокол более детально. Если Пегги не знает дискретного логарифма x, то она может попытаться обмануть Виктора. Ей удастся это сделать с вероятностью $1/Q$, что существенно меньше $1/2$, которая присутствовала в протоколе, основанном на изоморфизме графов.

Действительно ли Виктор не получает полезной информации из протокола? Ответ отрицателен, поскольку Виктор может имитировать записи протокола следующим образом:

- генерировать случайное число E по модулю Q;
- вычислить $R = G^S Y^{-E}$,
- выдать записи:

$$P \longrightarrow V : R,$$
$$V \longrightarrow P : E,$$
$$P \longrightarrow V : S.$$

Эта имитация приобретет важное значение при изучении стойкости подписи Шнорра в главе 17.

Проблема, связанная с доказательствами с нулевым разглашением, состоит в их интерактивной природе:

$$P \longrightarrow V : CO,$$
$$V \longrightarrow P : CH,$$
$$P \longrightarrow V : RE,$$

где CO — обязательство, CH — запрос и RE — ответ. Их легко можно сделать и не интерактивными (автономными), если заменить запрос вычислением криптографической хэш-функции, примененной к обязательству: $CH = H(CO)$. Идея здесь состоит в том, что доказывающий не может угадать запрос перед отправкой обязательства, поскольку для этого нужно было бы инвертировать хэш-функцию. Автономный протокол теряет свойство нулевого разглашения, поскольку его нельзя имитировать. Кроме того, обладающая неограниченными вычислительными ресурсами Пегги способна инвертировать хэш-функцию. Поэтому нужно ограничить возможности вычислений доказывающей стороны. Как только мы это сделаем, то будем говорить, что доказывающий приводит *аргумент* с нулевым разглашением (в отличие от термина «доказательство с нулевым разглашением»).

К аргументу хэш-функции можно добавить и другие данные, например, сообщение. Таким способом мы превращаем диалоговое доказательство какого-то знания в схему цифровой подписи:

$$CH = H(CO || \text{СООБЩЕНИЕ}).$$

Заметьте, что при использовании в доказательстве знания дискретного логарифма, хэш-значения обязательства и сообщения в качестве запроса, мы получаем в точности схему подписи Шамира.

Закончим параграф рассказом о протоколе, являющемся аргументом с нулевым разглашением, который потребуется при обсуждении схемы электронного голосования. Рассмотрим схему обязательств

$$D_a(x) = B^x G^a,$$

где $A = \langle G \rangle$ — конечная абелева группа простого порядка Q, B — элемент в A, чей дискретный логарифм по основанию G неизвестен, x — обязательство, a — случайное уникальное число. Нас интересует случай, когда обязательство принимает значение плюс или минус единица, т. е. $x \in \{-1, 1\}$. При реализации этой схемы в протоколе голосования нам будет важно доказать, что переданное число действительно лежит в указанном множестве, не раскрывая его истинного значения. С этой целью осуществляется следующий протокол:

- Вместе с публикацией обязательства $D_a(x)$ Пегги выбирает случайные числа d, r и w по модулю Q и публикует A_1 и A_2, где

$$A_1 = \begin{cases} G^r (D_a(x) B)^{-d}, & \text{если } x = 1, \\ G^w, & \text{если } x = -1; \end{cases}$$

$$A_2 = \begin{cases} G^w, & \text{если } x = 1, \\ G^r (D_a(x) B^{-1})^{-d}, & \text{если } x = -1. \end{cases}$$

- Виктор высылает Пегги случайный запрос C.
- Пегги отвечает

$$d' = C - d, \qquad r' = w + ad',$$

и выводит значения

$$(D_1, D_2, R_1, R_2) = \begin{cases} (d, d', r, r'), & \text{если } x = 1, \\ (d', d, r', r), & \text{если } x = -1. \end{cases}$$

– Виктор проверяет истинность равенств:

$$C = D_1 = D_2,$$
$$G^{R_1} = A_1(D_a(x)B)^{D_1},$$
$$G^{R_2} = A_2(D_a(x)B^{-1})^{D_2}.$$

Дабы убедиться в работоспособности протокола, нам надо показать, что

1. Если Пегги отвечает правдиво, то Виктор сможет убедиться в истинности трех равенств.
2. Если Пегги передала число, отличное от ± 1, то ей будет затруднительно выдать правильный ответ на запрос Виктора.
3. Протокол не дает Виктору никакой информации относительно переданного значения, кроме того, что оно принадлежит множеству $\{-1, 1\}$.

Оставим проверку перечисленных пунктов читателю. Заметим, что этот протокол можно выполнять и в автономном режиме, если определить C как

$$C = H(A_1||A_2||D_a(x)).$$

13.4. Система электронного голосования

Здесь представлена система электронного голосования, использующая достижения, описанные как в этой, так и в предыдущих главах. При этом мы хотим показать, как основные теоретические разработки комбинируются в сложном приложении, имеющем практическое значение. Будем предполагать, что в голосовании участвуют m лиц с правом голоса и n счетных комиссий. Использование большого числа счетных комиссий обеспечивает анонимность голосующего и предотвращает возможность фальсификации результатов голосования. Будем также считать, что избиратели могут отдать свой голос только за одного из кандидатов, например, демократа или республиканца.

Система голосования, с которой мы собираемся познакомиться, обладает следующими семью свойствами:

(1) голосовать имеют право только уполномоченные избиратели;
(2) ни один из голосующих не может отдать более одного голоса;
(3) ни один из участников процесса не может узнать, как проголосовал кто-то другой;
(4) никто не может продублировать голос какого-то другого участника компании;

(5) конечный результат будет корректно подсчитан;
(6) каждый из участников способен проверить, что результат подсчитан правильно;
(7) протокол будет работать и в случае, когда некоторые из его участников нечестны.

13.4.1. Установки системы

Каждая из счетных комиссий обладает шифрующей функцией с открытым ключом E_i. Будем предполагать, что фиксирована конечная абелева группа A простого порядка Q, в которой выбрана пара элементов B и G, причем никто (включая счетные комиссии) не знает решения уравнения

$$B = G^x.$$

Каждый из голосующих имеет алгоритм подписи с открытым ключом.

13.4.2. Заполнение бюллетеня

Каждый из m избирателей выбирает голос $v_j \in \{-1, 1\}$, случайное затемняющее число $a_j \in \mathbb{Z}/Q\mathbb{Z}$ и публикует свое решение:

$$D_j = D_{a_j}(v_j),$$

используя схему обязательств из параграфа 13.2. Это решение становится известным всем сторонам голосования: как счетным комиссиям, так и остальным избирателям. Вместе с B_j избиратель публикует автономную версию рассмотренного ранее протокола в подтверждение того, что его голос действительно выбран из множества $\{-1, 1\}$. Голос вместе с доказательством подписывается с помощью схемы подписи, принадлежащей избирателю.

13.4.3. Распределение бюллетеней

Для подведения итогов необходимо передать отданные голоса счетным комиссиям. Чтобы передать a_j и v_j счетным комиссиям, каждый избиратель применяет схему Шамира разделения секрета. С этой целью он выбирает два случайных многочлена по модулю Q степени $T < n$

$$R_j(X) = v_j + r_{1,j}X + \cdots + r_{T,j}X^T,$$
$$S_j(X) = a_j + s_{1,j}X + \cdots + s_{T,j}X^T,$$

и вычисляет

$$(u_{i,j}, w_{i,j}) = (R_j(i), S_j(i)) \quad \text{при } 1 \leqslant i \leqslant n.$$

Голосующий шифрует пары $(u_{i,j}, w_{i,j})$, используя алгоритм E_i i-ой счетной комиссии, и отсылает ей полученный результат. После этого избиратель передает многочлен $R_j(X)$, открыто регистрируя

$$D_{l,j} = D_{s_l,j}(r_{l,j}) \quad \text{при } 1 \leqslant l \leqslant T,$$

с помощью все той же описанной ранее схемы.

13.4.4. Проверка достоверности информации

Каждая из счетных комиссий должна проверить, что пара $(u_{i,j}, w_{i,j})$ действительно получена от избирателя с номером j и согласуется с переданным обязательством. Это достигается в результате проверки следующих равенств:

$$\begin{aligned} D_j \prod_{l=1}^{T} D_{l,j}^{i^l} &= D_{a_j}(v_j) \prod_{l=1}^{T} D_{s_l,j}(r_{l,j})^{i^l} = \\ &= B^{v_j} G^{a_j} \prod_{l=1}^{T} (B^{r_{l,j}} G^{s_{l,j}})^{i^l} = \\ &= B^{(v_j + \sum_{l=1}^{T} r_{l,j} i^l)} G^{(a_j + \sum_{l=1}^{T} s_{l,j} i^l)} = B^{u_{i,j}} G^{w_{i,j}}. \end{aligned}$$

13.4.5. Подсчет голосов

Каждая из n счетных комиссий подсчитывает бюллетени и публикует результаты:

$$U_i = \sum_{j=1}^{m} u_{i,j}.$$

Кроме того, она обнародует сумму затемняющих факторов:

$$W_i = \sum_{j=1}^{m} w_{i,j}.$$

Любой другой участник процедуры голосования, будь то счетная комиссия или избиратель, может убедиться в корректности опубликованных сумм, проверяя, что

$$\prod_{j=1}^{m} \left(D_j \prod_{l=1}^{T} B_{l,j}^{j^l} \right) = \prod_{j=1}^{m} B^{u_{i,j}} G^{w_{i,j}} = B^{U_i} G^{W_i}.$$

Каждая из сторон процесса может определить итог, беря T значений U_i и интерполируя по ним окончательный результат. Дело в том, что U_i — значение многочлена, представляющего сумму голо-

сов, в точке i. Чтобы убедиться в этом, рассмотрим

$$U_i = \sum_{j=1}^{m} u_{i,j} = \sum_{j=1}^{m} R_j(i) =$$
$$= \left(\sum_{j=1}^{m} v_j\right) + \left(\sum_{j=1}^{m} r_{1,j}\right) i + \cdots + \left(\sum_{j=1}^{m} r_{T,j}\right) i^T.$$

Если результат — отрицательное число, то большинство избирателей написало на бюллетене число «-1», а если положителен, то большинство поставило «$+1$». Осталось лишь показать, что этот протокол голосования обладает всеми семью свойствами, которые были анонсированы на стр. 347. Оставим это в качестве легкого упражнения.

Краткое содержание главы

- Схемы обязательств дают возможность участнику спрятать переданное значение и открыть его позже.
- Схемы обязательств должны обладать скрывающим и связывающим свойствами. Разработаны эффективные схемы, которые обладают или теоретико-информационно связывающим, или теоретико-информационно скрывающим свойством, но не одновременно.
- Интерактивное доказательство знания не разглашает никакой информации, если записи соответствующего протокола могут имитироваться без привлечения секретной информации.
- Интерактивные доказательства знания можно трансформировать в алгоритм цифровой подписи, если заменить запрос хэш-значением, связанным с сообщением.
- Используя основные примитивы шифрования, подписей, схем обязательств и доказательств с нулевым разглашением, можно создать довольно сложные протоколы. Как пример такой конструкции был приведен протокол электронного голосования.

Дополнительная литература

Книга Гольдрайха содержит множество подробностей о доказательствах с нулевым разглашением. С другой стороны, книга Стинсона дает хороший обзор этой темы. Протокол голосования, который был у нас описан, взят из статьи Крамера и др.

R. Cramer, M. Franklin, B. Schoenmakers and M. Yung. *Multi-authority secret-ballot elections with linear work*. In Advances in Cryptology — EuroCrypt '96, Springer-Verlag LNCS 1070, 72-83, 1996.

O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, 1999.

D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

Контрольные вопросы

13.1.1. Что такое схемы обязательств?

13.1.2. Что означают свойства связывания и сокрытия для схем обязательств?

13.1.3. Как можно использовать протокол доказательства с нулевым разглашением в целях идентификации?

13.1.4. Покажите, что если в протоколе об изоморфизме графов Пегги использует один и тот же граф H дважды, то Виктор сможет определить скрываемый изоморфизм графов.

13.1.5. Как можно переделать интерактивную идентификационную схему с нулевым разглашением в алгоритм цифровой подписи?

Упражнения

13.2.1. Пусть в группе A фиксированы четыре элемента G_1, G_2, B_1 и B_2, причем Пегги известен логарифм

$$\log_{G_1} B_1 = \log_{G_2} B_2 = x.$$

Разработайте протокол, с помощью которого Пегги может доказать Виктору, что эти два логарифма совпадают, не раскрывая значения x.

13.2.2. Покажите, что доказательство с нулевым разглашением справедливости обязательства $D_a(x)$, приведенное в тексте для $x \in \{-1, 1\}$, имеет два основных свойства (см. стр. 344).

13.2.3. Покажите, что протокол голосования обладает семью свойствами (см. стр. 347). Определите, какое число нечестных счетных комиссий может участвовать в протоколе, не нарушая его основных семи свойств.

ЧАСТЬ IV

ПРОБЛЕМЫ СТОЙКОСТИ

Описав основные примитивы криптографии с отрытым ключом, в которых у нас возникала потребность, покажем теперь, что их недостаточно. Слабые места возникают потому, что разработчики приемов часто не задумываются о том, как примитивы будут использоваться на практике. Чтобы разобраться в этом, мы сначала выделим некоторый список возможных атак и попытаемся сформулировать понятие стойкости по отношению к ним. Анализ ситуации покажет нам, что основные примитивы действительно не вполне стойки.

Осознав реальное положение дел, обратимся к двум подходам, позволяющим строить стойкие системы. Первый, основанный на чистой теории сложности, обречен на неудачу, поскольку чистая теория сложности имеет дело с самым плохим случаем, а не со средними, которые и встречаются на практике. Второй подход, связанный с доказуемой стойкостью, оказывается более подходящим для практических целей. Он, фактически, имеет высокое влияние на современную криптографию. Последний подход тоже происходит из теории сложности, но работает со средним, а не с наихудшим случаем.

ГЛАВА 14

АТАКИ НА СХЕМЫ С ОТКРЫТЫМ КЛЮЧОМ

Цели главы

- Объяснить атаку Винера, основанную на непрерывных дробях.
- Описать алгоритм приведения базиса решетки и показать на примерах, как его можно использовать для взлома криптографических систем.
- Рассказать о технике Копперсмита поиска малых корней полиномиальных уравнений над кольцами вычетов и ее криптографических приложениях.
- Ввести понятие частичной дискредитации ключа и анализа дефектов.

14.1. Введение

Мы опишем несколько атак, направленных на наивную эксплуатацию таких криптосистем, как *RSA* и *DSA*. Особое внимание будет уделено технике Копперсмита, основанной на приведении базиса решетки. Основная цель этой главы — продемонстрировать, что даже такого криптографического примитива, как односторонняя функция *RSA* с секретом

$$x \longrightarrow x^E \pmod{N},$$

недостаточно для построения стойкой системы шифрования. Криптостойкость системы главным образом зависит от способа использования этой функции. В следующих главах мы объясним, как можно разработать стойкую систему шифрования, опирающуюся как на *RSA*-функцию, так и на другую криптографическую технику, с которой мы уже знакомы.

14.2. Атака Винера на RSA

Мы уже отмечали, что в алгоритме *RSA* для ускорения операций с открытым ключом используют малые шифрующие экспоненты. В некоторых же приложениях этой криптосистемы существеннее ускорить процессы расшифровывания. Поэтому имеет смысл выбирать небольшую расшифровывающую экспоненту d. Ясно, что при этом получается большое значение открытой экспоненты E. Слишком маленькое число в качестве секретной экспоненты d мы брать не можем, поскольку атакующий определит ее простым перебором. Более того, учитывая изощренную атаку Винера, опирающуюся на непрерывные дроби, необходимо выбирать d среди чисел, размер которых не меньше, чем $\frac{1}{3}N^{\frac{1}{4}}$.

По вещественному числу $\alpha \in \mathbb{R}$ определим последовательности:

$$\alpha_0 = \alpha, \quad p_0 = q_0 = 1, \quad p_1 = a_0 a_1 + 1, \quad q_1 = a_1,$$
$$a_i = \lfloor \alpha_i \rfloor, \quad \alpha_{i+1} = \frac{1}{\alpha_i - a_i},$$
$$p_i = a_i p_{i-1} + p_{i-2} \text{ при } i \geqslant 2,$$
$$q_i = a_i q_{i-1} + q_{i-2} \text{ при } i \geqslant 2.$$

Целые числа $a_0, a_1, a_2, \ldots$ называются непрерывной дробью, представляющей α, а рациональные числа $\frac{p_i}{q_i}$ — подходящими дробями. Каждая из подходящих дробей несократима, а скорость роста их знаменателей сравнима с показательной.

Одним из важных результатов теории непрерывных дробей является то, что если несократимая дробь $\frac{p}{q}$ удовлетворяет неравенству:

$$\left| \alpha - \frac{p}{q} \right| \leqslant \frac{1}{2q^2},$$

то $\frac{p}{q}$ — одна из подходящих дробей в разложении α в непрерывную дробь.

Винер предлагает использовать непрерывные дроби при атаке на *RSA* следующим образом. Пусть у нас есть модуль $N = pq$, причем $q < p < 2q$. Допустим, что наша расшифровывающая экспонента удовлетворяет неравенству: $d < \frac{1}{3}N^{\frac{1}{4}}$, и нападающему это известно. Кроме того, ему дана шифрующая экспонента E, обладающая свойством

$$Ed = 1 \pmod{\varphi},$$

где $\varphi = \varphi(N) = (p-1)(q-1)$. Будем также считать, что $E < \varphi$, поскольку это выполнено в большинстве приложений. Заметим, из

предположений следует существование такого целого k, что

$$Ed - k\varphi = 1.$$

Следовательно,

$$\left|\frac{E}{\varphi} - \frac{k}{d}\right| = \frac{1}{d\varphi}.$$

Поскольку $\varphi \approx N$, получаем, что

$$|N - \varphi| = |p + q - 1| < 3\sqrt{N}.$$

Отсюда можно сделать вывод, что $\frac{E}{N}$ — довольно хорошее приближение в $\frac{k}{d}$. Действительно,

$$\left|\frac{E}{N} - \frac{k}{d}\right| = \left|\frac{Ed - Nk}{dN}\right| = \left|\frac{Ed - k\varphi - Nk + k\varphi}{dN}\right| =$$

$$= \left|\frac{1 - k(N - \varphi)}{dN}\right| \leqslant \left|\frac{3k\sqrt{N}}{dN}\right| = \frac{3k}{d\sqrt{N}}.$$

Поскольку $E < \varphi$, очевидно, $k < d$. Кроме того, по предположению $d < \frac{1}{4}N^{\frac{1}{4}}$. Значит,

$$\left|\frac{E}{N} - \frac{k}{d}\right| < \frac{1}{2d^2}.$$

Поскольку НОД $(k, d) = 1$, мы видим, что $\frac{k}{d}$ — подходящая дробь в разложении дроби $\frac{E}{N}$ в непрерывную. Таким образом, раскладывая число $\frac{E}{N}$ в непрерывную дробь, можно узнать расшифровывающую экспоненту, поочередно подставляя знаменатели подходящих дробей в выражение:

$$\left(M^E\right)^d = M \pmod N$$

для некоторого случайного числа M. Получив равенство, найдем d. Общее число подходящих дробей, которое нам придется при этом проверить, оценивается как $O(\ln N)$. Таким образом, изложенный метод дает линейный по сложности алгоритм определения секретного ключа в системе *RSA*, если последний не превосходит $\frac{1}{3}N^{\frac{1}{4}}$.

В качестве примера рассмотрим модуль *RSA*, равный

$$N = 9\,449\,868\,410\,449.$$

Пусть открытый ключ криптосистемы задан как

$$E = 6\,792\,605\,526\,025,$$

а секретный ключ удовлетворяет неравенству $d < \frac{1}{3}N^{\frac{1}{4}} \approx 584$. Разложим число $\alpha = \frac{E}{N}$ в непрерывную дробь и проверим знаменатель

каждой подходящей дроби: не является ли он секретным ключом. Подходящие дроби разложения α имею вид:

$$1, \frac{2}{3}, \frac{3}{4}, \frac{5}{7}, \frac{18}{25}, \frac{23}{32}, \frac{409}{569}, \frac{1659}{2308}, \cdots$$

Поочередно проверяя знаменатели, убедимся, что $d = 569$, т. е. знаменатель седьмой подходящей дроби — искомый секретный ключ.

14.3. Решетки и приведенные базисы

Позже в этой главе мы увидим, что приведение базиса решетки дает инструмент нападения на ряд криптосистем с открытым ключом. Поэтому здесь дается обзор этой важной области. Прежде всего необходимо напомнить некоторые сведения из линейной алгебры.

Пусть $x = (x_1, x_2, \ldots, x_n)$ — n-мерный вещественный вектор[1], т. е. $x_i \in \mathbb{R}$ при каждом i. Множество всех таких векторов обозначается символом $\mathbb{R}^n$. Для пары векторов определено *скалярное произведение*:

$$(x, y) = x_1y_1 + x_2y_2 + \cdots + x_ny_n,$$

являющееся вещественнозначной функцией от двух векторных аргументов. Вероятно Вам известно, что векторы x и y ортогональны (т. е. пересекаются под прямым углом) тогда и только тогда, когда

$$(x, y) = 0.$$

С помощью скалярного произведения можно определить абсолютную величину, или длину, вектора:

$$||x|| = \sqrt{(x, x)} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Это определение соответствует интуитивному понятию длины, при котором выполнены следующие свойства:

- *неотрицательность*: $||x|| \geqslant 0$, причем $||x|| = 0$ тогда и только тогда, когда $x = 0$;
- *неравенство треугольника*: любая пара векторов удовлетворяет неравенству:

$$||x + y|| \leqslant ||x|| + ||y||;$$

[1] Довольно часто (и в этой книге, в частности) векторы записывают в виде столбцов $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$. — *Прим. перев.*

– *пропорциональность*: для любого вектора x и числа $\lambda \in \mathbb{R}$ имеет место равенство:

$$||\lambda x|| = |\lambda| \cdot ||x||.$$

Множество векторов $\{b_1, \ldots, b_m\}$ в $\mathbb{R}^n$ называется *линейно независимым*, если уравнение

$$\lambda_1 b_1 + \lambda_2 b_2 + \cdots + \lambda_m b_m = 0$$

относительно чисел λ_i имеет только нулевое решение. Для линейно независимой системы векторов справедлива оценка: $m \leqslant n$.

Имея систему линейно независимых векторов $\{b_1, \ldots, b_m\}$ в $\mathbb{R}^n$, можно изучать множество всех их вещественных линейных комбинаций:

$$V = \left\{ \sum_{i=1}^{m} a_i b_i \middle| \; a_i \in \mathbb{R} \right\}.$$

Такое множество образует векторное подпространство в $\mathbb{R}^n$ размерности m, а система $\{b_1, \ldots, b_m\}$ называется базисом этого подпространства. Если из векторов базиса составить матрицу B, чей j-ый столбец совпадает с вектором b_j, то подпространство V можно описать как множество всевозможных произведений матрицы B на вектора из $\mathbb{R}^m$, т. е.

$$V = \{B \cdot x | \; x \in \mathbb{R}^m\}.$$

Такую матрицу B называют матрицей базиса.

В любом подпространстве V существует бесконечно много различных базисов. Как в научных, так и в технических приложениях довольно часто требуется выбор базиса со специальными свойствами. Одно из полезных свойств — попарная ортогональность базисных элементов:

$$(b_j, b_j) = 0 \text{ при } i \neq j.$$

Такие базисы называют *ортогональными*. К счастью, существует хорошо известный процесс *ортогонализации Грама–Шмидта*, позволяющий по произвольному базису $\{b_1, \ldots, b_m\}$ построить ортогональный $\{b_1^*, \ldots, b_m^*\}$ с помощью преобразований:

$$b_1^* = b_1, \quad \mu_{i,j} = \frac{\left(b_i, b_j^*\right)}{\left(b_j^*, b_j^*\right)} \text{ для } 1 \leqslant j < i \leqslant m, \quad b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*.$$

Пусть, например,

$$b_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad \text{и} \quad b_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Тогда

$$b_1^* = b_1 = \begin{pmatrix}2\\0\end{pmatrix}, \quad b_2^* = b_2 - \mu_{2,1} b_1^* = \begin{pmatrix}1\\1\end{pmatrix} - \frac{1}{2}\begin{pmatrix}2\\0\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix},$$

поскольку

$$\mu_{2,1} = \frac{(b_2, b_1^*)}{(b_1^*, b_1^*)} = \frac{2}{4} = \frac{1}{2}.$$

Теперь $(b_1^*, b_2^*) = 0$, так что построенный базис Грама–Шмидта ортогонален.

Решетка очень похожа на векторное подпространство V, о котором шла речь выше, но ее рассматривают не над вещественными, а над целыми числами. Зафиксировав линейно независимую систему векторов $\{b_1, \ldots, b_n\}$ в $\mathbb{R}^n$, решеткой считают множество всех целочисленных комбинаций базисных векторов:

$$L = \left\{ \sum_{i=1}^{m} a_i b_i \middle| \; a_i \in \mathbb{Z} \right\} = \{ B \cdot a \mid \; a \in \mathbb{Z}^m \}.$$

В этой ситуации, как и в случае векторных подпространств, векторы b_i называют базисными векторами решетки, а матрицу B — базисной матрицей. Чтобы понять, почему множество L называют именно решеткой, рассмотрим L, порожденную парой векторов

$$b_1 = \begin{pmatrix}2\\0\end{pmatrix} \quad \text{и} \quad b_2 = \begin{pmatrix}1\\1\end{pmatrix}.$$

Множество L состоит из всех векторов вида

$$\begin{pmatrix}2x + y\\ y\end{pmatrix}, \quad x, y \in \mathbb{Z}.$$

Если изобразить точки с такими координатами на координатной плоскости, то можно легко увидеть, что они будут располагаться в узлах сетки, или решетки.

Решетки — дискретный аналог векторных пространств. Ввиду их дискретности корректно определен наименьший элемент данной решетки, например, как ненулевой вектор с наименьшей длинной. Множество трудных вычислительных задач, особенно тех, которые возникают в криптографии, можно свести к нахождению наименьшего вектора в решетке. Далее мы разберем некоторые из приложений этой теории. Поиск наименьшего ненулевого вектора в общей многомерной решетке считается трудной задачей. Для маломерных решеток она не представляет серьезных проблем, чем мы и будем пользоваться.

Вернемся к общей теории решеток. Как и в случае векторных пространств, имея какой-то базис решетки, можно задаться вопро-

сом о поиске лучшего. Пусть B — матрица базиса решетки L. Единственный способ получения другой матрицы базиса B' состоит в умножении B на унимодулярную целочисленную матрицу U:

$$B' = B \cdot U,$$

т. е. целочисленную матрицу, определитель которой равен плюс или минус единице: $\det U = \pm 1$. Следовательно, модуль определителя матрицы базиса является инвариантом решетки, т. е. не зависит от выбора базиса. Поэтому уместен термин *дискриминант решетки*, который по матрице базиса определяется как

$$\Delta = \sqrt{|\det(B^t B)|}.$$

Если L — решетка полного ранга, т. е. B — квадратная матрица, то

$$\Delta = |\det B|.$$

Возникает вопрос о существовании ортогонального базиса в данной решетке L. В общем случае ответ отрицателен. Если внимательно посмотреть на процесс ортогонализации Грама–Шмидта, то легко увидеть, что если начать даже с целочисленного базиса, коэффициенты $\mu_{i,j}$ почти никогда не будут целыми числами. Следовательно, полученный таким способом набор ортогональных векторов будет образовывать базис векторного подпространства V, но решетка, которую он порождает, будет отлична от исходной. Дело в том, что переходя от одного базиса решетки к другому, мы не можем использовать нецелые коэффициенты. Тем не менее, можно попытаться выбрать новый базис решетки, который будет «близок» к ортогональному, т. е.

$$|\mu_{i,j}| \leqslant \frac{1}{2} \text{ для } 1 \leqslant j < i \leqslant n.$$

Эти соображения натолкнули Ленстру, Ленстру и Ловаса на определение *приведенного* базиса решетки, который часто называют LLL-приведенным базисом в честь изобретателей.

Определение 14.1. Базис $\{b_1, \ldots, b_m\}$ решетки называют *LLL-приведенным*, если для соответствующего ортогонализованного базиса Грама–Шмидта $\{b_1^*, \ldots, b_m^*\}$ выполнены условия:

$$|\mu_{i,j}| \leqslant \frac{1}{2} \text{ для } 1 \leqslant j < i \leqslant m, \tag{14.1}$$

$$||b_i^*||^2 \geqslant \left(\frac{3}{4} - \mu_{i,j}^2\right) ||b_{i-1}^*||^2 \text{ для } 1 < i \leqslant m. \tag{14.2}$$

Поистине удивительно, что

- LLL-приведенный базис можно найти за полиномиальное время (метод приведения описан ниже);
- первый вектор приведенного базиса самый короткий — фактически, он близок к наименьшему ненулевому вектору, в том смысле, что

$$||b_1|| \leqslant 2^{\frac{m-1}{2}} ||x|| \quad \forall x \in L \setminus \{0\};$$

- $||b_1|| \leqslant 2^{\frac{m}{4}} \Delta^{\frac{1}{m}}$.

Константа $2^{\frac{m-1}{2}}$, появляющаяся во втором свойстве, возникает из-за наиболее неблагоприятного случая. Практически, после применения LLL-алгоритма к базису большинства решеток разумной размерности получается LLL-приведенный базис, первый вектор которого имеет наименьшую из возможных положительных длин в решетке.

LLL-алгоритм работает следующим образом. Мы храним копию как текущего базиса решетки B, так и ассоциированного с ним базиса Грама-Шмидта B^*. В каждый момент времени исследуется фиксированный столбец с номером k, начиная с $k = 2$.

- Если условие (14.1) не выполнено для $\mu_{k,j}$ при $1 \leqslant j < k$, то мы меняем матрицу базиса так, чтобы оно стало верным.
- Если условие (14.2) не выполнено для столбцов с номерами k и $k-1$, то переставляем эти столбцы и уменьшаем значение k на единицу (если, конечно, $k \neq 2$). Если же условие (14.2) выполняется, мы увеличиваем значение k на единицу.

В какой-то момент мы получим $k = m$, и алгоритм остановится. Можно показать, что число шагов алгоритма, при котором понижается номер k, ограничено. Это гарантирует, что алгоритм останавливается. Ясно, что в результате работы описанного алгоритма получится LLL-приведенный базис.

В качестве примера рассмотрим базис двумерной решетки в $\mathbb{R}^2$, с которым мы уже встречались:

$$b_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \qquad b_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Соответствующий ортогональный базис Грама–Шмидта уже вычислен:

$$b_1^* = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \qquad b_2^* = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Но эти векторы не образуют базиса соответствующей решетки, поскольку не могут быть получены из $\{b_1, b_2\}$ унимодулярным преобразованием.

Применяем LLL-алгоритм с $k = 2$. Видно, что первое условие (14.1) выполнено, поскольку $\mu_{2,1} = \frac{1}{2}$. Однако второе соотношение (14.2) ложно, т. к.

$$1 = ||b_2^*||^2 \leqslant \left(\frac{3}{4} - \mu_{2,1}^2\right) ||b_1^*||^2 = \frac{1}{2} \cdot 4 = 2.$$

Поэтому нам надо переставить вектора исходного базиса и вновь применить процесс ортогонализации Грама–Шмидта. Новый базис решетки имеет вид:

$$b_1 = \begin{pmatrix}1\\1\end{pmatrix}, \qquad b_2 = \begin{pmatrix}2\\0\end{pmatrix},$$

а его ортогонализация записывается как

$$b_1^* = \begin{pmatrix}1\\1\end{pmatrix}, \qquad b_2^* = \begin{pmatrix}1\\-1\end{pmatrix}.$$

Снова проверяем первое условие. На этот раз $\mu_{2,1} = 1$, что противоречит (14.1). Чтобы его подправить, вычтем b_1 из b_2 и получим $\mu_{2,1} = 0$. Теперь базис решетки выглядит как

$$b_1 = \begin{pmatrix}1\\1\end{pmatrix}, \qquad b_2 = \begin{pmatrix}1\\-1\end{pmatrix},$$

причем соответствующий базис Грама–Шмидта совпадает с ним:

$$b_1^* = \begin{pmatrix}1\\1\end{pmatrix}, \qquad b_2^* = \begin{pmatrix}1\\-1\end{pmatrix}.$$

Проверяем второе условие. Имеем

$$2 = ||b_2^*||^2 \geqslant \left(\frac{3}{4} - \mu_{2,1}^2\right) ||b_1^*||^2 = \frac{3}{4} \cdot 2 = \frac{3}{2}.$$

Итак, оба условия выполнены и можно заключить, что

$$b_1 = \begin{pmatrix}1\\1\end{pmatrix}, \qquad b_2 = \begin{pmatrix}1\\-1\end{pmatrix}$$

является приведенным базисом решетки L.

Закончим введение в теорию решеток замечанием, что существует непосредственная связь между непрерывными дробями и приведением решеток. Разложение вещественного числа α в непрерывную дробь позволяет найти такие целые числа p и q с не очень большим q, что величина

$$|q\alpha - p|$$

будет достаточно маленькой. Аналогичный эффект может быть достигнут, если применить LLL-алгоритм к решетке L, порожденной

столбцами матрицы

$$\begin{pmatrix} 1 & 0 \\ C\alpha & -C\alpha \end{pmatrix}$$

с некоторой константой C. Он обеспечивается наличием «кратчайшего» вектора

$$\begin{pmatrix} q \\ C(q\alpha - p) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ C\alpha & -C\alpha \end{pmatrix} \cdot \begin{pmatrix} q \\ p \end{pmatrix}$$

в решетке L. Таким образом, LLL-алгоритм можно рассматривать как обобщение непрерывных дробей на многомерный случай.

14.4. Атаки на RSA, основанные на решетках

В этом параграфе мы узнаем, как можно использовать решетки в атаках на криптосистемы, если нападающему известна какая-то дополнительная информация о них. Эти атаки, по существу, аналогичны атаке Винера, рассмотренной ранее, которая взламывает *RSA* при условии, что нападающий осведомлен о небольшом размере расшифровывающей экспоненты. Многие работы на эту тему инициированы начальной работой Копперсмита, которая позже была упрощена Хоугрэйв-Грэхемом.

В основе атаки лежит один из способов решения следующей задачи. Пусть дан многочлен

$$F(x) = F_0 + F_1 x + \cdots + F_{D-1}x^{D-1} + x^D$$

степени D с целыми коэффициентами, причем известно, что он обладает *малым* корнем x_0 по модулю N (т. е. $F(x_0) = 0 \pmod N$), например, $|x_0| < N^{\frac{1}{D}}$. Можно ли эффективно найти этот корень? Ответ на этот вопрос неожиданно положителен и влечет множество интересных следствий для криптографии.

Идея решения задачи состоит в том, чтобы найти многочлен $h(x) \in \mathbb{Z}[x]$, имеющий тот же корень по модулю N, что и $F(x)$, но с малой нормой, т. е. число

$$||h||^2 = \sum_{i=0}^{\deg h} h_i^2$$

должно быть небольшим. Если такой многочлен $h(x)$ найден, то можно применить следующую лемму.

Лемма 14.2. Пусть $h(x) \in \mathbb{Z}[x]$ — многочлен, степень которого не превосходит n, а N и X — натуральные числа. Предположим, что

$$||h(xX)|| < \frac{N}{\sqrt{n}}.$$

Тогда если $|x_0| < X$ и $h(x_0) = 0 \pmod N$, то равенство $h(x_0) = 0$ имеет место и в кольце целых чисел, а не только в кольце вычетов.

Вернемся к исходному многочлену $F(x)$ степени D и заметим, что если $F(x_0) = 0 \pmod N$, то $(F(x_0))^k = 0 \pmod{N^k}$. Более того, если мы возьмем натуральное число M и рассмотрим

$$g_{U,V}(x) = N^{M-V} x^U \left(F(x)\right)^V,$$

то

$$g_{U,V}(x_0) = 0 \pmod{N^M}$$

для всех $0 \leqslant U < D$ и $0 \leqslant V \leqslant M$. Попытаемся теперь найти такие $a_{U,V} \in \mathbb{Z}$, при которых многочлен

$$h(x) = \sum_{U \geqslant 0} \sum_{V=0}^{M} a_{U,V} g_{U,V}(x)$$

с фиксированным M будет удовлетворять предположениям предыдущей леммы.

Иначе говоря, мы хотим найти такие целые числа $a_{U,V}$, при которых норма многочлена $h(x)$ подчиняется неравенству

$$||h(xX)|| \leqslant \frac{N^M}{\sqrt{D(M+1)}},$$

где

$$h(xX) = \sum_{U \geqslant 0} \sum_{V=0}^{M} a_{U,V} g_{U,V}(xX).$$

Задача о поиске таких коэффициентов называется задачей минимизации, и ее можно решить с помощью приведения базиса решетки, что мы сейчас продемонстрируем на простом примере.

Пусть

$$F(x) = x^2 + Ax + B.$$

Найдем такой x_0, что

$$F(x_0) = 0 \pmod N.$$

Положим в описанной выше конструкции $M = 2$ и вычислим

$$
\begin{aligned}
g_{0,0}(xX) &= N^2,\\
g_{1,0}(xX) &= XN^2x,\\
g_{0,1}(xX) &= BN + AXNx + NX^2x^2,\\
g_{1,1}(xX) &= BNXx + ANX^2x^2 + NX^3x^3,\\
g_{0,2}(xX) &= B^2 + 2BAXx + (A^2 + 2B)X^2x^2 + 2AX^3x^3 + X^4x^4,\\
g_{1,2}(xX) &= B^2Xx + 2BAX^2x^2 + (A^2 + 2B)X^3x^3 + 2AX^4x^4 + X^5x^5.
\end{aligned}
$$

Поищем линейную комбинацию этих шести многочленов с наименьшими коэффициентами результирующего многочлена. Фактически нам нужно найти наименьший вектор в решетке, порожденной столбцами матрицы, которые образованы коэффициентами этих многочленов:

$$
C = \begin{pmatrix}
N^2 & 0 & BN & 0 & B^2 & 0\\
0 & XN^2 & AXN & BNX & 2ABX & XB^2\\
0 & 0 & NX^2 & ANX^2 & (A^2+2B)X^2 & 2ABX^2\\
0 & 0 & 0 & NX^3 & 2AX^3 & (A^2+2B)X^3\\
0 & 0 & 0 & 0 & X^4 & 2AX^4\\
0 & 0 & 0 & 0 & 0 & X^5
\end{pmatrix}
$$

Определитель этой матрицы равен

$$\det C = N^6X^{15},$$

и применяя к ней LLL-алгоритм, мы получим новую матрицу базиса C', первый столбец которой

$$c'_1 = \begin{pmatrix} c'_{1,1}\\ c'_{2,1}\\ \vdots\\ c'_{6,1} \end{pmatrix}$$

будет удовлетворять неравенству

$$||c'_1|| \leqslant 2^{\frac{6}{4}}(\det C)^{\frac{1}{6}} = 2^{\frac{3}{2}}NX^{\frac{5}{2}}.$$

Следовательно, для многочлена

$$h(x) = c'_{1,1}g_{0,0}(x) + c'_{2,1}g_{1,0}(x) + \cdots + c'_{6,1}g_{1,2}(x)$$

выполнено неравенство

$$||h(xX)|| \leqslant 2^{\frac{3}{2}}NX^{\frac{5}{2}}.$$

Чтобы воспользоваться леммой 14.2, нам необходимо потребовать:

$$2^{\frac{3}{2}} N X^{\frac{5}{2}} < \frac{N^2}{\sqrt{6}}.$$

Значит, определяя целый корень многочлена $h(x)$, мы найдем малый корень x_0 многочлена $F(x)$ по модулю N, если

$$|x_0| \leqslant X = \frac{N^{\frac{2}{5}}}{48^{\frac{1}{5}}}.$$

В частности, метод будет работать, когда $|x_0| \leqslant N^{0,39}$.

Аналогичные рассуждения можно применить к любому многочлену степени D и получить следующую теорему:

Теорема 14.3. (Копперсмит.) Пусть $f \in \mathbb{Z}[x]$ — приведенный многочлен[1] степени D, а N — натуральное число. Если у многочлена f по модулю N есть корень x_0, удовлетворяющий неравенству $|x_0| \leqslant X = N^{\frac{1}{D-\varepsilon}}$, то этот корень можно найти за полиномиальное от $\ln N$ и $\frac{1}{\varepsilon}$ время при фиксированном значении D.

Сформулируем аналог леммы 14.2 для многочлена от двух переменных.

Лемма 14.4. Пусть $h(x, y) \in \mathbb{Z}[x, y]$ — сумма не более чем W мономов и

(а) $h(x_0, y_0) = 0 \pmod{N^E}$ для некоторых натуральных чисел N и E, причем целые x_0 и y_0 подчиняются неравенствам: $|x_0| < X$ и $|y_0| < Y$,

(б) $||h(xX, yY)|| < \frac{N^E}{\sqrt{w}}$.

Тогда соотношение $h(x_0, y_0) = 0$ имеет место и в кольце $\mathbb{Z}$.

А вот аналог теоремы 14.3 в случае многочленов от двух переменных носит эвристический характер.

Теперь будем применять теорему 14.3 и ее обобщения для описания атак, направленных на *RSA*.

14.4.1. Атака Хастада

В главе 7 мы рассмотрели следующую атаку на систему *RSA*. Пусть даны три открытых ключа (N_i, E_i) с одной и той же шифрующей экспонентой $E_i = 3$. Если пользователь посылает одно сообщение,

[1] Приведенным называется многочлен с единичным старшим коэффициентом. — *Прим. перев.*

зашифрованное этими ключами, то нападающий легко дешифрует сообщение, применив китайскую теорему об остатках.

Предположим теперь, что мы приняли меры против такой атаки, добавив к сообщению m перед шифрованием некоторые специфические для пользователя данные. Пусть, например, шифротекст, отправляемый i-му пользователю имеет вид

$$C_i = (i \cdot 2^H + m)^3 \pmod{N_i}.$$

Но и здесь сообщение все еще можно дешифровать, использовав метод, изобретенный Хастадом. Атака имеет непосредственное отношение к теореме Копперсмита, т. к. сценарий атаки с K пользователями и шифрующей экспонентой E можно интерпретировать как набор из K многочленов степени E:

$$g_i(x) = (i \cdot 2^H + x)^E - C_i, \quad 1 \leqslant i \leqslant K.$$

При этом нам известно, что существует такое m, что

$$g_i(m) = 0 \pmod{N_i}.$$

Наша цель — найти m. Можно считать, что m меньше каждого из модулей N_i. Положив $N = N_1 N_2 \cdots N_K$, с помощью китайской теоремы об остатках можно найти такие T_i, что многочлен

$$g(x) = \sum_{i=1}^{K} T_i g_i(x)$$

будет удовлетворять соотношению

$$g(m) = 0 \pmod{N}.$$

Теперь, если у нас есть достаточно много шифротекстов, т. е. $K > E$, то опираясь на теорему 14.3, мы узнаем m за полиномиальное время. Действительно, g — приведенный многочлен степени E и

$$m < \min_i N_i < N^{1/K} < N^{1/E}.$$

14.4.2. Атака Франклина–Рейтера и обобщение Копперсмита

Предположим, что у нас есть открытый ключ RSA, принадлежащий Алисе. Атака Франклина–Рейтера применяется в следующей ситуации. Боб хочет отправить Алисе два шифрованных сообщения m_1 и m_2, связанных друг с другом с помощью следующего открытого многочлена:

$$m_2 = f(m_1) \pmod{N}.$$

По соответствующим шифротекстам C_1 и C_2 нападающий имеет хорошие шансы раскрыть сообщения m_1 и m_2 при любой небольшой шифрующей экспоненте E. Атака наиболее проста, если

$$f = Ax + B \quad \text{и} \quad E = 3,$$

причем A и B фиксированы и известны атакующему. Нападающий знает, что m_2 — корень по модулю N многочленов

$$g_1(x) = x^3 - C_2 \quad \text{и} \quad g_2(x) = (f(x))^3 - C_1.$$

Поэтому линейная функция $x - m_2$ делит как $g_1(x)$, так и $g_2(x)$.

Найдем наибольший общий делитель многочленов $g_1(x)$ и $g_2(x)$. Строго говоря, это невозможно сделать в общей ситуации, поскольку кольцо $(\mathbb{Z}/N\mathbb{Z})[x]$ не является евклидовым. Однако если алгоритм Евклида при вычислении НОД (g_1, g_2) нарушается, то мы определим множители N и вскроем секретный ключ Алисы. Можно показать, что в случае

$$f = Ax + B \quad \text{и} \quad E = 3$$

искомый наибольший общий делитель (если таковой существует) должен быть линейным множителем многочлена $x - m_2$, т. е. совпадать с ним. Значит, нападающий найдет m_2, а затем m_1.

Обобщение Копперсмита атаки Франклина и Рейтера строится по пути, на котором можно получить дополнительный результат из атаки Хастада. Предположим, что сообщение m перед шифрованием пополняется некоторыми случайными данными. Например, если N — n-битовый модуль RSA, а m — k-битовое сообщение, то можно добавить $n - k$ случайных битов либо в начало, либо в конец сообщения. Пусть

$$m' = 2^{n-k}m + r,$$

где r — зависящее от m случайное число длины $n - k$. Копперсмит показал, что такое пополнение сообщения нестойко.

Предположим, что Боб дважды посылает одно сообщение Алисе, т. е. возникает два шифротекста C_1 и C_2, соответствующих сообщениям

$$m_1 = 2^{n-k}m + r_1, \quad m_2 = 2^{n-k}m + r_2,$$

где r_1 и r_2 — два разных случайных числа, состоящие из $n-k$ битов. Атакующий вводит обозначение $y_0 = r_2 - r_1$ и пытается решить систему уравнений:

$$\begin{cases} g_1(x, y) = x^E - C_1, \\ g_2(x, y) = (x + y)^E - C_2. \end{cases}$$

Он вычисляет результант $h(y)$ многочленов $g_1(x,y)$ и $g_2(x,y)$ относительно переменной x. Теперь y_0 — маленький корень многочлена $h(y)$ степени E^2. Опираясь на теорему Копперсмита 14.3, нападающий находит разность $r_2 - r_1$ и узнает m_2, используя метод атаки Франклина–Рейтера.

