

Б. Я. Рябко, А. Н. Фионов

# Криптография в информационном мире



Б. Я. Рябко, А. Н. Фионов

# Криптография в информационном мире

Москва  
Горячая линия Телеком  
2018

Р е ц е н з е н т : доктор физ.-мат. наук, профессор *Л. Б. Чубаров*

**Рябко Б. Я., Фионов А. Н.**

**Р98 Криптография в информационном мире.** – М.: Горячая линия – Телеком, 2018. – 300 с.: ил.  
ISBN 978-5-9912-0729-4.

Изложены основные подходы и методы современной криптографии и стеганографии для решения задач, возникающих при обработке, хранении и передаче информации. Рассмотрены основные шифры с открытыми ключами, методы цифровой подписи, криптографические протоколы, блоковые и потоковые шифры, хеш-функции, а также редко встречающиеся в литературе вопросы о конструкции доказуемо невскрываемых криптосистем, криптографии на эллиптических кривых, блокчейне и криптовалютах. Дано описание вопросов, связанных с использованием случайных и псевдослучайных чисел в системах защиты информации. Приведено описание основных идей и методов современной стеганографии. Подробно описаны алгоритмы, лежащие в основе современных отечественных и международных криптографических стандартов.

Для исследователей и специалистов, работающих в области защиты информации, будет полезна аспирантам и студентам, специализирующимся в данной области.

**ББК 32.81**

**Институт вычислительных технологий СО РАН  
Новосибирский государственный университет (НГУ)**

**Научное издание**

**Рябко Борис Яковлевич, Фионов Андрей Николаевич  
КРИПТОГРАФИЯ В ИНФОРМАЦИОННОМ МИРЕ**

**Обложка художника П. Б. Рябко  
Компьютерная верстка А. Н. Фионов**

Подписано в печать 28.02.2018 Печать цифровая. Формат 60×88/16 Уч. изд. л. 19

Тираж 500 экз. (1-й завод 100 экз.) Изд. №180729

ООО «Научно-техническое издательство «Горячая линия – Телеком», [WWW.TECHBOOK.RU](http://WWW.TECHBOOK.RU)

ISBN 978-5-9912-0729-4

© Б. Я. Рябко, А. Н. Фионов, 2018

© Научно-техническое издательство  
«Горячая линия – Телеком», 2018

# ПРЕДИСЛОВИЕ

В последние три-четыре десятилетия криптография из дисциплины, известной только небольшому числу специалистов по «закрытой» передаче информации, превратилась в мощную науку, преобразующую как мировую экономику, так и повседневную жизнь миллионов «обычных» людей. Речь идет не только о недавно появившихся криптовалютах и «идеальной» бухгалтерской книге блокчейн, но и таких операциях, как расчеты по банковским картам, управление счетами, заказ билетов, покупки, совершаемые через Интернет, и т. д. Криптографические методы позволяют защитить от нечестных или чрезмерно любопытных людей все эти операции, как и разговоры по мобильным телефонам, электронную почту и общение в Интернете.

Эта книга предназначена для исследователей инженеров, аспирантов и студентов, специализирующихся в области информационных технологий. Все необходимые сведения из теории чисел и теории вероятностей приводятся в книге, но не в виде отдельных разделов а по мере необходимости. Такой стиль, как мы надеемся, будет удобен читателям книги.

При изложении материала мы старались следовать принципу А. Эйнштейна «Все должно делаться настолько просто, насколько это возможно, но не проще» и соблюдать правило «Кратко и подробно», сформулированное одним из героев известной поэмы А. Твардовского. Поэтому мы не пытались описать всю современную криптографию и стeganографию на строгом математическом уровне и во всеми общинности, но, как нам кажется, рассмотрели основные идеи и методы без их вульгаризации. При этом в книге содержится точное описание целого ряда практически используемых методов, в том числе криптовалюты биткоин, системы блокчейн и стандартов на криптографические алгоритмы.

Мы надеемся, что эта книга поможет читателям не только понять основные задачи и методы современной криптографии и стeganографии, но и оценить красоту и изящество идей и результатов, лежащих в основе этих наук.

## Глава 1. ВВЕДЕНИЕ

Мы начинаем изложение основ криптографии с классической задачи передачи секретных сообщений от некоторого отправителя *A* к получателю *B*.

Отправитель сообщений и их получатель могут быть физическими лицами, организациями, какими-либо техническими системами. Иногда об *A* и *B* говорят как об абонентах некоторой сети, о пользователях некоторой компьютерной системы или, еще более формально, как об абстрактных «сторонах» (англоязычный термин «party») или «сущностях» (entity), участвующих в информационном взаимодействии. Но чаще бывает удобно отождествлять участников обмена с некоторыми людьми и заменить формальные обозначения *A* и *B* на Алиса и Боб.

Предполагается, что сообщения передаются по так называемому «открытым» каналу связи, в принципе доступному для прослушивания некоторым другим лицам, отличным от получателя и отправителя. Такая ситуация возникает при радиопередаче сообщений (например, посредством мобильного телефона) и возможна при использовании даже таких «проверенных» каналов связи, как проводочный телефон, телеграф, да и обычная почта. Особый интерес как средство передачи данных, стремительно завоевывающее лидирующие позиции во всем мире и в то же время чрезвычайно уязвимое с точки зрения возможности несанкционированного доступа третьих лиц, представляет Интернет. В этой среде легко реализуется не только копирование, но и подмена передаваемых сообщений.

В криптографии обычно предполагается, что у лица, передающего сообщения и (или) их принимающего, есть некоторый противник *E*, который может быть конкурентом в бизнесе, членом преступной группировки, представителем иностранной разведки или даже чрезмерно ревнивой женой, и этот противник может перехватывать сообщения, передаваемые по открытому каналу, и анализировать их. Часто удобно рассматривать противника как некую особу по имени Ева, которая имеет в своем распоряжении мощную вычислительную

технику и владеет методами криптоанализа. Естественно, Алиса и Боб хотят, чтобы их сообщения были непонятны Еве, и используют для этого специальные шифры.

Перед тем как передать сообщение по открытому каналу связи от *A* к *B*, *A* шифрует сообщение, а *B*, приняв зашифрованное сообщение, расшифровывает его, восстанавливая исходный текст. Важно то, что в рассматриваемой нами в этой главе задаче Алиса и Боб могут договариваться об используемом ими шифре (или, скорее, о некоторых его параметрах) не по открытому каналу, а по специальному «закрытому» каналу, недоступному для прослушивания противником. Такой «закрытый канал» может быть организован при помощи курьеров, или же Алиса и Боб могут обмениваться шифрами во время личной встречи и т.п. При этом надо учитывать, что обычно организация такого закрытого канала и передача по нему сообщений слишком дороги по сравнению с открытым каналом и (или) закрытый канал не может быть использован в любое время. Например, курьерская почта намного дороже обычной, передача сообщений с ее помощью происходит намного медленнее, чем, скажем, по телеграфу, да и использовать ее можно не в любое время суток и не в любой ситуации.

Чтобы быть более конкретными, рассмотрим пример шифра. Так как проблема шифрования сообщений возникла еще в глубокой древности, некоторые шифры связаны с именами известных исторических личностей и в качестве первых примеров обычно используют именно такие шифры. Мы также будем придерживаться этой традиции. Начнем с известного шифра Гая Юлия Цезаря (см., например, [2, 68]), адаптировав его к русскому языку. В этом шифре каждая буква сообщения заменяется на другую, номер которой в алфавите на три больше. Например, А заменяется на Г, Б на Д и т.д. Три последние буквы русского алфавита — Э, Ю, Я — шифруются буквами А, Б, В соответственно. Например, слово ПЕРЕМЕНА после применения к нему шифра Цезаря превращается в ТИУИПИРГ (если исключить букву Ё и считать, что в алфавите 32 буквы).

Последующие римские цезари модифицировали шифр, используя смещение в алфавите на четыре, пять и более букв. Мы можем описать их шифр в общем виде, если пронумеруем (закодируем) буквы русского алфавита числами от 0 до 31 (исключив букву Ё). Тогда

правило шифрования запишется следующим образом:

$$c = (m + k) \bmod 32, \quad (1.1)$$

где  $m$  и  $c$  — номера букв соответственно сообщения и шифротекста, а  $k$  — некоторое целое число, называемое ключом шифра (в рассмотренном выше шифре Цезаря  $k = 3$ ). (Здесь и в дальнейшем  $a \bmod b$  обозначает остаток от деления целого числа  $a$  на целое число  $b$ , причем остаток берется из множества  $\{0, 1, \dots, b - 1\}$ . Например,  $13 \bmod 5 = 3$ .)

Чтобы расшифровать зашифрованный текст, нужно применить «обратный» алгоритм

$$m = (c - k) \bmod 32. \quad (1.2)$$

Можно представить себе ситуацию, когда источник и получатель сообщений договорились использовать шифр (1.1), но для того чтобы усложнить задачу противника, решили иногда менять ключ шифра. Для этого Алиса каким-либо образом генерирует число  $k$ , передает его Бобу по закрытому каналу связи, и после этого они обмениваются сообщениями, зашифрованными с помощью этого ключа  $k$ . Замену ключа можно проводить, например, перед каждым сеансом связи или после передачи фиксированного числа букв (скажем, каждую десятку символов шифровать со своим  $k$ ) и т.п. В таком случае говорят, что ключ порождается источником ключа. Схема рассмотренной криптосистемы с секретным ключом приведена на рис. 1.1.

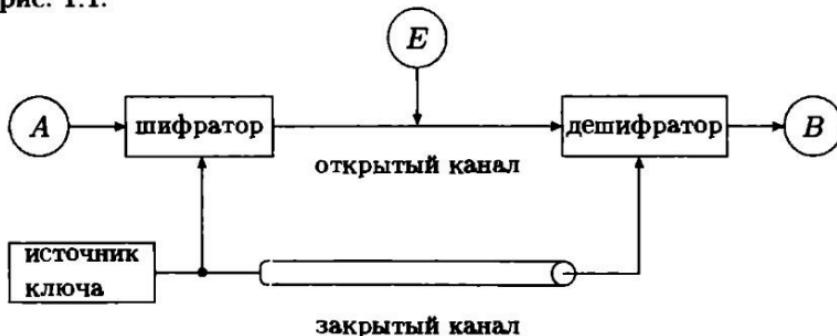


Рис. 1.1. Классическая система секретной связи

Обратимся теперь к анализу действий противника, пытающегося дешифровать сообщение и узнать секретный ключ, иными словами, вскрыть, или взломать шифр. Каждая попытка вскрытия шифра называется атакой на шифр (или на крипtosистему). В криптографии принято считать, что противник может знать использованный алгоритм шифрования, характер передаваемых сообщений и перехваченный шифротекст, но не знает секретный ключ. Это называется «правилом Керкхoffsа» (см. [68]) в честь ученого, впервые сформулировавшего основные требования к шифрам (A. Kerckhoffs, 1883). Иногда это правило кажется «перестраховкой», но такая «перестраховка» отнюдь не лишняя, если, скажем, передается распоряжение о переводе миллиона долларов с одного счета на другой.

В нашем примере Ева знает, что шифр был построен в соответствии с (1.1), что исходное сообщение было на русском языке и что был передан шифротекст ТИУИПИРГ, но ключ Еве не известен.

Наиболее очевидная попытка дешифровки — последовательный перебор всех возможных ключей (это так называемый метод «грубой силы» (brute-force attack)). Итак, Ева перебирает последовательно все возможные ключи  $k = 1, 2, \dots$ , подставляя их в алгоритм дешифрования и оценивая получающиеся результаты. Попробуем и мы использовать этот метод. Результаты дешифрования по (1.2) при различных ключах и шифротексте ТИУИПИРГ сведены в табл. 1.1. В большинстве случаев нам достаточно было дешифровать две-три буквы, чтобы отвергнуть соответствующий ключ (из-за отсутствия слова в русском языке, начинающегося с такого фрагмента).

Т а б л и ц а 1.1. Дешифровка слова ТИУИПИРГ путем перебора ключей

$k$	$m$	$k$	$m$	$k$	$m$	$k$	$m$
1	СЗТ	9	ЯЯ	17	БЧ	25	ЩП
2	РЖС	10	ИЮЙ	18	АЦБ	26	ШОЩ
3	ПЕРЕМЕНА	11	ЗЭИ	19	ЯХА	27	ЧН
4	ОДП	12	ЖЬ	20	ЮФ	28	ЦМ
5	НГ	13	ЕЫ	21	ЭУ	29	ХЛЦ
6	МВ	14	ДЬ	22	Ь	30	ФК
7	ЛВМ	15	ГЩ	23	Ы	31	УЙ
8	КАЛАЗ	16	ВШГ	24	Ъ	32	ТИУИПИРГ

Из табл. 1.1 мы видим, что был использован ключ  $k = 3$  и зашифровано сообщение ПЕРЕМЕНА. Причем для того чтобы про-

верить остальные возможные значения ключа, нам не требовалось дешифровать все восемь букв, а в большинстве случаев после анализа двух-трех букв ключ отвергался (только при  $k = 8$  надо было дешифровать пять букв, зато при  $k = 22, 23, 24$  хватало и одной, т.к. в русском языке нет слов, начинающихся с Ъ, Ъ, Ы).

Из этого примера мы видим, что рассмотренный шифр совершенно нестоеек: для его вскрытия достаточно проанализировать несколько первых букв сообщения и после этого ключ  $k$  однозначно определяется (и, следовательно, однозначно расшифровывается все сообщение).

В чем же причины нестойкости рассмотренного шифра и как можно было бы увеличить его стойкость? Рассмотрим еще один пример. Алиса спрятала важные документы в ячейке камеры хранения, снабженной пятидекадным кодовым замком. Теперь она хотела бы сообщить Бобу комбинацию цифр, открывающую ячейку. Она решила использовать аналог шифра Цезаря, адаптированный к алфавиту, состоящему из десятичных цифр:

$$c = (m + k) \bmod 10. \quad (1.3)$$

Допустим, Алиса послала Бобу шифротекст 26047. Ева пытается дешифровать его, последовательно перебирая все возможные ключи. Результаты ее попыток сведены в табл. 1.2.

Т а б л и ц а 1.2. Дешифровка сообщения  
26047 путем перебора ключей

$k$	$m$	$k$	$m$
1	15936	6	60481
2	04825	7	59370
3	93714	8	48269
4	82603	9	37158
5	71592	0	26047

Мы видим, что все полученные варианты равнозначны и Ева не может понять, какая именно комбинация истинна. Анализируя шифротекст, она не может найти значения секретного ключа. Конечно, до перехвата сообщения у Евы было  $10^5$  возможных значений кодовой комбинации, а после — только 10. Однако важно отметить то, что в данном случае всего 10 значений ключа. Поэтому при таком ключе

(одна десятичная цифра) Алиса и Боб и не могли рассчитывать на большую секретность.

В первом примере сообщение — текст на русском языке, поэтому оно подчиняется многочисленным правилам, различные буквы и их сочетания имеют различные вероятности и, в частности, многие наборы букв вообще запрещены (это свойство называется избыточностью текста). Поэтому-то и удалось легко подобрать ключ и дешифровать сообщение, т.е. избыточность позволила «взломать» шифр. В противоположность этому, во втором примере все комбинации цифр допустимы. «Язык» кодового замка не содержит избыточности. Поэтому даже простой шифр, примененный к сообщениям этого языка, становится невскрываемым. В классической работе К. Шеннона [28] построена глубокая и изящная теория шифров с секретным ключом и, в частности, предложена «правильная» количественная мера избыточности. Мы кратко коснемся этих вопросов в главе 7, а в главе 8 будут описаны современные шифры с секретным ключом.

Описанная в приведенных примерах атака называется атакой по *шифротексту*. Но часто на шифр может быть проведена атака по *известному тексту*. Это происходит, если Ева получает в свое распоряжение какие-либо открытые тексты, соответствующие ранее переданным зашифрованным. Сопоставляя пары «текст–шифротекст», Ева пытается узнать секретный ключ, чтобы с его помощью расшифровывать все последующие сообщения от Алисы к Бобу.

Можно представить себе и более «серезную» атаку — атаку по *выбранному тексту*, когда противник пользуется не только предоставленными ему парами «текст–шифротекст», но может и сам формировать нужные ему тексты и шифровать их с помощью того ключа, который он хочет узнать. Например, во время Второй мировой войны американцы, подкупив охрану, выкрали шифровальную машину в японском посольстве на два дня и имели возможность подавать ей на вход различные тексты и получать соответствующие шифровки. (Они не могли взломать машину с целью непосредственного определения заложенного в нее секретного ключа, т.к. это было бы замечено и повлекло бы за собой смену всех ключей.)

Может показаться, что атаки по известному и выбранному тексту надуманы и далеко не всегда возможны. Отчасти это так. Но разработчики современных крипtosистем стремятся сделать их неуязвимыми даже и по отношению к атакам по выбранному тексту, и

на этом пути достигнуты значительные успехи. Иногда считается, что более надежно использовать шифр, противостоящий атаке по выбранному тексту, чем организационно обеспечивать неосуществимость такой атаки, хотя наиболее осторожные пользователи делают и то, и другое.

Итак, мы познакомились с основными героями криптографии — Алисой, Бобом и Евой и с важными понятиями этой науки — шифром, ключом, атакой, открытым и защищенным каналом. Заметим, что с последним понятием связан один интригующий факт — возможно построение надежных криптосистем без защищенного канала! В таких системах Алиса и Боб вычисляют секретный ключ так, что Ева не может этого сделать. Это открытие было сделано в основополагающих работах Диффи, Хеллмана и Меркля (см., например, [48]) в 1976 году и открыло новую эру в современной криптографии. Большая часть этой книги будет связана именно с такими системами, называемыми криптосистемами с открытым ключом.

## Задачи и упражнения

- 1.1.** Определить ключи шифра Цезаря, если известны следующие пары открытый текст — шифротекст:
  - а. АПЕЛЬСИН — САЦЬНВЩЮ,
  - б. АБРИКОС — ЫЬЛГЕЙМ.
  
- 1.2.** Расшифровать следующие сообщения, зашифрованные шифром Цезаря с неизвестным ключом  $k$ ,  $0 < k < 32$ :
  - а. ФХНЗКЧ,
  - б. ЦЩЕБФ.

## Глава 2. КРИПТОСИСТЕМЫ С ОТКРЫТЫМ КЛЮЧОМ

### 2.1. Предыстория и основные идеи

Рассмотрим три задачи, решение которых поможет нам лучше понять идеи и методы криптографии с открытым ключом. Все эти задачи имеют важное практическое значение.

Первая задача — хранение паролей в компьютере. Мы знаем, что каждый пользователь в сети имеет свой секретный пароль. При входе в сеть пользователь указывает свое имя (несекретное) и затем вводит пароль. Проблема состоит в следующем: если храните пароль на диске компьютера, то Ева может прочитать его, а затем использовать для несанкционированного доступа (особенно легко это сделать, если Ева работает системным администратором этой сети). Поэтому необходимо организовать хранение паролей в компьютере так, чтобы такой «взлом» был невозможен.

Вторая задача возникла с появлением радиолокаторов и системы ПВО. При пересечении самолетом границы радиолокатор спрашивает пароль. Если пароль верный, то самолет «свой», в противном случае — «чужой». Здесь возникает такая проблема: так как пароль должен передаваться по открытому каналу (воздушной среде), то противник может прослушивать все переговоры и узнавать правильный пароль. Затем «чужой» самолет в случае запроса повторит перехваченный ранее «правильный» пароль в качестве ответа локатору и будет пропущен.

Третья задача похожа на предыдущую и возникает в компьютерных сетях с удаленным доступом, например, при взаимодействии банка и клиента. Обычно в начале сеанса банк запрашивает у клиента имя, а затем секретный пароль, но Ева может узнать пароль, т.к. линия связи открыта.

Сегодня все эти проблемы решаются с использованием криптографических методов. Решение всех этих задач основано на важном понятии односторонней функции (one-way function).

**Определение 2.1.** Пусть дана функция

$$y = f(x), \quad (2.1)$$

определенная на конечном множестве  $X$  ( $x \in X$ ), для которой существует обратная функция

$$x = f^{-1}(y). \quad (2.2)$$

Функция называется *односторонней*, если вычисление по формуле (2.1) — простая задача, требующая немного времени, а вычисление по (2.2) — задача сложная, требующая привлечения массы вычислительных ресурсов, например,  $10^6$ – $10^{10}$  лет работы мощного суперкомпьютера.

Данное определение, безусловно, неформально. Строгое определение односторонней функции может быть найдено в [61, 68], но для наших целей достаточно и вышеприведенного.

В качестве примера односторонней функции рассмотрим следующую:

$$y = a^x \bmod p, \quad (2.3)$$

где  $p$  — некоторое простое число (т.е. такое, которое делится без остатка только на себя и на единицу), а  $x$  — целое число из множества  $\{1, 2, \dots, p-1\}$ . Обратная функция обозначается

$$x = \log_a y \bmod p \quad (2.4)$$

и называется *дискретным логарифмом*.

Для того чтобы обеспечить трудность вычисления по (2.4) при использовании лучших современных компьютеров, в настоящее время используются числа размером более 512 бит. На практике часто применяются и другие односторонние функции, например, так называемые хеш-функции, оперирующие с существенно более короткими числами (они будут рассмотрены в главе 8).

Сначала мы покажем, что вычисление по (2.3) может быть выполнено достаточно быстро. Начнем с примера вычисления числа  $a^{16} \bmod p$ . Мы можем записать

$$a^{16} \bmod p = \left( \left( (a^2)^2 \right)^2 \right)^2 \bmod p,$$

т.е. значение данной функции вычисляется всего за 4 операции умножения вместо 15 при «наивном» варианте  $a \cdot a \cdot \dots \cdot a$ . На этом основан общий алгоритм.

Для описания алгоритма введем величину  $t = \lfloor \log_2 x \rfloor$  — целую часть  $\log_2 x$  (далее все логарифмы будут двоичные, поэтому в дальнейшем мы не будем указывать основание 2). Вычисляем числа ряда

$$a, \quad a^2, \quad a^4, \quad a^8, \quad \dots, \quad a^{2^t} \pmod{p}. \quad (2.5)$$

В ряду (2.5) каждое число получается путем умножения предыдущего числа самого на себя по модулю  $p$ . Запишем показатель степени  $x$  в двоичной системе счисления:

$$x = (x_t x_{t-1} \dots x_1 x_0)_2.$$

Тогда число  $y = a^x \pmod{p}$  может быть вычислено как

$$y = \prod_{i=0}^t a^{x_i \cdot 2^i} \pmod{p} \quad (2.6)$$

(все вычисления проводятся по модулю  $p$ ).

**Пример 2.1:** Пусть требуется вычислить  $3^{100} \pmod{7}$ . Имеем  $t = \lfloor \log 100 \rfloor = 6$ . Вычисляем числа ряда (2.5):

$$\begin{array}{cccccccc} a & a^2 & a^4 & a^8 & a^{16} & a^{32} & a^{64} \\ 3 & 2 & 4 & 2 & 4 & 2 & 4 \end{array} \quad (2.7)$$

Записываем показатель в двоичной системе счисления:

$$100 = (1100100)_2$$

и проводим вычисления по формуле (2.6):

$$\begin{array}{cccccccccc} a^{64} & a^{32} & & a^4 & & & & \\ 4 & 2 & \cdot & 1 & \cdot & 1 & \cdot & 4 & \cdot & 1 & \cdot & 1 & = & 4 \end{array} \quad (2.8)$$

Нам потребовалось всего 8 операций умножения (6 для вычисления ряда (2.7) и 2 для (2.8)).  $\square$

В общем случае справедливо следующее

**Утверждение 2.1** (о сложности вычислений (2.3)). *Количество операций умножения при вычислении (2.3) по описанному методу не превосходит  $2 \log x$ .*

**Доказательство.** Для вычисления чисел ряда (2.5) требуется  $t$  умножений, для вычисления  $y$  по (2.6) не более, чем  $t$  умножений (см. пример 2.1). Из условия  $t = \lfloor \log x \rfloor$ , учитывая, что  $\lfloor \log x \rfloor \leq \log x$ , делаем вывод о справедливости доказываемого утверждения.  $\square$

**Замечание.** Как будет показано в дальнейшем, при возведении в степень по модулю  $p$  имеет смысл использовать только показатели  $x < p$ . В этом случае мы можем сказать, что количество операций умножения при вычислении (2.3) не превосходит  $2 \log p$ .

Важно отметить, что столь же эффективные алгоритмы вычисления обратной функции (2.4) неизвестны. Один из методов вычисления (2.4), называемый «шаг младенца, шаг великана», будет подробно описан в разд. 3.2. Этот метод требует порядка  $2\sqrt{p}$  операций.

Покажем, что при больших  $p$  функция (2.3) действительно односторонняя, если для вычисления обратной функции используется метод «шаг младенца, шаг великана». Получаем следующий результат (табл. 2.1).

Таблица 2.1. Количество умножений для вычисления прямой и обратной функции

Количество десятичных знаков в записи $p$	Вычисление (2.3) ( $2 \log p$ умножений)	Вычисление (2.4) ( $2\sqrt{p}$ умножений)
12	$2 \cdot 40 = 80$	$2 \cdot 10^6$
60	$2 \cdot 200 = 400$	$2 \cdot 10^{30}$
90	$2 \cdot 300 = 600$	$2 \cdot 10^{45}$

Мы видим, что если использовать модули, состоящие из 50–100 десятичных цифр, то «прямая» функция вычисляется быстро, а обратная практически не вычислима. Рассмотрим, например, суперкомпьютер, который умножает два 90-значных числа за  $10^{-14}$  с (для современных компьютеров это пока не доступно). Для вычисления (2.3) такому компьютеру потребуется

$$T_{\text{выч.пр.}} = 600 \cdot 10^{-14} = 6 \cdot 10^{-12} \text{ с,}$$

а для вычисления (2.4) —

$$T_{\text{выч.обр.}} = 10^{45} \cdot 10^{-14} = 10^{31} \text{ с},$$

т.е. более  $10^{22}$  лет. Мы видим, что вычисление обратных функций практически невозможно при длине чисел порядка 90 десятичных цифр, и использование параллельных вычислений и компьютерных сетей существенно не меняет ситуацию. В рассмотренном примере мы предполагали, что обратная функция вычисляется за  $2\sqrt{p}$  операций. В настоящее время известны и более «быстрые» методы вычисления дискретного логарифма, однако общая картина та же — количество требуемых в них операций много больше  $2\log p$ . Таким образом, можно утверждать, что функция (2.3) действительно односторонняя, но с оговоркой. Никем не доказано, что обратная функция (2.4) не может быть вычислена столь же быстро, как и «прямая».

Используем одностороннюю функцию (2.3) для решения всех трех задач, описанных в начале данного раздела, не забывая, однако, что точно так же может быть использована и любая другая односторонняя функция.

Начнем с хранения паролей в памяти компьютера. Решение задачи основано на том, что пароли вообще не хранятся! Точнее, при регистрации в сети пользователь набирает свое имя и пароль; пусть, например, его имя — «фрукт», а пароль — «абрикос». Компьютер рассматривает слово «абрикос» как двоичную запись числа  $x$  и вычисляет (2.3), где  $a$  и  $p$  — два несекретные числа, возможно даже, всем известные. После этого в памяти компьютера заводится пара (имя,  $y$ ), где  $y$  вычислено по (2.3) при  $x$  = пароль. При всех дальнейших входах этого пользователя после ввода пары («фрукт», «абрикос»), компьютер вычисляет по (2.3) новое значение  $y$ нов с  $x$  = «абрикос» и сравнивает с хранящимся в памяти ранее вычисленным значением  $y$ . Если  $y$ нов совпадает с хранящимся в памяти  $y$ , соответствующим данному имени, то это законный пользователь. В противном случае это Ева.

Ева могла бы попытаться найти  $x$  по  $y$ . Однако мы видели, что уже при 90-значных числах для этого потребуется более чем  $10^{22}$  лет. Таким образом, представленная система хранения пароля весьма надежна и в настоящее время используется во многих реальных операционных системах.

Рассмотрим решение второй задачи (ПВО и самолет). Можно использовать следующий метод. Каждому «своему» самолету при-

сваивается секретное имя, известное системе ПВО и летчику, точнее, бортовому компьютеру. Пусть, например, одному из самолетов присвоено секретное имя СОКОЛ, и этот самолет приближается к границе 18 марта 2018 года в 12 час. 45 мин. Тогда перед приближением к границе бортовой компьютер самолета формирует слово

СОКОЛ 18 03 18 12 45  
(имя год месяц день часы минуты).

Другими словами, компьютер на самолете и станция ПВО прибавляют к секретному слову сведения о текущем времени и, рассматривая полученное слово как число  $x$ , вычисляют  $y = a^x \bmod p$ , где  $a$  и  $p$  не секретны. Затем самолет сообщает число  $y$  станции ПВО. Станция сравнивает вычисленное ею число  $y$  с полученным от самолета. Если вычисленное и полученное значения совпали, то самолет признается как «свой».

Противник не может взломать эту систему. Действительно, с одной стороны, он не знает секретного слова СОКОЛ и не может найти его по  $y$ , т.к. вычисление  $x$  по  $y$  занимает, скажем,  $10^{22}$  лет. С другой стороны, он не может просто скопировать  $y$  и использовать его в качестве ответа в будущем, т.к. время пересечения границы никогда не повторяется и последующие значения  $y$  будут отличаться от первоначального.

Рассмотренный вариант решения «задачи ПВО» требует точной синхронизации часов в самолете и в локаторе. Эта проблема достаточно легко решается. Например, служба навигации постоянно передает метки времени в открытом виде (время не секретно), и все самолеты и локаторы используют эти метки для синхронизации своих часов. Но есть более тонкие проблемы. Метка времени добавляется в слово  $x$  для того, чтобы все вычисляемые значения  $y$  были различны и противник не мог их повторно использовать. Однако противник может попытаться мгновенно повторить  $y$  в пределах текущей минуты. Как предотвратить эту возможность? Это первый вопрос. Другое затруднение возникает в ситуации, когда самолет посыпает число  $y$  в конце 45-й минуты, а локатор принимает его в начале 46-й. Мы предоставляем читателю возможность самостоятельно предложить вариант решения этих проблем.

Другой способ решения «задачи ПВО» возможен, если мы будем использовать дополнительный открытый канал передачи данных от локатора к самолету. Как и выше, каждый «свой» самолет и локатор

знают секретное слово (типа СОКОЛ), которое не заменяется. Обнаружив цель, локатор посыпает ей случайно сгенерированное число  $a$  («вызов»). Самолет вычисляет  $y = a^x \bmod p$ , где  $x$  — секретное слово («СОКОЛ»), и сообщает число  $y$  локатору. Локатор воспроизводит те же вычисления и сравнивает вычисленное  $y$  и принятое. В этой схеме не нужно синхронизировать часы, но, как и ранее, противник не может повторить число  $y$ , т.к. локатор всякий раз посыпает разные вызовы ( $a$ ). Интересно, что эта задача, по-видимому, была исторически первой, при решении которой использовались односторонние функции.

Третья задача решается совершенно аналогично, и оба рассмотренных метода формирования пароля применимы и используются в реальных сетевых протоколах.

## 2.2. Первая система с открытым ключом — система Диффи–Хеллмана

Эта криптосистема была открыта в середине 70-х годов американскими учеными Диффи (Whitfield Diffie) и Хеллманом (Martin Hellman) и привела к настоящей революции в криптографии и ее практических применениях. Это первая система, которая позволяла защищать информацию без использования секретных ключей, передаваемых по защищенным каналам. Для того чтобы продемонстрировать одну из схем применения таких систем, рассмотрим сеть связи с  $N$  пользователями, где  $N$  — большое число. Пусть мы хотим организовать секретную связь для каждой пары из них. Если мы будем использовать обычную систему распределения секретных ключей, то каждая пара абонентов должна быть снабжена своим секретным ключом, т.е. всего потребуется  $C_N^2 = \frac{N(N-1)}{2} \approx \frac{N^2}{2}$  ключей.

Если абонентов 100, то требуется 5000 ключей, если же абонентов  $10^4$ , то ключей должно быть  $5 \cdot 10^7$ . Мы видим, что при большом числе абонентов система снабжения их секретными ключами становится очень громоздкой и дорогостоящей.

Диффи и Хеллман решили эту проблему за счет открытого распространения и вычисления ключей. Перейдем к описанию предложенной ими системы.

Пусть строится система связи для абонентов  $A, B, C, \dots$ . У каждого абонента есть своя секретная и открытая информация. Для

организации этой системы выбирается большое простое число  $p$  и некоторое число  $g$ ,  $1 < g < p - 1$ , такое что все числа из множества  $\{1, 2, \dots, p - 1\}$  могут быть представлены как различные степени  $g \bmod p$  (известны различные подходы для нахождения таких чисел  $g$ , один из них будет представлен ниже). Числа  $p$  и  $g$  известны всем абонентам.

Абоненты выбирают большие числа  $X_A, X_B, X_C$ , которые хранят в секрете (обычно такой выбор рекомендуется проводить случайно, используя датчики случайных чисел). Каждый абонент вычисляет соответствующее число  $Y$ , которое открыто передается другим абонентам,

$$\begin{cases} Y_A = g^{X_A} \bmod p, \\ Y_B = g^{X_B} \bmod p, \\ Y_C = g^{X_C} \bmod p. \end{cases} \quad (2.9)$$

В результате получаем следующую таблицу.

Таблица 2.2. Ключи пользователей в системе Диффи-Хеллмана

Абонент	Секретный ключ	Открытый ключ
$A$	$X_A$	$Y_A$
$B$	$X_B$	$Y_B$
$C$	$X_C$	$Y_C$

Допустим, абонент  $A$  решил организовать сеанс связи с  $B$ , при этом обоим абонентам доступна открытая информация из табл. 2.2. Абонент  $A$  сообщает  $B$  по открытому каналу, что он хочет передать ему сообщение. Затем абонент  $A$  вычисляет величину

$$Z_{AB} = (Y_B)^{X_A} \bmod p \quad (2.10)$$

(никто другой кроме  $A$  этого сделать не может, т.к. число  $X_A$  секретно). В свою очередь, абонент  $B$  вычисляет число

$$Z_{BA} = (Y_A)^{X_B} \bmod p. \quad (2.11)$$

**Утверждение 2.2.**  $Z_{AB} = Z_{BA}$ .

**Доказательство.** Действительно,

$$\begin{aligned} Z_{AB} &= (Y_B)^{X_A} \bmod p = (g^{X_B})^{X_A} \bmod p = \\ &= g^{X_A X_B} \bmod p = (Y_A)^{X_B} \bmod p = Z_{BA}. \end{aligned}$$

(Здесь первое равенство следует из (2.10), второе и четвертое — из (2.9), последнее — из (2.11).)  $\square$

Отметим главные свойства системы:

- 1) абоненты  $A$  и  $B$  получили одно и то же число  $Z = Z_{AB} = Z_{BA}$ , которое не передавалось по открытой линии связи;
- 2) Ева не знает секретных чисел  $X_A, X_B, \dots$ , поэтому не может вычислить число  $Z_{AB}$  (вообще говоря, она могла бы попытаться найти секретное число  $X_A$  по  $Y_A$  (см. (2.9)), однако при больших  $p$  это практически невозможно (требуются миллионы лет)).

Абоненты  $A$  и  $B$  могут использовать  $Z_{AB}$  в качестве секретного ключа для шифрования и дешифрования данных. Таким же образом любая пара абонентов может вычислить секретный ключ, известный только им.

Остановимся теперь на упомянутой выше задаче выбора числа  $g$ . При произвольно заданном  $p$  она может оказаться трудной задачей, связанной с разложением на простые множители числа  $p - 1$ . Дело в том, что для обеспечения высокой стойкости рассмотренной системы число  $p - 1$  должно обязательно содержать большой простой множитель (в противном случае алгоритм Полига–Хеллмана, описанный, например, в [68], быстро вычисляет дискретный логарифм). Поэтому часто рекомендуют использовать следующий подход. Простое число  $p$  выбирается таким, чтобы выполнялось равенство

$$p = 2q + 1,$$

где  $q$  — также простое число. Тогда в качестве  $g$  можно взять любое число, для которого справедливы неравенства

$$1 < g < p - 1 \quad \text{и} \quad g^q \bmod p \neq 1.$$

**Пример 2.2.** Пусть  $p = 23 = 2 \cdot 11 + 1$  ( $q = 11$ ). Выберем параметр  $g$ . Попробуем взять  $g = 3$ . Проверим:  $3^{11} \bmod 23 = 1$  и значит, такое  $g$  не подходит. Возьмем  $g = 5$ . Проверим:  $5^{11} \bmod 23 = 22$ . Итак, мы выбрали параметры  $p = 23$ ,  $g = 5$ . Теперь каждый абонент выбирает секретное число и вычисляет соответствующее ему открытое число. Пусть выбраны  $X_A = 7$ ,  $X_B = 13$ . Вычисляем

$Y_A = 5^7 \bmod 23 = 17$ ,  $Y_B = 5^{13} \bmod 23 = 21$ . Пусть  $A$  и  $B$  решили сформировать общий секретный ключ. Для этого  $A$  вычисляет  $Z_{AB} = 21^7 \bmod 23 = 10$ , а  $B$  вычисляет  $Z_{BA} = 17^{13} \bmod 23 = 10$ . Теперь они имеют общий ключ 10, который не передавался по каналу связи.  $\square$

### 2.3. Элементы теории чисел

Многие криптографические алгоритмы базируются на результатах классической теории чисел. Мы рассмотрим необходимый минимум из этой теории. Классические теоремы Ферма, Эйлера и ряд других результатов из теории чисел будут даны без доказательств, которые могут быть найдены практически в любом учебнике по теории чисел (см., например, [3]). Читатели, знакомые с теорией чисел, могут непосредственно перейти к разд. 2.4.

**Определение 2.2.** Целое положительное число  $p$  называется *простым*, если оно не делится ни на какое другое число, кроме самого себя и единицы.

**Пример 2.3.** Числа 11, 23 — простые; числа 27, 33 — составные (27 делится на 3 и на 9, 33 делится на 3 и на 11).  $\square$

**Теорема 2.3** (основная теорема арифметики). *Любое целое положительное число может быть представлено в виде произведения простых чисел, причем единственным образом.*

**Пример 2.4.**  $27 = 3 \cdot 3 \cdot 3$ ,  $33 = 3 \cdot 11$ .  $\square$

**Определение 2.3.** Два числа называются *взаимно простыми*, если они не имеют ни одного общего делителя кроме единицы.

**Пример 2.5.** Числа 27 и 28 взаимно просты (у них нет общих делителей кроме единицы), числа 27 и 33 — нет (у них есть общий делитель 3).  $\square$

**Определение 2.4** (функция Эйлера). Пусть дано целое число  $N \geq 1$ . Значение функции Эйлера  $\varphi(N)$  равно количеству чисел в ряду  $1, 2, 3, \dots, N-1$ , взаимно простых с  $N$ .

Пример 2.6.

$$\begin{array}{ll} \varphi(10) = ? \\ 1, \cancel{2}, 3, \cancel{4}, \cancel{5}, \cancel{6}, 7, \cancel{8}, 9, \\ \varphi(10) = 4 \end{array}$$

$$\begin{array}{ll} \varphi(12) = ? \\ 1, \cancel{2}, \cancel{3}, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, \cancel{9}, \cancel{10}, 11 \\ \varphi(12) = 4 \end{array}$$

(здесь зачеркнуты числа, не взаимнопростые с аргументом).  $\square$

**Утверждение 2.4.** Если  $p$  — простое число, то  $\varphi(p) = p - 1$ .

**Доказательство.** В ряду  $1, 2, 3, \dots, p-1$  все числа взаимно просты с  $p$ , т.к.  $p$  — простое число и по определению не делится ни на какое другое число.  $\square$

**Утверждение 2.5.** Пусть  $p$  и  $q$  — два различных простых числа ( $p \neq q$ ). Тогда  $\varphi(pq) = (p-1)(q-1)$ .

**Доказательство.** В ряду  $1, 2, \dots, pq-1$  не взаимнопростыми с  $pq$  будут числа

$$p, 2p, 3p, \dots, (q-1)p \quad \text{и} \quad q, 2q, 3q, \dots, (p-1)q.$$

Всего таких чисел будет  $(q-1) + (p-1)$ . Следовательно, количество чисел, взаимнопростых с  $pq$ , будет

$$pq - 1 - (p-1) - (q-1) = pq - q - p + 1 = (p-1)(q-1).$$

$\square$

**Теорема 2.6 (Ферма).** Пусть  $p$  — простое число и  $0 < a < p$ . Тогда

$$a^{p-1} \bmod p = 1.$$

Пример 2.7.  $p = 13$ ,  $a = 2$ ;

$$2^{12} \bmod 13 = (2^2)^2 \cdot ((2^2)^2)^2 \bmod 13 = 3 \cdot 9 \bmod 13 = 1,$$

$$10^{10} \bmod 11 = 10^2 \cdot ((10^2)^2)^2 \bmod 11 = 1 \cdot 1 = 1. \quad \square$$

**Теорема 2.7 (Эйлер).** Пусть  $a$  и  $b$  — взаимно простые числа. Тогда

$$a^{\varphi(b)} \bmod b = 1.$$

Теорема Ферма является частным случаем теоремы Эйлера, когда  $b$  — простое число.

Пример 2.8.

$$\varphi(12) = 4,$$

$$5^4 \bmod 12 = (5^2)^2 \bmod 12 = (1^2)^2 \bmod 12 = 1.$$

$$\varphi(21) = 2 \cdot 6 = 12,$$

$$2^{12} \bmod 21 = 2^4 \cdot (2^4)^2 \bmod 21 = 16 \cdot 4 \bmod 21 = 1.$$

□

Нам понадобится еще одна теорема, близкая к теореме Эйлера.

**Теорема 2.8.** *Если  $p$  и  $q$  — простые числа,  $p \neq q$  и  $k$  — произвольное целое число, то*

$$a^{k\varphi(pq)+1} \bmod (pq) = a. \quad (2.12)$$

Пример 2.9. Возьмем  $p = 5$ ,  $q = 7$ . Тогда  $pq = 35$ , а функция Эйлера —  $\varphi(35) = 4 \cdot 6 = 24$ . Рассмотрим случай  $k = 2$ , т.е. будем возводить числа в степень  $2 \cdot 24 + 1 = 49$ . Получим

$$9^{49} \bmod 35 = 9, \quad 23^{49} \bmod 35 = 23.$$

Это не удивительно, т.к. каждое из чисел 9 и 23 взаимно просто с модулем 35, и по теореме Эйлера  $9^{24} \bmod 35 = 1$ ,  $23^{24} \bmod 35 = 1$ . Однако теорема 2.8 остается верной и для следующих чисел:

$$10^{49} \bmod 35 = 10, \quad 28^{49} \bmod 35 = 28,$$

в то время как теорема Эйлера для них не применима (каждое из чисел 10 и 28 не взаимно просто с модулем 35 и  $10^{24} \bmod 35 = 15$ ,  $28^{24} \bmod 35 = 21$ ). □

**Определение 2.5.** Пусть  $a$  и  $b$  — два целых положительных числа. Наибольший общий делитель чисел  $a$  и  $b$  есть наибольшее число  $c$ , которое делит и  $a$  и  $b$ :

$$c = \gcd(a, b).$$

(Обозначение  $\text{gcd}$  для наибольшего общего делителя происходит от английских слов *greatest common divisor* и принято в современной литературе.)

Пример 2.10.  $\text{gcd}(10, 15) = 5$ ;  $\text{gcd}(8, 28) = 4$ . □

Для нахождения наибольшего общего делителя можно использовать следующий алгоритм, известный как алгоритм Евклида.

### Алгоритм 2.1. Алгоритм Евклида

ВХОД: Положительные целые числа  $a, b$ ,  $a \geq b$ .

ВЫХОД: Наибольший общий делитель  $\text{gcd}(a, b)$ .

1. WHILE  $b \neq 0$  DO
2.         $r \leftarrow a \bmod b$ ,  $a \leftarrow b$ ,  $b \leftarrow r$ .
3. RETURN  $a$ .

Пример 2.11. Покажем, как с помощью алгоритма Евклида вычисляется  $\text{gcd}(28, 8)$ :

$$\begin{array}{r} a : 28 \quad 8 \quad 4 \\ b : \quad 8 \quad 4 \quad 0 \\ r : \quad 4 \quad 0 \end{array}$$

Здесь каждый столбец представляет собой очередную итерацию алгоритма. Процесс продолжается до тех пор, пока  $b$  не станет равным нулю. Тогда в значении переменной  $a$  содержится ответ (4). □

Для многих криптографических систем, рассматриваемых в следующих разделах и главах, актуален так называемый обобщенный алгоритм Евклида, с которым связана следующая теорема.

**Теорема 2.9.** Пусть  $a$  и  $b$  — два целых положительных числа. Тогда существуют целые (не обязательно положительные) числа  $x$  и  $y$ , такие что

$$ax + by = \text{gcd}(a, b). \quad (2.13)$$

Обобщенный алгоритм Евклида служит для отыскания  $\text{gcd}(a, b)$  и  $x, y$ , удовлетворяющих (2.13). Введем три строки  $U = (u_1, u_2, u_3)$ ,  $V = (v_1, v_2, v_3)$  и  $T = (t_1, t_2, t_3)$ . Тогда алгоритм записывается следующим образом.

**Алгоритм 2.2. Обобщенный алгоритм Евклида**

**ВХОД:** Положительные целые числа  $a, b$ ,  $a \geq b$ .

**ВЫХОД:**  $\gcd(a, b)$ ,  $x$ ,  $y$ , удовлетворяющие (2.13).

1.  $U \leftarrow (a, 1, 0)$ ,  $V \leftarrow (b, 0, 1)$ .
2. WHILE  $v_1 \neq 0$  DO
3.      $q \leftarrow u_1 \operatorname{div} v_1$ ;
4.      $T \leftarrow (u_1 \operatorname{mod} v_1, u_2 - qv_2, u_3 - qv_3)$ ;
5.      $U \leftarrow V$ ,  $V \leftarrow T$ .
6. RETURN  $U = (\gcd(a, b), x, y)$ .

Результат содержится в строке  $U$ .

Операция  $\operatorname{div}$  в алгоритме — это целочисленное деление

$$a \operatorname{div} b = \lfloor a/b \rfloor.$$

Доказательство корректности алгоритма 2.2 может быть найдено в [1, 13].

**Пример 2.12.** Пусть  $a=28$ ,  $b=19$ . Найдем числа  $x$  и  $y$ , удовлетворяющие (2.13).

$U$	28	1	0	
$V$	$U$	.19	0	1
$T$	$V$	9	1	$-1$
$T$	$V$	1	$-2$	3
$T$	$V$	0	19	$-28$
				$q = 9$

Поясним представленную схему. Вначале в строку  $U$  записываются числа  $(28, 1, 0)$ , а в строку  $V$  — числа  $(19, 0, 1)$  (это первые две строки на схеме). Вычисляется строка  $T$  (третья строка в схеме). После этого в качестве строки  $U$  берется вторая строка в схеме, а в качестве  $V$  — третья, и опять вычисляется строка  $T$  (четвертая строка в схеме). Этот процесс продолжается до тех пор, пока первый элемент строки  $V$  не станет равным нулю. Тогда предпоследняя строка в схеме содержит ответ. В нашем случае  $\gcd(28, 19) = 1$ ,  $x = -2$ ,  $y = 3$ . Выполним проверку:  $28 \cdot (-2) + 19 \cdot 3 = 1$ .  $\square$

Рассмотрим одно важное применение обобщенного алгоритма Евклида. Во многих задачах криптографии для заданных чисел  $c$ ,  $m$  требуется находить такое число  $d < m$ , что

$$cd \operatorname{mod} m = 1. \quad (2.14)$$

Отметим, что такое  $d$  существует тогда и только тогда, когда числа  $c$  и  $m$  взаимно простые.

**Определение 2.6.** Число  $d$ , удовлетворяющее (2.14), называется *инверсией*  $c$  по модулю  $m$  и часто обозначается  $c^{-1} \bmod m$ .

Данное обозначение для инверсии довольно естественно, т.к. мы можем теперь переписать (2.14) в виде

$$cc^{-1} \bmod m = 1.$$

Умножение на  $c^{-1}$  соответствует делению на  $c$  при вычислениях по модулю  $m$ . По аналогии можно ввести произвольные отрицательные степени при вычислениях по модулю  $m$ :

$$c^{-e} = (c^e)^{-1} = (c^{-1})^e \pmod{m}.$$

**Пример 2.13.**  $3 \cdot 4 \bmod 11 = 1$ , поэтому число 4 — это инверсия числа 3 по модулю 11. Можно записать  $3^{-1} \bmod 11 = 4$ . Число  $5^{-2} \bmod 11$  может быть найдено двумя способами:

$$5^{-2} \bmod 11 = (5^2 \bmod 11)^{-1} \bmod 11 = 3^{-1} \bmod 11 = 4,$$

$$5^{-2} \bmod 11 = (5^{-1} \bmod 11)^2 \bmod 11 = 9^2 \bmod 11 = 4.$$

При вычислениях по второму способу мы использовали равенство  $5^{-1} \bmod 11 = 9$ . Действительно,  $5 \cdot 9 \bmod 11 = 45 \bmod 11 = 1$ .  $\square$

Покажем, как можно вычислить инверсию с помощью обобщенного алгоритма Евклида. Равенство (2.14) означает, что для некоторого целого  $k$

$$cd - km = 1. \quad (2.15)$$

Учитывая, что  $c$  и  $m$  взаимно просты, перепишем (2.15) в виде

$$m(-k) + cd = \gcd(m, c), \quad (2.16)$$

что полностью соответствует (2.13), здесь только по-другому обозначены переменные. Поэтому, чтобы вычислить  $c^{-1} \bmod m$ , т.е. найти число  $d$ , нужно просто использовать обобщенный алгоритма Евклида для решения уравнения (2.16). Заметим, что значение переменной  $k$  нас не интересует, поэтому можно не вычислять вторые элементы строк  $U$ ,  $V$ ,  $T$ . Кроме того, если число  $d$  получается отрицательным, то нужно прибавить к нему  $m$ , т.к. по определению число  $a \bmod m$  берется из множества  $\{0, 1, \dots, m-1\}$ .

Пример 2.14. Вычислим  $7^{-1} \bmod 11$ . Используем такую же схему записи вычислений, как в примере 2.12 :

$$\begin{array}{rcc}
 11 & 0 \\
 7 & 1 \\
 4 & -1 & q = 1 \\
 3 & 2 & q = 1 \\
 1 & -3 & q = 1 \\
 0 & 11 & q = 3.
 \end{array}$$

Получаем  $d = -3$  и  $d \bmod 11 = 11 - 3 = 8$ , т.е.  $7^{-1} \bmod 11 = 8$ . Проверим результат:  $7 \cdot 8 \bmod 11 = 56 \bmod 11 = 1$ .  $\square$

Одной из важнейших операций в криптографии с открытыми ключами является операция возведения в степень по модулю. Идея построения эффективного алгоритма возведения степень была ранее проиллюстрирована с помощью (2.5) и (2.6). Рассмотренный алгоритм можно реализовать и без хранения в памяти ряда чисел (2.5). Дадим описание этого алгоритма в форме, пригодной для непосредственной программной реализации. В названии алгоритма отражен тот факт, что биты показателя степени просматриваются справа налево, т.е. от младшего к старшему.

### Алгоритм 2.3. ВОЗВЕДЕНИЕ В СТЕПЕНЬ (СПРАВА НАЛЕВО)

ВХОД: Целые числа  $a$ ,  $x = (x_t x_{t-1} \dots x_0)_2$ ,  $p$ .

ВЫХОД: Число  $y = a^x \bmod p$ .

1.  $y \leftarrow 1$ ,  $s \leftarrow a$ .
2. FOR  $i = 0, 1, \dots, t$  DO
3.     IF  $x_i = 1$  THEN  $y \leftarrow y \cdot s \bmod p$ ;
4.      $s \leftarrow s \cdot s \bmod p$ .
5. RETURN  $y$ .

Чтобы показать, что по представленному алгоритму действительно вычисляется  $y$  согласно (2.6), запишем степени переменных после каждой итерации цикла. Пусть  $x = 100 = (1100100)_2$ , как в примере 2.1, тогда:

$i:$	0	1	2	3	4	5	6
$x_i:$	0	0	1	0	0	1	1
$y:$	1	1	$a^4$	$a^4$	$a^4$	$a^{36}$	$a^{100}$
$s:$	$a^2$	$a^4$	$a^8$	$a^{16}$	$a^{32}$	$a^{64}$	$a^{128}$

В некоторых ситуациях более эффективным оказывается следующий алгоритм, в котором биты показателя степени просматриваются слева направо, т.е. от старшего к младшему.

#### Алгоритм 2.4. ВОЗВЕДЕНИЕ В СТЕПЕНЬ (СЛЕВА НАПРАВО)

ВХОД: Целые числа  $a$ ,  $x = (x_t x_{t-1} \dots x_0)_2$ ,  $p$ .

ВЫХОД: Число  $y = a^x \bmod p$ .

1.  $y \leftarrow 1$ .
2. FOR  $i = t, t-1, \dots, 0$  DO
3.      $y \leftarrow y \cdot y \bmod p$ ;
4.     IF  $x_i = 1$  THEN  $y \leftarrow y \cdot a \bmod p$ .
5. RETURN  $y$ .

Чтобы убедиться в том, что алгоритм 2.4 вычисляет то же самое, что и алгоритм 2.3, запишем степени переменной  $y$  после каждой итерации цикла для  $x = 100$ :

$i:$	6	5	4	3	2	1	0
$x_i:$	1	1	0	0	1	0	0
$y:$	$a$	$a^3$	$a^6$	$a^{12}$	$a^{25}$	$a^{50}$	$a^{100}$

Приведенных в данном разделе сведений из теории чисел будет достаточно для описания основных криптографических алгоритмов и методов.

## 2.4. Шифр Шамира

Этот шифр, предложенный Шамиром (Adi Shamir), был первым, позволяющим организовать обмен секретными сообщениями по открытой линии связи для лиц, которые не имеют никаких защищенных каналов и секретных ключей и, возможно, никогда не видели друг друга. (Напомним, что система Диффи-Хеллмана позволяет

сформировать только секретное слово, а передача сообщения потребует использования некоторого шифра, где это слово будет использоваться как ключ.)

Перейдем к описанию системы. Пусть есть два абонента  $A$  и  $B$ , соединенные линией связи.  $A$  хочет передать сообщение  $m$  абоненту  $B$  так, чтобы никто не узнал его содержание.  $A$  выбирает случайное большое простое число  $p$  и открыто передает его  $B$ . Затем  $A$  выбирает два числа  $c_A$  и  $d_A$ , такие что

$$c_A d_A \bmod (p - 1) = 1. \quad (2.17)$$

Эти числа  $A$  держит в секрете и передавать не будет.  $B$  тоже выбирает два числа  $c_B$  и  $d_B$ , такие что

$$c_B d_B \bmod (p - 1) = 1, \quad (2.18)$$

и держит их в секрете.

После этого  $A$  передает свое сообщение  $m$ , используя трехступенчатый протокол. Если  $m < p$  ( $m$  рассматривается как число), то сообщение  $m$  передается сразу, если же  $m \geq p$ , то сообщение представляется в виде  $m_1, m_2, \dots, m_t$ , где все  $m_i < p$ , и затем передаются последовательно  $m_1, m_2, \dots, m_t$ . При этом для кодирования каждого  $m_i$  лучше выбирать случайно новые пары  $(c_A, d_A)$  и  $(c_B, d_B)$  — в противном случае надежность системы понижается. В настоящее время такой шифр, как правило, используется для передачи чисел, например, секретных ключей, значения которых меньше  $p$ . Таким образом, мы будем рассматривать только случай  $m < p$ . Дадим описание протокола.

**Шаг 1.**  $A$  вычисляет число

$$x_1 = m^{c_A} \bmod p, \quad (2.19)$$

где  $m$  — исходное сообщение, и пересыпает  $x_1$  к  $B$ .

**Шаг 2.**  $B$ , получив  $x_1$ , вычисляет число

$$x_2 = x_1^{c_B} \bmod p \quad (2.20)$$

и передает  $x_2$  к  $A$ .

**Шаг 3.**  $A$  вычисляет число

$$x_3 = x_2^{d_A} \bmod p \quad (2.21)$$

и передает его  $B$ .

**Шаг 4.**  $B$ , получив  $x_3$ , вычисляет число

$$x_4 = x_3^{d_B} \bmod p. \quad (2.22)$$

**Утверждение 2.10** (свойства протокола Шамира).

- 1)  $x_4 = m$ , т.е. в результате реализации протокола от  $A$  к  $B$  действительна передается исходное сообщение;
- 2) злоумышленник не может узнать, какое сообщение было передано.

**Доказательство.** Вначале заметим, что любое целое число  $e \geq 0$  может быть представлено в виде  $e = k(p-1) + r$ , где  $r = e \bmod (p-1)$ . Поэтому на основании теоремы Ферма

$$\begin{aligned} x^e \bmod p &= x^{k(p-1)+r} \bmod p = \\ &= (1^k \cdot x^r) \bmod p = x^{e \bmod (p-1)} \bmod p. \end{aligned} \quad (2.23)$$

Справедливость первого пункта утверждения вытекает из следующей цепочки равенств:

$$\begin{aligned} x_4 &= x_3^{d_B} \bmod p = (x_2^{d_A})^{d_B} \bmod p = \\ &= (x_1^{c_B})^{d_A d_B} \bmod p = (m^{c_A})^{c_B d_A d_B} \bmod p = \\ &= m^{c_A d_A c_B d_B} \bmod p = m^{(c_A d_A c_B d_B) \bmod (p-1)} \bmod p = m \end{aligned}$$

(предпоследнее равенство следует из (2.23), а последнее выполняется в силу (2.17) и (2.18)).

Доказательство второго пункта утверждения основано на предположении, что для злоумышленника, пытающегося определить  $m$ , не существует стратегии более эффективной, чем следующая. Вначале он вычисляет  $c_B$  из (2.20), затем находит  $d_B$  и, наконец, вычисляет  $x_4 = m$  по (2.22). Но для осуществления этой стратегии злоумышленник должен решить задачу дискретного логарифмирования (2.20), что практически невозможно при больших  $p$ .  $\square$

Опишем метод нахождения пар  $c_A, d_A$  и  $c_B, d_B$ , удовлетворяющих (2.17) и (2.18). Достаточно описать только действия для абонента  $A$ , т.к. действия для  $B$  совершенно аналогичны. Число  $c_A$  выбираем случайно так, чтобы оно было взаимно простым с  $p - 1$  (поиск целесообразно вести среди нечетных чисел, т.к.  $p - 1$  четно). Затем вычисляем  $d_A$  с помощью обобщенного алгоритма Евклида, как это было объяснено в разд. 2.3.