Поскольку тривиальная схема пополнения сообщений дискредитирована, нужно найти стойкий метод пополнения сообщений для криптосистемы *RSA*. К этому вопросу мы вернемся в главе 17.

14.4.3. Обобщение атаки Винера

Напомним, что атака Винера на *RSA* возможна в том случае, если атакующему известно о неравенстве

$$d \leqslant \frac{1}{3}N^{\frac{1}{4}},$$

где d — секретная экспонента, а N — модуль *RSA*. Бонех и Дерфи, используя двумерный аналог теоремы Копперсмита (т. 14.3), с помощью эвристического алгоритма смогли обобщить атаку Винера на случай, когда

$$d \leqslant N^{0,292}.$$

Мы не будем рассказывать о нем подробно, но покажем, как авторы подходят к проблеме, известной под именем задачи о малом обратном.

Пусть у нас есть *RSA*-модуль N, шифрующая экспонента E и расшифровывающая экспонента d. По определению, существует такое целое число k, что

$$Ed + \frac{k\varphi}{2} = 1,$$

где $\varphi = \varphi(N)$ — порядок группы обратимых элементов в кольце $\mathbb{Z}/N\mathbb{Z}$. Подставляя в эту формулу значение φ, получим

$$Ed + k\left(\frac{N+1}{2} - \frac{p+q}{2}\right) = 1.$$

Положим

$$s = -\frac{p+q}{2}, \qquad A = \frac{N+1}{2}.$$

Теперь при малом d, т. е. $d < N^{\delta}$, его вычисление равносильно поиску двух малых решений k и s уравнения:

$$f(k,s) = k(A+s) = 1 \pmod{E}.$$

Чтобы увидеть, что k и s действительно малы по сравнению с мо-

дулем E этого уравнения, обратите внимание на то, что $E \approx N$, поскольку d мало, и поэтому

$$|s| < 2N^{0,5} \approx E^{0,5} \quad \text{и} \quad |k| < \frac{2dE}{\varphi} \leqslant \frac{3dE}{N} \approx E^{\delta}.$$

Поиск целых чисел k и s можно интерпретировать как поиск целого числа, близкого к A, обратный к которому по модулю E мал. Это и называется задачей о малом обратном. Бонех и Дерфи показали, что задача имеет решение при $\delta \leqslant 0,292$. Следовательно, они усилили атаку Винера. Способ, с помощью которого авторы добились успеха, основывается на многомерном аналоге методе Копперсмита, применяемом к многочлену $f(k, s)$.

14.5. Частичное раскрытие ключа

Частичное раскрытие ключа относится к следующему вопросу. Предположим, что в некоторой криптографической схеме нападающий узнал какое-то множество битов секретного ключа. Может ли он, опираясь на эту информацию, полностью восстановить секретный ключ? Иначе говоря, приводит ли частичное раскрытие секретного ключа к полному взлому криптосистемы? Приведем несколько примеров, связанных с *RSA*, хотя стоит отдавать себе отчет, что они не исчерпывают всех возможных ситуаций. Существует несколько теоретических результатов, посвященных частичному раскрытию ключа и в других схемах, например, *DSA* и криптосистемах с симметричным ключом.

14.5.1. Частичное раскрытие секретной экспоненты в *RSA*

Удивительно, но в наиболее общем случае эксплуатирования системы *RSA* с малой шифрующей экспонентой E можно тривиальным образом раскрыть половину высших значащих битов секретной экспоненты. Напомним, что существует такое натуральное число k, что $0 < k < E$ и

$$Ed - k(N - (p + q) + 1) = 1.$$

Теперь предположим, что для каждого возможного значения i ($0 < i \leqslant E$) нападающий вычисляет

$$D_i = \left\lfloor \frac{iN + 1}{E} \right\rfloor.$$

Тогда

$$|D_k - d| \leqslant \frac{k(p+q)}{E} \leqslant \frac{3k\sqrt{N}}{E} < 3\sqrt{N}.$$

Следовательно, D_k — хорошее приближение к истинному значению расшифровывающей экспоненты d.

Очевидно, число k должно быть четным. Поэтому при $E = 3$ мы должны получить $k = 2$. Значит, D_2 содержит половину значащих битов экспоненты d. К сожалению для нападающего и к счастью для пользователя криптосистемы, нет никакого способа узнать оставшиеся биты экспоненты, имея лишь половину главных значащих цифр.

14.5.2. Частичное раскрытие простых множителей модуля *RSA*

Предположим, что n-битовый модуль N алгоритма *RSA* имеет разложение на простые множители $N = pq$ с $p \approx q$ и атакующий узнал $\frac{n}{4}$ наименьших значащих битов числа p. Напомним, что общее количество знаков в p должно быть около $\frac{n}{2}$. Так что атакующий знает младшую половину всех значащих битов множителя p. Запишем

$$p = x_0 2^{\frac{n}{4}} + P_0, \qquad q = y_0 2^{\frac{n}{4}} + q_0,$$

где P_0 — известный набор знаков. Тогда

$$N = P_0 q_0 \pmod{2^{\frac{n}{4}}}$$

и мы можем определить величину[1] $q_0 = Q_0$. Далее выпишем многочлен

$$p(x,y) = (P_0 + 2^{\frac{n}{4}}x)(Q_0 + 2^{\frac{n}{4}}y) = P_0Q_0 + 2^{\frac{n}{4}}(P_0y + Q_0x) + 2^{\frac{n}{2}}xy.$$

Многочлен $p(x,y)$ второй степени зависит от двух переменных и имеет «известный» малый корень по модулю N, а именно (x_0, y_0), где $0 < x_0, y_0 \leqslant 2^{\frac{n}{4}} \approx N^{\frac{1}{4}}$. Следовательно, применяя обобщение теоремы Копперсмита 14.3, можно найти x_0 и y_0, после чего разложить N на множители.

Аналогичная атака имеет успех, когда нападающему известна старшая половина значащих цифр числа p.

14.5.3. Частичное раскрытие младших значащих цифр секретной экспоненты *RSA*

Предположим, что при малой шифрующей экспоненте E в *RSA* известна четверть младших значащих бит расшифровывающей экспо-

[1] И поэтому будем обозначать ее как Q_0. — *Прим. перев.*

ненты d, т. е.

$$d = D_0 + 2^{\frac{n}{4}} x_0,$$

где D_0 — известно и $0 \leqslant x_0 \leqslant 2^{\frac{3n}{4}}$. Напомним, что существует четное число k $(0 < k < E)$, при котором

$$Ed - k(N - (p + q) + 1) = 1.$$

Поскольку $N = pq$, получаем:

$$Edp - kp(N - p + 1) + kN = p.$$

Если положить $p_0 = p \pmod{2^{\frac{n}{4}}}$, то возникает уравнение

$$ED_0p_0 - kp_0(N - p_0 + 1) + kN - p_0 = 0 \pmod{2^{\frac{n}{4}}}, \qquad (14.3)$$

которое дает нам алгоритм полного восстановления экспоненты d. Для каждого четного числа k, не превосходящего E, решаем относительно p_0 уравнение (14.3) по модулю $2^{\frac{n}{4}}$. При этом получится $O(\frac{n}{4})$ возможных значений для p_0. Беря поочередно каждое из них, будем пытаться разложить N на множители, опираясь на технику предыдущего подпараграфа. Одно из таких p_0 совпадает с $p \pmod{2^{\frac{n}{4}}}$. В этом случае мы сможем разложить N на множители и тем самым восстановить d.

14.6. Анализ дефектов

Интересный класс атак сводится к попыткам внести сбой в работу криптосистемы. Ограничимся описанием такого сорта нападений на алгоритм подписи *RSA*, хотя подобные атаки возможны и на другие алгоритмы, как с открытым, так и с секретным ключом.

Вообразите, что у нас есть аппаратная реализация *RSA*, например, смарт-карта. Микропроцессор карты подписывает сообщения для нас, используя встроенный секретный ключ *RSA*, а нападающий, естественно, стремится его раскрыть. С этой целью он старается вынудить микропроцессор карты допустить ошибки в вычислениях, например нагрев ее или охладив, или как-то несильно повредив карту любым доступным способом.

Любопытный случай возникает, когда микропроцессор карты использует китайскую теорему об остатках в целях ускорения процедуры подписи, как это описано в главе 11 на стр. 293. При этом сначала вычисляется хэш-значение от сообщения:

$$H = h(M),$$

а затем

$$s_p = H^{d_p} \pmod{p}, \qquad s_q = h^{d_q} \pmod{q},$$

где $d_p = d \pmod{p-1}$ и $d_q = d \pmod{q-1}$. Окончательная подпись получается из s_p и s_q благодаря китайской теореме об остатках:

$$u = (s_q - s_p)t \pmod{q}, \qquad S = s_p + up,$$

где $t = p^{-1} \pmod{q}$.

Допустим теперь, что атакующий смог ввести дефекты в вычисления таким образом, что s_p оказалось вычислено неверно. Тогда он получит значение подписи S, удовлетворяющее соотношениям:

$$S^E \neq H \pmod{p}, \quad S^E = H \pmod{q}.$$

Следовательно, вычисляя

$$q = \text{НОД}\,(S^E - H, N),$$

можно разложить N на множители.

Краткое содержание главы

- Использование малой шифрующей экспоненты в *RSA* — неудачная идея ввиду атаки Винера.
- Решетки — дискретный аналог векторных пространств. В них есть наименьший ненулевой вектор.
- Приведение базиса часто дает наименьший ненулевой вектор в данной решетке.
- Теорема Копперсмита позволяет найти решение полиномиального уравнения в кольце вычетов, если известно, что решение мало. Метод поиска решения основан на построении решетки, зависящей от уравнения, и приведения ее базиса для отыскания наименьшего вектора.
- Атаки Хастада и Франклина–Рейтера показывают, что необходимо соблюдать осторожность при разработке схем пополнения сообщений в криптосистеме *RSA*.
- Опираясь на теорему Копперсмита, можно доказать, что частичное раскрытие знаков или p и q, или d в *RSA* может способствовать полному взлому криптосистемы.

Дополнительная литература

Основная работа об атаке Копперсмита на *RSA*, использующей решетки, принадлежит самому Копперсмиту. Этот подход к атакам

был упрощен в статье Хоугрэйв-Грэхема. Рассказ об атаках на *RSA* в этой главе близко следует обзору Бонеха. Полный обзор криптографических методов, основанных на решетках, дан в работе Нгуена и Стерна.

D. Boneh. *Twenty years of attacks on the RSA cryptosystem*. Notices of the American Mathematical Society (AMS), **46**, 203–213, 1999.

D. Coppersmith. *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*. J. Cryptology, **10**, 233–260, 1997.

N. Howgrave-Graham. *Finding small solutions of univariate modular equations revisited*. In Cryptography and Coding, Springer-Verlag LNCS 1355, 131–142, 1997.

P. Nguyen and J. Stern. *The two faces of lattices in cryptology*. In CALC '01, Springer-Verlag LNCS 2146, 146–180, 2001.

Контрольные вопросы

14.1.1. Как разложить вещественное число α в непрерывную дробь?

14.1.2. Какое число нужно разлагать в непрерывную дробь при атаке Винера на *RSA* с малой секретной экспонентой?

14.1.3. Что такое решетка?

14.1.4. Как Вы думаете, какое из свойств LLL-приведенных базисов решетки наиболее важно с точки зрения криптографии?

14.1.5. Предположим, что у нас есть уравнение $f(x_0) = 0 \pmod N$, где f — многочлен степени D. Насколько маленьким должен быть корень x_0, чтобы его можно было найти за полиномиальное время, используя метод Копперсмита?

14.1.6. Объясните, что подразумевают, говоря об атаке Хастада на криптосистему *RSA*?

Лабораторные работы

14.2.1. Реализуйте на программном уровне различные атаки из этой главы, такие как атака Винера и атаки на основе LLL-алгоритма. Исследуйте с их помощью следующие вопросы:

(а) существуют ли ситуации, когда эвристические аргументы в этой главе не работают?

(б) существуют ли ситуации, при которых атаки более успешны, чем позволяет теоретический анализ?

Упражнения

14.3.1. Покажите, что для LLL-приведенного базиса n-мерной решетки L выполняется условие: если x — ненулевой вектор решетки, то

$$||b_1|| \leqslant 2^{\frac{n-1}{2}}||x||.$$

14.3.2. Покажите на формальном уровне, что LLL-алгоритм, описанный в тексте, действительно завершается в течение полиномиального времени, если базис решетки имеет целые координаты.

14.3.3. Докажите лемму 14.2.

14.3.4. Покажите, что если в атаке Франклина–Рейтера $f = Ax + B$ и $E = 3$, то наибольший общий делитель, появляющийся в алгоритме, всегда линеен (при условии его существования).

14.3.5. Покажите, как разложить на множители модуль RSA, если известна половина старших значащих битов одного из его делителей.

ГЛАВА 15

ОПРЕДЕЛЕНИЯ СТОЙКОСТИ

Цели главы

- Объяснить различные понятия стойкости схем шифрования, в первую очередь семантическую стойкость и неразличимость шифрования.
- Описать различные модели атак, в частности, рассказать об адаптивной атаке с выбором шифротекста.
- Показать, как связаны концепции жесткости и адаптивной атаки с выбором шифротекста.
- Исследовать криптосистемы *RSA* и Эль-Гамаль с точки зрения этих определений стойкости и атак.
- Ввести понятия, имеющие отношение к стойкости схем подписи.
- Показать, что наивная (не хэшированная) схема подписи *RSA* нестойка из-за свойства гомоморфности.

15.1. Стойкость шифрования

До сих пор мы не исследовали детально вопроса о внедрении криптографических примитивов в реальные протоколы и системы. При проектировании криптографических систем нам нужно иметь абсолютно точное представление о том, какие именно свойства стойкости обеспечивает тот или иной примитив и как защитить разрабатываемые схемы от всех возможных атак.

По существу, криптографические примитивы напоминают собой или функцию *RSA*, или потенцирование в кольце вычетов. Обе эти функции являются односторонними, причем функция *RSA* — функция с секретом. При использовании примитивов в реальной схеме необходимо быть очень внимательным. Например, функция *RSA* полностью детерминирована, и при ее наивном использовании будут получаться тождественные шифротексты при шифровании тем же ключом одного сообщения дважды. Как мы убедились в главах 7

и 14, это плохое свойство. Но есть и более веские причины избегать шифрующих алгоритмов с детерминированным открытым ключом. Чтобы понять, почему, нам следует разобраться сначала в том, что означает стойкость криптосистемы. Начать надо с определения

- цели нападающего,
- соответствующих типов атак,
- вычислительной модели.

Мы будем обсуждать эти понятия в контексте криптосистем с открытым ключом, но аналогичные концепции есть и в симметричном шифровании.

15.1.1. Понятия стойкости

Нам надо разобраться, по существу, с тремя видами стойкости:

- теоретико-информационная или абсолютная стойкость;
- семантическая стойкость;
- полиномиальная стойкость.

Обсудим каждый из них поочередно. Эти понятия существенно более сильные, чем раскрытие секретного ключа или дешифрование криптограммы, о которых мы говорили раньше.

15.1.1.1. Теоретико-информационная стойкость. Напомним, что схема называется абсолютно стойкой (обладает теоретико-информационной стойкостью), если нападающий с неограниченными вычислительными возможностями не сможет извлечь никакой информации об открытом тексте, используя данный шифротекст. Теорема Шеннона (см. стр. 103) фактически утверждает, что ключ такой системы равен по длине самому сообщению, причем никакой ключ нельзя использовать дважды.

Очевидно, абсолютно стойкую систему невозможно создать в рамках модели шифрования с открытым ключом, поскольку шифрующий (открытый) ключ в такой модели должен использоваться долго, причем его длина мала по сравнению со средним сообщением.

15.1.1.2. Семантическая стойкость. Семантическая стойкость похожа на теоретико-информационную, но здесь предполагается, что атакующий обладает полиномиальными вычислительными возможностями. Говоря формально, при любом распределении вероятностей на пространстве сообщений все, что нападающий может вычислить за полиномиальное время о данном открытом тексте по

данному шифротексту, он может узнать и без шифротекста. Другими словами, обладание шифротекстом никак не помогает извлечь информацию о сообщении.

Приведем упрощенное формальное определение. Предположим, что мы хотим извлечь однобитовую информацию из пространства сообщений, т. е. значение фиксированной функции

$$g : M \longrightarrow \{0,1\}.$$

Допустим, что для любого m из пространства сообщений

$$P(g(m) = 1) = P(g(m) = 0) = \frac{1}{2},$$

где $P(g(m) = a)$ — вероятность события $g(m) = a$. Кроме того, будем считать, что открытые тексты и шифротексты имеют постоянную длину (в частности, количество знаков в шифротексте не даст никакой информации о подлежащем открытом тексте).

Под противником мы будем понимать алгоритм S, который по открытому ключу Y и шифротексту C, полученному с помощью секретного ключа x, соответствующего Y, пытается определить значение функции g от открытого текста m, ассоциированного с шифротекстом C. Таким образом, выходные данные алгоритма S состоят из единственного бита, равного значению $g(m)$.

Будем считать нападение успешным, если вероятность корректного определения значения $g(m)$ больше половины. Ясно, что противник может угадать ответ с вероятностью $\frac{1}{2}$ и не глядя на шифротекст. Поэтому успешным атакующим можно назвать лишь тот алгоритм, вероятность правильного ответа у которого увеличивается после изучения шифротекста. В связи с этим определим выигрыш противника по формуле:

$$\mathrm{Adv}_S = \left| P(S(C,Y) = g(d_x(C))) - \frac{1}{2} \right|,$$

где d_x обозначает расшифровывающую функцию с секретным ключом x, ассоциированным с открытым ключом Y. В этих терминах схема называется семантически стойкой, если

$$\mathrm{Adv}_S \leqslant \frac{1}{p(k)},$$

для всех противников S, всех функций g и всех многочленов $p(k)$ с достаточно большим параметром стойкости k.

15.1.1.3. Полиномиальная стойкость. Неприятность, связанная с семантической стойкостью, заключается в том, что очень

трудно проверить, обладает ли конкретная схема этим свойством. Полиномиальная стойкость, иногда называемая *неразличимостью шифрования*, является несравненно более просто проверяемым свойством. К счастью, мы покажем, что система, обладающая полиномиальной стойкостью, семантически стойка. Следовательно, чтобы проверить криптосистему на семантическую стойкость, достаточно продемонстрировать ее полиномиальную криптостойкость.

Будем говорить, что система обладает свойством неразличимости шифрования, или полиномиальной стойкостью, если никакой атакующий не сможет победить в следующей игре с вероятностью, превышающей 50%. Нападающий A, которому дана шифрующая функция f_Y, соответствующая открытому ключу Y, совершает два шага:

- *Поиск*. На этой стадии нападающий генерирует два открытых текста M_0 и M_1.
- *Гипотеза*. Нападающему дают шифротекст C_b, соответствующий одному из выбранных сообщений M_b, причем скрывают от него, какому именно. Цель нападающего — угадать значение b с вероятностью, большей 50%.

Заметим, что отсюда следует вероятностная природа полиномиально стойкой шифрующей функции, поскольку в противном случае атакующий может вычислить

$$C_1 = f_Y(M_1)$$

и сравнить полученное значение с C_b. Следовательно, выигрыш нападающего равен

$$\mathrm{Adv}_A = \left| P(A(\textit{гипотеза}, C_b, Y, M_0, M_1) = b) - \frac{1}{2} \right|,$$

т. к. гипотеза нападающего состоит в выборе одного из возможных значений b: 0 или 1. Схема называется полиномиально стойкой, если

$$\mathrm{Adv}_A \leqslant \frac{1}{p(k)}$$

при любом противнике A, произвольном многочлене p и достаточно большом k.

15.1.2. Виды атак

Теперь нам следует ввести различные модели атак. Существует три основных модели:

- *пассивная атака*, или атака с выбором открытого текста, *CPA* (от англ. chosen plaintext attack);
- *атака с выбором шифротекста*, *CCA1* (от англ. chosen ciphertext attack);
- *адаптивная атака с выбором шифротекста*, *CCA2* (от англ. adaptive chosen ciphertext attack).

15.1.2.1. Пассивная атака. Пассивная атака — очень слабая форма атаки. Нападающей Еве разрешается просматривать различные шифрованные сообщения. Кроме того, ей открыт доступ к «черному ящику», осуществляющему шифрование, но не расшифровывание. Таким образом, пассивная атака — простейшая модель нападения на криптосистему с открытым ключом, поскольку она предусматривает открытый доступ к функции шифрования.

15.1.2.2. Атака с выбором шифротекста. Атака с выбором шифротекста (*CCA1*), которую часто называют атакой обеденного перерыва, представляет собой несколько более сильную модель нападения. Здесь Еве предоставлен доступ к расшифровывающему «черному ящику». Она может предлагать ящику расшифровать полиномиальное число шифротекстов по ее выбору во время обеденного перерыва, пока законный пользователь аппарата подкрепляет свои силы. После этого, спустя некоторое время, Еве дают тестовый шифротекст и предлагают дешифровать его или восстановить какую-нибудь информацию о подлежащем открытом тексте собственными силами, не прибегая к помощи «черного ящика».

В контексте нашей игры в полиномиальную стойкость, описанной выше, эта атака означает, что нападающий может задавать вопросы расшифровывающему прибору на первой стадии, т. е. на стадии *поиска*, но не во время стадии *гипотезы*.

15.1.2.3. Адаптивная атака с выбором шифротекста. Адаптивная атака с выбором шифротекста (*CCA2*) — наиболее сильная форма атаки. В этом случае Еве можно пользоваться расшифровывающим прибором для расшифровывания любого шифротекста по ее выбору, кроме тестового. Принято считать, что любая вновь предлагаемая криптосистема с открытым ключом должна обладать полиномиальной стойкостью по отношению к адаптивной атаке с выбором шифротекста.

Следовательно, в нашей игре с полиномиальной стойкостью нападающий имеет право обращаться к расшифровывающему прибору на любой стадии игры. При этом ему запрещается лишь подавать

на вход прибора тестовый шифротекст C_b, иначе игра станет бессмысленной.

Сформулируем стандартное определение стойкости схемы шифрования с открытым ключом. Аналогичное определение применяется к симметричным системам шифрования.

Определение 15.1. Алгоритм шифрования с открытым ключом называют *стойким*, если он семантически стоек в отношении адаптивной атаки с выбором шифротекста.

Однако легче проверить, что данная криптосистема удовлетворяет следующему определению.

Определение 15.2. Алгоритм шифрования с открытым ключом называют *стойким*, если он полиномиально стоек в отношении адаптивной атаки с выбором шифротекста.

На самом деле эти определения связаны между собой. Докажем, например, следующий важный результат.

Теорема 15.3. При пассивном нападении полиномиально стойкая система обязана быть семантически стойкой.

Доказательство. Применим метод «от противного». Предположим, что существует алгоритм шифрования, не являющийся семантически стойким, т. е. найдется такой алгоритм S, что

$$\mathrm{Adv}_S > \frac{1}{p(k)}$$

для некоторого многочлена p и достаточно большого числа k. Покажем, что этот алгоритм не будет и полиномиально стойким. Для этого мы построим атаку A на полиномиальную стойкость схемы шифрования, использующую алгоритм S, нападающий на семантическую стойкость, как оракул.

На стадии *поиска* в атаке A генерируются два сообщения M_0 и M_1, для которых

$$g(M_0) \neq g(M_1),$$

что можно легко сделать, опираясь на упрощенное формальное определение семантической стойкости, поскольку значения функции g от сообщений распределены равномерно.

Противнику A выдается шифротекст C_b, соответствующий одному из выбранных сообщений, и предлагается определить b. На этапе *гипотеза* нападающий передает шифротекст C_b оракулу S, который находит наилучшую гипотезу для значения $g(m_b)$. Алго-

ритм A сравнивает эту гипотезу с $g(M_0)$ и $g(M_1)$ и определяет значение b.

Ясно, что если S достигает успеха при взломе семантической стойкости схемы, то A удается взломать ее полиномиальную стойкость. Поскольку

$$P(A(\textit{гипотеза}, C_b, Y, M_0, M_1) = b) = P(S(C, Y) = g(d_x(C))),$$

имеет место неравенство

$$\mathrm{Adv}_A = \mathrm{Adv}_S > \frac{1}{p(k)}.$$

Следовательно, полиномиальная стойкость влечет семантическую. ■

Заметим, что если брать более сложное определение семантической стойкости, то можно показать, что при пассивной атаке понятия семантической и полиномиальной стойкости эквивалентны.

15.1.3. Другие концепции стойкости

15.1.3.1. Жесткость. Говорят, что схема шифрования является жесткой, если по данной паре (открытый текст, шифротекст) (m, C) невозможно найти корректный шифротекст C' для «связанного» сообщения M', не зная M. Заметим, что смысл слова «связанный» здесь не совсем ясен и может варьироваться в зависимости от целей. Концепция жесткости приобретает важное значение в свете результата, который мы докажем неформально, основываясь на нашем не вполне строгом определении жесткости.

Лемма 15.4. Нежесткая схема шифрования нестойка в отношении адаптивной атаки с выбором шифротекста.

Доказательство. Пусть наша схема не является жесткой. Заменим данный нам шифротекст C_b на связанный с ним C'_b. При этом соответствующие открытые тексты M_b и M'_b тоже должны быть связанными друг с другом. После этого нападающий может потребовать от оракула дешифровать C'_b и раскрыть M'_b, что было разрешено в нашей предыдущей игре. Затем, зная M'_b, нападающий может узнать и M_b. ■

Позже мы увидим, что практически все схемы шифрования с открытым ключом, с которыми мы уже встречались, лишены свойства жесткости. Известно, однако, что жесткая схема в отношении атаки *CCA2* является и полиномиально стойкой относительно нее, и наоборот.

15.1.3.2. Текстозависимость. Свойством текстозависимости обладают наиболее стойкие схемы. Схема называется *текстозависимой*, если с вычислительной точки зрения практически невозможно построить корректный шифротекст, не имея под рукой соответствующего открытого текста. Следовательно, текстозависимость схемы влечет, что атака *CCA* на нее не может достичь успеха. Поскольку создание шифротекста требует информации об открытом тексте, Вам просто нечего будет подать на вход расшифровывающего оракула.

Понятие текстозависимости было определено в контексте модели *случайного оракула*. В этой модели предполагается, что существуют идеализированные хэш-функции, которые являются односторонними и

- вычисляют значения за полиномиальное время;
- наблюдатель не может их отличить от любых других случайных функций.

К модели случайного оракула прибегают в некоторых доказательствах стойкости. Эти доказательства ничего нам не говорят о реальной стойкости схемы, которую мы рассматриваем, но показывают, что любая реальная атака на нее должна использовать фактическое определение встроенной в схему хэш-функции. Обычная практика заключается в том, что мы доказываем стойкость протокола с помощью модели случайного оракула, а затем превращаем его в реальный протокол, заменяя случайного оракула на хэш-функцию типа *SHA-1* или *MD5*.

15.2. Стойкость актуальных алгоритмов шифрования

В этом параграфе мы покажем, что ни *RSA*, ни Эль-Гамаль, алгоритмы шифрования с открытым ключом, не удовлетворяют строгим требованиям семантической стойкости в отношении адаптивной атаки с выбором шифротекста. Это удивительно, поскольку мы утверждали, что *RSA* — наиболее используемый и важный алгоритм шифрования. Однако мы не показывали, как именно *RSA* применяется в реальных задачах криптографии. К этому моменту мы лишь дали простое математическое описание алгоритма. Но теперь мы уже знаем гораздо больше о криптографии. Поэтому, оставив математику позади, сосредоточим внимание на инженерном воплощении алгоритмов шифрования.

Для облегчения изложения будем учитывать эквивалентность понятий семантической стойкости и неразличимости шифрования (полиномиальной стойкости).

15.2.1. *RSA*

Лемма 15.5. *RSA* не обладает полиномиальной стойкостью.

Доказательство. Предположим, атакующий знает, что пользователь зашифровал только одно из двух сообщений,

$$M_1 \quad \text{или} \quad M_2.$$

Они могут означать: покупаю или продаю, да или нет, и т. д. Предполагается, что атакующему известен открытый ключ пользователя, а именно N и E. Получив шифротекст C, нападающий стремится определить, с каким из двух сообщений M_1 или M_2 совпадает подлежащий открытый текст m. Для решения этой задачи ему достаточно лишь вычислить

$$C' = M_1^E \pmod N.$$

Тогда

- если $C' = C$, то атакующий получает равенство $m = M_1$;
- если $C' \neq C$, то можно заключить, что $m = M_2$. ■

Очевидно, проблема связана с доступом противника к шифрующей функции, ведь как-никак, это схема с открытым ключом.

Предположим теперь, что процедура расшифровывания не инъективна, т. е. каждый открытый текст может соответствовать большому числу шифротекстов, причем то, какой именно шифротекст получится из данного сообщения, решается шифрующей функцией в процессе работы. Тогда описанная выше атака будет неудачной. Другими словами, в целях обеспечения стойкости природа алгоритма шифрования должна быть вероятностной, а не детерминированной. Позже мы увидим вариант *RSA*-системы, обладающий таким свойством. Поэтому описанная выше атака к нему не применима. Однако детерминированная функция шифрования — не единственная проблема *RSA*.

По существу, *RSA* — нежесткая система ввиду свойства гомоморфности.

Определение 15.6. (Свойство гомоморфности.) Имея зашифрованные сообщения m_1 и m_2, можно определить шифротекст, соответствующий произведению $m_1 \cdot m_2$, не зная самих сообщений.

Тот факт, что *RSA* обладает свойством гомоморфности, вытекает из уравнения:

$$(m_1 \cdot m_2)^E \pmod N = \left(\left(m_1^E \pmod N\right) \cdot \left(m_2^E \pmod N\right)\right) \pmod N.$$

Опираясь на это свойство, можно показать, что *RSA* не стойка в отношении адаптивной атаки с выбором шифротекста.

Лемма 15.7. *RSA* не является *CCA2*-стойкой.

Доказательство. Предположим, что Ева намерена взломать шифротекст

$$C = m^E \pmod N.$$

Она создает «связанный» шифротекст $C' = 2^E C$ и требует от своего оракула расшифровать C'. При этом получается некоторое сообщение M', после чего достаточно провести следующие вычисления:

$$\frac{M'}{2} = \frac{C'^d}{2} = \frac{(2^E C)^d}{2} = \frac{2^{Ed} C^d}{2} = \frac{2m}{2} = m. \qquad \blacksquare$$

15.2.2. Эль-Гамаль

Напомним, что проблема выбора Диффи–Хеллмана (известная как *ПВДХ*) состоит в определении истинности равенства

$$x \cdot y = z \pmod{\#\langle G\rangle}$$

по данным элементам G^x, G^y и G^z из циклической группы $\langle G\rangle$.

Лемма 15.8. Если *ПВДХ* — трудная задача в группе $\langle G\rangle$, то криптосистема Эль-Гамаль обладает полиномиальной стойкостью относительно пассивного нападения.

Доказательство. Допустим, что у нас есть полиномиальный алгоритм A, нарушающий полиномиальную стойкость системы Эль-Гамаль. Опираясь на алгоритм A, найдем решение задачи *ПВДХ*. Такую технику мы уже применяли раньше, когда использовали алгоритм, взламывающий Эль-Гамаль (дешифрующий шифротекст), для решения вычислительной задачи Диффи-Хеллмана. Сейчас, при более ограничительном определением стойкости, а именно, полиномиальной стойкости, мы можем только свести задачу взлома системы к решению (предположительно) более легкой проблемы.

Напомним, что алгоритм A проходит две стадии:

- Стадия *поиска*, имеющая на входе открытый ключ, выдающая два сообщения и некоторую дополнительную информацию.

- Стадия *гипотезы*, на вход которой подается шифротекст, открытый ключ, два сообщения и некая дополнительная информация. При этом требуется узнать, какое из сообщений соответствует данному шифротексту.

Кроме того, важно помнить, что шифротекст в Эль-Гамаль имеет вид

$$(G^k, m \cdot H^k),$$

где k — эфемерный, меняющийся с каждым сообщением, секретный ключ, а H — открытый ключ.

Наш алгоритм, решающий *ПВДХ*, работает следующим образом:

1. Входными данными служат G^x, G^y и G^z.
2. Положим $H = G^x$.
3. $(M_0, M_1, S) = A(\textit{поиск}, H)$.
4. Определим $C_1 = G^y$.
5. Выберем случайным образом $B \in \{0, 1\}$.
6. Положим $C_2 = M_B \cdot G^z$.
7. $B' = A(\textit{гипотеза}, (C_1, C_2), H, M_0, M_1, S)$.
8. Если $B = B'$, то напечатать ИСТИНА.
9. В противном случае напечатать ЛОЖЬ.

Чтобы показать, что этот алгоритм действительно решает *ПВДХ*, приведем следующие аргументы.

- Если $z = x \cdot y$, то зашифрованные данные, поданные на вход стадии *гипотезы* алгоритма A, будут представлять сообщение M_B. Следовательно, если алгоритм A может взломать семантическую стойкость системы Эль-Гамаль, то выходные данные B' будут верными и алгоритм напечатает ИСТИНА.
- Предположим теперь, что $z \neq x \cdot y$. В этом случае на вход стадии *гипотеза* почти наверняка попадут неверные данные, т. е. шифротекст не будет соответствовать ни M_0, ни M_1. Следовательно, то, что получится на выходе этой стадии, т. е. B', никак не будет зависеть от B. Значит, алгоритм после окончания работы напечатает ИСТИНА или ЛОЖЬ с одинаковой вероятностью.

Повторяя алгоритм несколько раз, мы получим вероятностный полиномиальный алгоритм решения *ПВДХ*. Но мы предположили, что такого алгоритма не существует. Поэтому и алгоритм A — мистификация. Таким образом, предположение о сложности *ПВДХ* влечет невозможность успешных атак на полиномиальную стойкость Эль-Гамаль с выбором открытого текста. ■

Однако, несмотря на семантическую стойкость в отношении атак с выбором открытого текста, Эль-Гамаль — нежесткая криптосистема.

Лемма 15.9. Эль-Гамаль — нежесткая криптосистема.

Доказательство. Предположим, что Ева видит шифротекст

$$(C_1, C_2) = (G^k, m \cdot H^k).$$

Тогда она может создать удовлетворительный шифротекст, соответствующий сообщению $2 \cdot m$, не зная ни m, ни эфемерного ключа k, ни секретного ключа x. Искомый шифротекст получается простым умножением второй его половины на 2:

$$(C_1, 2 \cdot C_2) = (G^k, 2 \cdot m \cdot H^k).$$ ■

Опираясь на это отсутствие жесткости, можно показать, как и в случае *RSA*, что Эль-Гамаль не стоек в отношении адаптивной атаки с выбором шифротекста.

Лемма 15.10. Эль-Гамаль не является *CCA2*-стойкой схемой.

Доказательство. Предположим, что Ева намерена взломать шифрованное сообщение

$$C = (C_1, C_2) = (G^k, m \cdot H^k).$$

Тогда она создает связанное сообщение

$$C' = (C_1, 2 \cdot C_2)$$

и требует у своего расшифровывающего оракула дешифровать C', и получить соответствующий открытый текст M'. После этого Ева вычисляет

$$\frac{M'}{2} = \frac{2C_2C_1^{-x}}{2} = \frac{2mH^kG^{-xk}}{2} = \\ = \frac{2mG^{xk}G^{-xk}}{2} = \frac{2m}{2} = m.$$ ■

15.3. Семантически стойкие системы

Мы уже видели, что *RSA* не является семантически стойкой даже в отношении пассивной атаки. Хорошо бы теперь привести пример семантически стойкой системы, основанной на чем-то близком к задаче факторизации целых чисел. Первая такая схема принадлежит Гольдвассеру и Микали, хотя ее и не используют в практических приложениях. Стойкость этой схемы основывается на сложно-

сти проблемы квадратичных вычетов. Действительно, очень трудно определить, является ли данное число a квадратичным вычетом по модулю составного N, если не знать разложения последнего на простые множители.

Напомним, что множество квадратов в группе $(\mathbb{Z}/N\mathbb{Z})^*$ обозначается через

$$Q_N = \left\{x^2 \pmod N \middle| \; x \in (\mathbb{Z}/N\mathbb{Z})^*\right\},$$

а символом J_N обозначают множество элементов, чей символ Якоби равен плюс единице, т. е.

$$J_N = \left\{x \in (\mathbb{Z}/N\mathbb{Z})^* \middle| \; \left(\frac{x}{N}\right) = 1\right\}.$$

Множество псевдо-квадратов совпадает с разностью $J_N \setminus Q_N$. Для *RSA*-подобных модулей $N = p{\cdot}q$ число элементов в J_N равно $\frac{(p-1)(q-1)}{2}$, в то время как множество Q_N насчитывает $\frac{(p-1)(q-1)}{4}$ элементов. Проблема квадратичных вычетов заключается в следующем: пусть дан $X \in J_N$, лежит ли он в Q_N? Ответить на вопрос очень трудно, в то время как убедиться в том, что $X \in J_N$ довольно легко.

Изложим систему шифрования Гольдвассера–Микали.

15.3.0.1. Генерирование ключей. В качестве секретного ключа выбираются два простых числа p и q и вычисляется открытый модуль $N = p \cdot q$. Кроме того, открытый ключ содержит целое число

$$Y \in J_N \setminus Q_N.$$

Чтобы найти Y, сначала определяются элементы $y_p \in \mathbb{F}_p^*$ и $y_q \in \mathbb{F}_q^*$, удовлетворяющие соотношениям

$$\left(\frac{y_p}{p}\right) = \left(\frac{y_q}{q}\right) = -1,$$

а потом Y вычисляется по y_p и y_q с помощью китайской теоремы об остатках. Очевидно, что полученный Y не будет лежать в Q_N, но он принадлежит множеству J_N, поскольку

$$\left(\frac{Y}{N}\right) = \left(\frac{Y}{p}\right) \cdot \left(\frac{Y}{q}\right) = \left(\frac{y_p}{p}\right) \cdot \left(\frac{y_q}{q}\right) = (-1) \cdot (-1) = 1.$$

15.3.0.2. Шифрование. В системе Гольдвассера–Микали шифруется один бит информации за один раз. Для шифрования бита b поступают следующим образом:

- берут произвольный элемент $x \in (\mathbb{Z}/N\mathbb{Z})^*$
- и вычисляют $C = Y^b x^2 \pmod N$.

Шифротекстом служит число C. Обратите внимание на то, что алгоритм весьма неэффективен, поскольку отдельный бит открытого текста занимает $\log_2 N$ бит в шифротексте.

15.3.0.3. Расшифровывание. Отметим, что любой шифротекст C принадлежит множеству J_N. Причем если бит b сообщения равен 0, то C будет квадратичным вычетом, в противном случае — квадратичным невычетом. Таким образом, для дешифрования сообщения необходимо уметь решать проблему квадратичных вычетов по модулю N. Законный пользователь, естественно, знает разложение N на простые множители. Поэтому он может вычислить символ Лежандра $\left(\frac{C}{p}\right)$. Если найденный символ равен $+1$, то C — квадратичный вычет и соответствующий бит открытого сообщения равен 0. Если же символ равен -1, то C — квадратичный невычет и соответствующий бит равен $+1$.

15.3.0.4. Доказательство стойкости. Здесь мы хотим показать, что система шифрования Гольдвассера–Микали защищена от пассивного нападения, если задача о квадратичных вычетах по модулю N является трудной.

Лемма 15.11. Если задача о квадратичных вычетах по модулю N трудная, то криптосистема Гольдвассера–Микали полиномиально стойка в отношении пассивного нападения.

Доказательство. Пусть у нас есть алгоритм A, атакующий нашу криптосистему. Покажем, что с его помощью можно решить задачу о квадратичных вычетах.

Допустим, нам дан элемент $L \in J_N$ и мы хотим определить, является ли он квадратичным вычетом, т. е. справедливо ли включение $L \in Q_N$? Поскольку наша криптосистема шифрует отдельные биты сообщения, на стадии *поиска* атакующий алгоритм A по открытому ключу (Y, N) создаст два сообщения

$$M_0 = 0 \quad \text{и} \quad M_1 = 1.$$

Сформируем шифротекст

$$C = L.$$

Заметим, что если L — квадратичный вычет, то шифротекст C будет соответствовать сообщению M_0, в противном случае C корректно отражает сообщение M_1. На стадии *гипотезы* можно потребовать от алгоритма A определить, какому именно сообщению из M_i соответствует шифротекст C, и по ответу на этот вопрос мы

легко можем установить, принадлежит элемент L множеству Q_N или нет. ■

15.3.0.5. Адаптивные противники. Заметим, что приведенные выше рассуждения ничего не говорят о том, является ли шифрующая схема Гольдвассера–Микали стойкой в отношении адаптивных нападений. Сейчас мы покажем, что она не обладает таким свойством.

Лемма 15.12. Схема шифрования Гольдвассера–Микали не стойка по отношению к адаптивной атаке с выбором шифротекста.

Доказательство. Пусть C — тестовый шифротекст, и мы хотим определить тот бит b, чьей шифрованной версией он является. Напомним, что

$$C = Y^b x^2 \pmod{N}.$$

Правила игры запрещают нам обращаться к расшифровывающему оракулу для дешифрования C, но мы можем потребовать от него расшифровать любой другой шифротекст по нашему выбору. Создадим новый шифротекст

$$C' = C \cdot Z^2 \pmod{N}$$

при некотором произвольном $Z \neq 0$. Легко видеть, что C' — шифротекст того же самого бита b. Значит, обращаясь к оракулу на законном основании, можно расшифровать C' и получить тот же открытый текст, что и для C. ■

15.4. Стойкость подписей

Существует много возможных понятий стойкости схем подписи. Как и в случае со стандартной собственноручной подписью, мы заинтересованы в том, чтобы подпись нельзя было подделать. Назовем три основных типа фальсификации подписи.

- *Полное раскрытие.* В этой ситуации злоумышленник может ставить подписи так же, как законный обладатель ключа. Такая ситуация, фактически, возникает при взломе секретного ключа и соответствует аналогичному взлому алгоритма шифрования.
- *Селективная фальсификация.* Здесь нападающий способен подделать подпись на единственном сообщении по его выбору.

Это аналогично ситуации, когда атакующий алгоритм шифрования может дешифровать одно из сообщений, но не взломать секретный ключ.

- *Экзистенциальная фальсификация.* Противник способен подделать подпись на единственном сообщении, которое может быть только случайной строкой битов. Такую ситуацию можно отождествить с семантической стойкостью криптосистемы.

На практике нас обычно волнуют селективные фальсификации. Поэтому мы стремимся к тому, чтобы подпись была стойкой именно в отношении селективной подделки. Однако нам неизвестно, как именно будет использоваться схема подписи. Не исключено, что она будет работать в протоколах запрос-ответ, где случайная строка битов подписывается различными сторонами. Следовательно, разумно настаивать на том, чтобы любая схема подписи была стойкой в отношении экзистенциальной фальсификации.

В соответствии с типом фальсификации возникают типы атак. Наиболее слабая атака производится пассивным противником, который, имея открытый ключ, пытается осуществить селективную или экзистенциальную подделку. Более сильная атака возникает со стороны адаптивного активного противника, имеющего доступ к оракулу, производящему законные подписи для данного открытого ключа. Цель активного противника самостоятельно подписать данное сообщение, не обращаясь для этого конкретного действия к подписывающему оракулу.

Определение 15.13. Схему подписи считают стойкой, если адаптивный противник не способен осуществить ее экзистенциальную фальсификацию.

Уже отмечалось, что критическим моментом в схемах подписи является использование хэш-функций. Это можно увидеть вновь, рассматривая грубый *RSA*-алгоритм подписи, определенный как

$$S = M^d \pmod{N}.$$

Осуществить здесь экзистенциальную фальсификацию с помощью пассивной атаки — дело тривиальное. Противник выбирает случайное значение S и вычисляет

$$M = S^E \pmod{N}.$$

Таким образом атакующий получает подпись S на сообщении M.

С другой стороны, активный противник легко сделает селективную фальсификацию в этой схеме. Допустим, он намерен поставить

«законную» подпись S на сообщении M. С этой целью противник генерирует случайное число $M_1 \in (\mathbb{Z}/N\mathbb{Z})^*$ и находит

$$M_2 = \frac{M}{M_1}.$$

Затем атакующий требует от оракула подписать сообщения M_1 и M_2. В результате он станет обладателем подписей S_1 и S_2, причем

$$S_i = M_i^d \pmod N.$$

Наконец, вычисляя произведение

$$S = S_1 \cdot S_2 \pmod N,$$

атакующий обретает долгожданную подпись, поскольку

$$S = S_1 \cdot S_2 \pmod N = M_1^d \cdot M_2^d \pmod N = \\ = (M_1 \cdot M_2)^d \pmod N = M^d \pmod N.$$

Краткое содержание главы

- Определение стойкости схемы может отличаться от наивного предположения.
- На сегодняшний день семантическая стойкость де-факто считается стандартным определением криптостойкости схем шифрования.
- Наличие семантической стойкости трудно доказать, но она имеет близкое отношение к более легко проверяемому свойству полиномиальной стойкости, часто называемой неразличимостью шифрования.
- При исследовании стойкости необходимо учитывать возможности противника. Нападения на схемы шифрования разделены на три типа: атака с выбором открытого текста, атака с выбором шифротекста и адаптивная атака с выбором шифротекста.
- Некоторые схемы, в частности, Эль-Гамаль и система Гольдвассера-Микали, полиномиально стойки в отношении пассивного нападения, однако не защищены в отношении активного адаптивного противника. Другие, например *RSA*, не обладают полиномиальной стойкостью даже в отношении пассивного нападения.
- Стойкость в отношении адаптивной атаки тесно связана с жесткостью криптосистемы.

– Аналогичные подходы применимы к понятиям стойкости схем подписи, где нас в первую очередь интересует понятие экзистенциальной фальсификации в результате активной атаки.

Дополнительная литература

Хорошее введение в теорию доказуемой стойкости и ее обобщения, как и фундаментальные идеи доказательств с нулевым разглашением, можно найти в книге Гольдрайха. С обзором оригинальных работ, написанных на эти темы вплоть до 1990 г., можно познакомится в статье Гольдвассера.

O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, 1999.

S. Goldwasser. *The Search for Provable Secure Cryptosystems*. In *Cryptology and Computational Number Theory*, Proc. Symposia in Applied Maths, Volume 42, 1990.

Контрольные вопросы

15.1.1. Что означает термин «семантическая стойкость» и какое отношение он имеет к абсолютной стойкости?

15.1.2. Дайте определение полиномиальной стойкости.

15.1.3. Какие ограничения накладываются на расшифровывающий оракул в адаптивной атаке с выбором шифротекста?

15.1.4. Назовите две проблемы, связанные с шифрованием *RSA*, которые были описаны в предыдущих главах.

15.1.5. В контексте схемы шифрования Гольдвассера–Микали покажите, что если C — корректная шифрограмма бита b, то таким же свойством обладает любой шифротекст, полученный по формуле:

$$C' = C \cdot Z^2 \pmod{N} \quad \forall\, Z \neq 0.$$

15.1.6. Назовите три основные цели, которые пытается достичь нападающий на схемы цифровой подписи.

ГЛАВА 16

ТЕОРЕТИЧЕСКАЯ СЛОЖНОСТЬ

Цели главы

- Ввести концепции теории сложности, необходимые для изучения криптографии.
- Объяснить, почему теория сложности не может самостоятельно подвести к созданию стойких криптографических систем.
- Объяснить систему Меркля–Хеллмана и причины ее слабости.
- Ввести понятие битовой стойкости.
- Ввести идею случайной саморедукции.

16.1. Классы полиномиальной сложности

Наиболее распространенной ошибкой при проектировании новых криптографических схем является выбор проблемы, вся трудность которой проявляется лишь в частных случаях, вместо такой задачи, которая была бы сложной в средней ситуации. Чтобы разобраться в этом замечании более подробно, нам необходимо усвоить некоторые из основных идей теории сложности.

Напомним, что *проблемой выбора* или *задачей принятия решений* ($\mathcal{DP}$) называется задача о выборе положительного или отрицательного ответа на вопрос I (который обычно называют *запросом*), закодированный каким-либо способом (например, как строка битов определенного размера n). Часто имеют в виду некоторое множество S и задают вопрос вида: верно ли, что $I \in S$? Например:

- I — целое число, а S — множество всех простых чисел. В этой ситуации проблема выбора звучит так: является ли данное число простым или нет?
- I — граф, а S — множество всех графов, которые можно раскрасить только k красками. Здесь проблема выбора сводится к выяснению: можно ли данный граф раскрасить по определенным правилам, используя ровно k красок? Напомним, что

граф, состоящий из конечного множества вершин V и конечного множества ребер E, считается раскрашенным k красками, если каждая его вершина покрашена одной из k красок так, что любая пара вершин, соединяющаяся ребром, имеет разный цвет.

Коль скоро мы заговорили о проблеме выбора, полезно переформулировать стандартные вычислительные проблемы в этих терминах. В качестве примера рассмотрим важную с точки зрения криптографии задачу о рюкзаке.

Определение 16.1. (**Проблема выбора в задаче о рюкзаке.**) Пусть дан набор из N предметов, каждый из которых имеет определенный вес W_i. Можно ли сложить некоторые из предметов в рюкзак так, чтобы общий вес рюкзака составил S? Иными словами, *можно* ли найти последовательность $\{b_i\}$, состоящую из нулей и единиц, для которой

$$S = b_i W_1 + b_2 W_2 + \cdots + b_N W_N?$$

Заметим, что время, требуемое на решение задачи о рюкзаке, в наихудшем случае растет как экспонента от числа предметов N.

Как было отмечено, задача о рюкзаке является проблемой выбора. А что если нам потребуется алгоритм вычисления элементов последовательности b_i?

Определение 16.2. (**Задача о рюкзаке.**) Пусть дан набор из N предметов, каждый из которых имеет определенный вес W_i. Требуется сложить некоторые из предметов в рюкзак так, чтобы общий вес рюкзака составил S. Иными словами, *нужно* найти последовательность $\{b_i\}$, состоящую из нулей и единиц, для которой

$$S = b_i W_1 + b_2 W_2 + \cdots + b_N W_N?$$

Предполагается, что если такая последовательность и существует, то только одна.

Имея оракула, решающего проблему выбора в задаче о рюкзаке, можно построить алгоритм решения задачи в классической постановке. Соответствующий алгоритм приведен ниже, где оператор `O(W[1],...,W[n],S)` обозначает оракула, решающего проблему выбора.

```
if (O(W[1],..., W[n],S)== False)
   { Output False; }
T=S;
```

```
b[1] = b[2] = ... = b[n] = 0.
for (i=1; i<=n; i++)
   { if (T==0)
        { Output (b[1], ..., b[n]). }
     if O(W[i+1], . . . , W[n],T-W[i])==True)
       { T=T-W[i]; b[i]=1; }
    }
```

Говорят, что проблема выбора $\mathcal{DP}$ принадлежит классу задач сложности $\mathcal{P}$, если существует алгоритм, отвечающий «да» на любой запрос I (на который должен последовать положительный ответ) за полиномиальное время. Мы измеряем время в терминах битовых операций, и полиномиальность означает, что число битовых операций ограничено некоторой функцией, полиномиально зависящей от битового размера запроса I. Если ответ на запрос должен быть отрицательным, то от алгоритма вообще не требуется ответа, но если он все же последует, то должен быть верным.

Заменяя в предыдущем определении положительный ответ на отрицательный, мы получим класс задач со $-\ \mathcal{P}$. К нему относятся те проблемы выбора, для которых существует алгоритм, идентифицирующий запросы с подразумеваемым отрицательным ответом за полиномиальное время.

Лемма 16.3.

$$\mathcal{P} = \text{со} - \mathcal{P}.$$

Доказательство. Пусть A — алгоритм, правильно отвечающий на запрос I, ответ на который должен быть положительным, за время n^c для некоторой константы c. Трансформируем его в полиномиальный алгоритм, дающий верный ответ на запрос с подразумеваемым отрицательным ответом за время $n^c + 1$. Для этого мы запустим алгоритм A и если он не даст ответа на запрос в течении n^c шагов, то остановим его и дадим ответ «нет». ■

К проблемам класса сложности $\mathcal{P}$ относятся те, для решения которых существует эффективный алгоритм. Другими словами, это задачи, в которых легко проводить вычисления. Например,

- проверка равенства $z = x \cdot y$ для данных целых чисел x, y и z принадлежит классу $\mathcal{P}$, если умножение — легкая операция;
- проверка соответствия данного шифротекста C ключу k и сообщению m в любимой Вами криптосистеме.

В последнем примере, естественно, предполагается, что алгоритмы

шифрования и расшифровывания в криптосистеме осуществляются за полиномиальное время. Если же Ваш любимый шифрующий алгоритм требует большего времени, то возникает вопрос, зачем Вы читаете этот учебник.

Проблему выбора относят к классу $\mathcal{NP}$ недетерминированного полиномиального времени, если для любого запроса с подразумеваемым положительным ответом существует свидетельство соответствующего ответа, которое может быть проверено за полиномиальное время. Если же ответ на запрос отрицателен, то алгоритм вообще может не закончиться, но если он все же остановится, то обязан дать ответ «нет». Свидетельство стоит представлять себе как доказательство принадлежности запроса I множеству S.

Примеры задач класса $\mathcal{NP}$.

- Является ли данное число N составным? Здесь в качестве свидетеля выступает любой нетривиальный делитель числа N. Проверка осуществляется за полиномиальное время, поскольку операция деления имеет полиномиальную сложность.
- Можно ли раскрасить данный граф k красками? Найти свидетеля здесь — это предъявить правильную раскраску.
- Проблема рюкзака при данном наборе весов. Свидетель — правильная последовательность $\{b_i\}$.

Обратите внимание на то, что ни в одном из этих примеров не предполагается, что свидетеля можно найти за полиномиальное время. Единственное требование — проверка должна проводиться за полиномиальное число операций. Очевидно, имеет место включение

$$\mathcal{P} \subset \mathcal{NP}.$$

Основная проблема в области теоретической информатики — это совпадают ли множества $\mathcal{P}$ и $\mathcal{NP}$? Большинство считает, что справедлива

Гипотеза 16.4.

$$\mathcal{P} \neq \mathcal{NP}.$$

Множество $\text{co}-\mathcal{NP}$ определяется аналогично классу $\text{co}-\mathcal{P}$ как совокупность задач выбора, имеющих свидетельство отрицательного ответа, которое может быть проверено за полиномиальное время. В отличие от положения дел с парой $\text{co}-\mathcal{P}$ и $\mathcal{P}$, имеет место утверждение:

$$\text{Если } \mathcal{P} \neq \mathcal{NP}, \text{ то } \mathcal{NP} \neq \text{co}-\mathcal{NP}.$$

Поэтому стоит считать, что $\mathcal{NP} \neq \text{co}-\mathcal{NP}$.

Выясним на примере, насколько малым может быть свидетель для задачи из класса $\mathcal{NP}$. Рассмотрим задачу о множителях, т. е. вопрос о наличии нетривиальных делителей у данного числа $n \in \mathbb{Z}$. Как уже было отмечено, эта задача принадлежит классу $\mathcal{NP}$. Доказать разложимость числа можно следующими способами.

- Предъявить нетривиальный множитель. Здесь размер свидетеля равен $O(\ln n)$.
- Предъявить свидетеля Миллера–Рабина a. Согласно обобщенной гипотезе Римана справедлива оценка

$$a \leqslant O((\ln n)^2).$$

Поэтому размер свидетеля оценивается как $O(\ln \ln n)$.

Говорят, что данная проблема выбора $\mathcal{DP}$ является $\mathcal{NP}$-полной, если любая задача класса $\mathcal{NP}$ может быть сведена к ней за полиномиальное время. Другими словами, это условие означает, что

$$\mathcal{DP} \in \mathcal{P} \quad \Longrightarrow \quad \mathcal{P} = \mathcal{NP}.$$

В некотором смысле, $\mathcal{NP}$-полные задачи являются самыми трудными из имеющих решение. Существует огромное число $\mathcal{NP}$-полных задач, из которых нас интересуют всего две:

- задача о раскраске графа,
- задача о рюкзаке.

Нам известно, что задача о делимости натурального числа принадлежит классу $\mathcal{NP}$, но не известно, является ли она $\mathcal{NP}$-полной. Принято считать, что все трудные задачи, на которых основываются криптографические примитивы, например, факторизация, проблема дискретного логарифмирования, и т. д., не являются $\mathcal{NP}$-полными, хотя и лежат в классе $\mathcal{NP}$.

Отсюда можно заключить, что задача факторизации не является очень сложной проблемой по сравнению с задачами о раскраске графа или укладке рюкзака. Почему же мы не берем $\mathcal{NP}$-полные проблемы для конструирования криптографических систем? Ведь они представляют собой класс хорошо изученных трудных задач, для которых существование эффективного решения крайне сомнительно.

Как мы увидим позже, после знакомства с системой Меркля и Хеллмана, основанной на задаче о рюкзаке, идея использования $\mathcal{NP}$-полных задач для криптографических примитивов плохо реализуется. До этого момента мы лишь упоминали, что теория сложности и, в частности, $\mathcal{NP}$-полнота имеет дело с наисложнейшим случаем. Для

криптографических же целей нам удобны задачи, которые при соответствующем выборе параметров являются трудными в среднем. Как мы позже убедимся, задача об укладке рюкзака, которую ранее предлагали использовать в криптографии, всегда имеет «среднюю» сложность, и при любом выборе параметров можно найти эффективный алгоритм, который ее решает.

Закончим этот параграф иллюстрацией отличий между трудными и средними задачами на примере задачи о раскраске графа тремя красками. Хотя в наихудшем случае задача о возможности раскраски графа тремя красками является $\mathcal{NP}$-полной, в среднем случае ее решение не вызывает больших затруднений. Связано это с тем, что средний граф, вне зависимости от его величины, не удается раскрасить в три цвета. Мы приведем алгоритм, который для почти любого графа даст отрицательный ответ о возможности правильной раскраски за постоянное число операций.

- Произвольно перенумеруем вершины графа: $v_1, \ldots, v_t$.
- Обозначим цвета как 1, 2 и 3.
- Будем обходить граф согласно только что выбранной нумерации вершин.
- Проходя очередную вершину, будем красить ее в наименьший из возможных цветов (т. е. наименьший элемент из $\{1, 2, 3\}$, который не использован для окрашивания смежной вершины).
- Если дальнейшая раскраска невозможна, то возвращаемся к последней окрашенной вершине и окрашиваем ее в следующий доступный цвет.
- Если будут исчерпаны все возможности окраски первой вершины, то алгоритм останавливается и делается вывод: данный граф невозможно раскрасить тремя красками.
- Если, следуя алгоритму, раскрашены все вершины, то делается вывод о том, что данный граф можно раскрасить тремя красками.

Интересно, что среднее число вершин случайного графа, через которые пройдет алгоритм в процессе работы, не зависит от числа всех вершин в графе и равно 197.

16.2. Криптосистемы, основанные на задаче о рюкзаке

В основу одной из наиболее ранних криптосистем с открытым ключом была заложена задача о рюкзаке, которая считалась трудной

при общем выборе начальных данных. Фактически, она принадлежит классу $\mathcal{NP}$-полных задач, однако, можно показать, что криптосистема, основанная на ней, не является стойкой.

Идея состоит в выборе двух наборов параметров задачи о рюкзаке: открытый набор делает задачу трудноразрешимой, а секретный — легкой. Кроме того, должна существовать некоторая дополнительная функция-ловушка, с помощью которой трудная задача трансформируется в легкую.

Такой подход сильно напоминает предположения в криптосистеме *RSA*: очень трудно извлечь корень степени e из заданного числа по составному модулю, но задача становится почти тривиальной, если модуль — простое число. Обладание дополнительной информацией, а именно, разложением модуля криптосистемы на простые делители, позволяет свести сложную задачу к легкой. Однако существенное различие между задачей *RSA* и проблемой рюкзака состоит в том, что при подходящем выборе разложение модуля на множители — задача трудная в среднем, а найти параметры для трудной в среднем проблемы рюкзака очень сложно, несмотря на то, что общая задача о рюкзаке труднее общей задачи факторизации.

В то время, как общая задача о рюкзаке очень сложна, существует большой класс начальных данных, при которых задача об укладке решается быстро. Эти параметры относятся к так называемым *прогрессивно возрастающим* последовательностям. Параметры для простых задач выбираются так, чтобы каждый из следующих весов был больше суммы всех предыдущих, т. е.

$$w_i \geqslant \sum_{j=1}^{i-1} w_j.$$

Например, последовательность

$$\{2, 3, 6, 13, 27, 52\}$$

обладает этим свойством. Можно взять и такие веса:

$$\{1, 2, 4, 8, 16, 32, 64, ...\}.$$

Если веса в задаче о рюкзаке обладают этим свойством, то выбор очередного веса, который следует положить в рюкзак — линейная операция. Продемонстрируем это на алгоритме:

```
for (i=n; i>=1; i--)
   { if (S>=w[i])
     { b[i]=1;
```

```
        S=S-w[i];
      }
      else
        { b[i]=0; }
    }
if (S==0) then
     напечатать (b[1],b[2],..., b[n])
else
     напечатать «нет решения»
```

Криптосистема Меркля–Хеллмана в качестве секретного ключа выбирает задачу о рюкзаке с прогрессивно возрастающей последовательностью весов и по ней (используя секретное преобразование) формулирует «сложную» задачу о рюкзаке — открытый ключ. Секретное преобразование опирается на скрываемые взаимно простые натуральные числа n и m. Оно заключается в умножении всех весов прогрессивно возрастающей последовательности на $n \pmod m$. Пусть, например, секретный ключ состоит из последовательности[1]

$$\{\mathbf{2}, \mathbf{3}, \mathbf{6}, \mathbf{13}, \mathbf{27}, \mathbf{52}\},$$

$n = \mathbf{31}$ и $m = \mathbf{105}$. Соответствующий открытый ключ — «сложная» задача о рюкзаке с весами

$$\{62, 93, 81, 88, 102, 37\}.$$

Считается, что только знающий числа m и n способен трансформировать «сложную» задачу о рюкзаке в простую.

Чтобы зашифровать сообщение, Боб разбивает открытый текст на блоки, размер которых совпадает с числом весов в задаче, и складывает те веса, у которых соответствующие биты в блоке равны 1. Пусть, например, открытый текст имеет вид:

$$\text{СООБЩЕНИЕ} = \mathbf{011000\ 110101\ 101110}.$$

Первый блок сообщения — **011000** говорит о том, что нужно сложить второй и третий веса. При этом получится $93 + 81 = 174$ — первое число шифротекста. Второй блок, **110101**, диктует сложить веса с номерами 1, 2, 4 и 6: $62 + 93 + 88 + 37 = 280$. Наконец, сумма, соответствующая последнему блоку, равна $62 + 81 + 88 + 102 = 333$. Таким образом, Боб получает шифротекст:

$$174, 280, 333.$$

[1] Чтобы подчеркнуть секретный характер информации, все секретные цифры набраны полужирным шрифтом. — *Прим. перев.*

Законный пользователь системы осведомлен о секретном ключе n, m и последовательности $\{\mathbf{2}, \mathbf{3}, \mathbf{6}, \mathbf{13}, \mathbf{27}, \mathbf{52}\}$. Поэтому, умножая каждый блок шифротекста на $n^{-1} \pmod m$, он сводит сложную задачу о рюкзаке к простой. В нашем примере $n^{-1} = 61 \pmod m$, откуда

$$174 \cdot \mathbf{61} = \mathbf{9} = \mathbf{3} + \mathbf{6} = \mathbf{011000},$$
$$280 \cdot \mathbf{61} = \mathbf{70} = \mathbf{2} + \mathbf{3} + \mathbf{13} + \mathbf{52} = \mathbf{110101},$$
$$333 \cdot \mathbf{61} = \mathbf{48} = \mathbf{2} + \mathbf{6} + \mathbf{13} + \mathbf{27} = \mathbf{101110}.$$

Процесс расшифровывания опирается на простую задачу о рюкзаке с весами $\{\mathbf{2}, \mathbf{3}, \mathbf{6}, \mathbf{13}, \mathbf{27}, \mathbf{52}\}$ и приведенный выше алгоритм ее решения.

Конечно, в нашем простом примере мы использовали всего 6 предметов. Обычно последовательность весов содержит 250 значений. Параметры m и n криптосистемы выбираются из 400-битовых чисел. Но даже увеличив параметры, мы не получим стойкой криптосистемы, поскольку она может быть взломана с помощью приведенных базисов решеток, используя метод, который мы сейчас объясним.

Определим плотность множества весов $\{w_1, \ldots, w_n\}$ в задаче о рюкзаке формулой:

$$d = \frac{n}{\max\{\log_2 w_i \mid 1 \leqslant i \leqslant n\}}.$$

Покажем, что задача о рюкзаке с небольшой плотностью легко решается с помощью приведения базисов. Так как конструкция Меркля – Хеллмана всегда дает последовательность весов с низкой плотностью, то соответствующую криптосистему несложно взломать.

Итак, пусть $\{W_1, \ldots, W_N\}$ — веса предметов и S — окончательный вес рюкзака, который нужно получить. Рассмотрим $N+1$-мерную решетку L, порожденную столбцами матрицы

$$A = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 1/2 \\ 0 & 1 & 0 & \ldots & 0 & 1/2 \\ 0 & 0 & 1 & \ldots & 0 & 1/2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 1/2 \\ W_1 & W_2 & W_3 & \ldots & W_N & S \end{pmatrix}.$$

Предположим, что строка бит $(b_1, \ldots, b_N)$ — решение нашей задачи. Вектор

$$y = A \cdot x$$

при $x = (b_1, \ldots, b_N, -1)$ принадлежит решетке. Его координаты

имеют вид

$$y_i = \begin{cases} b_i - \frac{1}{2}, & 1 \leqslant i \leqslant N, \\ 0, & i = N+1. \end{cases}$$

Значит, вектор y имеет очень маленькую длину, поскольку

$$||y|| = \sqrt{y_1^2 + \cdots + y_{N+1}^2} = \frac{\sqrt{N}}{2}.$$

Решетка, построенная по задаче о рюкзаке с низкой плотностью, как правило, обладает относительно большим детерминантом. Поэтому y — хороший кандидат на наименьший вектор в решетке. Применив LLL-алгоритм к матрице A, получим матрицу A' приведенного базиса. Ее первый столбец A'_1 тоже обычно имеет наименьшую длину. Следовательно, вероятность равенства $A'_1 = y$ весьма высока. Получив y, мы сможем найти x и решить исходную задачу о рюкзаке.

В качестве простого примера дешифруем сообщение, приведенное на стр. 400, т. е. найдем последовательность битов $\{b_i\}$, удовлетворяющую соотношению

$$b_1 62 + b_2 93 + b_3 81 + b_4 88 + b_5 102 + b_6 37 = 174.$$

Сформируем матрицу

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1/2 \\ 62 & 93 & 81 & 88 & 102 & 37 & 174 \end{pmatrix}.$$

Применив к ней LLL-алгоритм, построим новый базис решетки с матрицей

$$A' = \frac{1}{2}\begin{pmatrix} 1 & -1 & -2 & 2 & 3 & 2 & 0 \\ -1 & -3 & 0 & -2 & -1 & -2 & 0 \\ -1 & -1 & -2 & 2 & -1 & 2 & 0 \\ 1 & -1 & -2 & 0 & -1 & -2 & -2 \\ 1 & -1 & 0 & 2 & -3 & 0 & 4 \\ 1 & 1 & 0 & -2 & 1 & 2 & 0 \\ 0 & 0 & -2 & 0 & 0 & -2 & 2 \end{pmatrix}.$$

Обозначим ее первый столбец через y,

$$y = \frac{1}{2}\begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

и найдем x по формуле

$$x = A^{-1} \cdot y = \frac{1}{2}\begin{pmatrix} 0 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Таким образом, последовательность битов

$$(b_1, b_2, b_3, b_4, b_5, b_6) = (0, 1, 1, 0, 0, 0)$$

является решением задачи о рюкзаке и совпадает с первым блоком открытого текста.

16.3. Битовая стойкость

Ранее (см. стр. 393) мы ввели понятие проблемы выбора как задачи с односложным ответом: «да» или «нет». Мы показали, что некоторые другие задачи, например, задача о рюкзаке, сводится к такой проблеме. Аналогичная ситуация возникает в криптографии, где хотелось бы знать, является ли задача вычисления одного бита сообщения столь же сложной, как и задача вычисления всего сообщения.

Предположим, что используется *RSA*-функция

$$x \mapsto Y = x^E \pmod{N}.$$

Может возникнуть ситуация, когда противника волнует лишь вычисление $B = x \pmod 2$, а не полное восстановление x. Нам хотелось бы, чтобы решение этого уравнения было бы сравнимо по сложности с полным восстановлением сообщения. Другими словами, нам нужно исследовать так называемую битовую стойкость функции *RSA*.

Легко убедиться, что битовая стойкость тесно связана с семантической: если нападающий сможет определить четность открыто

текста, имея только соответствующий шифротекст, то семантическая стойкость алгоритма шифрования будет взломана.

Нам понадобится следующее понятие.

Определение 16.5. Пусть $f : S \longrightarrow T$ — одностороннее отображение конечных множеств S и T, а $B : S \longrightarrow \{0,1\}$ — булева функция (называемая *предикатом*). Предикат $B(x)$ называется *сильным* для f, если по данному $x \in S$ значение $B(x)$ вычисляется легко, а по $f(x)$ — очень трудно.

Один из способов проверки этого свойства предиката B в отношении односторонней функции f заключается в том, чтобы предположить существование оракула, вычисляющего $B(x)$ по данному элементу $f(x)$, и показать, что соответствующий оракул фактически обращает функцию f.

Понятие k-битового предиката и сильного k-предиката определяется аналогично, только значения он принимает в строках битов длины k, а не 1, как это было в предыдущем определении. Мы хотели бы показать, что различные предикаты, которые можно определить для полезной с точки зрения криптографии односторонней функции f, фактически являются сильными.

16.3.1. Сильные предикаты для дискретных логарифмов

Пусть A — конечная абелева группа простого порядка Q, порожденная элементом G. Рассмотрим предикат

$$B_2 : x \longrightarrow x \pmod 2$$

и докажем следующую теорему.

Теорема 16.6. Предикат B_2 является сильным для функции $x \mapsto G^x$.

Доказательство. Символом $\mathcal{O}(H,G)$ обозначим оракула, определяющего наименьший значащий бит дискретного логарифма от H по основанию G, т. е. вычисляющего $B_2(x)$, где $x = \log_G H$. Покажем, как использовать $\mathcal{O}$ для решения проблемы дискретного логарифмирования.

Пусть дано значение $H = G^x$. Вычислим $T = \frac{1}{2} \pmod Q$ и положим $Y = 0$, $Z = 1$. Теперь, пока H не станет равным 1, будем осуществлять следующие шаги:

- $B = \mathcal{O}(H,G)$;

- если $B = 1$, то положим $Y = Y + Z$ и $H = \frac{H}{G}$;
- переопределим $H = H^T$ и $Z = 2 \cdot Z$.

Как только H станет равным 1, мы получим, что $Y = \log_G H$. ■

Проследим за работой алгоритма на примере, взяв элемент $G = 64 \in \mathbb{F}_{607}$ порядка $Q = 111$. Предположим, что нам нужно вычислить $\log_{64} 56$. Руководствуясь алгоритмом из доказательства теоремы, заполняем табл.16.1.

Таблица 16.1. Таблица промежуточных значений алгоритма вычисления дискретного логарифма

H	$\mathcal{O}(H,G)$	Z	Y
56	0	1	0
451	1	2	2
201	1	4	6
288	0	8	6
100	1	16	22
454	0	32	22
64	1	64	86

Можно убедиться в верности найденного решения, проверив равенство:

$$64^{86} = 56 \pmod{607}.$$

16.3.2. Сильные предикаты для задачи *RSA*

Задача *RSA*, а именно, уравнение $C = m^E \pmod N$, обладает следующими сильными предикатами:

- $B_1(m) = m \pmod 2$;
- $B_h(m) = 0$, если $m < \frac{N}{2}$, $B_h(m) = 1$ в противном случае;
- $B_k(m) = m \pmod{2^k}$, где $k = O(\ln \ln N)$.

Обозначим соответствующие оракулы через $\mathcal{O}_1(C,N)$, $\mathcal{O}_h(C,N)$ и $\mathcal{O}_k(C,N)$. Мы не будем исследовать последний из них, но отметим, что первые два взаимосвязаны:

$$\mathcal{O}_h(C,N) = \mathcal{O}_1(C \cdot 2^E \pmod N, N),$$
$$\mathcal{O}_1(C,N) = \mathcal{O}_h(C \cdot 2^{-E} \pmod N, N).$$

Покажем, как используя оракулы $\mathcal{O}_h$ и $\mathcal{O}_1$, можно обратить функцию *RSA* при помощи алгоритма, основанного на стандартном двоичном поиске. Положим $Y = C$, $L = 0$ и $H = N$. Затем, до тех пор, пока $H - L \geqslant 1$, делаем следующее:

- $B = \mathcal{O}_h(Y, N)$;
- $Y = Y \cdot 2^E \pmod N$;
- $M = (H + L)/2$;
- если $B = 1$, то положим $L = M$; в противном случае положим $H = M$.

При выходе из цикла значение $\lfloor H \rfloor$ должно совпадать с прообразом C относительно функции RSA.

Разберем пример, в котором $N = 10\,403$ и $E = 7$ — открытая информация, а мы хотим найти прообраз элемента $C = 3$ относительно функции RSA, опираясь на оракул $\mathcal{O}_h(Y, N)$. Шаги алгоритма занесены в табл. 16.2.

Таблица 16.2. Таблица промежуточных значений алгоритма обращения функции RSA

Y	$\mathcal{O}(Y, N)$	L	H
3	0	0	10 403
$3 \cdot 2^7$	1	0	5201,5
$3 \cdot 4^7$	1	2600,7	5201,5
$3 \cdot 8^7$	1	3901,1	5201,5
$3 \cdot 16^7$	0	4551,3	5201,5
$3 \cdot 32^7$	0	4551,3	4876,4
$3 \cdot 64^7$	1	4551,3	4713,8
$3 \cdot 128^7$	0	4632,5	4713,8
$3 \cdot 256^7$	1	4632,5	4673,2
$3 \cdot 512^7$	1	4652,9	4673,2
$3 \cdot 1024^7$	1	4663,0	4673,2
$3 \cdot 2048^7$	1	4668,1	4673,2
$3 \cdot 4096^7$	1	4670,7	4673,2
$3 \cdot 8192^7$	0	4671,9	4673,2
–	–	4671,9	4672,5

Из последней строки таблицы следует, что прообразом элемента 3 относительно функции

$$x \longrightarrow x^7 \pmod{10\,403}$$

служит число $m = 4672$.

16.4. Случайная саморедукция

Мы уже отмечали, что недостатки криптографической схемы Меркля – Хеллмана и других криптосистем, основанных на теории слож-

ности, возникают ввиду того, что эти схемы ассоциированы с трудными в общем случае, но легкими при средних значениях параметров проблемами. Возникает естественный вопрос: откуда мы знаем, что задача *RSA* или проблема выбора Диффи–Хеллмана избавлены от этого недостатка? Не может ли так оказаться, что при известных модуле N и шифрующей экспоненте E уравнение

$$C = m^E \pmod N$$

трудноразрешимо при некоторых C, но легко решается для средних значений?

Оказывается, можно доказывать, что задача *RSA* при фиксированном модуле N и проблема выбора Диффи–Хеллмана с данной группой A являются трудными в среднем случае.

Техника доказательства основывается на случайной саморедукции, сводящей конкретные начальные условия проблемы к случайному набору исходных данных. Это означает, что возможность решения задачи в среднем случае обеспечивает нам успех и в самом наихудшем. Значит, сложность решения задачи реально не зависит от того, наихудший или средний случай Вы рассматриваете.

Лемма 16.7. Проблема *RSA* позволяет применить к ней случайную саморедукцию.

Доказательство. Пусть нам нужно решить уравнение

$$C = m^E \pmod N$$

относительно m, причем считается, что C здесь относится к «наиболее сложным» исходным данным. Сведем уравнение к «среднему» случаю, выбрав произвольным образом $S \in (\mathbb{Z}/N\mathbb{Z})^*$ и положив

$$C' = S^E C.$$

Теперь мы пытаемся решить уравнение

$$C' = {m'}^E \pmod N$$

относительно неизвестной m'. Если нам не повезло, и мы опять пришли к трудному случаю, будем выбирать другие значения S до тех пор, пока не получим «среднего» условия. Предполагая простоту среднего случая, мы можем решить уравнение $C' = {m'}^E \pmod N$ относительно m' и вычислить

$$m = \frac{m'}{S},$$

что даст нам решение поставленной задачи. ■

Можно показать, что проблема выбора Диффи–Хеллмана тоже обладает свойством случайной саморедукции, в том смысле, что сложность вопроса о том, является ли тройка

$$(X, Y, Z) = (G^a, G^b, G^c),$$

тройкой Диффи–Хеллмана (т. е. верно ли, что $c = a \cdot b$), не зависит от выбора значений a, b и c. Действительно, рассмотрим тройку:

$$(X', Y', Z') = (G^{a_1}, G^{b_1}, G^{c_1}) = (X^V G^{U_1}, Y G^{U_2}, Z^V Y^{U_1} X^{VU_2} G^{U_1 U_2})$$

со случайными U_1, U_2 и V. Если (X, Y, Z) — тройка Диффи – Хеллмана, то (X', Y', Z') — тоже тройка Диффи–Хеллмана, и наоборот.

Можно показать, что для тройки Диффи–Хеллмана (X, Y, Z) ассоциированные тройки (X', Y', Z') равномерно распределены в множестве всех троек Диффи–Хеллмана. А если начальная тройка не принадлежала к числу троек Диффи–Хеллмана, то ассоциированные тройки будут равномерно распределены по множеству всех троек чисел (а не в множестве троек Диффи–Хеллмана).

16.5. Рандомизированные алгоритмы

Закончим главу обсуждением рандомизированных алгоритмов. Сначала мы сформулируем определения различных типов алгоритмов, затем соотнесем их с классами сложности и, наконец, приведем несколько примеров.

Напомним, что к проблемам выбора относятся задачи, в которых спрашивается о выполнении или невыполнении какого-то конкретного свойства. Приведем определения, в которых названия типов соответствуют мировым центрам азартных игр.

Алгоритм Монте-Карло.

- Если ответ в проблеме выбора отрицателен, то алгоритм всегда выдает ответ «нет».
- Алгоритм выдает положительный ответ с вероятностью больше или равной $\frac{1}{2}$.
- В остальных случаях алгоритм дает отрицательный ответ, даже если истинный ответ в задаче положителен.

Алгоритм Атлантик-Сити.

- Выдает положительный ответ, причем вероятность его правильности не ниже $\frac{2}{3}$.
- Выдает отрицательный ответ с той же вероятностью правильного ответа.

Алгоритм Лас-Вегас.

- Заканчивает работу и выдает правильный ответ с вероятностью большей или равной $\frac{1}{2}$.
- В остальных случаях алгоритм не заканчивает работу.

В этих определениях мы предполагаем, что алгоритм работает в течение полиномиального времени относительно размера входных данных. Очевидно, можно обобщить эти определения и на класс задач, не относящихся к проблеме выбора.

Переключим внимание на классы сложности. Предположим, что у нас есть запрос I и возможный свидетель w, чей размер полиномиально зависит от размера запроса I. Нам нужно определить, принадлежит ли I множеству S.

Определение 16.8. Говорят, что проблема выбора принадлежит классу $\mathcal{RP}$, если найдется алгоритм A, который по введенным в него запросу I и свидетелю w осуществляет следующее:

- если $I \in S$, то по крайней мере для половины возможных свидетелей алгоритм определит, что $I \in S$;
- если $I \notin S$, то при любом свидетеле алгоритм выдаст ответ: $I \notin S$.

Поскольку из утверждения «по крайней мере для половины свидетелей алгоритм определит, что $I \in S$» непосредственно следует, что «по крайней мере для одного свидетеля алгоритм определит, что $I \in S$», имеет место включение $\mathcal{RP} \subset \mathcal{NP}$. Кроме того, очевидно, $\mathcal{P} \subset \mathcal{RP}$.

Класс задач $\mathcal{RP}$ имеет особое значение, т. к. для каждой задачи этого класса существует вероятностный алгоритм A, определяющий, принадлежит ли данный запрос I фиксированному множеству S при любом множестве S. Мы генерируем k случайных свидетелей w_i и прогоняем алгоритм $A(I, w_i)$ k раз при разных значениях i. Если этот алгоритм выдал ответ $I \in S$ хотя бы на одном проходе, можно сделать вывод, что I действительно принадлежит S. Если же каждый раз алгоритм будет выдавать сообщение $I \notin S$, то можно предположить, что $I \notin S$, причем предположение будет ошибочным только в одном случае из 2^k. Связывая это наблюдение с разбиением алгоритмов на типы, мы видим, что

если проблема выбора принадлежит классу $\mathcal{RP}$, то для нее найдется алгоритм Монте-Карло.

Определим еще один класс задач.

Определение 16.9. Проблему выбора относят к классу $\mathcal{BPP}$, если для нее найдется алгоритм A, который по введенным запросу I и свидетелю w делает следующее:

- если $I \in S$, то по крайней мере при $\frac{2}{3}$ всех возможных свидетелей алгоритм определяет, что $I \in S$;
- если $I \notin S$, то по крайней мере при $\frac{2}{3}$ всех возможных свидетелей алгоритм выдает ответ $I \notin S$.

Очевидно, имеет место утверждение:

если проблема выбора относится к классу $\mathcal{BPP}$, то для нее найдется алгоритм Атлантик-Сити.

Наконец, определим класс задач $\mathcal{ZPP}$ как

$$\mathcal{ZPP} = \mathcal{RP} \cap \text{co} - \mathcal{RP}.$$

Тогда

для любой задачи класса $\mathcal{ZPP}$ найдется алгоритм Лас-Вегас.

Имеют места следующие включения:

$$\mathcal{P} \subset \mathcal{ZPP} \subset \mathcal{RP} \subset \mathcal{NP} \cap \mathcal{BPP}.$$

Тест Ферма и тест Миллера–Рабина на простоту служат примерами алгоритма Монте-Карло, тестирующего числа на наличие нетривиальных делителей. Напомним, что упомянутые тесты по данному числу N работают так:

- если N — простое, то всегда выдают ответ ЛОЖЬ (или правдоподобно простое);
- если N — составное, то дают ответ ИСТИНА с вероятностью $\geqslant \frac{1}{2}$, в противном случае следует ответ ЛОЖЬ.

Следовательно,

задача о разложимости числа на множители лежит в $\mathcal{RP}$.

Повторяя тест Монте-Карло, мы можем приблизить вероятность правильного ответа сколь угодно близко к единице.

Алгоритм Адлемана–Хуанга доказательства простоты числа — пример алгоритма Лас-Вегас тестирования простоты. На вход этого алгоритма подается натуральное число N. Если оно составное,

то алгоритм может и не окончиться, но если он все же остановится, то скажет нам абсолютно верно, простое число N или нет. Значит,

задача о простоте числа принадлежит классу $\mathcal{ZPP}$.

Появившийся до Адлемана и Хуанга алгоритм доказательства простоты, основанный на эллиптических кривых, не гарантирует конечного времени работы при произвольном простом p, но практически всегда это время конечно.

Краткое содержание главы

- Теория сложности имеет дело с самыми плохими входными данными алгоритма, решающего данную проблему выбора.
- Некоторые задачи решаются легко при средних начальных данных, однако существуют такие условия, при которых их чрезвычайно трудно решить. Не стоит основывать криптосистемы на такого сорта задачах.
- Криптографические системы, основанные на задаче о рюкзаке, стали притчей во языцах ввиду того, что их легко взломать, опираясь на приведенные базисы решеток.
- Задачи, которые мы кладем в основу криптосистем с открытым ключом, такие как задача *RSA* или проблема дискретного логарифмирования, обладают тем свойством, что вычисление даже одного бита из ответа настолько же сложно, насколько трудно узнать весь ответ.
- Такие задачи, как *RSA* и проблема выбора Диффи–Хеллмана, являются трудными в среднем случае, поскольку опираясь на случайную саморедукцию можно свести задачу с фиксированными исходными данными к задаче со случайными начальными данными.

Дополнительная литература

Хорошее введение в теорию сложности можно найти в главе 2 Бэча и Шаллита. Обсуждение связей между теорией сложности и такими криптографическими концепциями как доказательство с нулевым разглашением, представлено книгой Гольдрайха. О криптосистемах, основанных на задаче о рюкзаке, и методах их взлома с помощью решеток рассказано в обзорной статье Одлыжко.

E. Bach and S. Shallit. *Algorithmic Number Theory, Volume 1: Efficient Algorithms*. MIT Press, 1996.

O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, 1999.

A. Odlyzko. *The Rise and Fall of Knapsack Cryptosystems*. In *Cryptology and Computational Number Theory*, Proc. Symposia in Applied Maths, Volume 42, 1990.

Контрольные вопросы

16.1.1. Дайте определение классов сложности $\mathcal{P}$, $\mathcal{NP}$ и $\text{co}-\mathcal{NP}$.

16.1.2. Какое распределение весов в задаче о рюкзаке называют прогрессивно возрастающим?

16.1.3. Как в криптосистеме Меркля–Хеллмана задача о рюкзаке с прогрессивно возрастающими весами транслируется в трудную задачу о рюкзаке?

16.1.4. Что подразумевают под сильным предикатом относительно функции и как это понятие связано с семантической стойкостью?

16.1.5. Чем отличаются типы алгоритмов Монте-Карло, Атлантик-Сити и Лас-Вегас?

16.1.6. Выясните, к какому из трех типов алгоритмов: Монте-Карло, Атлантик-Сити или Лас-Вегас относятся
(а) ρ-метод Полларда;
(б) тест на простоту Миллера–Рабина;
(в) алгоритм Шэнкса извлечения квадратного корня в $\mathbb{F}_p$.

Лабораторные работы

16.2.1. Реализуйте схему Меркля–Хеллмана и атаку на нее, основанную на решетках. Исследуйте практическим путем, насколько успешна эта атака.

Упражнения

16.3.1. Проверьте справедливость следующих равенств для оракулов в задаче *RSA*:

$$\mathcal{O}_h(C, N) = \mathcal{O}_1(C \cdot 2^E \pmod N, N),$$
$$\mathcal{O}_1(C, N) = \mathcal{O}_h(C \cdot 2^{-E} \pmod N, N).$$

16.3.2. [**Предполагается владение теорией графов.**] Покажите, что рекурсивный алгоритм, определяющий, можно ли раскрасить данный средний граф в три краски (см. стр. 398), в течение работы пройдет в среднем не более чем через 197 вершин вне зависимости от общего количества вершин в графе.

16.3.3. Покажите, что в нашем определении класса $\mathcal{BPP}$ и алгоритма Атлантик-Сити можно заменить константу $\frac{2}{3}$ на любую дробь из интервала $(\frac{1}{2}, 1)$.

16.3.4. Докажите, что предположение о существовании односторонних функций влечет неравенство $\mathcal{P} \neq \mathcal{NP}$.

ГЛАВА 17

ДОКАЗУЕМАЯ СТОЙКОСТЬ СО СЛУЧАЙНЫМ ОРАКУЛОМ

Цели главы

- Описать модель случайного оракула.
- Показать, как с помощью модели случайного оракула доказывается стойкость некоторых схем подписи.
- Показать, как из атаки с выбором шифротекста вытекает необходимость снабжать шифротекст некоторой избыточностью.
- Объяснить *RSA-OAEP* и привести набросок доказательства.
- Описать, как превратить криптосистему Эль-Гамаль в стойкую схему шифрования.

17.1. Введение

Современные способы проверки стойкости протоколов основываются на доказуемой стойкости. Это несколько дезориентирующее название соответствующей техники, поскольку она реально не доказывает криптостойкости в теоретико-информационном смысле, о котором мы говорили ранее. Сторонники доказуемой стойкости стремятся показать, что алгоритм успешной атаки на криптосистему можно использовать для создания другого алгоритма, существование которого считается невозможным.

Специалисты, например, пытаются показать, что атаку с выбором шифротекста, направленную на семантическую стойкость *RSA*, можно использовать для разложения на множители любого натурального числа. Иными словами, такое доказательство — относительный результат, при котором «доказательство» стойкости привязано к трудноразрешимости задачи факторизации.

Главный вклад доказуемой стойкости в криптографию состоит в точном определении понятий стойкого шифрования и стойкой схемы подписи. Многие из концепций, введенных в этой книге, — экзистенциальная фальсификация, семантическая стойкость, неразличимость шифрования, адаптивная атака с выбором шифротекста и т. д., — появились благодаря изучению доказуемой стойкости.

Объясним технику доказуемой стойкости на конкретных примерах. Предположим, что нам дан атакующий алгоритм, взламывающий некоторую стойкость алгоритма *RSA* (например, семантическую) с определенной достаточно большой вероятностью. Однако сначала необходимо определить, что здесь означает понятие «достаточно большая вероятность».

Предположим, что у схемы есть параметр стойкости k, измеряющий, например, количество битов в ключе. В частности, параметр стойкости алгоритма *RSA* мог бы равняться числу двоичных знаков модуля N. Скажем, что противник достигает успеха с *достаточно большой* вероятностью, если вероятность достижения им своей цели больше, чем

$$\frac{1}{p(k)},$$

где $p(k)$ — некоторый многочлен, зависящий от k.

Допустим на минуту, что наш противник A пассивен, т. е. не обращается за помощью к расшифровывающему «черному ящику». Нам хотелось бы создать новый алгоритм B_A, имеющий на входе натуральное число N и вызывающий алгоритм A некоторое число раз, полиномиально зависящее от k. Цель нового алгоритма — найти простые множители числа N с достаточно большой вероятностью. Существование алгоритма B_A показало бы, что наличие пассивного алгоритма A, взламывающего *RSA*, влечет возможность разложения N на множители за полиномиальное время с достаточно большой вероятностью. Поскольку мы не верим в изобретение полиномиального алгоритма факторизации целых чисел на данном уровне развития науки, можно сделать вывод, что соответствующий алгоритм A также неосуществим на сегодняшний день.

Мы уже использовали этот прием, когда показывали, что успешная пассивная атака на криптосистему Эль-Гамаль дает эффективный алгоритм решения проблемы выбора Диффи–Хеллмана, или при демонстрации того, что успех в пассивной атаке на схему Гольдвассера – Микали дает эффективный алгоритм решения задачи о квадратичных вычетах.

Итак, по данному алгоритму A мы строим новый алгоритм B_A, использующий A как стандартную процедуру. На вход алгоритму B_A подается трудная математическая задача, которую мы намерены решить, в то время как входными данными алгоритма A служит некоторая криптографическая задача. Трудности возникают, когда A моделирует активного противника, имеющего доступ к дешифрующему или подписывающему оракулу, соответствующему данному открытому ключу. Алгоритм B_A, использующий A как подпрограмму, должен обеспечить A ответами на вопросы, которые тот задает оракулу. В этой ситуации перед алгоритмом B_A возникает несколько проблем:

- Ответы, поставляемые алгоритму A, должны быть корректными, т. е. шифротексты необходимо расшифровывать, а подписи необходимо проверять. В противном случае алгоритм A может заметить, что оракул дает лживые ответы. Следовательно, теперь у алгоритма B_A нет никаких гарантий в том, что A достигает успеха с достаточной вероятностью.
- Ответы псевдо-оракула должны быть распределены с той плотностью вероятностей, которую A ожидает от истинного оракула, поскольку в противном случае A заметит подлог.
- Отвечать необходимо на любой вопрос, который алгоритм A задает оракулу.
- Алгоритм B_A должен генерировать ответы, не прибегая к секретному ключу. Например, в случае RSA, если B_A нацелен на поиск множителей числа N, то он едва ли сможет воспользоваться этими множителями для ответов алгоритму A прежде, чем определит их.

Последняя из перечисленных проблем наиболее существенна. Фактически, мы требуем от алгоритма B_A расшифровать или подписать сообщение, не опираясь на знание секретного ключа, но именно это и считается невозможным ввиду криптостойкости схемы.