**Пример 2.15.** Пусть  $A$  хочет передать  $B$  сообщение  $m = 10$ .  $A$  выбирает  $p = 23$ ,  $c_A = 7$  ( $\gcd(7, 22) = 1$ ) и вычисляет  $d_A = 19$ . Аналогично,  $B$  выбирает параметры  $c_B = 5$  (взаимно простое с 22) и  $d_B = 9$ . Переходим к протоколу Шамира.

Шаг 1.  $x_1 = 10^7 \pmod{23} = 14$ .

Шаг 2.  $x_2 = 14^5 \pmod{23} = 15$ .

Шаг 3.  $x_3 = 15^{19} \pmod{23} = 19$ .

Шаг 4.  $x_4 = 19^9 \pmod{23} = 10$ .

Таким образом,  $B$  получил передаваемое сообщение  $m = 10$ .  $\square$

## 2.5. Шифр Эль-Гамаля

Пусть имеются абоненты  $A, B, C, \dots$ , которые хотят передавать друг другу зашифрованные сообщения, не имея никаких защищенных каналов связи. В этом разделе мы рассмотрим шифр, предложенный Эль-Гамалем (Taher ElGamal), который решает эту задачу, используя, в отличие от шифра Шамира, только одну пересылку сообщения. Фактически здесь используется схема Диффи-Хеллмана, чтобы сформировать общий секретный ключ для двух абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот ключ. Для каждого следующего сообщения секретный ключ вычисляется заново. Переходим к точному описанию метода.

Для всей группы абонентов выбираются некоторое большое простое число  $p$  и число  $g$ , такие что различные степени  $g$  суть различные числа по модулю  $p$  (см. разд. 2.2). Числа  $p$  и  $g$  передаются абонентам в открытом виде (они могут использоваться всеми абонентами сети).

Затем каждый абонент группы выбирает свое секретное число  $c_i$ ,  $1 < c_i < p - 1$ , и вычисляет соответствующее ему открытое число  $d_i$ ,

$$d_i = g^{c_i} \bmod p. \quad (2.24)$$

В результате получаем таблицу 2.3.

Таблица 2.3. Ключи пользователей в системе Эль-Гамала

Абонент	Секретный ключ	Открытый ключ
$A$	$c_A$	$d_A$
$B$	$c_B$	$d_B$
$C$	$c_C$	$d_C$

Покажем теперь, как  $A$  передает сообщение  $m$  абоненту  $B$ . Будем предполагать, как и при описании шифра Шамира, что сообщение представлено в виде числа  $m < p$ .

**Шаг 1.**  $A$  формирует случайное число  $k$ ,  $1 \leq k \leq p - 2$ , вычисляет числа

$$r = g^k \bmod p, \quad (2.25)$$

$$e = m \cdot d_B^k \bmod p \quad (2.26)$$

и передает пару чисел  $(r, e)$  абоненту  $B$ .

**Шаг 2.**  $B$ , получив  $(r, e)$ , вычисляет

$$m' = e \cdot r^{p-1-c_B} \bmod p. \quad (2.27)$$

**Утверждение 2.11** (свойства шифра Эль-Гамала).

- 1) Абонент  $B$  получил сообщение, т.е.  $m' = m$ ;
- 2) противник, зная  $p$ ,  $g$ ,  $d_B$ ,  $r$  и  $e$ , не может вычислить  $m$ .

**Доказательство.** Подставим в (2.27) значение  $e$  из (2.26):

$$m' = m \cdot d_B^k \cdot r^{p-1-c_B} \bmod p.$$

Теперь вместо  $r$  подставим (2.25), а вместо  $d_B$  — (2.24):

$$\begin{aligned} m' &= m \cdot (g^{c_B})^k \cdot (g^k)^{p-1-c_B} \bmod p = \\ &= m \cdot g^{c_B k + k(p-1) - k c_B} \bmod p = m \cdot g^{k(p-1)} \bmod p. \end{aligned}$$

По теореме Ферма

$$g^{k(p-1)} \bmod p = 1^k \bmod p = 1,$$

и, таким образом, мы получаем первую часть утверждения.

Для доказательства второй части заметим, что противник не может вычислить  $k$  в равенстве (2.25), т.к. это задача дискретного логарифмирования. Следовательно, он не может вычислить  $m$  в равенстве (2.26), т.к.  $m$  было умножено на неизвестное ему число. Противник также не может воспроизвести действия законного получателя сообщения (абонента  $B$ ), т.к. ему не известно секретное число  $c_B$  (вычисление  $c_B$  на основании (2.24) — также задача дискретного логарифмирования).  $\square$

Пример 2.16. Передадим сообщение  $m = 15$  от  $A$  к  $B$ . Выберем параметры аналогично тому, как это было сделано в примере 2.2 стр. 19. Возьмем  $p = 23$ ,  $g = 5$ . Пусть абонент  $B$  выбрал для себя секретное число  $c_B = 13$  и вычислил по (2.24)

$$d_B = 5^{13} \bmod 23 = 21.$$

Абонент  $A$  выбирает случайно число  $k$ , например  $k = 7$ , и вычисляет по (2.25), (2.26):

$$r = 5^7 \bmod 23 = 17, \quad e = 15 \cdot 21^7 \bmod 23 = 15 \cdot 10 \bmod 23 = 12.$$

Теперь  $A$  посыпает к  $B$  зашифрованное сообщение в виде пары чисел  $(17, 12)$ .  $B$  вычисляет по (2.27)

$$m' = 12 \cdot 17^{23-1-13} \bmod 23 = 12 \cdot 17^9 \bmod 23 = 12 \cdot 7 \bmod 23 = 15.$$

Мы видим, что  $B$  смог расшифровать переданное сообщение.  $\square$

Ясно, что по аналогичной схеме могут передавать сообщения все абоненты в сети. Заметим, что любой абонент, знающий открытый ключ абонента  $B$ , может посыпать ему сообщения, зашифрованные с помощью открытого ключа  $d_B$ . Но только абонент  $B$ , и никто другой, может расшифровать эти сообщения, используя известный только ему секретный ключ  $c_B$ . Отметим также, что объем шифра в два раза превышает объем сообщения, но требуется только одна передача данных (при условии, что таблица с открытыми ключами заранее известна всем абонентам).

## 2.6. Односторонняя функция с «лазейкой» и шифр RSA

Названный в честь его разработчиков Ривеста (Ron Rivest), Шамира (Adi Shamir) и Адлемана (Leonard Adleman), этот шифр до сих пор является одним из самых широко используемых.

Мы видели, что шифр Шамира полностью решает задачу обмена сообщениями, закрытыми для прочтения, в случае, когда абоненты могут пользоваться только открытыми линиями связи. Однако при этом сообщение пересыпается три раза от одного абонента к другому, что является недостатком. Шифр Эль-Гамаля позволяет решить ту же задачу за одну пересылку данных, но объем передаваемого шифротекста в два раза превышает объем сообщения. Система RSA лишена подобных недостатков. Интересно то, что она базируется на другой односторонней функции, отличной от дискретного логарифма. Кроме того, здесь мы встретимся с еще одним изобретением современной криптографии – *односторонней функцией с «лазейкой»* (trapdoor function).

Эта система базируется на следующих двух фактах из теории чисел:

- 1) задача проверки числа на простоту является сравнительно легкой;
- 2) задача разложения чисел вида  $n = pq$  ( $p$  и  $q$  – простые числа) на множители является очень трудной, если мы знаем только  $n$ , а  $p$  и  $q$  – большие числа (это так называемая задача факторизации).

Пусть в нашей системе есть абоненты  $A, B, C, \dots$ . Каждый абонент выбирает случайно два больших простых числа  $P$  и  $Q$ . Затем он вычисляет число

$$N = PQ. \quad (2.28)$$

(Число  $N$  является открытой информацией, доступной другим абонентам.) После этого абонент вычисляет число  $\phi = (P - 1)(Q - 1)$  и выбирает некоторое число  $d < \phi$ , взаимно простое с  $\phi$ , и по обобщенному алгоритму Евклида находит число  $c$ , такое что

$$cd \bmod \phi = 1. \quad (2.29)$$

Таблица 2.4. Ключи пользователей в системе RSA

Абонент	Секретный ключ	Открытый ключ
$A$	$c_A$	$d_A, N_A$
$B$	$c_B$	$d_B, N_B$
$C$	$c_C$	$d_C, N_C$

Вся информация, связанная с абонентами и являющаяся их открытыми и секретными ключами, представлена в табл. 2.4.

Опишем протокол RSA. Пусть Алиса хочет передать сообщение  $m$  Бобу, причем сообщение  $m$  рассматривается как число, удовлетворяющее неравенству  $m < N_B$  (далее индекс  $B$  указывает на то, что соответствующие параметры принадлежат Бобу).

**Шаг 1.** Алиса шифрует сообщение по формуле

$$e = m^{d_B} \pmod{N_B}, \quad (2.30)$$

используя открытые параметры Боба, и пересыпает  $e$  по открытой линии.

**Шаг 2.** Боб, получивший зашифрованное сообщение, вычисляет

$$m' = e^{c_B} \pmod{N_B}. \quad (2.31)$$

**Утверждение 2.12.** Для описанного протокола  $m' = m$ , т.е. абонент  $B$  получает исходящее от  $A$  сообщение.

**Доказательство.** По построению протокола

$$m' = e^{c_B} \pmod{N_B} = m^{d_B c_B} \pmod{N_B}.$$

Равенство (2.29) означает, что для некоторого  $k$

$$c_B d_B = k \phi_B + 1.$$

Согласно утверждению 2.5

$$\phi_B = (P_B - 1)(Q_B - 1) = \varphi(N_B),$$

где  $\varphi(\cdot)$  — функция Эйлера. Отсюда и из теоремы 2.8 следует

$$m' = m^{k \varphi(N_B) + 1} \pmod{N_B} = m.$$

□

**Утверждение 2.13** (свойства протокола RSA).

- 1) Протокол шифрует и дешифрует информацию корректно;
- 2) злоумышленник, перехватывающий все сообщения и знающий всю открытую информацию, не сможет найти исходное сообщение при больших  $P$  и  $Q$ .

**Доказательство.** Первое свойство протокола следует из утверждения 2.12. Для доказательства второго свойства заметим, что злоумышленник знает только открытые параметры  $N$  и  $d$ . Для того чтобы найти  $c$ , он должен знать значение  $\phi = (P - 1)(Q - 1)$ , а для этого, в свою очередь, ему требуется знать  $P$  и  $Q$ . Вообще говоря, он может найти  $P$  и  $Q$ , разложив  $N$  на множители, однако это трудная задача (см. пункт 2 в начале раздела). Отметим, что выбор больших случайных  $P$  и  $Q$  возможен за приемлемое время, т.к. справедлив пункт 1.  $\square$

Односторонняя функция  $y = x^d \bmod N$ , применяемая в системе RSA, обладает так называемой «лазейкой», позволяющей легко вычислить обратную функцию  $x = \sqrt[d]{y} \bmod N$ , если известно разложение  $N$  на простые множители. (Действительно, легко вычислить  $\phi = (P - 1)(Q - 1)$ , а затем  $c = d^{-1} \bmod \phi$ .) Если  $P$  и  $Q$  неизвестны, то вычисление значения обратной функции практически невозможно, а найти  $P$  и  $Q$  по  $N$  очень трудно, т.е. знание  $P$  и  $Q$  – это «лазейка» или «потайной ход»). Такие односторонние функции с лазейкой находят применение и в других разделах криптографии.

Отметим, что для схемы RSA важно, чтобы каждый абонент выбирал собственную пару простых чисел  $P$  и  $Q$ , т.е. все модули  $N_A, N_B, N_C, \dots$  должны быть различны (в противном случае один абонент мог бы читать зашифрованные сообщения, предназначенные для другого абонента). Однако этого не требуется от второго открытого параметра  $d$ . Параметр  $d$  может быть одинаковым у всех абонентов. Часто рекомендуется выбирать  $d = 3$  (при соответствующем выборе  $P$  и  $Q$ , см. [68]). Тогда шифрование выполняется максимально быстро, всего за два умножения.

**Пример 2.17.** Допустим, Алиса хочет передать Бобу сообщение  $m = 15$ . Пусть Боб выбрал следующие параметры:

$$P_B = 3, Q_B = 11, N_B = 33, d_B = 3$$

(3 взаимно просто с  $\varphi(33) = 20$ ). Найдем  $c_B$  с помощью обобщенного алгоритма Евклида:

$$c_B = 7$$

(проверим:  $3 \cdot 7 \bmod 20 = 1$ ). Кодируем  $m$  по формуле (2.30):

$$e = 15^3 \bmod 33 = 15^2 \cdot 15 \bmod 33 = 27 \cdot 15 \bmod 33 = 9.$$

Число 9 Алиса передает Бобу по открытому каналу связи. Только Боб знает  $c_B = 7$ , поэтому он декодирует принятное сообщение, используя (2.31):

$$m' = 9^7 \bmod 33 = (9^2)^2 \cdot 9 \bmod 33 = 15^2 \cdot 15 \cdot 9 \bmod 33 = 15.$$

Таким образом, Боб расшифровал сообщение Алисы.  $\square$

Рассмотренная система невскрываема при больших  $P$  и  $Q$ , но обладает следующим недостатком:  $A$  передает сообщение  $B$ , используя открытую информацию абонента  $B$  (числа  $N_B$  и  $d_B$ ). Злоумышленник не может читать сообщения, предназначенные для  $B$ , однако он может передать сообщение к  $B$  от имени  $A$ . Избежать этого можно, используя более сложные протоколы, например, следующий.

$A$  хочет передать  $B$  сообщение  $m$ . Сначала  $A$  вычисляет число  $e = m^{c_A} \bmod N_A$ . Злоумышленник не может этого сделать, т.к.  $c_A$  секретно. Затем  $A$  вычисляет число  $f = e^{d_B} \bmod N_B$  и передает  $f$  к  $B$ .  $B$  получает  $f$  и вычисляет последовательно числа  $u = f^{c_B} \bmod N_B$  и  $w = u^{d_A} \bmod N_A$ .

В результате абонент  $B$  получает сообщение  $w = m$ . Как и в исходной схеме RSA, злоумышленник не может прочитать переданное сообщение, но здесь, в отличие от RSA, он не может также послать сообщение от имени  $A$  (поскольку не знает секретного  $c_A$ ).

Здесь мы встречаемся с новой ситуацией.  $B$  знает, что сообщение пришло от  $A$ , т.е.  $A$  как бы «подписал» его, зашифровав своим секретным  $c_A$ . Это пример так называемой *электронной* или *цифровой подписи*. Она — одно из широко используемых на практике изобретений современной криптографии и будет систематически изучаться в главе 4.

## Задачи и упражнения

- 2.1.** Привести результат выражений  $5, 16, 27, -4, -13, 3 + 8, 3 - 8, 3 \cdot 8, 3 \cdot 8 \cdot 5$ :
- по модулю 10,
  - по модулю 11.
- 2.2.** Вычислить, используя быстрые алгоритмы возведения в степень,  $2^8 \bmod 10, 3^7 \bmod 10, 7^{19} \bmod 100, 7^{57} \bmod 100$ .
- 2.3.** Разложить на простые множители числа 108, 77, 65, 30, 159.
- 2.4.** Определить, какие из пар чисел  $(25, 12), (25, 15), (13, 39), (40, 27)$  взаимно просты.
- 2.5.** Найти значения функции Эйлера  $\varphi(14), \varphi(20)$ .
- 2.6.** Используя свойства функции Эйлера, вычислить  $\varphi(53), \varphi(21), \varphi(159)$ .
- 2.7.** Используя теорему Ферма, вычислить  $3^{13} \bmod 13, 5^{22} \bmod 11, 3^{17} \bmod 5$ .
- 2.8.** Используя теорему Эйлера, вычислить  $3^9 \bmod 20, 2^{14} \bmod 21, 2^{107} \bmod 159$ .
- 2.9.** С помощью алгоритма Евклида найти  $\gcd(21, 12), \gcd(30, 12), \gcd(24, 40), \gcd(33, 16)$ .
- 2.10.** С помощью обобщенного алгоритма Евклида найти значения  $x$  и  $y$  в уравнениях
- $21x + 12y = \gcd(21, 12),$
  - $30x + 12y = \gcd(30, 12),$
  - $24x + 40y = \gcd(24, 40),$
  - $33x + 16y = \gcd(33, 16).$
- 2.11.** Вычислить  $3^{-1} \bmod 7, 5^{-1} \bmod 8, 3^{-1} \bmod 53, 10^{-1} \bmod 53$ .
- 2.12.** Выписать все простые числа, меньшие 100. Какие из них соответствуют виду  $p = 2q + 1$ , где  $q$  также простое?

- 2.13.** Найти все допустимые варианты выбора параметра  $g$  в системе Диффи–Хеллмана при  $p = 11$ .
- 2.14.** Вычислить секретные ключи  $Y_A$ ,  $Y_B$  и общий ключ  $Z_{AB}$  для системы Диффи–Хеллмана с параметрами:
- $p = 23$ ,  $g = 5$ ,  $X_A = 5$ ,  $X_B = 7$ ,
  - $p = 19$ ,  $g = 2$ ,  $X_A = 5$ ,  $X_B = 7$ ,
  - $p = 23$ ,  $g = 7$ ,  $X_A = 3$ ,  $X_B = 4$ ,
  - $p = 17$ ,  $g = 3$ ,  $X_A = 10$ ,  $X_B = 5$ ,
  - $p = 19$ ,  $g = 10$ ,  $X_A = 4$ ,  $X_B = 8$ .
- 2.15.** Для шифра Шамира с заданными параметрами  $p$ ,  $c_A$ ,  $c_B$  найти недостающие параметры и описать процесс передачи сообщения  $m$  от  $A$  к  $B$ :
- $p = 19$ ,  $c_A = 5$ ,  $c_B = 7$ ,  $m = 4$ ,
  - $p = 23$ ,  $c_A = 15$ ,  $c_B = 7$ ,  $m = 6$ ,
  - $p = 19$ ,  $c_A = 11$ ,  $c_B = 5$ ,  $m = 10$ ,
  - $p = 23$ ,  $c_A = 9$ ,  $c_B = 3$ ,  $m = 17$ ,
  - $p = 17$ ,  $c_A = 3$ ,  $c_B = 13$ ,  $m = 9$ .
- 2.16.** Для шифра Эль–Гамаля с заданными параметрами  $p$ ,  $g$ ,  $c_B$ ,  $k$  найти недостающие параметры и описать процесс передачи сообщения  $m$  пользователю  $B$ :
- $p = 19$ ,  $g = 2$ ,  $c_B = 5$ ,  $k = 7$ ,  $m = 5$ ,
  - $p = 23$ ,  $g = 5$ ,  $c_B = 8$ ,  $k = 10$ ,  $m = 10$ ,
  - $p = 19$ ,  $g = 2$ ,  $c_B = 11$ ,  $k = 4$ ,  $m = 10$ ,
  - $p = 23$ ,  $g = 7$ ,  $c_B = 3$ ,  $k = 15$ ,  $m = 5$ ,
  - $p = 17$ ,  $g = 3$ ,  $c_B = 10$ ,  $k = 5$ ,  $m = 10$ .
- 2.17.** В системе RSA с заданными параметрами  $P_A$ ,  $Q_A$ ,  $d_A$  найти недостающие параметры и описать процесс передачи сообщения  $m$  пользователю  $A$ :
- $P_A = 5$ ,  $Q_A = 11$ ,  $d_A = 3$ ,  $m = 12$ ,
  - $P_A = 5$ ,  $Q_A = 13$ ,  $d_A = 5$ ,  $m = 20$ ,

- в.  $P_A = 7, Q_A = 11, d_A = 7, m = 17,$
- г.  $P_A = 7, Q_A = 13, d_A = 5, m = 30,$
- д.  $P_A = 3, Q_A = 11, d_A = 3, m = 15.$

- 2.18. Пользователю системы RSA с параметрами  $N = 187, d = 3$  передано зашифрованное сообщение  $e = 100$ . Расшифровать это сообщение, взломав систему RSA пользователя.

## Темы лабораторных работ

- 2.19. Написать и отладить набор подпрограмм, реализующих базовые алгоритмы, используемые в изученных криптосистемах: возведение в степень по модулю ( $a^x \bmod m$ ), вычисление наибольшего общего делителя ( $\gcd(a, b)$ ), вычисление инверсии ( $x^{-1} \bmod m$ ).
- 2.20. Написать программу, реализующую систему Диффи–Хеллмана. Рекомендуемые значения параметров  $p = 30803, g = 2$ . Секретные ключи генерировать случайным образом.
- 2.21. Написать программу, реализующую шифр Шамира. В качестве простого модуля можно взять число  $p = 30803$ . Остальные параметры генерировать случайным образом.
- 2.22. Написать программу, реализующую шифр Эль–Гамаля. Рекомендуемые значения параметров  $p = 30803, g = 2$ . Секретные ключи и другие параметры генерировать случайным образом.
- 2.23. Написать программу, реализующую шифр RSA для передачи секретных сообщений в адрес абонентов  $A$  или  $B$ . Рекомендуемые значения параметров  $P_A = 131, Q_A = 227, P_B = 113, Q_B = 281, d_A = d_B = 3$ .

# Глава 3. МЕТОДЫ ВЗЛОМА ШИФРОВ, ОСНОВАННЫХ НА ДИСКРЕТНОМ ЛОГАРИФМИРОВАНИИ

## 3.1. Постановка задачи

Для построения надежной криптосистемы необходимо принимать во внимание те методы взлома, которые может применить злоумышленник, и выбирать параметры криптосистемы (в частности, длины чисел) так, чтобы сделать эти методы практически нереализуемыми. В данном параграфе мы рассмотрим два таких метода, для того чтобы дать читателю некоторое представление об этой «тайной» области.

Мы видели, что многие рассматриваемые шифры основываются на односторонней функции

$$y = a^x \pmod{p} \quad (3.1)$$

и знаем, что можно вычислить  $y$ , если даны  $a$  и  $x$ , затратив не более чем  $2 \log x$  операций (утверждение 2.1). Однако отыскание  $x$  по известным  $a$  и  $y$ , т.е. вычисление дискретного логарифма, — задача намного более трудная.

Как уже было показано при рассмотрении шифра Шамира (см. (2.23)), на основании теоремы Ферма при возведении в степень по простому модулю  $p$  показатели степени приводятся по модулю  $p-1$ . Поэтому нам достаточно рассматривать только показатели  $x$ , удовлетворяющие неравенству  $0 \leq x \leq p-1$ .

Обозначим через  $t_y$  число операций умножения, необходимых для вычисления  $y$  в (3.1) по  $a$  и  $x$ , и будем для краткости называть  $t_y$  временем вычисления. Время возведения в степень по алгоритмам из разд. 2.1 не больше  $2 \log x$ , причем  $x < p$ . Отсюда

$$t_y \leq 2 \log p \quad (3.2)$$

при любом показателе степени  $x$ .

Теперь перейдем к задаче отыскания  $x$  в (3.1) по данным  $a$  и  $y$ . Сначала оценим сложность прямого перебора. Для этого можно было бы сначала вычислить  $a^1$  и проверить, верно ли равенство  $a^1 = y$ . Если нет, то проверяем  $a^2 = y$ , если нет, то  $a^3 = y$  и т.д. до  $a^{p-1}$ . В среднем потребуется  $(p-1)/2$  раз умножать на  $a$  и проверять равенство. Таким образом, время прямого перебора

$$t_{\text{п.п.}} \approx p/2.$$

В описываемом ниже методе «шаг младенца, шаг великана» время отыскания  $x$  существенно меньше:

$$t_{\text{ш.м.ш.в.}} \approx 2 \cdot \sqrt{p},$$

а в методе исчисления порядка это время еще меньше:

$$t_{\text{и.п.}} \approx c_1 \cdot 2^{c_2 \sqrt{\log p \log \log p}},$$

где  $c_1, c_2$  — некоторые положительные константы.

Чтобы сделать сравнение более наглядным, выразим время вычисления через длину числа  $p$  в (3.1). Обозначим эту длину в битах через  $n$ . При вычислениях по модулю  $p$  имеем  $n \approx \log p$ . Поэтому порядок трудоемкости (в смысле количества операций) упомянутых алгоритмов будет следующий:

$$\begin{aligned} t_y &\approx n, \\ t_{\text{п.п.}} &\approx 2^{n-1}, \\ t_{\text{ш.м.ш.в.}} &\approx 2^{n/2}, \\ t_{\text{и.п.}} &\approx 2^{c_2 \sqrt{n \log n}}, \end{aligned}$$

где  $\approx$  означает «пропорционально».

Мы видим, что количество операций при возведении в степень растет линейно с ростом длины числа  $n$ , а время решения обратной задачи различными методами растет экспоненциально либо субэкспоненциально (для метода исчисления порядка). Вопрос о существовании более быстрых алгоритмов для вычисления дискретных логарифмов, как и для решения других обратных задач, возникающих в криptoанализе, остается открытым.

### 3.2. Метод «шаг младенца, шаг великана»

В открытой литературе этот метод был впервые описан Шенксом (Daniel Shanks); ссылки на него известны с 1973 года. Это был один из первых методов, который показал, что задача вычисления дискретного логарифма может быть решена значительно быстрее, чем методом перебора. Перейдем к описанию этого метода отыскания  $x$  в (3.1).

**Шаг 1.** Сначала берем два целых числа  $m$  и  $k$ , такие что

$$mk > p. \quad (3.3)$$

**Шаг 2.** Вычислим два ряда чисел

$$y, \quad ay, \quad a^2y, \quad \dots, \quad a^{m-1}y \pmod{p}; \quad (3.4)$$

$$a^m, \quad a^{2m}, \quad \dots, \quad a^{km} \pmod{p} \quad (3.5)$$

(все вычисления проводятся по модулю  $p$ ).

**Шаг 3.** Найдем такие  $i$  и  $j$ , для которых выполняется равенство

$$a^{im} = a^jy. \quad (3.6)$$

**Утверждение 3.1.** Число

$$x = im - j \quad (3.7)$$

является решением уравнения (3.1). Кроме того, целые числа  $i$ ,  $j$ , удовлетворяющие (3.6), существуют.

**Доказательство.** Справедливость (3.7) следует из приводимой ниже символической цепочки равенств, где все вычисления даны по модулю  $p$ , а деление соответствует умножению на обратный элемент:

$$a^x = a^{im-j} = \frac{a^{im}}{a^j} = \frac{a^{im}y}{a^jy} = \frac{a^{im}y}{a^{im}} = y.$$

Докажем теперь, что числа  $i$  и  $j$ , удовлетворяющие (3.6), существуют. Для этого сведем все числа вида (3.7) в таблицу 3.1.

Таблица 3.1. Распределение чисел вида  $im - j$ 

$i \downarrow$	$j \rightarrow$	0	1	2	...	$m - 1$
1		$m$	$m - 1$	$m - 2$	...	1
2		$2m$	$2m - 1$	$2m - 2$	...	$m + 1$
...		...	...	...	...	...
$k$		$km$	$km - 1$	$km - 2$	...	$(k - 1)m + 1$

Мы видим, что в таблице содержатся все числа от 1 до  $km$ . Значит, из (3.3) следует, что в таблице содержатся все числа от 1 до  $p$ . Таким образом, любой показатель степени  $x < p$  будет содержаться в таблице, т.е. число  $x$ , удовлетворяющее (3.1), может быть представлено в виде (3.7) и всегда найдется в таблице, поэтому уравнение (3.6) всегда имеет решение.  $\square$

**Пример 3.1.** Найдем решение уравнения  $2^x \bmod 23 = 9$ , используя метод «шаг младенца, шаг великана».

Выберем  $m$ ,  $k$ . Пусть  $m = 6$ ,  $k = 4$ . Мы видим, что (3.3) выполняется. Вычислим числа (3.4), (3.5):

$$(3.4) : \quad 9, 18, 13, 3, 6, 12;$$

$$(3.5) : \quad 18.$$

Дальнейшие вычисления не проводим, т.к. уже нашлись одинаковые числа в (3.4) и (3.5) при  $i = 1$ ,  $j = 1$ . По (3.7) получаем

$$x = 1 \cdot 6 - 1 = 5.$$

Проверим:  $2^5 \bmod 23 = 9$ . Действительно,  $x = 5$  есть решение.  $\square$

Объясним происхождение названия рассмотренного метода. Мы знаем, что в криптографии  $p$  — большое простое число, значит  $m$  и  $k$  тоже большие. В ряду (3.4) степень увеличивается на 1 (шаг младенца), а в ряду (3.5) степень увеличивается на  $m$  (шаг великана).

Оценим сложность этого метода.

**Утверждение 3.2.** Время вычислений по данному методу при больших  $p$  удовлетворяет неравенству

$$t_{\text{ш.м.ш.в.}} \leq \text{const} \cdot \sqrt{p} \log^2 p. \quad (3.8)$$

(Здесь речь идет о полном времени вычислений, а не о числе умножений).

**Доказательство.** Мы можем взять

$$k = m = \lfloor \sqrt{p} \rfloor + 1, \quad (3.9)$$

так что, очевидно, (3.3) выполняется. Тогда в (3.4) и (3.5) потребуется не более  $2\sqrt{p}$  операций умножения. Мы знаем, что для «обычных» («школьных») методов умножения и деления время вычисления результата для двух  $r$ -значных чисел пропорционально  $r^2$ . У нас все числа берутся из множества  $\{1, \dots, p\}$ , значит,  $r \leq \log p$ , и время вычисления пропорционально  $\log^2 p$ . Отсюда мы сразу получаем время, затраченное на вычисление рядов (3.4) и (3.5). Однако мы не учли все этапы алгоритма. Мы не учли время, требуемое для нахождения равных чисел в этих рядах. При больших  $k$  и  $m$  это далеко не простая задача. Она может быть решена следующим образом: сначала каждому числу припишем его номер в ряду и еще один бит, в котором указана принадлежность к ряду (3.4) или (3.5), затем преобразуем обе последовательности в список и отсортируем (упорядочим по величине). Длина общего ряда равна  $k + m \approx 2\sqrt{p}$ . Для лучших методов сортировки требуется  $S \log S$  операций сравнения, где  $S$  — число элементов в списке (см., например, [1]). В нашем случае  $S = 2\sqrt{p}$  и, следовательно, требуется  $2\sqrt{p} \log(2\sqrt{p}) \approx \sqrt{p} \log p$  операций сравнения над словами длины  $\log p$  бит, т.е. всего требуется порядка  $\sqrt{p} \log^2 p$  операций. После сортировки объединенного ряда его надо просмотреть и найти два равных числа из разных рядов (3.4), (3.5), используя битовый признак. Таким образом, суммируя время вычисления на всех этапах, получаем (3.8).  $\square$

### 3.3. Алгоритм исчисления порядка

Основные идеи *алгоритма исчисления порядка* (index-calculus algorithm) были известны в теории чисел еще с 20-х годов XX века. Однако только в 1979 году Адлеман, один из создателей RSA, указал на этот алгоритм как на средство решения уравнения (3.1) и исследовал его трудоемкость. В настоящее время алгоритм исчисления порядка и его улучшенные варианты дают наиболее быстрый способ вычисления дискретных логарифмов в уравнениях типа (3.1).

Для удобства описания алгоритма введем следующее понятие.

**Определение 3.1.** Число  $n$  называется  $p$ -гладким, если оно разлагается только на простые множители, меньшие либо равные  $p$ .

Пример 3.2. Числа 15, 36, 45, 270, 2025 являются 5-гладкими (в их разложении участвуют только множители 2, 3, и 5).  $\square$

Перейдем непосредственно к описанию алгоритма.

**Шаг 1.** Формируем множество базовых множителей

$$S = \{p_1, p_2, \dots, p_t\},$$

состоящее из первых  $t$  простых чисел (замечание о выборе значения  $t$  будет дано ниже).

**Шаг 2.** Задавая последовательно значения  $k = 1, 2, 3, \dots$ , находим  $t + \epsilon$  ( $\epsilon$  — небольшое целое число, см. ниже)  $p_t$ -гладких чисел вида  $a^k \bmod p$ , проверяя гладкость путем деления на элементы множества  $S$ . Каждое из найденных  $p_t$ -гладких чисел записывается через произведение базовых множителей:

$$a^k \bmod p = \prod_{i=1}^t p_i^{c_i}, \quad c_i \geq 0, \quad (3.10)$$

(для каждого значения  $k$  получаем свой набор чисел  $c_i$ ).

**Шаг 3.** Переходим к логарифмам в (3.10):

$$k = \sum_{i=1}^t c_i \log_a p_i \quad (3.11)$$

для каждого  $p_t$ -гладкого числа, найденного на шаге 2. Мы получили систему из  $t + \epsilon$  уравнений вида (3.11) с  $t$  неизвестными. В качестве неизвестных здесь выступают величины  $\log_a p_i$ , при этом число уравнений на  $\epsilon$  больше числа неизвестных, что повышает вероятность получения решения системы в случае, если некоторые из уравнений окажутся линейно зависимыми. Решаем систему методами линейной алгебры, проводя все вычисления по модулю  $p - 1$  (напомним, что показатели степени, а следовательно и логарифмы, приводятся по модулю  $p - 1$ ). В результате получаем значения логарифмов чисел из множества  $S$ :  $\log_a p_1, \log_a p_2, \dots, \log_a p_t$ .

**Шаг 4.** Случайным образом выбирая  $r$ , находим  $p_t$ -гладкое число вида  $(y \cdot a^r)$ :

$$y \cdot a^r \bmod p = \prod_{i=1}^t p_i^{e_i}, \quad e_i \geq 0. \quad (3.12)$$

**Шаг 5.** Логарифмируя (3.12), получаем конечный результат

$$x = \log_a y = \left( \sum_{i=1}^t e_i \log_a p_i - r \right) \bmod (p-1) \quad (3.13)$$

(величина  $r$  вычитается из всей суммы, а не из каждого слагаемого).

Справедливость описанного метода довольно очевидна из построения алгоритма, а его эффективность связана со следующим наблюдением. Если мы выбираем случайно наугад число из бесконечного множества целых чисел, то с вероятностью  $1/2$  оно делится на 2, с вероятностью  $1/3$  — на 3, с вероятностью  $1/5$  — на 5 и т.д. Поэтому мы можем ожидать, что в промежутке от 1 до  $p-1$  существует достаточно много чисел, в разложении которых участвуют только маленькие простые множители из множества  $S$ . Именно такие числа отыскиваются на шагах 2 и 4 алгоритма. Чем больше  $t$ , т.е. количество простых множителей в  $S$ , тем меньше неудач при поиске гладких чисел происходит на шагах 2 и 4, т.е. эти шаги выполняются быстрее. Однако при больших  $t$  резко увеличивается трудоемкость шага 3, когда приходится решать систему из  $t + \epsilon$  уравнений. Нахождение значения  $t$ , дающего минимальное общее время вычислений, обычно может быть выполнено с использованием численных методов. Аналитические выражения получить довольно трудно. Параметр  $\epsilon$  принимается равным небольшому целому числу для того, чтобы увеличить вероятность существования решения системы уравнений на шаге 3. Дело в том, что полученная система может содержать линейно зависимые уравнения (как это будет показано в приводимом ниже примере). Считается, что при больших  $p$  значение  $\epsilon$  порядка 10 гарантирует существование единственного решения системы с высокой вероятностью (см. [68]). Если же все-таки полученная система имеет бесконечно много решений, то необходимо вернуться к шагу 2 и использовать другие значения  $k$ .

Адлеман показал, что при оптимальном значении  $t$  для трудоемкости алгоритма имеем

$$t_{\text{и.п.}} < c_1 \cdot 2^{(c_2 + o(1))\sqrt{\log p \log \log p}},$$

где  $c_1, c_2$  — некоторые положительные константы.

Пример 3.3. Решим с помощью алгоритма исчисления порядка уравнение

$$37 = 10^x \pmod{47}. \quad (3.14)$$

Имеем  $y = 37$ ,  $a = 10$ ,  $p = 47$ . Возьмем множество базовых множителей  $S = \{2, 3, 5\}$ ,  $t = 3$ , и примем  $\epsilon = 1$ , т.е. будем строить систему из четырех уравнений. Обозначим логарифмы чисел из  $S$  через  $u_1, u_2, u_3$  соответственно, например,  $u_3 = \log_{10} 5 \pmod{47}$ . Мы выполнили первый шаг алгоритма, перейдем ко второму.

Проведем поиск четырех 5-гладких чисел:

$$\begin{aligned} 10^1 \pmod{47} &= 10 = 2 \cdot 5, & \checkmark \\ 10^2 \pmod{47} &= 6 = 2 \cdot 3, & \checkmark \\ 10^3 \pmod{47} &= 13 = 13, \\ 10^4 \pmod{47} &= 36 = 2 \cdot 2 \cdot 3 \cdot 3, & \checkmark \\ 10^5 \pmod{47} &= 31 = 31, \\ 10^6 \pmod{47} &= 28 = 2 \cdot 2 \cdot 7, \\ 10^7 \pmod{47} &= 45 = 3 \cdot 3 \cdot 5. & \checkmark \end{aligned}$$

Мы нашли четыре 5-гладких числа, соответствующих степеням 1, 2, 4 и 7.

Начинаем третий шаг алгоритма. Переходим к логарифмам и составим систему уравнений из равенств, отмеченных на предыдущем шаге символом  $\checkmark$ :

$$1 = u_1 + u_3, \quad (3.15)$$

$$2 = u_1 + u_2, \quad (3.16)$$

$$4 = 2u_1 + 2u_2, \quad (3.17)$$

$$7 = 2u_2 + u_3. \quad (3.18)$$

Мы видим, что в полученной системе уравнения (3.16) и (3.17) линейно зависимы, так что мы не зря нашли четвертое гладкое число. Чтобы решить систему, вычтем (3.15) из (3.16). Получим

$$1 = u_2 - u_3. \quad (3.19)$$

Прибавим (3.19) к (3.18). Получим

$$8 = 3u_2. \quad (3.20)$$

Из (3.20) непосредственно находим  $u_2$ :

$$u_2 = (8/3) \bmod 46 = 8 \cdot 3^{-1} \bmod 46 = 8 \cdot 31 \bmod 46 = 18.$$

Мы можем сделать проверку, вычислив  $10^{18} \bmod 47 = 3$ , таким образом,  $u_2$  — действительно логарифм числа 3. Теперь из (3.19) находим  $u_3$ :

$$u_3 = u_2 - 1 = 18 - 1 = 17$$

(действительно,  $10^{17} \bmod 47 = 5$ ). Наконец, из (3.16) находим  $u_1$ :

$$u_1 = 2 - u_2 = (2 - 18) \bmod 46 = -16 \bmod 46 = 30$$

( $10^{30} \bmod 47 = 2$ ).

Итак, теперь мы знаем логарифмы чисел из  $S$ . Самый трудоемкий этап алгоритма позади. Переходим к четвертому шагу. Начнем с  $k = 3$ :

$$37 \cdot 10^3 \bmod 47 = 37 \cdot 13 \bmod 47 = 11,$$

$$37 \cdot 10^4 \bmod 47 = 37 \cdot 36 \bmod 47 = 16 = 2 \cdot 2 \cdot 2 \cdot 2. \quad \checkmark$$

Переходим в последнем равенстве к логарифмам (это пятый шаг) и получаем конечный результат:

$$\log_{10} 37 = 4 \log_{10} 2 - 4 = (4 \cdot 30 - 4) \bmod 46 = 24.$$

Мы нашли решение уравнения (3.14)  $x = 24$ . Можем сделать проверку:  $10^{24} \bmod 47 = 37$ .  $\square$

Самым быстрым на данное время считается вариант рассмотренного алгоритма исчисления порядка, называемый Number Field Sieve. Этот метод использует тонкие алгебраические конструкции и довольно сложен для описания. Его трудоемкость дается оценкой

$$t_{\text{n.f.s.}} < c_1 \cdot 2^{(c_2 + o(1)) \sqrt[3]{\log p (\log \log p)^2}}, \quad (3.21)$$

где  $c_1, c_2$  — некоторые положительные константы. Именно этот метод диктует сегодня условия для выбора длин модулей крипто-систем, стойкость которых основана на трудности вычисления дискретных логарифмов (из рассмотренных во второй главе это система

Диффи-Хеллмана, шифры Шамира и Эль-Гамаля). Для достижения долговременной стойкости этих крипtosистем рекомендуется брать модули длиной не менее 1024 бит (по данным на 2017 год).

В заключение отметим, что в нашей книге мы не рассматриваем методы взлома крипtosистем, основанных на факторизации чисел (таких как RSA). Дело в том, что описание современных алгоритмов разложения числа на множители потребовало бы введения дополнительных понятий и алгоритмов из теории чисел, нигде больше в книге не используемых. Однако скажем, что по состоянию на 2010 год самые быстрые методы разложения чисел на множители характеризуются такой же оценкой времени, которую дает выражение (3.21). Как следствие, для обеспечения стойкости системы RSA длина модуля должна также быть не менее 1024 бит (т.е. простые числа, дающие в произведении модуль RSA, должны быть длиной минимум по 512 бит).

## Задачи и упражнения

**3.1.** Используя метод “шаг младенца, шаг великана”, решить следующие уравнения:

- а.  $2^x \bmod 29 = 21$ ,
- б.  $3^x \bmod 31 = 25$ ,
- в.  $2^x \bmod 37 = 12$ ,
- г.  $6^x \bmod 41 = 21$ ,
- д.  $3^x \bmod 43 = 11$ .

**3.2.** Используя алгоритм исчисления порядка, решить следующие уравнения:

- а.  $2^x \bmod 53 = 24$ ,
- б.  $2^x \bmod 59 = 13$ ,
- в.  $2^x \bmod 61 = 45$ ,
- г.  $2^x \bmod 67 = 41$ ,
- д.  $7^x \bmod 71 = 41$ .

## Темы лабораторных работ

**3.3.** Выполнить программную реализацию метода “шаг младенца, шаг великаны” и алгоритма исчисления порядка и решить с помощью компьютера следующие уравнения:

- а.  $2^x \bmod 30203 = 24322$ ,
- б.  $2^x \bmod 30323 = 21740$ ,
- в.  $2^x \bmod 30539 = 28620$ ,
- г.  $2^x \bmod 30803 = 16190$ ,
- д.  $5^x \bmod 31607 = 30994$

(при недостатке времени достаточно реализовать программно только отдельные шаги алгоритмов).

# Глава 4. ЭЛЕКТРОННАЯ, ИЛИ ЦИФРОВАЯ ПОДПИСЬ

## 4.1. Электронная подпись RSA

После появления криптографии с открытым ключом произошла настоящая революция в современных компьютерных и сетевых технологиях. Появилась возможность решать задачи, которые ранее считались неразрешимыми, а теперь находят широкое применение на практике. В настоящее время при помощи этих технологий ежедневно проводятся расчеты и совершаются сделки на многие миллиарды долларов, рублей, евро и т.п. Одним из важных элементов этих технологий является электронная, или цифровая подпись. Во многих странах и, в частности, в России введены стандарты на электронную (цифровую) подпись, а само это понятие введено в гражданское законодательство. Термин «электронная подпись» более привычен для России, хотя в последнее время все чаще используется принятый в других странах (прежде всего, в США) термин «цифровая подпись», что отражено и в новых российских стандартах. Оба термина означают одно и то же.

Прежде чем начать рассмотрение криптографической цифровой подписи, сформулируем три свойства, которым (в идеале) должна удовлетворять любая, в частности, обычная рукописная подпись:

1. Подписать документ может только «законный» владелец подписи (и, следовательно, никто не может подделать подпись).
2. Автор подписи не может от нее отказаться.
3. В случае возникновения спора возможно участие третьих лиц (например, суда) для установления подлинности подписи.

Разумеется, цифровая (электронная) подпись также должна обладать всеми этими свойствами, однако лица, подписывающие документы и проверяющие их подлинность, могут находиться за тысячи

километров друг от друга и взаимодействовать только через компьютерную сеть.

Кроме обычной подписи, в реальной жизни используется так называемая нотариальная подпись, когда специально выделяемое лицо (нотариус) заверяет документы своей подписью и печатью, так что любое другое лицо может удостовериться в их подлинности. Аналог этой подписи также востребован в киберпространстве. Электронная нотариальная подпись реализуется точно так же, как и подпись любого другого лица.

В этом разделе мы рассмотрим электронную подпись, базирующуюся на схеме RSA.

Если Алиса планирует подписывать документы, то она должна вначале выбрать параметры RSA точно так же, как это описано в разд. 2.6. Для этого Алиса выбирает два больших простых числа  $P$  и  $Q$ , вычисляет  $N = PQ$  и  $\phi = (P - 1)(Q - 1)$ . Затем она выбирает число  $d$ , взаимно простое с  $\phi$ , и вычисляет  $c = d^{-1} \bmod \phi$ . Наконец, она публикует числа  $N$  и  $d$ , например, помещает их на своем сайте, ассоциировав со своим именем, и хранит в секрете число  $c$  (остальные числа  $P$ ,  $Q$  и  $\phi$  можно забыть, они больше не потребуются). Теперь Алиса готова ставить свои подписи на документах или сообщениях.

Пусть Алиса хочет подписать сообщение  $\bar{m} = m_1 \dots m_n$ . Тогда вначале она вычисляет так называемую хеш-функцию

$$y = h(m_1 \dots m_n),$$

которая ставит в соответствие сообщению  $\bar{m}$  число  $y$ . Предполагается, что алгоритм вычисления хеш-функции всем известен. Но мы пока не будем останавливаться на свойствах и способах вычисления хеш-функции, т.к. этот вопрос будет более подробно рассмотрен в главе 8. Отметим только наиболее важное для нас свойство: практически невозможно изменить основной текст  $m_1 \dots m_n$ , не изменив  $y$ . Поэтому на следующем шаге Алисе достаточно снабдить подписью только число  $y$ , и эта подпись будет относиться ко всему сообщению  $\bar{m}$ .

Алиса вычисляет число

$$s = y^c \bmod N, \tag{4.1}$$

т.е. она возводит число  $y$  в свою секретную степень. Число  $s$  это и есть цифровая подпись. Она просто добавляется к сообщению  $\bar{m}$ , и

тем самым Алиса имеет сформированное подписанное сообщение

$$\langle \tilde{m}, s \rangle. \quad (4.2)$$

Теперь каждый, кто знает открытые параметры Алисы, ассоциированные с ее именем, т.е. числа  $N$  и  $d$ , может проверить подлинность ее подписи. Для этого необходимо, взяв подписанное сообщение (4.2), вычислить значение хеш-функции  $h(\tilde{m})$ , число

$$w = s^d \bmod N \quad (4.3)$$

и проверить выполнение равенства  $w = h(\tilde{m})$ .

**Утверждение 4.1.** *Если подпись подлинная, то  $w = h(\tilde{m})$ .*

**Доказательство.** Из (4.3), (4.1) и свойств схемы RSA (см. разд. 2.6) следует

$$w = s^d \bmod N = y^{cd} \bmod N = y = h(\tilde{m}).$$

□

**Утверждение 4.2.** *Описанная электронная подпись удовлетворяет всем требованиям, предъявляемым к подписи.*

**Доказательство.** Проверим первое свойство подписи. Никто не может разложить число  $N$  на простые множители (при больших  $N$  порядка 1024 бит по состоянию на 2010 год эта задача практически неразрешима). Поэтому, зная  $N$  и  $d$  невозможно найти  $s$ . Действительно, чтобы вычислить  $s = d^{-1} \bmod \phi$ , нужно знать  $\phi = (P - 1)(Q - 1)$ , а для этого нужно знать простые множители  $P$  и  $Q$ . Таким образом, первое свойство выполнено — никто, кроме Алисы, не может знать число  $s$  и поэтому не может подписать сообщение.

Второе свойство выполнено вследствие первого. Автор подписи не может от нее отказаться, т.к. никто другой не может «сфабриковать» подпись от его имени.

Третье свойство также очевидно — в случае спора заинтересованная сторона может предъявить судье все вычисления для их проверки и выяснения истины. □

**Пример 4.1.** Пусть  $P = 5$ ,  $Q = 11$ . Тогда  $N = 5 \cdot 11 = 55$ ,  $\phi = 4 \cdot 10 = 40$ . Пусть  $d = 3$ . Такой выбор  $d$  возможен,

т.к.  $\gcd(40, 3) = 1$ . Параметр  $c = 3^{-1} \pmod{40}$  вычисляем с помощью обобщенного алгоритма Евклида (см. разд. 2.3),  $c = 27$ .

Пусть, например, Алиса хочет подписать сообщение  $\bar{m} = abbbaa$ , для которого значение хеш-функции равно, скажем, 13:

$$y = h(abbbaa) = 13.$$

В этом случае Алиса вычисляет по (4.1)

$$s = 13^{27} \pmod{55} = 7$$

и формирует подписанное сообщение

$$\langle abbbaa, 7 \rangle.$$

Теперь тот, кто знает открытые ключи Алисы  $N = 55$  и  $d = 3$ , может проверить подлинность подписи. Получив подписанное сообщение, он заново вычисляет значение хеш-функции

$$h(abbbaa) = 13$$

(если содержание сообщения не изменено, то значение хеш-функции совпадет с тем, которое вычисляла Алиса) и вычисляет по (4.3)

$$w = 7^3 \pmod{55} = 13.$$

Значения  $w$  и хеш-функции совпали, значит, подпись верна.  $\square$

**З а м е ч а н и е.** Обратим внимание на то, что одна и та же схема RSA, сгенерированная Алисой, может использоваться для решения двух задач. Во-первых, Алиса может подписывать сообщения, как это было показано в данном разделе, используя свой *секретный ключ*  $c$ . Во-вторых, кто угодно может послать Алисе зашифрованное сообщение (число), расшифровать которое, как это было показано в разд. 2.6, сможет только она, используя для шифрования ее *открытый ключ*  $d$ .

## 4.2. Электронная подпись на базе шифра Эль-Гамаля

В предыдущем разделе была описана схема электронной подписи, необходимые свойства которой определяются сложностью решения задачи разложения числа на множители. В этом разделе мы опишем

вариант подписи, основанный на задаче дискретного логарифмирования.

Пусть, как и выше, Алиса собирается подписывать документы. Алиса выбирает большое простое число  $p$  и число  $g$ , такие что различные степени  $g$  суть различные числа по модулю  $p$  (см. разд. 2.2). Эти числа передаются или хранятся в открытом виде и могут быть общими для целой группы пользователей. Алиса выбирает случайное число  $x$ ,  $1 < x < p - 1$ , которое она держит в секрете. Это ее секретный ключ, только она его знает. Затем она вычисляет число

$$y = g^x \bmod p. \quad (4.4)$$

Это число  $y$  Алиса публикует в качестве своего открытого ключа. Заметим, что при больших  $p$ , зная  $y$ , невозможно найти  $x$  (это задача дискретного логарифмирования).

Теперь Алиса может подписывать сообщения. Допустим, она хочет подписать сообщение  $\bar{m} = m_1 \dots m_n$ . Опишем последовательность действий для построения подписи.

Вначале Алиса вычисляет значение хеш-функции  $h = h(\bar{m})$ , которое должно удовлетворять неравенству  $1 < h < p$ . Затем Алиса выбирает случайно число  $k$  ( $1 < k < p - 1$ ), взаимно простое с  $p - 1$ , и вычисляет число

$$r = g^k \bmod p. \quad (4.5)$$

Далее Алиса вычисляет числа

$$u = (h - xr) \bmod (p - 1), \quad (4.6)$$

$$s = k^{-1}u \bmod (p - 1). \quad (4.7)$$

Под  $k^{-1}$  в (4.7) подразумевается число, удовлетворяющее уравнению

$$k^{-1}k \bmod (p - 1) = 1. \quad (4.8)$$

Такое  $k^{-1}$  существует, т.к.  $k$  и  $p - 1$  взаимно просты, и может быть найдено по обобщенному алгоритму Евклида. Наконец, Алиса формирует подписанное сообщение

$$\langle \bar{m}; r, s \rangle. \quad (4.9)$$

Получатель подписанного сообщения (4.9), прежде всего, заново вычисляет значение хеш-функции  $h = h(\bar{m})$ . Затем он проверяет подпись, используя равенство

$$y^r r^s = g^h \bmod p. \quad (4.10)$$

**Утверждение 4.3.** Если подпись верна, то условие (4.10) выполняется.

**Доказательство.** Действительно,

$$y^r r^s = (g^x)^r (g^k)^s = g^{xr} g^{k(k^{-1}(h-xr))} = g^{xr} g^h g^{-xr} = g^h \bmod p.$$

(Здесь первое равенство следует из (4.4) и (4.5), второе из (4.7).)  $\square$

**Утверждение 4.4.** Описанная электронная подпись удовлетворяет всем требованиям, предъявляемым к подписи.

**Доказательство.** Проверим первое свойство подписи (никто не может подделать подпись, другими словами, никто кроме Алисы не может подписать сообщение ее подписью). Действительно, из (4.6) мы видим, что при формировании подписи используется секретное число  $x$ . Более того, сомножитель  $xr$ , используемый при формировании подписи в (4.6), меняется от сообщения к сообщению (т.к.  $k$  выбирается случайно, то и  $r$  случайно).

По той же самой причине Алиса не сможет отказаться от своей подписи, т.к. никто кроме нее не знает  $x$ , т.е. выполняется второе свойство подписи.

Понятно также, что в случае возникновения конфликта между Алисой и Бобом, они могут обратиться к третьим лицам для выявления истины. Судья может проверить все вычисления, если ему предъявят числа  $x$ ,  $\bar{m}$  и  $r$ .  $\square$

**Пример 4.2.** Пусть общие параметры для некоторого сообщества пользователей  $p = 23$ ,  $g = 5$ . Алиса выбирает свой секретный ключ  $x = 7$  и вычисляет открытый ключ  $y$  по (4.4):

$$y = 5^7 \bmod 23 = 17.$$

Пусть Алиса создала документ  $\bar{m} = baaaab$  и хочет его подписать.

Перейдем к вычислению подписи по алгоритму. Прежде всего она вычисляет хеш-функцию, пусть ее значение  $h(\bar{m}) = 3$ . Затем Алиса генерирует случайное число  $k$ , например,  $k = 5$ . Вычисления по (4.5), (4.6) дают

$$r = 5^5 \bmod 23 = 20,$$

$$u = (3 - 7 \cdot 20) \bmod 22 = 17.$$

Далее Алиса находит  $k^{-1} \bmod 22$ :

$$k^{-1} \bmod 22 = 5^{-1} \bmod 22 = 9.$$

Вычисления по (4.7) дают

$$s = 9 \cdot 17 \bmod 22 = 21.$$

Наконец, Алиса формирует подписанное сообщение в виде (4.9):

$$\langle baaaab, 20, 21 \rangle.$$

Подписанное сообщение передается, Боб его получает и проверяет подлинность подписи. Вначале он вычисляет значение хеш-функции

$$h(baaaab) = 3,$$

затем вычисляет левую часть (4.10)

$$17^{20} \cdot 20^{21} \bmod 23 = 16 \cdot 15 \bmod 23 = 10$$

и после этого правую часть (4.10)

$$5^3 \bmod 23 = 10.$$

Боб делает вывод, что подпись верна. □

Рассмотренный метод электронной подписи сложнее, чем RSA, а его стойкость базируется на другой, нежели в RSA, односторонней функции. Это важно для криптографии, т.к. в случае дискредитации одного метода можно использовать другой. Кроме того, на основе подписи Эль-Гамаля может быть построен более эффективный алгоритм, в котором время вычислений значительно сокращается за счет использования «коротких» показателей степени. Такой алгоритм представлен в следующем разделе.

#### 4.3. Стандарты на электронную (цифровую) подпись

Во многих странах сегодня существуют стандарты на электронную (цифровую) подпись. В этом разделе мы опишем российский государственный стандарт ГОСТ Р 34.10-94 [6] и стандарт США FIPS

186 [57]. Российский стандарт, как следует из его обозначения, был принят в 1994 году, американский — в 1991 (последняя редакция — в 2013 году). В основе обоих стандартов лежит по сути один и тот же алгоритм, называемый DSA (Digital Signature Algorithm) и являющийся вариацией подписи Эль-Гамаля. Мы подробно рассмотрим российскую версию алгоритма, а затем укажем на отличия американской версии. Отметим, что сегодня действуют новые стандарты на цифровую подпись, в которых используются эллиптические кривые (глава 6). Однако, как мы увидим, для новых стандартов важны алгоритмы, которые мы описываем в данном разделе.

Вначале для некоторого сообщества пользователей выбираются общие несекретные параметры. Прежде всего необходимо найти два простых числа,  $q$  длиной 256 бит и  $p$  длиной 1024 бита, между которыми выполняется соотношение

$$p = bq + 1 \quad (4.11)$$

для некоторого целого  $b$ . Старшие биты в  $p$  и  $q$  должны быть равны единице. Затем выбирается число  $a > 1$ , такое что

$$a^q \bmod p = 1. \quad (4.12)$$

В результате получаем три общих параметра —  $p$ ,  $q$  и  $a$ .

**З а м е ч а н и е.** Равенство (4.12) означает, что при возведении  $a$  в степени по модулю  $p$  показатели приводятся по модулю  $q$ , т.е.  $a^b \bmod p = a^{b \bmod q} \bmod p$  (мы уже проводили обоснование подобного феномена при доказательстве утверждения 2.10 на стр. 29). Такое приведение будет постоянно выполнятся при генерации и проверке подписи, в результате чего длина показателей степени в рамках рассматриваемого алгоритма никогда не будет превышать 256 бит, что упрощает вычисления.

Далее, каждый пользователь выбирает случайно число  $x$ , удовлетворяющее неравенству  $0 < x < q$ , и вычисляет

$$y = a^x \bmod p. \quad (4.13)$$

Число  $x$  будет секретным ключом пользователя, а число  $y$  — открытым ключом. Предполагается, что открытые ключи всех пользователей указываются в некотором несекретном, но «сертифицированном» справочнике, который должен быть у всех, кто собирается проверять подписи. Отметим, что в настоящее время найти  $x$  по  $y$  практически невозможно при указанной выше длине модуля  $p$ .

На этом этапе выбора параметров заканчивается, и мы готовы к тому, чтобы формировать и проверять подписи.

Пусть имеется сообщение  $\bar{m}$ , которое необходимо подписать. Генерация подписи выполняется следующим образом:

1. Вычисляем значение хеш-функции  $h = h(\bar{m})$  для сообщения  $\bar{m}$ , значение хеш-функции должно лежать в пределах  $0 < h < q$  (в российском варианте хеш-функция определяется ГОСТом Р 34.11-94 [7]).
2. Формируем случайное число  $k$ ,  $0 < k < q$ .
3. Вычисляем  $r = (a^k \bmod p) \bmod q$ . Если оказывается так, что  $r = 0$ , то возвращаемся к шагу 2.
4. Вычисляем  $s = (kh + xr) \bmod q$ . Если  $s = 0$ , то возвращаемся к шагу 2.
5. Получаем подписанное сообщение  $\langle \bar{m}; r, s \rangle$ .

Для проверки подписи делаем следующее.

1. Вычисляем хеш-функцию для сообщения  $h = h(\bar{m})$ .
2. Проверяем выполнение неравенств  $0 < r < q$ ,  $0 < s < q$ .
3. Вычисляем  $u_1 = s \cdot h^{-1} \bmod q$ ,  $u_2 = -r \cdot h^{-1} \bmod q$ .
4. Вычисляем  $v = (a^{u_1} y^{u_2} \bmod p) \bmod q$ .
5. Проверяем выполнение равенства  $v = r$ .

Если хотя бы одна из проверок на шагах 2 и 5 не дает нужного результата, то подпись считается недействительной. Если же все проверки удачны, то подпись считается подлинной.

**Утверждение 4.5.** *Если подпись к сообщению была сформирована законно, т.е. обладателем секретного ключа  $x$ , то  $v = r$ .*

**Доказательство.** Запишем следующую цепочку равенств, которая следует непосредственно из описания метода (напомним, что

показатели степени приводятся по модулю  $q$ ):

$$\begin{aligned}
 v &= \left( a^{sh^{-1}} y^{-rh^{-1}} \pmod{p} \right) \pmod{q} = \\
 &= \left( a^{(kh+xr)h^{-1}} a^{-xrh^{-1}} \pmod{p} \right) \pmod{q} = \\
 &= \left( a^{k+xrh^{-1}-xrh^{-1}} \pmod{p} \right) \pmod{q} = \\
 &= (a^k \pmod{p}) \pmod{q} = r.
 \end{aligned}$$

□

**Замечание.** Чтобы найти параметр  $a$ , удовлетворяющий (4.12), рекомендуется использовать следующий метод. Берем случайное число  $g > 1$  и вычисляем

$$a = g^{(p-1)/q} \pmod{p}. \quad (4.14)$$

Если  $a > 1$ , то это то, что нам нужно. Действительно, на основании (4.14) и теоремы Ферма имеем

$$a^q \pmod{p} = g^{((p-1)/q)q} \pmod{p} = g^{p-1} \pmod{p} = 1,$$

т.е. выполняется равенство (4.12). Если при вычислении по (4.14) мы получаем  $a = 1$  (крайне маловероятный случай), то нужно просто взять другое число  $g$ .

**Пример 4.3.** Выберем общие несекретные параметры

$$q = 11, \quad p = 6q + 1 = 67,$$

возьмем  $g = 10$  и вычислим

$$a = 10^6 \pmod{67} = 25.$$

Выберем секретный ключ  $x = 6$  и вычислим открытый ключ

$$y = 25^6 \pmod{67} = 62.$$

Сформируем подпись для сообщения  $\bar{m} = baabab$ . Пусть для хеш-функции этого сообщения  $h(\bar{m}) = 3$ . Возьмем случайно число  $k = 8$ . Вычислим

$$\begin{aligned}
 r &= (25^8 \pmod{67}) \pmod{11} = 24 \pmod{11} = 2, \\
 s &= (8 \cdot 3 + 6 \cdot 2) \pmod{11} = 36 \pmod{11} = 3.
 \end{aligned}$$

Получаем подписанное сообщение

$$\langle baaaab; 2, 3 \rangle.$$

Теперь выполним проверку подписи. Если сообщение не изменено, то  $h = 3$ . Вычислим

$$h^{-1} = 3^{-1} \bmod 11 = 4,$$

$$u_1 = 3 \cdot 4 \bmod 11 = 1,$$

$$u_2 = -2 \cdot 4 \bmod 11 = -8 \bmod 11 = 3,$$

$$v = (25^1 \cdot 62^3 \bmod 67) \bmod 11 =$$

$$= (25 \cdot 9 \bmod 67) \bmod 11 = 24 \bmod 11 = 2.$$

Мы видим, что  $v = r$ , значит подпись верна.  $\square$

Теперь остановимся на отличиях американского стандарта от российского. Они сводятся к следующему.

1. Длины чисел  $q, p$  выбираются из ряда  $(160, 1024), (224, 2048), (256, 2048)$  и  $(256, 3072)$  бит соответственно.
2. В качестве хеш-функции используются алгоритмы SHA-1 или SHA-2 [55].
3. При генерации подписи на шаге 4 параметр  $s$  вычисляется по формуле  $s = k^{-1}(h + xr) \bmod q$ .
4. При проверке подписи на шаге 3  $u_1$  и  $u_2$  вычисляются по формулам  $u_1 = h \cdot s^{-1} \bmod q$ ,  $u_2 = r \cdot s^{-1} \bmod q$ .

С учетом этих отличий нетрудно переписать всю схему подписи в «американском» стиле. Доказательство корректности алгоритма проводится совершенно аналогично.

## Задачи и упражнения

Во всех задачах будем предполагать, что  $h(m) = m$  для всех значений  $m$ .

**4.1.** Построить подпись RSA для сообщения  $m$  при следующих параметрах пользователя:

- а.  $P = 5, Q = 11, c = 27, m = 7,$
- б.  $P = 5, Q = 13, c = 29, m = 10,$
- в.  $P = 7, Q = 11, c = 43, m = 5,$
- г.  $P = 7, Q = 13, c = 29, m = 15,$
- д.  $P = 3, Q = 11, c = 7, m = 24.$

**4.2.** Для указанных открытых ключей пользователя RSA проверить подлинность подписанных сообщений:

- а.  $N = 55, d = 3: \langle 7, 28 \rangle, \langle 22, 15 \rangle, \langle 16, 36 \rangle,$
- б.  $N = 65, d = 5: \langle 6, 42 \rangle, \langle 10, 30 \rangle, \langle 6, 41 \rangle,$
- в.  $N = 77, d = 7: \langle 13, 41 \rangle, \langle 11, 28 \rangle, \langle 5, 26 \rangle,$
- г.  $N = 91, d = 5: \langle 15, 71 \rangle, \langle 11, 46 \rangle, \langle 16, 74 \rangle,$
- д.  $N = 33, d = 3: \langle 10, 14 \rangle, \langle 24, 18 \rangle, \langle 17, 8 \rangle.$

**4.3.** Абоненты некоторой сети применяют подпись Эль-Гамаля с общими параметрами  $p = 23, g = 5$ . Для указанных секретных параметров абонентов найти открытый ключ ( $y$ ) и построить подпись для сообщения  $m$ :

- а.  $x = 11, k = 3, m = h = 15,$
- б.  $x = 10, k = 15, m = h = 5,$
- в.  $x = 3, k = 13, m = h = 8,$
- г.  $x = 18, k = 7, m = h = 5,$
- д.  $x = 9, k = 19, m = h = 15.$

**4.4.** Для указанных открытых ключей ( $y$ ) пользователей системы Эль-Гамаля с общими параметрами  $p = 23, g = 5$  проверить подлинность подписанных сообщений:

- а.  $y = 22$ :  $\langle 15; 20, 3 \rangle, \langle 15; 10, 5 \rangle, \langle 15; 19, 3 \rangle,$
- б.  $y = 9$ :  $\langle 5; 19, 17 \rangle, \langle 7; 17, 8 \rangle, \langle 6; 17, 8 \rangle,$
- в.  $y = 10$ :  $\langle 3; 17, 12 \rangle, \langle 2; 17, 12 \rangle, \langle 8; 21, 11 \rangle,$
- г.  $y = 6$ :  $\langle 5; 17, 1 \rangle, \langle 5; 11, 3 \rangle, \langle 5; 17, 10 \rangle,$
- д.  $y = 11$ :  $\langle 15; 7, 1 \rangle, \langle 10; 15, 3 \rangle, \langle 15; 7, 16 \rangle.$

**4.5.** Сообщество пользователей ГОСТа Р34.10-94 имеют общие параметры  $q = 11$ ,  $p = 67$ ,  $a = 25$ . Вычислить открытый ключ ( $y$ ) и построить подпись для сообщения  $m$  при следующих секретных параметрах:

- а.  $x = 3$ ,  $h = m = 10$ ,  $k = 1$ ,
- б.  $x = 8$ ,  $h = m = 1$ ,  $k = 3$ ,
- в.  $x = 5$ ,  $h = m = 5$ ,  $k = 9$ ,
- г.  $x = 2$ ,  $h = m = 6$ ,  $k = 7$ ,
- д.  $x = 9$ ,  $h = m = 7$ ,  $k = 5$ .

**4.6.** Для указанных открытых ключей ( $y$ ) пользователей ГОСТа Р34.10-94 с общими параметрами  $q = 11$ ,  $p = 67$ ,  $a = 25$  проверить подлинность подписанных сообщений:

- а.  $y = 14$ :  $\langle 10; 4, 5 \rangle, \langle 10; 7, 5 \rangle, \langle 10; 3, 8 \rangle,$
- б.  $y = 24$ :  $\langle 1; 3, 5 \rangle, \langle 1; 4, 3 \rangle, \langle 1; 4, 5 \rangle,$
- в.  $y = 40$ :  $\langle 7; 7, 4 \rangle, \langle 7; 9, 2 \rangle, \langle 5; 9, 2 \rangle,$
- г.  $y = 22$ :  $\langle 6; 9, 5 \rangle, \langle 8; 8, 3 \rangle, \langle 7; 4, 1 \rangle,$
- д.  $y = 64$ :  $\langle 10; 7, 3 \rangle, \langle 7; 7, 10 \rangle, \langle 8; 7, 5 \rangle.$

## Темы лабораторных работ

**4.7.** Разработать программы для генерации и проверки подписей RSA. Параметры пользователей необходимо выбрать самостоятельно. Для тестирования программы проверки подписи рекомендуется использовать подписанное сообщение  $\langle 500, 46514 \rangle$  для открытых ключей пользователя  $N = 52891$ ,  $d = 3$  (предполагаем, что  $h(m) = m$ ). Данное сообщение должно признаваться подлинным. Любое изменение компонентов подписанного сообщения с большой вероятностью будет делать подпись недействительной.

- 4.8. Разработать программы для генерации и проверки подписей Эль-Гамаля. Рекомендуемые значения общих открытых параметров  $p = 31259$ ,  $g = 2$ . Остальные параметры пользователей выбрать самостоятельно. Для тестирования программы проверки подписи рекомендуется использовать подписанное сообщение  $\langle 500; 27665, 26022 \rangle$  для открытого ключа пользователя  $y = 16196$  (предполагаем, что  $h(m) = m$ ). Данное сообщение должно признаваться подлинным. Любое изменение компонентов подписанного сообщения с большой вероятностью будет делать подпись недействительной.
- 4.9. Разработать программы для генерации и проверки подписей по ГОСТ Р34.10-94. Рекомендуемые значения общих открытых параметров  $q = 787$ ,  $p = 31481$ ,  $a = 1928$ . Остальные параметры пользователей выбрать самостоятельно. Для тестирования программы проверки подписи рекомендуется использовать подписанное сообщение  $\langle 500; 655, 441 \rangle$  для открытого ключа пользователя  $y = 12785$  (предполагаем, что  $h(m) = m$ ). Данное сообщение должно признаваться подлинным. Любое изменение компонентов подписанного сообщения с большой вероятностью будет делать подпись недействительной.

## Глава 5. КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ

Рассмотренные в предыдущих главах криптографические методы часто используются в качестве инструментов для решения других практически важных задач. Современная криптография позволяет решать проблемы, которые ранее считались в принципе неразрешимыми. Причем в настоящее время многие такие возможности криптографии используются в реальных компьютерных системах. Это и заключение коммерческих сделок в режиме удаленного взаимодействия участников, и осуществление денежных расчетов по сети, и проведение выборов по компьютерным сетям, и многое другое. Методы решения подобных задач обычно описываются в форме так называемых криптографических протоколов. Некоторые из них будут представлены в этой главе.

Обратим внимание читателя на то, что криптографические алгоритмы не просто предоставляют новые возможности пользователю (например, не нужно ходить в банк, можно произвести все необходимые операции со своего домашнего компьютера). Важно то, что они способны обеспечивать надежность значительно более высокую, чем традиционные механизмы. Например, если бумажную банкноту можно подделать, и случаи подделок весьма многочисленны, то электронную банкноту, созданную при помощи криптографических методов, подделать практически невозможно.

Часто практически важные задачи формулируются в игровой, забавной форме, для того чтобы представить идею в «чистом» виде, не загромождая ее техническими деталями. Одна из таких задач — «ментальный покер» — рассматривается в следующем разделе.

### 5.1. Ментальный покер

Рассмотрим задачу проведения честной игры в карты, когда партнеры находятся далеко друг от друга, но связаны компьютерной сетью. Мы рассмотрим предельно упрощенную постановку задачи, где

участвуют всего два игрока и всего три карты. Однако все основные идеи будут продемонстрированы, а обобщения на другие случаи очевидны.

Задача ставится следующим образом. Имеются два игрока Алиса и Боб и три карты  $\alpha, \beta, \gamma$ . Необходимо раздать карты следующим образом: Алиса должна получить одну карту, Боб — также одну, а одна карта должна остаться в прикупе. При этом необходимо, чтобы:

- 1) каждый игрок мог получить с равными вероятностями любую из трех карт  $\alpha, \beta$  или  $\gamma$ , а одна карта оказалась в прикупе;
- 2) каждый игрок знал только свою карту, но не знал карту противника и карту в прикупе;
- 3) в случае спора возможно было пригласить судью и выяснить, кто прав, кто виноват;
- 4) при раздаче карт с помощью компьютерной сети никто не знал, кому какая карта досталась (хотя раздача происходит по открытой линии связи и Ева может записать все передаваемые сообщения).

Дадим описание протокола, позволяющего организовать такую раздачу карт. Его удобно разбить на два этапа.

Предварительный этап необходим для выбора параметров протокола. Участники выбирают несекретное большое простое число  $p$ . Затем Алиса выбирает случайно число  $c_A$ , взаимно простое с  $p - 1$ , и вычисляет по обобщенному алгоритму Евклида число  $d_A$ , такое что

$$c_A d_A \bmod (p - 1) = 1. \quad (5.1)$$

Независимо и аналогично Боб находит пару  $c_B, d_B$ , такую что

$$c_B d_B \bmod (p - 1) = 1. \quad (5.2)$$

Эти числа каждый игрок держит в секрете. Затем Алиса выбирает случайно три (различных) числа  $\hat{\alpha}, \hat{\beta}, \hat{\gamma}$  в промежутке от 1 до  $p - 1$ , в открытом виде передает их Бобу и сообщает, что  $\hat{\alpha}$  соответствует  $\alpha, \hat{\beta} - \beta, \hat{\gamma} - \gamma$  (т. е., например, число 3756 соответствует тузу и т.д.).

После этого начинается второй этап — собственно раздача карт, который удобно описать по шагам.

**Шаг 1.** Алиса вычисляет числа

$$u_1 = \hat{\alpha}^{c_A} \bmod p, \quad u_2 = \hat{\beta}^{c_A} \bmod p, \quad u_3 = \hat{\gamma}^{c_A} \bmod p$$

и высыпает  $u_1$ ,  $u_2$ ,  $u_3$  Бобу, предварительно перемешав их случайным образом.

**Шаг 2.** Боб получает три числа, выбирает случайно одно из них, например  $u_2$ , и отправляет его Алисе по линии связи. Это и будет карта, которая достанется ей в процессе раздачи. Алиса, получив это сообщение, может вычислить

$$\hat{u} = u_2^{d_A} \bmod p = \hat{\beta}^{c_A d_A} \bmod p = \hat{\beta}, \quad (5.3)$$

т.е. она узнает, что ей досталась карта  $\beta$  (можно и не вычислять (5.3), т.к. она знает, какое число  $u$ , какой карте соответствует).

**Шаг 3.** Боб продолжает свои действия. Он вычисляет для оставшихся двух чисел

$$v_1 = u_1^{c_B} \bmod p, \quad v_3 = u_3^{c_B} \bmod p. \quad (5.4)$$

С вероятностью  $1/2$  он переставляет эти два числа и отправляет Алисе.

**Шаг 4.** Алиса выбирает случайно одно из полученных чисел, например  $v_1$ , вычисляет число

$$w_1 = v_1^{d_A} \bmod p \quad (5.5)$$

и отправляет это число обратно к Бобу. Боб вычисляет число

$$z = w_1^{d_B} \bmod p \quad (5.6)$$

и узнает свою карту (у него получается  $\hat{\alpha}$ ). Действительно,

$$z = w_1^{d_B} = v_1^{d_A d_B} = u_1^{c_B d_B d_A} = \hat{\alpha}^{c_A c_B d_A d_B} = \hat{\alpha} \bmod p.$$

Карта, соответствующая  $v_2$ , остается в прикупе.

**Утверждение 5.1.** Описанный протокол удовлетворяет всем свойствам честной раздачи карт.

**Доказательство.** Мы дадим только идею доказательства. Алиса перемешивает числа  $u_1, u_2, u_3$  перед отправкой к Бобу. Затем Боб выбирает одно из этих чисел, не зная, какое число какой карте соответствует. При этом, если Боб выбирает карту случайно, обеспечивается то, что Алиса получает любую из карт с вероятностью  $1/3$ . Аналогично, если Алиса выбирает одну из оставшихся двух карт случайно с равными вероятностями, то Боб также получает любую из трех карт с вероятностью  $1/3$ . Очевидно, что при этих условиях и в прикупе каждая из карт может оказаться с вероятностью  $1/3$ .

Интересно отметить, что если Алиса или Боб будут нарушать некоторые требования протокола, то это может быть использовано им во вред. Поэтому каждый участник заинтересован в точном выполнении всех правил. Проверим это, считая, что игра повторяется многократно.

Предположим, что Алиса не перемешивает карты  $u_1, u_2, u_3$ , а всегда посыпает их в одной и той же последовательности или руководствуется каким-либо другим простым правилом. Если раздача карт выполняется несколько раз, то Боб может использовать это в своих интересах (например, он всегда будет отправлять Алисе самую младшую карту и в каждом случае будет знать, какая карта ей досталась), т.е. Алисе выгодно перемешивать карты. Аналогично, можно проверить, что при необходимости выбора каждому игроку лучше выбирать карту случайно, с равными вероятностями.

Проверим выполнение второго требования, предъявляемого к честной раздаче карт. Когда Боб выбирает число  $u_i$ , соответствующее карте Алисы (шаг 2), он не знает секретное  $c_A$ , следовательно, он не может узнать, какое  $u_i$  какой карте соответствует, а вычисление  $c_A$  по  $u_i$  эквивалентно задаче дискретного логарифмирования и практически невозможно при больших  $p$ . Вообще говоря, когда Алиса выбирает карту для Боба, а он для нее, никто из них не может определить достоинство этой карты, т.к. оно зашифровано при помощи либо  $c_A$ , либо  $c_B$ .

Заметим, что ни Алиса, ни Боб не могут также знать, какая карта осталась в прикупе, т.к. соответствующее число имеет вид  $a^{c_A c_B}$  (см. (5.4)). Алиса не знает  $d_B$ , а Боб не знает  $d_A$ .

Проверим третье свойство. В случае возникновения спора судья может повторить все вычисления по записанным предаваемым числам и выяснить, кто прав.

Проверим четвертое свойство. По линии связи передаются числа

$u_1, u_2, u_3, v_1, v_2, v_3$  и  $w_1$ . Каждое из них может быть представлено в виде  $a^x \bmod p$ , где  $x$  неизвестно Еве. Мы знаем, что нахождение  $x$  — задача дискретного логарифмирования, которая практически неразрешима. Значит Ева ничего не может узнать.  $\square$

Пример 5.1. Пусть Алиса и Боб хотят честно раздать три карты: тройку ( $\alpha$ ), семерку ( $\beta$ ) и туза ( $\gamma$ ). (Точнее, обычно в криптографии предполагается, что никто из них не хочет быть обманутым. Большой «честности» от них не ожидают.) Пусть на предварительном этапе выбраны следующие параметры:

$$p = 23, \quad \hat{\alpha} = 2, \quad \hat{\beta} = 3, \quad \hat{\gamma} = 5.$$

Алиса выбирает  $c_A = 7$ , Боб выбирает  $c_B = 9$ .

Найдем по обобщенному алгоритму Евклида  $d_A$  и  $d_B$ :  $d_A = 19$ ,  $d_B = 5$ .

Шаг 1. Алиса вычисляет

$$u_1 = 2^7 \bmod 23 = 13, \quad u_2 = 3^7 \bmod 23 = 2, \quad u_3 = 5^7 \bmod 23 = 17.$$

Затем она перемешивает  $u_1, u_2, u_3$  и высыпает их Бобу.

Шаг 2. Боб выбирает одно из полученных чисел, пусть, например, выбрано число 17. Он отправляет число 17 к Алисе. Она знает, что число 17 соответствует карте  $\gamma$ , и, таким образом, ее карта при раздаче — туз.

Шаг 3. Боб вычисляет

$$v_1 = 13^9 \bmod 23 = 3, \quad v_2 = 2^9 \bmod 23 = 6$$

и отправляет эти числа к Алисе, возможно, переставив их местами.

Шаг 4. Алиса получает числа 3 и 6, выбирает одно из них, пусть это будет 3, и вычисляет число

$$w_1 = 3^{19} \bmod 23 = 6.$$

Это число она отправляет Бобу, который вычисляет число

$$z = 6^5 \bmod 23 = 2$$

и узнает свою карту  $\alpha$ , т.е. ему досталась тройка. В прикупе осталась семерка, но ни Алиса, ни Боб этого не знают. Ева же, следившая за всеми передаваемыми сообщениями, не может ничего узнать в случае большого  $p$ .  $\square$

## 5.2. Доказательства с нулевым знанием

Рассмотрим следующую задачу, возникающую в некоторых криптографических приложениях. Снова участвуют Алиса и Боб. Алиса знает решение некоторой сложной задачи, она хочет убедить Боба в этом, однако так, чтобы Боб не узнал самого решения задачи. Т.е. в результате Боб должен убедиться в том, что Алиса знает решение, но не должен узнать что-нибудь о самом решении. На первый взгляд сама задача кажется абсурдной, а возможность ее решения — фантастической! Для того чтобы лучше понять ситуацию, рассмотрим случай из жизни пиратов. Пусть, например, Алиса знает карту острова, где спрятан клад, а Боб — капитан корабля, который может доставить ее на остров. Алиса хочет доказать, что карта у нее есть, не показывая ее Бобу (иначе Боб обойдется без Алисы, и весь клад достанется ему).

Такая же задача актуальна для компьютерных сетей в тех случаях, когда Боб (сервер или контроллер домена) должен принять решение о допуске Алисы к информации, хранящейся в сети, но при этом Алиса не хочет, чтобы кто-либо, прослушивающий канал передачи данных и сам сервер, получил какие-либо знания о ее пароле. Т.е. Боб получает «нулевое знание» о пароле (или карте) Алисы, но уверен, что у Алисы такой пароль (или карта) есть.

Итак, наша задача — построить протокол «доказательства с нулевым знанием». При этом мы считаем, что каждый из участников может вести «нечестную» игру и пытаться обмануть другого.

В качестве сложной задачи, решение которой известно Алисе, мы вначале рассмотрим задачу раскраски графа тремя красками. Мы опишем достаточно простой в идейном плане протокол доказательства для этой задачи. Затем мы рассмотрим задачу нахождения гамильтонова цикла в графе с более сложным в идейном плане, но более эффективным в плане реализации протоколом доказательства. Отметим, что обе задачи — раскраски графа тремя красками и нахождения гамильтонова цикла — являются NP-полными. Мы не приводим формального определения NP-полноты, которое может быть найдено, например, в [1]. Для читателя, не знакомого с этим определением, отметим только, что NP-полнота задачи неформально означает, что время решения задачи растет экспоненциально с ростом размера задачи (объема исходных данных).

### Задача о раскраске графа

В задаче о раскраске графа рассматривается граф с множеством вершин  $V$  и множеством ребер  $E$  (числа элементов в этих множествах будем обозначать через  $|V|$  и  $|E|$ ). Алиса знает правильную раскраску этого графа тремя красками (красной (R), синей (B) и желтой (Y)). Правильная раскраска — это такая, когда любые две вершины, соединенные одним ребром, окрашены разными цветами. Приведем пример (рис. 5.1).

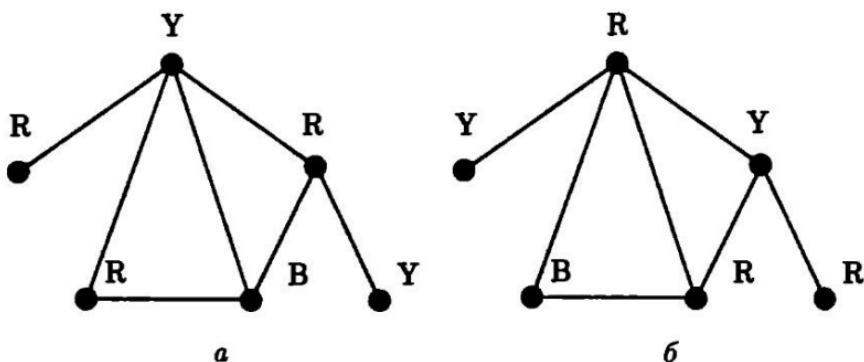


Рис. 5.1. Примеры раскрасок: *a* — правильная, *б* — неправильная

Для получения правильной раскраски графа тремя красками известны только экспоненциальные алгоритмы, т.е. такие, у которых время решения растет экспоненциально с ростом числа вершин и ребер в графе. Поэтому в случае больших  $|V|$  и  $|E|$  эта задача практически неразрешима.

Итак, Алиса знает (правильную) раскраску графа с большими  $|V|$  и  $|E|$ . Она хочет доказать это Бобу, но так, чтобы он ничего не узнал об этой раскраске.

Протокол доказательства состоит из множества одинаковых этапов. Опишем сначала один этап.

**Шаг 1.** Алиса выбирает случайно перестановку  $\Pi$  из трех букв R, B, Y и перенумеровывает все вершины графа согласно этой перестановке. Очевидно, что раскраска останется верной. Например, если  $\Pi = (Y, R, B)$ , то граф слева на рис. 5.1 превращается в граф на рис. 5.2.

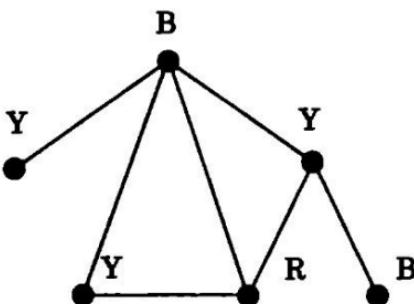


Рис. 5.2. Другой вариант раскраски

**Шаг 2.** Для каждой вершины  $v$  из множества  $V$  Алиса генерирует большое случайное число  $r$  и заменяет в нем два последних бита на 00, 01 или 10, что соответствует красной, синей или желтой вершинам.

**Шаг 3.** Для каждой вершины  $v$  Алиса формирует данные, используемые в RSA, а именно,  $P_v$ ,  $Q_v$ ,  $N_v = P_v Q_v$ ,  $c_v$  и  $d_v$ .

**Шаг 4.** Алиса вычисляет

$$Z_v = r_v^{d_v} \pmod{N_v}$$

и посыпает Бобу значения  $N_v$ ,  $d_v$  и  $Z_v$  для каждой вершины графа.

**Шаг 5.** Боб выбирает случайно одно ребро из множества  $E$  и сообщает Алисе, какое именно ребро он выбрал. В ответ Алиса высыпает числа  $c_{v_1}$  и  $c_{v_2}$ , соответствующие вершинам этого ребра. После этого Боб вычисляет

$$\hat{Z}_{v_1} = Z_{v_1}^{c_{v_1}} \pmod{N_{v_1}} = r_{v_1}, \quad \hat{Z}_{v_2} = Z_{v_2}^{c_{v_2}} \pmod{N_{v_2}} = r_{v_2}$$

и сравнивает два младших бита в полученных числах. При правильной раскраске два младших бита в числах  $\hat{Z}_{v_1}$  и  $\hat{Z}_{v_2}$  должны быть различны. Если значения совпали, значит, Алиса пыталась обмануть Боба, и на этом все заканчивается. Если не совпали, то весь описанный процесс повторяется  $a|E|$  раз, где  $a > 0$  — параметр.

**Утверждение 5.2.** Если Алиса не располагает правильной раскраской графа, то вероятность того, что она сможет обмануть Боба не превышает  $e^{-a}$ , где  $e \approx 2.718$  — число Эйлера (основание натурального логарифма).

**Замечание.** Если взять большое  $a$ , то вероятность обмана можно сделать сколь угодно малой. Например, при  $a = 5$  эта вероятность меньше 0.01.

**Доказательство.** Пусть Алиса не располагает правильной раскраской графа. Значит, хотя бы для одного ребра из  $E$  вершины окрашены в один цвет. Если Алиса будет действовать по протоколу, то вероятность того, что Боб обратится к такому ребру, не меньше  $1/|E|$  (в этом случае Алиса разоблачена). Значит, вероятность того, что Алиса не разоблачена во время одного этапа, не превышает  $1 - 1/|E|$  и, следовательно, вероятность того, что она не будет разоблачена за  $a|E|$  этапов, не превышает  $(1 - 1/|E|)^{a|E|}$ . Используя известное неравенство  $1 - x \leq e^{-x}$ , получаем

$$(1 - 1/|E|)^{a|E|} \leq \left(e^{-1/|E|}\right)^{a|E|} = e^{-a}.$$

□

Проверим все свойства, необходимые для протокола с нулевым знанием.

1. Мы видим, что вероятность возможности обмана для Алисы может быть сколь угодно малой.
2. Посмотрим, почему Боб не получает никакой информации о раскраске. Из-за того, что цвета переставляются случайно на каждом этапе (см. шаг 1), он не сможет узнать истинную раскраску, перебирая все ребра одно за другим, и вообще он ничего не узнает о правильной раскраске. То, что на втором шаге выбирается случайное число  $r_v$ , не позволяет Бобу вычислить по имеющимся  $N_v$  и  $d_v$  коды соответствующих красок. Он не может декодировать полученное  $Z_v$ , потому, что он не знает чисел  $c_v$ , т.к. они для всех вершин не высылаются, а вычислить их он не может, не зная  $P_v$  и  $Q_v$ .
3. Рассмотрим еще одну возможность обмана, которая в принципе может быть у Алисы. Казалось бы, Алиса может подменить  $c_v$

и  $c_{v_2}$ , если ей это выгодно. Однако это невозможно в силу того, что число  $c_v$ , удовлетворяющее равенству

$$c_v d_v \bmod ((P_v - 1)(Q_v - 1)) = 1,$$

единственно.

Таким образом, выполнены все свойства:

- 1) Алиса доказывает Бобу, что знает решение задачи, и вероятность того, что Боб обманут, не больше  $e^{-a}$ ;
- 2) Боб не получает никаких сведений о раскраске.

Рассмотрим последнюю возможность обмана для всех участников. Что будет, если они будут уклоняться от указанного алгоритма, выбирая параметры не случайно?

Пусть, например, Боб запрашивает ребра графа не случайно, а по какому-нибудь простому правилу (например, в соответствии с их номерами). В этом случае, если у Алисы нет правильной раскраски, то она сможет обмануть Боба, «правильно» раскрашивая те ребра, которые будут запрошены. Таким образом, Боб заинтересован в том, чтобы его запросы были случайны и не содержали в себе какой-либо закономерности.

Стойкость остальных шагов определяется стойкостью RSA, и при больших  $P_v$  и  $Q_v$  система достаточно надежна.

### Задача о нахождении гамильтонова цикла в графе

Рассматриваемая в данном разделе задача не просто предоставляет нам возможность описать еще одну схему построения протокола доказательства с нулевым знанием, но и имеет важное теоретическое значение. Блюм (Manuel Blum) показал, что, выражаясь неформально, любое математическое утверждение может быть представлено в виде графа, причем доказательство этого утверждения соответствует гамильтонову циклу в этом графе (см., например, [86]). Поэтому наличие протокола доказательства с нулевым знанием для гамильтонова цикла означает, что доказательство любого математического утверждения может быть представлено в форме доказательства с нулевым знанием.

**Определение 5.1.** Гамильтоновым циклом в графе называется непрерывный путь, проходящий через все вершины графа ровно по одному разу.

**Пример 5.2.** Рассмотрим граф, изображенный на рис. 5.3.

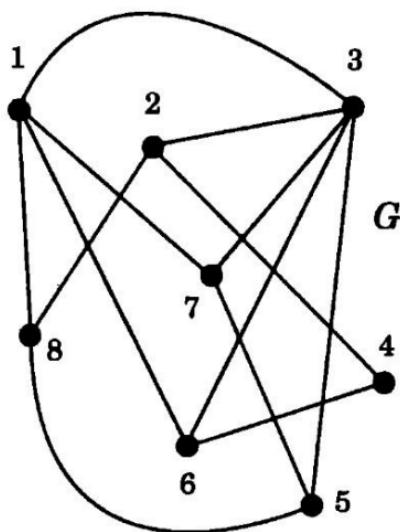


Рис. 5.3. Граф с гамильтоновым циклом  $(8, 2, 4, 6, 3, 5, 7, 1)$

Путь, проходящий последовательно через вершины  $8, 2, 4, 6, 3, 5, 7, 1$ , представляет собой гамильтонов цикл. Действительно, в этом пути содержатся все вершины графа, и каждая вершина посещается только один раз.  $\square$

Ясно, что если в графе  $G$  с  $n$  вершинами гамильтонов цикл существует, то при некоторой нумерации вершин он пройдет точно через вершины с последовательными номерами  $1, 2, 3, \dots, n$ . Поэтому путем перебора всех возможных нумераций вершин мы обязательно найдем гамильтонов цикл. Но количество возможных нумераций равно  $n!$ , и поэтому уже при умеренно больших  $n$ , например, при  $n = 100$ , такой подход становится практически нереализуемым. Доказано, что задача нахождения гамильтонова цикла в графе является NP-полной. Мы уже говорили кратко о понятии NP-полноты. Неформально, NP-полнота рассматриваемой задачи означает, что

для ее решения не существуют (точнее, неизвестны) алгоритмы существенно более быстрые, чем указанный метод перебора.

Нашей задачей будет построение криптографического протокола, с помощью которого Алиса будет доказывать Бобу, что она знает гамильтонов цикл в некотором графе  $G$  так, чтобы Боб не получил никаких знаний о самом этом цикле. Иными словами, Алиса будет предоставлять Бобу доказательство с нулевым знанием. Еще раз напомним читателю, что «нулевое знание» означает, что независимо от числа реализаций протокола доказательства Боб будет располагать точно такими же сведениями о гамильтоновом цикле, какие он мог бы получить, просто изучая представленный ему граф  $G$ .

Итак, допустим, что Алиса знает гамильтонов цикл в графе  $G$ . Теперь она может это доказывать Бобу и всем, кто имеет граф  $G$ , с помощью описываемого ниже протокола. Алиса может использовать это доказательство, например, для идентификации своей личности. Но прежде чем мы перейдем к описанию протокола, договоримся о некоторых обозначениях.

Мы будем обозначать графы буквами  $G, H, F$ , понимая под этим одновременно соответствующие матрицы смежности. Элемент матрицы  $H_{ij} = 1$ , если в графе  $H$  есть ребро, соединяющее вершины  $i$  и  $j$ ;  $H_{ij} = 0$  в противном случае. Символом  $\parallel$  будем обозначать конкатенацию (сцепление) двух чисел, точнее, двоичных слов, им соответствующих. Нам понадобится шифр с открытым ключом. Вообще говоря, это может быть любой шифр, но для определенности будем использовать шифр RSA (разд. 2.6). Будем считать, что Алиса сформировала систему RSA с открытыми параметрами  $N$  и  $d$ . Важно, что зашифрованные в этой системе сообщения может расшифровать только Алиса и больше никто.

Протокол доказательства состоит из следующих четырех шагов (пояснения будут даны ниже).

**Шаг 1.** Алиса строит граф  $H$ , являющийся копией исходного графа  $G$ , где у всех вершин новые, случайно выбранные номера. На языке теории графов говорят, что  $H$  изоморфен  $G$ . Иными словами,  $H$  получается путем некоторой перестановки вершин в графе  $G$  (с сохранением связей между вершинами). Алиса кодирует матрицу  $H$ , приписывая к первоначально содержащимся в ней нулям и единицам случайные числа  $r_{ij}$  по схеме  $\tilde{H}_{ij} = r_{ij} \parallel H_{ij}$ . Затем она шифрует элементы матрицы  $\tilde{H}$ , по-

лучая зашифрованную матрицу  $F$ ,  $F_{ij} = \tilde{H}_{ij}^d \bmod N$ . Матрицу  $F$  Алиса передает Бобу.

**Шаг 2.** Боб, получив зашифрованный граф  $F$ , задает Алисе один из двух вопросов.

1. Каков гамильтонов цикл для графа  $H$ ?
2. Действительно ли граф  $H$  изоморфен  $G$ ?

**Шаг 3.** Алиса отвечает на соответствующий вопрос Боба.

1. Она расшифровывает в  $F$  ребра, образующие гамильтонов цикл.
2. Она расшифровывает  $F$  полностью (фактически передает Бобу граф  $\tilde{H}$ ) и предъявляет перестановки, с помощью которых граф  $H$  был получен из графа  $G$ .

**Шаг 4.** Получив ответ, Боб проверяет правильность расшифровки путем повторного шифрования и сравнения с  $F$  и убеждается либо в том, что показанные ребра действительно образуют гамильтонов цикл, либо в том, что предъявленные перестановки действительно переводят граф  $G$  в граф  $H$ .

Весь протокол повторяется  $t$  раз.

Обсудим вначале кратко несколько вопросов по построению протокола.

1. Зачем Алиса строит изоморфный граф? Если бы она этого не делала, то Боб, получив ответ на свой вопрос номер один, узнал бы гамильтонов цикл в графе  $G$ .
2. Зачем Алиса кодирует матрицу  $H$ ? С этим приемом мы уже встречались при шифровании цветов вершин графа. Дело в том, что невозможно зашифровать непосредственно нули и единицы (с помощью шифра RSA они вообще не шифруются). Даже если заменить их на какие-то произвольные числа  $a$  и  $b$ , то мы получим всего два различных шифротекста, и Бобу не составит труда понять, какой из них какому числу соответствует. Т.е. структура графа не будет скрыта. Здесь мы сталкиваемся с типичной ситуацией, когда требуется использовать так называемый рандомизированный шифр. И такой шифр строится путем добавления случайных чисел в матрицу  $H$  перед

шифрованием. Закодированная матрица  $\tilde{H}$  точно также задает граф (нечетность числа означает наличие ребра, четность — его отсутствие), но после шифрования  $\tilde{H}$  структура графа полностью скрывается (мы используем известное свойство шифра RSA — он полностью скрывает четность числа [61]).

3. Зачем Боб задает два вопроса? Если бы он задавал только вопрос номер один, который по смыслу протокола является основным, то Алиса, не зная в действительности гамильтонова цикла в графе  $G$ , могла бы предъявить Бобу совсем другой граф с таким же количеством вершин и искусственно заложенным в него гамильтоновым циклом. Поэтому Боб иногда просит Алису доказать изоморфизм графов  $H$  и  $G$ . Важно, что Алиса не знает заранее, какой из двух вопросов задаст Боб.
4. Почему Боб не может задать сразу двух вопросов? В этом случае он узнал бы гамильтонов цикл в  $G$ , т.к. ему был бы показан гамильтонов цикл в  $H$  и правило перехода от  $H$  к  $G$ .
5. Зачем Боб проверяет правильность расшифровки? Если бы он этого не делал, то Алиса на четвертом шаге могла бы предоставить ему «выгодную» для себя информацию, а не ту, которую она посыпала ему на втором шаге.

Более точно основные детали протокола обосновываются в ходе доказательства двух основных утверждений.

**Утверждение 5.3.** *Вероятность обмана при  $t$  реализациях протокола не превосходит  $2^{-t}$ .*

**Доказательство.** Вначале покажем, что вероятность обмана в одной реализации протокола равна  $1/2$ . Заметим, что если Алиса действительно знает гамильтонов цикл в графе  $G$ , то она может правильно ответить на любой вопрос Боба. Если же она не знает гамильтонов цикл, то самое большее, что она может сделать, — это подготовиться к ответу на первый либо на второй вопрос. В ожидании первого вопроса, она создает новый граф с искусственно заложенным в него гамильтоновым циклом. Но в этом случае она не сможет доказать его изоморфизм графу  $G$ . В ожидании второго вопроса, она строит граф, изоморфный графу  $G$ . Но в этом случае она не сможет показать в нем гамильтонов цикл. Таким образом, вероятность успешности обмана равна вероятности угадывания номера

вопроса. В предположении, что Боб задает оба вопроса с одинаковыми вероятностями, мы получаем, что вероятность обмана равна  $1/2$ .

Так как Боб прекращает игру при первом же неправильном ответе, вероятность обмана при  $t$  реализациях протокола не превосходит  $(1/2)^t$ .  $\square$

**Утверждение 5.4.** *Представленный протокол реализует доказательство с нулевым знанием.*

**Доказательство.** Чтобы доказать, что Боб не получает никаких знаний в ходе реализации протокола, достаточно показать, что все, что он получает от Алисы, он мог бы получить сам, не вступая с ней ни в какое общение.

Рассмотрим вначале второй вопрос Боба. В ответ на этот вопрос он получает граф, изоморфный графу  $G$ . Но он сам мог строить сколько угодно изоморфных графов, и то, что присыпает ему Алиса, это просто один из них.

Случай с первым вопросом не столь очевиден. В ответ на первый вопрос Боб получает гамильтонов цикл в графе, изоморфном графу  $G$ . На первый взгляд может показаться, что это дает Бобу какую-то информацию. Однако это не так. Заметим, что если в  $G$  есть гамильтонов цикл, то при некоторой нумерации вершин существует изоморфный граф, который задается матрицей смежности вида

$$\left( \begin{array}{ccccccc} * & 1 & * & \cdots & * & * & * \\ * & * & 1 & \cdots & * & * & * \\ & & & \cdots & & & \\ * & * & * & \cdots & * & 1 & * \\ * & * & * & \cdots & * & * & 1 \\ 1 & * & * & \cdots & * & * & * \end{array} \right), \quad (5.7)$$

где  $*$  означает неопределенность в наличии или отсутствии ребра. Т.е. при такой нумерации гамильтонов цикл проходит через вершины в порядке возрастания номеров. Изменяя нумерацию вершин, Боб может получать из (5.7) всевозможные изоморфные матрицы. Когда Алиса, отвечая на его первый вопрос, открывает гамильтонов цикл, Боб видит как раз одну из таких матриц.

Таким образом, Боб не получает от Алисы никакой информации, которую он не мог бы получить сам.  $\square$

Рассмотрим пример, иллюстрирующий все основные этапы описанного протокола.

Пример 5.3. Возьмем в качестве основного граф  $G$ , изображенный на рис. 5.3. Его матрица смежности имеет вид

$$G = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & \boxed{1} \\ 2 & 0 & 0 & 1 & \boxed{1} & 0 & 0 & 0 & 1 \\ 3 & 1 & 1 & 0 & 0 & \boxed{1} & 1 & 1 & 0 \\ 4 & 0 & 1 & 0 & 0 & 0 & \boxed{1} & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 & \boxed{1} & 1 \\ 6 & 1 & 0 & \boxed{1} & 1 & 0 & 0 & 0 & 0 \\ 7 & \boxed{1} & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 8 & 1 & \boxed{1} & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

В матрице с помощью  $\boxed{\cdot}$  показан гамильтонов цикл. Алиса выбирает некоторую случайную нумерацию вершин, скажем, 7, 4, 5, 3, 1, 2, 8, 6, и получает изоморфный граф

$$H = \begin{pmatrix} & 7 & 4 & 5 & 3 & 1 & 2 & 8 & 6 \\ 7 & 0 & 0 & 1 & 1 & \boxed{1} & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \boxed{1} \\ 5 & \boxed{1} & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 3 & 1 & 0 & \boxed{1} & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & \boxed{1} & 1 \\ 2 & 0 & \boxed{1} & 0 & 1 & 0 & 0 & 1 & 0 \\ 8 & 0 & 0 & 1 & 0 & 1 & \boxed{1} & 0 & 0 \\ 6 & 0 & 1 & 0 & \boxed{1} & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Для шифрования матрицы будем использовать систему RSA с параметрами  $N = 55$ ,  $d = 3$ . Вначале закодируем матрицу  $H$ . В рамках данного примера просто припишем слева к каждому элементу матрицы выбранную случайно с равными вероятностями цифру из

множества  $\{1, 2, 3, 4, 5\}$ :

$$\tilde{H} = \begin{pmatrix} 50 & 20 & 11 & 31 & 21 & 40 & 20 & 10 \\ 40 & 30 & 50 & 20 & 10 & 41 & 50 & 21 \\ 41 & 30 & 50 & 11 & 30 & 20 & 51 & 40 \\ 11 & 10 & 41 & 30 & 51 & 41 & 30 & 21 \\ 31 & 20 & 40 & 11 & 50 & 10 & 41 & 31 \\ 50 & 41 & 20 & 21 & 40 & 10 & 21 & 50 \\ 40 & 30 & 31 & 50 & 41 & 21 & 30 & 40 \\ 20 & 41 & 10 & 51 & 41 & 20 & 30 & 40 \end{pmatrix}.$$

Теперь мы шифруем матрицу  $\tilde{H}$ , возводя каждый ее элемент в куб по модулю 55:

$$F = \begin{pmatrix} 40 & 25 & 11 & 36 & 21 & 35 & 25 & 10 \\ 35 & 50 & 40 & 25 & 10 & 06 & 40 & 21 \\ 06 & 50 & 40 & 11 & 50 & 25 & 46 & 35 \\ 11 & 10 & 06 & 50 & 46 & 06 & 50 & 21 \\ 36 & 25 & 35 & 11 & 40 & 10 & 06 & 36 \\ 40 & 06 & 25 & 21 & 35 & 10 & 21 & 40 \\ 35 & 50 & 36 & 40 & 06 & 21 & 50 & 35 \\ 25 & 06 & 10 & 46 & 06 & 25 & 50 & 35 \end{pmatrix}.$$

(При внимательном просмотре матрицы  $F$  может показаться, что использованный нами шифр плохо скрывает исходную матрицу  $\tilde{H}$ . Это объясняется тем, что, во-первых, модуль 55 слишком мал и, во-вторых, в матрице  $\tilde{H}$  много чисел, не взаимно простых с модулем. Для реальных систем RSA, где  $N$  — большое число, такая ситуация практически исключена.)

Боб получает матрицу  $F$  и задает один из двух вопросов. Если он просит доказать изоморфизм графов, то Алиса просто посыпает ему кодированную матрицу  $\tilde{H}$  и использованную нумерацию 7, 4, 5, 3, 1, 2, 8, 6. Боб проверяет соответствие матрицы  $\tilde{H}$  матрице  $F$ , т.е. выполнение равенств  $50^3 \bmod 55 = 40$ ,  $20^3 \bmod 55 = 25$  и т.д. Из матрицы  $\tilde{H}$  Боб получает граф  $H$  (просто отбросив старшую десятичную цифру). Затем он переставляет вершины графа  $G$  в соответствии с полученной нумерацией, как это делала Алиса, и убеждается в том, что  $H$  и  $G$  — один и тот же граф.

Если Боб просит показать ему гамильтонов цикл, то Алиса посыпает ему соответствующий список (закодированных) ребер графа

$H: (1, 5, 21), (5, 7, 41), (7, 6, 21), \dots, (3, 1, 41)$ . Каждый элемент содержит номера вершин и код ребра. Боб проверяет соответствие указанных в списке ребер матрице  $F$ , например,  $21^3 \bmod 55 = 21 = F_{1,5}$ ,  $41^3 \bmod 55 = 06 = F_{5,7}$  и т.д. Затем он убеждается, что указанный в списке путь проходит через все вершины графа по одному разу.  $\square$

### 5.3. Электронные деньги

Во многих странах люди оплачивают покупки при помощи электронных карточек, заказывают авиабилеты через Интернет, покупают самые разнообразные товары в Интернет-магазинах. Сведения о покупках накапливаются в магазинах и банках. Поэтому появилась новая проблема, иногда называемая «проблема Большого Брата».

Суть проблемы состоит в том, что исчезает анонимность процесса покупки, т.е. информация о покупках любого человека может стать известной третьим лицам и использоваться против него. Например, сведения о покупке билета на поезд или самолет могут представлять интерес для преступников, информация о закупках алкогольных напитков политическим деятелем может быть использована против него его противниками и т.д., и т.п.

Поэтому возникла идея разработать такие схемы электронных платежей, которые сохраняли бы анонимность покупателя в той же степени, что и при расчете наличными деньгами. Такие протоколы называются электронными или цифровыми деньгами (digital cache), что подчеркивает их основное свойство — они обеспечивают ту же степень анонимности, что и обычные деньги. Некоторые схемы уже используются в реальной жизни. Описываемая ниже схема была предложена Д. Чаумом (David Chaum), см. [2, 61].

Мы рассмотрим две «плохие» схемы, а затем «хорошую», чтобы было легче понять суть метода.

Вначале дадим более точную постановку задачи. Имеются три участника: банк, покупатель и магазин. Покупатель и магазин имеют соответствующие счета в банке, и покупатель хочет купить некоторый товар в магазине. Покупка осуществляется в виде трехступенчатого процесса:

- 1) покупатель снимает нужную сумму со своего счета в банке;
- 2) покупатель «пересыпает» деньги в магазин;

- 3) магазин сообщает об этом в банк, соответствующая сумма денег зачисляется на счет магазина, а покупатель забирает товар (или последний ему доставляется).

Наша цель — разработать такую схему, чтобы

- она была надежна;
- чтобы банк не знал, кто купил товар, т.е. была сохранена анонимность обычных денег.

Опишем первую «плохую» схему (она базируется на RSA). Банк имеет следующую информацию: секретные числа  $P, Q, c$  и открытые

$$N = PQ,$$

$$d = c^{-1} \bmod (P-1)(Q-1). \quad (5.8)$$

Допустим, покупатель решил израсходовать некоторую заранее оговоренную с банком сумму (например, 100\$). (Мы сначала рассмотрим случай, когда может использоваться «банкнота» только одного номинала (скажем, 100\$).) Покупатель высылает в банк число  $n$ , которое будет номером банкноты (обычно требуется, чтобы генерировалось случайное число в промежутке  $[2, N-1]$ ).

Банк вычисляет число

$$s = n^c \bmod N \quad (5.9)$$

и формирует банкноту  $\langle n, s \rangle$ , которую возвращает покупателю, предварительно уменьшив его счет на 100\$. Параметр  $s$  в банкноте — это подпись банка. Никто не может подделать подпись, т.к. число  $s$  секретно.

Покупатель предъявляет банкноту  $\langle n, s \rangle$  в магазине, чтобы купить товар. Магазин отправляет эту банкноту в банк для проверки. Прежде всего, банк проверяет правильность подписи (этую проверку мог бы сделать и магазин, используя открытые ключи банка). Но кроме этого банк хранит все номера возвратившихся к нему банкнот и проверяет, нет ли числа  $n$  в этом списке. Если  $n$  есть в списке, то платеж не принимается (кто-то пытается использовать банкноту повторно), и банк сообщает об этом магазину. Если же все проверки прошли успешно, то банк добавляет 100\$ на счет магазина, а магазин отпускает товар покупателю.

Недостаток этой схемы — отсутствует анонимность. Банк, а также все, кто имеет доступ к открытым линиям связи, могут запомнить, какому покупателю соответствует число  $n$ , и тем самым выяснить, кто купил товар.

Рассмотрим вторую «плохую» схему, которая уже обеспечивает анонимность. Эта схема базируется на так называемой «слепой подписи».

Снова покупатель хочет купить товар. Он генерирует число  $n$ , которое теперь *не будет* посыпаться в банк. Затем он генерирует случайное число  $r$ , взаимно простое с  $N$ , и вычисляет число

$$\hat{n} = (n \cdot r^d) \bmod N. \quad (5.10)$$

Число  $\hat{n}$  покупатель отправляет в банк.

Банк вычисляет число

$$\hat{s} = \hat{n}^c \bmod N \quad (5.11)$$

и отправляет  $\hat{s}$  обратно покупателю (не забыв при этом снять 100\$ с его счета).

Покупатель находит число  $r^{-1} \bmod N$  и вычисляет

$$s = (\hat{s} \cdot r^{-1}) \bmod N. \quad (5.12)$$

Заметим, что с учетом соотношений (5.11), (5.10) и (5.8) имеем

$$s = \hat{n}^c \cdot r^{-1} = (n \cdot r^d)^c \cdot r^{-1} = n^c r^{dc} \cdot r^{-1} = n^c r^1 r^{-1} = n^c \bmod N,$$

т.е. мы получили подпись банка к  $n$  (см. (5.9)), но самого числа  $n$  ни банк, ни кто либо другой не видел. Вычисление (5.11) называется «слепой подписью», т.к. реальное сообщение ( $n$ ) подписывающий не видит и узнать не может.

Таким образом, покупатель имеет число  $n$ , которое никому не известно и никогда не передавалось по каналам связи, и подпись банка  $s$ , совпадающую с вычисленной по (5.9). Покупатель формирует банкноту  $\langle n, s \rangle$  и действует так же, как в первой «плохой» схеме. Но теперь никто не знает, кому соответствует эта банкнота, т.е. она стала анонимной, как обычная бумажная банкнота.

Действия магазина и банка после предъявления покупателем банкноты  $\langle n, s \rangle$  ничем не отличаются от действий, описанных в первой схеме.

Почему же данная схема плохая? Она имеет следующий недостаток: можно сфабриковать фальшивую банкноту, если известны хотя бы две настоящие. Делается это так. Путь злоумышленник (будь то покупатель или магазин) имеет две настоящие банкноты  $\langle n_1, s_1 \rangle$  и  $\langle n_2, s_2 \rangle$ . Тогда он легко сможет изготовить фальшивую банкноту  $\langle n_3, s_3 \rangle$ , вычислив числа

$$n_3 = n_1 n_2 \bmod N,$$

$$s_3 = s_1 s_2 \bmod N.$$

Действительно,

$$n_3^c = (n_1 n_2)^c = n_1^c n_2^c = s_1 s_2 = s_3 \bmod N, \quad (5.13)$$

т.е.  $s_3$  является правильной подписью для  $n_3$ , и у банка нет никаких оснований, чтобы не принять эту фальшивую банкноту (он просто не сможет отличить ее от подлинной). Это так называемое «мульти-пликативное свойство» системы RSA.

Опишем, наконец, «хорошую» схему, в которой устраниены все недостатки первых двух. В одном варианте такой схемы используется некоторая односторонняя функция

$$f : \{1, \dots, N\} \rightarrow \{1, \dots, N\}$$

( $f$  вычисляется легко, а обратная к ней функция  $f^{-1}$  — очень трудно). Функция  $f$  не секретна и известна всем (покупателю, банку и магазину).

Банкнота теперь определяется как пара чисел  $\langle n, s_f \rangle$ , где

$$s_f = (f(n))^c \bmod N,$$

т.е. подписывается не  $n$ , а значение  $f(n)$ .

Покупатель генерирует  $n$  (никому его не показывая), вычисляет  $f(n)$ , подписывает в банке при помощи «слепой подписи» число  $f(n)$  и формирует банкноту  $\langle n, s_f \rangle$ . Эта банкнота обладает всеми хорошими свойствами, как и во второй схеме, в то же время подделать такую банкноту невозможно, т.к. невозможно вычислить  $f^{-1}$ . Для проверки подписи (т.е. подлинности банкноты) нужно вычислить  $f(n)$  и убедиться, что

$$s_f^d \bmod N = f(n).$$

Заметим, что при выборе односторонней функции нужно проявлять осторожность. Например, функция  $f(n) = n^2 \bmod N$ , которая действительно является односторонней, не годится для рассматриваемого протокола. Читатель может проверить, что банкноты, созданные с использованием такой функции, будут по-прежнему обладать мультипликативным свойством (5.13). На практике в качестве  $f(n)$  обычно используются криптографические хеш-функции, описываемые в главе 8.

Все остальные действия магазина и банка остаются такими же, как и в ранее описанных схемах.

Есть еще один, более простой, способ борьбы с мультипликативным свойством системы RSA — внесение избыточности в сообщение. Допустим, что длина модуля  $N = 1024$  бита. Такой же может быть и длина числа  $n$ . Будем записывать (случайно выбираемый) номер банкноты только в младшие 512 бит  $n$ , а в старшие 512 бит  $n$  запишем некоторое фиксированное число. Это фиксированное число может нести полезную информацию, такую, как номинал банкноты и наименование банка (с помощью 512 бит можно представить строку из 64 символов ASCII). Теперь банк при предъявлении ему банкноты будет обязательно проверять наличие фиксированного заголовка в параметре  $n$  и отвергать банкноту в случае его отсутствия. Вероятность того, что при перемножении двух чисел по модулю  $N$  результат совпадет с ними в 512 битах пренебрежимо мала. Поэтому получить фальшивую банкноту по формуле (5.13) не удастся.

**Пример 5.4.** Пусть в качестве секретных параметров банка выбраны числа  $P = 17$ ,  $Q = 7$ ,  $c = 77$ . Соответствующие им открытые параметры будут  $N = 119$ ,  $d = 5$ . Для исключения возможности подделки банкнот их допустимыми номерами считаются только числа, состоящие из двух одинаковых десятичных цифр, например, 11, 77, 99.

Когда покупатель хочет получить банкноту, он вначале случайным образом выбирает ее номер (из числа допустимых). Предположим, он выбрал  $n = 33$ . Затем он находит случайное число  $r$ , взаимнопростое со 119. Допустим,  $r = 67$ ,  $\gcd(67, 119) = 1$ . Далее, покупатель вычисляет

$$\hat{n} = (33 \cdot 67^5) \bmod 119 = (33 \cdot 16) \bmod 119 = 52.$$

Именно число 52 он посыпает в банк.