Для преодоления возникающих трудностей принято использовать так называемую «модель случайного оракула». Случайный оракул — это идеализированная хэш-функция, которая на каждый новый запрос выдает случайный ответ, равномерно распределенный по области значений, с условием: если один и тот же запрос поступит дважды, то ответ должен быть одинаковым.

В модели случайного оракула предполагается, что противник A при атаке на схему не пользуется явной хэш-функцией, определенной в этой схеме. Другими словами, противник A достигнет успеха

даже в том случае, если мы заменим настоящую хэш-функцию схемы случайным оракулом. Алгоритм B_A отвечает на запросы противника A, обманывая последнего «сфабрикованными» случайным оракулом ответами, чтобы удовлетворить свои собственные потребности. Чтобы увидеть, как это осуществляется на практике, стоит перейти к следующему разделу о доказательстве стойкости алгоритмов подписи.

Доказательство с моделью случайного оракула носит еще более относительный характер, чем предыдущие. Такие доказательства говорят, что в предположении о трудноразрешимости определенной задачи, например, факторизации, успешного нападения, не использующего подлежащей хэш-функции, быть не может. При этом, естественно, не утверждается, что нет успешных атак, которые имеют доступ к фактической хэш-функции криптосистемы.

Во всех наших доказательствах и определениях мы не слишком строги. Мы стремимся передать основные идеи и показать их особенности, не следя за особой математической точностью детального рассуждения. Читатель, желающий узнать более точные определения, может обратиться к научным статьям. Однако надо иметь в виду, что определения этой области могут незначительно меняться от статьи к статье.

17.2. Стойкость алгоритмов подписи

Сначала мы рассмотрим стойкость алгоритмов цифровой подписи, поскольку доказательства здесь более просты, чем в алгоритмах шифрования. Первый основной инструмент, о котором мы будем говорить, — это лемма, принадлежащая Стерну и Пойнтчевалю. Она применяется к определенному типу схем подписи, которые используют хэш-функции следующим образом: чтобы подписать сообщение

- производят (возможно пустое) обязательство Σ_1;
- вычисляют хэш-значение $H = H(\Sigma_1||M)$;
- находят Σ_2 — «подпись» на Σ_1 и H.

Чтобы запомнить запрос к хэш-функции, запишем выходные данные схемы в виде $(\Sigma_1, H(\Sigma_1||M), \Sigma_2)$. Например,

- *DSA*: $\Sigma_1 = \varnothing$, $H = H(M)$,

$$\Sigma_2 = \left(R, \frac{H + xR}{k} \pmod{Q}\right),$$

где $R = \left(Q^k \pmod P\right) \pmod Q$.

- *EC-DSA*: $\Sigma_1 = \varnothing$, $H = H(M)$,

$$\Sigma_2 = \left(R, \frac{H + xR}{k} \pmod Q\right),$$

где R совпадает с координатой x точки $[k]G$.

- *Схема подписи Шнорра*: $\Sigma_1 = G^k$, $H = H(\Sigma_1 || M)$

$$\Sigma_2 = xH + k \pmod Q.$$

Во всех перечисленных схемах предполагают, что хэш-функция принимает значения в $\mathbb{F}_Q$.

Напомним, что в модели случайного оракула хэш-функция позволяет алгоритму B_A генерировать произвольные ответы на запросы противника A. Предположим, что в модели случайного оракула нападающий алгоритм A может генерировать экзистенциальную фальсификацию для сообщения M с достаточно большой вероятностью. Тогда выходные данные алгоритма A имеют вид:

$$(M, \Sigma_1, H, \Sigma_2).$$

Можно считать, что противник создает критический хэш-запрос

$$H = H(\Sigma_1 || M),$$

поскольку в противном случае мы можем самостоятельно генерировать запрос нападающего.

Алгоритм B_A теперь дважды вызывает подпрограмму противника A, с неизменными случайными входными данными, но слегка измененным случайным оракулом. Противник A выполняет свою работу за полиномиальное время и при этом выдает полиномиальное количество хэш-запросов. Если на все хэш-запросы противник получает тот же самый ответ, то A должен будет выдать неизменную подпись. Однако алгоритм отвечает на все случайные запросы нападающего, как и раньше, кроме одного, выбранного случайным образом, и на него отвечает по-другому. С достаточно большой вероятностью этот «исключительный» запрос окажется критическим хэш-запросом и поэтому (с достаточно большой вероятностью) противник B_A получит две подписи на одном и том же сообщении, которые имеют разные ответы на хэш-запросы. Иными словами, мы получим

$$(M, \Sigma_1, H, \Sigma_2) \quad \text{и} \quad (M, \Sigma_1, H', \Sigma_2').$$

Теперь, опираясь на эти два набора выходных данных алгоритма

A, можно попытаться решить трудную задачу, составляющую цель алгоритма B_A. Подробности и полный разбор этого приема можно найти в работе Стерна и Пойнтчеваля (точная ссылка приведена в разделе «Дополнительная литература» в конце этой главы).

17.2.1. Примеры пассивного противника

Рассмотрим некоторые приложения.

17.2.1.1. Схема подписи Шнорра. Лемма Стерна и Пойнтчеваля позволяет доказать следующую теорему.

Теорема 17.1. Из предположения о трудноразрешимости задачи дискретного логарифмирования в данной группе вытекает (в модели случайного оракула), что успешная пассивная атака, направленная на схему подписи Шнорра с данной группой, невозможна.

Доказательство. Пусть входными данными в алгоритм B_A служит задача дискретного логарифмирования $Y = G^x$, которую мы намерены решить. Предположим, что мы вызываем процедуру пассивного противника A, обладающего открытым ключом Y в качестве входных данных, и попытаемся использовать аргументы леммы Стерна и Пойнтчеваля. С достаточно большой вероятностью мы получим две подписи

$$\left(M, \Sigma_1 = G^k, H, \Sigma_2 = xH + k \pmod{Q}\right)$$

и

$$\left(M, \Sigma_1' = G^{k'}, H', \Sigma_2' = xH' + k' \pmod{Q}\right),$$

где $H = H(\Sigma_1 || M)$ — ответ случайного оракула при первом проходе алгоритма A, а $H' = H(\Sigma_1' || M)$ — при втором.

Цель алгоритма B_A — найти неизвестное число x. Он делает вывод: $k = k'$, поскольку $\Sigma_1 = \Sigma_1'$. Следовательно,

$$Cx = D \pmod{Q},$$

где

$$C = H - H' \pmod{Q}, \qquad D = \Sigma_2 - \Sigma_2' \pmod{Q}.$$

Кроме того, известно, что $A \neq 0$, поскольку в противном случае два хэш-значения были бы равны между собой. Следовательно, алгоритм B_A может решить исходное логарифмическое уравнение, вычисляя

$$x = C^{-1}D \pmod{Q}.$$

■

Позже мы покажем, что схема подписи Шнорра в модели случайного оракула защищена и от активного нападения.

17.2.1.2. DSA-подписи. Как мы сейчас покажем, приведенные в предыдущем разделе рассуждения не применимы для схемы подписи *DSA*.

Пусть входными данными B_A является уравнение $Y = G^x$ относительно неизвестной x, которое мы намерены решить. Опять будем вызывать процедуру противника A, на вход которого подается открытый ключ Y, и будем применять лемму Стерна и Пойнтчеваля. С достаточно большой вероятностью мы получим подписи

$$(M, \Sigma_1 = \varnothing, H, \Sigma_2 = (R, S)) \quad \text{и} \quad \left(M, \Sigma_1' = \varnothing, H', \Sigma_2' = (R', S')\right),$$

где $H = H(M)$ — ответ случайного оракула из первого вызова A, а $H' = H(M)$ — из второго. Кроме того,

$$R = G^k \pmod P \pmod Q, \qquad R' = G^{k'} \pmod P \pmod Q,$$

$$S = \frac{H + xR}{k} \pmod Q, \qquad S' = \frac{H' + xR'}{k'} \pmod Q.$$

Цель алгоритма B_A — найти x. Однако здесь мы не можем считать, что $k = k'$, поскольку мы не знаем, верно ли равенство: $R = R'$. Значит, прием, сработавший при проверке стойкости схемы подписи Шнорра, в этой ситуации не применим. Фактически, доказательство стойкости схемы подписи *DSA* пока отсутствует.

Можно попытаться подправить доказательство, несколько видоизменив схему *DSA*, применяя хэш-функцию одновременно к M и R, а не только к M. Такое изменение схемы в терминах леммы Стерна и Пойнтчеваля влечет, что $\Sigma_1 = R$, $H = H(M||R)$ и $\Sigma_2 = \frac{H+xR}{k} \pmod Q$.

Даже при такой модификации, приближающей схему *DSA* к схеме Шнорра, мы не сможем доказать ее стойкости. По лемме мы получим две подписи, в которых

$$R = G^k \pmod P \pmod Q, \qquad R' = G^{k'} \pmod P \pmod Q,$$

$$S = \frac{H + xR}{k} \pmod Q, \qquad S' = \frac{H' + xR'}{k'} \pmod Q,$$

причем $R = R'$. Но мы все еще не сможем сделать вывод о равенстве $k = k'$, поскольку оно не вытекает из уравнения

$$G^k = G^{k'} \pmod P \pmod Q.$$

Препятствие — в приведении по модулю Q. Удалив последнюю операцию, мы получим возможность доказать стойкость подписи. Од-

нако при этом теряется привлекательное свойство малого размера подписи *DSA*.

17.2.1.3. EC-DSA-подписи. Аналогичные проблемы возникают при попытках применить лемму Стерна и Пойнтчеваля к схеме подписи *EC-DSA*. Однако здесь небольшое изменение схемы позволяет доказать ее стойкость в определенных ситуациях. Вновь предполагаем, что мы применяем хэш-функцию к M и R, а не только к M. Дважды вызывая подпрограмму противника и используя лемму Стерна и Пойнтчеваля, получаем

$$R = x\text{-коорд.}([k]P) \pmod{Q}, \qquad R' = x\text{-коорд.}([k']P) \pmod{Q},$$
$$S = \frac{H + xR}{k} \pmod{Q}, \qquad S' = \frac{H' + xR'}{k'} \pmod{Q},$$

причем снова $R = R'$ и $H = H(M||R)$, $H' = H(M||R')$ — два критических хэш-запроса. Если Q больше, чем размер конечного поля, из равенства $R = R'$ можно заключить, что $k = \pm k'$, и вывести отсюда равенство

$$(S \mp S')k = H - H' \pmod{Q}.$$

Таким образом мы получим два возможных значения для k, которые дадут нам два возможных значения x. Для правильного ответа остается лишь проверить истинность равенства $[x]P = Y$ для обоих кандидатов. Итак, мы доказали следующую теорему.

Теорема 17.2. Из предположения трудноразрешимости задачи дискретного логарифмирования на эллиптической кривой $E(\mathbb{F}_P)$ вытекает (в модели случайного оракула), что успешная пассивная атака, направленная на **модифицированную** схему подписи *EC-DSA* невозможна, если

$$Q = \#E(\mathbb{F}_P) > P.$$

Обратите внимание на то, что этот результат применим лишь к определенному подмножеству всех эллиптических кривых.

17.2.2. Активный противник

Для доказательства стойкости схем в отношении активного противника нам необходимо показать, как алгоритм B_A будет отвечать на требования подписи со стороны алгоритма A. С этой целью мы опять подключаем модель случайного оракула, в которой будем использовать способность алгоритма B_A выбирать значения хэш-функции. Заметим, что аргумент хэш-функции может быть неизве-

стен вплоть до получения подписи на сообщении. Если хэш-функция при этом применяется только к сообщению M и не зависит от других величин (как Σ_1 в предыдущих рассуждениях), то алгоритм A может послать запрос хэш-оракулу, входными данными которого служит сообщение M, перед тем, как подпись будет создана. В этой ситуации алгоритм B_A не способен изменить ответ по сравнению с предыдущим.

Процесс ответов алгоритма B_A на запросы подписи нападающего A без его участия и без информации о секретном ключе называется моделированием запросов подписи. Такое моделирование по существу означает, что активный противник обладает не бóльшими возможностями, чем пассивное нападение в модели случайного оракула, поскольку любой активный противник может быть превращен в пассивного простым моделированием запросов подписи.

17.2.2.1. Схема подписи Шнорра. Моделирование запросов подписи в доказательстве стойкости схемы Шнорра осуществляется довольно легко. Оно тесно связано с протоколами доказательств с нулевым разглашением из параграфа 13.3. Мы предполагаем, что симулятор хранит список L всех предыдущих запросов случайного оракула. По введенному в него сообщению M симулятор делает следующее:

1. Вычисляет случайные числа S и H, удовлетворяющие условию: $1 \leqslant S, H < Q$.
2. Полагает $R = Q^S Y^{-H}$.
3. Если $(R||M, H') \in L$ при $H' \neq H$, то симулятор возвращается к шагу 1.
4. Присоединяет к множеству L элемент $(R||M, H)$, т. е. на запрос $(R||M)$ хэш-оракул теперь будет всегда выдавать ответ H.
5. Выдает «подпись» (H, S).

Проверьте, что перечисленные шаги приводят к верной подписи и что алгоритм A не сможет отличить моделируемые значения H, генерируемые случайным оракулом, от тех, которые производит истинный алгоритм подписи. Итак, мы получили теорему.

Теорема 17.3. Если задача дискретного логарифмирования в группе A трудноразрешима, то (в модели случайного оракула) не существует успешной активной атаки на схему подписи Шнорра с группой A.

В отношении *DSA* аналогичных результатов пока не получено. Доказательство стойкости известно для схемы *EC-DSA*, где вместо моделирования хэш-функции и сведения стойкости к проблеме

дискретного логарифмирования в качестве главного объекта моделируется групповая операция, а стойкость сводится к проблеме повторяющихся значений реально используемой хэш-функции.

17.2.3. *RSA-FDH*

Следует отметить, что в нашем обсуждении схем Шнорра, *DSA* и *EC-DSA* предполагалось, что значения хэш-функции H принадлежат полю $\mathbb{F}_P$. Такие хэш-функции трудно построить на практике, хотя предыдущие рассуждения использовали именно это свойство.

Аналогичная ситуация возникает в одном из вариантов схемы подписи *RSA*, называемом *RSA-FDH* (от англ. *full domain hash*). Хэш-функция в ней — отображение

$$H : \{0,1\} \longrightarrow (\mathbb{Z}/N\mathbb{Z})^*,$$

где N — *RSA*-модуль. Такую хэш-функцию тоже трудно реализовать, но предположив, что она существует, и включив ее в модель случайного оракула, мы сможем доказать стойкость следующего алгоритма подписи *RSA*.

Пусть N — модуль алгоритма *RSA*, E — шифрующая, а d — расшифровывающая экспоненты. Обозначим через f функцию, действующую по правилу:

$$f : \begin{cases} (\mathbb{Z}/N\mathbb{Z})^* \longrightarrow (\mathbb{Z}/N\mathbb{Z})^*, \\ x \mapsto x^E. \end{cases}$$

В этих терминах задача *RSA* заключается в определении x по данному $Y = f(x)$. В схеме *RSA-FDH* подпись на сообщении имеет вид

$$S = H(M)^d = f^{-1}(H(M)).$$

Можно доказать следующую теорему.

Теорема 17.4. Если в модели случайного оракула существует активный противник A, способный добиться экзистенциальной фальсификации в схеме *RSA-FDH*, используя q_H хэш-запросов и q_S обращений к подписывающему оракулу, то найдется алгоритм, который по данному Y может восстановить соответствующий прообраз функции *RSA* с вероятностью $\frac{1}{q_H}$.

Доказательство. Опишем алгоритм B_A, который по данному $Y \in (\mathbb{Z}/N\mathbb{Z})^*$ вычисляет $x = f^{-1}(Y)$. Алгоритм B_A сначала выбирает значение $T \in [1, \ldots, q_H]$ и заводит нумерованный список всех сделанных хэш-запросов. Затем он вызывает подпрограмму A и отвечает на сделанный противником хэш-запрос для введенного

сообщения M_i следующим образом:

- если A делает хэш-запрос, совпадающий с уже сделанным ранее запросом под номером T, то B_A дает Y в качестве ответа и модифицирует список запросов так, чтобы выполнялось равенство $Y = H(M_T)$;
- если хэш-запрос противника A выглядит как M_I с $I \neq T$, то B_A вычисляет случайный элемент $S_i \in (\mathbb{Z}/N\mathbb{Z})^*$ и добавляет к списку хэш-запросов $H(M_I) = S_I^E \pmod N = H_I$, храня запись значения S_I, а в качестве ответа выдает H_I.

Если противник A требует подписать сообщение M_I перед хэш-запросом с этим сообщением, то перед ответом B_A моделирует соответствующий хэш-запрос вместо алгоритма A. После этого ответ на запрос о подписи выглядит так:

- если сообщение M_I совпадает с M_T, то алгоритм останавливается и сообщает о сбое;
- если $M_I \neq M_T$, то в качестве ответа на запрос подписи B_A выдает S_I.

Допустим, что противник закончит работу и выдаст в качестве результата подписанное сообщение (S, M). Без ограничения общности можно предполагать, что при этом алгоритм A делал хэш-запрос для M. Если $M \neq M_T$, то алгоритм B_A останавливается с сообщением о неудачной попытке обратить функцию. Если же $M = M_T$, то имеет место соотношение

$$f(S) = H(M_T) = Y.$$

Следовательно, мы вычислили требуемый прообраз функции f.

Анализируя алгоритм B_A, заметим, что при корректной работе противник A создал экзистенциальную подделку (M, S), и поэтому он не мог в процессе работы требовать от оракула подписать сообщение M. Выбранное в начале работы значение T не зависит от вида алгоритма A. Поэтому алгоритм A не может постоянно требовать поставить подпись на сообщение M_T. Значит, грубо говоря, вероятность успешного завершения алгоритма B_A приблизительно равна $\frac{1}{q_H}$, т. е. вероятности того, что экзистенциальная подделка была сделана именно на сообщении M_T, а не на каком-то другом. ■

17.2.4. *RSA-PSS*

Другой способ применения *RSA* в качестве подписывающего алгоритма основан на использовании так называемой *RSA-PSS* или

вероятностной схемы подписи *RSA* (от англ. *probabilistic signature scheme*). Ее стойкость тоже доказывается с помощью модели случайного оракула. Мы не станем давать здесь подробное доказательство, ограничившись простым описанием схемы, поскольку ее значение постоянно возрастает. Преимущество этой схемы перед *RSA-FDH* состоит в том, что в ней используется традиционная хэш-функция, принимающая значения в битовых строках длины t, а не в кольце вычетов по модулю составного числа.

Как обычно, фиксируем модуль N алгоритма *RSA*, открытую экспоненту E и секретную экспоненту d. Обозначим параметр безопасности через k, т. е. модуль N состоит из k двоичных знаков. Определим два целых числа k_0 и k_1 так, чтобы

$$k_0 + k_1 \leqslant k - 1.$$

Например, можно взять $k_i = 128$ или 160.

Кроме того, зададим две хэш-функции, одна из которых разворачивает данные, а другая их сжимает:

$$G : \{0,1\}^{k_1} \longrightarrow \{0,1\}^{k-k_1-1},$$
$$H : \{0,1\}^{*} \longrightarrow \{0,1\}^{k_1}.$$

Пусть

$$G_1 : \{0,1\}^{k_1} \longrightarrow \{0,1\}^{k_0}$$

обозначает функцию, сопоставляющую строке $w \in \{0,1\}^{k_1}$ первые k_0 знаков числа $G(w)$, а функция

$$G_2 : \{0,1\}^{k_1} \longrightarrow \{0,1\}^{k-k_0-k_1-1}$$

сопоставляет строке $w \in \{0,1\}^{k_1}$ оставшиеся $k - k_0 - k_1 - 1$ знаков числа $G(w)$.

Чтобы поставить подпись на сообщении M, делают следующее:

- генерируют случайную строку $R \in \{0,1\}^{k_0}$,
- вычисляют $W = H(M||R)$,
- определяют $Y = 0||W||(G_1(W) \oplus R)||G_2(W)$,
- выдают подпись в виде $S = Y^d \pmod N$.

Для проверки подписи (S, M)

- вычисляют $Y = S^E \pmod N$;
- разбивают Y на компоненты:

$$B||W||\alpha||\gamma,$$

где B — однобитовое слово, W — k_1-битовое, α — k_0-битовое, а γ состоит из $k - k_0 - k_1 - 1$ знаков;

– вычисляют $R = \alpha \oplus G_1(W)$;
– подпись имеет силу, если

$$B = 0, \qquad G_2(W) = \gamma \quad \text{и} \quad H(M||R) = W.$$

Если доверить моделирование хэш-функций G и H случайному оракулу, то можно показать, что изложенная схема подписи стойка в том смысле, что существование успешного алгоритма, осуществляющего экзистенциальную фальсификацию, влечет обращение функции *RSA*. Полное доказательство этого факта изложено в статье Белларе и Рогавей, точную ссылку на которую можно найти в конце этой главы.

17.3. Стойкость шифрующих алгоритмов

Мы видели, что в предположениях проблемы выбора Диффи – Хеллмана довольно легко создать семантически стойкую схему шифрования с открытым ключом, имея в виду лишь пассивного противника. Например, криптосистема Эль-Гамаль обладает этим свойством. Кроме того, мы убедились, что можно легко построить семантически стойкие шифры, основанные на проблеме квадратичных вычетов, опять-таки принимая во внимание только пассивных противников. К сожалению, система Гольдвассера и Микали, о которой идет речь, обладает весьма неприятным свойством, сильно увеличивающим объем сообщений. Оказывается, создание криптосистемы, основанной на проблеме дискретного логарифмирования, защищенной от активного противника или семантически стойкой в предположениях *RSA*, сталкивается со значительными трудностями.

В этом параграфе мы сначала остановимся на ранних попытках конструирования криптосистемы, основанной на примитиве Эль-Гамаль, которая была бы защищена от активного противника. На их примере удобно продемонстрировать основные принципы проектирования. Затем мы перейдем к описанию главной системы, построенной на *RSA*-примитиве, которая в модели случайного оракула защищена от активного противника, а именно, системы *RSA-OAEP*.

17.3.1. Иммунизация криптосистем, основанных на Эль-Гамаль

Напомним, что шифрование Эль-Гамаль имеет вид

$$(G^k, m \cdot Y^k),$$

где $Y = G^x$ — открытый ключ. Используя довольно элементарную технику, мы уже показали в главе 15, что в предположениях проблемы выбора Диффи-Хеллмана шифрование Эль-Гамаль обладает семантической стойкостью. Однако там же было продемонстрировано, что такая система не стойка в отношении активного противника, поскольку соответствующий шифротекст не обладает жесткостью.

Было понято, что проблема, связанная с активным противником, состоит в том, что нападающий может слишком легко создать корректный шифротекст. Дело в том, что если бы противнику было трудно моделировать шифротекст, поддающийся расшифрованию, то атака с выбором шифротекста не дала бы ему никаких особых преимуществ. Действительно, у него нет причин требовать расшифрования шифротекста, который он может получить только зашифровав самостоятельно какой-то открытый текст. Это означает, что необходима такая расшифровывающая процедура, которая по введенному в нее шифротексту выдает или подлежащий открытый текст, или сообщение о несоответствии этого шифротекста какому-либо открытому сообщению. Для этой цели нужно добавлять к шифротексту некоторую дополнительную информацию, по которой расшифровывающая процедура могла бы определять, получен ли этот шифротекст с помощью законной процедуры шифрования из какого-то связного сообщения. Стоит сравнить сделанное замечание с нашей дискуссией в главе 4, где мы аргументированно показывали, что шифротекст не должен содержать избыточной информации. Заметьте, что там мы защищались лишь от пассивного нападения, в то время как здесь мы пытаемся противостоять противнику со значительно бóльшими возможностями.

Женг и Себерри первыми применили эту философию для создания практической криптосистемы, которая определяет современный подход к разработке шифрующих функций с открытым ключом. Их работа имеет большое значение. Поэтому мы расскажем об этих ранних попытках разработать стойкие криптосистемы с открытым ключом в качестве иллюстрации такого подхода.

Первое, на что стоит обратить внимание: шифрование с открытым ключом обычно используется для того, чтобы скрыть ключ, с помощью которого будет шифроваться большое сообщение. Следовательно, нет необходимости в шифровании сообщения, которое принадлежит группе A (как в Эль-Гамаль). Однако можно воспользоваться идеей Эль-Гамаль для передачи ключа, с помощью которого будет выработан сеансовый ключ, шифрующий фактическое сообщение.

Введем некоторые обозначения.

- A — нескрываемая группа простого порядка Q, порожденная элементом G.
- $V(H)$ — функция, которая по элементу H группы генерирует случайную строку битов. Ее часто называют функцией, производящей ключи.
- H — хэш-функция со значением в l-битовых строках.
- $Y = G^x$ — открытый ключ, соответствующий секретному ключу x.

17.3.1.1. Схема Женга–Себерри 1. Чтобы зашифровать сообщение m, вычисляют

1. $k \in \{1, \ldots, Q-1\}$.
2. $z = V(Y^k)$.
3. $t = H(m)$.
4. $C_1 = G^k$.
5. $C_2 = z \oplus (m||t)$.
6. Шифротекст — (C_1, C_2).

При расшифровании предпринимают такие шаги.

1. $z' = V(C_1^x)$.
2. $w = z' \oplus C_2$.
3. t' — последние l битов строки w.
4. m' — первые $\#w - l$ знаков строки w.
5. Если $H(m') = t'$, то m' — расшифрованное сообщение.
6. В противном случае алгоритм выдает сообщение о некорректности шифротекста.

Особенность криптосистемы состоит в добавлении к шифротексту Эль-Гамаль специальной информации, а именно, зашифрованного хэш-значения от открытого текста. Поскольку (предположительно) хэш-функцию обратить трудно, практически невозможно написать корректный шифротекст не зная соответствующий открытый. Дописывание дополнительного хэш-значения вносит требуемую шифротексту избыточность, которая тестируется расшифровывающей функцией.

17.3.1.2. Схема Женга–Себерри 2. Вторая система использует универсальную одностороннюю хэш-функцию, которая является, по существу, параметрическим набором хэш-функций H_i с $i \leqslant \ell$. Ее

можно представлять себе как функцию, снабженную переключателями, или как MAC.

Говоря более формально, универсальная односторонняя хэш-функция — это функция H_k, снабженная переключателями, обладающая следующим свойством: если противнику дан X и выбран наугад секретный ключ k, то противнику должно быть крайне сложно подобрать такой элемент y, что

$$H_k(y) = H_k(X).$$

Чтобы зашифровать сообщение m во второй схеме Женга–Себерри, мы вычисляем

1. $k \in \{1, \ldots, Q-1\}$.
2. z — $\#m$ левых знаков числа $V(Y^k)$.
3. s — ℓ правых знаков числа $V(Y^k)$.
4. $C_1 = G^k$.
5. $C_2 = H_s(m)$.
6. $C_3 = z \oplus m$.
7. Шифротекст — (C_1, C_2, C_3).

Оставим читателю выписать шаги расшифровывающей процедуры в качестве полезного упражнения. Эта система аналогична первой, но здесь хэш-значение сообщения не шифруется, а пересылается в открытом виде. Эта система сложнее предыдущей, т. к. мы не знаем, какой конкретно хэш-функцией или ключом получено соответствующее хэш-значение. Вторая схема Женга–Себерри очень близка системе *DHIES*, которая на данный момент считается наилучшей практической реализацией функции шифрования Эль-Гамаль.

17.3.1.3. Схема Женга–Себерри 3. В третьей и последней схеме Женга и Себерри эксплуатируется *DSA*-подобная подпись, или «символ» шифруемого сообщения. Схема работает, комбинируя шифрование Эль-Гамаль с *DSA*-подписью, однако открытый ключ схемы подписи *DSA* носит эфемерный характер и составляет часть шифротекста. Здесь мы также ограничимся шагами процедуры шифрования, оставив читателю продумывание противоположного процесса.

1. $k, t \in \{1, \ldots, Q-1\}$.
2. $r = Y^{k+t}$.
3. $z = G(r)$.
4. $C_1 = G^k$.

5. $C_2 = G^t$.
6. $C_3 = (H(m) + xr)/k \pmod{Q}$.
7. $C_4 = z + m$.
8. Шифротекст — (C_1, C_2, C_3, C_4).

Женг и Себерри доказали стойкость своих схем при очень сильных предположениях, а именно, они считали, что пространство шифротекстов взаимно однозначно соответствует пространству сообщений, что является близким предположению о текстозависимости алгоритма шифрования. Однако можно показать, что первая из схем не стойка. Предположим, что на стадии *поиска* противник выдает два сообщения M_1 и M_2. Затем выбирают скрываемый бит b и предлагают противнику сообщение M_b в зашифрованном виде, т. е. шифротекст

$$C = (C_1, C_2) = \left(G^k, z \oplus (M_b || H(M_b))\right).$$

Противник на стадии *гипотезы* может предпринять следующие операции. Сначала он генерирует новое сообщение M_3, отличное от M_1 и M_2, но совпадающее с ними по длине. Затем нападающий требует от оракула расшифровать шифротекст

$$\left(C_1, C_2 \oplus (M_1 || H(M_1)) \oplus (M_3 || H(M_3))\right).$$

Если $b = 1$, то этот шифротекст — корректная шифрованная версия сообщения M_3. Поэтому оракул, расшифровав его, должен получить M_3. А если $b = 0$, то шифротекст с очень малой долей вероятности будет шифрованной версией хоть какого-то сообщения, не говоря уже о M_3. Таким образом мы получаем алгоритм, который за полиномиальное время определяет значение скрываемого бита b.

17.3.2. *RSA-OAEP*

Напомним, что необработанная *RSA*-функция не дает семантически стойкой шифрующей системы даже в отношении пассивного противника. Для создания стойкой системы нам нужно или добавлять избыточную информацию к открытому тексту перед шифрованием, или вставлять лишние данные в сам шифротекст. Кроме того, в целях получения недетерминированного процесса, добавляемые куски должны выбираться случайным образом. В *RSA* это достигается использованием специальной пополняющей схемы, и за последние годы было предложено много таких схем. Однако некоторые из самых ранних предложений на сегодняшний день считаются слишком слабыми.

Одна из наиболее удачных на данный момент пополняющих схем была изобретена Белларе и Рогавей. Она называется *оптимизированным асимметричным пополнением шифрования* или *OAEP* (от англ. Optimized Asymmetric Encryption Padding). *OAEP*— пополняющая схема, которую можно использовать с любой односторонней функцией с секретом, в частности, с *RSA*-функцией. Когда ее используют совместно с *RSA*, то обозначают *RSA-OAEP*.

В первое время после создания *RSA-OAEP* считалось, что эта система является текстозависимой, но вскоре стало ясно, что это не так. Однако с помощью модели случайного оракула можно показать, что *RSA-OAEP* семантически стойка в отношении адаптивной атаки с выбором шифротекста.

Расскажем сначала в общем виде о процедуре шифрования *OAEP*. Пусть f — односторонняя функция с секретом, которая отображает k-битовые строки в k-битовые строки. Если $k = 1024$, то такой функцией можно считать *RSA*-функцию $C = m^E$. Пусть k_0 и k_1 такие числа, что 2^{k_0} и 2^{k_1} нереализуемо (например, $k_0, k_1 > 128$). Положим $n = k - k_0 - k_1$ и обозначим через

$$G : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{n+k_1}, \quad H : \{0,1\}^{n+k_1} \longrightarrow \{0,1\}^{k_0}$$

хэш-функции. Пусть m — сообщение, состоящее из n битов. Зашифруем его, используя функцию

$$\varepsilon(m) = f\Big(\{m||0^{k_1} \oplus G(r)\}||\{r \oplus H(m0^{k_1} \oplus G(r))\}\Big),$$

где

- $m||0^{k_1}$ означает m, за которым стоит k_1 нулей,
- r — случайная строка битов длины k_0,
- знак $||$ означает конкатенацию строк.

Можно представлять себе *OAEP* как двухэтапную сеть Фейстеля (см. рис. 17.1).

Чтобы расшифровать сообщение $\varepsilon(m)$, мы вычисляем

$$A = \{T||\{r \oplus H(T)\}\} = \Big\{\{m||0^{k_1} \oplus G(r)\}||\{r \oplus H(m||0^{k_1} \oplus G(r))\}\Big\}.$$

Таким образом, мы знаем

$$T = m||0^{k_1} \oplus G(r).$$

Следовательно, можно вычислить $H(T)$ и найти r, зная $r \oplus H(T)$. По r получаем $G(r)$ и восстанавливаем сообщение m. Заметим, что нам необходимо проверить, действительно ли строка $T \oplus G(r)$ оканчивается k_1 нулями. Если это не так, то мы должны сделать вывод, что соответствующий шифротекст некорректен.

Сформулируем и докажем основной результат, посвященный системе *RSA-OAEP*.

Теорема 17.5. Если предположения *RSA* справедливы, то в модели случайного оракула система *RSA-OAEP* семантически стойка в отношении адаптивной атаки с выбором шифротекста при условии моделирования хэш-функций G и H случайным оракулом.

Доказательство. Мы приведем схематическое доказательство, оставив проработку деталей любознательному читателю. Прежде всего перепишем *RSA*-функцию в виде

$$f : \begin{cases} \{0,1\}^{n+k_1} \times \{0,1\}^{k_0} & \longrightarrow & (\mathbb{Z}/N\mathbb{Z})^*, \\ (s,t) & \mapsto & (s||t)^E \pmod N. \end{cases}$$

Заметим, что такую функцию невозможно выразить математической формулой[1], но предположим, что мы можем это сделать и определим *RSA-OAEP* как

$$s = (m||0^{k_1}) \oplus G(r), \quad t = r \oplus H(s).$$

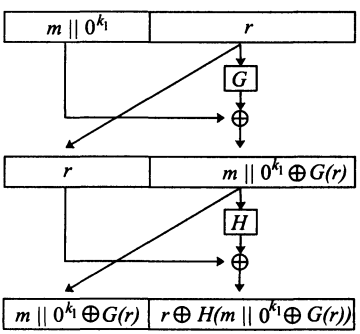


Рис. 17.1. *OAEP* как сеть Фейстеля

Можно доказать, что предположения *RSA* эквивалентны частичной односторонности функции f в том смысле, что задача восстановления s по значению $f(s,t)$ так же сложна, как и вычисление всего прообраза (s,t) по $f(s,t)$. Поэтому мы попытаемся встроить алгоритм A, взламывающий функцию *RSA-OAEP*, в алгоритм B_A, восстанавливающий частичный прообраз функции *RSA*. В частности, фиксировав модуль N, на вход алгоритма B_A подают элемент $C^* = f(s^*, t^*)$ и требуют определить s^*.

Алгоритм B_A вызывает стадию *поиска* противника A для генерирования двух сообщений M_0 и M_1. После этого B_A выбирает бит

[1] Не очень ясно, что тут автор имеет в виду, поскольку элементы множества $\{0,1\}^m$ можно воспринимать как двоичную запись натуральных чисел, лежащих от 0 до $2^{m+1}-1$. Запись $(s||t)$ с $s \in \{0,1\}^{n+k_1}$ и $t \in \{0,1\}^{k_0}$ означает число $s \cdot 2^{k_0} + t$. Поэтому формула для функции f — $f(s,t) = (s \cdot 2^{k_0} + t)^E \pmod N$. — *Прим. перев.*

b и предполагает, что C^* — шифрованная версия сообщения M_b. Затем шифротекст C^* передается на стадию *гипотезы* нападающего алгоритма A, и тот пытается определить значение b. Во время работы A алгоритм B_A обязан отвечать на хэш-запросы в отношении хэш-функций G и H и подменять собой расшифровывающий оракул. Для последовательности ответов B_A ведет два списка — H-список и G-список, — занося туда все хэш-запросы относительно значений функций H и G соответственно. На запросы противника алгоритм B_A отвечает по определенной схеме.

Запрос $G(\Gamma)$. Для каждого Δ из H-списка проверяется истинность равенства

$$C^* = f(\Delta, \Gamma \oplus H(\Delta)).$$

- Если оно верно, то мы частично обратили функцию f, что нам и было нужно. Мы все еще можем моделировать функцию G и положим
$$G(\Gamma) = \Delta \oplus (M_b || 0^{k_1}).$$
- Если равенство ложно для всех Δ из H-списка, то мы произвольно выбираем $G(\Gamma)$ из области значений, соблюдая равномерную плотность распределения вероятности выбора.

Запрос $H(\Delta)$. Выбирается произвольное случайное значение $H(\Delta)$ из множества значений функции H, и для всех Γ из G-списка проверяется равенство

$$C^* = f(\Delta, \Gamma \oplus H(\Delta)).$$

Если для какого-нибудь Γ получится верное равенство, то мы достигнем своей цели.

Запрос о расшифровании C. Ищем в наших G- и H-списках такую пару Γ, Δ, что числа

$$\Sigma = \Delta, \quad \Theta = \Gamma \oplus H(\Delta) \quad \text{и} \quad \Lambda = G(\Gamma) \oplus \Delta$$

удовлетворяют следующим условиям: $C = f(\Sigma, \Theta)$ и k_1 наименьших значащих бит числа Λ равны нулю. Найдя такую пару, мы даем в качестве соответствующего открытого текста n старших значащих цифр числа Λ. Если наш поиск окончится неудачей, то мы ответим, что шифротекст некорректен.

Заметим, что передав шифротекст, полученный корректным путем (т. е. с помощью вызова функций G и H и с применением законного шифрующего алгоритма), описанному выше расшифровывающему оракулу, мы получим исходный открытый текст.

Необходимо доказать, что спроектированный нами дешифрующий оракул достаточно долгое время способен «обманывать» противника A. Иначе говоря, нам нужно показать, что если оракулу передается шифротекст, который не был произведен с помощью предыдущих необходимых хэш-запросов, то получается значение, согласующееся с работой алгоритма A.

Наконец, нам нужно продемонстрировать, что если противник A имеет достаточно большие шансы взломать семантическую стойкость системы *RSA-OAEP*, то существует достаточно большая вероятность частичного обращения функции f.

Эти последние моменты доказываются с помощью тонкого анализа вероятностей, ассоциированных с некоторым числом частных случаев. Напомним, что B_A предполагает, что $C^* = f(s^*, t^*)$ — шифрованная версия M_b. Следовательно, должно найтись такое r^*, которое удовлетворяет условиям:

$$r^* = H(s^*) \oplus t^*, \qquad G(r^*) = s^* \oplus (M_b || 0^{k_1}).$$

Сначала доказывают незначительность вероятности неудачной работы расшифровывающего оракула. Затем, предполагая достаточно высокую вероятность успеха нападающего алгоритма A, показывают, что вероятность получения s^* в качестве запроса тоже не маленькая. Как только поступит хэш-запрос о числе s^*, мы можем остановить процесс, поскольку цель — частичное инвертирование функции f — будет достигнута.

К доказательству привлекается сложная техника теории вероятностей, и мы не будем ее здесь излагать. Любознательному читателю можно порекомендовать статью Фуджисаки, Окамото, Пойнтчеваля и Стерна, где написано полное доказательство. ■

17.3.3. Преобразование схем *CPA* в схемы *CCA2*

Предположим, что у нас есть схема шифрования с открытым ключом, семантически стойкая в отношении атаки с выбором открытого текста, например, Эль-Гамаль. По определению, такая схема должна быть недетерминированной, поэтому мы записываем шифрующую функцию как

$$E(m, r),$$

где m — предназначенное для шифрования сообщение, а r — случайная дополнительная информация. Расшифровывающую функцию будем, как обычно, обозначать через $D(C)$. Следовательно, в шифро-

вании Эль-Гамаль имеем

$$E(m, r) = (G^r, m \cdot H^r).$$

Фуджисаки и Окамото показали, как превратить такую схему в криптосистему, семантически стойкую в отношении адаптивного нападения в модели случайного оракула. Фактически, они доказывают текстозависимость трансформированной схемы. Мы опустим доказательство вообще, но приведем конструкцию трансформации, которая настолько же проста, насколько элегантна.

Меняем шифрующую функцию следующим образом:

$$E'(m, r) = E(m||r, F(m||r)),$$

где F — какая-то хэш-функция. Расшифровывающий алгоритм тоже несколько видоизменяется. Сначала вычисляют

$$m' = D(C),$$

а потом проверяют равенство

$$C = E(m', F(m')).$$

Если равенство истинно, то m восстанавливается из $m' = m||r$. В противном случае делается вывод о некорректности шифротекста.

В частности, для Эль-Гамаль получается шифрующий алгоритм

$$(G^{F(m||r)}, (m||r) \cdot H^{F(m||r)}),$$

который лишь немного менее эффективен, чем исходная криптосистема.

Краткое содержание главы

- Основной инструмент доказуемой стойкости состоит в использовании успешного нападения на криптосистему для решения предположительно трудной вычислительной задачи. Поскольку мы верим в сложность этой задачи, можно сделать вывод о невозможности успешной атаки.
- Модель случайного оракула — вычислительная модель, применяемая в доказательствах стойкости. Доказательство стойкости системы, полученное в модели случайного оракула, не означает, что данная криптосистема действительно стойка. Оно лишь говорит о том, что система может быть стойкой.
- В модели случайного оракула применяется лемма Стерна и Пойнтчеваля, чтобы показать, что определенная схема, осно-

ванная на дискретном логарифмировании, стойка. Чтобы получить доказательство в случае активного противника, используют случайный оракул для моделирования требований подписи противника.

- Можно показать, что две основные схемы подписи *RSA*, а именно *RSA-FDH* и *RSA-PSS*, применяемые на практике, стойки в модели случайного оракула.
- Доказательства стойкости алгоритмов шифрования несколько более хитрые. Ранние попытки таких доказательств, предпринятые Женгом и Себерри, были основаны на нестандартных предположениях.
- В модели случайного оракула можно показать, что стандартный метод шифрования *RSA*, а именно, *RSA-OAEP*, криптостоек.

Дополнительная литература

Доказуемая стойкость — быстро развивающаяся область криптографии, и число статей, посвященных этой теме, возрастает с каждым годом. Хорошее описание леммы Стерна и Пойнтчеваля и ее приложений даны в оригинальной работе Пойнтчеваля и Стерна. Модель случайного оракула и некоторые приложения, включая *RSA-FDH* и *RSA-PSS*, описаны в работе Белларе и Рогавея. Полное доказательство стойкости системы *RSA-OAEP* приведено в труде Фуджисаки и др.

M. Bellare and P. Rogaway. *Random oracles are practical: a paradigm for designing efficient protocols*. In Proc. 1st Annual Conf. on Comp. and Comms. Security, ACM, 62–73, 1993.

M. Bellare and P. Rogaway. *The exact security of digital signatures — How to sign with RSA and Rabin*. In Advances in Cryptology — EuroCrypt '96. Springer-Verlag LNCS 1070, 399–416, 1996.

E. Fujisaki, T. Okamoto, D. Pointcheval and J. Stern. *RSA-OAEP is secure under the RSA assumption*. In Advances in Cryptology — CRYPTO 2001, Springer-Verlag LNCS 2139, 260–274, 2001.

D. Pointcheval and J. Stern. *Security arguments for digital signatures and blind signatures*. J. Cryptology, **13**, 361–396, 2000.

Контрольные вопросы

17.1.1. Что скрывается под термином «достаточно большая вероятность»?

17.1.2. Что собой представляет модель случайного оракула и почему доказательства, полученные в рамках этой модели, не применимы на практике?

17.1.3. Сформулируйте лемму Стерна и Пойнтчеваля. Расскажите, как она применяется для доказательства стойкости схемы подписи Шнорра в модели случайного оракула.

17.1.4. Почему невозможно применить эту лемму для доказательства стойкости *DSA* в модели случайного оракула?

17.1.5. Почему в целях обеспечения стойкости шифрующих схем в отношении активного нападения проектировщики добавляют к шифротексту избыточную случайную информацию?

17.1.6. Какая пополняющая схема используется в *RSA-OAEP*?

17.1.7. Какое отношение схема пополнения из *RSA-OAEP* имеет к шифру Фейстеля?

Упражнения

17.2.1. Насколько легко построить хэш-функцию с множеством значений $(\mathbb{Z}/N\mathbb{Z})^*$, применяемую в *RSA-FDH*?

17.2.2. Выпишите расшифровывающие процедуры второй и третьей схем Женга и Себерри.

17.2.3. Доказательство стойкости *RSA-OAEP* исходит из предположения о том, что функция *RSA*, отображающая k-битовые строки в k-битовые строки, является односторонней функцией с секретом. Верное ли это предположение? И если оно ложно, то можно ли сконструировать успешную атаку на *RSA-OAEP*?

ГЛАВА 18

ДОКАЗУЕМАЯ СТОЙКОСТЬ БЕЗ СЛУЧАЙНОГО ОРАКУЛА

Цели главы

- Описать некоторые из наиболее современных схем, чью стойкость можно доказать, не прибегая к модели случайного оракула.
- Исследовать сильные предположения *RSA* и интерактивные хэш-предположения Диффи–Хеллмана.
- Объяснить алгоритмы подписи *GHR* и Крамера–Шоупа.
- Рассказать о шифрующем алгоритме Крамера–Шоупа.
- Познакомить с алгоритмом шифрования *DHIES*.

18.1. Введение

В предыдущей главе мы рассматривали алгоритмы шифрования и схемы цифровой подписи, чью стойкость можно доказать с помощью модели случайного оракула. Как уже отмечалось, модель случайного оракула реально не моделирует вычислений, встречающихся на практике. Доказательства, полученные в рамках этой модели, говорят лишь о *возможной* стойкости алгоритмов, применяющихся на практике, но не гарантируют ее. Эти доказательства можно интерпретировать так, что если существует удачное нападение на схему, то противник должен воспользоваться специально привлеченной хэш-функцией.

В этой главе приводится краткий обзор последних работ о попытках конструирования алгоритмов подписи и шифрования, не зависящих от модели случайного оракула. Мы ограничимся только теми схемами, которые имеют практическое значение, опустив

при этом подробные доказательства утверждений, но расскажем об основных моментах рассуждений. Читатель, интересующийся подробными доказательствами или схемами, не вошедшими в книгу, должен обратиться к разнообразной литературе в этой области.

Мы увидим, что в то время как стойкость естественных шифрующих алгоритмов может быть доказана без случайного оракула, в отношении схем подписи такого заявления сделать нельзя. Ситуация здесь противоположна той, которая возникала при привлечении случайного оракула к моделированию хэш-функций. В предыдущей главе схемы подписи выглядели более естественными по сравнению с алгоритмами шифрования. Но это вполне объяснимо: алгоритмы подписи интенсивно используют хэш-функции для обеспечения своей стойкости. Следовательно, имеет смысл считать, что они предъявляют такие строгие требования к хэш-функциям, что в реальном мире и не встретишь функций, удовлетворяющих этим требованиям.

Однако отказ от случайного оракула стоит дорого. Теперь нам нужно делать более строгие предположения, чем те, которые мы могли себе позволить со случайным оракулом. В следующем параграфе речь пойдет о двух таких новых предположениях. Они несомненно имеют близкое отношение к уже встречавшимся нам, например, трудноразрешимость проблемы выбора Диффи–Хеллмана или сложность задачи *RSA*. Но новая формулировка старых задач менее изучена, нежели уже знакомая Вам. Кроме того, модифицированные задачи окажутся менее сложными, поэтому предположение о их трудноразрешимости — более сильное, чем исходные.

18.2. Некоторые новые задачи

18.2.1. Сильные *RSA*-предположения

Мы уже изучали предположения *RSA*, которые говорят о трудности следующей задачи:

Определение 18.1. (Задача *RSA*.) Пусть дан модуль $N = p \cdot q$ алгоритма *RSA*, экспонента E, взаимно простая с $\varphi(N)$, и случайный элемент $C \in (\mathbb{Z}/N\mathbb{Z})^*$. Требуется найти элемент $m \in (\mathbb{Z}/N\mathbb{Z})^*$, для которого

$$m^E = C \pmod{N}.$$

Сильное предположение RSA состоит в трудноразрешимости такой задачи:

Определение 18.2. (Слабая задача *RSA*.) Пусть дан модуль $N = p \cdot q$ алгоритма *RSA* и случайный элемент $C \in (\mathbb{Z}/N\mathbb{Z})^*$. Требуется найти число $e > 1$ и элемент $m \in (\mathbb{Z}/N\mathbb{Z})^*$, для которых

$$m^e = C \pmod{N}.$$

Ясно, что решив задачу *RSA*, мы сможем найти решение и слабой задачи *RSA*. Это означает, что сильные предположения *RSA* действительно сильнее исходных предположений в том смысле, что решив слабую задачу *RSA*, мы все еще не сможем найти ответ в задаче *RSA*. Тем не менее, на данный момент мы будем гипотетически считать, что обе версии задачи *RSA* эквивалентны по сложности.

18.2.2. Интерактивные хэш-предположения Диффи – Хеллмана

Интерактивные хэш-предположения Диффи – Хеллмана состоят в трудноразрешимости определенного аналога проблемы выбора Диффи – Хеллмана. Прежде всего уточним, что мы имеем в виду под хэш-задачей Диффи–Хеллмана или *HDH* (от англ. Hash Diffie–Hellman problem). Предположим, что нам дана конечная циклическая группа A и хэш-функция L. В хэш-задаче Диффи–Хеллмана требуется по данным элементам группы G^x, G^y и хэш-значению H определить, истинно ли равенство:

$$H = L(G^x || G^{xy}).$$

Заметим, что эти предположения комбинируют информацию о взаимодействии хэш-функции со стандартными предположениями о трудности решения проблемы выбора Диффи–Хеллмана.

Предположим теперь, что нападающему на задачу *HDH* разрешен доступ к оракулу, который по скрытому значению y выдает хэш-значение

$$HDH_y(X) = L(X || X^y).$$

Интерактивные *HDH*-предположения были выведены из предположений проблемы выбора Диффи–Хеллмана с учетом возможностей адаптивного активного противника, вытекающих из доступа последнего к описанному выше оракулу. Ясно, что мы должны воспрепятствовать противнику вводить в HDH_y-оракул значение G^x, поскольку это прямой запрос на решение хэш-задачи Диффи–Хеллмана. Поэтому мы ограничим противника, не разрешая тому осуществить процедуру

$$HDH_y(G^x).$$

Но любые другие запросы к HDH-оракулу противнику дозволены. Интерактивные хэш-предположения Диффи–Хеллмана состоят в том, что противник, вооруженный таким оракулом, не сможет найти ответ в хэш-задаче Диффи–Хеллмана.

Заметим, что в определении адаптивной атаки требуется хэш-функция. Чтобы понять почему, допустим, что мы используем аналогичное определение адаптивной атаки, направленной на предположения проблемы выбора Диффи–Хеллмана. По данным элементам $B = G^x$, $C = G^y$ и $D = G^z$ нападающий мог мы запрашивать «нехэшированный» оракул Диффи–Хеллмана

$$DH_y(G \cdot B) = DH_y(G \cdot G^x),$$

который бы отвечал на этот запрос

$$G^{y(x+1)} = C \cdot G^{xy}.$$

В этом случае мы могли бы разделить ответ оракула на C и проверить истинность равенства

$$G^{xy} = D.$$

Комбинирование хэш-функции с предположениями Диффи – Хеллмана в одну группу предположений позволит нам (в схеме *DHIES*, приведенной ниже) отказаться от участия случайного оракула в доказательстве стойкости. Тем не менее, еще остается некоторая странность, поскольку определенная хэш-функция, используемая в схеме, взаимодействует с конкретной абелевой группой схемы. Это взаимодействие может привести, вообще говоря, к более легкой хэш-задаче Диффи–Хеллмана. Поэтому можно с уверенностью заявить, что использование хэш-предположений Диффи–Хеллмана в противовес случайному оракулу привносит некоторую философскую проблему.

18.3. Схемы подписи

Мы уже отмечали, что очень трудно создать схемы подписи, чью стойкость можно доказать, не используя модели случайного оракула. Найденные схемы с таким свойством кажутся несколько искусственными в сравнении с *RSA-PSS*, *DSA*, схемой Шнорра и другими применяемыми на практике схемами подписи. Первая из таких схем с доказуемой стойкостью в стандартной модели была предложена Гольдвассером, Микали и Ривестом. Правда, она была не очень практичной, т. к. в этой схеме сообщения ассоциировались с

листьями двоичного дерева, каждый узел в котором аутентифицирован своим отцом[1]. Это делает схему слишком медлительной.

В этом параграфе мы рассмотрим две «практичные» современные схемы подписи с доказуемой стойкостью. Однако мы увидим, что они приводят к некоторым проблемам, которых не возникает в стандартных схемах подписи.

18.3.1. Схема подписи Дженнаро–Галеви–Рабина

В 1999 г. Дженнаро, Галеви и Рабин изобрели схему подписи с доказуемой стойкостью, названную *GHR*-схемой подписи. Доказательство ее стойкости, которая основывается на сильных предположениях *RSA*, можно получить, не привлекая модель случайного оракула.

На стадии генерирования ключа выбирают модуль *RSA*

$$N = p \cdot q$$

так, чтобы простые числа p и q удовлетворяли дополнительному условию:

$$\text{числа } \frac{p-1}{2},\ \frac{q-1}{2} \quad \text{— простые.} \tag{18.1}$$

Такое требование на множители модуля означает, что найти нечетное целое число, взаимно простое с

$$\varphi(N) = (p-1)(q-1),$$

настолько же сложно, насколько трудно разложить число на простые множители. Кроме модуля N к открытому ключу добавляется случайный элемент $S \in (\mathbb{Z}/N\mathbb{Z})^*$.

Чтобы поставить подпись на сообщение M, законный пользователь схемы, знающий простые делители числа N, может найти такое число Σ, что

$$\Sigma^{H(M)} = S \pmod{N},$$

где H — фиксированная хэш-функция. Это же уравнение работает на стадии проверки подписи.

---

[1] Двоичным деревом называется связный граф, из каждой вершины, которого выходит не более двух ребер. Как правило, одна из вершин дерева выделяется и называется корнем. После этого на вершинах дерева вводится частичный порядок, согласно которому сравнимы вершины, расположенные на одной ветви дерева. Бóльшей считается та, которая расположена ближе к корню. Минимальные вершины называются листьями. Вершины, отличные от корня и листьев, называются узлами. Из двух соседних узлов на ветви отцом считается бóльшая вершина, а меньшая — сыном. — *Прим. перев.*

Легко понять, как эта схема соотносится с сильными предположениями *RSA*. Однако данная схема нуждается в очень специальном типе хэш-функции. Чтобы увидеть, почему, предположим, что активный противник намерен подписать сообщение M_1, причем ему удалось найти такое сообщение M_2, что

$$H(M_2) = Z \cdot H(M_1).$$

Допустим теперь, что этот нападающий требует от оракула поставить подпись на M_2. Оракул выдает значение Σ_2, для которого

$$\Sigma_2^{H(M_2)} = S \pmod N.$$

В этой ситуации атакующий может подделать подпись на сообщении M_1, вычисляя

$$\Sigma_1 = \Sigma_2^Z \pmod N,$$

поскольку

$$\Sigma_1^{H(M_1)} = \left(\Sigma_2^Z\right)^{H(M_1)} \pmod N = \Sigma_2^{ZH(M_1)} \pmod N =$$
$$= \Sigma_2^{H(M_2)} \pmod N = S.$$

Авторы схемы подписи предлагают самый простой способ пресечения такого типа атак, а именно, в качестве хэш-функции H стоит брать те, которые принимают значение в простых числах. Можно было бы еще потребовать от хэш-функции, чтобы она была защищена от повторяющихся значений. Хэш-функции, значения которых — простые числа, спроектировать можно, но они довольно плохо изучены и не выглядят «естественными».

18.3.2. Схема подписи Крамера–Шоупа

Схема подписи Крамера–Шоупа также основана на сильных предположениях *RSA* и обладает доказуемой стойкостью вне модели случайного оракула. Как и в предыдущей схеме, нам необходимо генерировать простые числа, но здесь они не должны являться значениями хэш-функции. Поэтому схема Крамера–Шоупа может довольствоваться стандартной хэш-функцией, например, *SHA-1*. Далее символом F мы будем обозначать «стандартную» хэш-функцию, значения которой — 160-битовые строки, интерпретируемые, как обычно, 160-битовыми натуральными числами.

При выборе открытого ключа мы вновь выбираем *RSA*-модуль N как произведение простых чисел p и q, удовлетворяющих свойству

(18.1). Фиксируются два квадратичных вычета

$$H, X \in Q_N,$$

где, как обычно, Q_N обозначает множество всех квадратичных вычетов по модулю N. Кроме того, генерируется случайное 160-значное двоичное простое число E'. Открытый ключ состоит из четверки

$$(N, H, X, E'),$$

а соответствующий секретный ключ — это простые числа p и q.

Чтобы подписать сообщение, необходимо генерировать еще одно 160-значное двоичное простое число E и дополнительный случайный квадратичный вычет $Y' \in Q_N$. После этого законный пользователь, зная делители модуля N, может найти решение Y уравнения

$$Y = \left(XH^{F(X')}\right)^{\frac{1}{E}} \pmod N,$$

где X' определяется формулой

$$X' = Y'^{E'} H^{-F(M)}.$$

Подпись выглядит следующим образом:

$$(E, Y, Y').$$

На этапе проверки прежде всего необходимо убедиться, что E' — нечетное число, не совпадающее с E. Затем нужно вычислить

$$X' = Y'^{E'} H^{-F(M)}$$

и проверить, что

$$X = Y^E H^{-F(X')}.$$

Исходя из предположения о том, что хэш-функция F защищена от повторений, а сильные *RSA*-предположения верны, можно показать, что эта схема подписи стойка в отношении активного противника. Приведем наиболее важные моменты доказательства, но опустим все детали, оставив их любознательному читателю для самостоятельного восполнения.

Предположим, что противник делает t запросов подписывающему оракулу. Встроим алгоритм нападения A в новый алгоритм B_A, который взламывает сильные *RSA*-предположения при фиксированном модуле N. Перед тем как ввести в алгоритм A открытый ключ, алгоритм B_A должен решить, какое именно простое число E_i будет выдаваться в качестве ответа на запросы подписи. Затем, зная E_i, алгоритм B_A подбирает такие значения для H и X из открытого ключа, чтобы ему были известны корни степени E_i из H и X.

Итак, получив запрос о подписи на сообщении M_i, (в качестве подписывающего оракула) B_A может вычислить имеющую силу подпись, не обращаясь к разложению числа N на множители, а генерируя $Y_i' \in Q_N$ случайным образом и вычисляя

$$X_i' = {Y_i'}^{E'} H^{-F(M_i)} \pmod N, \quad Y_i = X^{\frac{1}{E_i}} (H^{\frac{1}{E_i}})^{F(X_i')} \pmod N.$$

Подпись будет иметь вид

$$(M_i, Y_i, Y_i').$$

В полном доказательстве этот основной алгоритм, моделирующий подписывающего оракула, должен быть несколько изменен в зависимости от типа фальсификации, производимой алгоритмом A. Но основная идея доказательства состоит в том, что алгоритм B_A создает открытый ключ, позволяющий ему правдоподобно ответить на любой запрос подписи противника A.

18.4. Алгоритмы шифрования

В отличие от схем подписи, можно создать практичные близкие к применяемым в реальной жизни шифрующие системы, обладающие свойством доказуемой стойкости вне модели случайного оракула. Фактически, вторая из схем, рассматриваемых нами в этом параграфе, а именно, *DHIES*, шифрующий алгоритм с открытым ключом, основанный на проблеме дискретного логарифмирования в конечных полях или на эллиптической кривой, приводится в различных стандартных документах.

Недостаток системы *DHIES* заключается в том, что ее стойкость частично опирается на предположение о трудности решения хэш-задачи Диффи–Хеллмана. К сожалению, эта задача не так хорошо изучена, как обычная проблема выбора Диффи–Хеллмана. Поэтому мы начнем с представления доказуемо стойкой схемы, чья стойкость зиждется на обычной проблеме выбора Диффи–Хеллмана.

18.4.1. Схема шифрования Крамера–Шоупа

Параметры домена схемы шифрования Крамера–Шоупа состоят из конечной абелевой группы A простого порядка Q. Кроме того, фиксируется универсальное одностороннее семейство хэш-функций. Напомним, что это такое семейство $\{F_i\}$ хэш-функций, при котором противнику трудно выбрать число X и функцию F_i, для которых можно было бы подобрать другое число y, удовлетворяющее соот-

ношению

$$F_i(X) = F_i(y).$$

Для открытого ключа схемы выбираются случайные элементы

$$G_1, G_2 \in A, \quad x_1, x_2, y_1, y_2, z \in \mathbb{Z}/Q\mathbb{Z}.$$

По ним законный пользователь может вычислить

$$C = G_1^{x_1} G_2^{x_2}, \quad D = G_1^{y_1} G_2^{y_2}, \quad H = G_1^z.$$

Затем выбирается хэш-функция F из универсального одностороннего семейства $\{F_i\}$ и публикуется открытый ключ в виде набора

$$(G_1, G_2, C, D, H, F).$$

Соответствующий секретный ключ имеет вид:

$$(x_1, x_2, y_1, y_2, z).$$

Процедура шифрования в этой схеме напоминает аналогичный процесс в криптосистеме Эль-Гамаль. Сообщение m отождествляется с элементом абелевой группы A. Выбирается случайный эфемерный ключ $r \in \mathbb{Z}/Q\mathbb{Z}$ и производятся вычисления:

$$\begin{aligned} U_1 &= G_1^r, & \alpha &= F(U_1||U_2||E), \\ U_2 &= G_2^r, & V &= C^r D^{r\alpha}. \\ E &= m \cdot H^r, \end{aligned}$$

Шифротекстом служит четверка

$$(U_1, U_2, E, V).$$

Получив шифротекст, обладатель секретного ключа восстанавливает соответствующий открытый текст по следующей схеме. Сначала вычисляется $\alpha = F(U_1||U_2||E)$ и проверяется равенство

$$U_1^{x_1+y_1\alpha} U_2^{x_2+y_2\alpha} = V.$$

Если это равенство ложно, то шифротекст признается некорректным. Если оно истинно, то получатель расшифровывает шифротекст вычислением:

$$m = \frac{E}{U_1^z}.$$

Для проверки доказуемой стойкости схемы в предположениях о сложности проблемы выбора Диффи–Хеллмана и принадлежности функции F к универсальному одностороннему семейству хэш-функций допустим, что у нас есть алгоритм A, атакующий нашу

схему, и покажем, как его встроить в другой алгоритм B_A, пытающийся решить проблему выбора Диффи–Хеллмана.

Одна из формулировок проблемы выбора Диффи–Хеллмана имеет следующий вид: по данному набору элементов (G_1, G_2, U_1, U_2) абелевой группы A определить, является ли он случайным или образует четверку Диффи – Хеллмана, т. е. найдется такой $r \in \mathbb{Z}/Q\mathbb{Z}$, что $U_1 = G_1^r$ и $U_2 = G_2^r$. Таким образом, алгоритм B_A будет получать на входе набор элементов (G_1, G_2, U_1, U_2) группы и пытаться определить, случаен он или является четверкой Диффи–Хеллмана.

Прежде всего, алгоритм B_A должен выбрать открытый ключ. Делает он это нестандартным образом. Сначала алгоритм выбирает случайные элементы

$$x_1, x_2, y_1, y_2, z_1, z_2 \in \mathbb{Z}/Q\mathbb{Z},$$

затем производит вычисления

$$C = G_1^{x_1} G_2^{x_2}, \quad D = G_1^{y_1} G_2^{y_2}, \quad H = G_1^{z_1} G_2^{z_2}.$$

В последнюю очередь алгоритм B_A выбирает хэш-функцию F из универсального одностороннего семейства и получает открытый ключ:

$$(G_1, G_2, C, D, H, F).$$

Обратите внимание на то, часть ключа, соответствующая H, в алгоритме выбирается не так, как в реальной схеме, но с условием, что A не сможет заметить это изменение.

Выбрав открытый ключ, алгоритм B_A вызывает стадию *поиска* противника, отвечая на запрос по расшифровке шифротекста (U_1', U_2', E', V') вычислением

$$m = \frac{E'}{U_1'^{z_1} U_2'^{z_2}},$$

проверив предварительно корректность шифротекста. Выходными данными стадии *поиска* являются два открытых текста M_0 и M_1.

После стадии *поиска* алгоритм B_A выбирает бит B случайным образом и находит тестовый шифротекст (U_1, U_2, E, V):

$$E = M_B \cdot (U_1^{z_1} U_2^{z_2}), \quad \alpha = F(U_1 || U_2 || E), \quad V = U_1^{x_1 + y_1 \alpha} U_2^{x_2 + y_2 \alpha}.$$

Заметим, что при подаче на вход алгоритма B_A четверки Диффи–Хеллмана соответствующий шифротекст будет корректно расшифрован. Если же на вход подавать случайную четверку элементов, то почти наверняка шифротекст будет признан некорректным.

Тестовый шифротекст передается на стадию *гипотеза* противника A. Если нападающий правильно определит бит B, то мы запо-

дозрим, что введенная четверка является четверкой Диффи – Хеллмана. В противном случае — это случайный набор элементов. Повторяя процесс неоднократно, придем к статистическому тесту, определяющему, является ли данная четверка элементов четверкой Диффи – Хеллмана, причем с увеличением количества повторений мы увеличиваем вероятность правильного ответа.

Важно отметить, что приведенные выше рассуждения — всего лишь набросок доказательства. Необходимо еще показать, что нападающий алгоритм A не сможет отличить нашей игры от реально протекающей атаки, иначе бы он знал: что-то здесь не так. Например, нам нужно доказать, что ответы B_A на запросы нападающего с точки зрения последнего неотличимы от ответов истинного расшифровывающего оракула. Полное доказательство приведено в работе, ссылку на которую можно найти в разделе «Дополнительная литература» в конце этой главы.

Схема шифрования Крамера–Шоупа похожа на Эль-Гамаль, но существенно менее эффективна. Поэтому, несмотря на свойство доказуемой стойкости, схема Крамера–Шоупа не находит практического применения из-за неудобств в работе. Стойкость схемы шифрования *DHIES*, рассматриваемой далее, тоже доказывается без привлечения случайного оракула, хотя предположения о сложности задачи в этой схеме изучены меньше, чем проблема выбора Диффи–Хеллмана. Однако схема *DHIES* обладает лишь незначительно меньшей эффективностью, чем Эль-Гамаль из главы 7.

18.4.2. Схема шифрования *DHIES*

Схема шифрования *DHIES* сильно напоминает иммунизационные методы, предложенные Женгом и Себерри. Однако можно доказать стойкость схемы *DHIES* в отношении адаптивной атаки с выбором открытого текста, предполагая, что следующие три компоненты стойки сами по себе:

- конечная циклическая абелева группа, в которой выполнены интерактивные хэш-предположения Диффи–Хеллмана;
- шифрующая функция с симметричным ключом, семантически стойкая в отношении адаптивной атаки с выбором открытого текста;
- код аутентификации сообщения или хэш-функция с переключателями, для которой противник не может найти MAC «непрошенного сообщения». Об этой последней концепции имеет смысл думать как о MAC-версии экзистенциальной фальсификации.

Схема *DHIES* Белларе и Рогавей включает в себя все три выписанные компоненты, что объясняет ее название: *интегрированная схема шифрования Диффи–Хеллмана* или *DHIES* (от англ. Diffie-Hellman Integrated Encryption Scheme). Первоначально схема носила название расширенной схемы шифрования Диффи–Хеллмана или *DHAES* (от англ. Diffie-Hellman Augmented Encryption Scheme), которое не прижилось ввиду возможной путаницы с *AES* — новым стандартом шифрования.

Ключ в *DHIES* генерируется точно так же, как в криптосистеме Эль-Гамаль. К параметрам домена относятся циклическая конечная абелева группа A простого порядка Q, образующая этой группы G, симметричная шифрующая функция $\{E_k, D_k\}$, функция MAC, которую мы будем обозначать символом MAC_k, и хэш-функция F. Группу и хэш-функцию нужно выбирать так, чтобы для них были выполнены интерактивные хэш-предположения Диффи–Хеллмана.

Для создания ключевой пары генерируется случайный элемент $x \in \mathbb{Z}/Q\mathbb{Z}$ и вычисляется открытый ключ

$$P = G^x.$$

Чтобы зашифровать сообщение m, генерируется случайный эфемерный ключ k (свой для каждого сообщения) и вычисляется

$$v = P^k \quad \text{и} \quad U = G^k.$$

Подставив это в хэш-функцию, мы получим два ключа: один для симметричного алгоритма шифрования, а другой — для функции MAC:

$$(k_1, k_2) = F(U||v).$$

После этого вычисляем

$$C = E_{k_1}(m), \quad T = MAC_{k_2}(C).$$

Шифротекст передается в виде тройки

$$U||C||T.$$

Обратите внимание на то, что эта система дает возможность эффективно шифровать сколь угодно длинные сообщения с помощью алгоритма шифрования с открытым ключом. Величина U подменяет собой механизм передачи ключей, шифрование осуществляется симметричной функцией шифрования, а предотвращение адаптивных атак на всю схему обеспечивается дополнительным значением T функции MAC.

Чтобы расшифровать шифротекст $U||C||T$, законный обладатель секретного ключа может вычислить

$$v = U^x$$

и определить эфемерные ключи k_1, k_2 с помощью хэш-функции:

$$(k_1, k_2) = F(U||v).$$

Шифротекст корректен, если выполнено равенство:

$$T = MAC_{k_2}(C).$$

В этом случае сообщение восстанавливается вычислением

$$m = D_{k_1}(C).$$

Объясним основную идею доказательства стойкости *DHIES* в отношении адаптивного противника при условии правильного выбора симметричной системы шифрования, функции *MAC* и истинности интерактивных хэш-предположений Диффи–Хеллмана. Пусть A — алгоритм успешной атаки на *DHIES*. Покажем, как его можно использовать для решения интерактивной хэш-задачи Диффи–Хеллмана или для взлома симметричной системы шифрования, или для фальсификации функции *MAC*.

Сконструируем алгоритм B_A, предназначенный для решения интерактивной хэш-задачи Диффи–Хеллмана. На его вход подается три элемента

$$G^x,\ G^y \quad \text{и} \quad H.$$

Цель алгоритма — определить, выполнено ли равенство

$$H = F(G^y||G^{xy}). \tag{18.2}$$

В алгоритм B_A встроен оракул $\mathcal{O}$, который по секретному числу x вычисляет

$$HDH_x(X) = F(X||X^x).$$

Алгоритм B_A фиксирует открытый ключ

$$P = G^x$$

и вызывает стадию *поиска* противника A, на которой генерируются два сообщения M_0 и M_1. Затем B_A выбирает случайный бит B и по элементу H производит ключи K_1 и K_2:

$$K_1||K_2 = H.$$

На следующем шаге B_A вычисляет

$$C = E_{K_1}(M_B), \qquad T = MAC_{K_2}(C)$$

и получает шифротекст

$$(G^y, C, T),$$

который передается на стадию *гипотезы* алгоритма A. В результате работы противник A должен определить скрываемый бит B.

Если A правильно найдет этот бит, B_A делает вывод о том, что равенство (18.2) истинно; в противном случае это равенство скорее всего ложно. Доказывая стойкость схемы, показывают, что с помощью успешной атаки A алгоритм B_A способен достичь одной из следующих целей:

- установить истинность или ложность равенства (18.2);
- взломать семантическую стойкость привлеченной к схеме симметричной функции шифрования;
- фальсифицировать функцию MAC.

Детали доказательства очень интересны, но мы оставим их любознательному читателю для самостоятельной проработки. Единственное, что мы обсудим, это ответы алгоритма B_A на запросы дешифрования со стороны противника, да и то в упрощенном виде. Напомним, что B_A имеет доступ к оракулу $\mathcal{O}$, который решает хэш-задачу Диффи–Хеллмана для скрытого ключа x. Получив в качестве запроса шифротекст

$$U_i||C_i||T_i,$$

алгоритм B_A вызывает оракул $\mathcal{O}$ для вычисления

$$(k_1, k_2) = \mathcal{O}(U_i) = F(U_i||U_i^x).$$

После этого алгоритм B_A способен проверить корректность шифротекста и расшифровать его так же, как это сделала бы подлинная расшифровывающая функция. Проблема возникнет в том случае, когда от противника поступит запрос на расшифрование шифротекста с $U_i = G^y$, поскольку он фактически является запросом на решение интересующей нас хэш-проблемы, что недопустимо по правилам игры. Методы преодоления этой проблемы освещены в оригинальной работе, содержащей полное доказательство.

Заметим, что *интерактивная* природа хэш-предположений Диффи–Хеллмана требует моделирования расшифровывающего оракула, что дает простор для критики доказательства, поскольку оно сводит стойкость схемы *DHIES* к полностью нестандартным пред-

положениям. Это единственные интерактивные предположения, описанные в литературе. Можно также говорить о том, что в этом доказательстве возникают те же проблемы, что и в модели случайного оракула, поскольку в хэш-предположениях Диффи–Хеллмана скрыто нетривиальное взаимодействие хэш-функции с проблемой выбора Диффи–Хеллмана. Несмотря на указанные недостатки в доказательстве стойкости, схема *DHIES* применяется в реальной жизни ввиду ее высокой эффективности.

Краткое содержание главы

- Схемы подписи, чью стойкость удается доказать без случайного оракула, менее естественны, чем те, стойкость которых получается в рамках случайного оракула. Происходит это, видимо, потому, что подписи существенно используют хэш-функции, легко моделирующиеся случайным оракулом.
- Сильные предположения *RSA* — естественное ослабление стандартных *RSA*-предположений.
- Интерактивные хэш-предположения Диффи – Хеллмана считаются более слабыми, чем обычные предположения Диффи – Хеллмана. Они учитывают взаимодействие хэш-функции, используемой в задаче Диффи – Хеллмана, с соответствующей абелевой группой. Интерактивная природа этих предположений делает их полностью нестандартными.
- Схема шифрования Крамера–Шоупа в предположении сложности проблемы выбора Диффи – Хеллмана доказуемо стойка вне модели случайного оракула. Работает она примерно в три раза медленнее, чем обычный алгоритм шифрования Эль-Гамаль.
- Схема шифрования *DHIES* тоже доказуемо стойка без модели случайного оракула. Однако ее стойкость основывается на интерактивных хэш-предположениях Диффи – Хеллмана. К преимуществам этой схемы относится высокая эффективность.

Дополнительная литература

Схемы, упомянутые в этой главе, можно найти в следующих статьях.

M. Abdalla, M. Bellare and P. Rogaway. *DHAES: An encryption scheme based on the Diffie-Hellman problem*. Submission to IEEE P1363a standard.

R. Cramer and V. Shoup. *A practical public key cryptosystem provably*

secure against adaptive chosen ciphertext attack. In Advances in Cryptology — CRYPTO '98, Springer-Verlag LNCS 1462, 13–25, 1998.

R. Cramer and V. Shoup. *Signature schemes based on the strong RSA assumption*. ACM Transactions on Information and Systems Security, **3**, 161–185, 2000.

R. Gennaro, S. Halevi and T. Rabin. *Secure hash-and-sign signatures without the random oracle*. In Advances in Cryptology — EuroCrypt '99, Springer-Verlag LNCS 1592, 123–139, 1999.

Упражнения

18.1.1. Попытайтесь изобрести эффективную хэш-функцию, которую можно было бы использовать в схеме Дженнаро–Галеви–Рабина. Существуют ли какие-нибудь вычислительные предположения, позволяющие доказать, что Ваша хэш-функция защищена от повторений?

18.1.2. При моделировании подписей в схеме Крамера–Шоупа нам нужно случайным образом вычислять значения из Q_N, не зная простых делителей модуля N. С другой стороны, сложность определения принадлежности элемента множеству Q_N совпадает с трудностью задачи факторизации. Покажите, как эффективно реализовать эту стадию моделирования.

18.1.3. Покажите, что корректный шифротекст в криптосистеме Крамера – Шоупа выдерживает тест на справедливость соотношения

$$U_1^{x_1+y_1\alpha} U_2^{x_2+y_2\alpha} = V.$$

18.1.4. Проверьте, что тестовый шифротекст, подающийся на стадию *гипотезы* в доказательстве стойкости схемы шифрования Крамера–Шоупа, будет корректен, если входные данные алгоритма B_A — четверка Диффи – Хеллмана.

Приложение А

Основная математическая терминология

Здесь собраны математические понятия и термины, необходимые для чтения основного текста книги. Материал в приложении подается в более формализованной манере, нежели в первой части.

А.1. Множества

Начнем с самых основных определений, вошедших сюда для полноты картины.

Определение А.1. Для двух множеств A и B определены операции *объединения* ($\cup$), *пересечения* ($\cap$), *разности* ($\setminus$) и *прямого произведения* ($\times$):

$$A \cup B = \{x |\ x \in A \text{ или } x \in B\},$$
$$A \cap B = \{x |\ x \in A \text{ и } x \in B\},$$
$$A \setminus B = \{x |\ x \in A \text{ и } x \notin B\},$$
$$A \times B = \{(x, y) |\ x \in A \text{ и } x \in B\}.$$

Запись $A \subset B$ означает, что A является *подмножеством* в B, т. е.

$$x \in A \ \Rightarrow\ x \in B.$$

Непосредственно из определения следуют основные соотношения из теории множеств, которые иллюстрируются диаграммами Венна. Например, верна следующая лемма.

Лемма А.2. Если $A \subset B$ и $B \subset C$, то $A \subset C$.

Доказательство. Пусть x — элемент множества A. Нам надо показать, что он принадлежит множеству C. Поскольку $A \subset B$, имеем $x \in B$ по определению подмножества. Кроме того, нам известно, что $B \subset C$. Значит, $x \in C$. ■

Обратите внимание на то, что это рассуждение является строгим доказательством, в то время как соответствующая диаграмма Венна лишь иллюстрирует свойство, но не доказывает его. Диаграмма Венна, подтверждающая утверждение, которое Вы хотите

доказать, свидетельствует лишь о том, что Вы оказались не способны начертить диаграмму, противоречащую утверждению. Поэтому всегда остается шанс, что кому-то другому повезет больше.

А.2. Отношения

Здесь мы определяем отношения и рассматриваем некоторые их свойства. Отношения, в особенности отношение эквивалентности, играют первую скрипку в алгебре. Поэтому очень важно рассмотреть их здесь, чтобы облегчить дальнейшее восприятие материала.

Определение А.3. Произвольное подмножество в прямом произведении $A \times A$ называется (бинарным) *отношением* на множестве A.

Объясним это понятие на примере. Рассмотрим отношение «меньше чем или равно» на множестве натуральных чисел. Ясно, что оно дает нам совокупность

$$LE = \{(x, y) |\ x, y \in \mathbb{N},\ x \text{ меньше или равно } y\}.$$

Любое отношение, с которым Вы встречались ранее, может быть записано подобным теоретико-множественным образом. Есть даже более удобный способ определения отношения LE, а именно[1]

$$LE = \{(x, y) |\ x, y \in \mathbb{N},\ x - y \notin \mathbb{N}\}.$$

Очевидно, такими громоздкими обозначениями неудобно пользоваться на практике. Поэтому пару $(x, y) \in R$, т. е. упорядоченную пару элементов, находящуюся в отношении R, обозначают как $x\,R\,y$. Если теперь заменить обозначение LE на стандартное «$\leqslant$», то мы получим хорошо известное и полезное отношение порядка $x \leqslant y$.

Далеко не все отношения на множестве представляют интерес. Однако все используемые отношения в математике обладают одним или сразу несколькими из перечисленных в следующем определении свойств.

Определение А.4. Отношение R на множестве A называется

- *рефлексивным*, если для любого $x \in A$ имеет место включение $(x, x) \in R$;
- *симметричным*, если включение $(x, y) \in R$ влечет $(y, x) \in R$;

[1] В этом определении, в отличие от российской традиции, число 0 относится к натуральным числам. — *Прим. перев.*

- *антисимметичным*, если отношения xRy и yRx выполнены одновременно только для равных элементов $x = y$;
- *транзитивным*, если включения xRy и yRz влекут xRz.

Вернемся к нашему примеру «$\leqslant$». Очевидно, оно рефлексивно, поскольку $x \leqslant x$ для всякого числа x. Оно не симметрично, т. к. из неравенства $x \leqslant y$ совершенно не следует неравенство $y \leqslant x$. Зато оно антисимметрично. Действительно, все мы привыкли к тому, что совместное выполнение неравенств $x \leqslant y$ и $y \leqslant x$ возможно только тогда, когда $x = y$. Следует отметить, что отношение «$\leqslant$» обладает и транзитивностью.

Отношения, похожие на «$\leqslant$», встречаются столь часто, что им дали специальное название:

Определение А.5. Рефлексивное, антисимметричное и транзитивное отношение называется отношением *частичного порядка*.

Определение А.6. Антисимметричное и транзитивное отношение R называется отношением *линейного*, или *полного, порядка*, если для любой пары элементов x, y либо $(x, y) \in R$, либо $(y, x) \in R$.

Другой важный класс отношений — это отношения эквивалентности.

Определение А.7. Рефлексивное, симметричное и транзитивное отношение называется отношением *эквивалентности*.

Очевидным примером отношения эквивалентности на множестве натуральных чисел служит отношение «равно». К одной из главных задач в любой науке относится задача о классификации объектов, т. е. задача о причислении каждого из изучаемых объектов к одному из набора непересекающихся множеств. Каждое из таких множеств можно называть классом эквивалентности. Если интересующие нас свойства объектов постоянны на каждом классе эквивалентности, то можно ограничиться рассмотрением лишь самих классов, не заботясь об их содержании. Такой «упрощенный» взгляд помогает лучше понять ситуацию. На математическом жаргоне процесс перехода от объектов к классам их эквивалентности называется *факторизацией* по отношению эквивалентности. В алгебре и геометрии процесс факторизации используют для построения новых объектов: фактор-группы, фактор-многообразия и т. д. Пример, который мы сейчас рассмотрим, вероятно, наиболее близок Вам, поскольку касается арифметики остатков.

Пусть m — фиксированное натуральное число. Рассмотрим отношение эквивалентности на множестве целых чисел, согласно которому числа x и y эквивалентны, если их разность делится на m. Вам следует убедиться, что данное отношение действительно является отношением эквивалентности. Классы эквивалентности представлены следующим списком:

$$\overline{0} = \{\ldots, -2m, -m, 0, m, 2m, \ldots\},$$
$$\overline{1} = \{\ldots, -2m+1, -m+1, 1, m+1, 2m+1, \ldots\},$$
$$\ldots$$
$$\overline{m-1} = \{\ldots, -m-1, -1, m-1, 2m-1, 3m-1, \ldots\}.$$

Обратите внимание на то, что мы имеем ровно m классов эквивалентности, по одному для каждого из возможных остатков от деления на m. Их обычно называют *классами вычетов* по модулю m, а всю совокупность классов $\{\overline{0}, \ldots, \overline{m-1}\}$ обозначают через $\mathbb{Z}/m\mathbb{Z}$. Это множество имеет структуру кольца и называется *кольцом вычетов по модулю* m. Если $m = p$ — простое число, то кольцо вычетов является полем и обозначается символом $\mathbb{F}_p$.

А.3. Функции

Сформулируем два определения функции. Первое из них — словесное, и поэтому легче воспринимается неопытным читателем. Второе — теоретико-множественное.

Определение А.8. *Функцией* называется правило, согласно которому каждому элементу одного множества, *области определения*, ставится в соответствие элемент другого множества, *множества значений*. Причем каждому элементу области определения сопоставляется один и только один элемент множества значений[1].

Стоит отметить, что понятие «функция» включает в себя не только правило, скажем $f(x) = x^2$, но и два множества: область определения и множество значений.

Перейдем к примерам.

1. Формула $f(x) = \sqrt{x}$ не определяет функцию из $\mathbb{R}$ в $\mathbb{R}$, поскольку квадратный корень из отрицательных чисел не определен.

[1] В российской традиции такое правило называют *отображением*, оставляя термин «функция» для отображений с числовым множеством значений. — *Прим. перев.*

Это не будет функцией и из множества положительных вещественных чисел $\mathbb{R}_{\geqslant 0}$ в множество всех вещественных чисел $\mathbb{R}$, т. к. у каждого положительного вещественного числа есть два квадратных корня. С другой стороны, это правило задает функцию из $\mathbb{R}_{\geqslant 0}$ в $\mathbb{R}_{\geqslant 0}$.

2. Правило $f(x) = \frac{1}{x}$ не является функцией из $\mathbb{R}$ в $\mathbb{R}$, но задает функцию из $\mathbb{R} \setminus \{0\}$ в $\mathbb{R}$.
3. Заметим, что в определении функции ничего не говорится о том, что каждый элемент множества значений должен иметь прообраз. Поэтому правило $f(x) = x^2$, переводящее вещественное число в вещественное число, является функцией.

Наше определение не строго, поскольку мы четко не оговорили, что означает слово «правило». Дадим менее понятное, но более формализованное определение на теоретико-множественном языке.

Определение А.9. Функцией $F : A \longrightarrow B$ из множества A в множество B называется такое подмножество в прямом произведении $F \subset A \times B$, что

(1) если $(x, y) \in F$ и $(x, z) \in F$, то $y = z$;
(2) для любого $x \in A$ найдется такой $y \in B$, что $(x, y) \in F$.

Множество A называется областью определения, B — множеством значений функции. Первое условие определения означает, что каждый элемент области определения переходит не более чем в один элемент множества значений. Второе — каждый элемент области определения переходит по крайней мере в один элемент множества значений. Для функции $F : A \longrightarrow B$ и элемента $x \in A$ через $F(x)$ мы обозначаем тот единственный элемент из множества значений B, для которого $(x, F(x)) \in F$.

Иногда определена композиция функций. Если даны две функции $f : A \longrightarrow B$ и $g : B \longrightarrow C$, то их *композицией* называется функция $g \circ f : A \longrightarrow C$, состоящая из пар $(x, g(f(x)))$.

Лемма А.10. Пусть даны три функции: $f : A \longrightarrow B$, $g : B \longrightarrow C$ и $h : C \longrightarrow D$. Тогда

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

Доказательство. Пусть пара (a, d) лежит в $h \circ (g \circ f)$. Тогда по определению композиции существует $b \in B$, при котором $(a, b) \in f$ и $(b, d) \in (h \circ g)$. Применяя определение композиции еще раз, получаем, что найдется такой элемент $c \in C$, что $(b, c) \in g$ и $(c, d) \in h$.

Следовательно, $(a, c) \in (g \circ f)$, откуда $(a, d) \in h \circ (g \circ f)$. Значит,

$$h \circ (g \circ f) \subset (h \circ g) \circ f.$$

Аналогично доказывается обратное включение. ■

Особенно важное значение имеет так называемая тождественная функция.

Определение А.11. *Тождественной* функцией на множестве A называется функция $\mathrm{id}_A = \{(x, x) |\ x \in A\}$.

Лемма А.12. Любая функция $f : A \longrightarrow B$ удовлетворяет соотношениям:

$$f \circ \mathrm{id}_A = \mathrm{id}_B \circ f.$$

Доказательство. Пусть x — произвольный элемент множества A. Тогда

$$(f \circ \mathrm{id}_A)(x) = f(\mathrm{id}_A(x)) = f(x) = \mathrm{id}_B(f(x)) = (\mathrm{id}_B \circ f)(x).$$ ■

На протяжении всей книги мы пользовались следующими двумя свойствами:

Определение А.13. Функция $f : A \longrightarrow B$ называется *инъективной*, если равенство $f(x) = f(y)$ возможно только при $x = y$.

Функция $f : A \longrightarrow B$ называется *сюръективной*, если для каждого элемента $b \in B$ найдется прообраз, т. е. такой $a \in A$, что $f(a) = b$.

Если функция обладает обоими свойствами, т. е. она инъективна и сюръективна, ее называют *биективной* (или 1-1-*соответствием*). Обратимся к примерам.

1. Функция $f : \mathbb{R} \longrightarrow \mathbb{R}$, определенная формулой $f(x) = x + 2$, биективна.
2. Функция $f : \mathbb{N} \longrightarrow \mathbb{N}$, определенная формулой $f(x) = x + 2$, инъективна, но не сюръективна, поскольку числа 0 и 1 не имеют прообразов.
3. Формула $f(x) = x^2$ определяет сюръективную, но не инъективную функцию $f : \mathbb{R} \longrightarrow \mathbb{R}_{\geqslant 0}$, поскольку любое положительное вещественное число имеет два вещественных корня степени 2.

Следующее утверждение мотивирует изучение биективных функций.

Лемма А.14. Функция $f : A \longrightarrow B$ биективна тогда и только тогда, когда существует *обратная* ей функция $g : B \longrightarrow A$, т. е. такая функция, что $f \circ g$ и $g \circ f$ — тождественные функции. Функция g обычно обозначается через $g = f^{-1}$.

Оставим проверку этого утверждения в качестве упражнения. Заметим, что согласно лемме, функция g тоже будет биективной.

А.4. Перестановки

Пусть A — конечное множество, состоящее из n элементов. Без ограничения общности можно предполагать, что $A = \{1, 2, \ldots, n\}$. Биективная функция $f : A \longrightarrow A$ называется *перестановкой* или *подстановкой*. Множество всех перестановок n-элементного множества обозначается через S_n.

Пусть $A = \{1, 2, 3\}$. На нем есть, например, такая перестановка: $f(1) = 2$, $f(2) = 3$ и $f(3) = 1$. Такой способ задания перестановок слишком громоздок. Математики (будучи крайне ленивыми людьми) предлагают следующий способ задания перестановок:

$$f = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}.$$

Важно отметить, что в этом обозначении (которое легко распространяется на любое n) в любой строчке каждое число между 1 и n появляется лишь один раз. Первая строчка всегда содержит числа от 1 до n, записанные в порядке возрастания. Любая такая матрица изображает собой некоторую перестановку и любую перестановку можно записать такой матрицей. Замечание о соответствии между матрицами и перестановками подводит нас к следующей лемме.

Лемма А.15. Число перестановок в S_n равно $n!$.

Доказательство. Ясно, что число перестановок совпадает с количеством разных вторых строчек матриц, их изображающих. Попытаемся заполнять эти строки, следя за количеством возможностей. На первое место второй строки мы можем поставить любое из n чисел. На второе — уже любое из $n - 1$, поскольку одно число занято. На третье — любое из $n - 2$, и т. д. Осталось заметить, что число всех вариантов заполнения второй строки совпадает с произведением возможностей для каждой позиции: $|S_n| = n \cdot (n-1) \cdots 1 = n!$. ■

Поскольку перестановки — это функции, мы можем брать их композицию, что на математическом жаргоне называется умноже-

нием перестановок. Напомним, что $g \circ f$ означает, что мы сначала применяем функцию f, а потом g. Следовательно, композиция перестановок

$$g \circ f = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

действует по правилу:

$$1 \mapsto 3 \mapsto 1,$$
$$2 \mapsto 2 \mapsto 3,$$
$$3 \mapsto 1 \mapsto 2.$$

Значит, произведение этих перестановок равно

$$h = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}.$$

Как мы уже отмечали, по своей природе математики очень ленивы. Поэтому и матрица в качестве обозначения перестановки их не устраивает. Действительно, в каждой перестановке мы должны выписывать неизменную первую строку. Кроме того, отдельные столбцы матрицы излишни, например, первый столбец перестановки h. Введем еще одно обозначение перестановок, лаконичное и ясное, что вполне устраивает любого математика. Но сначала нам необходимо понятие цикла.

Определение А.16. *Циклом* или n-циклом $(x_1, \ldots, x_n)$ называется перестановка f, действующая по правилу: $f(x_1) = x_2$, $f(x_2) = x_3$, $\ldots$, $f(x_{n-1}) = x_n$, $f(x_n) = x_1$ и $f(x) = x$, если $x \notin \{x_1, \ldots, x_n\}$.