Банк списывает со счета покупателя 100\$ и отправляет ему свою слепую подпись

$$\hat{s} = 52^{77} \bmod 119 = 103.$$

Покупатель вычисляет  $r^{-1} = 67^{-1} \bmod 119 = 16$  и  $s = 103 \cdot 16 \bmod 119 = 101$  и получает платежеспособную банкноту

$$\langle n, s \rangle = \langle 33, 101 \rangle.$$

Эту банкноту он приносит (или посыпает) в магазин, чтобы купить товар.

Магазин предъявляет банкноту в банк. Банк делает следующие проверки:

- 1) номер банкноты ( $n = 33$ ) состоит из двух одинаковых десятичных цифр (т.е. содержит требуемую избыточность);
- 2) ранее банкнота с таким номером не предъявлялась;
- 2) подпись банка верна, т.е.  $33^5 \bmod 119 = 101$ .

Так как все проверки прошли успешно, банк зачисляет 100\$ (это фиксированный номинал банкноты) на счет магазина, о чём ему и сообщает. Магазин отпускает товар покупателю.  $\square$

В завершение разберем еще две проблемы, возникающие в связи с рассмотренной схемой электронных денег.

В представленной схеме независимо действующие покупатели или даже один покупатель, который не помнит номеров ранее использованных им банкнот, могут случайно сгенерировать две или более банкноты с одинаковыми номерами. По условиям протокола банк примет к оплате только одну из таких банкнот (ту, которая будет предъявлена первой). Однако примем во внимание размеры чисел, используемых в протоколе. Если номер банкноты — число длиной 512 бит и покупатели генерируют его действительно случайным образом, то вероятность получения когда либо двух одинаковых номеров пренебрежимо мала.

Вторая проблема состоит в том, что в рассмотренной схеме используются только банкноты одного фиксированного номинала, что, конечно, неудобно для покупателя. Решение проблемы использования банкнот разного номинала возможно следующим образом. Банк

заводит несколько пар  $(c_i, d_i)$ , обладающих свойством (5.8), и объявляет, что  $d_1$  соответствует, например, 1000 руб.,  $d_2$  — 500 руб. и т.д. Когда покупатель запрашивает слепую подпись в банке, он дополнительно сообщает, какого номинала банкноту он хочет получить. Банк снимает с его счета сумму, равную указанному номиналу, и формирует подпись, используя соответствующее секретное число  $c_i$ . Когда впоследствии банк получает подписанную банкноту, он использует для проверки подписи по очереди числа  $d_1, d_2$  и т.д. Если подпись оказалась верна для какого-то  $d_i$ , то принимается банкнота  $i$ -го номинала. В случае, когда параметр  $n$  банкноты содержит фиксированный заголовок с указанием ее номинала, задача проверки подписи облегчается — банк сразу использует нужный ключ  $d_i$ .

#### 5.4. Взаимная идентификация с установлением ключа

В данном разделе мы рассмотрим криптографически стойкий протокол, в результате реализации которого два абонента сети  $A$  и  $B$  взаимно идентифицируют друг друга (т.е.  $A$  убеждается в том, что взаимодействует с  $B$ , а  $B$  — в том, что он взаимодействует с  $A$ ) и формируют общий секретный ключ, который может использоваться в дальнейшем для шифрования передаваемых ими сообщений. В реальной жизни в качестве  $A$  и  $B$  могут выступать пользователь и компьютерная система или две различные компьютерные системы — суть описываемого ниже протокола от этого не меняется.

В процессе описания мы рассматриваем различные, все более изощренные типы атак и средства защиты от них. Так мы уже рассматривали ранее (см. разд. 2.1 и 2.2) подходы к решению задачи идентификации и установления ключа. Однако мы исходили из того, что противник может только прослушивать информацию, передаваемую по открытому каналу. Но в современных сетях передачи данных, например в Интернете, информация от одного пользователя передается другому через множество промежуточных узлов (маршрутизаторы, шлюзы, почтовые серверы и т.д.), не контролируемых этими пользователями. В результате противник, обосновавшийся на одном таком промежуточном узле, может не только прослушивать информацию, т.е. играть чисто пассивную роль, но и осуществлять активные воздействия, например, изменять, добавлять или удалять

сообщения.

Разберем, например, типичную атаку на систему Диффи-Хеллмана в сети связи с активным противником. Алиса выбирает свое секретное число  $X_A$  и посыпает Бобу  $g^{X_A}$ . Боб выбирает свое секретное число  $X_B$  и посыпает Алисе  $g^{X_B}$ . Однако Ева перехватывает эти числа и посыпает вместо них и Алисе, и Бобу  $g^{X_E}$ , где  $X_E$  — ее число. Все эти числа выглядят как совершенно случайные, так что ни Алиса, ни Боб ничего не подозревают. В результате Алиса формирует ключ  $K_A = g^{X_E X_A}$ , а Боб — ключ  $K_B = g^{X_E X_B}$ . Оба этих ключа могут быть легко вычислены и Евой. Теперь, когда Алиса посыпает Бобу сообщение, зашифрованное с ключом  $K_A$ , Ева расшифровывает его, снова шифрует с ключом  $K_B$  и отправляет Бобу. Аналогично Ева действует и при передаче сообщений в обратном направлении. Боб и Алиса взаимодействуют, как им кажется, в защищенном режиме, но на самом деле Ева читает все их сообщения.

Такая атака становится невозможной, если Алиса и Боб не передают открытые ключи (в системе Диффи-Хеллмана это  $Y_A = g^{X_A}$  и  $Y_B = g^{X_B}$ ) по каналу связи, а выбирают их из некоторой таблицы или справочника, который был получен ими ранее из «надежного» источника (как это предполагалось в разд. 2.2).

Вообще, большинство криптосистем с открытыми ключами требуют наличия некоторой организационной структуры, занимающейся сертификацией открытых ключей. Такая структура может, например, выглядеть следующим образом. В сети, которой принадлежат Алиса и Боб, имеется «честный» пользователь Трент (абонент  $T$ ), который заинтересован только в том, чтобы сеть работала надежно (скорее всего это не человек, а хорошо охраняемый компьютер, работающий по жестко заложенной программе). Трент располагает какой-либо надежной криптосистемой (например, RSA с длиной модуля порядка 10000 бит) с соответствующими открытыми ключами и выполняет всего две функции:

- 1) он добавляет в свою базу данных информацию об открытом ключе пользователя, присыпаемую в виде сообщения, зашифрованного с использованием открытого ключа Трента;
- 2) он сообщает информацию о чьем-либо открытом ключе, снабженную своей подписью.

Открытые ключи Трента доводятся до сведения всех пользователей каким-либо способом, исключающим вмешательство Евы. На-

пример, они публикуются в виде рекламируемого сообщения в газете. Теперь Алиса, вычислив свой открытый ключ, формирует сообщение из своего имени и этого ключа, шифрует его с использованием открытого ключа Трента и посыпает Тренту (никто кроме Трента не может расшифровать это сообщение). Боб, когда ему нужен открытый ключ Алисы, посыпает запрос Тренту, и Трент присыпает ему подпсанный ключ Алисы (никто не может подделать подпись Трента). Боб проверяет подпись Трента, используя его открытый ключ, и принимает ключ Алисы как достоверный. Таким образом, каждый пользователь сети получает достоверную информацию об открытых ключах других пользователей, и Ева никак не может вмешаться в этот процесс.

Итак, если Алиса и Боб пользуются достоверными открытыми ключами, то схема Диффи–Хеллмана решает задачу установления секретного ключа. Однако она непосредственно не обеспечивает идентификацию пользователей. Действительно, если вместо Алисы выступала Ева, которая не знает секретного ключа Алисы, то у них с Бобом будут сформированы различные секретные ключи, но это может выясниться только позднее, на стадии обмена данными, когда Боб, например, не сможет расшифровать переданное ему сообщение или обнаружит, что «Алиса» не понимает того, что он посыпает ей. Часто требуется обеспечить явную идентификацию, чтобы по завершении протокола стороны точно знали, кто есть кто.

У схемы Диффи–Хеллмана есть и другой недостаток: секретный ключ, который формируют Алиса и Боб, будет всегда один и тот же, пока они не поменяют открытые ключи. Но смена открытых ключей — это относительно долгий процесс (например, обычно требуется оповестить всех пользователей сети об изменении какого-то открытого ключа, чтобы они могли скорректировать информацию в своих справочниках). Хотелось бы иметь протокол, обеспечивающий оперативное создание каждый раз различных, случайно выбираемых секретных ключей.

Решение состоит в использовании какого-либо шифра с открытым ключом для передачи секретных ключей. Обозначим шифр сообщения  $x$ , построенный с использованием открытого ключа пользователя  $A$ , через  $P_A(x)$ . (Например,  $P_A(x)$  может быть шифром RSA или шифром Эль-Гамаля. В случае RSA  $P_A(x) = x^{d_A} \bmod N_A$ , где пара чисел  $d_A$  и  $N_A$  представляет собой открытый ключ пользователя  $A$ .) Все, кто знает открытый ключ  $A$ , могут вычислить  $P_A(x)$

для сообщения  $x$ . В то же время только  $A$ , знающий соответствующий секретный ключ, может получить  $x$  из  $y = P_A(x)$ . Аналогично ( $P_B(x)$ ) будем обозначать шифр, построенный с помощью открытого ключа пользователя  $B$ . Символом  $\parallel$ , как и ранее, будем обозначать конкатенацию чисел. Мы опишем протокол поэтапно, чтобы не «утопить» читателя в деталях.

Напомним, что мы решаем следующую задачу: Алиса и Боб хотят взаимно идентифицировать друг-друга и установить общий секретный ключ. Рассмотрим вначале следующий (плохой) протокол, состоящий из трех шагов, чтобы обсудить несколько важных вопросов.

**Шаг 1.** Алиса придумывает секретный ключ  $k_1$ , шифрует его, используя открытый ключ Боба, и посыпает Бобу:

$$A \longrightarrow B : \quad P_B(k_1). \quad (5.14)$$

**Шаг 2.** Боб расшифровывает  $k_1$ , снова шифрует его, используя открытый ключ Алисы, и посыпает Алисе:

$$A \longleftarrow B : \quad P_A(k_1). \quad (5.15)$$

**Шаг 3.** Алиса расшифровывает  $k_1$  и сравнивает его с тем, который она придумала на шаге 1.

Что мы имеем в результате реализации этого протокола? Во-первых, Алиса и Боб получили общий секретный ключ  $k_1$ , неизвестный Еве (Ева не может расшифровать ни  $P_B(k_1)$ , ни  $P_A(k_1)$ ). Во-вторых, Алиса получила криптографически стойкую идентификацию Боба, т.к. никто кроме него не смог бы расшифровать  $k_1$ . Очевидно, что в данном протоколе Боб не получает никакой идентификации Алисы (сообщение (5.14) мог послать кто угодно). Он мог бы провести симметричный протокол со своей стороны:

$$A \longleftarrow B : \quad P_A(k_2), \quad (5.16)$$

$$A \longrightarrow B : \quad P_B(k_2) \quad (5.17)$$

и получить такую идентификацию. Проблема, однако, здесь состоит в логической независимости двух протоколов, в результате чего нет гарантии, что оба протокола проводятся одними и теми же участниками.

Но есть и более тонкая проблема. Алиса может использовать описанный протокол для вскрытия криптосистемы Боба! Делается это так. Допустим, Алиса перехватила какое-то сообщение  $y$ , предназначенное для Боба, т. е.  $y = P_B(x)$ . Она притворяется, что хочет войти в систему Боба, и запускает протокол (5.14), (5.15). Однако вместо  $P_B(k_1)$  она передает Бобу сообщение  $y$ . Так как  $k_1$  — произвольно выбранное число, то Боб не может ничего заподозрить. Он честно выполняет свой шаг в протоколе и расшифровывает для Алисы  $x$ !

Урок, который следует отсюда извлечь, следующий: никогда ни для кого не следует расшифровывать случайные числа. Это может повредить вашей безопасности. Средство борьбы с такой «опасной» случайностью — внесение избыточности в сообщения, например, введение какого-либо элемента, известного получателю и ожидаемого им. В частности, в (5.14) Алиса могла бы послать свое имя. Она могла бы построить сообщение, отведя 512 бит под случайное число  $k_1$  и 512 бит под свое имя, адрес, фрагмент открытого ключа и другую легко проверяемую информацию (будем обозначать все это вместе через  $\hat{A}$ ), и послать Бобу  $P_B(k_1\|\hat{A})$ . В этом случае Боб не стал бы посыпать Алисе сообщение  $x$ , т.к. его соответствующие 512 бит наверняка не содержали бы  $\hat{A}$ .

Все вышеизложенное приводит нас к следующему протоколу Нидхама–Шредера (Needham, Schroeder, см., например, [68]), который полностью решает поставленную в начале раздела задачу.

**Шаг 1.** Алиса выбирает случайное число  $k_1$ , объединяет его со своей открытой информацией  $\hat{A}$  и посыпает Бобу

$$A \longrightarrow B : \quad P_B(k_1\|\hat{A}). \quad (5.18)$$

**Шаг 2.** Боб расшифровывает (5.18) и убеждается в том, что полученное сообщение содержит открытую информацию Алисы  $\hat{A}$ . Затем он выбирает случайное число  $k_2$ , объединяет его с  $k_1$  и посыпает Алисе

$$A \longleftarrow B : \quad P_A(k_1\|k_2). \quad (5.19)$$

**Шаг 3.** Алиса расшифровывает (5.19) и убеждается в том, что полученное сообщение содержит  $k_1$ . Это является для нее надежным признаком идентификации Боба, т.к. никто другой не мог

бы извлечь  $k_1$  из (5.18). Алиса посыпает Бобу

$$A \longrightarrow B : \quad P_B(k_2). \quad (5.20)$$

**Шаг 4.** Боб расшифровывает (5.20) и убеждается в том, что он получил  $k_2$ . Это является для него надежным признаком идентификации Алисы, т.к. никто другой не мог бы извлечь  $k_2$  из (5.19).

Теперь Алиса и Боб могут сформировать из  $k_1$ ,  $k_2$  общий ключ, например,  $k = k_1 \oplus k_2$ , где  $\oplus$  — побитовая сумма по модулю 2, или использовать  $k_1$  и  $k_2$  по-отдельности для шифрования входящих и исходящих сообщений.

**Пример 5.5.** Пусть в некоторой сети используется шифр Эль-Гамаля с открытыми параметрами  $p = 107$ ,  $g = 2$ . Пользователи  $A$  и  $B$  имеют открытые ключи  $d_A = 58$ ,  $d_B = 28$ , которым соответствуют секретные  $c_A = 33$ ,  $c_B = 45$ . Рассмотрим реализацию протокола Нидхама-Шредера для взаимной идентификации пользователей  $A$  и  $B$  и установления общего секретного ключа. Учитывая небольшую величину модуля  $p$  в нашем примере, будем использовать в качестве идентификаторов пользователей одну цифру в десятичной записи, пусть  $\hat{A} = 1$ ,  $\hat{B} = 2$ , и секретный ключ будем получать также в виде одной десятичной цифры.

На первом шаге протокола  $A$  выбирает секретный ключ, пусть  $k_1 = 3$ , и формирует сообщение  $m = k_1 \parallel \hat{A} = 31$ . Это сообщение шифруется шифром Эль-Гамаля на открытом ключе пользователя  $B$ :

$$\begin{aligned} k &= 15, \quad r = g^k \bmod p = 2^{15} \bmod 107 = 26, \\ e &= m \cdot d_B^k \bmod p = 31 \cdot 28^{15} \bmod 107 = 47. \end{aligned}$$

Пара чисел  $(26, 47)$  и есть тот шифротекст, который необходимо послать  $B$ . В использованных при описании протокола обозначениях  $P_B(k_1 \parallel \hat{A}) = (26, 47)$  и

$$A \longrightarrow B : \quad (26, 47).$$

На втором шаге протокола  $B$  расшифровывает  $(26, 47)$ , используя свой секретный ключ:

$$m' = e \cdot r^{p-1-c_B} \bmod p = 47 \cdot 26^{106-45} \bmod 107 = 31.$$

*B* убеждается, что младшая цифра полученного сообщения содержит идентификационный номер пользователя *A* и извлекает  $k_1 = 3$ . Затем он выбирает свое секретное число, пусть  $k_2 = 7$ , формирует сообщение  $m = k_1 \| k_2 = 37$  и шифрует его на открытом ключе *A*:

$$k = 77, \quad r = g^k \bmod p = 2^{77} \bmod 107 = 63,$$

$$e = m \cdot d_A^k \bmod p = 37 \cdot 58^{77} \bmod 107 = 18.$$

Пара чисел  $(63, 18)$  — это то, что нужно послать пользователю *A*. Т.е.  $P_A(k_1 \| k_2) = (63, 18)$  и

$$A \leftarrow B : (63, 18).$$

На третьем шаге *A* расшифровывает  $(63, 18)$ :

$$m' = e \cdot r^{p-1-c_A} \bmod p = 18 \cdot 63^{106-33} \bmod 107 = 37.$$

*A* убеждается в том, что старшая цифра содержит  $k_1 = 3$  и извлекает  $k_2 = 7$ . Теперь *A* шифрует  $k_2$  для *B*:

$$k = 41, \quad r = g^k \bmod p = 2^{41} \bmod 107 = 82,$$

$$e = m \cdot d_B^k \bmod p = 7 \cdot 28^{41} \bmod 107 = 49,$$

и посыпает *B*

$$A \rightarrow B : (82, 49).$$

На четвертом шаге *B* расшифровывает  $(82, 49)$ :

$$m' = e \cdot r^{p-1-c_B} \bmod p = 49 \cdot 82^{106-45} \bmod 107 = 7.$$

*B* убеждается в том, что он получил свое число  $k_2 = 7$ .

Теперь *A* и *B* могут сформировать общий ключ по заранее оговоренной схеме, например,

$$k = k_1 \oplus k_2 = 3 \oplus 7 = (011)_2 \oplus (111)_2 = (100)_2 = 4.$$

□

## Задачи и упражнения

- 5.1.** Для реализации протокола “ментальный покер” выбраны следующие общие параметры:  $p = 23$ ,  $\hat{\alpha} = 5$ ,  $\hat{\beta} = 7$ ,  $\hat{\gamma} = 14$ . Кроме того, имеются следующие варианты секретных параметров и действий для Алисы и Боба:

- a.  $c_A = 13$ ,  $c_B = 5$ , Алиса перемешивает карты по правилу  $(1, 2, 3) \rightarrow (3, 2, 1)$ , Боб выбирает первое число и использует перестановку  $(1, 2) \rightarrow (2, 1)$ . Алиса выбирает второе из полученных чисел.
- б.  $c_A = 7$ ,  $c_B = 15$ , Алиса перемешивает карты по правилу  $(1, 2, 3) \rightarrow (1, 3, 2)$ , Боб выбирает второе число и использует перестановку  $(1, 2) \rightarrow (1, 2)$ . Алиса выбирает первое из полученных чисел.
- в.  $c_A = 19$ ,  $c_B = 3$ , Алиса перемешивает карты по правилу  $(1, 2, 3) \rightarrow (2, 1, 3)$ , Боб выбирает второе число и использует перестановку  $(1, 2) \rightarrow (2, 1)$ . Алиса выбирает второе из полученных чисел.
- г.  $c_A = 9$ ,  $c_B = 7$ , Алиса перемешивает карты по правилу  $(1, 2, 3) \rightarrow (3, 2, 1)$ , Боб выбирает третье число и использует перестановку  $(1, 2) \rightarrow (1, 2)$ . Алиса выбирает второе из полученных чисел.
- д.  $c_A = 15$ ,  $c_B = 5$ , Алиса перемешивает карты по правилу  $(1, 2, 3) \rightarrow (1, 2, 3)$ , Боб выбирает первое число и использует перестановку  $(1, 2) \rightarrow (2, 1)$ . Алиса выбирает первое из полученных чисел.

Определить, какие карты достанутся Алисе и Бобу. Какие передаваемые числа будет наблюдать Ева?

- 5.2.** В системе электронных денег выбраны секретные параметры банка  $P = 17$ ,  $Q = 7$ ,  $c = 77$ , а соответствующие им открытые параметры  $N = 119$ ,  $d = 5$ . Сформировать электронные банкноты со следующими номерами:

- а.  $n = 11$  при  $r = 5$ ,
- б.  $n = 99$  при  $r = 6$ ,
- в.  $n = 55$  при  $r = 10$ ,
- г.  $n = 44$  при  $r = 15$ ,
- д.  $n = 77$  при  $r = 30$ .

## Темы лабораторных работ

- 5.3. Выполнить компьютерную реализацию протокола “Ментальный покер”, самостоятельно выбрав все необходимые параметры.
- 5.4. Выполнить компьютерную реализацию протокола доказательства с нулевым знанием на основе задачи о раскраске графа, все необходимые параметры выбрать самостоятельно.
- 5.5. Выполнить компьютерную реализацию протокола доказательства с нулевым знанием на основе задачи о гамильтоновом цикле в графе, все необходимые параметры выбрать самостоятельно.
- 5.6. Выполнить компьютерную реализацию протокола “Электронные деньги”, все необходимые параметры выбрать самостоятельно.
- 5.7. Выполнить компьютерную реализацию протокола Нидхама–Шредера, конкретный вид шифра с открытым ключем и все необходимые параметры выбрать самостоятельно.

# Глава 6. КРИПТОСИСТЕМЫ НА ЭЛЛИПТИЧЕСКИХ КРИВЫХ

## 6.1. Введение

В этой главе мы рассмотрим одно из интересных направлений криптографии с открытыми ключами — системы на эллиптических кривых. Эллиптические кривые давно изучались в математике, но их использование в криптографических целях было впервые предложено Коблицом (Neal Koblitz) и Миллером (Victor Miller) в 1985 году. Интенсивные исследования этих систем подтвердили их полезные свойства и привели к открытию множества эффективных методов реализации. С 1998 года использование эллиптических кривых для решения криптографических задач, таких как цифровая подпись, было закреплено в стандартах США ANSI X9.62 и FIPS 186-2, а в 2001 году аналогичный стандарт, ГОСТ Р 34.10-2001, был принят и в России. В настоящее время действуют усовершенствованные варианты этих стандартов — FIPS 186-4 [57] и ГОСТ Р 34.10-2012 [9].

Основное достоинство криптосистем на эллиптических кривых состоит в том, что по сравнению с «обычными» криптосистемами, которые мы рассматривали в предыдущих главах, они обеспечивают существенно более высокую стойкость при равной трудоемкости или, наоборот, существенно меньшую трудоемкость при равной стойкости. Это объясняется тем, что для вычисления обратных функций на эллиптических кривых известны только алгоритмы с экспоненциальным ростом трудоемкости, тогда как для обычных систем предложены субэкспоненциальные методы. В результате тот уровень стойкости, который достигается, скажем, в RSA при использовании 1024-битовых модулей, в системах на эллиптических кривых реализуется при размере модуля 160 бит, что обеспечивает более простую как программную, так и аппаратную реализацию.

Детальное изучение эллиптических кривых требует знаний высшей алгебры, в особенности алгебраической геометрии. Мы, однако, постараемся изложить материал без привлечения сложных алгебра-

ических конструкций и в объеме, достаточном для понимания принципов построения и работы соответствующих криптосистем. Более подробное изложение теории эллиптических кривых и их использования в криптографии может быть найдено, например, в [38, 67, 91].

## 6.2. Математические основы

Кривая третьего порядка  $E$ , задаваемая уравнением вида

$$E : Y^2 = X^3 + aX + b, \quad (6.1)$$

называется эллиптической кривой (на самом деле уравнение (6.1) получено путем замены переменных из более общего уравнения, которое нас не будет интересовать).

Поскольку  $Y = \pm\sqrt{X^3 + aX + b}$ , график кривой симметричен относительно оси абсцисс. Чтобы найти точки его пересечения с осью абсцисс, необходимо решить кубическое уравнение

$$X^3 + aX + b = 0. \quad (6.2)$$

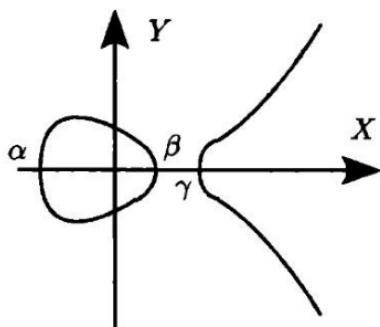
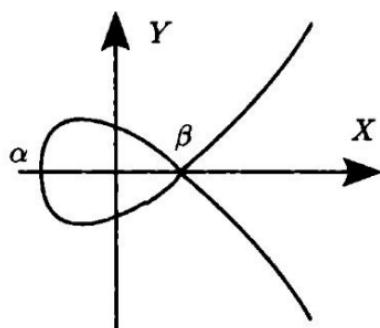
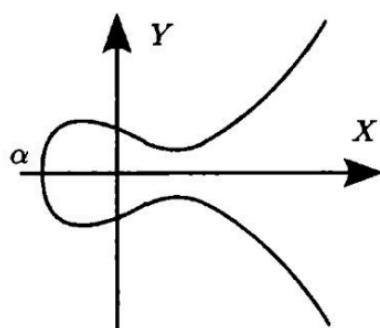
Это можно сделать с помощью известных формул Кардано. Дискриминант этого уравнения

$$D = \left(\frac{a}{3}\right)^3 + \left(\frac{b}{2}\right)^2. \quad (6.3)$$

Если  $D < 0$ , то (6.2) имеет три различных действительных корня  $\alpha, \beta, \gamma$ ; если  $D = 0$ , то (6.2) имеет три действительных корня, скажем,  $\alpha, \beta, \beta$ , по крайней мере два из которых равны; наконец, если  $D > 0$ , уравнение (6.2) имеет один действительный корень  $\alpha$  и два комплексно сопряженных. Вид кривой во всех трех случаях представлен на рисунках 6.1–6.3.

Кривая, представленная на рис. 6.2, называется *сингулярной*. В ее точке сингулярности  $(\beta, 0)$  имеются две касательные. Сингулярные кривые мы будем исключать из нашего рассмотрения. Поэтому при задании кривой с помощью параметров  $a$  и  $b$  потребуем выполнения условия  $D \neq 0$ , что эквивалентно условию

$$4a^3 + 27b^2 \neq 0. \quad (6.4)$$

Рис. 6.1. Эллиптическая кривая  $D < 0$ Рис. 6.2. Эллиптическая кривая  $D = 0$ Рис. 6.3. Эллиптическая кривая  $D > 0$

Итак, пусть эллиптическая кривая  $E$  задана уравнением (6.1) с ограничением на коэффициенты (6.4). Определим операцию композиции точек на кривой. Возьмем какие-либо две точки  $P = (x_1, y_1)$ ,  $Q = (x_2, y_2) \in E$  и проведем через них прямую (рис. 6.4). Эта прямая

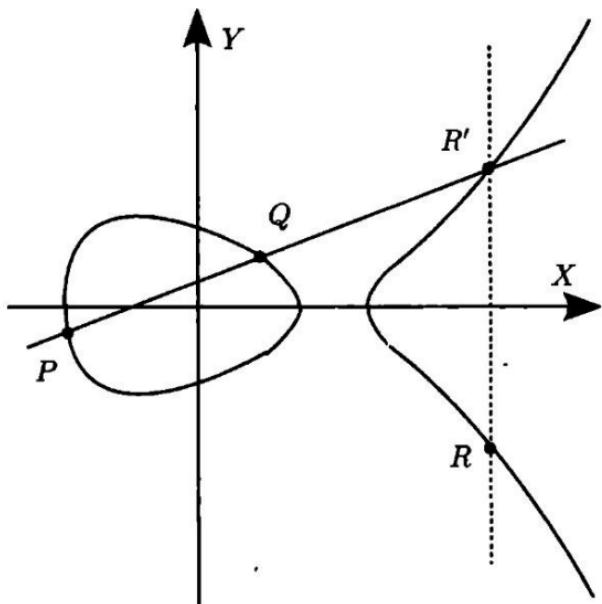


Рис. 6.4. Композиция точек  $R = P + Q$

обязательно пересечет кривую в третьей точке, которую обозначим через  $R'$ . (Третья точка обязательно существует. Дело в том, что кубическое уравнение, получаемое после подстановки уравнения прямой в (6.1), имеет два действительных корня, соответствующих точкам  $P$  и  $Q$ , следовательно, его третий корень, соответствующий  $R'$ , также действителен.) Точку  $R = (x_3, y_3)$  получим путем изменения знака ординаты точки  $R'$ . Будем обозначать описанную операцию композиции точек следующим образом:  $R = P + Q$ .

Пусть точка  $P \in E$  имеет координаты  $(x, y)$ . Тогда точку с координатами  $(x, -y)$  будем обозначать  $-P$ . Будем считать, что вертикальная прямая, проходящая через  $P$  и  $-P$ , пересекает кривую в бесконечно удаленной точке  $\mathcal{O}$ , т.е.  $P + (-P) = \mathcal{O}$ . По соглашению

$P + \mathcal{O} = \mathcal{O} + P = P$ . Как мы увидим в дальнейшем, точка  $\mathcal{O}$  будет играть роль нуля в операциях на эллиптической кривой.

Теперь представим, что точки  $P$  и  $Q$  (рис. 6.4) сближаются друг с другом и, наконец, сливаются в одну точку  $P = Q = (x_1, y_1)$ . Тогда композиция  $R = (x_3, y_3) = P + Q = P + P$  будет получена путем проведения касательной в точке  $P$  и отражения ее второго пересечения с кривой  $R'$  относительно оси абсцисс (рис. 6.5). Будем использовать следующее обозначение:  $R = P + P = [2]P$ .

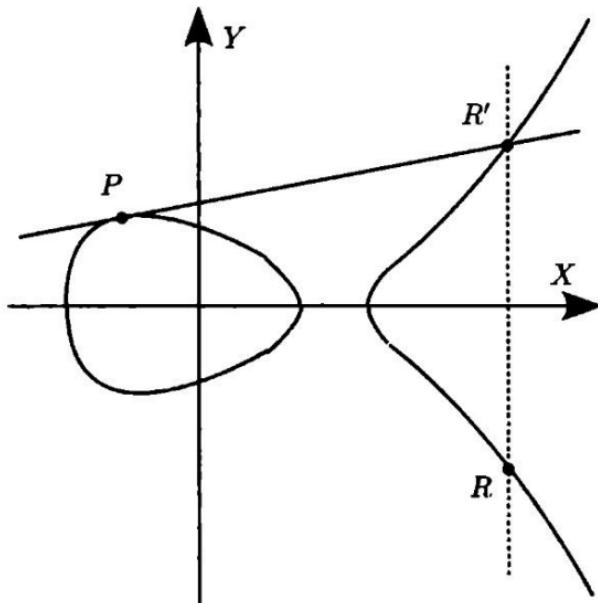


Рис. 6.5. Удвоение точки  $R = P + P = [2]P$

Выведем формулы для определения координат результирующей точки  $R = (x_3, y_3)$  на основе координат исходных точек  $P = (x_1, y_1)$  и  $Q = (x_2, y_2)$ . Рассмотрим вначале случай, когда  $P \neq \pm Q$  (рис. 6.4). Обозначим через  $k$  угловой коэффициент прямой, проходящей через  $P$  и  $Q$ . Очевидно, что

$$k = \frac{y_2 - y_1}{x_2 - x_1}. \quad (6.5)$$

Тогда уравнение прямой будет иметь вид  $Y - y_1 = k(X - x_1)$ , откуда

$$Y = y_1 + k(X - x_1). \quad (6.6)$$

Подставим найденное выражение для переменной  $Y$  в уравнение кривой (6.1). Получим

$$(y_1 + k(X - x_1))^2 = X^3 + aX + b.$$

Возводя в квадрат и группируя подобные члены, получим кубическое уравнение

$$X^3 - k^2 X^2 + \dots = 0.$$

Известно, что сумма корней кубического уравнения равна коэффициенту при  $X^2$ , взятому с противоположным знаком (теорема Виета для кубических уравнений), т.е.

$$x_1 + x_2 + x_3 = k^2,$$

откуда

$$x_3 = k^2 - x_1 - x_2. \quad (6.7)$$

Подставив найденное значение  $x_3$  в уравнение прямой (6.6), найдем ординату точки  $R'$ ,  $y'_3 = y_1 + k(x_3 - x_1)$ , и, изменив знак, получим

$$y_3 = k(x_1 - x_3) - y_1. \quad (6.8)$$

Итак, мы нашли координаты интересующей нас точки  $R = P + Q$ .

Теперь рассмотрим случай, когда  $P = Q$  и результирующая точка  $R = [2]P$  (рис. 6.5). Дифференцируя обе части (6.1) по  $X$ , получим

$$2YY' = 3X^2 + a.$$

Угловой коэффициент касательной равен значению производной в точке  $P$ ,

$$k = \frac{3x_1^2 + a}{2y_1}. \quad (6.9)$$

Дальнейшие рассуждения аналогичны первому случаю, и координаты точки  $R$  определяются по тем же формулам (6.7) и (6.8). Заметим, что если ордината точки  $P$  равна нулю, то касательная проходит параллельно оси ординат и  $[2]P = \mathcal{O}$ .

Используя полученные формулы для вычисления композиции и принятые соглашения относительно точки в бесконечности, можно доказать следующие свойства точек на эллиптической кривой:

- 1)  $P + Q = Q + P$  для всех точек  $P, Q \in E$ ;
- 2)  $P + (Q + S) = (P + Q) + S$  для всех точек  $P, Q, S \in E$ ;
- 3) существует нулевой элемент  $\mathcal{O}$  (точка в бесконечности), такой что  $P + \mathcal{O} = \mathcal{O} + P = P$  для всех  $P \in E$ ;
- 4) для каждой точки  $P \in E$  существует точка  $-P \in E$ , такая что  $P + (-P) = \mathcal{O}$ .

Перечисленные свойства точек совпадают со свойствами целых чисел при использовании операции сложения. Поэтому композицию точек часто называют сложением точек, а операцию  $[2]P$  — удвоением точки.

Продолжая аналогию со сложением чисел, удобно ввести следующие обозначения. Для целого  $m$

$$\begin{aligned}[m]P &= \underbrace{P + P + \cdots + P}_m, \\ [0]P &= \mathcal{O}, \\ [-m]P &= -(\underbrace{P + P + \cdots + P}_m).\end{aligned}$$

Теперь мы готовы к тому, чтобы сделать последний шаг, необходимый для криптографического использования эллиптических кривых. Мы видим, что при вычислении композиции точек на кривой (см. формулы (6.5), (6.9), (6.7) и (6.8)) используются только операции сложения, вычитания, умножения и деления чисел. Это значит, что все приведенные выше тождества сохраняются, если мы будем выполнять вычисления с целыми числами по модулю простого числа  $p$ . В этом случае сложение и умножение чисел выполняются по модулю  $p$ , разность  $u - v$  вычисляется как  $u + (p - v) \bmod p$ , а деление  $u/v$  выполняется путем умножения  $u$  на  $v^{-1} \bmod p$  (простота модуля гарантирует, что для любого положительного числа  $v < p$  существует число  $v^{-1}$ , такое что  $vv^{-1} \bmod p = 1$ ).

В результате мы получаем кривую

$$E : \quad Y^2 = X^3 + aX + b \pmod{p}. \quad (6.10)$$

В уравнении (6.10) переменные  $X, Y$  и коэффициенты  $a, b$  принимают целочисленные значения, а все вычисления выполняются по

модулю  $p$ . В соответствии с (6.4) на  $a, b$  налагается ограничение

$$(4a^3 + 27b^2) \bmod p \neq 0. \quad (6.11)$$

Множество  $E_p(a, b)$  состоит из всех точек  $(x, y)$ ,  $0 \leq x, y < p$ , удовлетворяющих уравнению (6.10), и точки в бесконечности  $\mathcal{O}$ . Количество точек в  $E_p(a, b)$  будем обозначать  $\#E_p(a, b)$ . Эта величина имеет важное значение для криптографических приложений эллиптических кривых.

Пример 6.1. Рассмотрим кривую

$$E_7(2, 6) : \quad Y^2 = X^3 + 2X + 6 \pmod{7}. \quad (6.12)$$

Проверим условие (6.11):

$$4 \cdot 2^3 + 27 \cdot 6^2 = 4 \cdot 1 + 6 \cdot 1 = 3 \neq 0 \pmod{7}.$$

Итак, данная кривая несингулярна. Найдем какую-нибудь (случайную) точку в  $E_7(2, 6)$ . Пусть  $x = 5$ . Тогда

$$Y^2 = 5^3 + 2 \cdot 5 + 6 = 6 + 3 + 6 = 1 \pmod{7}$$

и  $y = 1 \pmod{7}$  или  $y = -1 = 6 \pmod{7}$ . Мы нашли сразу две точки:  $(5, 1)$  и  $(5, 6)$ . Найдем еще пару точек путем вычисления композиции. Вначале найдем  $[2](5, 1)$ . Используя (6.9), (6.7) и (6.8), вычисляем

$$k = \frac{3 \cdot 5^2 + 2}{2 \cdot 1} = \frac{0}{2} = 0 \pmod{7},$$

$$x_3 = 0 - 2 \cdot 5 = 4 \pmod{7},$$

$$y_3 = 0 \cdot (5 - 4) - 1 = 6 \pmod{7}.$$

Мы получили  $[2](5, 1) = (4, 6)$  (можно убедиться, что полученная точка лежит на кривой, подставив ее координаты в уравнение (6.12)). Найдем еще одну точку  $[3](5, 1) = (5, 1) + (4, 6)$ . Используя (6.5), (6.7) и (6.8), вычисляем

$$k = \frac{6 - 1}{4 - 5} = \frac{5}{6} = 5 \cdot 6 = 2 \pmod{7},$$

$$x_3 = 2^2 - 5 - 4 = 2 \pmod{7},$$

$$y_3 = 2 \cdot (5 - 2) - 1 = 2 \cdot 3 - 1 = 5 \pmod{7}.$$

Мы получили  $[3](5, 1) = (2, 5)$ . Итак, мы нашли четыре точки. Для криптографического использования кривой важно знать, сколько всего точек в множестве  $E_7(2, 6)$ . Ответ на этот вопрос мы дадим в разд. 6.6.  $\square$

Скажем несколько слов о свойствах множества точек  $E_p(a, b)$ . Очевидно, что это множество конечно, т.к. в него входят только точки с целочисленными координатами  $0 \leq x, y < p$ . Существует прямая аналогия между  $E_p(a, b)$  и множеством степеней целых чисел, вычисляемых по модулю  $p$ . Так,  $E_p(a, b)$  имеет генератор, т.е. такую точку  $G$ , что ряд  $G, [2]G, [3]G, \dots, [n]G$ , где  $n = \#E_p(a, b)$ , содержит все точки множества  $E_p(a, b)$ , причем  $[n]G = \mathcal{O}$  (аналогично «действовал» параметр  $g$  в разд. 2.2). Число точек на кривой, при надлежащем выборе параметров  $p$ ,  $a$  и  $b$ , может быть простым числом,  $\#E_p(a, b) = q$ . В этом случае любая точка (кроме  $\mathcal{O}$ ) является генератором всего множества точек. Такая кривая предпочтительна во многих отношениях и всегда может быть найдена за практически приемлемое время. Если по каким-то причинам такую кривую найти не удалось и  $\#E_p(a, b) = hq$ , где  $q$  — опять простое число, то в  $E_p(a, b)$  существует подмножество из  $q$  точек (т.е. мощности  $q$ ), генератором которого может служить любая точка  $G \neq \mathcal{O}$ , такая что  $[q]G = \mathcal{O}$ . В дальнейшем, без потери общности, мы будем считать, что работаем с таким подмножеством мощности  $q$  (а при выборе кривой будем стремиться получить  $q = \#E_p(a, b)$ ).

Основная криптографическая операция на эллиптической кривой —  $m$ -кратная композиция, т.е. вычисление

$$Q = [m]P = \underbrace{P + P + \dots + P}_m. \quad (6.13)$$

Эта операция выполняется очень эффективно и требует не более  $2 \log m$  композиций точек. Подходы к ее реализации абсолютно те же, что и к возведению в степень. Например, чтобы получить точку  $Q = [21]P$ , вычисляем  $[2]P$ ,  $[4]P$ ,  $[8]P$ ,  $[16]P$ , каждый раз удваивая предыдущую точку, и складываем  $P + [4]P + [16]P = Q$  (всего 4 удвоения и 2 сложения).

Обратная задача, которая по традиции называется дискретным логарифмированием на эллиптической кривой, формулируется следующим образом. Зная точки  $P$  и  $Q$ , найти такое число  $m$ , что  $[m]P = Q$ . Эта задача оказывается очень трудной. Если тщательно

выбрать параметры кривой (как описывается в следующем разделе), то наилучшие известные в настоящее время алгоритмы для нахождения  $t$  требуют  $O(\sqrt{q})$  операций на кривой, где  $q$  — мощность подмножества точек, которому принадлежат точки  $P$  и  $Q$ . Все вычисления на кривой проводятся по модулю  $p$ , т.е. с числами длины  $t \approx \log p$  бит. Для криптографических приложений  $\log q \approx \log p$ , поэтому  $O(\sqrt{q}) = O(2^{t/2})$  означает экспоненциальный рост трудоемкости при увеличении длины чисел.

### 6.3. Выбор параметров кривой

В этом разделе мы изложим основные рекомендации по выбору параметров эллиптической кривой, предназначеннной для решения криптографических задач, а именно, по выбору коэффициентов  $a$ ,  $b$  и модуля  $p$ . Фактически критерием выбора служит невозможность осуществления определенного рода атак, предложенных для некоторых классов кривых. Излагаемые ниже рекомендации следуют стратегии выбора *случайной* кривой. Эта стратегия считается наиболее надежной с точки зрения обеспечения стойкости результирующей криптосистемы. Альтернативный подход, не рассматриваемый нами, состоит в систематическом конструировании кривой с заданными свойствами, что обычно оказывается более эффективным с вычислительной точки зрения. Для реализации этого подхода предложены специальные методы, но получаемые кривые фактически выбираются из относительно небольшого класса и вызывают подозрения на наличие некоторых специфических свойств, которые могут позволить со временем изобрести алгоритмы для их взлома.

Опишем по шагам процесс формирования хорошей случайной кривой.

1. Выбираем случайно простое число  $p$ . Как будет показано в разд. 6.6, количество точек на кривой — величина того же порядка, что и  $p$ . Поэтому битовая длина числа  $p$ ,  $t = \lfloor \log p \rfloor + 1$ , должна быть такой, чтобы сделать невозможным применение общих методов нахождения логарифмов на кривой, имеющих трудоемкость  $O(2^{t/2})$ . Величина  $t = 128$  бит (четыре машинных слова на 32-битовых компьютерах) сегодня недостаточна, т.к. имеются сообщения о взломе соответствующих кривых за несколько месяцев интенсивных распределенных вычислений.

Но величина  $t = 160$  бит (пять машинных слов) в настоящее время недостаточна для криптоаналитиков и может служить отправной точкой. Другое рассуждение основано на том, что шифр на эллиптической кривой должен быть не менее стойким, чем блочный шифр AES (см. разд. 8.2). Считается, что стойкость AES обеспечивается полной длиной его ключа, которая составляет 128, 196 или 256 бит. Так как стойкость шифра на эллиптической кривой определяется величиной  $t/2$ , длина модулей эллиптических кривых должна составлять соответственно 256, 392 и 512 бит.

2. Выбираем случайные числа  $a$  и  $b$ , такие что  $a, b \neq 0 \pmod{p}$  и  $4a^3 + 27b^2 \neq 0 \pmod{p}$ . Обратим внимание на то, что при вычислении композиции точек параметр  $b$  нигде не фигурирует. Поэтому для повышения эффективности счета иногда рекомендуют случайно выбирать только  $b$ , а  $a$  принимать равным небольшому целому числу. Так, стандарт США FIPS 186 предполагает использование кривых с параметром  $a = -3$ , что, как мы увидим в разд. 6.5, немногого упрощает вычисления.
3. Определяем число точек на кривой  $n = \#E_p(a, b)$  (это самый трудоемкий этап описываемого процесса, его базовый алгоритм будет рассмотрен в разд. 6.6). Важно, чтобы  $n$  имело большой простой делитель  $q$ , а лучше всего само было простым числом,  $n = q$ . Если  $n$  разлагается на маленькие множители, то в  $E_p(a, b)$  существует много маленьких подмножеств со своими генераторами, и алгоритм Полига-Хеллмана [68] быстро вычисляет логарифм на кривой через логарифмы в этих маленьких подмножествах. Если поиск кривой с  $n = q$  занимает слишком много времени, то можно допустить  $n = hq$ , где  $h$  — небольшое число. Еще раз подчеркнем, что стойкость крипtosистемы на эллиптической кривой определяется не модулем  $p$ , а числом элементов  $q$  в подмножестве точек кривой. Но если множитель  $h$  — небольшое число, то  $q$  является величиной того же порядка, что и  $p$ . Если  $n$  не соответствует предъявляемым требованиям, то необходимо вернуться к шагу 2.
4. Проверяем, выполняются ли неравенства  $(p^k - 1) \bmod q \neq 0$  для всех  $k$ ,  $0 < k < 32$ . Если нет, то возвращаемся к шагу 2. Эта проверка предотвращает возможность MOV-атаки (на-

званной по фамилиям ее авторов Menezes, Okamoto, Vanstone), а также исключает из рассмотрения так называемые суперсингулярные кривые и кривые с  $\#E_p(a, b) = p - 1$  (см. [38, 67]). Метод MOV и указанные особые типы кривых позволяют свести задачу вычисления логарифма на кривой к более простым задачам.

5. Проверяем, выполняется ли неравенство  $q \neq p$ . Если нет, то возвращаемся к шагу 2. Дело в том, что для кривых с  $q = p$ , которые называются аномальными, предложены эффективные методы вычисления логарифмов (см. [38, 67]).
6. На данном шаге подходящая для криптографических приложений кривая получена. Мы имеем параметры  $p, a, b$ , число точек  $n$  и размер подмножества точек  $q$ . Обычно еще требуется найти точку  $G$  — генератор этого подмножества. Если  $q = n$ , то любая точка (кроме  $\mathcal{O}$ ) является генератором. Если  $q < n$ , то выбираем случайные точки  $G'$ , пока не получим  $G = [n/q]G' \neq \mathcal{O}$ . Чтобы получить случайную точку на кривой, берем случайное число  $x < p$ , вычисляем  $e = (x^3 + ax + b) \bmod p$  и пытаемся извлечь квадратный корень  $y = \sqrt{e} \bmod p$ . Если корень существует, то получаем точку  $(x, y)$ , в противном случае пробуем другое число  $x$ . Алгоритмы вычисления квадратных корней по модулю простого числа могут быть найдены, например, в [68, гл. 3].

## 6.4. Построение криптосистем

Любая криптосистема, взлом которой основан на дискретном логарифмировании, легко может быть перенесена на эллиптические кривые. Основной принцип построения системы состоит в замене операции  $y = g^x \bmod p$  на  $Y = [x]G \bmod p$  (во втором случае указание модуля не общепринято, хотя реально все вычисления на кривой проводятся по модулю  $p$ ). Переход наиболее непосредствен, когда показатель степени  $x$  приводится по модулю  $q$  (как, например, в стандарте цифровой подписи в разд. 4.3). На эллиптической кривой с мощностью множества точек  $q$  то же самое происходит с множителем  $[x]$ . Отличие состоит в том, что  $y$  — это число, а  $Y$  — точка, и обычно требуется переходить от точки к числу. Наиболее простой способ такого перехода — использовать абсциссу точки.

На эллиптических кривых можно построить и аналог системы RSA. В этом случае кривая определяется по модулю  $N$ , где  $N$  — составное число. Однако здесь не получается выигрыша, т.к. длина модуля остается такой же, как в первоначальном варианте системы RSA, чтобы было невозможно разложить  $N$  на простые множители.

Мы продемонстрируем технику использования эллиптических кривых на примере аналогов шифра Эль-Гамаля и стандартов цифровой подписи.

### Шифр Эль-Гамаля на эллиптической кривой

Для пользователей некоторой сети выбираются общая эллиптическая кривая  $E_p(a, b)$  и точка  $G$  на ней, такие что  $G, [2]G, [3]G, \dots, [q]G$  суть различные точки и  $[q]G = \mathcal{O}$  для некоторого простого числа  $q$ .

Каждый пользователь  $U$  выбирает число  $c_U$ ,  $0 < c_U < q$ , которое хранит как свой секретный ключ, и вычисляет точку на кривой  $D_U = [c_U]G$ , которая будет его открытым ключом. Параметры кривой и список открытых ключей передаются всем пользователям сети.

Допустим, пользователь  $A$  хочет передать сообщение пользователю  $B$ . Будем считать, что сообщение представлено в виде числа  $m < p$ .  $A$  делает следующее:

- 1) выбирает случайное число  $k$ ,  $0 < k < q$ ;
- 2) вычисляет  $R = [k]G$ ,  $P = [k]D_B = (x, y)$ ;
- 3) шифрует  $e = mx \bmod p$ ;
- 4) посыпает  $B$  шифротекст  $(R, e)$ .

Пользователь  $B$ , после получения  $(R, e)$ ,

- 1) вычисляет  $Q = [c_B]R = (x, y)$ ;
- 2) расшифровывает  $m' = ex^{-1} \bmod p$ .

Дадим обоснование протокола. Для этого достаточно показать, что

$$[c_B]R = [c_B]([k]G) = [k]([c_B]G) = [k]D_B,$$

т.е.  $Q = P$ . Поэтому  $m' = m$ .

Координата  $x$  точки  $Q$  остается секретной для противника, т.к. он не знает числа  $k$ . Противник может попытаться вычислить  $k$  из точки  $R$ , но для этого ему нужно решить проблему дискретного логарифмирования на кривой, что считается невозможным.

Наиболее вероятным вариантом использования представленного протокола будет передача в качестве числа  $m$  секретного ключа для блокового или потокового шифра. В этом случае разумно выбирать параметры кривой так, чтобы  $\log q$  примерно вдвое превышал длину ключа шифра.

### Цифровая подпись по ГОСТ Р 34.10-2012

Данный метод полностью аналогичен описанному ранее методу генерации и проверки подписи ГОСТ Р 34.10-94, но возведение в степень заменяется операцией композиции на кривой [9]. Для сообщества пользователей выбирается общая эллиптическая кривая  $E_p(a, b)$  и точка  $G$  на ней, такая что  $G, [2]G, [3]G, \dots, [q]G$  суть различные точки, и  $[q]G = \mathcal{O}$  для некоторого простого числа  $q$  (длина числа  $q$  равна 256 или 512 бит).

Каждый пользователь  $U$  выбирает случайное число  $x_U$  (секретный ключ),  $0 < x_U < q$ , и вычисляет точку на кривой  $Y_U = [x_U]G$  (открытый ключ). Параметры кривой и список открытых ключей передаются всем пользователям.

Чтобы подписать сообщение  $\bar{m}$ , пользователь  $A$  делает следующее:

- 1) вычисляет значение хеш-функции сообщения  $h = h(\bar{m})$  (используется алгоритм Стрибог (ГОСТ Р 34.10-2012) [9]);
- 2) выбирает случайно число  $k$ ,  $0 < k < q$ ;
- 3) вычисляет  $P = [k]G = (x, y)$ ;
- 4) вычисляет  $r = x \bmod q$  (при  $r = 0$  возвращается к шагу 2);
- 5) вычисляет  $s = (kh + rx_A) \bmod q$  (при  $s = 0$  возвращается к шагу 2);
- 6) подписывает сообщение парой чисел  $(r, s)$ .

Для проверки подписанного сообщения  $(\bar{m}; r, s)$  любой пользователь, знающий открытый ключ  $Y_A$ , делает следующее:

- 1) вычисляет  $h = h(\bar{m})$ ;
- 2) убеждается, что  $0 < r, s < q$  (если это не так, подпись отвергается);
- 3) вычисляет  $u_1 = s \cdot h^{-1} \pmod{q}$  и  $u_2 = -r \cdot h^{-1} \pmod{q}$ ;
- 4) вычисляет композицию точек на кривой  $P = [u_1]G + [u_2]Y_A$ ;
- 5) если  $P = \mathcal{O}$  или абсцисса точки  $P \pmod{q} \neq r$ , то подпись отвергается;
- 6) если все проверки были успешны (абсцисса точки  $P \pmod{q} = r$  при  $0 < r, s < q$ , то подпись считается подлинной и принимается).

Доказательство корректности протокола абсолютно аналогично доказательству, приведенному в разд. 4.3.

### Алгоритм ECDSA

Алгоритм ECDSA (Elliptic Curve Digital Signature Algorithm) используется в стандарте цифровой подписи США FIPS 186. Данный алгоритм является предшественником ГОСТ Р 34.10-2012.

Выбор кривой и построение ключей в основном аналогичны процессу, описанному нами выше (следует заметить, что ECDSA разрешает также использование кривых над так называемым бинарным полем, не рассмотренных в нашей книге).

Чтобы подписать сообщение  $\bar{m}$ , пользователь  $A$ :

- 1) вычисляет значение хеш-функции сообщения  $h = h(\bar{m})$  (в различных реализациях ECDSA могут использоваться различные хеш-функции);
- 2) выбирает случайно число  $k$ ,  $0 < k < q$ ;
- 3) вычисляет  $P = [k]G = (x, y)$ ;
- 4) вычисляет  $r = x \pmod{q}$  (при  $r = 0$  возвращается к шагу 2);
- 5) вычисляет  $s = k^{-1}(h + rx_A) \pmod{q}$  (при  $s = 0$  возвращается к шагу 2);
- 6) подписывает сообщение парой чисел  $(r, s)$ .

Для проверки подписанного сообщения  $(\bar{m}; r, s)$  любой пользователь:

- 1) вычисляет  $h = h(\bar{m})$ ;
- 2) убеждается, что  $0 < r, s < q$  (если это не так, подпись отвергается);
- 3) вычисляет  $u_1 = h \cdot s^{-1} \pmod{q}$  и  $u_2 = r \cdot s^{-1} \pmod{q}$ ;
- 4) вычисляет композицию точек на кривой  $P = [u_1]G + [u_2]Y_A$ ;
- 5) если  $P = \mathcal{O}$  или абсцисса точки  $P \pmod{q} \neq r$ , то подпись отвергается;
- 6) если все проверки были успешны (абсцисса точки  $P \pmod{q} = r$  при  $0 < r, s < q$ ), то подпись считается подлинной и принимается.

Как видим, отличия от ГОСТ Р 34.10-2012 минимальны (на самом деле мы уже говорили об этих отличиях в конце разд. 4.3).

## 6.5. Эффективная реализация операций

В этом разделе мы рассмотрим основной подход, обычно необходимый для эффективной реализации операций на эллиптической кривой. Как уже говорилось, вычисление  $m$ -кратной композиции ( $m$  — большое число) осуществляется с помощью тех же методов, которые используются для возведения в степень (умножение в них заменяется на сложение точек). Здесь для конкретности мы все же опишем наиболее простой алгоритм — двоичный левосторонний метод (ср. с алгоритмом 2.4). В литературе описаны более эффективные методы возведения в степень, которые с успехом могут быть применены и для вычисления композиций на эллиптических кривых. Эти методы обычно дают несущественный выигрыш во времени (порядка 25%), и их описание выходит за рамки нашей книги (см. описание этих методов, например, в [68]).

**Алгоритм 6.1. Вычисление  $t$ -кратной композиции**ВХОД: Точка  $P$ , число  $t = (m_t m_{t-1} \dots m_1)_2$ .ВЫХОД:  $Q = [t]P$ .

- (1)  $Q \leftarrow \mathcal{O}$ ;
- (2) FOR  $i = t, t-1, \dots, 1$  DO
- (3)      $Q \leftarrow [2]Q$ ,
- (4)     IF  $m_i = 1$  THEN  $Q \leftarrow Q + P$ ;
- (5) RETURN  $Q$ .

Данный алгоритм требует не более  $t$  сложений и  $t$  удвоений точек (в среднем  $t$  удвоений и  $t/2$  сложений).

Пример 6.2. Проиллюстрируем работу алгоритма на примере вычисления  $[21]P$ . Здесь  $21 = (10101)_2$ ,  $t = 5$ . Покажем, что происходит на каждой итерации цикла алгоритма.

- $$\begin{aligned} [i = 5 \ m_5 = 1] : \quad Q &\leftarrow \mathcal{O}, & Q &\leftarrow Q + P = P; \\ [i = 4 \ m_4 = 0] : \quad Q &\leftarrow [2]Q = [2]P; \\ [i = 3 \ m_3 = 1] : \quad Q &\leftarrow [2]Q = [4]P, & Q &\leftarrow Q + P = [5]P; \\ [i = 2 \ m_2 = 0] : \quad Q &\leftarrow [2]Q = [10]P; \\ [i = 1 \ m_1 = 1] : \quad Q &\leftarrow [2]Q = [20]P, & Q &\leftarrow Q + P = [21]P. \end{aligned}$$

□

Напомним, что  $P_3 = P_1 + P_2$ , где  $P_1 \neq \pm P_2$  (строка 4 алгоритма) вычисляется с помощью формул

$$\begin{aligned} k &= \frac{y_2 - y_1}{x_2 - x_1}, \\ x_3 &= k^2 - x_1 - x_2, \\ y_3 &= k(x_1 - x_3) - y_1 \pmod{p}. \end{aligned} \tag{6.14}$$

Удвоение  $P_3 = [2]P_1$  (строка 3 алгоритма) вычисляется с помощью формул

$$\begin{aligned} k &= \frac{3x_1^2 + a}{2y_1}, \\ x_3 &= k^2 - 2x_1, \\ y_3 &= k(x_1 - x_3) - y_1 \pmod{p}. \end{aligned} \tag{6.15}$$

Говорят, что при вычислениях по формулам (6.14) и (6.15) используется аффинное представление для точек, т.е.  $P = (x, y)$ . Точка

в бесконечности  $\mathcal{O}$  не имеет такого представления. Фактически в алгоритме 6.1 она является просто «флажком», показывающим, что нужно пропустить операции удвоения  $Q \leftarrow [2]Q$  до первого сложения  $Q \leftarrow Q + P$ , которое выполняется как простое присваивание  $Q \leftarrow P$ .

Вычисления по формулам (6.14) и (6.15) были нами рассмотрены в примере 6.1.

Обозначим через  $M$  и  $I$  стоимость (время) умножения и инверсии по модулю  $p$ . Тогда из (6.14) и (6.15) следует, что при аффинном представлении стоимость сложения точек равна  $1I + 3M$ , а стоимость удвоения равна  $1I + 4M$  (операции сложения и умножения на маленькие коэффициенты заметного влияния на время не оказывают).

Соотношение стоимости инверсии и умножения может быть различным в зависимости от реализации, но  $I$  всегда больше  $M$ . Если умножение реализуется за счет сложений и сдвигов, то оно, вероятнее всего, будет незначительно (в 2–3 раза) опережать инверсию. В этом случае использование аффинного представления для точек на кривой вполне оправдано. Однако, если процессор, на котором выполняются вычисления, имеет встроенный параллельный умножитель (как, например, Pentium®), то инверсия будет вычисляться существенно медленнее произведения. Можно дать грубую оценку стоимости соответствующих вычислений. Если  $t$  — длина чисел в битах, то умножение потребует порядка  $(t/32)^2 = t^2/1024$  машинных операций (32 бита — размер слова). С другой стороны, количество итераций в обобщенном алгоритме Евклида, вычисляющем инверсию, пропорционально  $t$  по своей природе. Даже если мы сможем на каждой итерации реализовать линейные по  $t$  вычисления с помощью 32-битовых машинных операций (что возможно), мы получим общую трудоемкость порядка  $t(t/32) = t^2/32$ , т.е. одна инверсия будет соответствовать тридцати двум умножениям. В действительности на каждой итерации алгоритма Евклида нужно делать несколько операций, а умножение может быть реализовано более быстрыми методами, так что  $I > 32M$ .

Мы можем избавиться от инверсий на каждом шаге алгоритма 6.1, если будем работать с координатами в виде рациональных чисел, производя вычисления отдельно с числителем и знаменателем.

Наиболее выгодной оказывается следующая замена переменных:

$$x \rightarrow \frac{X}{Z^2}, \quad y \rightarrow \frac{Y}{Z^3}. \quad (6.16)$$

В этом случае точка на кривой представляется тройкой  $(X, Y, Z)$ , и говорят о переходе к взвешенному проективному представлению (далее для простоты будем опускать слово «взвешенное»). Переход от аффинной точки к проективной делается очень просто:

$$(x, y) \rightarrow (x, y, 1). \quad (6.17)$$

После этого все вычисления проводятся в проективных координатах (без вычисления инверсий). Обратный переход от проективного к аффинному представлению осуществляется следующим образом:

$$(X, Y, Z) \rightarrow (X/Z^2, Y/Z^3), \quad (6.18)$$

и стоит  $1I + 4M$  (одно вычисление инверсии  $Z^{-1}$ , два умножения для получения  $Z^{-2}$  и  $Z^{-3}$  и, наконец, два умножения  $XZ^{-2}$  и  $YZ^{-3}$ ).

Получим формулы для сложения точек в проективном представлении. Вначале в выражении (6.14) для  $x_3$  произведем замену переменных в соответствии с (6.16). После приведения к общему знаменателю и сокращения дроби получим

$$x_3 = \frac{X_3}{Z_3^2} = \frac{(Y_2 Z_1^3 - Y_1 Z_2^3)^2 - (X_1 Z_2^2 + X_2 Z_1^2)(X_2 Z_1^2 - X_1 Z_2^2)^2}{Z_1^2 Z_2^2 (X_2 Z_1^2 - X_1 Z_2^2)^2}. \quad (6.19)$$

Отсюда находим выражения для  $X_3$  и  $Z_3$ , взяв соответственно числитель и квадратный корень из знаменателя правой части (6.19). Чтобы получить выражение для  $Y_3$ , прибегнем к хитрости, позволяющей в конечном итоге сэкономить одну операцию умножения. Заметим, что  $P_1 + P_2 = P_2 + P_1$  и

$$y_3 = \left( (x_1 - x_3) \frac{y_2 - y_1}{x_2 - x_1} - y_1 + (x_2 - x_3) \frac{y_2 - y_1}{x_2 - x_1} - y_2 \right) / 2.$$

Произведем замену переменных по правилу (6.16), причем вместо  $Z_3$  используем его представление из (6.19). После приведения к общему

знаменателю, сокращения и приведения подобных членов получаем

$$y_3 = \frac{Y_3}{Z_3^3} = \frac{((X_1 Z_2^2 + X_2 Z_1^2)(X_2 Z_1^2 - X_1 Z_2^2)^2 - 2X_3)(Y_2 Z_1^3 - Y_1 Z_2^3)}{2Z_1^3 Z_2^3 (X_2 Z_1^2 - X_1 Z_2^2)^3} - \frac{(Y_1 Z_2^3 + Y_2 Z_1^3)(X_2 Z_1^2 - X_1 Z_2^2)^3}{2Z_1^3 Z_2^3 (X_2 Z_1^2 - X_1 Z_2^2)^3}. \quad (6.20)$$

На основании выражений (6.19) и (6.20) запишем следующий алгоритм.

### Алгоритм 6.2. СЛОЖЕНИЕ В ПРОЕКТИВНОМ ПРЕДСТАВЛЕНИИ

ВХОД:  $P_1 = (X_1, Y_1, Z_1)$ ,  $P_2 = (X_2, Y_2, Z_2)$ ,

$P_1, P_2 \neq \mathcal{O}$ ,  $P_1 \neq \pm P_2$

ВЫХОД:  $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

(все вычисления выполняются по модулю  $p$ )

$$\begin{array}{ll}
 \lambda_1 = X_1 Z_2^2 & 2M \\
 \lambda_2 = X_2 Z_1^2 & 2M \\
 \lambda_3 = \lambda_2 - \lambda_1 & \\
 \lambda_4 = Y_1 Z_2^3 & 2M \\
 \lambda_5 = Y_2 Z_1^3 & 2M \\
 \lambda_6 = \lambda_5 - \lambda_4 & \\
 \lambda_7 = \lambda_1 + \lambda_2 & \\
 \lambda_8 = \lambda_4 + \lambda_5 & \\
 Z_3 = Z_1 Z_2 \lambda_3 & 2M \\
 X_3 = \lambda_6^2 - \lambda_7 \lambda_3^2 & 3M \\
 \lambda_9 = \lambda_7 \lambda_3^2 - 2X_3 & \\
 Y_3 = (\lambda_9 \lambda_6 - \lambda_8 \lambda_3^3)/2 & \frac{3M}{16M}
 \end{array}$$

Как видно из описания алгоритма, трудоемкость сложения в проективных координатах равна  $16M$ . Мы не учитываем простые операции сложения, вычитания, а также умножения и деления на 2 (чтобы пояснить, как вычисляется  $v/2 \bmod p$ , заметим, что если  $v$  четно, то  $v/2 \bmod p = v/2$  (сдвиг вправо на один бит); если  $v$  нечетно, то  $v/2 \bmod p = (v + p)/2 \bmod p$ ,  $v + p$  четно, поэтому  $v/2 \bmod p = (v + p)/2$ ). Если одна из точек, скажем  $P_2$ , задана в

аффинных координатах, т.е.  $Z_2 = 1$ , то стоимость сложения снижается до  $11M$ . Это называется смешанным сложением. Алгоритм 6.1 построен таким образом, что он всегда использует смешанные сложения, т.е. стоимость шага 4 алгоритма равна  $11M$ .

Пример 6.3. Рассмотрим вычисление суммы  $(5, 1) + (4, 6)$  в проективных координатах для кривой из примера 6.1 (все вычисления проводятся по модулю 7).

$$P_1 = (5, 1, 1), \quad P_2 = (4, 6, 1).$$

$$\lambda_1 = 5 \cdot 1 = 5,$$

$$\lambda_2 = 4 \cdot 1 = 4,$$

$$\lambda_3 = 4 - 5 = 6,$$

$$\lambda_4 = 1 \cdot 1 = 1,$$

$$\lambda_5 = 6 \cdot 1 = 6,$$

$$\lambda_6 = 6 - 1 = 5,$$

$$\lambda_7 = 5 + 4 = 2,$$

$$\lambda_8 = 1 + 6 = 0,$$

$$Z_3 = 1 \cdot 1 \cdot 6 = 6,$$

$$X_3 = 5^2 - 2 \cdot 6^2 = 2,$$

$$\lambda_9 = 2 \cdot 6^2 - 2 \cdot 2 = 2 - 4 = 5,$$

$$Y_3 = (5 \cdot 5 - 0 \cdot 6^3)/2 = 25/2 = (25 + 7)/2 = 16 = 2;$$

$$P_3 = (2, 2, 6).$$

Для проверки переведем точку  $P_3$  в аффинное представление. Для этого вычислим

$$6^{-1} = 6, \quad 6^{-2} = 6 \cdot 6 = 1, \quad 6^{-3} = 1 \cdot 6 = 6.$$

В итоге получаем

$$P_3 = (2 \cdot 1, 2 \cdot 6) = (2, 5),$$

что совпадает с результатом примера 6.1.  $\square$

Заметим, что условие  $P_1, P_2 \neq \mathcal{O}$  эквивалентно  $Z_1, Z_2 \neq 0$ , а условие  $P_1 \neq \pm P_2$  эквивалентно  $X_1 \neq X_2$ . Если вычисления на кривой проводятся, как это предполагалось нами, в подмножестве точек мощности  $q$ , причем  $q$  — простое число, и  $m \neq 0 \pmod{q}$ , то условия для сложения на шаге 4 алгоритма 6.1 автоматически выполняются (за исключением самого первого сложения, которое заменяется присваиванием).

Для удвоения точки в проективных координатах делаем замену (6.16) в (6.15):

$$x_3 = \frac{X_3}{Z_3^2} = \frac{(3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2}{(2Z_1Y_1)^2}, \quad (6.21)$$

$$y_3 = \frac{Y_3}{Z_3^3} = \frac{(4X_1Y_1^2 - X_3)(3X_1^2 + aZ_1^4)Z_1^2 - 8Y_1^4}{(2Z_1Y_1)^3}, \quad (6.22)$$

что приводит к следующему алгоритму.

### Алгоритм 6.3. УДВОЕНИЕ В ПРОЕКТИВНОМ ПРЕДСТАВЛЕНИИ

ВХОД:  $P_1 = (X_1, Y_1, Z_1)$ ,  $P_1 \neq \mathcal{O}$

ВЫХОД:  $P_3 = (X_3, Y_3, Z_3) = [2]P_1$

$$\begin{array}{rcl}
 \lambda_1 &= 3X_1^2 + aZ_1^4 & 4M \\
 \lambda_2 &= 4X_1Y_1^2 & 2M \\
 Z_3 &= 2Y_1Z_1 & 1M \\
 X_3 &= \lambda_1^2 - 2\lambda_2 & 1M \\
 \lambda_3 &= 8Y_1^4 & 1M \\
 Y_3 &= \lambda_1(\lambda_2 - X_3) - \lambda_3 & \frac{1M}{10M} \\
 \end{array}$$

Мы видим, что стоимость удвоения в проективных координатах в общем случае равна  $10M$ . Однако, если параметр кривой  $a = -3$ , то  $\lambda_1 = 3(X_1 - Z_1^2)(X_1 + Z_1^2)$ , и стоимость удвоения уменьшается до  $8M$ .

Отметим, что в вышеизложенных алгоритмах все вычисления проводятся по модулю  $p$ . Поэтому многое зависит от эффективной реализации модульной арифметики. Для случайно выбранных модулей наилучший на сегодняшний день подход — использование для чисел представления Монтгомери (см., например, [68]). В представлении Монтгомери умножение по модулю  $p$  эквивалентно двум обычным умножениям.

## 6.6. Определение количества точек на кривой

В этом разделе мы рассмотрим алгоритм Схоофа (René Schoof, 1985) для определения  $\#E_p(a, b)$ , т.е. количества точек, координаты ко-

торых удовлетворяют уравнению кривой (6.10) и являются неотрицательными целыми числами, меньшими  $p$ . Алгоритм Схоффа был первым полиномиальным алгоритмом подсчета количества точек на эллиптической кривой, его трудоемкость составляет  $O(\log^6 p)$  операций по модулю  $p$ . Этот алгоритм лежит в основе всех современных методов, применяемых для случайных кривых.

Мы начнем с теоремы, доказанной Хассе (Helmut Hasse) еще в 1933 году.

**Теорема 6.1** (Хассе).  $\#E_p(a, b)$  удовлетворяет неравенствам

$$p + 1 - 2\sqrt{p} \leq \#E_p(a, b) \leq p + 1 + 2\sqrt{p}.$$

Оказывается удобным представлять  $\#E_p(a, b)$  в виде

$$\#E_p(a, b) = p + 1 - t. \quad (6.23)$$

Параметр  $t$  в (6.23) может принимать как положительные, так и отрицательные значения или быть равным нулю и называется следом Фробениуса для  $E_p(a, b)$ . В соответствии с теоремой Хассе

$$|t| \leq 2\sqrt{p}. \quad (6.24)$$

До сих пор нас интересовали только точки на кривой с целочисленными неотрицательными координатами, меньшими  $p$ . Но теперь мы рассмотрим все множество решений уравнения кривой (6.10). След Фробениуса замечательным образом связан с модулем  $p$ .

**Теорема 6.2.** Для всех чисел  $x$  и  $y$ , удовлетворяющих уравнению кривой (6.10) с параметрами  $0 \leq a, b < p$ , справедливо равенство

$$(x^{p^2}, y^{p^2}) + [p](x, y) = [t](x^p, y^p) \quad (6.25)$$

(сложение в формуле означает композицию точек на кривой).

Чтобы научиться находить общее решение уравнения (6.25), заметим, что композиция точек вида  $Q = [m]P$  может быть выражена через координаты точки  $P$ . Например,

$$[2](x, y) = \left( x' = \left( \frac{3x^2 + a}{2y} \right)^2 - 2x, y' = \frac{3x^2 + a}{2y}(x - x') - y \right) =$$

$$= \left( \frac{x^4 - 2ax^2 - 8bx + a^2}{4y^2}, \frac{x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3}{8y^3} \right).$$

Точка  $[3](x, y)$  может быть получена как  $[2](x, y) + (x, y)$  путем подстановки найденных выражений для координат точки  $[2](x, y)$  в формулы (6.5), (6.7), (6.8) (желающие могут это проделать). Процесс получения выражений для следующих точек кажется довольно сложным, тем не менее он описывается следующей простой рекурсивной схемой.

$$\begin{aligned}\psi_0 &= 0, \\ \psi_1 &= 1, \\ \psi_2 &= 2y, \\ \psi_3 &= 3x^4 + 6ax^2 + 12bx - a^2, \\ \psi_4 &= 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3), \\ \psi_{2m+1} &= \psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3, \quad m \geq 2, \\ \psi_{2m} &= (\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2)\psi_m/2y, \quad m > 2.\end{aligned}$$

Для  $m \geq 2$  и  $P = (x, y)$

$$[m]P = \left( \frac{\psi_m^2 x - \psi_{m-1}\psi_{m+1}}{\psi_m^2}, \frac{\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2}{4y\psi_m^3} \right). \quad (6.26)$$

Полином  $\psi_m(x, y)$  называется полиномом деления порядка  $m$ . Поскольку точка  $[m]P$  лежит на кривой, приводимой по модулю  $p$ , вычисление коэффициентов полиномов  $\psi$  достаточно также выполнять по модулю  $p$ . С использованием приведенной выше рекурсивной схемы полином деления порядка  $m$  может быть вычислен за  $O(\log m)$  шагов (читатель, не знакомый с арифметикой полиномов, может обратиться к [13]). Заметим, что при нечетном  $m$  полиномы  $\psi_m$  зависят только от одной переменной  $x$ , т.к. вторая переменная  $y$  входит в них только в четных степенях, а  $y^2$  заменяется правой частью уравнения кривой (6.10). Заметим также, что полиномы «второго слоя» ( $\psi_{2m}, \psi_{2m+1}$ ) включают в себя произведения четырех полиномов «первого слоя» (от  $\psi_{m-2}$  до  $\psi_{m+2}$ ), поэтому степень полинома  $\psi_m$  растет как  $O(m^2)$ .

Точка  $P$  на кривой  $E$  называется торсионной точкой порядка  $m$ , если  $[m]P = \mathcal{O}$ . Вследствие (6.26) достаточно очевидно, что точка

$P = (x, y)$  является торсионной порядка  $m$  тогда и только тогда, когда  $\psi_m(x, y) = 0$ .

Идея алгоритма Схоофа состоит в нахождении решений уравнения (6.25) (относительно  $t$ ) на множествах торсионных точек малых порядков с последующим вычислением общего решения. Для торсионных точек порядка  $m$  множители приводятся по модулю  $m$ , а полиномы — по модулю  $\psi_m$ , так что уравнение (6.25) принимает вид

$$(x^{p^2}, y^{p^2})_m + [p_m](x, y) = [t_m](x^p, y^p)_m, \quad (6.27)$$

где  $p_m = p \bmod m$ ,  $t_m = t \bmod m$ ,  $(x^k, y^k)_m = (x^k \bmod \psi_m, y^k \bmod \psi_m)$ , кроме того,  $x$  и  $y$  связаны уравнением кривой  $y^2 = x^3 + ax + b$ . Возьмем в качестве модулей  $m$  простые числа вплоть до некоторого  $m_{\max}$ , такого что

$$\prod_{\substack{m \text{ простое} \\ 2 \leq m \leq m_{\max}}} m > 4\sqrt{p}. \quad (6.28)$$

Тогда по найденным числам  $t_m$  мы однозначно восстановим след Фробениуса  $t$ , удовлетворяющий (6.24), по китайской теореме об остатках (см. [13, 68]).

Рассмотрим кратко метод решения уравнения (6.27) для случая  $m > 2$ . Вначале вычисляем полином  $\psi_m$ . Так как  $m$  нечетно, то  $\psi_m = \psi_m(x)$ . (Далее все действия с полиномами производятся по модулю  $\psi_m$ , степень  $y$  понижается до единицы с помощью уравнения кривой, коэффициенты вычисляются по модулю  $p$ .) Вычисляем полиномы  $x^p, y^p, x^{p^2}, y^{p^2}$ , используя тот же алгоритм возведения в степень, что и для целых чисел. По формуле (6.26) вычисляем  $Q = [p_m](x, y)$  и, используя формулы сложения на кривой, находим в символьическом виде композицию точек  $R = (x^{p^2}, y^{p^2})_m + Q$ . Если  $R = \mathcal{O}$ , то  $t_m = 0$ . В противном случае, чтобы найти  $t_m$ , вычисляем абсциссу точки  $P = [\tau](x^p, y^p)_m$  для всех  $\tau$ ,  $0 < \tau < m$ . Для каждого значения  $\tau$  нужно проверить равенство  $x_R$  и  $x_P$ . В общем случае разность абсцисс представляется в виде  $x_R - x_P = u(x) - yv(x) = 0$ . Берем отсюда  $y = u(x)/v(x)$ , подставляем в уравнение кривой и получаем некоторый полином  $h_x(x) = 0$ . Если  $h_x \bmod \psi_m \neq 0$ , то  $x_R$  и  $x_P$  не равны, и нужно пробовать другое значение  $\tau$ . Если же  $x_R = x_P$ , вычисляем ординату точки  $P$  и, пользуясь аналогичными приемами, переводим разность  $y_R - y_P$  в полином  $h_y(x) = 0$ . Если  $h_y \bmod \psi_m = 0$ , то  $t_m = \tau$ , иначе  $t_m = -\tau \bmod m$ .

Когда  $m = 2$ ,  $\psi_2 = y$  и возникает затруднение с вычислением  $x^p \bmod y$ . Однако в этом случае помогает одно простое рассуждение. Так как мы исключаем сингулярные кривые, то приведенная по модулю  $p$  эллиптическая кривая может иметь одно или три пересечения с осью  $x$ . Все другие точки идут парами  $(x, y), (x, -y)$ , и есть одна точка в бесконечности  $\mathcal{O}$ . Таким образом, количество точек на кривой четно или нечетно в зависимости от того, разлагается ли  $X^3 + aX + b$  на множители по модулю  $p$  или нет. Известен простой критерий неразложимости полинома по модулю  $p$  (см., например, [68, 4]).

**Теорема 6.3.** Полином третьей степени  $F(X)$  неразложим на множители по модулю  $p$  тогда и только тогда, когда

$$\gcd(F(X), X^p - X) = 1.$$

В результате имеем  $t_2 = 1$ , если  $\gcd(X^3 + aX + b, X^p - X) = 1$ , и  $t_2 = 0$  в противном случае.

Оценим трудоемкость алгоритма Схоофа. Вначале сформулируем следующее известное свойство простых чисел.

**Утверждение 6.4** (о простых числах). Количество простых чисел, меньших  $n$ , примерно равно  $n/\ln n$ .

(Точная формулировка и доказательство могут быть найдены в [3].) Из утверждения 6.4 следует, что  $m_{\max} = O(\log p)$  и количество модулей, для которых ведутся вычисления, равно  $O(\log p)$ . Наиболее трудоемкая операция в алгоритме — вычисление  $x^{p^2}$  и других аналогичных полиномов. При использовании быстрых алгоритмов возведения в степень эти вычисления требуют  $O(\log p)$  операций типа умножения с полиномами степени  $O(m^2) = O(\log^2 p)$ . Каждая такая операция требует  $O(\log^4 p)$  операций типа умножения по модулю  $p$ , т.е. всего получается  $O(\log^5 p)$  операций по модулю  $p$ . Одна операция по модулю  $p$  выполняется с помощью  $O(\log^2 p)$  битовых операций. Таким образом, вычисление  $x^{p^2}$  требует  $O(\log^7 p)$  битовых операций. Принимая во внимание число различных модулей, для которых нужно вычислять  $x^{p^2}$ , получаем общую трудоемкость  $O(\log^8 p)$  битовых операций.

**Пример 6.4.** Определим количество точек на кривой, использованной в примере 6.1 на стр. 104:

$$Y^2 = X^3 + 2X + 6 \pmod{7} \quad (a = 2, b = 6).$$

Вначале воспользуемся алгоритмом, у которого время вычислений растет экспоненциально. Будем задавать значения  $X$  от 0 до 6 и вычислять соответствующие им значения  $Y$ . Запишем вначале таблицу квадратов по модулю 7:

$$\begin{aligned}0^2 &= 0, \\1^2 &= 1, \\2^2 &= 4, \\3^2 &= 9 = 2, \\4^2 &= 2, \\5^2 &= 4, \\6^2 &= 1 \pmod{7}.\end{aligned}$$

Используя эту таблицу, найдем множество точек  $E_7(2, 6)$ :

$$\begin{array}{lll}x = 0, & y^2 = 6, & y \text{ не существует} \\x = 1, & y^2 = 1 + 2 + 6 = 9 = 2, & y = 3 \text{ и } y = -3 = 4, \\x = 2, & y^2 = 8 + 4 + 6 = 4, & y = 2 \text{ и } y = -2 = 5, \\x = 3, & y^2 = 27 + 6 + 6 = 4, & y = 2 \text{ и } y = -2 = 5, \\x = 4, & y^2 = 64 + 8 + 6 = 1, & y = 1 \text{ и } y = -1 = 6, \\x = 5, & y^2 = 125 + 10 + 6 = 1, & y = 1 \text{ и } y = -1 = 6, \\x = 6, & y^2 = 216 + 12 + 6 = 3, & y \text{ не существует}.\end{array}$$

Подсчитывая количество полученных точек и добавляя к ним точку в бесконечности, получаем

$$\#E_7(2, 6) = 11.$$

Ясно, что этот метод не годится при больших  $p$ , но мы будем использовать полученное значение для проверки.

Приступим к выполнению алгоритма Схоофа. Нам достаточно будет трех модулей  $m = 2, 3, 5$ , т.к.  $2 \cdot 3 \cdot 5 = 30 > 4\sqrt{7} = 10.58$  (на самом деле двух модулей  $m = 3, 5$  было бы достаточно, но мы используем еще  $m = 2$  для демонстрации алгоритма).

В целях упрощения обозначений договоримся записывать полиномы в виде десятичных чисел (величина модуля позволяет нам это сделать). Так, например,

$$\begin{aligned}x^4 + 5x^2 + 2 &= 1x^4 + 0x^3 + 5x^2 + 0x + 2 = 10502, \\x^4 + x &= 1x^4 + 0x^3 + 0x^2 + 1x + 0 = 10010.\end{aligned}$$

Все вычисления с коэффициентами полиномов мы будем проводить по модулю 7. Вначале вычислим необходимые полиномы деления:

$$\begin{aligned}\psi_0 &= 0, \\ \psi_1 &= 1, \\ \psi_2 &= 2y, \\ \psi_3 &= 30523, \\ \psi_4 &= 4y \cdot 1031115 = 4054446y, \\ \psi_5 &= \psi_4\psi_2^3 - \psi_1\psi_3^3 = 4054446 \cdot 1026^2 - 6025554626356 = \\ &= 5055036550230.\end{aligned}$$

Решим уравнение (6.27) для  $m = 3$ . Вычисляем по модулю  $\psi_3$ :

$$\begin{aligned}x^7 &= 1363, \\ y^7 &= (y^2)^3y = 1026^3y = 1360y, \\ x^{49} &= 10 = x, \\ y^{49} &= 1026^{24}y = 6y, \\ p_3 &= 7 \bmod 3 = 1.\end{aligned}$$

Левая часть (6.27) превращается в

$$R = (x, 6y) + (x, y) = \mathcal{O}.$$

Значит,  $t_3 = 0$ .

Теперь решим уравнение (6.27) для  $m = 5$ . Вычисляем по модулю  $\psi_5$ :

$$\begin{aligned}x^7 &= 10000000, \\ y^7 &= 1064524266y, \\ x^{49} &= 531353334500, \\ y^{49} &= 650465522521y, \\ p_5 &= 2.\end{aligned}$$

Находим точку  $Q = [2](x, y)$ . В общем случае это делается по формуле (6.26):

$$Q = \left( \frac{4y^2x - \psi_3}{4y^2}, \frac{\psi_4}{4y^4} \right) = \left( \frac{10314}{4013}, \frac{1031115y}{1045431} \right).$$

Видим, что  $x_Q \neq x^{49}$ , поэтому будем искать  $\tau > 0$ . Найдем точку  $R = (x^{49}, y^{49}) + Q$  по формулам (6.5), (6.7), (6.8):

$$k = \frac{\frac{1031115y}{1045431} - 650465522521y}{\frac{10314}{4013} - 531353334500} =$$

$$= \frac{(1031115 - 650465522521 \cdot 1045431)4013y}{1045431(10314 - 531353334500 \cdot 4013)} =$$

$$= \frac{541024434205y}{115461562234},$$

$$x_R = \frac{541024434205^2 y^2}{115461562234^2} - 531353334500 - \frac{10314}{4013} =$$

$$= \frac{552631612401}{533030166456},$$

$$y_R = \left( 531353334500 - \frac{552631612401}{533030166456} \right) \cdot \frac{541024434205y}{115461562234} -$$

$$- 650465522521y = \frac{515441613166y}{115165441243}.$$

Пробуем  $\tau = 1$ ,  $P = (x^7, y^7)$ . Проверяем гипотезу  $x_R - x_P = 0$ :

$$h_x = 552631612401 - 533030166456 \cdot 10000000 =$$

$$= 61115566241 \neq 0 \bmod \psi_5.$$

Пробуем  $\tau = 2$ . Вычисляем  $P = [2](x^7, y^7)$ . Здесь удобно использовать формулы сложения точек, т.к. мы прибавляем  $(x^7, y^7)$  к предыдущей точке  $P$ . Получаем

$$k = \frac{3 \cdot 10000000^2 + 2}{2 \cdot 1064524266y} = \frac{434232361462}{2051341455y},$$

$$x_P = \frac{434232361462^2}{2051341455^2 y^2} - 2 \cdot 10000000 = \frac{213203662514}{220445441503}.$$

Проверяем гипотезу  $x_R - x_P = 0$ :

$$h_x = 552631612401 \cdot 220445441503 - 213203662514 \cdot 533030166456 = \\ = 0 \pmod{\psi_5}.$$

Значит,  $x_R = x_P$ . Теперь нужно сравнить  $y_R$  и  $y_P$ . Имеем

$$y_P = \left( 10000000 - \frac{213203662514}{220445441503} \right) \cdot \frac{434232361462}{2051341455y} - 1064524266y = \\ = \frac{510334350655}{221015611231y}.$$

Проверяем гипотезу  $y_R - y_P = 0$ :

$$h_y = 515441613166y \cdot 221015611231y - 510334350655 \cdot 115165441243 = \\ = 0 \pmod{\psi_5}.$$

Значит,  $t_5 = 2$ .

Наконец, определим  $t_2$ . Нам нужно найти наибольший общий делитель для  $x^3 + ax + b$  и  $x^p - x$ . Сделаем это с помощью алгоритма Евклида для полиномов. Обратим внимание на то, что при больших  $p$ , используемых в криптографии, мы не сможем непосредственно записать полином  $x^p - x$ . Однако на первом шаге алгоритма Евклида вычисляется остаток  $(x^p - x) \pmod{(x^3 + ax + b)}$ , поэтому достаточно подать на вход алгоритма не сам полином  $x^p - x$ , а его остаток. Вычисляем  $x^p \pmod{(x^3 + ax + b)}$  с помощью быстрых методов возведения в степень и вычитаем  $x$ . После этого используем алгоритм Евклида. В нашем примере

$$x^7 = 304 \pmod{1026}, \quad x^7 - x = 364, \quad \gcd(1026, 364) = 1.$$

Значит,  $t_2 = 1$ .

Теперь используем китайскую теорему об остатках [13, 68]. Имеем

$$t = 1 \pmod{2}, \quad t = 0 \pmod{3}, \quad t = 2 \pmod{5},$$

$$N = 2 \cdot 3 \cdot 5 = 30.$$

Решение  $t' = t \pmod{N}$  находится по формуле

$$t' = \sum_{i=1}^3 a_i N_i M_i \pmod{N},$$

где

$$\begin{aligned} a_1 &= 1, & N_1 &= 30/2 = 15, & M_1 &= 15^{-1} \bmod 2 = 1, \\ a_2 &= 0, & N_2 &= 30/3 = 10, & M_2 &= 10^{-1} \bmod 3 = 1, \\ a_3 &= 2, & N_3 &= 30/5 = 6, & M_3 &= 6^{-1} \bmod 5 = 1. \end{aligned}$$

Подставляя числа, получаем

$$t' = 1 \cdot 15 \cdot 1 + 0 \cdot 10 \cdot 1 + 2 \cdot 6 \cdot 1 = 27.$$

Чтобы получить решение, удовлетворяющее неравенству (6.24), вычитаем модуль:

$$t = t' - N = -3.$$

По формуле (6.23) находим

$$\#E_7(2, 6) = 7 + 1 - (-3) = 11. \quad \square$$

Как видим, определение числа точек на кривой — не слишком простая задача. Ее решение требует использования мощной вычислительной техники. На практике используются улучшенные варианты алгоритма Схоффа, основанные на тонких конструкциях высшей алгебры, главное достоинство которых состоит в снижении степени полиномов деления с  $O(m^2)$  до  $O(m)$ . В результате трудоемкость снижается до  $O(\log^6 p)$  и может быть снижена до  $O(\log^{4+\epsilon} p)$  за счет использования асимптотически более быстрых методов умножения и деления (что обычно не оправдано при тех длинах чисел, которые актуальны для крипtosистем на эллиптических кривых).

## 6.7. Использование стандартных кривых

Ввиду того что формирование случайных кривых по рекомендациям разд. 6.3, особенно выполнение этапа подсчета количества точек на кривой, может быть слишком трудной задачей, на практике часто бывает достаточным использовать кривые, предлагаемые различными стандартами или другими источниками. Например, в американском стандарте FIPS 186 приводятся параметры эллиптических кривых для различных длин модулей. Вообще говоря, нет никаких ограничений на использование всеми одной хорошо выбранной кривой. Однако такая кривая, при широком ее использовании, становится слишком притягательной для злоумышленников. Не исключено, что

со временем они смогут найти какие-либо эффективные атаки именно на эту кривую, используя ее характерные особенности, не принятые ранее во внимание. Но все это лишь возможность, вероятность осуществления которой считается многими специалистами ничтожно малой.

Приведем пример реальной кривой, рекомендуемой в FIPS 186 (Curve P-256). Обратным слэшем \ обозначим продолжение числа на следующей строке. Предполагается, что кривая задана уравнением  $Y^2 = X^3 + aX + b \bmod p$ , число точек на кривой  $\#E_p(a, b) = n$ , точка  $G = (x_G, y_G)$  является генератором подмножества точек мощности  $q$ , где  $q$  — простое число.

$$\begin{aligned}
 p &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 = \\
 &= 115792\ 089210356\ 248762697\ 446949407\ 573530086\ \backslash \\
 &\quad 143415290\ 314195533\ 631308867\ 097853951 \\
 a &= -3 \\
 b &= 0x\ 5ac635d8\ aa3a93e7\ b3ebbd55\ 769886bc\ 651d06b0\ \backslash \\
 &\quad cc53b0f6\ 3bce3c3e\ 27d2604b \\
 n &= 115792\ 089210356\ 248762697\ 446949407\ 573529996\ \backslash \\
 &\quad 955224135\ 760342422\ 259061068\ 512044369 \\
 &\quad (\text{простое число}) \\
 q &= n \\
 x_G &= 0x\ 6b17d1f2\ e12c4247\ f8bce6e5\ 63a440f2\ 77037d81\ \backslash \\
 &\quad 2deb33a0\ f4a13945\ d898c296 \\
 y_G &= 0x\ 4fe342e2\ fe1a7f9b\ 8ee7eb4a\ 7c0f9e16\ 2bce3357\ \backslash \\
 &\quad 6b315ece\ cbb64068\ 37bf51f5
 \end{aligned}$$

Мы видим, что  $p$  и  $a$  вряд ли можно назвать случайно выбранными числами и вся «случайность» кривой определяется случайным выбором параметра  $b$  (это считается вполне обоснованным).

Первый вопрос, который возникает, когда мы видим кривую, подобную вышеприведенной, — это нет ли ошибки в записи параметров. Три проверки могут быть сделаны для выяснения этого вопроса.

1. Проверяем, является ли число  $p$  простым.
2. Проверяем, удовлетворяет ли точка  $G = (x_G, y_G)$  уравнению кривой.
3. Удостоверяемся в том, что  $n$  — это действительно количество точек на кривой. Заметим, что такую проверку целесообразно

делать и в том случае, когда мы сами вычисляем  $n$ , например, с помощью алгоритма Схоофа. В общем случае  $n = hq$ , где  $h$  — небольшое число, а  $q$  — простое. Прежде всего путем последовательного деления и проверки на простоту следует убедиться, что  $n$  соответствует заявленной форме. Затем выбираем случайно точку на кривой  $P$  (можно взять  $P = [k]G$ , где  $k$  — случайно выбранное число). Число  $n$  гарантированно является числом точек на кривой, если одновременно  $[n]P = \mathcal{O}$  и  $[h]P \neq \mathcal{O}$ . При невыполнении этого условия возможны два варианта. Если  $[n]P \neq \mathcal{O}$ , то  $n$  — это не число точек. Если же  $[h]P = \mathcal{O}$  (вероятность этого крайне мала), то нужно взять другую точку  $P$ .

В дополнение к указанным проверкам можно проверить выполнение всех других требований, изложенных в разд. 6.3.

Второй вопрос, который возникает, когда нам предъявляют «готовую» кривую, это действительно ли она была сгенерирована случайным образом. Этот вопрос вообще актуален для многих задач криптографии. Возможно, предложенная кривая обладает каким-либо редким свойством, позволяющим «взламывать» крипtosистему, и человек, изготавивший эту кривую, может в дальнейшем получить доступ, скажем, к информации, зашифрованной при помощи этой кривой. Задача доказательства отсутствия заранее заданных специальных свойств кривой сводится к задаче подобного же доказательства для случайно выбираемых параметров. Например, для рекомендованной выше кривой мы должны доказать случайность выбора параметра  $b$ . Эту задачу можно решить следующим образом. Пусть  $h(x)$  — криптографически стойкая хеш-функция. Чтобы сгенерировать число  $b$ , вначале выбираем число  $s$ , а затем вычисляем  $b = h(s)$  и предъявляем оба числа  $b$  и  $s$ . Если хеш-функция удовлетворяет всем требованиям стойкости, то  $b$  не может иметь никаких заранее заданных свойств и мы можем спокойно им пользоваться. Число  $s$  является «сертификатом», доказывающим «чистоту» числа  $b$ . Приведенная выше эллиптическая кривая имеет подобный сертификат, основанный на хеш-функции SHA-1 [54], и стандарт оговаривает процедуру использования этого сертификата. Так что мы можем быть уверены в том, что кривая была сформирована действительно случайным образом.

## Задачи и упражнения

- 6.1. Эллиптическая кривая задана параметрами  $p = 11$ ,  $a = 4$ ,  $b = 7$ . Определить, принадлежат ли кривой следующие точки:  $(1,1)$ ,  $(1,2)$ ,  $(2,1)$ ,  $(2,2)$ ,  $(5,8)$ .
- 6.2. Для эллиптической кривой с параметрами  $p = 7$ ,  $a = 2$ ,  $b = 6$  вычислить следующие композиции точек:  $[2](2,2)$ ,  $[2](4,6)$ ,  $(1,3) + (1,4)$ ,  $(2,2) + (3,2)$ ,  $(3,5) + (5,1)$ .

## Темы лабораторных работ

В лабораторных работах рекомендуется использовать эллиптическую кривую со следующими параметрами:

$$p = 31991, \quad a = -3 = 31988, \quad b = 1000.$$

Количество точек на этой кривой

$$n = 32089 \quad (\text{простое число}).$$

В качестве генератора можно взять точку

$$G = (0, 5585).$$

Точку в бесконечности  $\mathcal{O}$  удобно представлять как точку с координатами  $(0,0)$ .

- 6.3. Написать набор подпрограмм для вычисления композиции и  $m$ -кратной композиции точек на эллиптической кривой. Приведем несколько тождеств для тестирования разработанных

подпрограмм:

$$\begin{aligned}
 (51, 7858) + (91, 5500) &= (7252, 18353), \\
 (7777, 10935) + (16000, 20400) &= (12395, 26268), \\
 (12405, 28624) + (2963, 16300) &= (14905, 2313), \\
 (8020, 1740) + (8020, 30251) &= \mathcal{O}, \\
 [2](0, 5585) &= (8, 19435), \\
 [2](23161, 17992) &= (26775, 10831), \\
 [2](110, 13171) &= (26948, 16087), \\
 [10000](31122, 9) &= (31180, 29596), \\
 [12345](13140, 5033) &= (9362, 27046), \\
 [11111](11007, 23704) &= (850, 6718).
 \end{aligned}$$

- 6.4.** Выполнить программную реализацию шифра Эль-Гамаля на эллиптической кривой. При отладке и тестировании программы можно воспользоваться следующим примером построения шифра:

$$\begin{aligned}
 c &= 5103, \quad D = (12507, 2027); \\
 m &= 10000, \quad k = 523; \\
 R &= (9767, 11500), \quad P = (25482, 16638); \\
 e &= 11685.
 \end{aligned}$$

Полученный шифротекст  $((9767, 11500), 11685)$  должен дешифроваться в сообщение 10000 при помощи секретного ключа 5103.

- 6.5.** Выполнить программную реализацию алгоритмов генерации и проверки цифровой подписи по ГОСТ Р 34.10-2012 (как обычно, полагаем  $h(m) = m$ ). В качестве  $q$  берем  $n = 32089$  (число точек на кривой). Подписанное сообщение

$$(1000; 4615, 5944)$$

должно признаваться подлинным для пользователя, имеющего открытый ключ  $Y = (12224, 7207)$ .

# Глава 7. ТЕОРЕТИЧЕСКАЯ СТОЙКОСТЬ КРИПТОСИСТЕМ

## 7.1. Введение

Одна из первых открытых работ по криптографии появилась в 1949 году и принадлежит Шенонну (Claude Shannon [28, 88]). В ней рассматривается классическая схема криптосистемы с секретным ключом, которая была представлена на рис. 1.1 (стр. 6).

В этой схеме имеется защищенный канал, предназначенный для передачи секретных ключей. Однако отметим, что в настоящее время можно рассмотреть в качестве защищенного канала схему вычисления секретного ключа на основе методов криптографии с открытым ключом, например, схему Диффи–Хеллмана или протокол Нидхама–Шредера. В дальнейшем мы будем рассматривать только классическую схему с секретным ключом, но многие результаты переносятся на случай создания секретного канала средствами криптографии с открытым ключом.

Все методы шифрования можно грубо разделить на два больших класса:

- 1) схемы принципиально не вскрываемые, что доказывается математически строго;
- 2) схемы, стойкость которых основана на том, что для дешифрования без ключа, которое, вообще говоря, возможно, требуется перебор очень большого количества вариантов.

В этой главе мы будем заниматься системами первого типа, стойкость которых теоретически доказана. Системы второго типа будут рассмотрены в следующей главе.

## 7.2. Теория систем с совершенной секретностью

Пусть  $\mathcal{M} = \{M_1, M_2, M_3, \dots, M_m\}$  — множество всех возможных сообщений (например, множество всех текстов длины не более чем 1000 букв),  $\mathcal{K} = \{K_1, K_2, K_3, \dots, K_n\}$  — множество всех возможных ключей,  $\mathcal{E} = \{E_1, E_2, \dots, E_k\}$  — множество всех криптограмм (т.е. зашифрованных сообщений). Зашифрованные сообщения зависят от исходного сообщения и ключа, т.е.  $E_j = f(M_i, K_l)$ .

Мы будем считать, что на всем множестве сообщений  $\mathcal{M}$  задано распределение вероятностей  $P$ , т.е. определены вероятности  $P(M_i)$ ,  $i = 1, 2, \dots, m$ . Это априорное распределение вероятностей, которое известно и противнику. Запись вида  $P(A|B)$  будет, как обычно, обозначать условную вероятность события  $A$  при условии наступления события  $B$ .

**Определение 7.1.** Криптосистема называется *совершенно секретной*, если выполняется равенство

$$P(M_i|E_j) = P(M_i) \quad (7.1)$$

при всех  $M_i$ ,  $K_l$  и  $E_j = f(M_i, K_l)$ .

Поясним это определение. Пусть Ева перехватила криптограмму  $E_j$ . Если (7.1) выполняется для всех возможных сообщений, то это означает, что она не получила никакой информации о переданном сообщении, т.е. знание  $E_j$  совершенно бесполезно для нее. Рассмотрим схематичный пример. Пусть  $\mathcal{M}$  — множество всех сообщений из шести букв на русском языке. Пусть априори известно, что для некоторой системы

$$P(\text{сообщение} = \text{«доллар»}) = 0,00015,$$

$$P(\text{сообщение} = \text{«бутыль»}) = 0,0000012 \text{ и т.д.}$$

Допустим, мы имеем несовершенную систему, и Ева после перехвата и вычислений получила следующие данные:

$$P(\text{сообщение} = \text{«доллар»}) = 10^{-20},$$

$$P(\text{сообщение} = \text{«бутыль»}) = 0,9999.$$

Это означает, что Ева практически расшифровала сообщение: она практически уверена, что передано слово «бутыль», т.к. вероятность, что передано другое сообщение, меньше 0,0001.

Если же для рассмотренной системы при любой перехваченной криптограмме  $E_j$  мы получаем

$$\begin{aligned} P(\text{сообщение} = \text{«доллар»} | E_j) &= 0,00015, \\ P(\text{сообщение} = \text{«бутыль»} | E_j) &= 0,0000012 \end{aligned}$$

и такие же равенства выполняются для всех остальных сообщений, то Ева вообще может не обращать внимание на перехваченный шифротекст  $E_j$ , а, например, отгадывать сообщение на основе исходных вероятностей. Другими словами, (7.1) — действительно разумное определение совершенно секретной системы.

Исследуем свойства совершенно секретной системы.

**Теорема 7.1.** *Если система является совершенно секретной (выполняется (7.1)), то справедливо равенство*

$$P(E_j | M_i) = P(E_j) \quad (7.2)$$

при всех  $i$  и  $j$ . Верно и обратное утверждение: если (7.2) выполняется, то система совершенно секретна.

**Доказательство.** По определению условной вероятности

$$P(A|B) = \frac{P(AB)}{P(B)},$$

при  $P(B) \neq 0$  (см., например, [26]). Поэтому при  $P(E_j) \neq 0$  можно записать

$$P(M_i | E_j) = \frac{P(M_i E_j)}{P(E_j)} = \frac{P(M_i) P(E_j | M_i)}{P(E_j)}.$$

Принимая во внимание (7.1), получаем

$$P(M_i | E_j) = \frac{P(M_i | E_j) P(E_j | M_i)}{P(E_j)},$$

т.е.

$$\frac{P(E_j | M_i)}{P(E_j)} = 1.$$

Таким образом, (7.2) доказано. Обратное утверждение теоремы доказывается «обратным ходом» приведенных равенств, см. [28].  $\square$

### 7.3. Шифр Вернама

Этот шифр был предложен в 1926 году американским инженером Вернамом (Gilbert Vernam) и использовался на практике, но доказательство его невскрываемости было получено значительно позже Шенном [28, 88]. Для шифра Вернама часто используется название «одноразовый блокнот» (one-time pad). Мы опишем этот шифр для случая двоичного алфавита, чтобы упростить обозначения.

Пусть множество сообщений  $\mathcal{M}$  состоит из слов двоичного алфавита длины  $n$ , т.е. всего сообщений не более чем  $2^n$ . В шифре Вернама множество ключей также состоит из слов той же длины  $n$  и каждый ключ используется с вероятностью  $1/2^n$ . Другими словами, все ключи используются с одинаковой вероятностью.

Пусть необходимо зашифровать сообщение  $M = m_1 m_2 \dots m_n$  и пусть выбран ключ  $K = k_1 k_2 \dots k_n$ . Тогда зашифрованное сообщение  $E = e_1 e_2 \dots e_n$  получается по формуле:

$$e_i = m_i \oplus k_i, \quad (7.3)$$

где  $i = 1, 2, \dots, n$  и  $\oplus$  обозначает сложение по модулю 2. Другими словами, сообщение шифруется по схеме

$$\begin{array}{r} \oplus \quad m_1 \quad m_2 \quad \dots \quad m_n \\ \hline k_1 \quad k_2 \quad \dots \quad k_n \\ \hline e_1 \quad e_2 \quad \dots \quad e_n \end{array}.$$

Так как сложение и вычитание по модулю 2 совпадают, то легко видеть, что дешифрование осуществляется по формуле

$$m_i = e_i \oplus k_i. \quad (7.4)$$

**Пример 7.1.** Пусть  $M = 01001$ ,  $K = 11010$ . Тогда получаем  $E = 10011$ . Сложив  $E$  с  $K$ , восстанавливаем  $M$ .  $\square$

**Теорема 7.2.** Шифр Вернама является совершенно секретной крипtosистемой.

**Доказательство.** Согласно теореме 7.1 достаточно доказать справедливость (7.2). Имеем

$$\begin{aligned} P(E_j | M_i) &= P(e^n | m^n) = \\ &= P(k_1 = e_1 \oplus m_1; k_2 = e_2 \oplus m_2; \dots; k_n = e_n \oplus m_n) = \\ &= P(\text{ключ} = k_1 \dots k_n) = 2^{-n} \end{aligned}$$

(в последнем равенстве мы использовали то, что, по предположению, все ключи равновероятны).

Найдем  $P(E_j)$ . По формуле полной вероятности

$$P(E_j) = \sum_{i=1}^{2^n} P(M_i)P(E_j|M_i).$$

Учитывая, что  $P(E_j|M_i) = 2^{-n}$ , получаем

$$P(E_j) = 2^{-n} \sum_{i=1}^{2^n} P(M_i).$$

Так как сумма вероятностей всех возможных сообщений равна 1, получаем

$$P(E_j) = 2^{-n}.$$

Таким образом, справедливо (7.2). Теорема доказана.  $\square$

Известно, что шифр Вернама использовался при защите правительственной связи: например, на так называемой горячей линии «Москва – Вашингтон» [68], а также в других системах, где можно позволить дорогой способ доставки секретного ключа. Однако шифр Вернама можно использовать во многих других практически важных ситуациях. Например, на основе этого шифра легко организовать связь между банком и его филиалами (или связи между банком и клиентами), когда они находятся в одном городе, или защитить электронные письма, скажем, для студентов Алисы и Боба, расстающихся на каникулы.

Другой практически важный вопрос, возникающий при использовании шифра Вернама, связан с требованием равновероятности всех ключей. В случае двоичного алфавита это требование эквивалентно тому, что символы ключевой последовательности независимы и вероятности 0 и 1 равны. Для выполнения этого условия надо использовать генераторы двоичных символов, гарантирующие выполнение этих свойств. Но реальные генераторы могут, вообще говоря, немного отличаться от «идеального» (генерирующего равновероятные независимые символы) и возникает вопрос о возможности использования таких генераторов в шифре Вернама. Например, если генератор порождает независимые символы с вероятностями  $P(0) = 0.501$ ,  $P(1) = 0.499$ , то сохранится ли основное свойство

шифра Вернама? Ответ на этот вопрос будет дан в разделе, следующим за описанием основных понятий теории информации, а пока только отметим, что шифр Вернама устойчив к небольшим отклонениям ключа (или генератора, его порождающего) от равновероятности и независимости символов.

## 7.4. Элементы теории информации

Мы доказали, что шифр Вернама совершенен, однако при его использовании длина ключа равна длине сообщения. К. Шеннон [28] показал, что у любой совершенной системы длина ключа должна быть больше энтропии сообщения (с которой мы кратко познакомимся ниже), т.е. пропорциональна его длине. Во многих практически важных ситуациях приходится использовать короткие ключи (скажем, сотни или тысячи бит) для шифрования длинных сообщений (сотни килобайт и более). В этом случае можно построить только так называемые идеальные системы, впервые описанные Шенноном. Для построения идеальных систем и исследования их свойств Шеннон предложил использовать понятия теории информации. В этом разделе мы определим и кратко проиллюстрируем эти понятия. Достаточно полное и строгое их изложение может быть найдено, например, в [4].

Начнем с определения основного понятия — энтропии Шеннона. Пусть дана дискретная случайная величина  $\xi$ , принимающая значения  $a_1, a_2, \dots, a_r$  с вероятностями  $P_1, P_2, \dots, P_r$ .

**Определение 7.2.** Энтропия случайной величины  $\xi$  определяется равенством

$$H(\xi) = - \sum_{i=1}^r P_i \log P_i, \quad (7.5)$$

где  $0 \log 0 = 0$ .

Если используется двоичный логарифм, то энтропия измеряется в битах, что общепринято в криптографии, теории информации и в компьютерных науках. В случае натуральных логарифмов единица измерения — нат, в случае десятичных — дит.

При  $r = 2$  можно (7.5) записать иначе, введя следующие обозначения:  $P_1 = p$ ,  $P_2 = 1 - p$ . Тогда

$$H = -(p \log p + (1 - p) \log (1 - p)). \quad (7.6)$$

График энтропии для этого случая приведен на рис. 7.1.

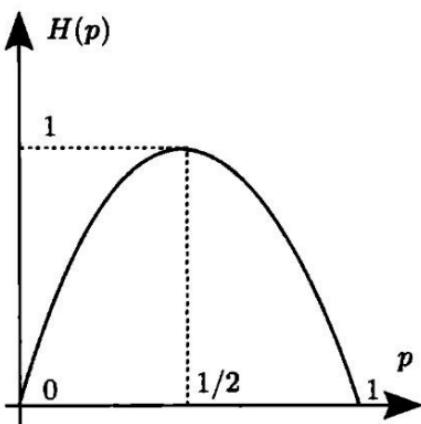


Рис. 7.1. График двоичной энтропии

Рассмотрим простейшие свойства энтропии.

**Утверждение 7.3.**

- 1)  $H(\xi) \geq 0$ ;
- 2)  $H(\xi) \leq \log r$ ;
- 3)  $H(\xi) = \log r$  при  $P_i = 1/r$ ,  $i = 1, 2, \dots, r$ .

**Доказательство.** Первое свойство довольно очевидно (см. (7.5)). Второе свойство докажем только для случая  $r = 2$ , т.к. общий случай аналогичен. Исследуем график энтропии. Нужно найти максимум функции (7.6). Для этого мы найдем первую и вторую производные  $H(p)$ , считая, что логарифм натуральный.

$$H'(p) = - \left( \ln p + p \cdot \frac{1}{p} - \ln(1-p) - \frac{1-p}{1-p} \right) = -\ln p + \ln(1-p).$$

Отсюда  $H'(p) = 0$  при  $p = 1/2$ . Найдем вторую производную

$$H''(p) = -\frac{1}{p} - \frac{1}{1-p}.$$

Мы видим, что  $H''(p) < 0$  при  $p \in (0; 1)$ . Это означает, что функция  $H(p)$  достигает максимума в точке  $1/2$  и выпукла на отрезке  $(0; 1)$ . Таким образом, график, изображенный на рис. 7.1, обоснован. Очевидно, при любом основании логарифма график будет аналогичный.

Для доказательства третьего свойства заметим, что наибольшее значение энтропии для  $r = 2$  равно

$$\max H(p) = H(1/2) = -((1/2) \log(1/2) + (1/2) \log(1/2)) = 1,$$

т.е. составляет один бит в случае двоичного логарифма.  $\square$

«Физический» смысл энтропии состоит в том, что энтропия — это количественная мера неопределенности. В качестве примера рассмотрим три случайные величины для  $r = 2$ . Иными словами, будем считать, что у нас есть три источника сообщений, которые порождают буквы  $a_1, a_2$ , т.е. имеются три случайные величины  $\xi_i, i = 1, 2, 3$ , принимающие значения  $a_1$  или  $a_2$ :

$$\begin{aligned}\xi_1 : \quad P(a_1) &= 1, & P(a_2) &= 0; \\ \xi_2 : \quad P(a_1) &= 0,5, & P(a_2) &= 0,5; \\ \xi_3 : \quad P(a_1) &= 0,01, & P(a_2) &= 0,99.\end{aligned}$$

Интуитивно ясно, что неопределенность случайной величины  $\xi_1$  равна нулю. И действительно,

$$H(\xi_1) = -(1 \cdot \log 1 + 0 \cdot \log 0) = 0.$$

Посмотрим на  $\xi_2$  и  $\xi_3$ . Интуитивно кажется, что неопределенность у  $\xi_2$  выше неопределенности у  $\xi_3$ . Вычислим энтропии:

$$H(\xi_2) = 1 \text{ бит}$$

(уже считали выше),

$$H(\xi_3) = -(0,01 \cdot \log 0,01 + 0,99 \cdot \log 0,99) \approx 0,08 \text{ бит.}$$

Мы видим, что энтропия действительно является разумной мерой неопределенности. Но главное, конечно, не примеры такого типа, а то, что эта величина играет ключевую роль во многих задачах теории передачи и хранения информации. В частности, энтропия характеризует максимальную степень сжатия данных. Точнее, если источник сообщений порождает текст достаточно большой длины  $n$

с определенной ниже предельной энтропией  $h$  на бит сообщения, то этот текст может быть «сжат» в среднем до величины сколь угодно близкой к  $nh$ . Например, если  $h = 1/2$ , то текст сжимается вдвое и т.п. Подчеркнем, что речь идет о так называемом неискажающем сжатии, когда по «сжатому» сообщению можно восстановить исходное.

Рассмотрим теперь двумерную случайную величину, заданную рядом распределения

$$P_{ij} = P(\xi_1 = a_i, \xi_2 = b_j), \quad 1 \leq i \leq r, \quad 1 \leq j \leq s. \quad (7.7)$$

**Пример 7.2.** Из многолетнего опыта преподавания в некотором вузе известно, что оценки за первый и второй контрольный срок по математике (соответственно  $\xi_1$  и  $\xi_2$ ) подчиняются закону распределения, задаваемому табл. 7.1.  $\square$

Т а б л и ц а 7.1. Распределение оценок за контрольный срок

$\xi_1 \downarrow \xi_2 \rightarrow$	0	1	2
0	0,20	0,05	0
1	0,05	0,30	0,05
2	0	0,05	0,30

Введем следующие обозначения:

$$P_{i \cdot} = P(\xi_1 = a_i) = \sum_{j=1}^s P_{ij},$$

$$P_{\cdot j} = P(\xi_2 = b_j) = \sum_{i=1}^r P_{ij}.$$

Напомним некоторые элементарные соотношения, известные из теории вероятностей (см. [26]):

$$P(A|B) = \frac{P(AB)}{P(B)}, \quad (7.8)$$

$$P(AB) = P(A)P(B|A) = P(B)P(A|B), \quad (7.9)$$

$$P(A) = \sum_i P(H_i)P(A|H_i) \quad (7.10)$$

((7.10) — уже встречавшаяся формула полной вероятности, в которой  $H_i$  — попарно несовместные события, сумма которых содержит событие  $A$ ).

В соответствии с (7.5) определим энтропию двумерной случайной величины

$$H(\xi_1, \xi_2) = - \sum_{i=1}^r \sum_{j=1}^s P_{ij} \log P_{ij}. \quad (7.11)$$

Аналогично для трехмерной случайной величины  $(\xi_1, \xi_2, \xi_3)$  и распределения вероятностей  $P_{ijk}$  определим

$$H(\xi_1, \xi_2, \xi_3) = - \sum_i \sum_j \sum_k P_{ijk} \log P_{ijk}. \quad (7.12)$$

Подобным же образом определяется энтропия для  $n$ -мерной случайной величины.

Представим теперь, что значение случайной величины  $\xi_1$  известно, а  $\xi_2$  — неизвестно. Тогда естественно определить условную энтропию

$$H(\xi_2|\xi_1) = - \sum_{i=1}^r P_i \sum_{j=1}^s \frac{P_{ij}}{P_i} \log \frac{P_{ij}}{P_i}. \quad (7.13)$$

Это — средняя условная энтропия случайной величины  $\xi_2$  при условии, что значение  $\xi_1$  известно.

**Утверждение 7.4** (свойство двумерной энтропии).

$$H(\xi_1, \xi_2) = H(\xi_1) + H(\xi_2|\xi_1), \quad (7.14)$$

в частности, для независимых случайных величин  $\xi_1$  и  $\xi_2$

$$\begin{aligned} H(\xi_2|\xi_1) &= H(\xi_2), \\ H(\xi_1, \xi_2) &= H(\xi_1) + H(\xi_2). \end{aligned} \quad (7.15)$$

Напомним, что  $\xi_1$ ,  $\xi_2$  независимы, если  $P_{ij} = P_i P_j$  для всех  $i$  и  $j$ . Доказательство утверждения достаточно просто и может быть найдено в [4]. Мы ограничимся только его интерпретацией. Пусть в первом опыте порождается  $\xi_2$ , во втором —  $\xi_1$ . Тогда общая неопределенность эксперимента должна быть равна неопределенности первого опыта, сложенной с условной неопределенностью второго опыта. В случае независимых  $\xi_1$  и  $\xi_2$  знание одной величины не несет никакой информации о другой, что соответствует (7.15).