Например, перестановка

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

является циклом, который можно записать по-разному: $(1,2,3) = (2,3,1) = (3,1,2)$. Заметим, что не любая перестановка является циклом. Приведем пример, когда перестановка равна произведению двух 2-циклов и 1-цикла. Причем 1-цикл математики, как правило, выбрасывают из произведения, правомерно считая его тождественной перестановкой, т. е. перестановкой, которая реально ничего не переставляет.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 5 & 4 \end{pmatrix} = (1,3)(2)(4,5) = (1,3)(4,5) = (4,5)(1,3).$$

Два цикла $(x_1, x_2, \ldots, x_n)$ и $(y_1, y_2, \ldots, y_m)$ называются *независи-*

мыми, если $\{x_1, x_2, \ldots, x_n\} \cap \{y_1, y_2, \ldots, y_m\} = \varnothing$. Легко показать, что независимые циклы σ и τ перестановочны в произведениях:

$$\sigma \circ \tau = \tau \circ \sigma.$$

С другой стороны, свойство перестановочности не выполнено для зависимых циклов:

$$(1,2,3,4) \circ (3,5) = (1,2,3,5,4) \neq (1,2,5,3,4) = (3,5) \circ (1,2,3,4).$$

Что делает циклы по настоящему интересными, так это следующее утверждение.

Лемма А.17. Любую перестановку можно представить в виде произведения независимых циклов.

Доказательство. Пусть $\sigma \in S_n$. Обозначим через σ_1 цикл

$$(1, \sigma(1), \sigma(\sigma(1)), \ldots, \sigma(\ldots\sigma(1)\ldots)),$$

в котором мы применяем σ до тех пор, пока не получим снова 1. Затем берем такое число $x \in \{1, \ldots, n\}$, что $\sigma_1(x) = x$, и определяем второй цикл

$$\sigma_2 = (x, \sigma(x), \sigma(\sigma(x)), \ldots, \sigma(\ldots\sigma(x)\ldots)).$$

После этого возьмем число, которое остается неподвижным как при действии σ_1, так и σ_2, и определяем цикл σ_3. Будем продолжать процесс до тех пор, пока не исчерпаем все натуральные числа от 1 до n. Полученные в результате циклы $\sigma_1, \ldots, \sigma_t$, очевидно, независимы, а их произведение равно исходной перестановке σ. ■

Очень приятно, что приведенное доказательство конструктивно, в том смысле, что дает алгоритм разложения произвольной перестановки в произведение независимых циклов. Разберем пример, взяв в качестве σ перестановку

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 3 & 7 & 6 & 8 & 4 & 1 & 5 & 9 \end{pmatrix}.$$

Тогда $\sigma(1) = 2$, $\sigma(2) = 3$, $\sigma(3) = 7$ и $\sigma(7) = 1$. Получили первый цикл:

$$\sigma_1 = (1,2,3,7).$$

Следующий элемент, который мы должны рассмотреть — это 4, поскольку это минимальный элемент множества $\{1, \ldots, 9\}$, который неподвижен при действии σ_1. $\sigma(4) = 6$, $\sigma(6) = 4$. Поэтому $\sigma_2 = (4,6)$. Продолжая процесс, получаем $\sigma_3 = (5,8)$ и $\sigma_4 = (9)$, Следовательно,

$$\sigma = (1,2,3,7)(4,6)(5,8)(9) = (1,2,3,7)(4,6)(5,8).$$

А.5. Операции

В математике встречается большое количество бинарных операций: сложение и умножение чисел, композиция функций, сложение и умножение матриц, умножение перестановок, и т. д., всего и не вспомнишь сразу, поскольку имя таким операциям — легион. Бинарные операции имеют много общего, но в чем-то и отличаются друг от друга. Например, для вещественных чисел x и y справедливо тождество: $x \cdot y = y \cdot x$, но матрицы 2×2 с вещественными коэффициентами этим свойством не обладают, т. е. для общих матриц A и B выполнено обратное: $A \cdot B \neq B \cdot A$. Для изучения общих и индивидуальных свойств бинарных операций вводятся определенные понятия, которые мы приведем после этого абзаца. Опираясь на эти понятия, мы докажем некоторые полезные свойства операций. Таким образом, абстрактные свойства помогут нам одним рассуждением доказать большое число конкретных свойств.

Определение А.18. *Бинарной операцией* называется любая функция $f : A \times A \longrightarrow A$.

Так, например, если $A = \mathbb{R}$, то такой функцией может служить сумма: $f(x, y) = x + y$. Однако обозначать операцию $f(x, y)$ очень неудобно. Поэтому символ функции заменяется подходящим символом операции, который ставится между аргументами:

$$\begin{array}{ccc} x \cdot y, & x + y, & xy, \\ x \circ y, & x \odot y, & x \diamond y, \\ x \wedge y, & x \vee y, & x \star y. \end{array}$$

Чаще всего мы пишем $x + y$ и xy для обозначения операций сложения и умножения. Однако надо иметь ввиду, что эти обозначения могут использоваться и для нестандартных операций сложения и умножения.

Перейдем к свойствам операций[1].

Определение А.19. Говорят, что операция «$\diamond$» *ассоциативна*, если для любых элементов x, y и z выполнено тождество:

$$(x \diamond y) \diamond z = x \diamond (y \diamond z).$$

Все упомянутые выше конкретные операции ассоциативны. Су-

[1]Значок, стоящий в качестве операции в определениях, обозначает любую бинарную операцию. При этом не утверждается, что конкретная операция обязана обладать соответствующим свойством. — *Прим. перев.*

ществуют и неассоциативные операции, но они не интересны с точки зрения криптографии. Отметим, что для ассоциативной операции корректно определено выражение

$$w \diamond x \diamond y \diamond z,$$

поскольку при неизменном положении операндов порядок выполнения действий значения не имеет.

Определение А.20. Операция «$\vee$» называется *коммутативной*, если для любой пары x и y

$$x \vee y = y \vee x.$$

Сложение и умножение чисел и сложение матриц — коммутативные операции, в то время как произведение матриц или перестановок — некоммутативно.

Определение А.21. Говорят, что операция «$\cdot$» обладает *единицей* (единичным или нейтральным элементом), если существует такой элемент e, что для любого x справедливы равенства:

$$e \cdot x = x \cdot e = x.$$

Прежде всего отметим, что любая из приведенных выше операций обладает единицей, в то время как для вычитания вещественных чисел нейтрального элемента не существует. Действительно, если e такой элемент, то из второго равенства $x - e = x$ следует, что $e = 0$. Но это противоречит первому требованию: $e - x = x$. Следующее утверждение показывает, что для любой операции существует не более одной единицы.

Лемма А.22. Если для операции «$\cdot$» существует единица, то она только одна.

Доказательство. Предположим, что существует два единичных элемента: e и e'. Тогда $e \cdot e' = e'$ (поскольку e — единичный элемент). С другой стороны, $e \cdot e' = e$, ибо e' — тоже единица. Значит,

$$e' = e \cdot e' = e,$$

откуда $e = e'$. ■

Как правило, используя знак «+» для обозначения операции, мы называем единичный элемент нулем и обозначаем 0, а говоря об умножении, т. е. операции «$\cdot$,», единичный элемент упоминается как единица и обозначается 1 или e.

Определение А.23. Пусть «+» — бинарная операция на множестве A, обладающая нейтральным элементом 0, а x — элемент из A. *Обратным*, или *противоположным*, элементом к x называется такой $y \in A$, что

$$x + y = y + x = 0.$$

При аддитивном обозначении операции, т. е. при использовании знака «+», противоположный элемент к x обозначается $-x$, а при мультипликативном — x^{-1}.

Каждое вещественное число обладает противоположным по сложению. Любое ненулевое вещественное число обладает мультипликативным обратным. Каждой из перестановок можно сопоставить обратную ей относительно умножения перестановок. Однако только квадратные матрицы с ненулевым определителем имеют обратную матрицу относительно умножения.

Следующий результат говорит о том, что у любого элемента существует не более одного обратного относительно ассоциативной операции.

Лемма А.24. Пусть на множестве A задана ассоциативная операция с единицей e. Если $x \in A$ обладает обратным элементом относительно этой операции, то только одним.

Доказательство. Предположим, что найдется еще один обратный к x элемент y'. Тогда

$$y = ye = y(xy') = (yx)y' = ey' = y'. \qquad \blacksquare$$

Обратите внимание на то, как в доказательстве использована ассоциативность операции.

С этого момента будем считать, что все используемые нами операции ассоциативны. Введем обозначения для многократного повторения операций. В случае аддитивной и мультипликативной операций приняты обозначения

$$\underbrace{x + x + \cdots + x}_{n} = n \cdot x = nx, \qquad \underbrace{x \cdot x \cdots x}_{n} = x^n, \qquad \text{где } n \in \mathbb{N}.$$

Следующее утверждение можно доказать методом математической индукции.

Лемма А.25. (Правила действия со степенями) Для любой ассоциативной операции выполнены свойства

$$g^m \circ g^n = g^{m+n}, \qquad (g^m)^n = g^{m+n}.$$

Если операция обладает единицей, то эти правила можно продолжить на любые целые показатели, определив $(-n) = n \cdot (-x)$ и $x^{-n} = (x^{-1})^n$.

Следующая лемма очевидна, но применяя ее, неопытные «математики» часто совершают ошибки, поскольку она противоречит интуиции. Чтобы лучше ее усвоить, представьте, что работаете с матрицами.

Лемма А.26. Пусть на множестве G задана ассоциативная операция с единицей, а x и y — элементы в G, обладающие обратными. Тогда

(1) $(x^{-1})^{-1} = x$,
(2) $(xy)^{-1} = y^{-1}x^{-1}$.

Доказательство. Для проверки первого равенства достаточно осознать, что x и x^{-1} *взаимно* обратны: $x^{-1} \cdot x = e = x \cdot x^{-1}$. Второе равенство проверяется непосредственным вычислением с учетом единственности обратного:

$$(x \cdot y) \cdot (y^{-1} \cdot x^{-1}) = x \cdot (y \cdot y^{-1}) \cdot x^{-1} = x \cdot e \cdot x^{-1} = x \cdot x^{-1} = e. \quad \blacksquare$$

Приведем небольшой словарик соответствий понятий в аддитивной и мультипликативной записи.

Сложение	Умножение
$x + y$	xy
0	1 или e
$-x$	x^{-1}
$n \cdot x$	x^n

А.6. Группы

Определение А.27. *Группой* называется множество G с бинарной операцией, обладающей следующими свойствами:

- ассоциативность;
- существование единицы;
- каждый элемент из G обладает обратным.

Обратите внимание на то, что в определении группы мы не говорили о замкнутости операции, т. е. о том, что результат операции над любыми элементами группы принадлежит группе. Это свойство

вытекает непосредственно из определения самой операции. В случае коммутативной групповой операции говорят о коммутативных, или абелевых, группах.

В следующем списке представлены группы. В качестве упражнения мы предлагаем читателю в каждом случае выявить единицу, обратный элемент к каждому элементу группы и выяснить, какие из этих групп абелевы.

- Целые числа $\mathbb{Z}$ по сложению: $\mathbb{Z}^+$.
- Рациональные числа $\mathbb{Q}$ по сложению: $\mathbb{Q}^+$.
- Вещественные числа $\mathbb{R}$ по сложению: $\mathbb{R}^+$.
- Комплексные числа $\mathbb{C}$ по сложению: $\mathbb{C}^+$.
- Рациональные числа $\mathbb{Q} \setminus \{0\}$ без нуля по умножению: $\mathbb{Q}^*$.
- Вещественные числа $\mathbb{R} \setminus \{0\}$ без нуля по умножению: $\mathbb{R}^*$.
- Комплексные числа $\mathbb{C} \setminus \{0\}$ без нуля по умножению: $\mathbb{C}^*$.
- Множество векторов с n целыми, рациональными, вещественными, комплексными координатами с операцией сложения векторов.
- Множество $n \times m$ матриц с целыми, рациональными, вещественными, комплексными элементами с операцией сложения матриц. Эта группа обозначается через $M_{n\times m}(A)$, где A — множество, откуда берутся матричные элементы.
- Полная линейная группа, т. е. квадратные матрицы фиксированного размера с ненулевым определителем и элементами из $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ или $\mathbb{C}$. Если коэффициенты матриц из множества A, то такая группа обозначается через $\mathrm{GL}_n(A)$.
- Специальная линейная группа, т. е. квадратные матрицы фиксированного размера с определителем ± 1 и элементами из $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ или $\mathbb{C}$. Если коэффициенты матриц из множества A, то такая группа обозначается через $\mathrm{SL}_n(A)$.
- Множество S_n всех перестановок n-элементного множества с операцией умножения перестановок. Такая группа называется симметрической группой.
- Множество непрерывных (дифференцируемых) функций из $\mathbb{R}$ в $\mathbb{R}$ с операцией поточечного сложения.
- и так далее.

Список групп можно продолжать бесконечно. Группа — одно из базисных понятий математики. Однако не всякий математический объект является группой. В следующем списке Вам следует установить причину, почему то или иное множество не является группой.

- Множество $\mathbb{N}$ натуральных чисел с операцией сложения или умножения.
- Множество $\mathbb{Z}$ целых чисел с операцией вычитания или умножения.

Приведем некоторые определения из теории групп.

Определение А.28. *Порядком группы* G называется число ее элементов; оно обозначается через $|G|$ или $\#G$.

Порядком элемента $g \in G$ называется наименьшее натуральное число n, для которого $g^n = e$. Если такого числа нет, то говорят, что элемент g имеет бесконечный порядок.

Циклической называется такая группа G, в которой найдется элемент g, такой, что любой другой элемент из G запишется в виде g^m для некоторого натурального m. Такой элемент g называют образующей группы, а саму группу обозначают через $\langle g \rangle = G$.

Заметим, что только единица группы имеет порядок 1. Кроме того, порядки элементов x и x^{-1} совпадают.

Лемма А.29. Если $G = \langle g \rangle$ и порядок элемента g конечен и равен n, то $\#G = n$.

Доказательство. Любой элемент группы G имеет вид g^m для некоторого $m \in \mathbb{Z}$, но поскольку порядок элемента g равен n, то существует лишь n разных степеней, т. к.

$$g^{n+1} = g^n g = eg = g.$$

Поэтому в группе G ровно n элементов. ■

Соотнесем этот результат с перестановками, о которых мы говорили раньше. Напомним, что множество перестановок образует группу S_n относительно композиции. Легко видеть, что если $\sigma \in S_n$ — k-цикл, то порядок этой перестановки равен k. Кроме того, несложно доказать, что порядок произведения независимых циклов равен наименьшему общему кратному порядков циклов.

Говорят, что подмножество S группы G порождает G, если любой элемент группы может быть записан как произведение элементов из S и обратных к ним. Например,

- группа S_3 порождена множеством $\{(1,2),(1,2,3)\}$;
- группа $\mathbb{Z}^+$ порождена единицей;
- группа $\mathbb{Q}^*$ порождена множеством простых чисел и поэтому имеет бесконечное число образующих.

Заметим, что порядок группы ничего не говорит о числе ее образующих.

Важный класс конечных групп, в которых легко разобраться, доставляется арифметикой остатков. Напомним, что группа $\mathbb{Z}/m\mathbb{Z}$ состоит из элементов $\{0, 1 \ldots, m-1\}$. Ее групповая операция — сложение по модулю m. Относительно умножения это множество не образует группу в общей ситуации, даже если мы выбросим из него 0. Однако оно содержит подмножество, которое наделено структурой группы. Положим

$$(\mathbb{Z}/m\mathbb{Z})^* = \{x \in \mathbb{Z}/m\mathbb{Z} |\ \text{НОД}\,(m, x) = 1\}.$$

Оказывается, это группа по умножению. Результатом умножения двух элементов a и b является неотрицательный остаток от деления ab на m. Порядок группы $(\mathbb{Z}/m\mathbb{Z})^*$ обозначается через $\varphi(m)$ и называется значением функции Эйлера на числе m. Функция Эйлера играет заметную роль в теории чисел. Например, ее значение на простом числе p равно $\varphi(p) = p - 1$. Вернемся к этой функции чуть позже, а сейчас займемся подгруппами.

Определение А.30. Подгруппой H группы G называется подмножество $H \subset G$, которое является группой относительно групповой операции в G. В этом случае мы будем писать[1] $H < G$.

Согласно определению, группа $\mathrm{GL}_n(\mathbb{R})$ не является подгруппой в $M_n(\mathbb{R})$, хотя с теоретико-множественной точки зрения $\mathrm{GL}_n(\mathbb{R}) \subset \subset M_n(\mathbb{R})$. Дело в том, что групповой операцией в $\mathrm{GL}_n(\mathbb{R})$ является матричное умножение, тогда как в $M_n(\mathbb{R})$ — сложение матриц. С другой стороны, легко выписать цепочки подгрупп:

$$\mathbb{Z}^+ < \mathbb{Q}^+ < \mathbb{R}^+ < \mathbb{C}^+,$$
$$\mathbb{Q}^* < \mathbb{R}^* < \mathbb{C}^*.$$

Если отождествить $x \in \mathbb{Z}$ с диагональной матрицей $\mathrm{diag}(x, \ldots, x)$, то мы получим, что $\mathbb{Z}^+$ — подгруппа в $M_n(\mathbb{Z})$, и т. д.

В качестве важного примера рассмотрим множество $2\mathbb{Z}$ всех четных чисел. Они образуют подгруппу в $\mathbb{Z}^+$. Если обозначить $\mathbb{Z}^+ = 1\mathbb{Z}$, то можно утверждать, что $n\mathbb{Z} < m\mathbb{Z}$ тогда и только тогда, когда m делит n, где $m\mathbb{Z}$ — множество всех целых чисел, кратных m. Таким образом возникают разные цепочки подгрупп в $\mathbb{Z}^+$:

$$18\mathbb{Z} < 6\mathbb{Z} < 2\mathbb{Z} < \mathbb{Z}^+,$$

[1] В российской традиции используют обозначение $H \subset G$. — *Прим. перев.*

$$18\mathbb{Z} < 9\mathbb{Z} < 3\mathbb{Z} < \mathbb{Z}^+,$$
$$18\mathbb{Z} < 6\mathbb{Z} < 3\mathbb{Z} < \mathbb{Z}^+.$$

Покажем, что в $\mathbb{Z}^+$ другого типа подгрупп не бывает.

Лемма А.31. *Любая подгруппа в $\mathbb{Z}^+$ имеет вид $n\mathbb{Z}$ для некоторого неотрицательного целого числа n.*

Доказательство. Пусть H — подгруппа в $\mathbb{Z}^+$. Поскольку H не пусто, в ней должен найтись элемент x и обратный к нему $-x$. Если $x = 0$ и других элементов H не содержит, то $H = 0\mathbb{Z}$. В противном случае в H найдется хотя бы одно натуральное число. Обозначим через n наименьшее натуральное число, лежащее в подгруппе H, и пусть m — произвольный ненулевой ее элемент. Разделим m на n с остатком, т. е. найдем такие $q, r \in \mathbb{Z}$, что $0 < r < n$ и

$$m = nq + r.$$

Но тогда по свойствам подгруппы $r \in H$. По выбору n остаток r должен быть нулевым, т. е. любой элемент подгруппы H кратен n. ■

Из леммы непосредственно следует, что любая ненулевая подгруппа в $\mathbb{Z}^+$ имеет бесконечный порядок. Кроме того, комбинируя ее с предыдущим замечанием о цепочках подгрупп, можно получить хорошее определение простых чисел.

Определение А.32. Простое число является (положительной) образующей собственной подгруппы H в $\mathbb{Z}^+$, т. е. не нулевой и не совпадающей с $\mathbb{Z}^+$, для которой нет таких собственных подгрупп в $\mathbb{Z}^+$, отличных от H, которые содержали бы все элементы H.

Это определение хорошо тем, что оно непосредственно не апеллирует к мультипликативной структуре группы $\mathbb{Z}^+$. Кроме того, из него сразу следует, что ни 0, ни 1 не являются простыми числами. Кроме того, его можно обобщить на другие ситуации. Подробности обобщений можно узнать из стандартного учебника по абстрактной алгебре.

А.6.1. Нормальные подгруппы и классы смежности

Нормальные подгруппы очень важны в теории групп. Не следует считать, что эти подгруппы получаются каким-то нормальным способом, название — всего лишь дань традиции. Перед определением нормальной подгруппы следует сказать несколько слов о сопряженных элементах.

Определение А.33. Говорят, что элементы x и y группы G сопряжены, если существует такой элемент $g \in G$, что $x = g^{-1}yg$.

Легко проверить, что два сопряженных элемента имеют одинаковый порядок. Если N — подгруппа в G, а $g \in G$, то введем обозначение

$$g^{-1}Ng = \left\{g^{-1}xg |\ x \in N\right\}.$$

Очевидно, $g^{-1}Ng$ тоже является подгруппой в G. Ее обычно называют подгруппой, *сопряженной* с N.

Определение А.34. Подгруппа $N < G$ называется *нормальной*, если для каждого $g \in G$ имеет место включение $g^{-1}Ng \subset N$. Нормальность подгруппы обозначают как $N \triangleleft G$.

В любой группе есть по крайней мере две нормальные подгруппы: $G \triangleleft G$ и $\{e\} \triangleleft G$. Если G — абелева группа, то любая ее подгруппа будет нормальной. Особое значение нормальных подгрупп состоит в том, что по ним можно факторизовать. Факторизация имеет непосредственное отношение к классам смежности, которые мы сейчас опишем.

Определение А.35. Пусть H — подгруппа (не обязательно нормальная) в группе G. Фиксируем $g \in G$ и определим *левый класс смежности* как

$$gH = \{gh |\ h \in H\}.$$

Аналогично определяется *правый класс смежности*:

$$Hg = \{hg |\ h \in H\}.$$

Можно показать что множество всех левых (правых) классов смежности по подгруппе H образует разбиение группы G. Кроме того, если $a, b \in G$, то $aH = bH$ в том и только том случае, когда $b^{-1}a \in H$, что эквивалентно условию $a^{-1}b \in H$. Проверку этих фактов мы оставим читателю. Обратите внимание на то, что бывают равные смежные классы $aH = bH$ с разными представителями $a \neq b$.

Сформулированные утверждения говорят о том, что отношение на множестве элементов группы G, определяемое правилом

$$a \sim b \Longleftrightarrow ab^{-1} \in H,$$

является отношением эквивалентности. Классы эквивалентности — это в точности левые классы смежности по подгруппе H.

Число левых классов смежности по подгруппе H в G обозначается символом $(G : H)_L$, а число правых — $(G : H)_R$. Теперь мы в состоянии доказать одну из фундаментальных теорем теории конечных групп, а именно, теорему Лагранжа.

Теорема А.36. (Лагранжа) Пусть H — подгруппа конечной группы G. Тогда

$$|G| = (G : H)_L \cdot |H| = (G : H)_R \cdot |H|.$$

Доказательству этого результата мы предпошлем формулировку важного следствия из него.

Следствие А.37. Имеет место равенство $(G : H)_L = (G : H)_R$. Это число обозначается через $(G : H)$ и называется индексом подгруппы H в G. Как порядок подгруппы, так и ее индекс делят порядок группы. Если порядок группы G — простое число, то она не имеет собственных подгрупп.

Вернемся к доказательству теоремы Лагранжа.

Доказательство. Индуктивно сформируем последовательность разных смежных классов по подгруппе H, начиная с представителя $g_1 = e$. Первый левый смежный класс eH совпадает с самой подгруппой. Предположим, что мы уже построили разные смежные классы g_1H, g_2H, ..., g_iH. Возьмем элемент g_{i+1}, не попавший ни в один из построенных смежных классов. Тогда новый класс $g_{i+1}H$ не будет совпадать ни с одним из построенных ранее, и мы можем удлинить последовательность. Ясно, что этот процесс кончится в тот момент, когда мы исчерпаем все элементы группы. Таким образом, мы получили множество непересекающихся левых смежных классов, покрывающих всю группу:

$$G = \bigcup_{1 \leqslant i \leqslant (G:H)_L} g_iH.$$

Кроме того, для любой пары индексов имеет место равенство $|g_iH| = |g_jH|$, что следует из биективности отображения

$$H \longrightarrow gH, \qquad h \mapsto gh.$$

Следовательно,

$$|G| = \sum_{1 \leqslant i \leqslant (G:H)_L} |g_iH| = (G : H)_L |H|.$$

Равенство для правых смежных классов получается аналогично. ■

Из следствия после теоремы Лагранжа выводится следующая лемма.

Лемма А.38. Группа простого порядка является циклической.

Доказательство. Если $g \in G$ не равен единице, то $\langle g \rangle$ — подгруппа в G, порядок которой больше 1. Но этот порядок должен делить порядок G, т. е. простое число. Значит $|\langle g \rangle| = |G|$, откуда $\langle g \rangle = G$, что означает цикличность группы G. ■

С помощью теоремы Лагранжа можно выписать все подгруппы в группах небольшого порядка. Например, рассмотрим симметрическую группу S_3. Ее порядок равен 6. Поэтому, согласно теореме Лагранжа, порядок ее подгрупп может быть равен 1, 2, 3 или 6. Легко видеть, что все ее подгруппы представлены следующим списком:

- одна подгруппа порядка 1, а именно $\langle (1) \rangle$;
- три подгруппы порядка 2: $\langle (1,2) \rangle$, $\langle (1,3) \rangle$ и $\langle (2,3) \rangle$;
- одна подгруппа порядка 3, а именно $\langle (1,2,3) \rangle$;
- одна подгруппа порядка 6 — сама G.

А.6.2. Факторгруппы

Всюду в этом разделе символ G закреплен за группой с нормальной подгруппой N. Следующая элементарная лемма, доказательство которой мы опять оставим читателю, объясняет наш повышенный интерес к нормальным подгруппам.

Лемма А.39. Пусть H — подгруппа в G. Тогда следующие утверждения эквивалентны:

1. $xH = Hx$ для любого $x \in G$.
2. $x^{-1}Hx = H$ для всех $x \in G$.
3. $H \lhd G$.
4. $x^{-1}hx \in H$ для всех $x \in G$ и $h \in H$.

Обозначим через G/N множество всех левых смежных классов N. Заметим, что оно совпадает с множеством правых смежных классов. Напомним, что два смежных класса g_1H и g_2H равны тогда и только тогда, когда $g_1^{-1}g_2 \in N$.

Мы хотим наделить множество G/N структурой группы. Если получится, то эта группа будет называться факторгруппой G по подгруппе N. Для этого нам необходимо определить групповую операцию на множестве смежных классов. Пусть g_1N и g_2N — два

смежных класса. Определим их произведение как смежный класс $(g_1g_2)N$.

Прежде всего следует проверить корректность определения операции, т. е. что результат умножения смежных классов не зависит от выбора представителей. Возьмем другие представители g_1' и g_2'. Тогда $g_1^{-1}g_1' = n_1 \in N$ и $g_2^{-1}g_2' = n_2 \in N$. Надо убедиться, что

$$(g_1g_2)N = (g_1'g_2')N.$$

Пусть $x \in (g_1g_2)N$. Тогда $x = g_1g_2n$ для некоторого $n \in N$. Этот элемент можно переписать в виде $x = g_1'n_1^{-1}g_2'n_2^{-1}n$. Но так как N — нормальная подгруппа (левые классы смежности совпадают с правыми), то ${n_1'}^{-1}g_2' = g_2'n_3$ для некоторого $n_3 \in N$. Следовательно,

$$x = g_1'g_2'n_3n_2^{-1}n \in g_1'g_2'N.$$

Это рассуждение дает нам включение $(g_1g_2)N \subset (g_1'g_2')N$, а обратное получается аналогично. Значит, $(g_1g_2)N = (g_1'g_2')N$ и операция определена корректно. Можно показать, что если таким же способом задать групповую операцию на множестве левых смежных классов по произвольной подгруппе H, то она будет корректно определена в том и только том случае, когда H — нормальная подгруппа в G.

Итак, у нас есть корректно определенная операция на G/N и нам нужно проверить, что она удовлетворяет всем аксиомам группы.

- В качестве единицы выступает смежный класс $eN = N$, поскольку для всех $g \in G$

$$eN \cdot gN = (eg)N = gN.$$

- Обратным к классу gN служит смежный класс $g^{-1}N$. Действительно,

$$gN \cdot g^{-1}N = (gg^{-1})N = eN = N.$$

- Ассоциативность следует из преобразований:

$$\begin{aligned}(g_1N)(g_2N \cdot g_3N) &= g_1N((g_2g_3)N) = (g_1(g_2g_3))N = \\ &= ((g_1g_2)g_3N = ((g_1g_2)N)g_3N = \\ &= (g_1N \cdot g_2N)(g_3N).\end{aligned}$$

Приведем несколько примеров.

1. Пусть G — произвольная конечная группа, порядок которой больше 1. Тогда $H = G$ и $H = \{e\}$ — нормальные подгруппы в G.

2. Если $H = G$, то существует всего один класс смежности по подгруппе H. Поэтому в этом случае $G/G = \{e\}$ — группа порядка 1.
3. Если $H = \{e\}$, то любой смежный класс по H состоит ровно из одного элемента группы G. Очевидно, здесь $G/H = G$.
4. Рассмотрим группу S_3 и ее подгруппу N, состоящую из перестановок $\{(1), (1,2,3), (1,3,2)\}$. Поскольку $|S_3| = 6$, а $|N| = 3$, существует только два смежных класса. Отсюда легко получить, что N — нормальная подгруппа. Более того, $|G/N| = 2$. Значит мы получаем группу, состоящую из двух элементов: a и e. Легко понять, что на ней существует единственно возможная групповая операция, а именно: $ea = ae = a$, $aa = e$ и $ee = e$. Значит, это циклическая группа порядка 2.
5. Если G абелева группа, то любая ее подгруппа H нормальна и существует факторгруппа G/H.
6. Поскольку $\mathbb{Z}^+$ абелева группа, то $m\mathbb{Z}$ — ее нормальная подгруппа и можно рассмотреть факторгруппу $\mathbb{Z}/m\mathbb{Z}$. Получится группа целых чисел по модулю n с операцией суммы.

А.6.3. Гомоморфизмы

Пусть G_1 и G_2 — группы. Нам хотелось бы исследовать функции из G_1 в G_2, но не все, а лишь те, которые сохраняют групповую операцию.

Определение А.40. *Гомоморфизмом групп* (или просто гомоморфизмом) из группы G_1 в группу G_2 называется функция

$$f : G_1 \longrightarrow G_2,$$

сохраняющая групповую операцию, т. е. для любых $x, y \in G$ выполнено свойство

$$f(x \cdot y) = f(x) \cdot f(y).$$

Заметим, что умножение в левой части равенства происходит по групповому закону в группе G_1, а в правой части — в группе G_2. Обратимся к примерам.

1. Тождественное отображение $\mathrm{id}_G : G \longrightarrow G$, при котором $\mathrm{id}_G(g) = g$ для всех $g \in G$, является гомоморфизмом групп.
2. Функция $\exp : \mathbb{R}^+ \longrightarrow \mathbb{R}^*$, заданная формулой $\exp(x) = e^x$, является гомоморфизмом групп, поскольку

$$e^{x+y} = e^x e^y.$$

3. Гомоморфизмом будет и функция abs: $\mathbb{C}^* \longrightarrow \mathbb{R}^*$, определяемая формулой: $\operatorname{abs}(z) = |z|$.
4. Функция $\det : \mathrm{GL}_n(\mathbb{C}) \longrightarrow \mathbb{C}$, сопоставляющая квадратной матрице ее определитель, — тоже гомоморфизм. т. к.

$$\det(AB) = \det(A) \cdot \det(B)$$

для любой пары квадратных матриц.

Следующая лемма объединяет два элементарных свойства гомоморфизмов.

Лемма А.41. Пусть $f : G_1 \longrightarrow G_2$ — гомоморфизм групп. Тогда
1. $f(e_1) = e_2$, где e_i — единица группы G_i.
2. Для всех $x \in G_1$ выполнено тождество $f(x^{-1}) = (f(x))^{-1}$.

Доказательство. В первом случае получаем $e_2 f(x) = f(x) = f(e_1 x) = f(e_1)f(x)$. Поэтому

$$e_2 = f(x)f(x)^{-1} = f(e_1)f(x)f(x)^{-1} = f(e_1).$$

Во втором случае

$$f(x^{-1})f(x) = f(x^{-1}x) = f(e_1) = e_2.$$

Значит, $f(x^{-1})$ и $f(x)$ взаимно обратны. ■

С каждым гомоморфизмом канонически ассоциированы две специальные подгруппы.

Определение А.42. *Ядром* гомоморфизма f называется множество

$$\operatorname{Ker} f = \{x \in G_1 |\ f(x) = e_2\}.$$

Образом гомоморфизма f называется множество

$$\operatorname{Im} f = \{y \in G_2 |\ y = f(x),\ x \in G_1\}.$$

Лемма А.43. $\operatorname{Ker} f$ — нормальная подгруппа в G_1.

Доказательство. Прежде всего покажем, что ядро образует подгруппу. Оно непусто, поскольку $e_1 \in \operatorname{Ker} f$. Пусть $x \in \operatorname{Ker} f$. Тогда $f(x^{-1}) = f(x)^{-1} = e_2^{-1} = e_2$. Значит, вместе с каждым элементом в $\operatorname{Ker} f$ туда попадает и обратный к нему. Поэтому нам достаточно показать, что $\operatorname{Ker} f$ замкнуто относительно умножения, т. е. если $x, y \in \operatorname{Ker} f$, то $x \cdot y \in \operatorname{Ker} f$. Но

$$f(x \cdot y) = f(x)f(y) = e_2 \cdot e_2 = e_2.$$

Итак, $\operatorname{Ker} f$ — действительно подгруппа в G_1. Теперь нужно показать, что это нормальная подгруппа. Согласно лемме А.39, достаточно проверить, что для любых элементов $x \in G_1$ и $h \in \operatorname{Ker} f$ выполнено включение: $x^{-1}hx \in \operatorname{Ker} f$. Это следует из определения ядра и преобразований

$$f(x^{-1}hx) = f(x)^{-1}f(h)f(x) = f(x)^{-1}e_2 f(x) = f(x)^{-1}f(x) = e_2. \quad \blacksquare$$

Лемма А.44. $\operatorname{Im} f$ является подгруппой в G_2.

Доказательство. Ясно, что $\operatorname{Im} f$ — непустое множество, поскольку $f(e_1) = e_2$, т. е. $e_2 \in \operatorname{Im} f$. Пусть $y \in \operatorname{Im} f$. Это означает, что найдется $x \in G_1$, для которого $f(x) = y$. Тогда $f(x^{-1}) = f(x)^{-1} = y^{-1}$. Значит, $y^{-1} \in \operatorname{Im} f$.

Осталось показать, что вместе с любыми y_1, $y_2 \in \operatorname{Im} f$ произведение $y_1 \cdot y_2$ тоже лежит в $\operatorname{Im} f$. По предположению существуют такие $x_1, x_2 \in G_1$, что $f(x_i) = y_i$. Следовательно, $f(x_1 \cdot x_2) = f(x_1) \cdot f(x_2) = y_1 \cdot y_2$. Таким образом, $y_1 \cdot y_2 \in \operatorname{Im} f$. $\blacksquare$

Ясно, что $\operatorname{Im} f$ в некотором смысле определяет, насколько функция f близка к (или далека от) сюръективной. Более точно, f сюръективна тогда и только тогда, когда $\operatorname{Im} f = G_2$. Фактически, множество смежных классов $G_2/\operatorname{Im} f$ служит для этой цели лучше. С другой стороны, подгруппа $\operatorname{Ker} f$ говорит, насколько функция f отличается от инъективной, благодаря следующему результату.

Лемма А.45. Гомоморфизм $f : G_1 \longrightarrow G_2$ инъективен тогда и только тогда, когда $\operatorname{Ker} f = \{e_1\}$.

Доказательство. Предположим, что f — инъективная функция. Тогда из равенства $f(x) = e_2 = f(e_1)$ немедленно следует, что $x = e_1$. Значит, $\operatorname{Ker} f = \{e_1\}$ в этом случае.

Допустим теперь, что $\operatorname{Ker} f = \{e_1\}$, но найдется пара элементов x, $y \in G_1$, для которой $f(x) = f(y)$. Тогда

$$f(xy^{-1}) = f(x)f(y^{-1}) = f(x)f(y)^{-1} = f(y)f(y)^{-1} = e_2.$$

Отсюда следует, что $xy^{-1} \in \operatorname{Ker} f$, т. е. $xy^{-1} = e_1$, откуда $x = y$. Значит, f — инъективная функция. $\blacksquare$

Биективные гомоморфизмы позволяют нам говорить о «равенстве» групп.

Определение А.46. Гомоморфизм $f : G_1 \longrightarrow G_2$ называется *изоморфизмом*, если f — биективная функция. Группы, между которыми существует изоморфизм, называются *изоморфными*. Этот факт обозначается символом $G_1 \simeq G_2$.

Заметим, что изоморфные конечные группы обладают одинаковым количеством элементов. На самом деле, во многих приложениях изоморфные группы можно считать просто равными друг другу, поскольку они выглядят абсолютно одинаково с точностью до переобозначения элементов. Изоморфизмы обладают следующими свойствами:

- любая группа G изоморфна себе ввиду наличия тождественного изоморфизма, т. е. отношение изоморфности на множестве всех групп рефлексивно;
- если $f : G_1 \longrightarrow G_2$ и $g : G_2 \longrightarrow G_3$ — изоморфизмы, то композиция $g \circ f$ тоже будет изоморфизмом, т. е. отношение изоморфности является транзитивным;
- обратная функция $f^{-1} : G_2 \longrightarrow G_1$ к любому изоморфизму $f : G_1 \longrightarrow G_2$ является изоморфизмом, что свидетельствует о симметричности отношения изоморфности.

Из этих свойств вытекает, что понятие изоморфизма групп индуцирует отношение эквивалентности на множестве всех групп, что оправдывает наше стремление отождествить понятие изоморфизма с понятием равенства.

Пусть G_1 и G_2 — группы. Определим прямое произведение групп $G_1 \times G_2$ как множество упорядоченных пар (g_1, g_2), где $g_i \in G_i$, с покомпонентной операцией:

$$(g_1, g_2) \cdot (g_1', g_2') = (g_1 \cdot g_1', g_2 \cdot g_2'),$$

где умножение в первой компоненте пары происходит по групповому закону в первой группе, а во второй компоненте — во второй. В качестве примера рассмотрим отображение

$$\mathbb{C}^+ \longrightarrow \mathbb{R}^+ \times \mathbb{R}^+, \qquad z \mapsto (\mathrm{Re}(z), \mathrm{Im}(z)).$$

Это отображение, очевидно, является биективным гомоморфизмом. Так что $\mathbb{C}^+ \simeq \mathbb{R}^+ \times \mathbb{R}^+$.

Теперь мы сформулируем важную теорему, показывающую, что понятие факторгруппы и образа гомоморфизма практически совпадают.

Теорема А.47. (Первая теорема об изоморфизме групп.) Пусть $f : G_1 \longrightarrow G_2$ — гомоморфизм групп. Тогда $G_1/\mathrm{Ker}\, f \simeq \mathrm{Im}\, f$.

Доказательство теоремы можно найти в любом учебнике по высшей алгебре. Заметим, что факторгруппа $G_1/\mathrm{Ker}\, f$ определена корректно, поскольку $\mathrm{Ker}\, f$ — нормальная подгруппа в G_1.

А.7. Кольца

Кольцом называют абелеву группу по сложению с дополнительной операцией умножения. Умножение должно быть ассоциативной операцией, причем умножение и сложение должны быть связаны законом *дистрибутивности*:

$$a \cdot (b + c) = a \cdot b + a \cdot c, \quad (b + c) \cdot a = b \cdot a + c \cdot a.$$

Если умножение обладает единичным элементом, то говорят о кольце с единицей. Кольцо с коммутативной операцией умножения называют коммутативным.

Примеры коммутативных колец с единицей:

- целые числа с обычными операциями сложения и умножения;
- множество $\mathbb{Z}[x]$ многочленов от переменной x с коэффициентами в $\mathbb{Z}$;
- целые числа по модулю n — кольцо вычетов по модулю n, обозначаемое $\mathbb{Z}/n\mathbb{Z}$.

Хотя в теории колец можно рассматривать подкольца, это не интересно для нас. Куда большее значение имеет понятие идеала. Пусть R — кольцо. *Идеалом* I (правым) в кольце R называется аддитивная подгруппа, выдерживающая умножения на элементы кольца (справа), т. е.

$$\forall i \in I, \quad \forall r \in R \qquad i \cdot r \in I.$$

Простейшим примером идеала служат главные идеалы, т. е. идеалы вида

$$(a) = aR = \{a \cdot r | \ r \in R\},$$

порожденные одним элементом $a \in R$. Если $R = \mathbb{Z}$, то любой идеал I в нем главный, т. е. $I = m\mathbb{Z}$ для некоторого целого числа m. Но по таким идеалам мы можем взять фактор, в частности, $\mathbb{Z}/m\mathbb{Z}$ является кольцом, где сложение и умножение происходит по модулю m. Это естественным образом подводит нас к китайской теореме об остатках.

Теорема А.48. (Китайская теорема об остатках.) Пусть $m = p_1^{z_1} \cdots p_t^{z_t}$ — разложение числа m на простые множители. Тогда отображение

$$f: \begin{array}{ccc} \mathbb{Z}/m\mathbb{Z} & \to & \mathbb{Z}/p_1^{z_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_t^{z_t}\mathbb{Z} \\ x & \mapsto & \left(x \pmod{p_1^{z_1}}, \ldots, x \pmod{p_t^{z_t}}\right) \end{array}$$

является изоморфизмом колец.

Этот факт может быть доказан индукцией по числу простых делителей числа m. Оставим его доказательство читателю.

Вернемся к функции Эйлера φ, встреченной нами ранее. Напомним, что $\varphi(n)$ обозначает порядок мультипликативной группы $(\mathbb{Z}/n\mathbb{Z})^*$. Нам бы хотелось легко вычислять значения функции φ.

Лемма А.49. Пусть $m = p_1^{z_1} \cdots p_t^{z_t}$ — разложение на простые множители числа m. Тогда

$$\varphi(m) = \varphi(p_1^{z_1}) \cdots \varphi(p_t^{z_t}).$$

Доказательство. Это равенство непосредственно следует из китайской теоремы об остатках, т. к. изоморфизм колец

$$\mathbb{Z}/m\mathbb{Z} \simeq \mathbb{Z}/p_1^{z_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_t^{z_t}\mathbb{Z}$$

индуцирует изоморфизм групп

$$(\mathbb{Z}/m\mathbb{Z})^* \simeq (\mathbb{Z}/p_1^{z_1}\mathbb{Z})^* \times \cdots \times (\mathbb{Z}/p_t^{z_t}\mathbb{Z})^*. \qquad \blacksquare$$

Теперь для вычисления значений функции Эйлера нам достаточно следующей леммы.

Лемма А.50. Если p — простое число, то $\varphi(p^e) = p^{e-1}(p-1)$.

Доказательство. Прежде всего отметим, что $\varphi(m)$ совпадает с количеством взаимно простых с m чисел k, которые удовлетворяют условию: $1 \leqslant k < m$. Общее количество чисел, не превосходящих p^e, очевидно равно $p^e - 1$. Теперь нам надо выбросить из них те, которые имеют с p^e нетривиальный общий делитель, т. е. числа вида $k = rp$, где $1 \leqslant rp < p^e$. Последнее неравенство влечет, что $1 \leqslant r < p^{e-1}$. Значит существует ровно $p^{e-1} - 1$ не взаимно простых с p^e чисел, принадлежащих кольцу $\mathbb{Z}/p^e\mathbb{Z}$. Следовательно,

$$\varphi(p^e) = (p^e - 1) - (p^{e-1} - 1) = p^{e-1}(p-1). \qquad \blacksquare$$

Идеал I кольца называется максимальным, если он не содержится ни в каком другом идеале кольца (исключая само кольцо). Например, идеал $m\mathbb{Z}$ будет максимальным тогда и только тогда, когда m — простое число. Факторкольцо по максимальному идеалу будет уже не только кольцом, но еще и полем.

А.8. Поля

Поле — это, по существу, структуры двух абелевых групп на одном множестве, связанные между собой законом дистрибутивности.

Определение А.51. *Полем* называется такая аддитивная абелева группа F, что $F\setminus\{0\}$ тоже образует абелеву группу, но относительно умножения. Сложение и умножение в поле должны удовлетворять следующему свойству дистрибутивности:

$$a\cdot(b+c)=a\cdot b+a\cdot c.$$

Чаще всего встречающиеся поля имеют бесконечно много элементов. Любое поле содержит в качестве подполя либо рациональные числа, либо поле $\mathbb{F}_p=\mathbb{Z}/p\mathbb{Z}$. В первом случае говорят, что характеристика поля равна 0, а во втором — p. Любое конечное поле состоит из p^r элементов с простым p и натуральным r. Все поля с таким числом элементов изоморфны и обозначаются $\mathbb{F}_{p^r}$. Иногда конечные поля называют полями Галуа.

Пусть F — поле, символом $F[X]$ обозначим кольцо многочленов от переменой X с коэффициентами из поля F. Множеством $F(X)$ рациональных функций от X называют совокупность функций вида $f(X)/g(X)$, где $f(X),g(X)\in F[X]$ и $g(X)$ — ненулевой многочлен. Рациональные функции образуют поле относительно очевидных операций сложения и умножения. Следует помнить о разных скобках, обозначающих кольцо многочленов $F[X]$ и поле рациональных функций $F(X)$.

Пусть f — неприводимый многочлен степени n с коэффициентами из $\mathbb{F}_p$. Обозначим через θ его формальный корень и рассмотрим множество

$$\mathbb{F}_p(\theta)=\left\{a_0+a_1\theta+\cdots+a_{n-1}\theta^{n-1}\mid a_i\in\mathbb{F}_p\right\}.$$

Два таких выражения можно сложить, сложив соответствующие коэффициенты, и умножить по обычному правилу, но потом взять остаток от деления произведения на $f(\theta)$. Эти операции наделяют $\mathbb{F}_p(\theta)$ структурой поля. Существует изоморфизм полей

$$\mathbb{F}_{p^n}=\mathbb{F}_p(\theta)=\mathbb{F}_p[X]/(f),$$

где (f) — главный идеал кольца $\mathbb{F}_p[X]$, порожденный f. Для конкретности рассмотрим простое число p вида $p=4k+3$ (k — натуральное число) и многочлен $f(X)=X^2+1$. Легко проверить, что поскольку $p=3\pmod 4$, многочлен $f(X)$ неприводим над полем $\mathbb{F}_p$ и факторкольцо $\mathbb{F}_p[X]/(f)$ является полем, изоморфным полю $\mathbb{F}_{p^2}$.

Пусть i — формальный корень многочлена $f(X)=X^2+1$. Поле $\mathbb{F}_{p^2}=\mathbb{F}_p(i)$ состоит из чисел вида

$$a+bi,$$

где a и b — целые числа по модулю p. Складываются эти числа по правилу

$$(a+bi)+(c+di)=(a+c)+(b+d)i,$$

а умножаются так:

$$(a+bi)\cdot(c+di)=(ac+(ad+bc)i+bdi^2)=(ac-bd)+(ad+bc)i.$$

Рассмотрим еще один пример. Пусть θ — формальный корень многочлена X^3+2. Тогда элементы поля $\mathbb{F}_{7^3}=\mathbb{F}_7(\theta)$ записываются как

$$a+b\theta+c\theta^2.$$

Перемножим два таких элемента:

$$\begin{aligned}&(a+b\theta+c\theta^2)\cdot(a'+b'\theta+c'\theta^2)=\\&=aa'+\theta(a'b+b'a)+\theta^2(ac'+bb'+ca')+\theta^3(bc'+cb')+cc'\theta^4=\\&=(aa'-2bc'-2cb')+\theta(a'b+b'a-2cc')+\theta^2(ac'+bb'+ca').\end{aligned}$$

А.9. Векторные пространства

Определение А.52. *Векторным (линейным) пространством* V над полем K называется абелева группа (тоже обозначаемая через V) с дополнительной операцией $K\times V\longrightarrow V$ (называемой умножением на скаляры), удовлетворяющая следующим аксиомам: если $\lambda,\mu\in K$ и $x,y\in V$, то

(а) $\lambda(\mu x)=(\lambda\mu)x$;
(б) $(\lambda+\mu)x=\lambda x+\mu x$;
(в) $\lambda(x+y)=\lambda x+\lambda y$;
(г) $1\cdot x=x$.

Элементы группы V обычно называют векторами, а элементы поля K — числами или скалярами. Обратите внимание, что об умножении векторов на векторы или о делении их речи не идет. Начнем с примеров.

– Пусть K — поле и $K^n=\underbrace{K\times K\times\cdots\times K}_{n}$ — прямое произведение n экземпляров поля. Это множество является векторным пространством относительно обычного покомпонентного сложения и умножения на числа. Особый случай $n=1$ свидетельствует, что любое поле можно рассматривать как векторное пространство над собой. Если $K=\mathbb{R}$ и $n=2$, то $\mathbb{R}^2$ — знакомая система геометрических векторов на плоскости. Если

$K = \mathbb{R}$ и $n = 3$, то получается совокупность векторов в пространстве. Следовательно, можно считать, что K^n — n-мерное векторное пространство.

- Рассмотрим множество многочленов $K[X]$ над полем K. Оно является векторным пространством над K с обычным сложением и умножением на элементы K.
- Пусть E — произвольное множество. Наделим множество функций $f : E \longrightarrow K$ со значениями в поле K структурой векторного пространства. Для этого положим по определению:

$$(f + g)(x) = f(x) + g(x) \quad \text{и} \quad (\lambda f)(x) = \lambda f(x).$$

- Множество всех непрерывных функций $f : \mathbb{R} \longrightarrow \mathbb{R}$ является векторным пространством над $\mathbb{R}$, поскольку сумма непрерывных функций непрерывна и произведение непрерывной функции на число — тоже непрерывная функция.

А.9.1. Подпространства

Пусть V — векторное пространство над полем K. Его подпространством W называют такое подмножество в V, что W — подгруппа в V относительно сложения и для всех $w \in W$ и $\lambda \in K$ имеет место включение: $\lambda w \in W$. Последнее свойство именуют замкнутостью W относительно умножения на скаляры. Фактически, подпространство само является векторным пространством над тем же полем относительно операций в основном пространстве. В любом векторном пространстве V есть два тривиальных подпространства: нулевое $\{0\}$ и само V. Обратимся к менее тривиальным примерам.

- $V = K^n$ и $W = \{(\xi_1, \ldots, \xi_n) \in K^n | \ \xi_n = 0\}$;
- $V = K^n$ и $W = \{(\xi_1, \ldots, \xi_n) \in K^n | \ \xi_1 + \cdots + \xi_n = 0\}$;
- $V = K[X]$ и $W = \{f \in K[X] | \ f = 0 \text{ или } \deg f \leqslant 10\}$;
- $\mathbb{C}$ как естественное векторное пространство над $\mathbb{Q}$ и $\mathbb{R}$ как подпространство в $\mathbb{C}$;
- пусть V — множество всех непрерывных функций из $\mathbb{R}$ в $\mathbb{R}$, тогда W, множество всех дифференцируемых функций из $\mathbb{R}$ в $\mathbb{R}$, является подпространством в V.

А.9.2. Свойства векторов

Для дальнейшего рассказа нам понадобятся некоторые свойства семейств векторов линейного пространства. В следующем определе-

нии через V обозначено векторное пространство, а через $x_1, \ldots, x_n$ его элементы, т. е. векторы.

Определение А.53. Вектор x называется *линейной комбинацией* векторов $x_1, \ldots, x_n$, если для некоторых скаляров $\lambda_i \in K$ выполнено равенство

$$x = \lambda_1 x_1 + \cdots \lambda_n x_n.$$

Векторы $x_1, \ldots, x_n$ называются *линейно независимыми*, если соотношение

$$\lambda_1 x_1 + \cdots \lambda_n x_n = 0$$

возможно только при $\lambda_1 = \cdots = \lambda_n = 0$. Если это не так, то векторы называются *линейно зависимыми*.

Подмножество A векторного пространства V называется *линейно независимым* или *свободным*, если любое его конечное подмножество векторов линейно независимо.

Подмножество A векторного пространства называется *полным*, если любой вектор из V представляется линейной комбинацией конечной системы векторов из A.

Векторное пространство называется *конечномерным*, если в нем существует конечная полная система векторов.

Приведем примеры конечномерных пространств.

- Пространство $V = K^n$ конечномерно, т. к. система векторов

$$e_1 = (1, 0, 0, \ldots, 0),$$
$$e_2 = (0, 1, 0, \ldots, 0),$$
$$\ldots$$
$$e_n = (0, 0, 0, \ldots, 1)$$

  является полной в V. Обратите внимание на аналогию этого примера с геометрической плоскостью.
- $\mathbb{C}$ — конечномерное пространство над $\mathbb{R}$, т. к. элементы $\{1, \sqrt{-1}\}$ образуют полную систему.
- Как $\mathbb{R}$, так и $\mathbb{C}$ — бесконечномерные векторные пространства над $\mathbb{Q}$. Доказательство следует из Канторовской теории множеств: $\mathbb{Q}$ —счетное множество, поэтому любое конечномерное пространство над $\mathbb{Q}$ должно состоять из счетного числа векторов. Но как $\mathbb{R}$, так и $\mathbb{C}$ — несчетные множества.

Приведем примеры линейно независимых систем векторов.

- В пространстве $V = K^n$ определенные в предыдущих примерах векторы $e_1, \ldots, e_n$ являются линейно независимыми.

- Векторы $x_1 = (1, 2, 3)$, $x_2 = (-1, 0, 4)$ и $x_3 = (2, 5, -1)$ линейно независимы как векторы из $\mathbb{R}^3$.
- С другой стороны, векторы $y_1 = (2, 4, -3)$, $y_2 = (1, 1, 2)$ и $y_3 = (2, 8, -17)$ линейно зависимы, т. к. $3y_1 - 4y_2 - y_3 = 0$.
- В векторном пространстве $K[X]$ над полем K бесконечная система векторов
$$\{1, X, X^2, X^3, \ldots\}$$
линейно независима.

А.9.3. Размерность и базисы

Определение А.54. Полная линейно независимая система векторов в векторном пространстве V называется *базисом*.

Если в пространстве V фиксирован базис $x_1, \ldots, x_n$, то любой вектор x из V однозначно представляется линейной комбинацией
$$x = \lambda_1 x_1 + \cdots \lambda_n x_n.$$
Действительно, если есть еще одна такая линейная комбинация $x = \mu_1 x_1 + \cdots \mu_n x_n$, то вычитая одну из другой, получим
$$0 = x - x = (\lambda_1 - \mu_1)x_1 + \cdots + (\lambda_n - \mu_n)x_n.$$
Поскольку векторы базиса линейно независимы, последнее равенство возможно только когда $\lambda_i = \mu_i$ для всех i.

Примеры.

- Векторы $e_1, \ldots, e_n$ в K^n являются базисом, который называют *стандартным*.
- Множество $\{1, \sqrt{-1}\}$ — базис $\mathbb{C}$ над $\mathbb{R}$.
- Бесконечная последовательность $\{1, X, X^2, X^3, \ldots\}$ образует базис пространства $K[X]$.

Чтобы упростить формулировку следующей теоремы, будем считать, что базисом нулевого векторного пространства V является пустое множество.

Теорема А.55. Пусть в конечномерном векторном пространстве V фиксировано конечное полное множество векторов C, а в нем выбрано линейно независимое подмножество A. Тогда в V найдется базис B, удовлетворяющий условию: $A \subset B \subset C$.

Доказательство. Можно предполагать, что $V \neq \{0\}$. Рассмотрим совокупность всех линейно независимых подмножеств в C, содержащих A. К такой совокупности относится и само множество A,

поскольку оно линейно независимо. Возьмем среди них такое множество B, которое содержит максимально возможное количество элементов и покажем, что оно полно в V.

Поскольку C — полная система векторов в V, нам достаточно показать, что любой вектор $x \in C$ выражается в виде линейной комбинации векторов из B.

Вектор из B, естественно, представляется линейной комбинацией векторов из B. Поэтому будем считать, что $x \notin B$ и рассмотрим новое множество $B' = B \cup \{x\} \subset C$. Оно содержит больше элементов, чем B. Значит, B' — линейно зависимая система векторов. Если $x_1, \ldots, x_r, x$ — все векторы из B', то по определению линейной зависимости существует соотношение

$$\lambda_1 x_1 + \cdots + \lambda_r x_r + \lambda x = 0,$$

в котором не все коэффициенты $\lambda_1, \ldots, \lambda_r, \lambda$ равны нулю. Если при этом $\lambda = 0$, то мы получим, что векторы $x_1, \ldots, x_r$ — линейно зависимы, что противоречит предположению о линейной независимости B. Следовательно, $\lambda \neq 0$ и вектор x легко выражается из этого соотношения. Таким образом, мы сможем выразить любой элемент из C (а значит и из V) в виде линейной комбинации векторов из B. ■

Следствие А.56. Любое конечномерное векторное пространство V обладает базисом.

Доказательство. Опять можно считать, что $V \neq \{0\}$. Пусть C — конечная полная система векторов из V (существующая по определению конечномерного пространства), а $A = \{e_1\}$ — подмножество в C, состоящее из любого ненулевого вектора. Ясно, что A — линейно независимо, поэтому осталось применить теорему А.55. ■

Теорема А.55 и следствие из нее останутся верными, даже если опустить предположение о конечномерности пространства V. Но в общем случае их доказательство существенно сложнее. Следующий результат имеет большое значение в теории векторных пространств, т. к. позволяет ввести понятие размерности, обобщающее естественное понятие двумерной плоскости и трехмерного пространства.

Теорема А.57. Пусть $A = \{x_1, x_2, \ldots, x_m\}$ — полная система векторов в V, а $B = \{y_1, y_2, \ldots, y_n\}$ — линейно независимая система в V. Тогда $n \leqslant m$.

Доказательство. Благодаря полноте системы A, вектор $y_1 \in B$

можно представить в виде линейной комбинации

$$y_1 = \lambda_1 x_1 + \cdots + \lambda_m x_m.$$

Вектор $y_1 \neq 0$, поэтому хотя бы один из коэффициентов комбинации тоже отличен от нуля (пусть, например, $\lambda_1 \neq 0$). Тогда x_1 можно выразить через $x_2, \ldots, x_m$ и y_1. Следовательно, множество $A_1 = \{y_1, x_2, \ldots, x_m\}$ — полная система векторов в V и через нее можно выразить вектор y_2, и т. д. Повторив процесс m раз, мы получим полную систему векторов $A_m = \{y_1, y_2, \ldots, y_m\}$. (Для большей строгости рассуждение можно оформить в виде математической индукции, но мы не будем этим заниматься[1].)

Если $m < n$, то в множестве B найдется еще по крайней мере один y_{m+1}, который из-за полноты системы A_m должен представляться в виде линейной комбинации

$$y_{m+1} = \lambda_1 y_1 + \cdots + \lambda_m y_m,$$

что противоречит линейной независимости B. Значит $m \geqslant n$. ■

Пусть в конечномерном векторном пространстве V мы нашли два базиса — A из m векторов, и B из n векторов. По предыдущей теореме $m \geqslant n$, т. к. A — полная, а B — линейно независимая система векторов. С другой стороны, можно считать, что B — полная, а A — линейно независимая. Поэтому $m \leqslant n$. Но такое возможно только при $m = n$. Отсюда вытекает теорема.

Теорема А.58. Все базисы конечномерного пространства V состоят из одинакового числа элементов. Это число называется *размерностью* пространства V и обозначается символом $\dim V$.

Ясно, что $\dim K^n = n$. Это согласуется с интуитивным представлением: n-компонентные векторы живут в n-мерном мире. Заметим, что говоря о размерности, необходимо четко понимать, о каком множестве скаляров идет речь, т. е. над каким полем рассматривается векторное пространство. Для этого иногда используется обозначение $\dim_K V$. Важность скаляров для размерности следует, например из следующих равенств:

$$\dim_{\mathbb{C}} \mathbb{C} = 1, \qquad \dim_{\mathbb{R}} \mathbb{C} = 2, \qquad \dim_{\mathbb{Q}} \mathbb{C} = \infty.$$

Последние утверждения приложения оставим в качестве упражнений.

---

[1] Подумайте, почему на каждом шаге мы можем менять какой-то x на очередной y, т. е. почему, например, не может быть такого равенства: $y_3 = \lambda_1 y_1 + \lambda_2 y_2 + 0x_3 + \cdots + 0x_m$? — *Прим. перев.*

Теорема А.59. Пусть V — (ненулевое) конечномерное векторное пространство размерности n. Тогда

1. Любую линейно независимую систему векторов $A \subset V$ можно дополнить до базиса $B \supset A$.
2. В любой полной системе векторов $C \subset V$ можно выбрать базис $B \subset C$.
3. Число векторов в любой линейно независимой системе векторов из V не превосходит n.
4. Линейно независимая система, состоящая из n векторов, является базисом.
5. В любой полной системе векторов найдется не менее n векторов.
6. Полная система, состоящая из n векторов, является базисом.

Теорема А.60. Пусть W — подпространство конечномерного векторного пространства V. Тогда $\dim W \leqslant \dim V$, причем равенство размерностей говорит о том, что $W = V$.

Приложение Б

Примеры на языке *Java*

В этом приложении приведены примеры использования стандартных библиотек программ, встроенных в язык программирования *Java*, для реализации некоторых концепций этой книги. Стандартные классы библиотек в *Java* 1.2 и выше обеспечивают поддержку цифровых подписей и хэш-функций.

Чтобы получить поддержку шифрования, необходимо инсталлировать криптографические инструменты *Java* (Java Cryptographic Engine (or JCE)). Один из них, а именно *SUN JCE*, входит в стандартный пакет *Java* 1.4. Однако версия *SUN JCE* в Интернете представлена не полностью. Поэтому в наших примерах мы используем *Cryptix JCE*, доступный по адресу `www.cryptix.org`.
Этот пакет совместим с *SUN JCE* и поддерживает большое число криптографических алгоритмов, о которых мы даже не упоминали в этой книге. Кроме того, он легкодоступен через Интернет и распространяется свободно.

Мы не приводим программ для фактически реализованных алгоритмов — *RSA*, *DSA* и т. д. Мы просто показываем, как Вы можете использовать их, когда их реализует другой программист, следуя стандартам *API*. Вам, однако, стоит самостоятельно попытаться запрограммировать свои собственные версии *RSA* и т. д., поскольку это ведет к лучшему пониманию их специфических свойств.

Все примеры программ используют следующую функцию, которая преобразует массив байтов в строку их шестнадцатеричного представления:

```
static String To_Hex(byte[] bs)
   { StringBuffer str = new StringBuffer();
       for(int i=0; i < bs.length; i++)
         { String b = Integer.toHexString(bs[i]);
           if(b.length() < 2) b = "0" + b;
           str.append( b.substring(b .length()-2) );
         }
       return str.toString();
  }
```

Б.1. Блочные шифры

Сначала мы покажем, как пользоваться *DES*, утроенным *DES* (называемым *DESede* в *Java*) и *Rijndael* с помощью *Cryptix JCE*. Следующая программа показывает, как генерируются ключи и как можно применять различные режимы шифрования, в том числе *CBC* и *ECB*.

Прежде всего необходимо остановиться на каком-то конкретном типе шифрующего алгоритма, например, *DES*, режиме шифрования (*CBC*) и дополняющей системы. Дополняющая система, использующаяся в программе, приведенной ниже, называется *PKCS 5*. Она определена в пятых стандартах криптографии с открытым ключом, опубликованных лабораториями *RSA* (Public Key Cryptography Standards), откуда и берется аббревиатура.

Заметим, что изучив, как используется один блочный шифр, Вы будете знать, как применяются они все.

```
import Java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;

import cryptix.jce.provider.CryptixCrypto;

public class Symmetric_Demo
{
/* Заметьте, что это не зависит от алгоритма */
static void Encrypt(String Alg,String Mode,String Padding)
             throws NoSuchAlgorithmException,
                    InvalidKeyException,
                    IllegalBlockSizeException,
                    NoSuchPaddingException,
                    BadPaddingException,
                    InvalidAlgorithmParameterException,
                    NoSuchProviderException
{ System.out.println("\nDemo of "+Alg+" "+Mode+" "+Padding);
   KeyGenerator keygen = KeyGenerator.getInstance(Alg);
   if (Alg.equals("DES"))              { keygen.init(56); }
   else if (Alg.equalsC'Rijndael"))    { keygen.init(128); }
   else                                { keygen.init(168); }
   SecretKey sKey = keygen.generateKey();
```

```
    Cipher cipher;
    // Create the cipher
    cipher = Cipher.getInstance(Alg+"/"+Mode+"/"+Padding);

    // Initialize the cipher for encryption
    IvParameterSpec spec = null;
    if (Mode.equals("CBC"))
       { SecureRandom sr=new SecureRandom();
         byte[] iv=new byte[cipher.getBlockSize()];
         sr.nextBytes(iv);
         spec=new IvParameterSpec(iv);
         cipher.init(Cipher.ENCRYPT_MODE, sKey, spec);
       }
    else
       { cipher.init(Cipher.ENCRYPT_MODE, sKey); }

    // Our cleartext
    byte[] cleartext = "This is just an example".getBytes();

    System.out.println("PlainText : "+To_Hex(cleartext));

    // Encrypt the cleartext
    byte[] ciphertext = cipher.doFinal(cleartext);

    System.out.println("CipherText : "+To_Hex(ciphertext));
    // Initialize the same cipher for decryption
    if (Mode.equals("CBC"))
      { cipher.init(Cipher.DECRYPT_MODE, sKey, spec); }
    else
      { cipher.init(Cipher.DECRYPT_MODE, sKey); }
    // Decrypt the ciphertext
    byte[] cleartextl = cipher.doFinal(ciphertext);
    System.out.println("DecryptedText : "+To_Hex(cleartextl));
}
public static void main(String[] args)
             throws NoSuchAlgorithmException,
                    NoSuchProviderException,
                    InvalidAlgorithmParameterException,
                    InvalidKeyException,
                    InvalidKeyException,
                    IllegalBlockSizeException,
```

```
                    NoSuchPaddingException,
                    BadPaddingException
{
   Provider c_jce = new CryptixCrypto();
   int result = Security.addProvider(c_jce);
   if (result==-1)
     { System.out.println("Provider already exists.\n"); }
   else
     { System.out.println("Provider added.\n"); }
   // ECB Mode
   Encrypt("DES","ECB","PKCSSPadding");
   Encrypt("DESede","ECB","PKCS5Padding");
   Encrypt("Rijndael","ECB","PKCSSPadding");

   // CBC Mode
   Encrypt("DES","CBC","PKCSSPadding");
   EncryptC'DESede","CBC","PKCSSPadding");
   Encrypt("Rij ndael","CBC","PKCSSPadding");
}
}
```

Б.2. Шифрование с открытым ключом

Следующая программа демонстрирует, как можно с помощью *Cryptix JCE* осуществить шифрование в алгоритмах *RSA* и Эль-Гамаль. Обратите внимание на то, насколько она похожа на программу из предыдущего параграфа. Шифрование будет происходить в режиме *ECB*, а в качестве дополняющей системы выбрана *PKCS 1*.

```
import Java.security.*;
import javax.crypto.*;

import cryptix.jce.provi der.CryptixCrypto;

public class Assymetric_Demo
{
  /* Обратите внимание как это не зависит от алгоритма */
  static void Encrypt(String Alg,String Mode,String Padding)
              throws NoSuchAlgorithmException,
                     InvalidKeyException,
```

```
                    IllegalBlockSizeException,
                    NoSuchPaddingException,
                    BadPaddingException,
                    InvalidAlgorithmParameterException,
                    NoSuchProviderException
{ System.out.println("\nDemo of "+Alg+" "+Mode+" "+Padding);

    KeyPairGenerator Kgen = KeyPairCenerator.getlnstance(Alg);
  Kgen.initialize(512);

  KeyPair kpair = Kgen.generateKeyPair();

  Cipher cipher;

  // Create the cipher
  cipher = Cipher.getInstance(Alg+"/"+Mode+"/"+Padding);

  // Initialize the cipher for encryption
  cipher.init(Cipher.ENCRYPT_MODE, kpair.getPubli c());

  // Our cleartext
  byte[] cleartext = "This is just an example".getBytes();

  System.out.println("PlainText: "+To_Hex(cleartext));

  // Encrypt the cleartext
  byte[] ciphertext = cipher.doFinal(cleartext);

  System.out.println("CipherText: "+To_Hex(ciphertext));
   cipher.init(Cipher.DECRYPT_MODE, kpair.getPrivate());

  // Decrypt the ciphertext
  byte[] cleartextl = cipher.doFinal(ciphertext);

  System.out.println("DecryptedText: "+To_Hex(cleartext1));
}

  public static void main(String[] args)
            throws NoSuchAlgorithmException,
                   NoSuchProviderException,
```

```
                    InvalidAlgorithmParameterException,
                    InvalidKeyException,
                    InvalidKeyException,
                    IllegalBlockSizeException,
                    NoSuchPaddingException,
                    BadPaddingException
{
  Provider c_jce = new CryptixCrypto();
  int result = Security.addProvider(c_jce);
  if (result==-1)
   { System.out.println("Provider already exists.\n"); }
  else
  { System.out.print"ln("Provider added.\n"); }

  Encrypt("ElGamal","ECB","PKCS#1");
  Encrypt("RSA","ECB","PKCS#1");
}
}
```

Б.3. Хэш-функции

Здесь мы показываем, как можно применять хэш-функции, используя стандартную библиотеку *Java*, встроенную в *Java JDK*, т. е. не обращаясь к *JCE*. В программе объясняется, как можно вызывать *SHA-1* и *MD5*

```
import Java. security.*;
import java.io.*;

public class Hash_Demo
{
  static void Digest_Demo(String Alg)
            throws NoSuchAlgorithmException,
                   UnsupportedEncodingException
  {
  byte[] data;
  int i;
  data=new byte [100];
  System.out.println("\nDemo of "+Alg);
  MessageDigest MD=MessageDigest.getInstance(Alg);
```

```
  for (i=0; i<100; i++) { data[i]=(byte) i; }
  MD.update(data);
  byte[] hash=MD.digest();
  System.out.println(To_Hex(hash));
  for (i=0; i<100; i++) { data[i]=(byte) (i+10); }
  MD.update(data);
  hash=MD.digest();
  System.out.println(To_Hex(hash));

  System.out.flush();
  }

public static void main(String[] args)
             throws NoSuchAlgorithmException,
                    UnsupportedEncodingException
 {
   Digest_Demo("SHA");
   Digest_Demo("MD5");
  }
}
```

Б.4. Цифровые подписи

Следующая программа показывает, как можно подписать данные с помощью стандартной библиотеки *Java*. Она поддерживает как *DSA*, так и *RSA*, хотя последний основывается на наивном алгоритме хэш-функции. Чтобы реализовать *RSA*-алгоритм подписи со свойством доказуемой стойкости, такой, например, как *RSA-PSS*, необходимо инсталлировать *Cryptix JCE*.

В версию *DSA* встроены такие параметры домена, как p, q и G. Однако мы показываем, как можно самостоятельно использовать класс `BigInteger` в *Java*.

```
import java.security.*;
import java.security.spec.*;
import java.security.interfaces.*;
import java.math.BigInteger;
import java.io.*;
```

```
import cryptix.jce.provider.CryptixCrypto;

public class Signature_Demo
{
  static void Do_Signature(Signature Alg,
                    PrivateKey priv.PublicKey pub)
              throws InvalidKeyException,
                    SignatureException,
                    UnsupportedEncodingException
{
  /* Получаем сообщение */
  byte[] message=new byte[100];
  for (int i=0; i<100; i++) { message[i]=(byte) i; }
  /* Теперь делаем некоторое подписывание */
  Alg.initSign(priv);
  Alg.update(message);
  byte[] signature=Alg.sign();
  System.out.println("Signature = " + To_Hex(signature));

  /* Делаем проверку */
  Alg.initVerify(pub);
  Alg.update(message);
  boolean verifies = Alg.verify(signature);
  System.out.println("Signature Verifies : " + verifies);
  System.out.flush();
}
static void RSA_Sig (String Alg)
              throws NoSuchAlgorithmException,
                    NoSuchProviderException,
                    InvalidKeyException,
                    SignatureException,
                    UnsupportedEncodi ngException
{
  System.out.println("\n\n"+Alg+" Demo");
  SecureRandom random= new SecureRandom();
  KeyPairGenerator KGen=KeyPairGenerator.getInstance ("RSA");

  KeyPair pair = KGen.generateKeyPair();
  PrivateKey priv=pair.getPrivate();
  PublicKey pub=pair.getPublic();
  System.out.println("Private key is " +
```

```
    ((RSAPrivateKey) priv).getPrivateExponent().toString());

   System.out.println("Public key is " +
    ((RSAPublicKey) pub).getPublicExponent().toString());

   System.out.println("Modulus is " +
    ((RSAPublicKey) pub).getModulus().toString());
   /* Делаем подпись*/
   Signature rsa=Signature.get!nstance(Alg);

   Do_Signature(rsa,priv,pub);
}
/* flag=1 влечет использование встроенных параметров домена
       =0 влечет самостоятельное генерирование параметров */
static void DSA(int flag)
             throws NoSuchAlgorithmException,
                    NoSuchProviderException,
                    InvalidAlgorithmParameterException,
                    InvalidKeyException,
                    SignatureException,
                    UnsupportedEncodingExcepfion
 {
   System.out.println("\n\nDSA Demo");

   SecureRandom random= new SecureRandom();

   /* Генерируем параметры домена */
   KeyPairGenerator KGen=KeyPairGenerator.getInstance("DSA");
   if (flag==0)
    { /* Генерируем собственные параметры домена */
    BigInteger p,q,g,t,one=BigInteger.ONE;
    /* Находим q */
    q=new BigInteger(160,20,random);
    /* Ищем p */
    do
     { t=new BigInteger(1024-160,random);
       p=q.multiply(t); p=p.add(one);
     }
    while ( !p.isProbablePrime(20)
           || (p.bitLength()!=1024) );
    /* Подбираем G */
    do
```

```
        { G=new BigInteger(1024,random);
          G=G.modPow(t,p);
        }
      while ( G.equals(one));

      DSAParameterSpec dsaSpec
                          = new DSAParameterSpec(p,q,G);
      KGen.initialize(dsaSpec,random);
      }
     else
      { KGen.initialize(1024,random); }
     /* Генерируем ключевую пару */
     KeyPair pair = KGen.generateKeyPair();
     PrivateKey priv=pair.getPrivate();
     PublicKey pub=pair.getPublic();
     DSAParams Params=((DSAPublicKey) pub).getParams();
     System.out.pnintln(" p = " + Params.getP().toString());
     System.out.println(" q = " + Params.getQ()).toString());
     System.out.println(" G = " + Params.getG().toString());
     System.out.println("Private key is "
               + ((DSAPrivateKey) priv).getX().toString());
     System.out.println("Public key is "
               + ((DSAPublicKey) pub).getY().toString());
     /* Делаем подпись */
     Signature dsa=Signature.getInstance("SHA1withDSA");

     Do_Signature(dsa,priv,pub);
}

public static void main(String[] args)
             throws NoSuchAlgorithmException,
                    NoSuchProviderException,
                    InvalidAlgorithmParameterException,
                    InvalidKeyException,
                    SignatureException,
                    UnsupportedEncodingException,
                    InvalidKeyException
 {
   Provider c_jce = new CryptixCrypto();
   int result = Security.addProvider(c_jce);
   if (result==-1)
    { System.out.println("Provider already exists.\n"); }
```

```
    else
     { System.out.println("Provider added.\n"); }
    RSA_Sig("SHAlwithRSA");          // Uses only standard API
    RSA_Sig("RSASSA-PSS/SHA-1");     // Needs Cryptix
    DSA(1);                          // Uses only standard API
    DSA(0);                          // Uses only standard API
  }
}
```

Б.5. Доказательства с нулевым разглашением и обязательства

Закончим приложение примерами, которые непосредственно не используют криптографическую библиотеку *API* в языке *Java*. Здесь мы реализуем протоколы обязательств из главы 13, которые применялись в протоколе голосования. Кроме того, мы приведем программу доказательства того, что в данном обязательстве передана плюс или минус единица.

На этих примерах мы хотим продемонстрировать, как класс больших целых чисел, встроенный в *Java*, можно использовать для создания наиболее продвинутых схем этой книги, которые не реализованы в стандартных пакетах. Программа, приведенная ниже, не использует возможностей ни *SUN*, ни *Cryptix JCE*, поэтому может быть выполнена в рамках стандартного пакета *Java*.

Б.5.1. Parameters.java

Первый класс просто содержит параметры домена для нашей схемы обязательств, которые, по существу, совпадают с открытым ключом *DSA*.

```
import Java.security.*;
import java.security.interfaces.*;
import Java.math.BigInteger;
class Parameters
{
  private BigInteger p,q,g,h;
  public Parameters(SecureRandom random)
              throws NoSuchAlgorithmException
   {
```

```
    /* Получаем параметры домена, инсталлированные SUN */
    KeyPairGenerator KGen=KeyPairGenerator.getInstance("DSA");
    KGen.initialize(1024,random);
    KeyPair pair = KGen.generateKeyPair();
    DSAPublicKey pub=(DSAPublicKey) pair.getPublic();
    DSAParams Params=pub.getParams();

    p=Params.getP();
    q=Params.getQ();
    g=Params.getG();
    h=pub.getY();
   }

  public BigInteger getP()   { return p; }
  public BigInteger getQ()   { return q; }
  public BigInteger getG()   { return g; }
  public BigInteger getH()   { return h; }
}
```

Б.5.2. Private_Commitment.java

Второй класс содержит секретные величины, ассоциированные с обязательством

$$B = G^a H^c.$$

```
import Java.math.BigInteger;

class Private_Commitment
{
  private BigInteger a,c,B;
  private Parameters PK;

  public Private_Commitment(Parameters P)
     { PK=P; }

  public void assign(BigInteger a1,BigInteger b1,BigInteger c1)
     { a=a1; B=b1; c=c1; }

  public BigInteger getA()          { return a; }
  public BigInteger getB()          { return B; }
  public BigInteger getC()          { return c; }
  public Parameters get_Params()    { return PK; }
}
```

Б.5.3. Public_Commitment.java

Третий класс содержит только открытые величины обязательства.

```
import Java.math.BigInteger;
class Public_Commitment
{
  private BigInteger b;
  private Parameters PK;
  public Public_Commitment(Parameters P)
     { PK=P; }
  public void assign(BigInteger b1)    { b=b1; }
  public Parameters get_Params()       { return PK; }
  public BigInteger getB()             { return b; }
}
```

Б.5.4. Commitment_Factory.java

Этот четвертый класс создает открытую–секретную пару обязательства. Идея состоит в том, что пользователь использует этот класс для создания пары, открытую часть которой он публикует, а секретную хранит в тайне до определенного момента.

```
import Java.math.BigInteger;
import Java.security.*;
class Commitment_Factory
{
  private Parameters PK;
  private Public_Commitment Pub;
  private Private_Commitment Priv;
  public Commitment_Factory(Parameters P)
     { PK=P; }
  public void Commit(int cc,SecureRandom random)
              throws ArithmeticException
    {
       BigInteger q=PK.getQ();
       BigInteger p=PK.getP();
       BigInteger g=PK.getG();
       BigInteger h=PK.getH();
       if (cc!=1 && cc!=-1)
         { throw new ArithmeticException
```

```
            ("Commitment should be +/- 1");
          }
        BigInteger a,b,c,t1,t2;
        a=new BigInteger(q.bitLength(),random);
        a=a.mod(q);
        c=BigInteger.ONE;
        if (cc<0) { c=c.negate(); }
        t1=g.modPow(a,p);
        t2=h.modPow(c,p);
        b=t1.multiply(t2);
        b=b.mod(p);
        Pub=new Public_Commitment(PK);
        Pub.assign(b);
        Priv=new Private_Commitment(PK);
        Priv.assign(a,b,c);
      }
  public Public_Commitment getPublic() { return Pub; }
  public Private_Commitment getPrivate() { return Priv; }
}
```

Б.5.5. Proof.java

Теперь мы можем представить класс, который генерирует доказательство с секретной частью обязательства на входе и проверяет доказательство, если на вход подается открытая часть обязательства. Здесь используется хэш-функция *SHA-1*, встроенная в стандартный пакет *Java*.

```
import Java.math.BigInteger;
import Java.security.*;
class Proof
{
  private BigInteger d1,d2,r1,r2,alpha1,alpha2;
  public Proof() { ; }
  public void Assign_Proof( BigInteger D1,BigInteger D2,
                            BigInteger R1,BigInteger R2,
                            BigInteger A1,BigInteger A2)
   { d1=D1;     d2=D2;
     r1=R1;     r2=R2;
     alpha1=A1; alpha2=A2;
   }
```

```
public BigInteger get_d1()          { return d1; }
public BigInteger get_d2()          { return d2; }
public BigInteger get_r1()          { return r1; }
public BigInteger get_r2()          { return r2; }
public BigInteger get_alpha1()      { return alpa1; }
public BigInteger get_alpha2()      { return alpa2; }

public void Make_Proof (Private_Commitment priv,
                SecureRandom random
            throws NoSuchAlgorithmException
 {
  Parameters P=priv.get_Params();
  BigInteger p=P.getP();
  BigInteger q=P.getQ();
  BigInteger g=P.getC();
  BigInteger h=P.getH();
  // Form the Commitment
  BigInteger r,d,w;
  r=new BigInteger(q.bitLength(),random); r=r.mod(q);
  d=new BigInteger(q.bitLength(),random); d=d.mod(q);
  w=new BigInteger(q.bitLength(),random); w=w.mod(q);
  BigInteger t1,t2,t3;
  t1=g.modPow(r,p);
  t2=g.modPow(w,p);
  BigInteger a,b,c;
  a=priv.getA();
  b=priv.getB();
  c=priv.getC();
  if (c.equals(BigInteger.ONE))
     { t3=b.multiply(h);
       t3=t3.mod(p);
     }
  else
     { t3=h.modInverse(p);
       t3=t3.multiply(b);
       t3=t3.mod(p);
     }
  t3=t3.modPow(d,p);
  t3=t3.modInverse(p);
  t1=t1.multipiy(t3); t1=t1.mod(p);
  if (c.equals(BigInteger.ONE))
   { alpha1=t1; alpha2=t2; }
```

```
    else
     { alpha1=t2; alpha2=t1; }
    // Form the challenge
    MessageDigest MD=MessageDigest.getInstance("SHA");
    MD.update(b.toByteArray());
    MD.update(alpha1.toByteArray());
    MD.update(alpha2.toByteArray());
    byte[] hash=MD.digest();
    BigInteger challenge=new BigInteger(hash);
    challenge=challenge.mod(q);
    // Form the response
    if (c.equals(BigInteger.ONE))
     { d1=d;
       d2=challenge.subtract(d1);
       r1=r;
       r2=a.multiply(d2);
       r2=r2.add(w);
     }
    else
     { d2=d;
       d1=challenge.subtract(d2);
       r2=r;
       r1=a.multiply(d1);
       r1=r1.add(w);
     }
    d1=d1.mod(q); d2=d2.mod(q);
    r1=r1.mod(q); r2=r2.mod(q);
  }
public boolean Check_Proof(Public_Commitment pub)
            throws NoSuchAlgorithmException
 {
  Parameters P=pub.get_Params();
  BigInteger p=P.getP();
  BigInteger q=P.getQ();
  BigInteger g=P.getC();
  BigInteger h=P.getH();
  BigInteger b=pub.getB();
  BigInteger e1,e2;

  // Form the challenge
  MessageDigest MD=MessageDigest.getInstance("SHA");
  MD.update(b.toByteArray());
```

```
    MD.update(alphal.toByteArray());
    MD.update(alpha2.toByteArray());
    byte[] hash=MD.digest();
    BigInteger challenge=new BigInteger(hash);
    challenge=challenge.mod(q);

    e1=d1.add(d2); e1=e1.mod(q);
    if (!e1.equals(challenge)) { return false; }

    e1=g.modPow(r1,p);
    e2=e2.mod(p);      e2=b.multiply(h);
    e2=e2.modPow(dl,p);
    e2=e2.multiply(alpha1);      e2=e2.mod(p);
    if (!e1.equals(e2)) { return false; }

    e1=g.modPow(r2,p);
    e2=h.modInverse(p);
    e2=b.multiply(e2);      e2=e2.mod(p);
    e2=e2.modPow(d2,p);
    e2=e2.multiply(alpha2);      e2=e2.mod(p);
    if (!e1.equals(e2)) { return false; }

    return true;
   }
}
```

Б.5.6. prog.java

В заключение приведем тестовую программу, собирающую все предыдущие вместе.

```
import Java,security.*;
import Java.math.BigInteger;

public class prog
{
  public static void check(int cc,Commitment_Factory CF,
                           SecureRandom random)
            throws NoSuchAlgorithmException
   {
    System.out.println("\n\nChecking with cc = "+cc+"\n");

    // Make the commitment
    CF.Commit(cc,random);

    Public_Commitment pub=CF.getPublic() ;
    Private_Commitment priv=CF.getPrivate();
```

```
    { BigInteger a,b,c;
     b=pub.getB();
     a=priv.getA(); c=priv.getC();
     System.out.println("Private Commitment");
     System.out.println(" a = " + a +"\n c = " + c);
     System.out.println("\nPublic Commitment");
     System.out.println(" b = " + b);
    }
    /* Делаем доказательство */
    Proof Sender=new Proof ();
    Sender.Make_Proof(priv, random);
    BigInteger d1,d2,r1,r2,alpha1,alpha2;
    d1=Sender.get_d1(); d2=Sender.get_d2();
    r1=Sender.get_r1(); r2=Sender.get_r2();
    alpha1=Sender.get_alpha1(); alpha2=Sender.get_alpha2();
    System.out.println("\nProof");
    System.out.println(" d1 = "+d1);
    System.out.println(" d2 = "+d2);
    System.out.println(" r1 = "+r1);
    System.out.println(" r2 = "+r2);
    System.out.println(" alpha1 = "+alpha1);
    System.out.println(" alpha2 = "+alpha2);
    /* Проверка доказательства */
    System.out.println("\nChecking Proof");
    Proof Reciever=new Proof();
    Reciever.Assign_Proof(d1,d2,r1,r2,alpha1,alpha2);
    System.out.pintln(Reciever.Check_Proof(pub));
   }
public static void main(String[] args)
            throws NoSuchAlgorithmException,
                   NoSuchProviderException
   {
    SecureRandom random=new SecureRandom();
    Parameters domain=new Parameters(random);
    Commitment_Factory CF = new Commitment_Factory(domain);
    check(1,CF,random);
    check(-1,CF,random);
   }
}
```

Дополнение I
Зашифрованные поисковые системы

Владимир Насыпный, Галина Насыпная

Одно из направлений совершенствования современных информационных поисковых систем — это повышение их интеллектуализации с сохранением высокого уровня безопасности. Использование стохастической информационной технологии позволяет комплексно повысить «интеллект» поисковой системы без ослабления ее защищенности. В данной работе впервые показывается возможность проведения семантического анализа, определяемого требованиями к поисковой системе, зашифрованных текстовых документов с использованием стохастически защищенных баз знаний и программных средств.

Д.1.1. Введение

Современные поисковые системы становятся все более интеллектуальными, что, однако, не должно ослаблять безопасность доступа к информации. Использование стохастической информационной технологии [1–3] позволяет комплексно повысить «интеллект» поисковой системы без ослабления ее защищенности. Это стало возможно за счет случайного кодирования и хэширования символьной информации с целью ее адаптации к конкретной компьютерной среде. В работе [2] доказана возможность создания на основе новой технологии интеллектуальных систем точного поиска, реализующих функции извлечения знаний из текстов и формирования ответов, релевантных запросам пользователей. Вместе с тем, применение стохастической информационной технологии позволяет решить и другую задачу — обеспечение безопасного поиска зашифрованной текстовой информации различного уровня конфиденциальности. При этом создается замкнутый безопасный поисковый контур. Запрос, поступивший от пользователя, шифруется и передается в поисковую машину, где, не расшифровываясь, подвергается дополнительному шифрованию. Это обеспечивает реализацию процедуры интеллектуального поиска на зашифрованных текстовых документах, не раскрывая их содержания. Полученный ответ, релевантный запросу, также будет зашифрован, передан по линии связи и расшифрован на рабочем месте пользователя. Таким образом, исключается возможность доступа к информации, хранящейся в текстовых документах поисковой системы, а также доступ к содержанию вопросов и ответов, передаваемых по сети. Это открывает новые возможности в области создания безопасных поисковых систем, работающих с конфиденциальной информацией.

Д.1.2. Стохастическая технология и семантический анализ текста

Применение стохастической информационной технологии позволяет комплексно решать проблему реализации точного поиска и обеспечения безопасности ин-

формации. Под точным поиском понимается нахождение системой ответа, релевантного запросу пользователя. При этом запрос формулируется на естественном языке в виде вопросительного предложения. Точный поиск предполагает нахождение ответа с максимально возможной релевантностью — мерой, определяющей, насколько полно тот или иной документ отвечает критериям, которые указаны в запросе. Как показано в [2], точный поиск может быть получен в виде одного предложения текста (краткий ответ) или группы предложений (подробный ответ). При этом критерием релевантности является возможность эквивалентного преобразования с помощью интеллектуальной обработки полученного ответа к виду запроса. Если такое преобразование возможно, то полученный ответ считается в полной мере релевантным запросу или точным. В противном случае производится попытка повторного формирования ответа с использованием дополнительной текстовой информации. Если получение указанного ответа на предоставленном объеме текстовой информации невозможно, то считается, что в данном случае точный ответ не может быть получен.

Точный ответ либо непосредственно содержится в текстовой информации в виде одного или нескольких предложений, либо на основе имеющейся информации происходит извлечение знаний из документов и формируются новые предложения, релевантные запросу, которых в явном виде в тексте нет. Важнейшую роль в этом процессе играют семантический анализ текстовой информации и логическая обработка фрагментов текста с целью получения новых, семантически связанных текстовых структур, соответствующих требованиям точного ответа.

Основные принципы построения и функционирования системы точного поиска на основе стохастической информационной технологии подробно описаны в [2, 5], а в данном материале мы более детально опишем реализацию семантического анализа и логической обработки текстовой информации в зашифрованном виде с целью формирования точного ответа.

В общем случае зашифрованная система точного поиска включает базу зашифрованных текстовых документов и криптографически защищенные средства ее интеллектуальной обработки: стохастически индексированные базы знаний грамматического и семантического анализа, базы знаний, определяющие правила эквивалентного преобразования, подсистему логического вывода и библиотеку прикладных зашифрованных программ, непосредственно реализующие функции поиска и обработки стохастически преобразованной информации. Выполнение программ также осуществляется в зашифрованном виде, что в сочетании с зашифрованной обработкой информации создает комплексную защиту системы от хакеров, программных закладок и вирусов [3, 7].

При формировании базы текстовых документов поисковой системы производится стохастическое кодирование символьной информации. Стохастическое индексирование выполняется с использованием специальной хэш-функции. Эта хэш-функция обеспечит преобразование различных элементов текстовой информации в их хэш-значения, представленные в виде двоичной комбинации заданной длины, которые принимаются в качестве стохастических индексов. За счет свойств хэш-функции и выбора длины комбинации индекса достигается их гарантированная уникальность для различных элементов текста со сколь угодно малой заданной вероятностью коллизий [1]. При этом сначала формируются стохастические индексы отдельных слов (их основ), которые затем используются для получения индексов словосочетаний, входящих в предложения текста, и самих предложений. На основе стохастических индексов предложений получают

индексы абзацев. Названия глав, разделов и самих текстовых документов также преобразуют в соответствующие стохастические индексы.

Полученные индексы обеспечивают произвольный доступ к соответствующим элементам и структурам текстовой информации, которые при этом стохастически кодируются с использованием одноразовой системы шифрования с открытой передачей ключей [3]. Ключи, применяемые при шифровании текстов, записываются в конце каждого зашифрованного предложения. Для перевода слов или словосочетаний из одной системы шифрования в другую используются процессы перекодирования символьной информации без раскрытия ее содержания [7]. Для раскодирования текстовой информации имеются соответствующие декодеры. При этом система формирования и передачи одноразовых открытых ключей обеспечивает реализацию в реальном времени описанных функций кодирования, перекодирования и декодирования текстовой информации. Отметим, что после каждого обращения к соответствующему массиву зашифрованного текста происходит его перешифровка с использованием нового открытого ключа.