Пусть дана  $n$ -мерная случайная величина  $(\xi_1, \xi_2, \dots, \xi_n)$ . Справедливо следующее соотношение [4]:

$$H(\xi_1, \dots, \xi_n) = H(\xi_1) + H(\xi_2 | \xi_1) + H(\xi_3 | \xi_1, \xi_2) + \dots + H(\xi_n | \xi_1, \dots, \xi_{n-1}). \quad (7.16)$$

Для независимых случайных величин

$$H(\xi_1, \dots, \xi_n) = \sum_{i=1}^n H(\xi_i) \quad (7.17)$$

(заметим, что (7.14), (7.15) — частный случай (7.16), (7.17)).

В общем случае

$$H(\xi_k | \xi_1, \dots, \xi_{k-1}) \leq H(\xi_k). \quad (7.18)$$

Рассмотрим последовательность случайных величин  $\xi_1, \xi_2, \xi_3, \dots$  ( $\xi_i$  принимают значения в  $A$ ), которую можно рассматривать как случайный процесс с дискретным временем. Мы будем считать, что этот процесс стационарный, т.е., неформально, вероятностные характеристики для  $(\xi_1 \dots \xi_n)$  те же, что и для  $(\xi_{\Delta+1} \dots \xi_{\Delta+n})$  при всех положительных  $n$  и  $\Delta$  (точное определение дано в [4]).

Пусть  $H(\xi_1, \dots, \xi_n)$  — энтропия  $n$ -мерной случайной величины  $(\xi_1 \dots \xi_n)$ . Обозначим через

$$h_n^+ = \frac{1}{n} H(\xi_1, \dots, \xi_n)$$

удельную энтропию  $n$ -го порядка и определим

$$h_n^- = H(\xi_n | \xi_1, \dots, \xi_{n-1}).$$

Отметим следующие свойства:

$$h_n^+ \leq h_{n-1}^+, \quad n > 1, \quad (7.19)$$

$$h_n^- \leq h_n^+, \quad (7.20)$$

$$h_n^- \leq h_{n-1}^-, \quad n > 1. \quad (7.21)$$

Их доказательство может быть найдено в [4].

Для независимых  $\xi_1, \xi_2, \dots, \xi_n$  справедливы равенства

$$h_n^+ = h_n^- = h.$$

(Процесс, порождающий независимые случайные величины, называется процессом без памяти.)

**Теорема 7.5.** Для стационарного процесса существуют пределы  $\lim_{n \rightarrow \infty} h_n^+$  и  $\lim_{n \rightarrow \infty} h_n^-$ , причем эти пределы равны.

Доказательство теоремы см. в [4].

Обозначим общее значение этих пределов через  $h_\infty$ ,

$$h_\infty = \lim_{n \rightarrow \infty} h_n^+ = \lim_{n \rightarrow \infty} h_n^-. \quad (7.22)$$

Пусть дан алфавит  $A = (a_1, a_2, \dots, a_r)$ . Мы знаем, что

$$\max H(\xi_1) = \log r$$

для процесса без памяти, поэтому, принимая во внимание (7.21) и (7.22), получаем  $\max h_\infty = \log r$ , причем максимум достигается для процессов без памяти, у которых все буквы порождаются с равными вероятностями  $1/r$ . Естественно ввести величину

$$R = \log r - h_\infty, \quad (7.23)$$

называемую избыточностью (на букву сообщения). Неформально, это как бы неиспользованная часть алфавита. Избыточность — количественная мера взаимной зависимости символов и их «неравновероятности». Отметим, что в примере из первой главы во втором случае, когда даже простой шифр Цезаря не вскрываем, избыточность шифруемого сообщения равна нулю, т.к. все десять символов независимы и равновероятны, т.е.  $h_\infty = \log 10$  и  $R = 0$ .

## 7.5. Устойчивость шифра Вернама к небольшим отклонениям ключа от случайности

Рассмотрим обобщение шифра Вернама на случай, когда исходный текст  $m_1 \dots m_n$ , ключевая последовательность  $k_1 \dots k_n$  и зашифрованная последовательность  $e_1 \dots e_n$  принадлежат одному алфавиту  $A = \{0, 1, \dots, r-1\}$ , где  $r \geq 2$ . Шифрование и расшифрование определяются равенствами

$$e_i = (m_i + k_i) \bmod r, \quad m_i = (e_i - k_i) \bmod r. \quad (7.24)$$

В случае  $r = 2$  соотношения (7.24) можно представить в уже знакомом для нас виде

$$e_i = (m_i \oplus k_i), \quad m_i = (e_i \oplus k_i), \quad (7.25)$$

где  $a \oplus b = (a + b) \bmod 2$ .

Мы знаем, что этот шифр совершенно секретен, если символы ключевой последовательности равновероятны и независимы, т.е. для каждого слова  $k_1 \dots k_n$ ,  $k_i \in A$ ,  $P(\text{ключ} = k_1 \dots k_n) = r^{-n}$ . Используя введенное понятие энтропии Шеннона, это свойство можно представить в виде  $H(M|E) = H(M)$ , где, как и ранее,  $M$  — множество всех сообщений,  $E$  — множество перехваченных криптограмм.

Если текст порождается стационарным эргодическим источником, то совершенная секретность шифра Вернама имеет простую интерпретацию, поясняющую смысл этого понятия. Как следует из известной в теории информации теоремы Шеннона–Макмиллана–Бреймана [43, 60], все множество сообщений  $m_1 \dots m_n$  при больших  $n$  можно разбить на две части:  $2^{H(m_1 \dots m_n)}$  сообщений, вероятности которых близки по величине и в сумме дают почти единицу, и множество остальных сообщений, суммарная вероятность которых близка к нулю. Ева знает, что, почти наверное, зашифрованное сообщение принадлежит первому подмножеству, но все сообщения в этом подмножестве имеют близкие вероятности, причем количество таких сообщений растет экспоненциально как  $2^{hn}$ , где  $h = h_\infty$  — предельная энтропия источника сообщений (в дальнейшем, для краткости, мы будем называть ее просто энтропией источника). Именно по этой причине Ева не может определить исходное сообщение.

Но что будет происходить с шифром (7.24), когда ключевая последовательность порождается стационарным эргодическим источником и немного отличается от последовательности равновероятных и независимых символов? Ответ на этот вопрос был дан в [19]. В этой работе было показано, что свойства такого шифра остаются в определенном смысле близкими к шифру Вернама. Точнее, в этом случае множество близких по вероятности сообщений, суммарная вероятность которых близка к 1, растет как  $2^{n(h-R_K)}$ , где, по-прежнему,  $h$  — энтропия источника, а  $R_K$  — избыточность источника ключа,  $R_K = \log r - h(K)$ , где  $r$  — число букв алфавита,  $h(K)$  — энтропия источника ключа. Если избыточность  $R_K$  приближается к нулю, то число элементов в высоковероятном множестве приближается к  $2^{hn}$ , т.е. к шифру Вернама. Это и позволяет утверждать, что шифр Вернама устойчив к малым отклонениям.

Интересно, что К. Шеннон в своей основополагающей статье [88] отметил, что “с криптографической точки зрения секретная система почти тождественна системе связи при наличии шума”. Поэтому

математически задачи дешифрования текстов и фильтрации случайных процессов очень близки. Развиваемый здесь подход близок к методам из работы [84], в которой рассматривается задача фильтрации стационарных процессов.

Перейдем к более строгому обоснованию высказанных положений. Вначале сформулируем упомянутую выше теорему Шеннона–Макмиллана–Бреймана:

**Теорема 7.6.** *Пусть  $U = U_1 U_2 U_3 \dots$  – стационарный эргодический процесс. Тогда для любых положительных  $\epsilon, \delta$ , для почти всех  $U_1 U_2 U_3 \dots$  существует  $n'$  такое, что при  $n > n'$*

$$P \left\{ \left| -\frac{1}{n} \log P(U_1 \dots U_n) - h(U) \right| < \epsilon \right\} \geq 1 - \delta. \quad (7.26)$$

Напомним, что мы рассматриваем случай, когда шифруемый текст  $m_1, m_2, \dots$  и последовательность ключа  $k_1, k_2, \dots$  независимо генерируются стационарными эргодическими процессами с одинаковым конечным алфавитом  $A = \{0, 1, \dots, r - 1\}$ ,  $r \geq 2$ , а зашифрованная последовательность  $e_1, e_2, \dots$  определяется соотношением (7.24). Энтропия Шеннона  $n$ -ого порядка и предельная энтропия задаются равенствами

$$h_n(M) = -\frac{1}{n} \sum_{u \in A^n} P_M(u) \log P_M(u), \quad h(M) = \lim_{n \rightarrow \infty} h_n(M), \quad (7.27)$$

где  $P_M(u)$  – вероятность того, что  $m_1 m_2 \dots m_n = u$ . Определим также условную энтропию

$$h_n(M|E) = h_n(M, E) - h_n(E), \quad h(M|E) = \lim_{n \rightarrow \infty} h_n(M|E). \quad (7.28)$$

Основной результат этого раздела следующий:

**Теорема 7.7.** *Пусть текст  $M = m_1 m_2 \dots$  и секретный ключ  $K = k_1 k_2 \dots$  – независимые случайные процессы с алфавитом  $A = \{0, 1, \dots, r - 1\}$ , такие что  $(M, K) = (m_1, k_1), (m_2, k_2), \dots$  – стационарный и эргодический процесс, а  $E = e_1 e_2 \dots$  определяется соотношением (7.24). Тогда, с вероятностью 1, для каждого  $\epsilon > 0$  и  $\delta > 0$  существует такое целое  $n'$ , что для каждого  $n > n'$  и  $E_1^n = e_1 e_2 \dots e_n$  существует множество расшифровок  $\Psi(E_1^n)$ , для которого:*

- 1)  $P(\Psi(E_1^n)) > 1 - \delta$ ;
- 2) для каждого  $M^1 = m_1^1 \dots m_n^1$ ,  $M^2 = m_1^2 \dots m_n^2$  из  $\Psi(E_1^n)$ 

$$\frac{1}{n} |\log P(M^1|E_1^n) - \log P(M^2|E_1^n)| < \varepsilon;$$
- 3)  $\liminf_{n \rightarrow \infty} \frac{1}{n} \log |\Psi(E_1^n)| \geq h(M|E)$ .

**Доказательство.** Так как процесс  $(M, E)$  является детерминированной функцией процесса  $(M, K)$ , то из эргодичности и стационарности процесса  $(M, K)$  следует эргодичность и стационарность процесса  $(M, E)$ .

Перейдем к построению множества  $\Psi(E_1^n)$ . Для этого применим теорему 7.6, но сначала заметим, что  $h(M|E) = h(M, E) - h(E)$  и  $\log P(m_1 \dots m_n | e_1 \dots e_n) = \log P(m_1 \dots m_n; e_1 \dots e_n) - \log P(e_1 \dots e_n)$ . Теорему 7.6 можно записать следующим образом: для любых положительных  $\varepsilon, \delta$  и для почти всех  $(M_1, E_1), (M_2, E_2), \dots$  существует  $n'$  такое, что при  $n > n'$

$$P \left\{ \left| -\frac{1}{n} \log P(m_1 \dots m_n | e_1 \dots e_n) - h(M|E) \right| < \varepsilon/2 \right\} \geq 1 - \delta. \quad (7.29)$$

Определим  $\Psi(E_1^n)$  как множество всех сообщений, удовлетворяющих (7.29), и получим первое утверждение теоремы.

Для доказательства второго утверждения теоремы заметим, что для  $M^1 = m_1^1 \dots m_n^1$ ,  $M^2 = m_1^2 \dots m_n^2$  из  $\Psi(E_1^n)$  из (7.29) получаем

$$\begin{aligned} \frac{1}{n} |\log P(M^1|E_1^n) - \log P(M^2|E_1^n)| &\leq \frac{1}{n} |\log P(M^1|E_1^n) - h(M|E)| \\ &+ \frac{1}{n} |\log P(M^2|E_1^n) - h(M|E)| < \varepsilon/2 + \varepsilon/2 = \varepsilon. \end{aligned}$$

Из определения множества  $\Psi(E_1^n)$  и утверждения 1) получаем:  $|\Psi(E_1^n)| > (1 - \delta)2^{n(h(M|E) - \varepsilon)}$ . Учитывая, что это выполняется для любых  $\varepsilon > 0$ ,  $\delta > 0$  и  $n > n'$ , мы получаем третье утверждение теоремы.  $\square$

Таким образом, множество возможных расшифровок  $\Psi(E_1^n)$  растет экспоненциально, его суммарная вероятность близка к 1, и вероятности слов внутри этого множества близки по величине.

Приведенная теорема дает возможность оценить характеристики шифра при помощи условной энтропии  $h(M|E)$ . Следующие оценки не требуют вычисления условной энтропии, а базируются на проще вычисляемых величинах.

**Следствие 7.8.** Для почти всех  $e_1 e_2 \dots$

$$\liminf_{n \rightarrow \infty} \frac{1}{n} \log |\Psi(E_1^n)| \geq h(M) + h(K) - \log r, \quad (7.30)$$

$$h(M|E) \geq h(M) + h(K) - \log r. \quad (7.31)$$

**Доказательство.** Из рассмотренного в предыдущем разделе представления  $h(M, E) = h(M) + h(E|M)$  получаем:

$$h(M|E) = h(M, E) - h(E) = h(E|M) + h(M) - h(E).$$

Учитывая, что  $\max h(E) = \log r$ , где  $r$  — число букв алфавита, мы получаем из последнего равенства:  $h(M|E) \geq h(E|M) + h(M) - \log r$ . Из независимости  $M$  и  $K$  и определения шифра (7.24) видно, что  $h(E|M) = h(K)$ . Отсюда и предыдущего равенства получаем (7.31). Учитывая два последних неравенства и третье утверждение теоремы 7.7, получаем доказательство (7.30).  $\square$

Используя ранее введенное понятие избыточности, мы можем выразить скорость роста множества  $\Psi(E_1^n)$  следующим образом:

$$\liminf_{n \rightarrow \infty} \frac{1}{n} \log |\Psi(E_1^n)| \geq h(M) - R_K, \quad (7.32)$$

$$\liminf_{n \rightarrow \infty} \frac{1}{n} \log |\Psi(E_1^n)| \geq h(K) - R_M,$$

$$\liminf_{n \rightarrow \infty} \frac{1}{n} \log |\Psi(E_1^n)| \geq \log r - (R_M + R_K),$$

где  $R_K = \log r - h(K)$  и  $R_M = \log r - h(M)$  — избыточности. Кроме того, из определения избыточности и (7.31) получаем неравенство

$$h(M|E) \geq h(K) - R_M.$$

Эти неравенства позволяют количественно оценить влияние избыточности на стойкость шифра и подтверждают количественно известный в криптографии и теории вероятности факт: уменьшение избыточности повышает надежность шифра.

Вернемся теперь к вопросу о влиянии отклонений от случайности ключевой последовательности шифра Вернама. Пусть дан шифруемый текст  $m_1 m_2 \dots, m_i \in \{0, 1\}$ , и пусть последовательность символов ключа  $k_1 k_2 \dots, k_i \in \{0, 1\}$ , генерируется источником, отличным от источника Бернулли с  $P(0) = P(1) = 0.5$  (например, ключ  $k_1 k_2 \dots$  генерируется бернуллиевским источником с вероятностями символов  $P(0) = 0.5 - \tau$ ,  $P(1) = 0.5 + \tau$ , где  $\tau$  — небольшое число). Из (7.32) мы можем видеть, что величина множества высоковероятных расшифровок  $\Psi(E_1^n)$  растет экспоненциально с показателем, не меньшим  $h(M) - R_K$ , где  $R_K = 1 - h(K)$  — избыточность ключа. Видно, что если  $R_K$  стремится к нулю, то размер множества высоковероятных расшифровок приближается к величине соответствующего множества в шифре Вернама. Действительно, в этом случае  $h(K) = 1$  и, следовательно, избыточность  $R_K = 0$ , как в шифре Вернама. Неформально, можно сказать, что шифр Вернама устойчив к небольшим отклонениям от случайности ключа.

## 7.6. Шифры с бегущим ключом

В предыдущем разделе мы рассмотрели шифр, определяемый соотношениями (7.24). Мы показали, что этот шифр остается невскрываемым даже в случае, когда последовательность символов ключа генерируется с небольшими отклонениями от случайности. Однако в эпоху, когда не было компьютеров и удобных в использовании генераторов случайных символов, шифр (7.24) применялся таким способом, когда в качестве ключевой последовательности брался некоторый текст, как правило, совпадающий по природе с текстом, который требовалось зашифровать. Например, текст на английском языке шифровался с помощью другого текста на английском языке, а секретным ключом фактически была информация о том, откуда взять этот второй текст (это могла быть конкретная книга, имеющаяся в библиотеке как отправителя, так и получателя, с указанием конкретной страницы, номера строки и т.п.). При таком способе использования шифр (7.24) называется «шифром с бегущим ключом» (running-key cipher).

Результаты, представленные в предыдущей главе, позволяют нам сделать заключение о стойкости шифра с бегущим ключом. Покажем это на примере текстов на английском языке.

Пример 7.3. Согласно исследованиям, представленным в таких работах, как [89, 92], энтропия английского языка  $h(M) \approx 1$  бит на символ при мощности алфавита  $r = 26$  букв (эта оценка получена с учетом сильной зависимости между символами в тексте, благодаря чему текст воспринимается не как некоторый случайный процесс, а как нечто, имеющее смысл). Если в качестве ключа использовать так же текст на английском языке, то и  $h(K) \approx 1$ . На основании (7.31) имеем

$$h(M|E) \geq 1 + 1 - \log 26 = 2 - 4,7 = -2,7.$$

□

Такая оценка энтропии не гарантирует стойкости шифра. И действительно, известны методы (см. [68]), которые позволяют восстановить из зашифрованного текста одновременно сообщение и ключ (это, в частности, означает, что  $h(M|E) = 0$ , что не противоречит полученной оценке). Таким образом, «классический» шифр с бегущим ключом совершенно не стоек.

Однако еще в 1949 году Шеннон предложил два способа улучшения шифра с бегущим ключом [28, 88], которые мы рассмотрим более подробно. Строгий математический анализ этих способов был проведен в [77].

Один способ состоит в том, чтобы повысить энтропию текста, используемого в качестве ключа, путем исключения зависимости между символами. Это можно сделать, если брать, скажем, каждую десятую букву текста. На расстоянии в 10 букв зависимость между символами практически не просматривается.

Пример 7.4. Следующая последовательность букв была получена из начального параграфа одного из важнейших исторических документов США:

WCUSSYOSOAHCIRUHFTTLOLUAEERTNNSSLUITR

(предварительно из текста были удалены пробелы и знаки препинания, буквы переведены в верхний регистр). Таким образом мы получаем последовательность из независимых символов, но частоты появления букв, характерные для английского языка, сохраняются. Энтропия такого текста по оценке Шеннона [89] составляет 4,14 бит на символ, что заметно выше 1 бита на символ в случае исходного текста. Оценим стойкость шифра с бегущим ключом при таком

выборе ключевой последовательности, используя неравенство (7.31):

$$h(M|E) \geq 1 + 4,14 - 4,7 = 0,44.$$

Положительная неопределенность означает невозможность однозначного дешифрования сообщения (без знания секретного ключа). Например, при длине зашифрованного сообщения  $n = 1000$  символов мощность множества почти равновероятных расшифровок (т.е. количество правдоподобных текстов на английском языке) у противника, согласно теореме 7.7,

$$|\Psi(E_1^{1000})| \geq 2^{1000h(M|E)} = 2^{440}.$$

□

Такой шифр в принципе невозможно вскрыть.

Другой способ усиления шифра с бегущим ключом состоит в построении ключевой последовательности путем суммирования нескольких текстов. В этой конструкции шифра имеется сообщение  $M = m_1 m_2 \dots$  и  $d$  текстов, используемых в качестве ключей —  $K^1 = k_1^1 k_2^1 \dots, \dots, K^d = k_1^d k_2^d \dots$ . Для простоты представления результата будем полагать, что сообщение и все ключи являются различными порождениями одного стационарного эргодического источника с алфавитом из  $r$  букв,  $A = \{0, 1, \dots, r-1\}$ . На практике это означает, что сообщение и ключи — тексты на одном, например, на английском, языке. Символы шифротекста  $E = e_1 e_2 \dots$  вычисляются по формуле

$$e_i = (m_i + k_i^1 + \dots + k_i^d) \bmod r. \quad (7.33)$$

Формула для расшифрования очевидна.

Оценку стойкости данного шифра дает следующая

**Теорема 7.9.** *Если шифр с бегущим ключом строится на основе  $d$  ключей по правилу (7.33), сообщение  $M$  и все ключи генерируются одним стационарным эргодическим источником, то неопределенность символа сообщения  $M$  при наличии перехваченного шифротекста  $E$ ,  $h(M|E)$ , характеризуется следующим неравенством:*

$$h(M|E) \geq \frac{1}{d}((d+1)h(M) - \log r), \quad (7.34)$$

где  $h(M)$  — энтропия источника.

**Доказательство.** Так как сообщение и ключи независимы, то в соответствии с (7.17)

$$h(M) + h(K^1) + \cdots + h(K^d) = h(M, K^1, \dots, K^d).$$

Шифротекст  $E$  однозначно определяется по  $M, K^1, \dots, K^d$ . Поэтому

$$h(M, K^1, \dots, K^d) = h(E, M, K^1, \dots, K^d).$$

Применяя формулу (7.16), получаем

$$\begin{aligned} h(E, M, K^1, \dots, K^d) &= h(E) + h(M|E) + h(K^1|E, M) + \cdots \\ &+ h(K^{d-1}|E, M, K^1, \dots, K^{d-2}) + h(K^d|E, M, K^1, \dots, K^{d-1}). \end{aligned} \quad (7.35)$$

Заметим, что последнее слагаемое в (7.35) равно нулю, т.к.  $K^d$  однозначно определяется из (7.33) при известных  $E, M, K^1, \dots, K^{d-1}$ . Также заметим, что в силу независимости сообщения и ключей все условные энтропии в (7.35) обусловлены только шифротекстом  $E$ . Поэтому мы можем записать результирующее равенство

$$\begin{aligned} h(M) + h(K^1) + \cdots + h(K^d) &= \\ &= h(E) + h(M|E) + h(K^1|E) + \cdots + h(K^{d-1}|E). \end{aligned} \quad (7.36)$$

Так как сообщение и ключи порождены одним источником,

$$h(M) = h(K^1) = \cdots = h(K^d).$$

Более того, т.к. сообщение и ключи участвуют в (7.33) абсолютно равноправно,

$$h(M|E) = h(K^1|E) = \cdots = h(K^{d-1}|E).$$

Это позволяет нам представить (7.36) в виде

$$(d+1)h(M) = h(E) + dh(M|E),$$

откуда

$$h(M|E) = \frac{(d+1)h(M) - h(E)}{d}.$$

Учитывая, что  $h(E) \leq \log r$ , получаем (7.34). □

Пример 7.5. Пусть сообщение и ключи — тексты на английском языке с энтропией 1 бит на символ. Будем использовать 4 ключа. Тогда в соответствии с (7.34)

$$h(M|E) \geq \frac{1}{4}((4+1) \cdot 1 - 4,7) = 0,075.$$

При длине зашифрованного сообщения  $n = 1000$  символов мощность множества почти равновероятных расшифровок у противника

$$|\Psi(E_1^{1000})| \geq 2^{75}.$$

Это означает, что шифр невозможно вскрыть. Если бы использовалось только 3 ключа, то мы получили бы  $h(M|E) \geq -0,23$ . В этом случае теоретико-информационных гарантий стойкости шифра нет. Остается только найти алгоритм, с помощью которого сообщение и, возможно, все ключи будут однозначно восстановлены из шифротекста.  $\square$

## 7.7. Расстояние единственности шифра с секретным ключом

В предыдущих разделах мы познакомились с конструкциями шифров, которые невозможно вскрыть. Но для их реализации нужны длинные одноразовые ключи, которые далеко не всегда могут быть легко установлены между отправителем и получателем сообщений. Во многих практических схемах криптографии применяются короткие (скажем, 128–256 бит) одноразовые ключи. В последних разделах этой главы мы изучим теоретико-информационные свойства шифров с короткими одноразовыми ключами и обсудим, какие гарантии стойкости могут существовать в этом случае.

Рассмотрим крипtosистему с секретным ключом, схема которой показана на рис. 1.1 (стр. 6). Пусть источник порождает сообщение  $M = m_1m_2\dots m_n$ . Алиса и Боб обладают секретным ключом  $K$  фиксированной длины, известным только им, и пусть  $E = e_1e_2\dots e_n$  — сообщение, зашифрованное при помощи этого ключа.

Пример 7.6. Пусть источник порождает буквы из алфавита  $A = \{a, b, c\}$  с вероятностями  $P(a) = 0,8$ ,  $P(b) = 0,15$ ,  $P(c) = 0,05$ , и пусть это источник без памяти. Пусть шифратор, применяя ключ

$K$ , заменяет буквы в исходном сообщении, используя какую-либо перестановку символов:

$$\begin{array}{ll} (a, b, c) & K = 1 \\ (a, c, b) & K = 2 \\ (b, a, c) & K = 3 \\ (b, c, a) & K = 4 \\ (c, a, b) & K = 5 \\ (c, b, a) & K = 6, \end{array}$$

т.е. ключ принимает значения от 1 до 6, и если, например,  $K = 5$ , то в исходном тексте осуществляется следующая замена символов:  $a \rightarrow c$ ,  $b \rightarrow a$ ,  $c \rightarrow b$ .

Пусть Ева перехватила зашифрованное сообщение

$$E = cccbc$$

и хочет определить значение ключа. Оценим количественно вероятности использования всех возможных ключей, используя формулу Бейеса

$$P(K_i|E) = \frac{P(K_i)P(E|K_i)}{\sum_{j=1}^t P(K_j)P(E|K_j)},$$

где  $E$ ,  $K_1, \dots, K_t$  — некоторые события, причем  $K_i$  попарно несовместны и  $E \subset \sum_{i=1}^t K_i$ . В нашем случае событие  $E$  — это получение зашифрованного сообщения  $E = cccbc$ ,  $t = 6$ , а  $K_i$  означает, что выбран ключ  $K = i$ .

Мы предполагаем, что все ключи равновероятны, т.е.

$$P(K_1) = P(K_2) = P(K_3) = P(K_4) = P(K_5) = P(K_6) = 1/6.$$

Тогда

$$\begin{aligned} P(E|K_1) &= P(M = cccbc) = 0,05^4 \cdot 0,15 \approx 0,000001, \\ P(E|K_2) &= P(M = bbbcb) = 0,15^4 \cdot 0,05 \approx 0,000025, \\ P(E|K_3) &= P(M = cccac) = 0,8 \cdot 0,05^4 \approx 0,000005, \\ P(E|K_4) &= P(M = bbbab) = 0,8 \cdot 0,15^4 \approx 0,000405, \\ P(E|K_5) &= P(M = aaaca) = 0,8^4 \cdot 0,05 \approx 0,020480, \\ P(E|K_6) &= P(M = aaaba) = 0,8^4 \cdot 0,15 \approx 0,061440. \end{aligned}$$

Отсюда легко находим

$$\sum_{j=1}^6 P(K_j) P(E|K_j) \approx 0,013726,$$

и получаем по формуле Бейеса апостериорную вероятность того, что был использован ключ  $K = 1$ , при условии, что получено сообщение  $E = cccbc$ :

$$P(K_1|E) = P(M = cccbc|E = cccbc) \approx \frac{(1/6) \cdot 0,000001}{0,013726} \approx 0,000011.$$

Продолжая аналогично, находим, что с наибольшей вероятностью были использованы ключи  $K = 5$  и  $K = 6$ :

$$P(K_5|E) = P(M = aaaca|E = cccbc) \approx 0,25,$$

$$P(K_6|E) = P(M = aaaba|E = cccbc) \approx 0,75,$$

а вероятности всех остальных ключей меньше 0,01.  $\square$

Мы видим, что, перехватив всего 5 букв, Ева может определить ключ почти однозначно. Таким образом, из этого примера и из примера с шифром Цезаря в первой главе мы заключаем, что, по-видимому, существует некоторая длина перехваченного сообщения, после которой ключ может быть найден с вероятностью, близкой к единице.

**Утверждение 7.10** (о расстоянии единственности шифра (Шенон [28])). *Пусть рассматривается система с секретным ключом, и пусть  $H(K)$  – энтропия ключа. Пусть  $R$  – избыточность шифруемого сообщения, а  $n$  – длина сообщения, такая что*

$$H(K|e_1 \dots e_n) \approx 0, \quad (7.37)$$

*т.е. при этой длине перехваченного сообщения ключ почти однозначно восстановлен. Тогда справедливо неравенство*

$$n \geq \frac{H(K)}{R}. \quad (7.38)$$

Дадим несколько замечаний к этому утверждению.

1. Число  $n$ , удовлетворяющее неравенству (7.38), называется расстоянием единственности шифра. Это означает, что в среднем достаточно перехватить  $n$  букв зашифрованного сообщения для восстановления ключа.
2. Мы видим, что если избыточность сообщения  $R = 0$ , то ключ никогда не будет определен, т.к.  $n = \infty$ . Т.е. шифр невозможно взломать (мы видели это в примере с номером замка из первой главы).
3. Уменьшение избыточности может быть достигнуто за счет сжатия данных. Дело в том, что при сжатии данных энтропия «сжатого» текста сохраняется, а длина уменьшается. Следовательно, энтропия на букву в сжатом тексте больше, чем в исходном, а избыточность меньше, см. (7.23). Значит, после сжимающего кодирования расстояние единственности шифра увеличивается.
4. С практической точки зрения лучше использовать системы, в которых ключ меняется задолго до достижения расстояния единственности шифра.

**Доказательство.** Мы дадим здесь только основную идею доказательства. Пусть противник, перехватив переданный шифротекст  $E = e_1 e_2 \dots e_n$ , однозначно восстановил ключ, а тем самым и исходное сообщение. Значит, неопределенность противника уменьшилась на  $H(K) + H(m_1 \dots m_n)$ , т.к. он узнал и ключ, и исходное сообщение. При этом он получил  $n$  букв из  $r$ -буквенного алфавита  $A = \{a_1, \dots, a_r\}$ . Мы знаем, что максимальное значение энтропии  $h_\infty = \log r$ , значит неопределенность противника не может уменьшаться больше чем на  $n \log r$ . Отсюда

$$n \log r \geq H(K) + H(m_1 \dots m_n),$$

следовательно,

$$n (\log r - H(m_1 \dots m_n) / n) \geq H(K),$$

откуда получаем, что

$$n \geq \frac{H(K)}{\log r - H(m_1 \dots m_n) / n} = \frac{H(K)}{R}$$

(здесь мы воспользовались тем, что  $H(m_1 \dots m_n)/n \rightarrow h_\infty$ , и определением избыточности (7.23)).  $\square$

Пример 7.7. Оценим расстояние единственности для шифра из примера 7.6. Имеем  $H(K) = \log 6 \approx 2,58$ ,  $\log r = \log 3 \approx 1,58$ , и энтропия на букву источника равна

$$H = -(0,8 \log 0,8 + 0,15 \log 0,15 + 0,05 \log 0,05) \approx 0,88.$$

Поэтому

$$n \geq \frac{2,58}{1,58 - 0,88} \approx 3,7.$$

Мы видели, что пяти букв было практически достаточно для раскрытия ключа. И неравенство (7.38) хорошо согласуется с нашим примером.  $\square$

Поясним еще на одном примере, как взаимная зависимость символов увеличивает избыточность и тем самым уменьшает расстояние единственности.

Пример 7.8. Пусть дан марковский источник, т.е. источник с памятью, в котором вероятность появления очередного символа зависит только от предыдущего символа. Источник описывается следующей матрицей переходов:

$$\begin{array}{ccc} & a & b & c \\ a & 0 & 0,9 & 0,1 \\ b & 0 & 0,1 & 0,9 \\ c & 0,4 & 0,3 & 0,3 \end{array}$$

и начальными вероятностями  $P(a) = 0,19$ ,  $P(b) = 0,34$ ,  $P(c) = 0,47$ . Пусть, как и в примере 7.6, используется шифр с шестью возможными ключами, и пусть перехвачен зашифрованный текст

$$E = bbacbac.$$

Мы видим по матрице переходов, что сочетание  $aa$  невозможно (после буквы  $a$  вероятность появления снова буквы  $a$  равна нулю), а сочетание  $bb$  — маловероятно (вероятность появления  $b$  после  $b$  равна 0,1). Следовательно, скорее всего первая пара букв соответствует буквам  $cc$  исходного сообщения, т.е. при шифровании была использована подстановка  $c \rightarrow b$ . Тогда  $ac$  соответствует исходным парам

$ab$  или  $ba$ . По матрице мы видим, что сочетание  $ba$  невозможно, а возможно только  $ab$ . Поэтому мы можем предположить, что в качестве ключа была использована перестановка номер 2:

$$K = 2 \quad (a \rightarrow a, \ b \rightarrow c, \ c \rightarrow b).$$

Вычислим точные вероятности использования различных ключей, как в примере 7.6. Заметим, что вероятность конкретного сообщения источника равна произведению вероятности начальной буквы и вероятностей переходов от одной буквы к другой.

$$P(E|K_1) = P(M = bbacbac) = 0,34 \cdot 0,1 \cdot 0 = 0,$$

$$P(E|K_2) = P(M = ccabcab) = 0,47 \cdot 0,3 \cdot 0,4 \cdot 0,9 \cdot 0,9 \cdot 0,4 \cdot 0,9 = \\ = 0,016446,$$

$$P(E|K_3) = P(M = aabcabc) = 0,19 \cdot 0 = 0,$$

$$P(E|K_4) = P(M = aacbabc) = 0,19 \cdot 0 = 0,$$

$$P(E|K_5) = P(M = ccbacba) = 0,47 \cdot 0,3 \cdot 0,3 \cdot 0 = 0,$$

$$P(E|K_6) = P(M = bbcabca) = 0,34 \cdot 0,1 \cdot 0,9 \cdot 0,4 \cdot 0,9 \cdot 0,9 \cdot 0,4 = \\ = 0,003966.$$

Отсюда находим

$$\sum_{j=1}^6 P(K_j) P(E|K_j) \approx 0,003402,$$

и получаем по формуле Бейеса апостериорные вероятности использованных ключей при условии, что получено сообщение  $E = bbacbac$ :

$$P(K_1|E) = 0,$$

$$P(K_2|E) = P(M = ccabcab|E = bbacbac) \approx 0,8,$$

$$P(K_3|E) = 0,$$

$$P(K_4|E) = 0,$$

$$P(K_5|E) = 0,$$

$$P(K_6|E) = P(M = bbcabca|E = bbacbac) \approx 0,2.$$

Эти вычисления подтверждают справедливость приведенного ранее неформального рассуждения.  $\square$

Оценку расстояния единственности шифра можно использовать при конструировании криптосистем. Например, кажется разумным менять ключ на новый, когда общая длина зашифрованных с его помощью сообщений приближается к величине расстояния единственности.

Новые подходы к построению теоретически стойких криптосистем, связанные с применением методов специального кодирования, изложены в работах [18, 24, 25, 27, 78, 79] авторов данной книги. Предлагаемые там методы достаточно сложны для описания, однако они эффективны с вычислительной точки зрения и позволяют строить невскрываемые шифры с секретным ключом. Основная идея этих методов состоит в обеспечении путем кодирования нулевой избыточности сообщения, которое подлежит шифрованию. Один из таких методов будет рассмотрен в следующем разделе.

## 7.8. Идеальные криптосистемы

В разд. 7.2 было введено понятие совершенной секретности, а затем было показано, что шифр Вернама совершенно секретен. Мы видели, что в этом шифре длина ключа равна длине сообщения и ключ используется всего один раз. Если же мы хотим использовать короткий многоразовый ключ (что актуально для большинства практических приложений), то какой наилучший результат в смысле стойкости шифра мы можем достичь? При обсуждении утверждения 7.10 указывалось, что при нулевой избыточности сообщения расстояние единственности шифра бесконечно. Это означает, что даже короткий (или, что эквивалентно, применяемый много раз) ключ, используемый для шифрования очень длинного сообщения нулевой избыточности, не может быть раскрыт. А это, в свою очередь, означает, что у противника, пытающегося разгадать зашифрованный текст, остается неопределенность, равная неопределенности ключа. Очевидно, это лучшее, что может быть достигнуто в данных условиях (здесь можно снова вспомнить пример с кодовым замком из первой главы). Эти рассуждения подводят нас к понятию строго идеального шифра, впервые введенному Шенномоном [28].

Пусть сообщение  $M = m_1 m_2 \dots$  шифруется при помощи секретного ключа  $K = k_1 k_2 \dots k_s$ , в результате чего получается зашифрованное сообщение  $E = e_1 e_2 \dots$  (длины сообщения и шифротекста могут быть произвольными, в то время как ключ всегда имеет конеч-

ный размер  $s$ ). Пусть  $H(M)$ ,  $H(E)$  и  $H(K)$  — соответственно энтропии сообщения, шифротекста и ключа. Тогда  $H(M|E)$  представляет неопределенность сообщения, а  $H(K|E)$  — неопределенность ключа при условии, что известен шифротекст  $E$ .

**Определение 7.3.** Шифр называется *строго идеальным*, если

$$H(M|E) = H(K|E) = \min\{H(M), H(K)\}. \quad (7.39)$$

Если энтропия ключа меньше энтропии сообщения (что как раз и актуально для рассматриваемой нами ситуации), то (7.39) упрощается:

$$H(M|E) = H(K|E) = H(K). \quad (7.40)$$

В дальнейшем договоримся в качестве определения строго идеального шифра использовать (7.40).

Неформально, строгая идеальность шифра означает, что количество решений криптограммы равно количеству различных ключей и все решения равновероятны, как в примере с кодовым замком.

В этом разделе мы рассмотрим конструкцию идеальной системы, предложенную в [18], ограничившись описанием только основной идеи применительно к случаю, когда сообщение порождается двоичным источником без памяти с неизвестной статистикой, иными словами, генерируются буквы из алфавита  $A = \{a_1, a_2\}$ , причем буквы независимы, но вероятности их выбора неизвестны.

Пусть источник порождает потенциально неограниченные сообщения  $M = m_1 m_2 m_3 \dots$  и имеется ключ фиксированной длины  $K = k_1 k_2 \dots k_s$ ,  $s \geq 1$ . (Как мы упомянули выше, предполагается также, что энтропия источника на букву отлична от нуля, т.к. в противном случае вообще нет необходимости в передаче сообщений.) Будем последовательно разбивать сообщение источника на блоки символов длины  $n$ , где  $n > 1$  — параметр метода. Обозначим один из таких блоков через  $\bar{m}$ . Опишем преобразования, выполняемые над каждым таким  $n$ -буквенным блоком.

Вначале определим количество букв  $a_1$  и  $a_2$  в блоке  $\bar{m}$ . Пусть имеется  $n_1$  букв  $a_1$  и  $n_2 = n - n_1$  букв  $a_2$ . Определим  $u(\bar{m})$  как слово длины  $\lceil \log(n+1) \rceil$  бит, кодирующее  $n_1$ .

Теперь рассмотрим множество  $S$  всех последовательностей, состоящих из  $n_1$  букв  $a_1$  и  $n_2$  букв  $a_2$ . В этом множестве

$$|S| = \binom{n}{n_1} = \frac{n!}{n_1! n_2!}$$

элементов. Несмотря на то что нам не известны вероятности последовательностей из множества  $S$ , одно мы можем сказать точно — все они равны между собой (в силу независимости выбора отдельных букв сообщения). Зададим на множестве  $S$  некоторый порядок, например, расположим сообщения в порядке возрастания соответствующих им двоичных чисел (считая, что  $a_1 = 0$ ,  $a_2 = 1$ ). Вычислим номер данного конкретного блока  $\bar{m}$  внутри упорядоченного множества  $S$  (для вычисления такого номера известен эффективный алгоритм [17], описание которого выходит за рамки нашей книги). Обозначим этот номер через  $\omega(\bar{m})$ .

Разобьем множество  $S$  на непересекающиеся подмножества  $S_0, S_1, \dots, S_v$  с числами элементов, равными различным степеням двойки (например, если  $|S| = 21$ , то получаем три подмножества с числами элементов 16, 4 и 1). По  $\omega(\bar{m})$  определим, какому подмножеству принадлежит  $\bar{m}$  (обозначим номер такого подмножества через  $u(\bar{m})$ ), и найдем номер  $\bar{m}$  внутри данного подмножества (обозначим этот номер через  $w(\bar{m})$ ).

Посмотрим внимательно на номер сообщения внутри подмножества,  $w(\bar{m})$ . Замечательно то, что  $w(\bar{m})$  — это полностью случайная последовательность нулей и единиц (т.е. такая, где символы независимы, а вероятности нуля и единицы равны). Действительно,  $w(\bar{m})$  — это номер одной из равновероятных последовательностей букв в подмножестве из  $2^b$  элементов (для некоторого  $b$ ). Номера всех таких последовательностей — это всевозможные последовательности из  $b$  двоичных цифр. Но если все последовательности из  $b$  двоичных цифр равновероятны, то отдельные символы равновероятны и независимы.

Итак, обрабатывая описанным образом последовательные блоки сообщения, мы представляем сообщение в виде

$$M = u(\bar{m}_1)v(\bar{m}_1)w(\bar{m}_1)u(\bar{m}_2)v(\bar{m}_2)w(\bar{m}_2)\dots.$$

Теперь перейдем к описанию процесса шифрования преобразованного сообщения. На первый взгляд это может показаться странным, но слова  $u(\cdot)$  и  $v(\cdot)$  вообще не шифруются! Шифруются только слова  $w(\cdot)$  с использованием секретного ключа  $Kk$ . В качестве шифра можно, например, использовать побитовое сложение по модулю 2 с периодически продолженным ключом. Для описания этого шифра удобно занумеровать символы слов  $w(\cdot)$  подряд и обозначить их

через  $w_1 w_2 w_3 \dots$ . Тогда шифрование проводится по формуле

$$z_i = w_i \oplus k_i \bmod s.$$

В результате применения описанного метода мы зашифровали сообщение следующим образом:

$$M \longrightarrow E = u(\bar{m}_1)v(\bar{m}_1)z(\bar{m}_1)u(\bar{m}_2)v(\bar{m}_2)z(\bar{m}_2) \dots \quad (7.41)$$

По построению алгоритма ясно, что из правой части (7.41) можно восстановить сообщение, если знать  $K$ . Вначале нужно дешифровать символы слов  $w(\cdot)$ , используя формулу

$$w_i = z_i \oplus k_i \bmod s, \quad (7.42)$$

а затем из слов  $u(\cdot)v(\cdot)w(\cdot)$  восстанавливаются последовательные блоки сообщения.

Пример 7.9. Пусть требуется зашифровать сообщение

$$M = a_2 a_2 a_1 a_2 a_2 a_2 a_1 a_2 a_2 a_1$$

с трехбитовым ключом  $K = 011$ . Разобьем сообщение на два блока по пять символов,  $n = 5$ .

Выполним преобразование для первого блока  $\bar{m}_1 = a_2 a_2 a_1 a_2 a_2$ . Для этого блока  $n_1 = 1$  и  $u(\bar{m}_1) = (001)_2$ . Рассмотрим теперь упорядоченное множество всех сообщений, состоящих из одной буквы  $a_1$  и четырех букв  $a_2$  (табл. 7.2). Всего таких сообщений  $\frac{5!}{1!4!} = 5$ , поэтому имеем два подмножества  $S_0$  и  $S_1$  с числом элементов соответственно 4 и 1. Мы видим, что  $\bar{m}_1$  входит в  $S_0$  под номером  $2 = (10)_2$ .

Таблица 7.2. Множество равновероятных сообщений;  $n_1 = 1$ ,  $n_2 = 4$

Сообщение	Номер в $S$	$S_v$	$w$
$a_1 a_2 a_2 a_2 a_2$	000		00
$a_2 a_1 a_2 a_2 a_2$	001		01
$a_2 a_2 a_1 a_2 a_2$	010	$S_0$	10
$a_2 a_2 a_2 a_1 a_2$	011		11
$a_2 a_2 a_2 a_2 a_1$	100	$S_1$	-

Таким образом, мы получаем следующие два слова:  $v(\bar{m}_1) = 0$ ,  $w(\bar{m}_1) = (10)_2$ .

Теперь выполним преобразование для второго блока сообщения  $\bar{m}_2 = a_2a_1a_2a_2a_1$ . Для этого блока  $n_1 = 2$  и  $u(\bar{m}_2) = (010)_2$ . Рассмотрим упорядоченное множество всех сообщений, состоящих из двух букв  $a_1$  и трех букв  $a_2$  (табл. 7.3). Всего таких сообщений  $\frac{5!}{2!3!} = 10$ , поэтому имеем два подмножества  $S_0$  и  $S_1$  с числом элементов соответственно 8 и 2. Мы видим, что  $\bar{m}_2$  входит под номером 6 =  $(110)_2$

Таблица 7.3. Множество равновероятных сообщений;  $n_1 = 2$ ,  $n_2 = 3$

Сообщение	Номер в $S$	$S_v$	$w$
$a_1a_1a_2a_2a_2$	0000	$S_0$	000
$a_1a_2a_1a_2a_2$	0001		001
$a_1a_2a_2a_1a_2$	0010		010
$a_1a_2a_2a_2a_1$	0011		011
$a_2a_1a_1a_2a_2$	0100		100
$a_2a_1a_2a_1a_2$	0101		101
$a_2a_1a_2a_2a_1$	0110		110
$a_2a_2a_1a_1a_2$	0111		111
$a_2a_2a_1a_2a_1$	1000	$S_1$	0
$a_2a_2a_2a_1a_1$	1001	$S_1$	1

в  $S_0$ . Таким образом, мы получаем  $v(\bar{m}_2) = 0$ ,  $w(\bar{m}_2) = (110)_2$ .

В результате мы получили следующий двоичный код преобразованного сообщения:

001 0 10 010 0 110

(пробелы здесь поставлены только для удобства чтения; для однозначного декодирования они не нужны).

Теперь зашифруем преобразованное сообщение. Периодически продолженный ключ имеет вид  $\bar{k} = 011011\dots$ . Сложение битов слов  $w(\cdot)$  с этим ключом дает

$$\begin{array}{r} \oplus \begin{array}{ccccc} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array} \\ \hline 1 & 1 & 0 & 1 & 1 \end{array}.$$

Зашифрованное сообщение выглядит следующим образом:

$E = 001 0 11 010 0 011.$

□

Остановимся теперь на основных свойствах рассмотренного метода.

**Утверждение 7.11.** *Построенный шифр строго идеален.*

**Доказательство.** Как уже отмечалось при изложении метода, слово  $w(\bar{m})$  для каждого блока  $\bar{m}$  состоит из равновероятных и независимых символов 0 и 1, другими словами, «полностью» случайно. Так как блоки в сообщении независимы, то в преобразованном сообщении последовательность слов  $w(\bar{m}_1)w(\bar{m}_2)\dots = w_1w_2w_3\dots$  также полностью случайна. Но любая последовательность  $w_1w_2w_3\dots$  соответствует некоторому сообщению, и все такие сообщения равновероятны. Поэтому при подстановке любого ключа в дешифрующее выражение (7.42) мы получаем какое-либо решение, причем все решения равновероятны. В результате, имея только шифротекст  $E$ , мы ничего не можем сказать об использованном ключе, т.е.

$$H(K|E) = H(K).$$

Далее, каждому конкретному сообщению  $M$  соответствует одна и только одна последовательность  $w_1w_2w_3\dots$ , и при достаточно большой длине сообщения, а именно, такой, что длина последовательности  $w_1w_2w_3\dots$  не меньше длины ключа, каждому ключу, подставляемому в (7.42), соответствуют различные равновероятные сообщения. Поэтому

$$H(M|E) = H(K).$$

Таким образом, мы доказали, что (7.40) выполняется.  $\square$

Особенностью предложенного шифра является то, что шифрованию подвергается не все преобразованное сообщение, а только его часть. В приведенном примере даже может показаться, что слишком много информации остается «открытой». Какая все-таки доля информации скрывается этим шифром? Следующее утверждение дает ответ на этот вопрос.

**Утверждение 7.12.** *Пусть сообщение порождается источником без памяти с энтропией  $h$  на букву. Тогда для каждого блока  $\bar{m}$  из  $n$  символов сообщения средняя длина шифруемого слова  $w(\bar{m})$  удовлетворяет неравенству*

$$E(|w(\bar{m})|) > nh - 2 \log(n + 1) \quad (7.43)$$

(здесь  $E(\cdot)$  — математическое ожидание, а  $|\cdot|$  — длина).

**Доказательство.** Компонента кода  $u(\bar{m})$  может принимать любое значение от 0 до  $n$ , и поэтому ее максимальная энтропия равна  $\log(n+1)$ .

Слово  $v(\bar{m})$  может принимать любое значение от 0 до  $\nu$ , что связано с разбиением  $S$  на подмножества  $S_0, S_1, \dots, S_\nu$ . Очевидно, что  $\nu \leq \lfloor \log |S| \rfloor$ . Из известного неравенства  $|S| = \binom{n}{n_1} < 2^n$  получаем  $\log |S| < n$  и  $\nu \leq \log |S| < n$ . Поэтому максимальная энтропия слова  $v(\bar{m})$  строго меньше  $\log(n+1)$ .

Энтропия блока равна  $H(\bar{m}) = nh$  (т.к. символы порождаются источником без памяти). При преобразовании блока энтропия не изменяется, поэтому для энтропии слова  $w(\bar{m})$  имеем

$$H(w(\bar{m})) > nh - 2 \log(n+1)$$

(из общей энтропии мы вычли верхнюю границу максимальной энтропии слов  $u(\bar{m})$  и  $v(\bar{m})$ ). Но так как слово  $w(\bar{m})$  состоит из равновероятных и независимых символов 0 и 1, его средняя длина совпадает с энтропией, что завершает доказательство.  $\square$

Неформально, утверждение 7.12 говорит о том, что «почти вся» информация сообщения содержится в шифруемой компоненте кода  $w$ , если длина блока  $n$  достаточно велика. Иными словами, представленный шифр скрывает «почти всю» информацию. Причем даже полный перебор ключей не позволяет вскрыть шифр.

Конечно, рассмотренная конструкция идеальной криптосистемы может иметь различные варианты построения. Например, может представлять интерес вариант, в котором часть ключа используется для «закрытия» префикса (т.е. компонент  $u$  и  $v$  кода).

## Задачи и упражнения

**7.1.** Зашифровать сообщение  $\bar{m}$  шифром Вернама с ключом  $\bar{k}$ :

- $\bar{m} = 1001101011, \bar{k} = 0110100101,$
- $\bar{m} = 0011101001, \bar{k} = 1100011100,$
- $\bar{m} = 1000011100, \bar{k} = 1001011010,$
- $\bar{m} = 0011100010, \bar{k} = 0110111001,$

д.  $\bar{m} = 1001101011$ ,  $\bar{k} = 1000111010$ .

7.2. Пусть источник без памяти порождает буквы из алфавита  $A = \{a, b, c\}$  с вероятностями  $P(a)$ ,  $P(b)$ ,  $P(c)$ . Шифратор заменяет буквы, используя одну из шести возможных перестановок, как это делалось в примере 7.6. Определить апостериорные вероятности использованных ключей для заданного зашифрованного сообщения  $\bar{e}$ :

- $P(a) = 0.1$ ,  $P(b) = 0.7$ ,  $P(c) = 0.2$ ,  $\bar{e} = abaaacac$ ,
- $P(a) = 0.9$ ,  $P(b) = 0.09$ ,  $P(c) = 0.01$ ,  $\bar{e} = cbaccca$ ,
- $P(a) = 0.14$ ,  $P(b) = 0.06$ ,  $P(c) = 0.8$ ,  $\bar{e} = bbabbcab$ ,
- $P(a) = 0.7$ ,  $P(b) = 0.05$ ,  $P(c) = 0.25$ ,  $\bar{e} = cccacbbc$ ,
- $P(a) = 0.1$ ,  $P(b) = 0.7$ ,  $P(c) = 0.2$ ,  $\bar{e} = abbbbab$ .

7.3. Для источников задачи 7.2 вычислить энтропию и расстояние единственности.

7.4. По имеющемуся зашифрованному сообщению  $\bar{e}$  найти апостериорные вероятности использованных ключей и соответствующие им сообщения, если известно, что используется шифр примера 7.6, а сообщения порождаются марковским источником, описанным в примере 7.8:

- $\bar{e} = bcacbcacc$ ,
- $\bar{e} = caaabaaaba$ ,
- $\bar{e} = aacabcaac$ ,
- $\bar{e} = bcaaaacaa$ ,
- $\bar{e} = aaacaaaca$ .

# Глава 8. СОВРЕМЕННЫЕ ШИФРЫ С СЕКРЕТНЫМ КЛЮЧОМ

## 8.1. Введение

В этой главе мы рассмотрим вычислительно стойкие шифры с секретным ключом, которые, в принципе, могут быть вскрыты, но требуют для этого очень большого количества вычислений, скажем,  $10^{20}$  лет работы суперкомпьютера. Эти шифры обеспечивают шифрование и расшифрование данных со скоростями, значительно превышающими скорости шифров с открытыми ключами и теоретически стойких шифров, что и объясняет их широкое практическое использование. В последующих разделах мы опишем некоторые наиболее популярные шифры и режимы их функционирования, однако вначале, чтобы пояснить принципы построения этих шифров, продолжим пример с шифром Цезаря, начатый в первой главе.

Там мы рассмотрели атаку по шифротексту на шифр Цезаря. Было показано, что в случае избыточных сообщений шифр легко вскрывается путем перебора ключей. Поищем возможности повышения стойкости шифра Цезаря. Пожалуй, самое простое, что приходит в голову — увеличить количество возможных значений ключа. В этом случае Еве придется перебирать больше ключей, прежде чем она найдет единственный правильный.

Естественный способ увеличить количество возможных значений ключа для шифра Цезаря — использовать разные ключи для разных букв сообщения. Например, мы можем шифровать каждую нечетную букву ключом  $k_1$ , а четную — ключом  $k_2$ . Тогда секретный ключ  $k = (k_1, k_2)$  будет состоять из двух чисел, и количество возможных ключей будет  $32^2 = 1024$ . Зашифруем наше прежнее сообщение из первой главы ключом  $k = (3, 5)$ :

$$\text{ПЕРЕМЕНА} \xrightarrow{3,5} \text{ТКУКПКРЕ}. \quad (8.1)$$

Эта схема легко обобщается на произвольную длину секретного клю-

ча  $k = (k_1, k_2, \dots, k_t)_{32}$ . При  $t$  порядка 10 и выше задача полного перебора ключей становится практически нерешаемой.

Тем не менее, данный шифр легко вскрывается путем так называемого частотного криптоанализа. Для этого используется статистика языка, на котором написано передаваемое сообщение. При частотном криптоанализе перебор начинают с ключей, соответствующих наиболее часто встречающимся буквам и их сочетаниям. Например, известно, что в типичном тексте на русском языке буква О встречается чаще других. Смотрим на ТКУКПКРЕ в (8.1) и определяем, какие буквы встречаются чаще других на четных и нечетных местах. На четных местах это К. Предполагаем, что здесь зашифрована буква О, следовательно,  $k_2 = K - O = 28 \pmod{32}$ . На нечетных местах все буквы различны, поэтому для поиска  $k_1$  знание частот букв языка не помогает (дело в том, что для нашего примера взято очень короткое сообщение). Пытаемся, как и прежде, найти  $k_1$  путем перебора, но убеждаемся, что приемлемых расшифровок не получается. Это означает, что наша гипотеза о том, что О заменяется в шифре на К, неверна. Берем вместо О другую часто встречающуюся букву — букву Е. Вычисляем  $k_2 = K - E = 5$ . Повторяем аналогичные действия для поиска  $k_1$  и на этот раз находим решение  $k_1 = 3$ . В результате, чтобы расшифровать сообщение, из всех возможных 1024 ключей нам понадобилось проверить только 36 (мы проверяли  $(0, 28), \dots, (31, 28), (0, 5), \dots, (3, 5)$ ).

Попробуем слегка усложнить шифр, чтобы затруднить частотный криптоанализ. Нам нужно каким-то образом перемешать символы сообщения, заставить их влиять друг на друга, чтобы скрыть индивидуальные частоты их появления. По-прежнему будем использовать ключ  $k = (k_1, k_2)$  и шифровать сообщение блоками по два символа  $m_i, m_{i+1}$ . Один из простейших вариантов шифра может выглядеть так:

$$\begin{aligned} \tilde{m}_i &= m_i + m_{i+1}, \\ \tilde{m}_{i+1} &= m_{i+1} + \tilde{m}_i, \\ c_i &= \tilde{m}_i + k_1, \\ c_{i+1} &= \tilde{m}_{i+1} + k_2 \quad (\text{mod } 32) \end{aligned} \tag{8.2}$$

(все суммы вычисляются по модулю 32). Здесь  $m_i$  — нечетная буква исходного текста,  $m_{i+1}$  — четная буква,  $k_1, k_2$  — символы ключа, а  $c_i, c_{i+1}$  — получаемые символы шифротекста. Например, пара

символов ПЕ шифруется ключом  $k = (3, 5)$  следующим образом:

$$\begin{aligned}\tilde{m}_i &= \Pi + \text{E} = \Phi, \\ \tilde{m}_{i+1} &= \text{E} + \Phi = \text{Щ}, \\ c_i &= \Phi + 3 = \text{Ч}, \\ c_{i+1} &= \text{Щ} + 5 = \text{Ю},\end{aligned}$$

т.е. ПЕ превращается в ЧЮ.

Отметим, что этот шифр можно расшифровать. Алгоритм расшифрования, называемый обычно обратным шифром, выглядит следующим образом:

$$\begin{aligned}\tilde{m}_{i+1} &= c_{i+1} - k_2, \\ \tilde{m}_i &= c_i - k_1, \\ m_{i+1} &= \tilde{m}_{i+1} - \tilde{m}_i, \\ m_i &= \tilde{m}_i - m_{i+1} \pmod{32}.\end{aligned}\tag{8.3}$$

Применяя к нашему сообщению шифр (8.2) с ключом  $(3, 5)$ , получаем

$$\text{ПЕРЕМЕНА} \rightarrow \text{ФЩХЪСЦНН} \xrightarrow{3,5} \text{ЧЮШЯФЫРТ}.\tag{8.4}$$

Здесь для наглядности после первой стрелки показан промежуточный результат, получающийся после выполнения первых двух операций в (8.2) (это «перемешанный», но еще не зашифрованный текст сообщения). Мы видим, что данный шифр скрывает частоты появления отдельных символов, затрудняя частотный анализ. Конечно, он сохраняет частоты появления пар символов, но мы можем скрыть и их, если будем шифровать сообщения блоками по три символа и т.д.

Вообще, шифр (8.2) выглядит более сложным для Евы по сравнению с шифром (8.1), и он дает нам возможность рассмотреть еще одну ситуацию, связанную с ее действиями. До сих пор мы рассматривали атаки только по шифротексту. Но что произойдет, если Ева каким-то образом достала открытый текст, соответствующий ранее переданному зашифрованному сообщению? (Т.е. мы находимся в условиях атаки второго типа, см. главу 1, стр. 9.) Например, Ева имеет пару (ПЕРЕМЕНА, ТКУКПКРЕ) для шифра (8.1). Тогда она сразу вычисляет секретный ключ,  $k_1 = \text{T} - \Pi = 3$ ,  $k_2 = \text{K} - \text{E} = 5$ , и легко расшифровывает все последующие сообщения от Алисы к Бобу. При шифровании по (8.2) пара (ПЕРЕМЕНА, ЧЮШЯФЫРТ)

уже не дает такого очевидного решения, хотя и здесь оно довольно просто. Ева применяет две первые операции из (8.2) (не требующие знания секретного ключа) к слову ПЕРЕМЕНА, получает промежуточный результат ФЩХЪСЦНН и уже по паре (ФЩХЪСЦНН, ЧЮШЯФЫРТ), как и в первом случае, находит ключ  $k = 3, 5$ .

Как затруднить такие действия Евы? Идея проста. Будем при шифровании сообщения использовать шифр (8.2) два раза. Тогда получим:

$$\text{ПЕРЕМЕНА} \xrightarrow{3,5} \text{ЧЮШЯФЫРТ} \xrightarrow{3,5} \text{ШШЪЫТПЕЩ}. \quad (8.5)$$

Теперь, имея пару (ПЕРЕМЕНА, ШШЪЫТПЕЩ), Ева не может вычислить ключ, по крайней мере алгоритм ее действий не очевиден (она не может получить промежуточное значение ЧЮШЯФЫРТ, т.к. при его построении уже был использован секретный ключ, ей не известный).

В представленной схеме (8.5) отдельная реализация алгоритма (8.2) называется *раундом* или *итерацией* шифра.

Мы проиллюстрировали влияние на стойкость шифра таких параметров, как длина ключа, размер блока, количество раундов, а также показали необходимость введения «перемешивающих» преобразований. В реальных шифрах используются, в принципе, те же преобразования, но над более длинными цепочками символов и обладающие целым рядом дополнительных свойств, в частности, свойством нелинейности (в рассмотренном примере использованы только линейные операции). Это связано с наличием развитых методов криптоанализа, таких как дифференциальный и линейный криптоанализ; их описание может быть найдено во многих источниках, см., например, [87].

## 8.2. Блоковые шифры

**Блоковый шифр** (также «блочный шифр») можно определить как зависящее от ключа  $K$  обратимое преобразование слова  $X$  из  $n$  двоичных символов. Преобразованное с помощью шифра слово (или блок) будем обозначать через  $Y$ . Для всех рассматриваемых в этом разделе шифров длина слова  $Y$  равна длине слова  $X$ .

Таким образом, блочный шифр — это обратимая функция  $E$  (другим словами, такая, для которой существует обратная функ-

ция). Конкретный вид  $E_K$  этой функции определяется ключом  $K$ ,

$$\begin{aligned} Y &= E_K(X), \\ X &= E_K^{-1}(Y) \quad \text{для всех } X. \end{aligned}$$

Здесь  $E_K^{-1}$  обозначает преобразование расшифрования и называется *обратным шифром*.

Для криптографических приложений блоковый шифр должен удовлетворять ряду требований, зависящих от ситуации, в которой он используется. В большинстве случаев достаточно потребовать, чтобы шифр был стоек по отношению к атаке по выбранному тексту. Это автоматически подразумевает его стойкость по отношению к атакам по шифротексту и по известному тексту. Следует заметить, что при атаке по выбранному тексту шифр всегда может быть взломан путем перебора ключей. Поэтому требование стойкости шифра можно уточнить следующим образом.

Шифр *стоек* (при атаке по выбранному тексту), если для него *не существуют* алгоритмы взлома, существенно более быстрые, чем прямой перебор ключей.

Нам будет достаточно такого нестрогого определения стойкости. На самом деле, по состоянию на сегодняшний день, ни для одного используемого шифра не доказано соответствие этому определению стойкости. Реально можно говорить о следующем.

Шифр *считается стойким* (при атаке по выбранному тексту), если для него *неизвестны* алгоритмы взлома, существенно более эффективные, чем прямой перебор ключей.

Ниже мы приведем примеры некоторых практически используемых блоковых шифров. Наша задача будет состоять не только в том, чтобы дать достаточно подробное описание алгоритмов (их описание может быть найдено в литературе), но и в объяснении основных принципов построения блоковых шифров. Кроме того, наше описание может облегчить понимание материала, изложенного в официальных документах. Далее, на протяжении всей главы, мы будем изучать технику использования блоковых шифров для решения различных задач криптографии.

Раньше ни одна книга по криптографии не обходилась без описания шифра DES (Data Encryption Standard). Этот шифр был принят в качестве стандарта США в 1977 году. Его основные параметры: размер блока 64 бита, длина ключа 56 бит, 16 раундов. Этот шифр

интенсивно использовался более двух десятков лет и даже сегодня встречается в некоторых работающих системах. Несмотря на многочисленные атаки против DES, он так и не был взломан. Однако высокий уровень развития вычислительных средств позволяет сегодня вскрывать DES путем перебора ключей. Например, еще в 1993 году было опубликовано техническое описание системы стоимостью в один миллион долларов, позволяющей взламывать любой ключ DES за 7 часов. В результате DES не рекомендуется использовать во вновь создаваемых криптографических системах, и поэтому мы не описываем этот шифр. В 2001 году после специально объявленного конкурса в США был принят новый стандарт на блоковый шифр, названный AES (Advanced Encryption Standard), в основу которого был положен шифр Rijndael, разработанный бельгийскими специалистами.

Большинство современных блоковых шифров строятся по схемам, значительно отличающимся от DES. Тем не менее есть один действующий шифр, построенный на тех же принципах, что и DES, и представляющий для нас особый интерес. Это российский блоковый шифр, известный под кодовым названием «Магма», закрепленный первоначально в ГОСТ 28147-89 и «переутвержденный» с небольшими уточнениями в новом ГОСТ Р 34.12-2015.

## Шифр Магма

Шифр Магма был определен в ГОСТ 28147-89 [5], принятом в качестве стандарта в 1989 году. Затем в 2015 году он наряду с новым шифром «Кузнецик» (описание которого идет далее на стр. 189) вошел в состав нового стандарта ГОСТ Р 34.12-2015 [11]. Основные параметры шифра Магма следующие: длина ключа 256 бит, размер блока 64 бита, 32 раунда. Этот шифр более удобен для программной реализации, чем DES, имеются сведения о выигрыше по времени примерно в 1.5 раза. В отличие от DES, шифр Магма, по-видимому, не был предметом столь глубокого анализа со стороны мирового криптологического сообщества. Тем не менее, как отмечают специалисты, консервативный дизайн и величина основных параметров (длина ключа, размер блока, количество раундов) позволяют утверждать, что шифр вряд ли может быть слабым. Никаких эффективных атак против шифра Магма не опубликовано.

В основе шифра Магма, так же как и DES, лежит так называем-

мая структура Фейстела. Блок разбивается на две одинаковые части, правую  $R$  и левую  $L$ . Правая часть объединяется с ключевым элементом и посредством некоторого алгоритма шифрует левую часть. Перед следующим раундом левая и правая части меняются местами. Такая структура позволяет использовать один и тот же алгоритм как для шифрования, так и для расшифрования блока. Это особенно важно при аппаратной реализации, т.к. прямой и обратный шифры формируются одним и тем же устройством (различается только порядок подачи элементов ключа).

Перейдем к непосредственному описанию шифра Магма. Введем необходимые определения и обозначения. Последовательность из 32 бит будем называть словом. Блок текста  $X$  (64 бита), также как блок шифротекста  $Y$ , состоит из двух слов — левого  $L$  и правого  $R$ , причем  $L$  — старшее слово, а  $R$  — младшее. Секретный ключ  $K$  (256 бит) рассматривается состоящим из восьми слов  $K = K_0K_1 \dots K_7$ . На его основе строится так называемый раундовый (итерационный) ключ  $W = W_0W_1 \dots W_{31}$ , состоящий из 32 слов (метод построения раундового ключа будет дан позже).

Для работы шифра нужны 8 таблиц  $S_0, S_1, \dots, S_7$  (называемых также S-боксами). Каждая таблица содержит 16 четырехбитовых элементов, нумеруемых с 0 по 15. Будем обозначать через  $S_i[j]$   $j$ -й элемент  $i$ -й таблицы. Уточнение, установленное в стандарте 2015 года, предписывает такое наполнение этих таблиц:

$$\begin{aligned}
 S_0 &= (12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1) \\
 S_1 &= (6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15) \\
 S_2 &= (11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0) \\
 S_3 &= (12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11) \\
 S_4 &= (7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12) \\
 S_5 &= (5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0) \\
 S_6 &= (8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7) \\
 S_7 &= (1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2)
 \end{aligned}$$

В шифре используются следующие операции:

$+$  — сложение слов по модулю  $2^{32}$ ;

$\leftarrow$  — циклический сдвиг слова влево на указанное число бит;

- ⊕ – побитовое «исключающее или» двух слов, т.е. побитовое сложение по модулю 2.

**Алгоритм 8.1. БАЗОВЫЙ ЦИКЛ ШИФРА МАГМА**

**ВХОД:** Блок  $L, R$ , раундовый ключ  $W$ .

**ВЫХОД:** Преобразованный блок  $L, R$ .

1. FOR  $i = 0, 1, \dots, 31$  DO
2.      $k \leftarrow R + W_i$ ,  $k = (k_7 \dots k_0)_{16}$ ;
3.     FOR  $j = 0, 1, \dots, 7$  DO
4.          $k_j \leftarrow S_j[k_j]$ ;
5.          $L \leftarrow L \oplus (k \leftarrow 11)$ ;
6.          $L \longleftrightarrow R$ ;
7. RETURN  $R, L$ .

(На шаге 4 алгоритма используются отдельные четырехбитовые элементы переменной  $k$ ; на шаге 7 делается ещё одна перестановка слов, необходимая для правильного расшифрования.)

С помощью базового цикла осуществляется шифрование и расшифрование блока. Чтобы зашифровать блок, строим раундовый ключ

$$W = K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7 K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7 \\ K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7 K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0, \quad (8.6)$$

подаем на вход  $X$  и на выходе получаем  $Y$ .

Чтобы расшифровать блок, строим раундовый ключ

$$W = K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7 K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0 \\ K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0 K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0, \quad (8.7)$$

подаем на вход  $Y$  и на выходе получаем  $X$ .

Программная реализация обычно требует переработки цикла 3 алгоритма 8.1, т.к. работа с полубайтами неэффективна. Ясно, что то же самое преобразование может быть выполнено с использованием четырех таблиц по 256 байт или двух таблиц по 65536 полуслов. Например, при работе с байтами имеем  $k = (k_3 \dots k_0)_{256}$ , и шаги 3–4 алгоритма 8.1 переписываются следующим образом:

3.     FOR  $j = 0, 1, 2, 3$  DO
4.          $k_j \leftarrow T_j[k_j]$ .

Таблицы  $T_j$ ,  $j = 0, 1, 2, 3$ , вычисляются предварительно из S-боксов:

FOR  $i = 0, 1, \dots, 255$  DO  
 $T_j[i] \leftarrow S_{2j}[i \bmod 16] + 16S_{2j+1}[i \div 16]$ .

## Шифр RC6

RC6 был предложен Ривестом (Ronald Rivest) в 1998 году. Этот шифр принимал участие в конкурсе на новый стандарт блокового шифра США, проводимом в 1999–2001 годах, прошел в финал, но по совокупности таких показателей, как быстродействие, удобство использования и т.п., уступил первое место другому шифру (Rijndael). Тем не менее, активные исследования RC6 в ходе проведения конкурса не выявили в нем каких-либо слабых мест, и данный шифр высоко оценивается многими специалистами.

В RC6 пользователь задает размер слова ( $w$ ) 16, 32 или 64 бита, количество раундов ( $r$ ), длину ключа ( $l$ ) от 0 до 255 байт. Размер блока всегда составляет четыре слова. Конкретный вариант шифра обозначается по схеме RC6- $w/r/l$ . Например, RC6-32/20/16 — размер блока 128 бит, длина ключа 128 бит, 20 раундов (такой шифр исследовался в качестве кандидата на стандарт США).

В шифре используются следующие операции:

$+$ ,

$-$  — сложение и вычитание слов по модулю  $2^w$ ;

$*$  — умножение по модулю  $2^w$ ;

$\oplus$  — побитовое сложение по модулю 2 или, что то же самое, «исключающее или» двух слов;

$\leftrightarrow$ ,

$\hookleftarrow$  — циклические сдвиги слова влево или вправо на указанное число бит (заметим, что при длине слова  $w$  бит величина циклического сдвига фактически приводится по модулю  $w$ , причем, как правило, это приведение выполняется автоматически на машинном уровне, т.е. не требует дополнительных вычислений — процессор просто использует младшие  $\log w$  бит числа, задающего величину сдвига).

Шифрование и расшифрование блока данных производится с использованием одного и того же раундового ключа  $W$  длиной  $2r+4$  слова (нумерация слов с нуля), получаемого из секретного ключа  $K$  (метод формирования раундового ключа будет рассмотрен ниже).

### Алгоритм 8.2. RC6: шифрование блока данных

**ВХОД:** Блок из четырех слов  $(a, b, c, d)$ , раундовый ключ  $W$ .

**ВЫХОД:** Зашифрованный блок  $(a, b, c, d)$ .

1.  $b \leftarrow b + W_0$ ,  $d \leftarrow d + W_1$ ;
2. FOR  $i = 1, 2, \dots, r$  DO
  3.  $t \leftarrow (b * (2b + 1)) \leftrightarrow \log w$ ,
  4.  $u \leftarrow (d * (2d + 1)) \leftrightarrow \log w$ ,
  5.  $a \leftarrow ((a \oplus t) \leftrightarrow u) + W_{2i}$ ,
  6.  $c \leftarrow ((c \oplus u) \leftrightarrow t) + W_{2i+1}$ ,
  7.  $(a, b, c, d) \leftarrow (b, c, d, a)$ ;
8.  $a \leftarrow a + W_{2r+2}$ ,  $c \leftarrow c + W_{2r+3}$ ;
9. RETURN  $(a, b, c, d)$ .

Для расшифрования просто «прокручиваем» процесс в обратном порядке.

### Алгоритм 8.3. RC6: расшифрование блока данных

**ВХОД:** Блок из четырех слов  $(a, b, c, d)$ , раундовый ключ  $W$ .

**ВЫХОД:** Расшифрованный блок  $(a, b, c, d)$ .

1.  $c \leftarrow c - W_{2r+3}$ ,  $a \leftarrow a - W_{2r+2}$ ;
2. FOR  $i = r, r-1, \dots, 1$  DO
  3.  $(a, b, c, d) \leftarrow (d, a, b, c)$ ,
  4.  $t \leftarrow (b * (2b + 1)) \leftrightarrow \log w$ ,
  5.  $u \leftarrow (d * (2d + 1)) \leftrightarrow \log w$ ,
  6.  $a \leftarrow ((a - W_{2i}) \leftrightarrow u) \oplus t$ ,
  7.  $c \leftarrow ((c - W_{2i+1}) \leftrightarrow t) \oplus u$ ;
8.  $d \leftarrow d - W_1$ ,  $b \leftarrow b - W_0$ ;
9. RETURN  $(a, b, c, d)$ .

Формирование раундового ключа в RC6 более сложно, чем в шифре Магма, что характерно для большинства современных шифров. По сути дела речь идет о развертывании секретного ключа  $K$ .

в более длинную псевдослучайную последовательность  $W$  с целью затруднить криптоанализ шифра.

Обозначим через  $c$  число слов в ключе,  $c = 8l/w$ . Посредством нижеприведенного алгоритма секретный ключ  $K$  развертывается в раундовый ключ  $W$ :

$$K_0 K_1 \cdots K_{c-1} \longrightarrow W_0 W_1 \cdots W_{2r+3}.$$

В алгоритме используются «магические» числа:  $P_w$  — первые  $w$  бит двоичного разложения числа  $e - 2$ , где  $e$  — число Эйлера, служащее основанием натурального логарифма, и  $Q_w$  — первые  $w$  бит двоичного разложения числа  $\phi - 1$ , где  $\phi = (\sqrt{5} - 1)/2$  — «золотое сечение». В табл. 8.1 значения  $P_w$  и  $Q_w$  приведены в шестнадцатиричной системе счисления для различных длин слов  $w$ .

#### Алгоритм 8.4. RC6: ФОРМИРОВАНИЕ РАУНДОВОГО КЛЮЧА

**ВХОД:** Секретный ключ  $K$ .

**ВЫХОД:** Раундовый ключ  $W$ .

1.  $W_0 \leftarrow P_w$ ;
2. FOR  $i = 1, 2, \dots, 2r + 3$  DO  $W_i \leftarrow W_{i-1} + Q_w$ ;
3.  $a \leftarrow 0, b \leftarrow 0, i \leftarrow 0, j \leftarrow 0$ ;
4.  $k \leftarrow 3 \max(c, 2r + 4)$ ;
5. DO  $k$  раз
  6.  $W_i \leftarrow (W_i + a + b) \leftrightarrow 3, a \leftarrow W_i$ ,
  7.  $K_j \leftarrow (K_j + a + b) \leftrightarrow (a + b), b \leftarrow K_j$ ,
  8.  $i \leftarrow i + 1 \bmod 2r + 4, j \leftarrow j + 1 \bmod c$ ;
9. RETURN  $W$ .

Таблица 8.1. «Магические» числа для RC6

$w$	16	32	64
$P_w$	b7e1	b7e15163	b7e15162 8aed2a6b
$Q_w$	9e37	9e3779b9	9e3779b9 7f4a7c15

Рассмотрим кратко основные идеи построения алгоритма шифрования RC6. Заметим прежде всего, что, как и в шифрах DES и Магма, в каждом раунде RC6 одна половина блока используется для шифрования другой. Действительно, значение переменных  $t$  и  $u$

(строки 3–4 алгоритма 8.2) определяется только словами  $b$  и  $d$  соответственно. Затем эти переменные используются для модификации слов  $a$  и  $c$  перед сложением с элементами ключа (строки 5–6). Таким образом, в  $a$  и  $c$  вносится зависимость от  $b$  и  $d$ . В следующем раунде пары  $a, c$  и  $b, d$  меняются ролями, причем  $b$  и  $d$  переставляются между собой (строка 7 алгоритма). Вследствие такой структуры шифра количество раундов должно быть четным.

Выбранная для вычисления переменных  $t$  и  $u$  функция вида  $f(x) = (2x^2 + x) \bmod 2^w$  характеризуется сильной зависимостью старших бит своего значения от всех бит аргумента. Именно старшие биты  $f$  должны определять величину сдвига в строках 5–6 алгоритма. Поэтому необходимое их количество переносится в младшие разряды  $t$  и  $u$  посредством вращения  $\leftrightarrow \log w$ . При модификации  $a$  и  $c$  использование «исключающего или» достаточно традиционно, тогда как зависимое от данных вращение (операция  $\leftrightarrow$ ) — характерная особенность RC6 (заимствованная у его более простого предшественника RC5).

Строки 1 и 8 алгоритма 8.2 скрывают значения слов, которые не изменились в ходе первого и последнего раундов.

Рекомендуемое количество раундов  $r = 20$  связано с результатами исследования стойкости шифра по отношению к дифференциальному и линейному криптоанализу.

В ходе исследований шифра слабых ключей не обнаружено, т.е. любой ключ, даже нулевой, обеспечивает заявляемую высокую стойкость шифра. Предполагается, что для RC6 не существует алгоритма взлома, лучшего, чем перебор ключей.

## Шифр Rijndael (AES)

Авторами шифра Rijndael (читается «Рейндэл») являются бельгийские специалисты В. Реймен (Vincent Rijmen) и Й. Дэмен (Joan Daemen). Этот шифр победил в упомянутом выше конкурсе AES и в 2001 году был принят в качестве нового стандарта США. Возможно, он будет играть такую же важную роль в практической криптографии, какая на протяжении десятилетий принадлежала DES. Rijndael заметно сложнее для описания, чем RC6 и Магма, хотя его компьютерная реализация достаточно эффективна. Мы представим только основную идею построения шифра. Детальное описание и примеры реализации Rijndael даны в [47, 58].

Шифр Rijndael/AES характеризуется размером блока 128 бит, длиной ключа 128, 192 или 256 бит и количеством раундов 10, 12 или 14 в зависимости от длины ключа.

Договоримся об основных обозначениях. Под словом будем понимать последовательность из четырех байт (32 бита). Байты в слове нумеруются с 0 (младший байт) по 3 (старший байт). Блок данных состоит из четырех слов, которые нумеруются так же с 0 по 3. Для упрощения обозначений условимся, что номера байтов в слове и слов в блоке всегда приводятся по модулю 4 без явного указания на это. Блок данных рассматривается как матрица размером  $4 \times 4$  байта, причем слова соответствуют не строкам (как это обычно принято), а столбцам. Например  $X_{2,3}$  обозначает второй байт третьего слова блока  $X$ . Однако запись с одним индексом обозначает слово, например,  $X_3$  — третье слово блока. Мы будем следовать этому соглашению, чтобы не менять связанный с ним терминологии Rijndael.

Для выполнения преобразований в блоке используется раундовый ключ  $W$ , получаемый из секретного ключа  $K$ . Размер секретного ключа  $l$  определяет общее число раундов шифра  $r$ :

$$\begin{aligned} l = 128 &\Rightarrow r = 10, \\ l = 192 &\Rightarrow r = 12, \\ l = 256 &\Rightarrow r = 14. \end{aligned}$$

Раундовый ключ состоит из блоков (по 128 бит), количество которых равно числу раундов плюс 1,

$$W = W_0, W_1, \dots, W_r.$$

В основу разработки Rijndael были положены три критерия: стойкость по отношению ко всем известным атакам, скорость и компактность кода, простота дизайна. В отличие от предыдущих рассмотренных нами шифров, Rijndael не использует какой-либо аналог структуры Фейстела. Каждый раунд состоит из трех различных обратимых преобразований, называемых слоями:

- 1) линейный смешивающий слой гарантирует высокую степень взаимопроникновения символов блока для маскировки статистических связей;
- 2) нелинейный слой реализован с помощью S-боксов, имеющих оптимальную нелинейность, и предотвращает возможность ис-

пользования дифференциального, линейного и других современных методов криптоанализа;

- 3) слой сложения с ключом выполняет непосредственно шифрование.

Шифр начинается и заканчивается сложением с ключом. Это позволяет закрыть вход первого раунда при атаке по известному тексту и сделать криптографически значимым результат последнего раунда.

#### Алгоритм 8.5. RIJNDAEL: ШИФРОВАНИЕ БЛОКА

**ВХОД:** Блок  $X$ , раундовый ключ  $W$ .

**ВЫХОД:** Блок  $Y$ .

1.  $Y \leftarrow X \oplus W_0$ ;
2. FOR  $i = 1, 2, \dots, r - 1$  DO
  3.  $Y \leftarrow \text{SubBytes}(Y)$ ,
  4.  $Y \leftarrow \text{ShiftRows}(Y)$ ,
  5.  $Y \leftarrow \text{MixColumns}(Y)$ ,
  6.  $Y \leftarrow Y \oplus W_i$ ;
  7.  $Y \leftarrow \text{SubBytes}(Y)$ ,
  8.  $Y \leftarrow \text{ShiftRows}(Y)$ ,
  9.  $Y \leftarrow Y \oplus W_r$ ;
10. RETURN  $Y$ .

Процедура  $\text{SubBytes}$  (замена байтов) реализует слой нелинейного преобразования. Две другие процедуры  $\text{ShiftRows}$  (сдвиг строк) и  $\text{MixColumns}$  (перемешивание столбцов) представляют линейный смешивающий слой. Слой сложения с ключом реализован с помощью побитового «исключающего или»  $\oplus$ .

Обратим внимание на то, что в последнем раунде отсутствует преобразование  $\text{MixColumns}$ . На первый взгляд это кажется странным решением, ухудшающим структуру шифра. Но это не так. Обозначим шаги с 3 по 6 алгоритма соответственно через  $B$ ,  $R$ ,  $C$  и  $K$ , и запишем всю последовательность действий в виде линейной цепочки:

$$KBRCKBRCK \cdots BRCKBRK. \quad (8.8)$$

Чтобы расшифровать блок, нужно выполнить все действия в обратном порядке, используя обратные операции. Как будет показано ни-

же, преобразования B и R можно поменять местами без изменения результата, преобразования C и K также можно поменять местами при условии некоторого изменения раундового ключа. При выполнении такой перестановки последовательность (8.8) запишется в виде

$$\text{KRBKCRBKC} \dots \text{RBKCRBK}. \quad (8.9)$$

Последовательность (8.9), прочитанная справа налево, теперь точно совпадает с (8.8). Это означает, что блок можно дешифровать, используя ту же последовательность действий, что и при его шифровании. Здесь важно то, что, как будет показано ниже, последовательность действий BRCK может быть эффективно реализована с помощью табличных вычислений, причем все необходимые таблицы задаются константно, т.е. не зависят ни от ключа, ни от данных. Сейчас мы перейдем непосредственно к описанию преобразований BRC, хотя в интересах практической реализации шифра Rijndael можно просто использовать табличные алгоритмы, приведенные ниже, и не вдаваться в подробности выполнения преобразований.

В шифре Rijndael, как и во многих других современных шифрах, используется арифметика полиномов. Мы рассмотрим несколько базовых операций этой арифметики, но для более глубокого изучения вопроса адресуем читателя к литературе (см., например, [13]).

Каждый байт данных (в общем случае любую последовательность бит) можно рассматривать как полином с коэффициентами 0 и 1, причем действия с коэффициентами производятся по модулю 2. Если  $a$  — некоторое двоичное число, то  $a(x)$  — соответствующий ему полином. Например,

$$a = 10010011 \Rightarrow a(x) = x^7 + x^4 + x + 1,$$

т.е. степень  $x$  равна номеру бита в числе (при нумерации бит справа налево, начиная с нуля), а значение бита — это коэффициент при  $x$  (степени  $x$  с нулевыми коэффициентами не записываются). Иногда, для компактности, число  $a$  можно записать и в другой системе счисления, например, в десятичной, но соответствующий ему полином останется тем же самым:

$$a = 147 \Rightarrow a(x) = x^7 + x^4 + x + 1.$$

Первая важная операция — сложение полиномов. Оно выполняется просто путем побитового суммирования по модулю 2 соответ-

ствующих чисел:

$$a(x) + b(x) \equiv a \oplus b.$$

Отметим, что вследствие суммирования по модулю 2 вычитание полиномов эквивалентно сложению.

Умножение полинома на  $x$  выполняется путем сдвига влево на один бит, а деление на  $x$  (с отбрасыванием остатка) — путем сдвига вправо на один бит соответствующего числа:

$$a(x) \cdot x \equiv a \ll 1, \quad a(x)/x \equiv a \gg 1.$$

Вторая важная операция — модульное умножение полиномов. В частности, в шифре Rijndael полиномы-байты необходимо перемножать по модулю полинома  $m(x) = x^8 + x^4 + x^3 + x + 1$ . Для перемножения полиномов невысокой степени целесообразно использовать алгоритм умножения «в столбик». Как и в случае с целыми числами, если в результате необходимо получить остаток, то все промежуточные операции можно делать над остатками. При умножении полинома  $a(x)$  на  $b(x)$  нам необходимо многократно умножать  $a(x)$  на  $x$  и прибавлять результат к формируемому произведению, если в  $b(x)$  стоит единичный коэффициент при соответствующей степени  $x$ . Будем после каждого умножения на  $x$  выполнять приведение по модулю. Так как степени перемножаемых полиномов всегда меньше степени  $m(x)$ , то степень  $a(x) \cdot x$  может остаться меньшей либо стать равной степени  $m(x)$ . Если степень осталась меньше, то приведения по модулю не требуется. Если же она стала равной степени  $m(x)$ , то для приведения по модулю достаточно один раз вычесть полином  $m(x)$ . Эти рассуждения приводят нас к следующему алгоритму.

### Алгоритм 8.6. Умножение полиномов по модулю

**ВХОД:** Полиномы  $a(x)$ ,  $b(x)$  степени меньше  $n$ ,  
полином  $m(x)$  степени  $n$ .

**ВЫХОД:** Полином  $c(x) = a(x) \cdot b(x) \bmod m(x)$ .

1.  $c \leftarrow 0$ .
2. WHILE  $b \neq 0$  DO
  3. IF  $b_0 = 1$  THEN  $c \leftarrow c \oplus a$ ;
  4.  $a \leftarrow a \ll 1$ ;
  5. IF  $a_n = 1$  THEN  $a \leftarrow a \oplus m$ ;
  6.  $b \leftarrow b \gg 1$ .
7. RETURN  $c$ .

Использующийся в шифре Rijndael полином  $m(x) = x^8 + x^4 + x^3 + x + 1$  нельзя представить как произведение других полиномов с двоичными коэффициентами (это аналог простого числа в арифметике полиномов). В результате любой полином  $a(x) \neq 0$  имеет инверсию, т.е. полином  $a^{-1}(x)$ , такой, что  $a(x) \cdot a^{-1}(x) \bmod m(x) = 1$  (инверсия вычисляется с помощью обобщенного алгоритма Евклида, в котором все числа заменены на полиномы). На языке теории групп говорят, что в этом случае полиномы-байты образуют поле  $\mathbb{F}_{2^8}$ .

Каждое слово данных представляется уже как полином с коэффициентами в  $\mathbb{F}_{2^8}$ . Например,

$$0x7500a302 = (0x75)x^3 + (0xa3)x + 2$$

(префикс 0x означает запись числа в шестнадцатеричной системе).

Преобразование  $\text{SubBytes}(Y)$  действует независимо на каждый байт  $b$  в  $Y$ :

$$b(x) \leftarrow b^{-1}(x) \bmod m(x) \quad (0 \leftarrow 0),$$

$$b(x) \leftarrow ((x^7 + x^6 + x^5 + x^4 + 1)b(x) + (x^7 + x^6 + x^2 + x)) \bmod (x^8 + 1).$$

Результаты этого преобразования, заранее подсчитанные для каждого байта от 0 до 255, заносятся в таблицу  $S$ . Тогда преобразование  $\text{SubBytes}(Y)$  сводится к выполнению для каждого байта  $b$  в  $Y$  операции

$$b \leftarrow S[b].$$

Содержимое таблицы  $S$  в явном виде приведено в [47].

Преобразование  $\text{ShiftRows}(Y)$  действует на каждую строку  $r_i$  в  $Y$ , т.е. на  $i$ -ю ( $i = 0, 1, 2, 3$ ) последовательность байт слов блока, по правилу

$$r_i \leftarrow r_i \leftarrow i$$

(здесь операция  $\leftarrow$  означает циклический сдвиг влево на  $i$  байт).

Преобразование  $\text{MixColumns}(Y)$  действует на каждый столбец  $c_i$  в блоке  $Y$ , т.е. на каждое машинное слово,  $i = 0, 1, 2, 3$ , по правилу

$$c_i(x) \leftarrow a(x) \cdot c_i(x) \bmod (x^4 + 1),$$

где  $a(x) = 3x^3 + x^2 + x + 2$ . Эта операция может быть записана в

матричном виде

$$c_t = \begin{bmatrix} c_{0,t} \\ c_{1,t} \\ c_{2,t} \\ c_{3,t} \end{bmatrix} \leftarrow \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} c_{0,t} \\ c_{1,t} \\ c_{2,t} \\ c_{3,t} \end{bmatrix}. \quad (8.10)$$

При умножении матрицы на вектор операции выполняются в поле  $\mathbb{F}_{2^8}$ , т.е. как сложение байтов-полиномов и их умножение по модулю  $m(x)$ .

Очевидно, что операцию SubBytes можно поменять местами с ShiftRows, не изменяя результат, т.к. эти преобразования воздействуют только на индивидуальные байты. Далее, используя дистрибутивность умножения полиномов можем записать

$$\text{MixColumns}(Y \oplus W_t) = \text{MixColumns}(Y) \oplus \text{MixColumns}(W_t).$$

Таким образом, MixColumns и сложение с ключом также можно поменять местами при условии, что соответствующие блоки раундового ключа (кроме первого и последнего) были предварительно подвергнуты обратному преобразованию  $\text{MixColumns}^{-1}(W_t)$ . Все это служит обоснованием идентичности последовательностей действий (8.8) и (8.9) с модифицированным ключом. В результате получаем следующий алгоритм обратного шифра.

### Алгоритм 8.7. RIJNDAEL: РАСШИФРОВАНИЕ БЛОКА

**ВХОД:** Блок  $Y$ , раундовый ключ  $W$ .

**ВЫХОД:** Блок  $X$ .

1.  $X \leftarrow Y \oplus W_r;$
2. FOR  $i = r-1, r-2, \dots, 1$  DO
  3.  $X \leftarrow \text{SubBytes}^{-1}(X),$
  4.  $X \leftarrow \text{ShiftRows}^{-1}(X),$
  5.  $X \leftarrow \text{MixColumns}^{-1}(X),$
  6.  $X \leftarrow X \oplus W_t;$
  7.  $X \leftarrow \text{SubBytes}^{-1}(X),$
  8.  $X \leftarrow \text{ShiftRows}^{-1}(X),$
  9.  $X \leftarrow X \oplus W_0;$
10. RETURN  $X$ .

Обратные преобразования, использованные в алгоритме, определяются естественным образом.

Преобразование  $\text{SubBytes}^{-1}(X)$  действует на каждый байт  $b$  в блоке  $X$ :

$$b(x) \leftarrow (b(x) - (x^7 + x^6 + x^2 + x))(x^7 + x^5 + x^2) \bmod (x^8 + 1),$$

$$b(x) \leftarrow b^{-1}(x) \bmod m(x) \quad (0 \leftarrow 0),$$

где  $x^7 + x^5 + x^2 = (x^7 + x^6 + x^5 + x^4 + 1)^{-1} \bmod (x^8 + 1)$ . Результаты этого преобразования заносятся в таблицу  $S^{-1}$ . Содержимое таблицы  $S^{-1}$  в явном виде приведено в [47].

Преобразование  $\text{ShiftRows}^{-1}(X)$  действует на каждую строку  $r_i$  в  $X$  по правилу

$$r_i \leftarrow r_i \hookrightarrow i$$

(здесь операция  $\hookrightarrow$  означает циклический сдвиг вправо на указанное число байт).

Преобразование  $\text{MixColumns}^{-1}(X)$  действует на каждый столбец  $c_i$  в блоке  $X$  по правилу

$$c_i(x) \leftarrow a^{-1}(x) \cdot c_i(x) \bmod (x^4 + 1),$$

где  $a^{-1}(x) = 11x^3 + 13x^2 + 9x + 14$ . То же самое в матричной записи выглядит так:

$$c_i = \begin{bmatrix} c_{0,i} \\ c_{1,i} \\ c_{2,i} \\ c_{3,i} \end{bmatrix} \leftarrow \begin{bmatrix} 14 & 11 & 13 & 09 \\ 09 & 14 & 11 & 13 \\ 13 & 09 & 14 & 11 \\ 11 & 13 & 09 & 14 \end{bmatrix} \cdot \begin{bmatrix} c_{0,i} \\ c_{1,i} \\ c_{2,i} \\ c_{3,i} \end{bmatrix}. \quad (8.11)$$

Как и в прямом преобразовании, операции выполняются в поле  $\mathbb{F}_{2^8}$ .

Особенностью представленных алгоритмов является то, что они могут быть значительно ускорены с помощью предварительно вычисленных и хранимых в памяти таблиц. Причем эти таблицы не зависят от секретного ключа или данных, т.е. определяются только самим алгоритмом могут быть «защиты» в программу в виде констант. Мы рассмотрим табличную реализацию шифра на компьютере с длиной слова 32 бита.

Обозначим текущее состояние блока через  $U$ . Последовательность шагов 3–6 алгоритма шифрования переводит блок данных из

состояния  $U$  в новое состояние  $Y$ . С учетом характера преобразований, выполняемых на шагах 3–6, мы можем записать, как вычисляется каждое  $j$ -е слово (т.е. каждый столбец) в  $Y$ :

$$\begin{bmatrix} Y_{0,j} \\ Y_{1,j} \\ Y_{2,j} \\ Y_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} S[U_{0,j}] \\ S[U_{1,j-1}] \\ S[U_{2,j-2}] \\ S[U_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} W_{i,0,j} \\ W_{i,1,j} \\ W_{i,2,j} \\ W_{i,3,j} \end{bmatrix}$$

(самый правый столбец в выражении — это  $j$ -е слово  $W_i$ ). Раскрывая матричное умножение, получаем

$$Y_j = S[U_{0,j}] \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \\ 3 \end{bmatrix} \oplus S[U_{1,j-1}] \cdot \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix} \oplus S[U_{2,j-2}] \cdot \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} \oplus \dots \oplus S[U_{0,j-3}] \cdot \begin{bmatrix} 1 \\ 1 \\ 3 \\ 2 \end{bmatrix} \oplus W_{i,j}.$$

Определим четыре таблицы

$$T_0[b] = \begin{bmatrix} S[b] \cdot 2 \\ S[b] \\ S[b] \\ S[b] \cdot 3 \end{bmatrix}, \quad T_1[b] = \begin{bmatrix} S[b] \cdot 3 \\ S[b] \cdot 2 \\ S[b] \\ S[b] \end{bmatrix},$$

$$T_2[b] = \begin{bmatrix} S[b] \\ S[b] \cdot 3 \\ S[b] \cdot 2 \\ S[b] \end{bmatrix}, \quad T_3[b] = \begin{bmatrix} S[b] \\ S[b] \\ S[b] \\ S[b] \cdot 3 \\ S[b] \cdot 2 \end{bmatrix}.$$

Каждая таблица строится для  $b$ , пробегающего значения от 0 до 255, и состоит из 256 четырехбайтовых слов. Операция умножения при вычислении таблиц — это умножение полиномов по модулю  $m(x)$ . Таблицы  $T$  не зависят ни от ключа, ни от данных и могут быть сформированы заранее. С использованием этих таблиц  $j$ -е слово блока вычисляется следующим образом:

$$Y_j = T_0[U_{0,j}] \oplus T_1[U_{1,j-1}] \oplus T_2[U_{2,j-2}] \oplus T_3[U_{3,j-3}] \oplus W_{i,j}.$$

Сейчас мы готовы записать алгоритм шифрования в табличной форме.

**Алгоритм 8.8. RIJNDAEL: ШИФРОВАНИЕ БЛОКА  
(ТАБЛИЧНЫЙ ВАРИАНТ)**

**ВХОД:** Блок  $X$ , раундовый ключ  $W$ .

**ВЫХОД:** Блок  $Y$ .

1.  $U \leftarrow X \oplus W_0$ ;
2. FOR  $i = 1, 2, \dots, r - 1$  DO
  3. FOR  $j = 0, 1, 2, 3$  DO
    4.  $Y_j \leftarrow T_0[U_{0,j}] \oplus T_1[U_{1,j-1}] \oplus T_2[U_{2,j-2}] \oplus T_3[U_{3,j-3}] \oplus W_{i,j}$ ;
  5.  $U \leftarrow Y$ ;
  6. FOR  $i = 0, 1, 2, 3$  DO
    7. FOR  $j = 0, 1, 2, 3$  DO
      8.  $Y_{i,j} \leftarrow S[U_{i,j-i}]$ ;
    9.  $Y \leftarrow Y \oplus W_r$ ;
  10. RETURN  $Y$ .

Заметим, что S-боксы в большом количестве содержатся в таблицах  $T$ . Например, в качестве S-бокса на шаге 8 можно использовать младшие байты таблицы  $T_2$ .

Чтобы построить табличный алгоритм для обратного шифра, нужно повторить рассуждения, приведенные выше, по отношению к обратным преобразованиям. В результате мы получаем обратные таблицы  $T^{-1}$ :

$$T_0^{-1}[b] = \begin{bmatrix} S^{-1}[b] \cdot 14 \\ S^{-1}[b] \cdot 9 \\ S^{-1}[b] \cdot 13 \\ S^{-1}[b] \cdot 11 \end{bmatrix}, \quad T_1^{-1}[b] = \begin{bmatrix} S^{-1}[b] \cdot 11 \\ S^{-1}[b] \cdot 14 \\ S^{-1}[b] \cdot 9 \\ S^{-1}[b] \cdot 13 \end{bmatrix},$$

$$T_2^{-1}[b] = \begin{bmatrix} S^{-1}[b] \cdot 13 \\ S^{-1}[b] \cdot 11 \\ S^{-1}[b] \cdot 14 \\ S^{-1}[b] \cdot 9 \end{bmatrix}, \quad T_3^{-1}[b] = \begin{bmatrix} S^{-1}[b] \cdot 9 \\ S^{-1}[b] \cdot 13 \\ S^{-1}[b] \cdot 11 \\ S^{-1}[b] \cdot 14 \end{bmatrix}.$$

**Алгоритм 8.9. RIJNDAEL: РАСШИФРОВАНИЕ БЛОКА  
(ТАБЛИЧНЫЙ ВАРИАНТ)**

**ВХОД:** Блок  $Y$ , раундовый ключ  $W$ .

**ВЫХОД:** Блок  $X$ .

1.  $U \leftarrow Y \oplus W_0;$
2. FOR  $i = r - 1, r - 2, \dots, 1$  DO
3.     FOR  $j = 0, 1, 2, 3$  DO
4.          $X_j \leftarrow T_0^{-1}[U_{0,j}] \oplus T_1^{-1}[U_{1,j+1}] \oplus T_2^{-1}[U_{2,j+2}] \oplus \oplus T_3^{-1}[U_{3,j+3}] \oplus W_{i,j};$
5.      $U \leftarrow X;$
6.     FOR  $i = 0, 1, 2, 3$  DO
7.         FOR  $j = 0, 1, 2, 3$  DO
8.              $X_{i,j} \leftarrow S^{-1}[U_{i,j+1}];$
9.      $X \leftarrow X \oplus W_r;$
10. RETURN  $X$ .

Последнее, что нам осталось рассмотреть, это процесс формирования раундового ключа. В прямом и обратном шифре нам было удобно делить раундовый ключ  $W$  на блоки по четыре слова. Однако формирование ключа проходит в пословном режиме, поэтому условимся обозначать буквой  $w$  с индексом отдельное слово в  $W$ , нумеруя слова с нуля. Как следует из описания шифра, раундовый ключ  $W$  должен состоять из  $r + 1$  блоков, где  $r$  — количество раундов в шифре. Поэтому количество слов в  $W$  равно  $4(r + 1)$ . Число же слов в секретном ключе  $K$ , которое будем обозначать через  $c$ , равно 4, 6 или 8. Алгоритм построения  $W$  представлен ниже.

В данном алгоритме  $\text{SubWord}(t)$  — функция, применяющая  $S$ -бокс шифра к каждому байту слова  $t$  по схеме

$$[t_0, t_1, t_2, t_3] \longrightarrow [S[t_0], S[t_1], S[t_2], S[t_3]].$$

Преобразование  $\text{RotWord}(t)$  осуществляет циклический сдвиг слова  $t$  на один байт влево:

$$[t_0, t_1, t_2, t_3] \longrightarrow [t_1, t_2, t_3, t_0].$$

Массив раундовых констант  $Rcon$  состоит из слов

$$Rcon[i] = [y_i, 0, 0, 0], \quad y_i = x^{i-1} \bmod m(x).$$

**Алгоритм 8.10. RIJNDAEL: ФОРМИРОВАНИЕ РАУНДОВОГО КЛЮЧА**

**ВХОД:** Секретный ключ  $K$  из  $c$  слов.

**ВЫХОД:** Раундовый ключ  $W$  из  $4(r + 1)$  слов.

```

1.    $W \leftarrow K$  (с слов);
2.   FOR  $i = c, c + 1, \dots, 4(r + 1) - 1$  DO
3.        $t \leftarrow w_{i-1}$ ;
4.       IF  $i \bmod c = 0$  THEN
5.            $t \leftarrow \text{SubWord}(\text{RotWord}(t)) \oplus \text{Rcon}[i \bmod c]$ ;
6.       ELSE IF  $c = 8$  AND  $i \bmod c = 4$  THEN
7.            $t \leftarrow \text{SubWord}(t)$ ;
8.        $w_i \leftarrow w_{i-c} \oplus t$ ;
9.   RETURN  $w_0 \dots w_{4(r+1)-1}$ .

```

Выбранный метод построения раундового ключа должен способствовать решению следующих задач:

- 1) затруднить атаки на шифр при частично известном секретном ключе или при использовании зависимых (связанных общими правилами построения) ключей;
- 2) устраниить имеющиеся симметрии внутри раунда шифра и между раундами (для этого используется массив раундовых констант Rcon).

Как отмечалось при рассмотрении обратного шифра, чтобы получить раундовый ключ для дешифрования, необходимо подвергнуть преобразованию  $\text{MixColumns}^{-1}$  блоки  $W$  с первого по предпоследний.

## Шифр Кузнечик

Шифр Кузнечик является новым алгоритмом, включенным наряду с шифром Магма (стр. 171) в стандарт РФ ГОСТ Р 34.12-2015 [11]. Данный шифр был разработан ФСБ России с участием компании ИнфоТeKC и впервые представлен в [90]. Основные параметры шифра Кузнечик следующие: длина ключа 256 бит, размер блока 128 бит, 9 раундов. Блок состоит из 16 байт, которые мы будем нумеровать с 0 по 15,  $Y = (y_0, y_1, \dots, y_{15})$ .

По своей структуре шифр Кузнечик повторяет шифр Rijndael. В его раунде, как и в Rijndael, можно выделить три слоя (хотя такая терминология авторами шифра не используется):

- 1) линейный смещающий слой, построенный на базе линейного отображения байт;
- 2) нелинейный слой, реализованный с помощью байтового S-бокса;
- 3) слой сложения с раундовым ключом.

Рассмотрим вначале эти базовые преобразования. Начнем с нелинейного преобразования. Имеется массив  $S$ , последовательные элементы которого индексируются как  $S[0], S[1], \dots, S[255]$ :

$$S = (252, 238, 221, 017, 207, 110, 049, 022, 251, 196, 250, 218, 035, 197, 004, 077, 233, 119, 240, 219, 147, 046, 153, 186, 023, 054, 241, 187, 020, 205, 095, 193, 249, 024, 101, 090, 226, 092, 239, 033, 129, 028, 060, 066, 139, 001, 142, 079, 005, 132, 002, 174, 227, 106, 143, 160, 008, 011, 237, 152, 127, 212, 211, 031, 235, 052, 044, 081, 234, 200, 072, 171, 242, 042, 104, 162, 253, 058, 206, 204, 181, 112, 014, 086, 008, 012, 118, 018, 191, 114, 019, 071, 156, 183, 093, 135, 021, 161, 150, 041, 016, 123, 154, 199, 243, 145, 120, 111, 157, 158, 178, 177, 050, 117, 025, 061, 255, 053, 138, 126, 109, 084, 198, 128, 195, 189, 013, 087, 223, 245, 036, 169, 062, 168, 067, 201, 215, 121, 214, 246, 124, 034, 185, 003, 224, 015, 236, 222, 122, 148, 176, 188, 220, 232, 040, 080, 078, 051, 010, 074, 167, 151, 096, 115, 030, 000, 098, 068, 026, 184, 056, 130, 100, 159, 038, 065, 173, 069, 070, 146, 039, 094, 085, 047, 140, 163, 165, 125, 105, 213, 149, 059, 007, 088, 179, 064, 134, 172, 029, 247, 048, 055, 107, 228, 136, 217, 231, 137, 225, 027, 131, 073, 076, 063, 248, 254, 141, 083, 170, 144, 202, 216, 133, 097, 032, 113, 103, 164, 045, 043, 009, 091, 203, 155, 037, 208, 190, 229, 108, 082, 089, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 057, 075, 099, 182).$$

Содержимое массива  $S$  можно рассматривать как некоторую «случайную» перестановку всех возможных байт (однако в [40] был найден алгоритм, генерирующий эту перестановку). Нелинейное преобразование блока  $Y$  будем обозначать  $S(Y)$ . Оно заключается в замене каждого отдельного байта  $b$  в блоке по схеме

$$b \leftarrow S[b].$$

Для описания линейного преобразования введем функцию  $\text{lm}$ . Данная функция вычисляет линейную комбинацию байт блока  $Y$  по следующей формуле:

$$\begin{aligned} \text{lm}(Y) = & 148 \cdot y_0 \oplus 32 \cdot y_1 \oplus 133 \cdot y_2 \oplus 16 \cdot y_3 \oplus 194 \cdot y_4 \oplus 192 \cdot y_5 \oplus \\ & \oplus y_6 \oplus 251 \cdot y_7 \oplus y_8 \oplus 192 \cdot y_9 \oplus 194 \cdot y_{10} \oplus 16 \cdot y_{11} \oplus \\ & \oplus 133 \cdot y_{12} \oplus 32 \cdot y_{13} \oplus 148 \cdot y_{14} \oplus y_{15}, \end{aligned} \quad (8.12)$$

где операция  $\cdot$  обозначает умножение полиномов-байт по модулю  $m(x) = x^8 + x^7 + x^6 + x + 1$  (для выполнения такого умножения можно использовать алгоритм 8.6). Линейное преобразование блока, обозначаемое  $L(Y)$ , записывается в виде следующего алгоритма:

ВХОД: Блок  $Y = (y_0, y_1, \dots, y_{15})$ .

ВЫХОД:  $L(Y)$ .

DO 16 раз

$$b \leftarrow \text{lm}(Y);$$

$$Y \leftarrow Y \leftarrow 1;$$

$$y_0 \leftarrow b.$$

RETURN  $Y$ .

Операция  $\leftarrow$  означает циклический сдвиг блока вправо на 1 байт.

Раундовый ключ  $W = W_0, W_1, \dots, W_9$  состоит из 10 блоков по 128 бит и получается из секретного ключа  $K$  с помощью алгоритма, который будет описан позже.

$S$ -бокс для обратного преобразования  $S^{-1}(X)$  представляет собой перестановку байт, обратную по отношению к  $S$ :

$$S^{-1}[S[b]] = b, \quad b = 0, 1, \dots, 255.$$

Обратное линейное преобразование  $L^{-1}(X)$  выполняется с помощью следующего алгоритма:

ВХОД: Блок  $X = (x_0, x_1, \dots, x_{15})$ .

ВЫХОД:  $L^{-1}(X)$ .

DO 16 раз

$X \leftarrow X \leftarrow 1$ ;

$b \leftarrow \text{lm}(X)$ ;

$x_{15} \leftarrow b$ .

RETURN  $X$ .

Операция  $\leftarrow$  означает циклический сдвиг блока влево на 1 байт.

### Алгоритм 8.11. Кузнецик: шифрование блока

ВХОД: Блок  $X$ , раундовый ключ  $W$ .

ВЫХОД: Блок  $Y$ .

1.  $Y \leftarrow X \oplus W_0$ ;
2. FOR  $i = 1, 2, \dots, 9$  DO
3.      $Y \leftarrow S(Y)$ ,
4.      $Y \leftarrow L(Y)$ ,
5.      $Y \leftarrow Y \oplus W_i$ .
6. RETURN  $Y$ .

Для расшифрования блока все операции необходимо выполнить в обратном порядке с использованием обратных отображений.

### Алгоритм 8.12. Кузнецик: расшифрование блока

ВХОД: Блок  $Y$ , раундовый ключ  $W$ .

ВЫХОД: Блок  $X$ .

1.  $X \leftarrow Y \oplus W_9$ ;
2. FOR  $i = 8, 7, \dots, 0$  DO
3.      $X \leftarrow L^{-1}(X)$ ,
4.      $X \leftarrow S^{-1}(X)$ ,
5.      $X \leftarrow X \oplus W_i$ .
6. RETURN  $X$ .

Алгоритм формирования раундового ключа строится на базе тех же преобразований, которые используются при шифровании блока. Отдельные элементы секретного и раундового ключей — это слова, равные по размеру блоку шифра (128 бит).

**Алгоритм 8.13. КУЗНЕЧИК: ФОРМИРОВАНИЕ РАУНДОВОГО КЛЮЧА**

ВХОД: Секретный ключ  $K = K_0, K_1$ .

ВЫХОД: Раундовый ключ  $W = W_0, W_1, \dots, W_9$ .

1. FOR  $i = 0, 2, 4, 6$  DO
2.      $W_i \leftarrow K_0, W_{i+1} \leftarrow K_1$ ;
3.     FOR  $j = 1, 2, \dots, 8$  DO
4.          $X \leftarrow K_0 \oplus L(4i + j)$ ,
5.          $X \leftarrow S(X)$ ,
6.          $X \leftarrow L(X)$ ,
7.          $X \leftarrow X \oplus K_1$ ,
8.          $K_1 \leftarrow K_0, K_0 \leftarrow X$ ;
9.      $W_8 \leftarrow K_0, W_9 \leftarrow K_1$ .
10.    RETURN  $W$ .

Мы видим, что практически все операции в шифре Кузнечик требуют работы с отдельными байтами. Это довольно неэффективно для современных компьютеров и даже микроконтроллеров, в которых длина машинного слова составляет 32 или 64 бита. К счастью, шифр Кузнечик, подобно шифру Rijndael, допускает быструю реализацию на 32- и 64-битных платформах за счет предварительно вычисленных таблиц, хранимых в памяти. Метод табличного ускорения шифра Кузнечик был предложен в [41]. Перейдем к его описанию.

Так как преобразование  $L(Y)$  линейно, то существует матрица  $A$  такая, что

$$L(Y) = YA = (y_0, y_1, \dots, y_{15}) \cdot \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,15} \\ a_{1,0} & a_{1,1} & \dots & a_{1,15} \\ & & \dots & \\ a_{15,0} & a_{15,1} & \dots & a_{15,15} \end{pmatrix}. \quad (8.13)$$

В соответствии с алгоритмом реализации преобразования  $L$  при умножении вектора на матрицу байты рассматриваются как полиномы и операции умножения и сложения выполняются по правилам полиномиальной арифметики с приведением результата умножения по модулю  $m(x) = x^8 + x^7 + x^6 + x + 1$ . Матрица  $A$  может быть легко вычислена путем применения преобразования  $L$  к блокам, содержа-

щим один единичный байт. Действительно,

$$\begin{aligned} L(1, 0, 0, \dots, 0) &= (a_{0,0}, a_{0,1}, \dots, a_{0,15}), \\ L(0, 1, 0, \dots, 0) &= (a_{1,0}, a_{1,1}, \dots, a_{1,15}), \\ &\dots \\ L(0, 0, \dots, 0, 1) &= (a_{15,0}, a_{15,1}, \dots, a_{15,15}). \end{aligned}$$

Перегруппировав слагаемые в произведении вектора и матрицы (8.13) мы можем записать

$$L(Y) = L_0(y_0) \oplus L_1(y_1) \oplus \dots \oplus L_{15}(y_{15}),$$

где

$$L_i(b) = (a_{i,0} \cdot b, a_{i,1} \cdot b, \dots, a_{i,15} \cdot b).$$

Мы можем заранее вычислить значения векторов  $L_i$  для всех возможных байт  $b$  от 0 до 255.

Теперь обратим внимание на то, что при шифровании блока преобразованию  $L$  предшествует преобразование  $S$ , которое заменяет каждый байт блока на байт из таблицы замены  $S$ . В результате мы можем скомбинировать последовательность преобразований  $S$  и  $L$  и заранее вычислить 16 таблиц  $T_0, \dots, T_{15}$  по правилу

$$T_i[b] = L_i(S(b)),$$

т.е.

$$\begin{aligned} T_i[0] &= (a_{i,0} \cdot S[0], a_{i,1} \cdot S[0], \dots, a_{i,15} \cdot S[0]), \\ &\dots \\ T_i[255] &= (a_{i,0} \cdot S[255], a_{i,1} \cdot S[255], \dots, a_{i,15} \cdot S[255]). \end{aligned}$$

Табличная версия алгоритма шифрования приведена ниже.

Отметим, что каждый элемент таблиц  $T$  имеет длину равную размеру блока (128 бит). Поэтому табличная реализация шифра может полностью использовать возможности современных процессоров выбирать из памяти и выполнять операции со словами длиной 32, 64 или даже 128 бит.

Точно так же последовательность преобразований  $S$  и  $L$  может быть оптимизирована в алгоритме формирования раундового ключа. Константы  $L(1), L(2), \dots, L(32)$ , необходимые для выполнения

четвертой строки алгоритма, также могут быть вычислены заранее и храниться в памяти.

**Алгоритм 8.14. КУЗНЕЧИК: ШИФРОВАНИЕ БЛОКА  
ТАБЛИЧНЫЙ ВАРИАНТ**

**ВХОД:** Блок  $X$ , раундовый ключ  $W$ .

**ВЫХОД:** Блок  $Y$ .

1.  $Y \leftarrow X \oplus W_0$ ;
2. FOR  $i = 1, 2, \dots, 9$  DO
3.      $Y \leftarrow T_0[y_0] \oplus T_1[y_1] \oplus \dots \oplus T_{15}[y_{15}]$ ,
4.      $Y \leftarrow Y \oplus W_i$ .
5. RETURN  $Y$ .

Теперь рассмотрим возможность табличной реализации алгоритма расшифрования. Здесь применимы те же рассуждения, которые привели нас к построению таблиц для алгоритма зашифрования. Однако возникает проблема, состоящая в том, что преобразование  $S^{-1}$  выполняется не перед, а после преобразования  $L^{-1}$ , и мы не сможем их скомбинировать, как мы это сделали в алгоритме зашифрования. Чтобы получить возможность скомбинировать эти два преобразования, нам понадобится немного изменить структуру обратного шифра.