В предложенной системе стохастической индексации формирование индексов непосредственно на основе самих символьных объектов обеспечивает возможность ввода новых, исключения старых объектов, изменения порядка их следования, а также модификацию сетевых структур баз знаний в реальном масштабе времени. При этом происходит автоматическая модификация только тех структур, которые непосредственно связаны с вновь вводимыми или исключаемыми объектами без изменений всей индексной системы. В этом принципиальное отличие стохастического индексирования от регулярного индексирования текстовых документов, при котором любое изменение состава символьных объектов или их связей требует полной реструктуризации системы. Полученная стохастическая индексная система является открытой к изменению состава и содержания поисковой системы в процессе ее функционирования, что делает возможным применение широкой адаптации индексирования к процессам поиска для повышения скорости обработки при проведении семантического анализа текстов. Например, в ходе анализа часто возникает необходимость поиска соответствующих фрагментов текста не только по отдельным словам, но и по словосочетаниям, определяющим различные термины, понятия, предикативную основу, а также другие типы отношений в предложении. Для этого в системе реализована возможность быстрого перехода от индексов отдельных слов к индексам указанных словосочетаний. В результате обеспечивается произвольный доступ к текстовой информации с целью нахождения нужных предложений, а также выполнение функций логического вывода, классификации и рубрикации текстов. Индексные таблицы автоматически модифицируются для включения строк, связывающих индексы отмеченных словосочетаний с индексами соответствующих предложений абзацев и текстов. За счет этого повышается скорость реализации семантического анализа.

Д.1.3. Логический вывод на основе стохастической технологии

Все перечисленное относится также к построению и функционированию баз знаний, основанных на стохастически индексированных правилах продукций. Применение стохастических индексов предикатов, процедур и правил позволяет образовывать сетевые структуры, в которых время логического вывода линейно зависит от числа используемых правил продукций. При этом полностью

снимается проблема «комбинаторного взрыва», характерного для существующих продукционных систем, и обеспечивается реальное время логической обработки независимо от объема базы знаний. Образованная сетевая структура правил продукций является открытой к изменению их состава и содержания. Часто используемые цепочки правил могут быть преобразованы в одно правило путем их агрегации, что повышает скорость обработки информации текстов при семантическом анализе и поиске. Отметим, что построение правил продукций на основе стохастических индексов априори шифрует содержание правил и логику их обработки.

Рассмотрим пример стохастического преобразования правил продукций и выполнения логического вывода в зашифрованном виде.

Два правила из базы знаний семантического анализа, связанных с определением обстоятельства места (каждый предикат, входящий в первое или второе правило, имеет соответствующий номер):

1. Если в предложении имеется словосочетание, (Д.1.1)
   и это словосочетание включает глагол, (Д.1.2)
   и этот глагол относится к глаголам класса движения, (Д.1.3)
   и это словосочетание включает существительное, (Д.1.4)
   и это существительное относится к существительным класса пространства, (Д.1.5)
   то словосочетание, включающее глагол и существительное, относится к типу пространственных отношений. (Д.1.6)
2. Если словосочетание, включающее глагол и существительное, относится к типу пространственных отношений, (Д.1.7)
   то существительное данного словосочетания относится к обстоятельству места. (Д.1.8)

Шифрование указанных правил производится с использованием хэш-функций. В результате каждый предикат с номером (i, j) будет преобразован в уникальный стохастический индекс $I_{\xi ij}^{(p)}$ — двоичную комбинацию заданной длины. Каждое правило продукций с номером i будет преобразовано в стохастический индекс $I_{\xi i}^{(pp)}$. В силу свойств односторонности хэш-функции восстановить исходное содержание предиката и правила продукций по полученному стохастическому индексу невозможно. Для исключения подбора исходного текста по известной хэш-функции перед формированием стохастического индекса текстовая информация может быть предварительно зашифрована.

Раскрытие содержания указанных текстовых элементов осуществляется с использованием секретного словаря, который хранится в зашифрованном виде. Доступ к соответствующей строке секретного словаря производится по индексу $I_{\xi ij}^{(p)}$. Эта строка $I_{\xi ij}^{(p)}$ описывает содержание данного предиката в текстовом защищенном виде. После шифрования правила примут следующий вид:

$$I_{\xi 1}^{(pp)}: \; I_{\xi 11}^{(p)} \wedge I_{\xi 12}^{(p)} \wedge I_{\xi 13}^{(p)} \wedge I_{\xi 14}^{(p)} \wedge I_{\xi 15}^{(p)} \longrightarrow I_{\xi 16}^{(p)};$$
$$I_{\xi 2}^{(pp)}: \; I_{\xi 21}^{(p)} \longrightarrow I_{\xi 22}^{(p)}.$$

Анализируя правила продукций (1) и (2), представленные в текстовом виде, можно увидеть, что предикаты (Д.1.6) и (Д.1.7) идентичны по содержанию (без союза «если... то», который играет формальную роль). Поэтому оба правила при логическом выводе образуют связанную цепочку через отмеченный идентичный предикат. Учитывая свойства хэш-функции после шифрования данных правил,

индексы $I_{\xi 16}^{(p)}$ и $I_{\xi 21}^{(p)}$ также будут идентичными. В итоге указанные правила в зашифрованном виде образуют логическую цепочку:

$$I_{\xi 1}^{(pp)} \longrightarrow I_{\xi 2}^{(pp)} \longrightarrow I_{\xi 22}^{(p)}.$$

Здесь предикат $I_{\xi 22}^{(p)}$, являясь заключением логической цепочки, определяет тип обстоятельства в защищенном виде. Результат логического вывода может быть расшифрован с использованием функции обращения по индексу $I_{\xi 22}^{(p)}$ к секретному словарю предикатов. Так как данный словарь зашифрован с применением одноразовой системы шифрования, расшифровка указанного предиката производится в декодере с использованием соответствующего открытого ключа.

Д.1.4. Семантический анализ зашифрованных текстов

Как известно, цель семантического анализа — анализ смысла составных частей каждого предложения [4]. Для этого в описываемой интеллектуальной поисковой системе используется процесс извлечения знаний из лингвистической литературы. Применяются стохастически индексированные толковые и семантические словари, проблемно-ориентированные словари терминов и определений, энциклопедии, справочники, учебные пособия и др. За счет этого реализуется режим самообучения поисковой системы с использованием логического вывода в указанных текстах, с автоматическим накоплением знаний для проведения грамматического и семантического анализа [2]. Сформированные базы знаний содержат как процедурные знания в виде правил продукций, так и семантические сети, включающие термины и наименования объектов предметной области, предикативные основы предложений текста, а также словосочетания, описывающие типы отношений в каждом предложении.

Отметим, что уровень семантического анализа зависит от требований точного поиска. Исходя из этого, классификация объектов и отношений между ними, представленная в словосочетаниях каждого предложения текста, является определяющей. Процесс классификации осуществляется автоматически путем реализации логического вывода в стохастически индексированных толковых, семантических словарях, а также в словарях терминов и определений. Предварительно проведенное стохастическое индексирование указанных текстовых документов позволяет по индексу основы слова, представляющего некоторое понятие, по индексу словосочетаний, определяющих типы отношений данного предложения, получить произвольный доступ к соответствующим статьям словаря, где определены указанные элементы текста. Далее по индексам понятий и словосочетаний, которые входят в указанные определения, логический вывод может быть продолжен путем доступа к другим статьям текста, описывающим их. В итоге будут генерироваться дерево или цепочки логического вывода, реализуемые с помощью индексов указанного текстового документа, до тех пор, пока исходное понятие или словосочетание не будет сведено к базовым понятиям пространства, времени, причины, цели, образа действия, меры или степени и др. Глаголы, представляющие сказуемое, соответственно, будут отнесены к классам глаголов движения, перемещения, конкретного действия, физического или душевного состояния, мыслительно-речевого действия, а также других существующих классов глаголов. При этом на основе классификации понятий и терминов, описывающих объекты предметной области, а также типы отношений между ними, с использованием правил продукций представленной выше базы знаний точно определяются члены каждого предложения.

Разберем пример классификации глаголов и существительных некоторого исходного предложения, которое включает словосочетание, содержащее глагол и существительное.

Возьмем предложение «Мальчик бежит полем», как пример, иллюстрирующий применение правил продукций для определения типа обстоятельства, которое выражено существительным. Сначала проанализируем это предложение в незашифрованном виде. Выберем словосочетание «бежит полем», содержащее глагол «бежит» и существительное «полем». Переведем глагол «бежит» в неопределенную форму «бежать». Для доказательства того, что этот глагол входит в класс глаголов движения, сначала обратимся к толковому словарю (Ожегов С.И. Словарь русского языка. — М.: Сов. Энциклопедия, 1973. — 846 с.):

С.39. **Бежать** (...) 1. *Двигаться* быстрым, резко отталкивающимся от земли шагом.

Как видим, в соответствии с толковым словарем глагол «бежать» определяется глаголом «двигаться». Далее с помощью правила эквивалентного преобразования глагола в существительное преобразуем глагол «двигаться» в существительное «движение». Правило эквивалентного преобразования: если требуется преобразовать глагол в неодушевленное существительное, то сначала выделяем основу глагола, обращаемся к формату словаря, ищем неодушевленное существительное, основа которого имеет общую часть, включающую корень (возможно, с приставкой или с суффиксом, с введением новых согласных, с их чередованием), с основой преобразуемого глагола, после этого, используя общую часть их основ, заменяем суффикс *-а(ть), -и(ть)* преобразуемого глагола на суффикс неодушевленного существительного *-ени(е): двигать(ся) — движение.*

Преобразовав глагол «двигаться» в существительное «движение», которое совпадает с названием класса глаголов, можно сделать вывод, что глагол «бежать» относится к классу движения.

Обратимся теперь к существительному «полем». Переведем данное существительное в именительный падеж. Для доказательства того, что существительное «поле» относится к существительным класса пространства опять обратимся к толковому словарю:

С.507. **Поле** (...) 1. Безлесная равнина, *пространство.*

Из определения следует, что существительное «поле» относится к существительным класса пространства. Проведенный анализ доказывает, что в соответствии с правилом продукций словосочетание «бежать полем»относится к типу пространственных отношений. Поэтому, как следует из цепочки логического вывода, существительное «полем» относится к обстоятельству места.

Теперь представим, как описанный процесс семантического анализа можно произвести с зашифрованным текстом, используя стохастически индексированный зашифрованный толковый словарь. Обозначим как $K(U11)$ закодированное слово «бежит», выбранное из указанного зашифрованного предложения. Используя отмеченную процедуру преобразования глаголов, которая представлена в виде зашифрованной подпрограммы, и применяя перекодер, преобразуем исходный код $K(U11)$ в код $K(U12)$. Этот код соответствует глаголу «бежать». По полученному коду $K(U12)$ в блоке формирования хэш-функции будет сформирован стохастический индекс этого слова $I^{(u)}_{\xi 12}$. По данному индексу выполняется произвольный доступ к указанной статье стохастически индексированного

толкового словаря. Выберем из него код $K(U13)$, который здесь соответствует глаголу «двигаться». Затем, используя зашифрованную подпрограмму перевода данного кода в код соответствующего ему существительного, получим код $K(Ud)$. Этот код в данном случае соответствует зашифрованному коду класса глаголов движения. В результате получим следующую цепочку логического вывода:

$$K(U11) \longrightarrow K(U12) \longrightarrow I^{(u)}_{\xi 12} \longrightarrow K(U13) \longrightarrow K(Ud).$$

Для классификации существительного «полем», которое в закодированном виде обозначим $K(U21)$, будет реализована другая логическая цепочка:

$$K(U21) \longrightarrow K(U22) \longrightarrow I^{(u)}_{\xi 22} \longrightarrow K(Up).$$

Здесь $K(U22)$ — код слова «поле», $I^{(u)}_{\xi 22}$ — стохастический индекс, образованный из указанного кода (он используется для доступа к соответствующей статье стохастически индексированного толкового словаря). Зашифрованное значение $K(Up)$, найденное в статье словаря, соответствует коду класса пространства. После определения классов глагола и существительного в закодированном виде с использованием приведенной цепочки правил продукций будет определен код, который соответствует обстоятельству места.

Таким образом, в зашифрованном виде может быть проведена классификация понятий, отношений и определения типов обстоятельств и других членов предложения. Здесь используются коды слов, их стохастические индексы, зашифрованные правила продукций и подпрограммы. При этом реализуются зашифрованные цепочки логического вывода на множестве правил продукций и кодов стохастически индексированного словаря.

Семантический анализ является необходимым дополнением синтаксического, в результате которого члены предложения определяются с недостаточной точностью, что не отвечает требованиям точного поиска. Главным является то, что на основе проведенного семантического анализа определяются типы отношений каждого словосочетания (в том числе наиболее сложные, включающие обстоятельства) в предложении и вопросы, на которые они отвечают.

Все понятия, выраженные словосочетаниями предложения, будут семантически представлять конкретные типы отношений, а именно: родо-видовые, «часть – целое», причинно-следственные, определительные, функциональные, пространственные, временные, образа действия, меры или степени и др. Этим типам отношений соответствуют конкретные вопросы. Подобному анализу подвергается также вопросительное предложение запроса, поэтому, если в ходе поиска будет найдено или сформировано предложение, содержащее словосочетание, соответствующее вопросительному слову (словосочетанию) запроса (при условии, что все остальные словосочетания запроса и данного предложения идентичны), то это означает, что в процессе поиска получен точный ответ на запрос.

Важным методом определения типа отношений в словосочетании (предикативная основа, включающая подлежащее и сказуемое, словосочетание, связывающее сказуемое с одним из видов обстоятельств и др.) является образование нового словосочетания с ключевыми словами, которые позволяют однозначно определить тип исходного словосочетания. Далее следует проверка корректности употребления данного сочетания путем обращения к базе текстов для поиска предложений, содержащих идентичные словосочетания. Если в процессе поиска по указанным ключевым словам будет найдено одно или несколько предложений,

включающих сформированное словосочетание, то это позволяет точно определить предикативную основу предложения или вид обстоятельства в исходном словосочетании. Таким образом, возможность обработки в реальном времени большого количества индексированных текстов позволяет получить новое качество в виде определения семантики (типа отношения) исходного словосочетания.

Полученные классы предикатов каждого предложения заносятся в концептуальную часть базы знаний текстовых документов по конкретной тематике с указанием индексов текстов, абзацев и предложений, в которых данные классы содержатся. Эти данные могут быть представлены как в виде фреймовой структуры, так и в табличном виде.

Другим важным процессом, который реализуется в ходе семантического анализа, является рубрикация текстов с точностью до каждого абзаца. Она осуществляется в процессе автоматического анализа терминов, понятий, определений, а также отношений между ними. На основе этого формируется рубрикатор текстовых документов с указанием основных терминов и понятий, представленных в названных документах по данной теме с отметкой индексов текстов и абзацев, содержание которых посвящено описанию отмеченных объектов предметной области. Кроме указанных классификатора и рубрикатора в ходе грамматического и семантического анализа текстов формируются пословные стохастические индексы [2], позволяющие по определенной совокупности ключевых слов запроса определять индексы текстов, абзацев и предложений, в которых они содержатся. Таблицы индексов включают все необходимые грамматические и семантические характеристики каждого предложения, необходимые для подробного анализа текста. Отметим, что эти индексы могут формироваться уже в процессе поиска ответа на введенный пользователем вопрос, в реальном времени после выбора с помощью классификатора и рубрикатора текстов, абзацев и предложений, которые семантически соответствуют поставленному вопросу.

Таким образом, в интеллектуальной информационно-поисковой системе поиск проводится в трехмерном пространстве: классификатор, рубрикатор по определенной теме, индексы текстового документа. При реализации системы в программно-аппаратном виде все три типа поиска могут выполняться параллельно в различных процессорах, что обеспечивает существенное (не менее, чем в три раза) сокращение времени предварительной обработки текстов.

После нахождения абзацев и предложений, соответствующих семантике запроса, на основе указанных элементов текста формируется точный ответ. Здесь используются процедуры образования семантически связанных структур, эквивалентных преобразований и логического вывода. В результате может быть получен краткий точный ответ, релевантный запросу.

Опыт разработки и использования поисковых систем показал, что существующие словари синонимов не отвечают требованиям точного поиска. Это обусловлено тем, что представленные синонимические ряды в названных словарях оторваны от содержания текстовых документов, поэтому они не могут включать близкие по смыслу слова, которые необходимы при нахождении точного ответа. Кроме этого синонимы должны быть согласованы со множеством слов, понятий, терминов словаря по данной тематике, который формируется при индексировании текстов. Указанную проблему решает метод контекстной синонимии на основе индексируемых текстовых документов [2], позволяющий повысить эффективность интеллектуальной поисковой системы. Для этого применяется логический вывод по индексированным толковым словарям, словарям терминов и определений по конкретным темам с использованием вместо индексов отдель-

ных слов индексов словосочетаний, в которые они входят. Здесь учитываются классификация и рубрикация каждого обрабатываемого текста.

Д.1.5. Универсальность защищенных поисковых систем

В результате применения стохастической информационной технологии интеллектуальная поисковая система может одинаково эффективно работать как с открытыми, так и с зашифрованными текстами. Для реализации точного поиска в зашифрованных текстах документы, подлежащие индексированию, предварительно шифруются с использованием одноразовой системы шифрования [3]. С этой целью применяются стохастический кодер и система формирования открытых и закрытых ключей. Шифрование производится методом стохастического кодирования. Зашифрованный текст переводится в специальный формат, позволяющий выделять отдельные абзацы, предложения, слова и знаки препинания. При этом каждое предложение шифруется с помощью одноразового ключа, который в открытом виде записывается в конце предложения. На основании полученного зашифрованного текста производится формирование описанной системы индексов отдельных слов, словосочетаний, предложений, абзацев и текстов.

Процесс формирования стохастического индекса заключается в перекодировании индексируемого элемента текста с помощью перекодера в соответствующий код. С выхода перекодера полученный зашифрованный текст посимвольно поступает в блок формирования хэш-функции, который в результате проведенной в нем обработки преобразует данный текстовый элемент в уникальный стохастический индекс — двоичную комбинацию заданной длины. В итоге все функции интеллектуальной обработки текстовой информации можно реализовывать с помощью стандартного набора процедур: перекодирование соответствующих элементов текста, формирование на их основе стохастического индекса, по которым осуществляется произвольный доступ к требуемым предложениям зашифрованного текстового документа (в том числе индексированно зашифрованных толковых словарей, словарей терминов и определений и другой стохастически индексированной и зашифрованной текстовой информации).

После доступа к заданному фрагменту текста поиск в нем необходимой информации осуществляется путем сравнения соответствующих элементов данного зашифрованного текста с исходным или эталонным текстом, после его перекодирования — с помощью открытых ключей к виду обрабатываемого зашифрованного текста [3]. В результате, используя типовой формат зашифрованного текста, могут быть найдены идентичные слова или словосочетания, необходимые для реализации интеллектуальной обработки текстов без раскрытия их содержания. Затем на основе найденных слов или словосочетаний формируются новые индексы для продолжения логической цепочки поиска. При этом могут быть реализованы все функции стохастической индексации текстов, классификация и рубрикация терминов, понятий и отношений, логического вывода на текстовой информации, эквивалентные преобразования слов, словосочетаний и предложений для формирования точного ответа, представленного в зашифрованном виде.

Декодирование ответа осуществляется в компьютере пользователя, выдавшего исходный запрос. При этом передача по линии связи также осуществляется в зашифрованном виде после необходимого перекодирования зашифрованного вопроса или ответа.

Приведем пример формирования запроса, обработки текста и получения ответа в зашифрованном виде.

Сначала представим запрос, сформированный пользователем, предварительно выбранный абзац в процессе анализа текста и полученный точный ответ в открытом виде. Именно в таком виде может получить указанную информацию администратор защищенной поисковой системы после ее расшифрования с помощью мастер-ключа.

Запрос: Какие устройства персонального компьютера называются периферийными?

Предварительно выбранный абзац: Персональный компьютер предназначен для создания, хранения, обработки и передачи данных. Он состоит из различных блоков и устройств. При этом устройства, расположенные внутри системного блока, называются внутренними. Устройства, расположенные снаружи, — внешними. Дополнительные подключаемые внешние устройства относятся к периферийным устройствам. Принтер для печати информации на бумаге — пример периферийного устройства.

Точный ответ: Дополнительные подключаемые внешние устройства персонального компьютера (например, принтер для печати информации) называются периферийными.

В процессе формирования точного ответа в качестве базового было выбрано следующее предложение: «Дополнительные подключаемые внешние устройства относятся к периферийным устройствам». Затем, используя отношения «часть–целое», в него было введено словосочетание «персональный компьютер» из первого предложения абзаца в соответствующем падеже (внешние устройства — часть компьютера). После этого, применяя отношения «род–вид», в базовое предложение включено словосочетание «принтер для печати информации» из последнего предложения абзаца (принтер для печати информации относится к классу периферийных устройств). К этому словосочетанию было добавлено вводное слово «например». Полученная группа слов «(например, принтер для печати информации)» представлена в базовом предложении как вставная конструкция и, соответственно, выделена скобками. Словосочетание «относятся к периферийным устройствам» заменяется на близкое по смыслу словосочетание «называются периферийными устройствами». В итоге выполненного семантического анализа и логической обработки текста формируется точный ответ, представленный выше.

Для контроля релевантности полученного ответа преобразуем его к виду запроса. При этом группа слов определения «Дополнительные подключаемые внешние» заменена на вопросительное слово «Какие». Также была исключена вставная конструкция, которая имеет уточняющее значение. В результате из сформированного ответа получено вопросительное предложение «Какие устройства персонального компьютера называются периферийными?», которое идентично запросу. Это доказывает релевантность полученного точного ответа запросу пользователя.

Как было отмечено выше, в поисковой системе описанные преобразования выполняются с зашифрованным текстом. Для шифрования используется одноразовый многоалфавитный кодер. Представим зашифрованную информацию в символьном виде (с некоторыми сокращениями ввиду иллюстративного значения данного примера).

Запрос: ЖЛПUЪЗЬNLQTПWKHEMRAФЮFOЗРTGDSAЩГBWFZИБОДСESV НВРЙIYKУИГДFDZKРЮФДTVQS

Предварительно выбранный абзац: WFRCKTЩГOQNЩИБSDVИВЙЬVGYЯХМ ЪYDЖЕГГОЯЧМЦZЕКДВЩЮZHYЖТГБIQFBХШЧHWSHЪЖИДБVZRKH ВЮМХТСФЯУKWQЪСЮАЦWLUЩСТKQMSHФЗЬЦLR

.....

EWЖФНЮАЩDZXОРЙЧZYВНЯБАTNWMЪФГЙFQJКНЮШOGCЖЭВЫV
PSЪДЬАКР

Точный ответ: КРАЦАIFДБМТНJМВЛГЙСКОЕНЩФDBPLУРЗЪUWIЖЫ
ШТRGЯЬИЭZSЮХОСNQVЧ

.....

АЕЈЮЦЕКНУХОЗБШSMOGMТЯ

Именно в таком виде информация будет представлена злоумышленнику при попытке несанкционированного доступа к системе в процессе обработки, хранения или передачи информации.

Отметим, что пользователь имеет доступ к содержанию запроса (до его кодирования с целью передачи в поисковую систему), а также полученного точного ответа (после его декодирования). Вся текстовая база системы, включая приведенный выше предварительно выбранный абзац, является для пользователя зашифрованной.

Таким образом, в системе реализуется полностью замкнутый зашифрованный контур точного поиска информации с выполнением функций интеллектуальной обработки текстов, включающих необходимые элементы семантического анализа.

Литература

[1] Насыпный В.В. Развитие теории построения открытых систем на основе информационной технологии искусственного интеллекта. М.: Воениздат, 1994. — 248 с.

[2] Насыпный В.В., Насыпная Г.А. Способ синтеза самообучающейся системы извлечения знаний из текстовых документов для поисковых систем. Международная заявка на изобретение № PCT/RU 02/00258 от 28.05.2002

[3] Насыпный В.В. Способ комплексной защиты распределенной обработки информации в компьютерных системах и система для осуществления способа. Международная заявка на изобретение № PCT/RU 01/00272 от 05.07.2001

[4] Лайонз Дж. Лингвистическая семантика: Введение. — М.: Языки славянской культуры, 2003. — 400с.

[5] Насыпный В.В., Насыпная Г.А. Поисковая машина для карманных компьютеров // Мир ПК, 2003, № 6, с. 77.

[6] Введение в криптографию / Под общ. Ред. В.В.Ященко. — М:МЦНМО: «ЧеРо», 1999. — 272с.

[7] Насыпный В.В. Защищенные стохастические системы // Открытые системы, 2004, № 8, с. 60-61.

Дополнение 2
Сетевая система с абсолютной стойкостью

Владимир Насыпный

Представлен разработанный автором новый метод построения сетевых одноразовых систем, обеспечивающий достижение абсолютной надежности шифра при обмене информацией между абонентами сети через центр сертификации, формирования ключей и перекодирования. Показано, что основу полученного метода составляет способ перекодирования информации без раскрытия ее содержания, который реализует засекреченную связь между любыми абонентами сети в одноразовом режиме шифрования с использованием случайных асимметричных ключей.

Д.2.1. Процесс формирования и использования сетевых одноразовых ключей

Развитие средств компьютерной техники (прежде всего рост производительности процессоров и существенное увеличение объема памяти) дает возможность реализовать новые методы построения криптографических систем, обеспечивающих значительное повышение стойкости. В данной статье представлена одноразовая система, позволяющая при обмене информацией в сети между абонентами гарантировать абсолютную надежность шифра, которой ранее обладала только традиционная система Г. Вернама.

Созданная система является совершенно секретной (по К. Шеннону) и базируется на идеях способа построения одноразовой системы перекодирования информации, представленной в [1, 2].

В состав данной системы входят центр сертификации, формирования ключей и перекодирования (ЦСФКП), серверы распределенной обработки и пользовательские устройства. Серверы распределенной обработки и пользовательские устройства далее будем именовать абонентскими комплектами (АК).

К основным задачам ЦСФКП относятся подключение пользовательских устройств и серверов к системе защиты, их сертификация, формирование и распределение ключей между абонентскими комплектами. Важнейшей функцией ЦСФКП является организация засекреченной связи между любыми абонентами сети в одноразовом режиме с применением перекодеров. Указанные перекодеры обеспечивают перевод засекреченной информации, зашифрованной одноразовым ключом абонента A, в засекреченную информацию с использованием ключа абонента B без раскрытия ее содержания.

В соответствии с новым методом построения сетевой одноразовой системы в ЦСФКП образуется заданное множество N одноразовых сетевых ключевых карт. Каждая из таких карт содержит множество M одноразовых ключей для каждого из числа M абонентов сети. Эти ключи являются случайными и независимыми. Они формируются на основе датчика случайных чисел. Каждый из них представляет собой случайно заполненную кодами длиной m таблицу размером

$n \times n$. Полученные таблицы являются базовыми асимметричными одноразовыми ключами (базовыми таблицами) для каждого из числа M абонентов сети.

Далее описанным порядком происходит формирование заданного множества N одноразовых карт. При этом в каждой карте содержится множество M различных случайных базовых таблиц абонентов. Подчеркнем, что все полученные карты независимы.

Одновременно на основе каждой сетевой карты для абонентов сети формируется множество M одноразовых сетевых блокнотов. Каждый блокнот i ($i = 1 - M$) включает базовую таблицу. Данная таблица соответствует базовому одноразовому ключу для абонента с номером i, который содержится в сетевой карте.

Таким образом, для каждой карты получают M различных одноразовых сетевых блокнотов (по числу абонентов сети). Вместе с этим, для каждого абонента формируются ряды одноразовых сетевых блокнотов, каждый из которых входит в одну из числа N сетевых карт. Указанные блокноты нумеруются в порядке от 1 до N. При этом каждый номер блокнота j равен номеру сетевой карты j ($j = 1 - N$), из которой он получен. Затем каждый номер хэшируется с помощью специальной хэш-функции. Каждому произвольному блокноту с номером i присваивается уникальный стохастический индекс $I_{\xi i}^{(k)}$, который взаимооднозначно определяет данный блокнот.

В итоге будет получен ряд одноразовых сетевых блокнотов для каждого из M абонентов, содержащий N одноразовых ключей. При необходимости этот ряд ключей засекречивается с помощью шифратора, который имеет высокую стойкость защиты. Таким шифратором может быть стохастический кодер, описанный в [2]. В результате образуется $M \times N$ различных одноразовых независимых сетевых блокнотов.

Полученные ряды из N одноразовых сетевых блокнотов записываются в M комплектов флэш-памяти, соответственно, для каждого из числа M абонентов сети. Флэш-память с множеством одноразовых (зашифрованных) сетевых блокнотов выдается очередному абоненту сети при его сертификации в ЦСФКП вместе со смарт-картой, содержащей пароль и pin-код данного пользователя. Полученные флэш-память и смарт-карта вводятся в АК каждого пользователя для организации сетевой засекреченной связи.

Отметим важную роль перекодирования, реализуемого в ЦСФКП. Именно с использованием этой функции обеспечивается засекреченная связь данного абонента с любым из числа M_i заданных других абонентов, хотя их базовые ключевые таблицы случайны, независимы и, следовательно, асимметричны. Указанная функция реализуется в ЦСФКП при обращении к нему отмеченной пары абонентов для организации зашифрованной связи. После расшифрования (например, с помощью стохастического декодера) перевод базовых таблиц абонентов A и B в симметричное состояние не требуется. Рассмотрим этот процесс подробнее.

При организации засекреченной связи между двумя абонентами по сетевой карте с номером i после выполнения процедур идентификации и аутентификации указанных абонентов ЦСФКП передает им хэш-функции соответствующих таблиц, которые необходимо использовать для этой связи. Если заявлен продолжительный сеанс, то ЦСФКП передает им столько значений хэш-функций ключей различных карт, сколько потребуется для засекречивания в одноразовом режиме всего сеанса связи. Укажем, что в общем случае для связи двух абонентов могут быть использованы таблицы (ранее не применявшиеся) разных сетевых карт. Это достигается за счет свойств перекодера, функции которого будут описаны ниже.

Затем по специальной команде ЦСФКП происходит поочередное расшифрование данных таблиц в АК указанных двух абонентов. После применения выделенных ЦСФКП базовых таблиц одноразовых блокнотов они стираются.

За счет названных мер обеспечивается случайность и непредсказуемость набора ключей, записанных в базовой таблице абонентов, а также необходимая секретность одноразовых ключей, которая заложена в методе Г. Вернама [3]. Это предотвращает попытки расшифрования информации противником как в процессе, так и после ее передачи. В итоге, как будет показано ниже, на базе сетевой одноразовой системы с перекодированием реализуется абсолютно надежный шифр.

Д.2.2. Реализация одноразового режима шифрования в системе с применением перекодера ЦСФКП

Рассмотрим процесс передачи информации в сети по линии связи, соединяющей пользователей А и В, которая коммутируется в ЦСФКП [2]. Для этого введем в традиционную схему Г. Вернама необходимые дополнительные элементы. На стороне А и В в схему засекреченной связи (рис. Д.2.1) входят:

- блок управления элементами комплекта, взаимодействующий с ЦСФКП, который также включает датчик случайных чисел;
- флэш-память, содержащая в открытом или зашифрованном виде N одноразовых блокнотов с базовыми таблицами;
- дешифратор для рассекречивания соответствующих одноразовых блокнотов;
- буфер для хранения очередной базовой таблицы ключа;
- блок для хранения базовой таблицы действующего одноразового ключа с регистрами перестановок, которые заполняются от датчика случайных чисел;
- блоки шифрования (расшифрования) для шифрования (расшифрования) исходного текста, передаваемого по сети связи;
- перекодер, входящий в состав ЦСФКП, который участвует в обеспечении данной засекреченной связи.

Перед началом сеанса связи с помощью ЦСФКП выполняются процессы идентификации и аутентификации абонентов. Для этого могут быть использованы алгоритмы, описанные в [2]. Затем происходит выборка из флэш-памяти комплектов А и В зашифрованных одноразовых блокнотов, которые указаны в значениях хэш-функций $I_{\xi i}^{(k)}$, $I_{\xi j}^{(k)}$. Эти значения передаются из ЦСФКП, соответственно, абонентам A, B и записываются в блок управления каждого абонента. Если заявлен продолжительный сеанс, то ЦСФКП передает столько значений хэш-функций неиспользованных одноразовых ключей различных сетевых карт, сколько потребуется для шифрования в одноразовом режиме всего сеанса связи между указанными абонентами.

По команде блоков управления в дешифраторах абонентов происходит расшифрование сетевых одноразовых блокнотов, соответствующих первой из переданных значений хэш-функций $I_{\xi i}^{(k)}$, запись базовых таблиц через буфер в блок таблицы одноразового ключа комплектов.

Для таблицы кодов ASCII $m = 8$ (бит), $n = 256$ (байт), поэтому для шифрования текста можно использовать таблицу одноразовых ключей размером 256×256 (байт). Если во флэш-памяти комплектов одноразовые блокноты не зашифрованы, то с помощью регистров происходит перестановка таблиц и строк базовой таблицы. Это делается с целью обеспечения ее дополнительной защиты. Более подробно данный процесс описан в [2].

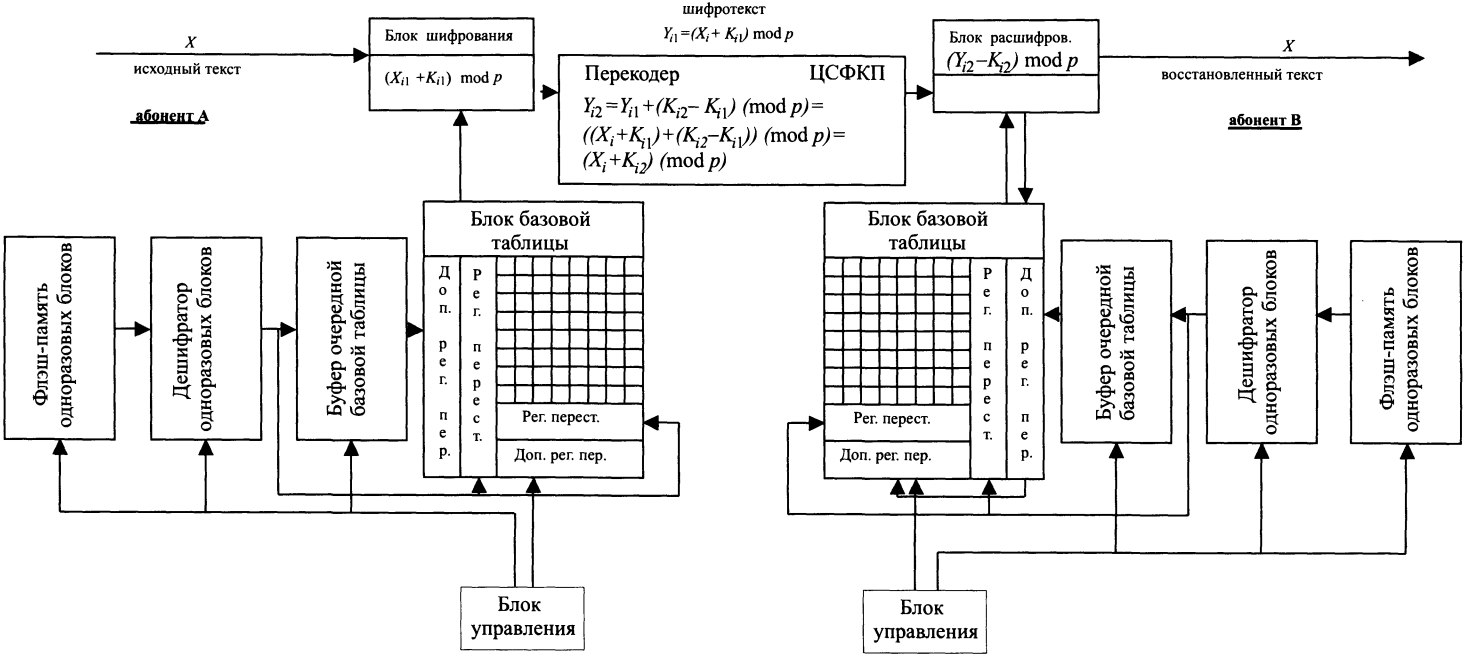


Рис. Д.2.1. Схема одноразового режима передачи с перекодером

В ходе шифрования информации из базовой таблицы абонента A будут поочередно считываться ключевые последовательности вида

$$K_i = K_{i,0}, K_{i,1}, \ldots, K_{i,n-1} \ (i = 1, n). \qquad (Д.2.9)$$

При этом каждый символ $K_{i,j}$ имеет длину m (бит).

Как известно (см. [3]), одноразовая система шифрует исходный открытый текст

$$X = (X_0, X_1, \ldots, X_{n-1}) \qquad (Д.2.10)$$

шифротекст

$$Y_1 = (Y_0, Y_1, \ldots, Y_{n-1}) \qquad (Д.2.11)$$

посредством подстановки Цезаря с помощью базовой таблицы абонента A

$$Y_{i,1} = (X_i + K_{i1}) \ (\mathrm{mod} p), \ 0 < i < n, \qquad (Д.2.12)$$

где K_{i1} — i-й элемент случайной ключевой последовательности базовой таблицы абонента A, p — число символов в алфавите.

Отметим, что ключевое пространство К одноразовой системы представляет собой набор дискретных случайных величин и содержит $V = p^n$ значений.

Полученный с помощью базовой таблицы A шифротекст Y_1 поступает в ЦСФКП, где для него уже подготовлен соответствующий перекодер. Подчеркнем, что в системе реализуется функция обеспечения целостности (имитозащиты) передаваемой информации, разработанная в [2]. Перекодер включает базовую таблицу абонента A, базовую таблицу абонента B, а также схему перекодирования (рис. Д.2.1). Процесс перекодирования производится с использованием шифротекста Y и объединенной ключевой последовательности, формируемой на основе базовых таблиц абонентов A и B. При этом процесс перекодирования описывается следующим выражением:

$$\begin{aligned} Y_{i,2} &= (Y_{i,1} + (K_{i,2} - K_{i,1})) \ (\mathrm{mod} p) = \\ &= ((X_i + K_{i,1}) + (K_{i,2} - K_{i,1})) \ (\mathrm{mod} p) = \\ &= (X_i + K_{i,2}) \ (\mathrm{mod} p), \end{aligned} \qquad (Д.2.13)$$

где $Y_{i,1}$ и $K_{i,1}$ — символы шифротекста и ключа, полученные с помощью таблицы A, $Y_{i,2}$ и $K_{i,2}$ — символы шифротекста и ключа после перекодирования в соответствии с таблицей B.

Подчеркнем, что, как следует из (Д.2.13), в процессе перекодирования символы шифротекста не расшифровываются.

После передачи перекодированной информации из ЦСФКП в комплект B процедура расшифрования, выполняемая в данном комплекте, описывается соотношением

$$X_i = (Y_{i,2} - K_{i,2}) \ (\mathrm{mod} p), \qquad (Д.2.14)$$

где K_{i2} — i-й элемент случайной ключевой последовательности таблицы B, соответствующей $Y_{i,2}$.

Для описанной выше структуры апостериорная вероятность входного сообщения при известном выходном сообщении будет равна априорной как при передаче шифротекста от абонента A к перекодеру, так и от перекодера к абоненту B.

В результате обеспечивается совершенная секретность системы согласно теореме К. Шеннона, которая изложена в его статье «Теория связи в секретных системах», приведенной в [3].

В процессе доведения зашифрованного текста на основании поступивших из ЦСФКП индексов одноразовых блокнотов по командам блока управления представленным выше порядком выбираются одноразовые блокноты с индексами $I^{(k)}_{\xi i+1}$, $I^{(k)}_{\xi j+1}$ следующей сетевой одноразовой карты. Они расшифровываются, необходимые данные записываются в буферы комплектов A и B для подготовки замены базовых таблиц. Одновременно соответствующие базовые таблицы, хранящиеся в ЦСФКП, заносятся в перекодер.

После передачи $n \times n$ символов шифротекста и, следовательно, использования всех $n \times n$ символов ключа в одноразовом режиме осуществляется стирание всей информации из блока таблиц одноразовых ключей комплектов A и B. Далее происходит запись новых значений таблиц одноразовых ключей из буферов для продолжения шифрования информации в одноразовом режиме. Осуществляется замена базовых таблиц в перекодере ЦСФКП.

Таким образом, функции стирания информации и замены таблиц будут производиться во взаимодействии с перекодером ЦСФКП до тех пор, пока не произойдет передача всей необходимой исходной информации в режиме одноразового шифрования от абонента A к абоненту B. При этом обеспечивается совершенная секретность представленной системы в процессе организации связи между любыми двумя абонентами.

За счет создания соответствующей схемы перекодеров ЦСФКП возможна также одновременная связь одного абонента с заданным множеством других абонентов или одной группы абонентов с другой группой абонентов. Это существенно расширяет функциональные возможности данной системы связи с абсолютной надежностью шифра.

Отметим, что на базе описанного метода шифрования информации с перекодированием можно реализовать одноразовые системы с открытой передачей ключа [1, 2]. Подобные системы обеспечивают заданный гарантированный уровень шифра и существенно экономят использование одноразовых ключей, записанных во флэш-память абонентских комплектов. Это достигается за счет случайного преобразования базовых таблиц с помощью стохастических перестановок и замен их столбцов и строк [2]. В результате осуществляется возможность повторного применения с целью шифрования информации случайно преобразованных базовых таблиц и достижения заданного гарантированного уровня стойкости шифра. Такие системы предназначены для использования при защите информации в процессе обработки ее в вычислительных комплексах [4], в том числе и для создания зашифрованных интеллектуальных поисковых систем.

Литература

[1] Насыпный В.В. Одноразовое шифрование с открытым распределением ключей// Открытые системы, 2004, № 1, с.66–69.

[2] Насыпный В.В. Способ комплексной защиты процесса обработки информации в компьютерных системах и система для осуществления способа. Международная заявка на изобретение № PCT/ RU 01/00272 от 05.07.2001.

[3] Введение в криптографию. Под общ. ред. В.В.Ященко. — М.: МЦНМО: «ЧеРо», 1999. — 272с.

[4] Насыпный В.В. Защищённые стохастические системы// Открытые системы, 2004, № 8, с. 60–61.

Предметный указатель

НОВЫЕ КНИГИ ИЗДАТЕЛЬСТВА "ТЕХНОСФЕРА"

Серия "Мир электроники"

Т. Ратхор Цифровые измерения. Методы и схемотехника

К. Фрике Вводный курс цифровой электроники. 2-е изд.

В. Варадан, К. Виной, К. Джозе ВЧ МЭМС и их применение

Э. Розеншер Оптоэлектроника

О. Ермаков Прикладная оптоэлектроника

В. Немудров, Г. Мартин Системы-на-кристалле. Проектирование и развитие

А. Медведев Печатные платы. Конструкции и материалы

А. Медведев Печатные платы. Электрохимические процессы

Дж. Фрайден Современные датчики. Справочник

Б. Эггинс Химические и биологические сенсоры

В. Федоров, Н. Сергеев, А. Кондрашин Контроль и испытания в проектировании и производстве радиоэлектронных средств

В. Мелешин Транзисторная преобразовательная техника

А. Лапин Интерфейсы. Выбор и реализация

Д. Крекрафт Аналоговая электроника

С. Редди Основы силовой электроники

Х. Шмидт-Вальтер Справочник инженера-схемотехника

Л. Каплан, К. Уайт Практические основы аналоговых и цифровых схем

М. Масленников Справочник комплектатора

Коллектив авторов под редакцией В. Лучинина Основы тестирования микросхем

Т. Зимина, В. Лучинин, А. Корляков Лаборатория на чипе

НОВЫЕ КНИГИ ИЗДАТЕЛЬСТВА "ТЕХНОСФЕРА"

Серия "Мир математики"

А. Купиллари Трудности доказательств. Как преодолеть страх перед математикой
В. Назайкинский, Б. Стернин, В. Шаталов Методы некоммутативного анализа
Дж. Шарма, К. Сингх Уравнения в частных производных для инженеров
К. Блаттер Вейвлет-анализ. Основы теории

Серия "Мир программирования"

Д. Макконнелл Основы современных алгоритмов. 2-е дополненное изд.
Р. Хаггарти Дискретная математика для программистов
Д. Сэломон Сжатие данных, изображений и звука
М. Вернер Основы кодирования

Серия "Мир цифровой обработки"

Р. Гонсалес, Р. Вудс Цифровая обработка изображений

Серия "Мир материалов и технологий"

Д. Брандон, У. Каплан Микроструктура материалов. Методы исследования и контроля
П. Харрис Углеродные нанотрубы и родственные структуры. Новые материалы XXI века
Ч. Пул, Ф. Оуэнс Нанотехнологии
Ф. Мэтьюз, Р. Роллингс Композитные материалы. Механика и технологии
В. Пантелеев, О. Егорова, Е. Клыкова Компьютерная микроскопия

Серия "Мир связи"

К. Одуан, Б. Гино Измерение времени. Основы GPS
И. Шахнович Современные технологии беспроводной связи
Р. Фриман Волоконно-оптические системы связи. 2-е дополненное изд.
Р. Морелос-Сарагоса Искусство помехоустойчивого кодирования
В. Вишневский, А. Ляхов, С. Портной, И. Шахнович Современные широкополосные сети передачи информации
Н. Слепов Англо-русский толковый словарь сокращений в области связи, компьютерныхи информационных технологий
У. Томаси Электронные системы связи

Заявки на книги присылайте по адресу:
125319 Москва, а/я 594
Издательство «Техносфера»
e-mail: knigi@technosphera.ru
sales@technosphera.ru
факс: (095) 956 33 46

В заявке обязательно указывайте
свой почтовый адрес!

Подробная информация о книгах на сайте
http://www.technosphera.ru

Н. Смарт
Криптография

Компьютерная верстка – С. А. Кулешов
Научный редактор – С. К. Ландо
Ответственный за выпуск – Л. Ф. Соловейчик

Формат 70x100/16. Печать офсетная.
Гарнитура Ньютон
Печ.л. 33. Тираж 1500 экз. Зак. № 326
Бумага офсет №1, плотность 65г/м^2

Издательство «Техносфера»
Москва, Лубянский проезд, дом 27/1

Диапозитивы изготовлены ООО «Европолиграфик»
Отпечатано в ППП «Типография «Наука»
Академиздатцентра «Наука» РАН,
121099 Москва, Шубинский пер., 6