Запишем для наглядности последовательность действий, выполняемых при расшифровании блока, раскрыв цикл в алгоритме 8.12:

$$W_9 \quad L^{-1}S^{-1}W_8 \quad L^{-1}S^{-1}W_7 \quad \dots \quad L^{-1}S^{-1}W_0.$$

Действия выполняются слева направо,  $W_i$  обозначает сложение с соответствующим раундовым ключом, с помощью пробела условно показана группировка действий в алгоритме 8.12. Теперь добавим в начало этой цепочки два взаимно компенсирующих преобразования  $S$  и  $S^{-1}$  и по-другому сгруппируем действия:

$$S \quad S^{-1}W_9L^{-1} \quad S^{-1}W_8L^{-1} \quad S^{-1}W_7 \dots \quad L^{-1} \quad S^{-1}W_0.$$

Рассмотрим тройку  $S^{-1}W_iL^{-1}$ . Эта последовательность действий для текущего состояния блока  $X$  вычисляет  $L^{-1}(S^{-1}(X) \oplus W_i)$ . Вследствие линейности преобразования  $L^{-1}$  справедливо следующее равенство [41]:

$$L^{-1}(S^{-1}(X) \oplus W_i) = L^{-1}(S^{-1}(X)) \oplus L^{-1}(W_i).$$

В результате мы получили связку, в которой преобразование  $S^{-1}$  предшествует преобразованию  $L^{-1}$ . Теперь по аналогии с тем, как мы это делали для алгоритма зашифрования, мы можем вычислить 16 таблиц для быстрого выполнения последовательности преобразований  $S^{-1} L^{-1}$  при расшифровании. Обозначим эти таблицы через  $T_0^{-1}, \dots, T_{15}^{-1}$ . Мы готовы записать алгоритм расшифрования в табличной версии.

**Алгоритм 8.15. Кузнецник: расшифрование блока  
табличный вариант**

**ВХОД:** Блок  $Y$ , раундовый ключ  $W$ .

**ВЫХОД:** Блок  $X$ .

1.  $X \leftarrow S(Y);$
2. FOR  $i = 9, 8, \dots, 1$  DO
3.      $X \leftarrow T_0^{-1}[x_0] \oplus T_1^{-1}[x_1] \oplus \dots \oplus T_{15}^{-1}[x_{15}],$
4.      $X \leftarrow X \oplus L^{-1}(W_i);$
5.      $X \leftarrow S^{-1}(X);$
6.      $X \leftarrow X \oplus W_0.$
7. RETURN  $X.$

Мы видим, что по сравнению с табличным алгоритмом зашифрования здесь добавляются два дополнительных нелинейных преобразования. Преобразованные блоки раундового ключа  $L^{-1}(W_1), \dots, L^{-1}(W_9)$  вычисляются заранее (при формировании раундового ключа). Блок  $W_0$  остается в не преобразованном виде.

**8.3. Основные режимы функционирования  
блоковых шифров**

Блоковые шифры применяются для решения многих задач криптографии. В этом разделе мы рассмотрим основные режимы их использования.

В предыдущем разделе были даны примеры реальных блоковых шифров. Теперь мы можем думать о (идеализированном) блоковом шифре, как о преобразовании входного блока  $X$  в выходной блок  $Y$  с участием секретного ключа  $K$ ,

$$Y = E_K(X),$$

причем это преобразование должно иметь следующие свойства:

- 1) при известном  $Y$ , но неизвестном  $K$  практически невозможно узнать  $X$ ;
- 2) при произвольных известных  $X$  и  $Y$ , но неизвестном  $K$  практически невозможно узнать  $K$ .

Вначале рассмотрим классическую задачу шифрования сообщений при помощи блоковых шифров.

### Режим ECB

Название режима ECB (Electronic CodeBook) можно перевести как электронная кодовая книга.

Сообщение  $X$  разбивается на блоки  $X = X_1, X_2, \dots, X_t$ . Каждый блок шифруется блоковым шифром

$$Y_i = E_K(X_i), \quad 1 \leq i \leq t.$$

Получаем зашифрованное сообщение  $Y = Y_1, Y_2, \dots, Y_t$ . Дешифрование выполняется по правилу

$$X_i = E_K^{-1}(Y_i), \quad 1 \leq i \leq t.$$

Нетрудно видеть, что дешифрование сообщения можно производить, выбирая блоки шифротекста в произвольном порядке. Такой режим удобен во многих реальных ситуациях. Например, можно работать с базой данных, хранящейся в зашифрованном виде. Однако при таком использовании одинаковые записи будут зашифрованы одинаково. Говорят, что в режиме ECB шифр сохраняет «образ данных», т.е. некий «рисунок» или шаблон, характерный для данных. Это может дать некоторую информацию противнику. Например, если количество различных записей в базе данных невелико (что нередко случается), то противник может составить словарь шифротекстов и вскрыть базу на основе частотного анализа. Заметим, что ему в этом случае не понадобится вскрывать сам шифр.

Некоторые авторы рекомендуют использовать режим ECB только в случаях, когда размер отдельного элемента данных в сообщении, к которому требуется осуществлять непосредственный (прямой) доступ, меньше размера блока. Остаток блока, свободный от данных,

рекомендуется заполнять случайно выбираемыми битами. Тогда даже одинаковые элементы данных будут иметь разные шифротексты. При дешифровании биты заполнения просто отбрасываются.

Если размер элемента данных превышает размер блока, то часто рекомендуют использовать режим CBC.

## Режим CBC

Название режима CBC (Cipher-Block Chaining) переводится как сцепление блоков шифра.

Зашифрованное сообщение получается по следующему правилу:

$$Y_i = E_K(X_i \oplus Y_{i-1}), \quad 1 \leq i \leq t,$$

т.е. каждый последующий блок открытого текста предварительно закрывается предыдущим зашифрованным блоком. Слово  $Y_0$  должно быть определено заранее и известно при шифровании и дешифровании. Полученное зашифрованное сообщение можно дешифровать следующим образом:

$$X_i = Y_{i-1} \oplus E_K^{-1}(Y_i), \quad 1 \leq i \leq t.$$

Мы получаем шифротекст, в котором каждый следующий блок зависит от предыдущих. Данный режим разрушает «образ данных». Даже если все блоки  $X$  идентичны, шифротекст будет состоять из различных блоков  $Y$ . Этот режим предпочтителен при шифровании сообщений, размер которых превышает размер блока. Однако дешифровать сообщение можно только последовательно, начиная с первого блока.

## 8.4. Потоковые шифры

В главе 7 мы рассмотрели шифр Вернама и установили, что он является совершенно секретным, т.е. при его использовании противник, перехвативший зашифрованное сообщение, не получает никакой информации об исходном сообщении. В шифре Вернама зашифрованное сообщение  $y_1, y_2, \dots, y_k$  получается из исходного сообщения  $x_1, x_2, \dots, x_k$  и ключа  $z_1, z_2, \dots, z_k$  при помощи операции шифрования, задаваемой равенством

$$y_i = x_i \oplus z_i, \quad i = 1, 2, \dots, k. \quad (8.14)$$

Мы видели, что данный шифр совершенен только в том случае, если ключ  $z_1, z_2, \dots, z_k$  образован из независимых и равновероятных символов и используется только один раз. Это приводит к необходимости генерировать случайные последовательности очень большого объема и передавать их по закрытым каналам связи, что весьма затруднительно. Поэтому давно была высказана идея использовать в качестве ключа последовательности не случайные, а порожденные при помощи генераторов псевдослучайных чисел. В этом случае в качестве секретного ключа используется начальное значение или состояние генератора. Однако надо четко осознавать, что в этом случае система уже не будет совершенно секретной. Максимум на что мы можем надеяться, это то, что для раскрытия этой системы потребуется очень много времени (например, нужно будет выполнить перебор всех возможных начальных состояний генератора). В качестве возмещения за потерю совершенности мы получаем возможность использовать короткие (обычно несколько сотен бит) секретные ключи, которые значительно проще распределять и хранить (а при использовании систем с открытым ключом их можно и вычислять).

**Определение 8.1.** Шифр, построенный на основе (8.14), где в качестве  $z_1, z_2, \dots, z_k$  используется псевдослучайная последовательность, называется *потоковым (или поточным) шифром (stream cipher)*.

Как правило, исходное сообщение и ключевая последовательность представляют собой независимые потоки бит. Так как шифрующее (и дешифрующее) преобразование для всех потоковых шифров одно и то же, они различаются только способом построения генераторов псевдослучайных чисел. Действительно, чтобы из шифротекста  $y_1, y_2, \dots, y_k$ , полученного по формуле (8.14), восстановить сообщение  $x_1, x_2, \dots, x_k$ , необходимо сгенерировать точно такую же последовательность  $z_1, z_2, \dots, z_k$ , что и при шифровании, и использовать для дешифрования формулу

$$x_i = y_i \oplus z_i, \quad i = 1, 2, \dots, k. \quad (8.15)$$

**Пример 8.1.** Один из самых простых генераторов псевдослучайных чисел (линейный конгруэнтный генератор) работает по схеме

$$z_{i+1} = (az_i + b) \bmod c, \quad (8.16)$$

где  $a$ ,  $b$ ,  $c$  — некоторые константы, а  $z_{i+1}$  — очередное псевдослучайное число, вычисляемое по предыдущему  $z_i$ . Обязательно задается начальное значение  $z_0$ . Возьмем в качестве примера  $a = 5$ ,  $b = 12$ ,  $c = 23$ , и пусть  $z_0 = 4$ . Вычислим несколько элементов последовательности:

$$\begin{aligned}z_1 &= (4 \cdot 5 + 12) \bmod 23 = 9, \\z_2 &= (9 \cdot 5 + 12) \bmod 23 = 11, \\z_3 &= (11 \cdot 5 + 12) \bmod 23 = 21, \\z_4 &= (21 \cdot 5 + 12) \bmod 23 = 2, \\z_5 &= (2 \cdot 5 + 12) \bmod 23 = 22, \\z_6 &= (22 \cdot 5 + 12) \bmod 23 = 7, \\z_7 &= (7 \cdot 5 + 12) \bmod 23 = 1.\end{aligned}$$

Полученная последовательность внешне выглядит как довольно случайная.  $\square$

Для использования в криптографических целях генератор должен удовлетворять следующим основным требованиям:

- 1) период последовательности должен быть очень большой;
- 2) порождаемая последовательность должна быть «почти» неотличима от действительно случайной, в частности, вычисление числа  $z_{i+1}$  по известным предыдущим элементам последовательности без знания ключа должно быть трудной, практически нерешаемой задачей.

Рассмотренный выше линейный конгруэнтный генератор совершенно не годится для криптографических целей, т.к. известны простые алгоритмы, позволяющие полностью восстановить параметры генератора всего по нескольким элементам порождаемой им последовательности.

В качестве примеров криптостойких генераторов псевдослучайных чисел мы рассмотрим режимы OFB и CTR блоковых шифров и алгоритм RC4.

### Режим OFB блокового шифра

Название режима OFB (Output FeedBack) переводится как обратная связь по выходу. В этом режиме блоковый шифр на основе секретного ключа  $K$  и некоторого инициализирующего вектора  $Y_0$

формирует псевдослучайную последовательность  $r$ -битовых чисел  $z_1, z_2, \dots, z_k$ , которая может использоваться в (8.14) и (8.15) соответственно для шифрования и дешифрования сообщения. Будем считать, как и ранее, что размер блока шифра равен  $n$  бит.

Псевдослучайная последовательность получается по схеме

$$Y_i = E_K(Y_{i-1}),$$

$$z_i = r \text{ старших бит } Y_i, \quad 1 \leq i \leq k$$

(здесь  $r$ ,  $1 \leq r \leq n$ , — параметр метода).

При использовании стойкого блокового шифра можно получить криптостойкий генератор, отвечающий приведенным выше требованиям. А именно, средняя длина периода псевдослучайной последовательности (при случайно выбранных  $K$  и  $Y_0$ ) составляет примерно  $r2^{n-1}$  бит [68]. Кроме того, псевдослучайная последовательность «непредсказуема» для противника, т.к. возможность предсказания (вычисления)  $z_{i+1}$  на основе  $z_1, \dots, z_i$  означала бы нестойкость шифра по отношению к атаке по известному тексту. Предсказание  $z_{i+1}$  становится даже более трудной задачей, чем взлом блокового шифра, если  $r < n$  [68].

Обратим внимание на одну особенность, характерную для всех потоковых шифров. Для шифрования каждого отдельного сообщения необходимо использовать разные  $K$  и/или  $Y_0$ . В противном случае несколько сообщений будут шифроваться с помощью одних и тех же последовательностей  $z$ , и такой шифр может быть раскрыт. Поясним суть проблемы. Пусть два сообщения  $u_1, u_2, \dots, u_k$  и  $v_1, v_2, \dots, v_k$  зашифрованы с помощью одной и той же последовательности  $z$ . Тогда шифротексты будут иметь вид

$$u_1 \oplus z_1, u_2 \oplus z_2, \dots, u_k \oplus z_k \quad \text{и}$$

$$v_1 \oplus z_1, v_2 \oplus z_2, \dots, v_k \oplus z_k.$$

Сложим оба шифротекста и, с учетом равенства  $z_i \oplus z_i = 0$ , получим последовательность

$$u_1 \oplus v_1, u_2 \oplus v_2, \dots, u_k \oplus v_k.$$

Мы получили аналог так называемого «шифра с бегущим ключом», когда один текст шифруется с помощью другого текста, взятого из определенного места определенной книги. Известно, что такой шифр

не стоек, хотя использовался в эпоху «донаучной» криптографии [68]. Статистический анализ, основанный на избыточности текстов, позволяет в большинстве случаев достаточно точно восстановить оба сообщения (мы подробно рассматривали эту проблему в разд. 7.6).

Расшифрование сообщений для описанного режима блокового шифра может производиться только с начала, т.к. невозможно получить произвольный элемент последовательности  $z$ , не вычислив предыдущие. В этом смысле режим аналогичен режиму CBC. Преимущество режима OFB заключается в том, что последовательность  $z$  может быть сформирована заранее для того, чтобы быстро шифровать или дешифровать сообщения с помощью (8.14), (8.15) в момент их поступления. Это может быть актуально для систем, обрабатывающих данные в реальном масштабе времени.

### Режим CTR блокового шифра

Название данного режима происходит от слова CounTeR — счетчик. Этот режим очень похож на OFB, но в нем шифруется не предыдущий выход шифра, а просто счетчик, увеличиваемый на каждом шаге на постоянное число (обычно 1). Точнее, схема выглядит следующим образом:

$$z_i = r \text{ старших бит } E_K(Y_0 + i), \quad i = 1, 2, 3, \dots,$$

где  $r$  — параметр.

При использовании «идеального» блокового шифра режим CTR обеспечивает те же параметры стойкости, что и OFB. Преимущество режима CTR состоит в том, что любой элемент последовательности  $z$  может быть вычислен непосредственно. Это дает возможность шифровать и дешифровать любые фрагменты сообщения независимо друг от друга.

### Алгоритм RC4

Алгоритм RC4, предложенный Ривестом в 1994 году, относится к классу алгоритмов, разработанных специально для потоковых шифров. Генераторы псевдослучайных чисел, построенные с помощью таких алгоритмов, как правило, значительно быстрее генераторов, основанных на блоковых шифрах.

Алгоритм RC4 работает с  $n$ -битовыми словами. Все вычисления проводятся по модулю  $2^n$  (остаток  $x \bmod 2^n$  вычисляется очень быстро путем выделения  $n$  младших бит в  $x$  с помощью логической операции «и»). RC4 использует  $L$ -словный ключ  $K = K_0 K_1 \dots K_{L-1}$  и генерирует последовательность слов  $\bar{z} = z_1 z_2 z_3 \dots$ , конкретный вид которой определяется ключом  $K$ . Состояние генератора задается таблицей  $S$  из  $2^n$  слов и двух переменных  $i$  и  $j$ . В каждый момент времени таблица  $S$  содержит все возможные  $n$ -битовые числа в перемешанном виде. Так как каждый элемент таблицы принимает значения в промежутке  $[0, 2^n - 1]$ , то его можно трактовать двояко: либо как число, либо как номер другого элемента в таблице. Секретный ключ задает только начальное перемешивание чисел в таблице, которое формируется с помощью следующего алгоритма:

```

 $j \leftarrow 0, \quad S \leftarrow (0, 1, \dots, 2^n - 1);$ 
FOR  $i = 0, 1, \dots, 2^n - 1$  DO
     $j \leftarrow (j + S_i + K_i \bmod L) \bmod 2^n,$ 
     $S_j \leftrightarrow S_i;$ 
     $i \leftarrow 0, \quad j \leftarrow 0.$ 

```

После этого генератор готов к работе. Генерация очередного псевдослучайного слова  $z_i$  осуществляется следующим образом:

```

 $i \leftarrow (i + 1) \bmod 2^n;$ 
 $j \leftarrow (j + S_i) \bmod 2^n;$ 
 $S_j \leftrightarrow S_i;$ 
 $t \leftarrow (S_i + S_j) \bmod 2^n;$ 
 $z_i \leftarrow S_t.$ 

```

Пример 8.2. Пусть  $n = 3$ ,  $K = 25$  ( $L = 2$ ).

Сформируем начальную перестановку чисел в таблице  $S$  (все вычисления проводим по модулю 8):

$$\begin{array}{ll}
 j = 0, & S = (0, 1, 2, 3, 4, 5, 6, 7), \\
 i = 0, \quad j = 0 + 0 + 2 = 2, & S = (2, 1, 0, 3, 4, 5, 6, 7), \\
 i = 1, \quad j = 2 + 1 + 5 = 0, & S = (1, 2, 0, 3, 4, 5, 6, 7), \\
 i = 2, \quad j = 0 + 0 + 2 = 2, & S = (1, 2, 0, 3, 4, 5, 6, 7), \\
 i = 3, \quad j = 2 + 3 + 5 = 2, & S = (1, 2, 3, 0, 4, 5, 6, 7), \\
 i = 4, \quad j = 2 + 4 + 2 = 0, & S = (4, 2, 3, 0, 1, 5, 6, 7), \\
 i = 5, \quad j = 0 + 5 + 5 = 2, & S = (4, 2, 5, 0, 1, 3, 6, 7), \\
 i = 6, \quad j = 2 + 6 + 2 = 2, & S = (4, 2, 6, 0, 1, 3, 5, 7), \\
 i = 7, \quad j = 2 + 7 + 5 = 6, & S = (4, 2, 6, 0, 1, 3, 7, 5).
 \end{array}$$

Теперь вычислим несколько первых элементов псевдослучайной последовательности  $\bar{z}$ :

$$\begin{aligned} i = 1, j = 0 + 2 = 2, S = (4, 6, 2, 0, 1, 3, 7, 5), t = 2 + 6 = 0, z_1 = 4, \\ i = 2, j = 2 + 2 = 4, S = (4, 6, 1, 0, 2, 3, 7, 5), t = 1 + 2 = 3, z_2 = 0, \\ i = 3, j = 4 + 0 = 4, S = (4, 6, 1, 2, 0, 3, 7, 5), t = 2 + 0 = 2, z_3 = 1, \\ i = 4, j = 4 + 0 = 4, S = (4, 6, 1, 2, 0, 3, 7, 5), t = 0 + 0 = 0, z_4 = 4, \\ i = 5, j = 4 + 3 = 7, S = (4, 6, 1, 2, 0, 5, 7, 3), t = 5 + 3 = 0, z_5 = 4, \\ i = 6, j = 7 + 7 = 6, S = (4, 6, 1, 2, 0, 5, 7, 3), t = 7 + 7 = 6, z_6 = 7 \end{aligned}$$

и т.д. Чтобы воспользоваться формулой (8.14) для получения шифра, числа  $z_i$  записываем в двоичном виде. В рассмотренном примере каждое число  $z_i$  представляется тремя битами, и мы получаем последовательность

$$\bar{z} = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1 \dots$$

□

## Алгоритм HC-128

В области потоковых шифров не наблюдается такой стандартизации методов, которую мы видим во многих других направлениях криптографии. Однако, необходимо упомянуть о попытке выработки европейского стандарта, для чего в 2004 году был организован открытый конкурс проектов eSTREAM [53]. Конкурс завершился в 2008 году и его победителями стали 7 потоковых шифров (4 для программной реализации и 3 для аппаратной). Все победители были рекомендованы к широкому применению. Одним из победителей стал шифр HC-128, который мы подробно опишем в данном подразделе.

Автором HC-128 [97] является Хунцзюнь У (Hongjun Wu), который в момент создания шифра работал в Католическом университете Лёвена (Бельгия).

Шифр HC-128 предполагает 128-битный секретный ключ  $K$  и 128-битный инициализирующий вектор  $IV$ .

Конструкция HC-128 напоминает RC4. В нем имеются две таблицы по 512 32-битовых слов, играющие роль S-боксов, которые обозначаются через  $P$  и  $Q$ . Элементы таблиц индексируются с 0 по 511. На каждом шаге одно слово в таблицах обновляется в результате вычисления нелинейной функции так, что все слова обеих таблиц

обновляются через каждые 1024 шага. На каждом шаге генерируется 32 бита псевдослучайной последовательности  $z_i$ , где  $i$  — номер шага.

В HC-128 используются следующие операции:

- + — сложение слов по модулю  $2^{32}$  (т.е. обычное сложение 32-битных целых чисел с потерей переноса);
- — (при вычислении индексов таблиц) вычитание по модулю 512 (обычное вычитание целых чисел с выделением 9 младших бит результата с помощью побитовой логической операции «и»);
- $\oplus$  — побитовое «исключающее или» двух слов, т.е. побитовое сложение по модулю 2;
- $\gg$  — сдвиг слова вправо на указанное число бит;
- $\ll$  — сдвиг слова влево на указанное число бит;
- $\rightarrow$  — циклический сдвиг слова вправо на указанное число бит;
- $\leftarrow$  — циклический сдвиг слова влево на указанное число бит.

Слово  $x$  состоит из 4 байт:  $x = (x_3, x_2, x_1, x_0)$ , причем  $x_0$  — наименее значащий байт. Над словами определены 6 функций:

$$\begin{aligned} f_1(x) &= (x \leftarrow 7) \oplus (x \leftarrow 18) \oplus (x \gg 3), \\ f_2(x) &= (x \leftarrow 17) \oplus (x \leftarrow 19) \oplus (x \gg 10), \\ g_1(x, y, z) &= ((x \leftarrow 10) \oplus (z \leftarrow 23)) + (y \leftarrow 8), \\ g_2(x, y, z) &= ((x \leftarrow 10) \oplus (z \leftarrow 23)) + (y \leftarrow 8), \\ h_1(x) &= Q[x_0] + Q[256 + x_2], \\ h_2(x) &= P[x_0] + P[256 + x_2]. \end{aligned}$$

Функции  $f$  заимствованы из алгоритма SHA-256. Нелинейность функций обусловлена сочетанием операций из различных алгебраических групп (+ и  $\oplus$ , обычные (арифметические) и циклические сдвиги).

Как и в RC4, в HC-128 присутствует фаза инициализации, в которой создается начальное заполнение таблиц  $P$  и  $Q$ , определяемое секретным ключом и инициализирующим вектором. Для выполнения инициализации вводится вспомогательный массив  $W$  размером 1280 слов.

**Алгоритм 8.16. HC-128: ИНИЦИАЛИЗАЦИЯ**

**ВХОД:** Секретный ключ  $K = K_0, K_1, K_2, K_3$ ,  
инициализирующий вектор  $IV = IV_0, IV_1, IV_2, IV_3$ .

**ВЫХОД:** Сформированные таблицы  $P, Q$ .

1.  $W_0, \dots, W_7 \leftarrow K_0, \dots, K_3, K_0, \dots, K_3$ ;
2.  $W_8, \dots, W_{15} \leftarrow IV_0, \dots, IV_3, IV_0, \dots, IV_3$ ;
3. FOR  $i = 16, 17, \dots, 1279$  DO
  4.  $W_i \leftarrow f_2(W_{i-2}) + W_{i-7} + f_1(W_{i-15}) + W_{i-16} + i$ ;
  5. FOR  $i = 0, 1, \dots, 511$  DO
    6.  $P[i] \leftarrow W_{256+i}$ ,  $Q[i] \leftarrow W_{768+i}$ ;
    7. FOR  $i = 0, 1, \dots, 511$  DO
      8.  $P[i] \leftarrow (P[i] + g_1(P[i-3], P[i-10], P[i-511])) \oplus h_1(P[i-12])$ ;
    9. FOR  $i = 0, 1, \dots, 511$  DO
      10.  $Q[i] \leftarrow (Q[i] + g_1(Q[i-3], Q[i-10], Q[i-511])) \oplus h_2(Q[i-12])$ ;
  11. RETURN  $P, Q$ .

Теперь мы готовы генерировать псевдослучайную последовательность  $z_0, z_1, z_2, \dots$ . Каждый очередной элемент последовательности  $z_i$  (32 бита) генерируется в помощь следующего простого алгоритма:

**Алгоритм 8.17. HC-128: ГЕНЕРАЦИЯ ОЧЕРЕДНОГО СЛОВА**

**ВХОД:** Номер слова  $i$  ( $i = 0, 1, 2, \dots$ ).

**ВЫХОД:** Слово  $z_i$ .

1.  $j \leftarrow i \bmod 512$ .
2. IF  $(i \bmod 1024) < 512$  THEN
  3.  $P[j] \leftarrow P[j] + g_1(P[j-3], P[j-10], P[j-511])$ ,
  4.  $z_i \leftarrow h_1(P[j-12]) \oplus P[j]$ ;
5. ELSE
  6.  $Q[j] \leftarrow Q[j] + g_2(Q[j-3], Q[j-10], Q[j-511])$ ,
  7.  $z_i \leftarrow h_2(Q[j-12]) \oplus Q[j]$ ;
8. RETURN  $z_i$ .

На каждом шаге генерируется 32 бита псевдослучайной последовательности и обновляется один элемент в таблице. Причем на

первых 512 шагах обновляется таблица  $P$ , в то время как  $Q$  (через функцию  $h_1$ ) играет роль S-бокса. На следующих 512 шагах роли таблиц меняются: обновляется  $Q$ , а  $P$  используется как S-бокс (через функцию  $h_2$ ).

Согласно исследованиям, проведенным автором НС-128, для каждой пары  $K$  и  $IV$  могут быть сгенерированы  $2^{64}$  бит, прежде чем можно будет обнаружить отличие последовательности от полностью случайной; отсутствуют слабые ключи, поиск секретного ключа возможен только путем полного перебора. При тщательно выполненной программной реализации скорость генерации последовательности  $z$  составляет 3.05 цикла на байт (на процессоре Pentium M). Это примерно в 5–6 раз выше, чем скорость шифра AES на том же процессоре.

## 8.5. Криптографические хеш-функции

Мы уже встречались с понятием хеш-функции (hash function, в русскоязычной литературе иногда используется написание «хэш») в главе 4 при рассмотрении методов генерации электронной подписи. Там хеш-функции использовались в качестве «представителей» подписываемых сообщений, т.е. подпись реально вычислялась только для значения хеш-функции, но предполагалось, что это значение существенным образом зависит от всех символов сообщения и никто не может изменить сообщение так, чтобы это значение сохранилось.

На самом деле в современной криптографии хеш-функции применяются значительно более широко. Например, задачи надежного хранения паролей и реализации протокола «свой–чужой», рассмотренные в разд. 2.1, могут быть эффективно решены с помощью хеш-функций. В главе 9 будет рассмотрено применение хеш-функций для обеспечения уникальной идентификации и обеспечения целостности цифровых данных, а также для решения задачи «доказательства выполнения работы». С помощью хеш-функций можно строить криптоустойчивые генераторы псевдослучайных чисел и решать еще много задач.

В этом разделе мы более точно сформулируем основные требования к криптографически стойким хеш-функциям и дадим обзор основных методов их вычисления.

**Определение 8.2.** Хеш-функцией называется любая функция

$$y = h(x_1 x_2 \dots x_n),$$

которая строке (сообщению)  $x_1 x_2 \dots x_n$  произвольной длины  $n$  ставит в соответствие целое число фиксированной длины.

Примером хеш-функции может служить контрольная сумма для сообщения. В этом случае

$$h(x_1 x_2 \dots x_n) = (x_1 + x_2 + \dots + x_n) \bmod 2^w,$$

где  $w$  — размер машинного слова. Длина слова, получаемого как значение этой хеш-функции, составляет  $w$  бит независимо от длины сообщения. Контрольные суммы очень часто используются для обнаружения непреднамеренных ошибок в сообщении (при изменении одного символа контрольная сумма изменится). Однако очень легко внести преднамеренную ошибку в сообщение и сохранить при этом значение контрольной суммы. Если бы такая хеш-функция использовалась, например, при генерации электронной подписи, то было бы очень легко изменить содержание подписанного сообщения. Поэтому рассмотренная хеш-функция не годится для криптографических применений.

Сформулируем основные требования, предъявляемые к криптографическим хеш-функциям. Пусть  $x$  — некоторая строка (сообщение). Тогда

- 1) для любого заданного  $x$  вычисление  $h(x)$  должно выполняться относительно быстро;
- 2) при известном  $y$  должно быть трудно (практически невозмож-но) найти  $x$ , для которого  $y = h(x)$ ;
- 3) при известном сообщении  $x$  должно быть трудно найти другое сообщение  $x' \neq x$ , такое, что  $h(x') = h(x)$ ;
- 4) должно быть трудно найти какую-либо пару различных сообщений  $x$  и  $x'$ , для которых  $h(x') = h(x)$ .

Отметим, что первое требование должно выполняться всегда, в противном случае хеш-функция теряет какое-либо практическое значение. Остальные требования важны для тех или иных приложений. Например, если пароли для входа в систему хранятся в виде

значений соответствующих им хеш-функций, то хеш-функция должна удовлетворять второму требованию. В схеме электронной подписи актуально третье требование. Четвертое требование важно в некоторых криптографических протоколах. Заметим, что четвертое требование более сильное, чем третье (т.е. при выполнении четвертого автоматически выполняется и третье). Часто говорят, что второе свойство означает, что хеш-функция является односторонней. Четвертое свойство означает, что хеш-функция устойчива к коллизиям (коллизией называют ситуацию, когда есть два различных аргумента, дающие одно и то же значение хеш-функции).

Значение хеш-функции часто называют просто «хеш-значением» (hash value) или «дайджестом сообщения» (message digest). Для криптографических применений важна длина хеш-значения, обычно выражаемая в битах. Обозначим эту длину через  $t$ . Тогда выполнение свойств 2 и 3 означает, что наилучший алгоритм поиска аргумента  $x$  требует  $O(2^t)$  операций, а выполнение свойства 4 означает, что самый быстрый алгоритм, находящий произвольную пары аргументов с одинаковым значением хеш-функции, требует  $O(2^{t/2})$  операций (вследствие парадокса дней рождений). Сегодня принято считать, что при современном и в перспективе на ближайшие 100 лет состоянии вычислительной техники «вычислительно невозможное» число операций — это  $2^{80}$ . Поэтому, для удовлетворения четвертого требования, длина хеш-значения должна быть не меньше 160 бит.

Разработка хеш-функции, удовлетворяющей всем четырем требованиям — задача непростая. В настоящее время предложены и практически используются несколько базовых конструкций, на базе которых строятся многочисленные хеш-функции.

Первая, наиболее старая и хорошо изученная конструкция, носит имя Меркля–Дамгарда (Ralph Merkle, Ivan Damgård, 1979, см., например, [68]). Она базируется на так называемой «функции сжатия», в качестве которой может использоваться блоковый шифр или его аналог (желательно, чтобы функция сжатия не была обратимой). Рассмотрим две реализации этого подхода с применением блокового шифра. Пусть дан блоковый шифр  $E$ , который для заданного блока  $X$  и ключа  $K$  формирует зашифрованный блок  $Y$ ,

$$Y = E_K(X).$$

Мы представим два алгоритма, для которых длина слова, получаемого как значение хеш-функции, равна размеру блока в шифре,

но отметим, что известны конструкции, позволяющие получать хеш-функции с длинами слов, кратными размеру блока.

В первом алгоритме сообщение вначале представляется в виде последовательности блоков  $X_1, X_2, \dots, X_n$ . В последний блок дописывается некоторая дополняющая последовательность бит. Стандартное дополнение строится как последовательность нулей, завершаемая одной единицей, за которой следует двоичное число фиксированного размера (например, 64 бита), равное длине сообщения. Количество нулей выбирается таким, чтобы общая длина дополненного сообщения была кратна размеру блока. Значение хеш-функции  $h$  получается как результат выполнения следующего итерационного процесса:

```

 $h \leftarrow 0;$ 
FOR  $i = 1, 2, \dots, n$  DO
     $h \leftarrow E_h(X_i) \oplus h.$ 

```

В качестве начального значения  $h$  можно использовать не ноль, а какое-либо «магическое» число, но это не имеет существенного значения. В данном алгоритме значение  $h$ , полученное на предыдущей итерации, используется в качестве ключа шифра в следующей итерации. Поэтому неявно полагается, что длина ключа в шифре равна длине блока. Прибавление к зашифрованному блоку исходного блока делает функцию сжатия необратимой.

Однако, как мы видели, во многих шифрах длина ключа превышает размер блока (например, в RC6 при максимальной длине блока 256 бит длина ключа может достигать 255 байт, или 2040 бит). В таких случаях более эффективен другой алгоритм.

В этом алгоритме сообщение вначале представляется в виде последовательности  $X_1, X_2, \dots, X_m$ , в которой размер каждого элемента равен длине ключа в шифре. Последний элемент заполняется так же, как и в первом алгоритме. Значение хеш-функции  $h$  вычисляется следующим образом:

```

 $h \leftarrow 0;$ 
FOR  $i = 1, 2, \dots, m$  DO
     $h \leftarrow E_{X_i}(h) \oplus h.$ 

```

Здесь уже элементы сообщения выполняют роль ключей в шифре.

Представленные алгоритмы вычисления хеш-функций удовлетворяют всем четырем требованиям, предъявляемым к криптографическим хеш-функциям, в предположении стойкости используемых блочных шифров (см. [61, 68]). Однако есть одна проблема: блочные шифры обычно исследуются в предположении, что ключ неизвестен. В представленной же конструкции известно все — и исходный текст, и зашифрованный текст, и ключ. Поэтому к блочному шифру предъявляются дополнительные требования, например, отсутствие слабых или связанных ключей (т.е. ключей, при которых можно получить почти одинаковые шифротексты). Сегодня мы имеем уже несколько дискредитированных хеш-функций, которые имели в основе данную конструкцию и активно использовались. Поэтому ведется поиск других конструкций для построения хеш-функций.

Одной из альтернативных конструкций является так называемая «губка» (sponge construction). Данная конструкция возникла в 2007 году и была впервые представлена в работе [36]. В основе конструкции лежит состояние (битовый массив)  $S$ , разделенное на две части  $S = (S^r, S^c)$  длиной соответственно  $r$  и  $c$  бит ( $r$  и  $c$  — параметры метода), и некоторое перемешивающее преобразование  $f$ , применяемое к всему состоянию в целом по схеме  $S \leftarrow f(S)$ . Сообщение представляется в виде последовательности  $r$ -битовых блоков  $X_1, X_2, \dots, X_k$ , последний блок дополняется обычным образом. Формирование хеш-значения происходит в два этапа: этап поглощения губкой блоков сообщения и этап выжимания губки с извлечением нужного числа бит результата. Поглощение описывается следующим алгоритмом:

```

 $S \leftarrow 0;$ 
FOR  $i = 1, 2, \dots, k$  DO
   $S^r \leftarrow S^r \oplus X_i,$ 
   $S \leftarrow f(S).$ 

```

Выжимание губки производится так:

```

FOR  $i = 1, 2, \dots, t/r$  DO
   $h_i \leftarrow S^r,$ 
   $S \leftarrow f(S).$ 

```

Результатом является значение хеш-функции  $h = h_1, h_2, \dots, h_{t/r}$ , построенное из отдельных блоков по  $r$  бит. Здесь мы неявно допустили,

что длина хеш-значения  $t$  кратна  $r$ , что обычно и бывает. Если это не так, то можно просто обрезать последний блок.

Обратим внимание на то, что часть состояния  $S^c$  никогда непосредственно не взаимодействует ни со входом, ни с выходом. Она участвует только в перемешивающем преобразовании  $f$ . Эта часть состояния определяет как бы внутреннюю емкость губки. Авторами конструкции показано, что для достижения криптографической стойкости хеш-функции на уровне  $t$  бит должно выполняться равенство  $c = 2t$ . Разумеется, стойкость также существенно зависит от выбора преобразования  $f$ . Теоретические свойства конструкции доказывались авторами в предположении, что  $f$  — полностью случайная перестановка. Но на практике эта перестановка, конечно, создается некоторыми алгоритмами, т.е. является псевдослучайной.

Приведем краткий обзор наиболее популярных хеш-функций, включив в него для полноты и те, которые уже вышли из употребления, но имели важное значение для развития этой области.

Первая линейка хеш-функций представлена алгоритмами MD2, MD4, MD5 и MD6. Аббревиатура MD здесь имеет двойкий смысл: с одной стороны это Message Digest (дайджест сообщения), а с другой — Меркль–Дамгард в честь соответствующей конструкции, лежащей в основе, которую мы описали выше. Данная линейка алгоритмов разрабатывалась под руководством уже известного нам Р. Ривеста. Первая хеш-функция, MD2, появилась в 1989 году и с тех пор активно использовалась в различных протоколах Интернета. Ряд найденных против нее теоретических атак, хотя и не всегда осуществимых на практике, постепенно привел к отказу от ее использования. Алгоритм MD4 не нашел широкого применения, во-первых, потому, что в нем практически сразу были найдены недостатки, а во-вторых, т.к. он был быстро заменен на более совершенный аналог MD5.

Хеш-функция MD5 с длиной хеш-значения 128 бит была предложена в 1991 году и с тех пор стала стандартом де-факто [75] для многих систем компьютерной безопасности. Она до сих пор используется как криптографическая контрольная сумма для проверки целостности файлов и для хранения паролей в компьютерах. Но с этой хеш-функцией связана одна из наиболее драматичных историй взлома (дискредитации) криптографических алгоритмов. Сначала в работе [96] был описан практически реализуемый метод нахождения коллизий для MD5. В ряде последующих работ были предложены более быстрые алгоритмы, что в конечном итоге привело к возмож-

ности нахождения коллизии всего за одну минуту на персональном компьютере. Эта ситуация с MD5 стимулировала переход с использованию более длинных хеш-значений (порядка 256 бит и выше) и к поиску альтернативных конструкций алгоритмов.

Последняя в линейке, хеш-функция MD6 [93] была разработана в 2008 году. Она устраняет выявленные в MD5 слабости, имеет длину хеш-значения до 512 бит и (наиболее отличительный признак) позволяет распараллеливать вычисление за счет использования древовидной структуры промежуточных хеш-значений. Однако MD6 не прошла даже во второй раунд конкурса NIST hash function competition (см. ниже) и в настоящее время широко не используется.

Вторая важная линейка хеш-функций — это стандарты США, разрабатываемые по заказу Национального института стандартов и технологий США (NIST). Данная линейка включает четыре поколения алгоритмов: SHA-0, SHA-1, SHA-2 и SHA-3. Аббревиатура SHA означает Secure Hash Algorithm (стойкий алгоритм хеширования). Первые два алгоритма (SHA-0 и SHA-1) сошли со сцены по тем же причинам, что и MD5. Третье поколение (SHA-2) [55] включает в себя четыре хеш-функции, называемые SHA-224, SHA-256, SHA-384 и SHA-512. Эти хеш-функции были разработаны в ответ на дискредитацию MD5 и предполагаемую дискредитацию похожей на нее SHA-1 [54], для которой позже действительно были найдены коллизии, хотя и с помощью более трудоемкого алгоритма, чем для MD5. Семейство SHA-2, также как и предшественники, строится на основе конструкции Меркля–Дамгарда и имеет увеличенную длину хеш-значений, обозначаемую числом после SHA (в битах). В настоящее время хеш-функции SHA-2 считаются стойкими, в литературе описаны атаки только на их ослабленные версии (с уменьшенным количеством раундов). Однако история с их предшественниками, базирующими на сходных конструкциях, поставила вопрос о создании новых алгоритмов, способных обеспечить стойкость хеш-функций на годы (столетия) вперед.

В 2007 году NIST объявил открытый конкурс на разработку стандарта хеш-функций нового поколения SHA-3 — NIST Hash Function Competition. В конкурсе приняли участие несколько десятков команд разработчиков со всего мира. После нескольких раундов отбора в 2012 году победителем было объявлено семейство алгоритмов Кессак (Кеччак), разработанное бельгийскими специалистами [37]. Данное семейство хеш-функций основано на конструкции губ-

ки, описанной выше. В стандарт SHA-3 вошли шесть вариантов хеш-функций: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 и SHAKE256. Во всех этих вариантах длина состояния губки составляет 1600 бит (в прежних обозначениях  $r + c = 1600$ ). В первых четырех вариантах число в обозначении алгоритма указывает длину хеш-значения. Емкость губки, как мы говорили ранее, должна быть равной удвоенной длине хеш-значения. Исходя из этого условия нетрудно получить величины  $r$  и  $c$  для всех четырех вариантов. Например, для SHA3-224 имеем  $c = 2 \times 224 = 448$ ,  $r = 1600 - 448 = 1152$ . Заметим, что  $r$  всегда получается большие длины хеш-значения. Это означает, что на этапе выжимания губки выполняется только одна итерация (точнее, значение хеш-функции берется из  $S^r$  с отсечением лишних бит). Последние два алгоритма позволяют генерировать хеш-значения произвольной длины, что достигается путем много-кратного выжимания губки. Однако криптографическая стойкость при этом определяется не длиной хеш-значения, а емкостью  $c = 256$  в случае SHAKE128 и  $c = 512$  в случае SHAKE256.

Следующая линейка хеш-функций — RIPEMD, RIPEMD-160, RIPEMD-256 и RIPEMD-320 [74, 49] была разработана группой ученых из Католического университета Лёвена, Бельгия, и часто рассматривается как неофициальный стандарт Европы, т.к. разработка начиналась в рамках программы RACE (Research in Advanced Communications in Europe — исследования в области передовых коммуникаций в Европе). Все хеш-функции этой линейки построены на базе конструкции Меркля–Дамгарда. Первая в линейке хеш-функция (RIPEMD) была успешно атакована вместе с MD5 [96] и вышла из употребления. В основе остальных хеш-функций лежит модифицированный алгоритм с увеличенными длинами хеш-значений (которые указываются числом, входящим в обозначение). Эти алгоритмы считаются стойкими, описаны атаки только на их ослабленные версии. Линейка RIPEMD менее популярна, чем SHA, но используется в современных системах. Например, в системе Bitcoin (глава 9) RIPEMD-160 используется для хеширования открытых ключей.

Для разработчиков систем защиты информации в России большой интерес представляют и часто просто обязательны для использования отечественные хеш-функции, закрепленные в стандартах: ГОСТ Р 34.11-94 [7] и Стрибог (ГОСТ Р 34.11-2012) [10]. Обе хеш-функции построены на базе конструкции Меркля–Дамгарда. Хеш-функция ГОСТ Р 34.11-94 была разработана ФАПСИ. В качестве

функции сжатия в ней используется блоковый шифр Магма, но с другими значениями S-боксов, не закрепленными стандартом и выбираемыми свободно. С 2013 года данная хеш-функция официально выведена из употребления в связи с введением в действие нового стандарта ГОСТ Р 34.11-2012. Новая хеш-функция, носящая название Стрибог, была разработана ФСБ России с участием ОАО «ИнфоТeКС» (напомним, что они же были разработчиками нового российского шифра Кузнечик). Стрибог может формировать хеш-значения двух длин — 256 и 512 бит — с помощью одного и того же алгоритма. Функция сжатия идеологически построена подобно шифру Кузнечик, в частности, использует в точности такой же S-бокс для реализации нелинейного слоя. В отличие от зарубежных хеш-функций, описанных выше, алгоритм Стрибог не создавался в рамках открытых программ или конкурсов. Сразу после его опубликования стали появляться публикации об атаках на ослабленные версии алгоритма (с уменьшенным числом раундов функции сжатия). В ответ на это компания «ИнфоТeКС» объявила открытый конкурс на лучшую атаку против Стрибога. Победителем конкурса в 2015 году стала работа [62], в которой была предложена атака нахождения второго прообраза (т.е. атака, нарушающая третье свойство криптографических хеш-функций, см. начало раздела), требующая  $2^{266}$  вызовов функции сжатия для сообщений, содержащих более  $2^{259}$  блоков. Ясно, что при такой сложности атака не осуществима на практике, хотя в теоретическом плане она значительно более эффективна, чем предполагаемая изначально сложность взлома порядка  $2^{512}$  операций.

Сходное со Стрибогом устройство имеет хеш-функция Whirlpool [94], только она использует в качестве функции сжатия аналог шифра AES. Данная хеш-функция была разработана одним из авторов AES (Rijndael) В. Рейменом и П. Баррето (Paulo Barreto) в рамках исследовательского проекта NESSIE (New European Schemes for Signatures, Integrity and Encryption — новые европейские схемы для подписей, целостности и шифрования), который выполнялся в 2000–2003 годах в качестве европейской альтернативы AES. Хеш-функция Whirlpool была рекомендована NESSIE и вошла в международный стандарт ISO/IEC 10118-3, принятый совместно Международной организацией по стандартизации (ISO) и Международной электротехнической комиссией (IEC). Длина хеш-значения Whirlpool составляет 512 бит. В настоящее время не известно никаких атак против этой

хеш-функции.

Наконец, назовем еще одну довольно интересную линейку хеш-функций — BLAKE и BLAKE2. Они разрабатывались международной командой криптографов. Хеш-функция BLAKE [31] участвовала в конкурсе SHA-3, дошла до финала, где уступила алгоритму Кессак. Затем была разработана ее оптимизированная по скорости версия BLAKE2 [32], которая сегодня наиболее активно используется. Хеш-функции линейки BLAKE базируются на еще одной альтернативной конструкции HAIFA [39] и в качестве ядра используют не блоковый, а потоковый шифр, а именно шифр ChaCha [35]. BLAKE2 — это на самом деле целое семейство хеш-функций, обладающих подобно SHA-3 различными, в том числе и произвольными, длинами хеш-значений, имеющих варианты реализаций на компьютерах с размерами слов от 64 до 8 бит, допускающих распараллеливание на различное число ядер процессора. Авторы позиционируют эти хеш-функции как более быстрые, чем MD5, SHA-1, SHA-2 и SHA-3, см. официальный сайт <https://blake2.net>. В настоящее время не опубликовано никаких успешных атак на хеш-функции линейки BLAKE. Данные хеш-функции применяются в многочисленных программных продуктах, среди которых отметим проекты GNU, OpenSSL, WhatsApp, WinRAR.

В заключение отметим, что все рассмотренные в нашем обзоре криптографические хеш-функции, кроме RC6, не защищены патентами и могут свободно использоваться.

# Глава 9. КРИПТОВАЛЮТЫ И БЛОКЧЕЙН

## 9.1. Введение

В последние годы некоторые криптографические методы и основывающиеся на них системы стали находить применение в экономике, финансовой системе, да и жизни общества в целом. Среди них мы назовем систему блокчейн (blockchain) и криптовалюты, первая из которых — биткоин (bitcoin) стала поистине всемирно известна.

Криптовалюта биткоин использует некоторые криптографические методы, представляющие и самостоятельный интерес. Поэтому в этой главе мы сначала описываем отдельно эти методы, а в последних разделах показываем, как они используются в конструкции биткоин.

## 9.2. Доказательство выполнения работы (proof-of-work) и Хэшкэш (Hashcash)

По-видимому, первая система «доказательства выполнения работы» (proof-of-work) была предложена в работе Дворк и Наора [51] как средство борьбы со спамом. Напомним, что спамеры рассылают по сотням тысяч, а то и миллионам адресов один и тот же рекламный текст, засоряя почтовые ящики пользователей электронной почтой.

Авторы [51] рассматривали взаимодействие почтового сервера с клиентами, которые обращаются к нему для отправки электронных писем. В [51] предложили схему, в которой каждый клиент, обратившийся к серверу для отправки письма, должен сначала решить некоторую вычислительную задачу, причем нахождение решения данной задачи требует от клиента некоторого заметного времени вычисления (скажем, несколько секунд), что, собственно, и объясняет название термина «доказательство выполнения работы». При этом задача выбирается такой, что проверка правильности решения сервером проводится очень быстро.

Предполагается, что затрата нескольких секунд для решения задачи при отправке письма не очень осложнит жизнь «честного» клиента, но будет серьезным препятствием для спамера, отправляющего тысячи копий, т.к. ему пришлось бы решать задачу «доказательства работой» для каждой из них заново, и суммарные затраты его времени могли бы составить несколько часов.

Хотя данный метод борьбы со спамом не нашел практического применения, он оказался полезен при построении криптовалют и решении ряда других задач, и в настоящее время известны различные варианты его реализации. Мы в этой книге рассмотрим подробно один такой алгоритм, предложенный в 2002 году Бэком [33], и получивший название «Хэшкэш» (Hashcash).

В рассматриваемом протоколе два участника — сервер и клиент. Клиент хочет получить некоторые услуги от сервера, а сервер требует решения некоторой трудоемкой задачи до предоставления этих услуг. (Пример услуг — отправка электронного письма, доступ к базе данных и т.п.) Мы предполагаем, что клиент и сервер связаны компьютерной сетью и могут использовать одну и ту же хеш-функцию  $h$ . Перейдем к пошаговому описанию.

**Шаг 1.** Клиент сообщает серверу, что хочет получить некоторую услугу.

**Шаг 2.** Сервер генерирует случайное слово  $s$  и высылает его клиенту вместе с целым числом  $k$ . (В реально применяемых протоколах длина двоичной записи  $s$  несколько сотен бит, а величина  $k$  — несколько десятков.)

**Шаг 3** («работа» клиента). Клиент находит такое слово  $x$ , для которого

$$h(s||x) |_1^k = 00\dots 0, \quad (9.1)$$

где  $w |_1^k$  —  $k$  первых символов слова  $w$ ,  $a||b$  — конкатенация слов  $a$ ,  $b$  и  $00\dots 0$  — слово, состоящее из  $k$  нулей.

**Шаг 4.** Клиент высылает серверу слово  $x$ . Сервер вычисляет величину  $h(s||x)$  и, если  $k$  первых символов этого слова нули, то предоставляет клиенту требуемые услуги, иначе — не предоставляет.

Изучим свойства этого протокола, начав с рассмотрения основного, третьего, шага. Главный вопрос — как найти слово  $x$ , удовлетворяющее (9.1). Оказывается, что для «идеальной» хеш-функции нет лучшего подхода, чем последовательное вычисление значений  $h(s||x_1)$ ,  $h(s||x_2)$ ,  $h(s||x_3)$ , ..., где  $x_1, x_2, x_3, \dots$  — генерируемые случайно слова (на практике это часто просто последовательные числа 1, 2, 3, ...).

Для того чтобы оценить время работы описанного протокола, нам понадобится вспомогательное

**Утверждение 9.1.** *Пусть вероятность наступления при одном испытании некоторого события  $A$  равна  $p$ . Пусть испытания проводятся независимо до первого наступления  $A$  и  $\nu$  — случайная величина, равная количеству этих испытаний. Тогда*

$$E(\nu) = 1/p, \quad (9.2)$$

$$P\{\nu \geq n\} = q^{n-1}, \quad (9.3)$$

где  $E(\nu)$  — среднее значение  $\nu$ ,  $q = 1 - p$ ,  $n$  — целое число.

**Доказательство.** Легко видеть, что  $P\{\nu = n\} = pq^{n-1}$ . (Действительно,  $\nu = n$ , если сначала  $A$  не происходит  $n - 1$  раз — вероятность этого  $q^{n-1}$ , а затем происходит — вероятность этого  $p$ ). Такие вероятности событий задают так называемое геометрическое распределение, для которого среднее значение равно (9.2), см. [26], а (9.3) следует из следующих равенств:

$$P\{\nu \geq n\} = p \sum_{i=n-1}^{\infty} q^i = pq^{n-1} \sum_{i=0}^{\infty} q^i = pq^{n-1} (1/(1-q)) = q^{n-1}.$$

(Здесь мы использовали  $p + q = 1$  и формулу для суммы геометрической прогрессии.)  $\square$

С учетом известного равенства  $\lim_{p \rightarrow 0} (1-p)^{(1/p)} = e^{-1}$  при малых  $p$  мы можем записать результат утверждения 9.1 в виде

$$P\{\nu \geq n\} \approx e^{-np}, \quad (9.4)$$

где  $e = 2.71828\dots$  — число Эйлера.

Вернемся теперь к рассматриваемому протоколу и получим полезное

**Следствие 9.2.** Предположим, что перебор слов  $x$  на шаге 3 до выполнения условия (9.1) организован так, что оно выполняется с вероятностью  $2^{-k}$ . Тогда в рассматриваемом протоколе среднее количество перебираемых значений до нахождения требуемого на шаге 3 слова  $x$  равно  $1/2^{-k} = 2^k$ , а вероятность того, что потребуется вычисление более  $\lambda 2^k$  значений хеш-функции, примерно равна  $e^{-\lambda}$ .

Доказательство следствия следует из (9.2) и (9.4) при  $p = 2^{-k}$ .

Описанный протокол имеет довольно много полезных свойств. Во-первых, необходимый объем вычислений для решения задачи легко варьируется путем изменения величины  $k$ , т.к. среднее число операций пропорционально  $2^k$ , см. следствие 9.2.

Во-вторых, мы описали так называемый интерактивный протокол, когда сервер и клиент обмениваются несколькими сообщениями. Его можно легко превратить в неинтерактивный. Для этого сервер может потребовать от клиента выполнения шага 2, т.е. поручить ему генерировать и слово  $s$  (конечно, пара  $s||x$  может порождаться как одно слово  $u$ ). Клиент, нашедший такое слово, может отправлять его серверу для получения услуги; предварительный обмен сообщениями не требуется.

Здесь, однако, возникают две легкопреодолимые трудности. Клиент может попробовать использовать найденное им слово  $u$  несколько раз. Для защиты от этого серверу достаточно хранить список ранее полученных слов  $u$  и при получении от какого-либо клиента нового  $u$  предоставлять услугу только при отсутствии  $u$  в списке ранее полученных.

Другая, сходная трудность — клиент может попытаться использовать слово  $u$ , найденное для одного сервера, для получения услуги от другого. Для решения этой проблемы сервер может потребовать от клиентов присыпать ему слова вида  $servername||u$ , для которых, по-прежнему,  $k$  начальных символов в  $h(servername||u)$  равны 0. (Здесь  $servername$  — имя сервера.) Предполагается, что имя сервера общеизвестно, скажем, хранится на его сайте. В этом случае клиент не сможет использовать найденное им слово  $servername||u$  для другого сервера.

В заключение этого раздела мы рассмотрим еще одну схему «доказательства работой», находящую широкое практическое применение. В предыдущей схеме трудоемкость задачи определялась числом

начальных нулевых символов в значении хеш-функции — в среднем требовалось вычислить  $2^k$  значений хеш-функции. Как мы отмечали, иногда может возникнуть необходимость в увеличении трудоемкости задачи (например, при появлении более мощных компьютеров). Такое увеличение легко получить, перейдя от  $k$  к  $k + 1$  и, тем самым, увеличив среднее число перебираемых вариантов вдвое, до  $2^{k+1}$ . В некоторых ситуациях такой шаг — увеличение в два раза — может оказаться слишком большим, и система с возможностью более плавного увеличения трудоемкости может оказаться предпочтительной. Такая система известна и также базируется на хеш-функции.

Для ее описания рассмотрим хеш-функцию  $h$ , значения которой — слова длины  $\alpha$  бит. Заметим сразу, что их можно рассматривать как целые числа из диапазона  $0, 2^\alpha - 1$  (в современных стандартах хеш-функций  $\alpha = 256$  или  $512$  бит). Заменим в вышеописанном протоколе шаг 3 следующим образом:

**Шаг 3\*** («работа» клиента). Клиент находит такое слово  $x$ , для которого

$$h(s||x) \leq \Omega, \quad (9.5)$$

где целое число  $\Omega$  — параметр метода, для которого должны выполняться условия  $0 \leq \Omega \leq 2^\alpha - 1$ . Понятно, что при случайному выборе слова  $s||x$  вероятность выполнения неравенства (9.5) равна  $\Omega/2^\alpha$  и, следовательно, при увеличении  $\Omega$  на единицу эта вероятность увеличивается на  $1/2^\alpha$ . При  $\alpha = 256$  или  $512$  эта величина очень мала, поэтому возможно плавное уменьшение  $\Omega$ , и увеличение обратной величины, равной, как мы знаем, среднему количеству перебираемых значений хеш-функции (т.е. объему работы). Стоит отметить, что первая система (с  $k$  нулями в начале значения хеш-функции) фактически является частным случаем второй, если мы возьмем  $\Omega = 2^{-k}$ .

### 9.3. Датирование документов (time-stamping)

Самые различные фирмы, организации, банки, а иногда и отдельные люди, ведут записи, фиксирующие расходы и доходы. До появления «безбумажной информатики» в крупных организациях такие записи собирались в так называемые бухгалтерские книги (ledger), которые велись таким образом, чтобы, насколько возможно, исключить замену (подмену) уже заполненных страниц (например, для этого книги

прошивались и опечатывались, хранились в сейфах и т.п.). Заметим, что необходимость надежно хранить записи о каких-либо событиях возникает не только в финансовой сфере, но и, например, в организациях, фиксирующих браки и разводы, в нотариальных службах, в больницах при ведении истории болезни, в патентных бюро и многих других областях.

С наступлением информационной эры записи о последовательности событий стали хранить в компьютерных базах данных, и перед специалистами по защите информации возникла задача организовать их таким образом, чтобы отдельные записи не могли быть несанкционированно заменены.

Одно из первых решений было предложено в начале 90-х годов прошлого века в работах Хабера, Сторнетты и Байера [63, 34], которые рассматривали задачу «датирования» цифровых документов, т.е. снабжения их информацией о времени создания (time-stamp). Датирование напоминает нанесение на почте штемпеля на конверт, где указано время его отправки (с точностью до часа), что и объясняет название time-stamp. Мы начнем этот раздел с рассмотрения задачи датирования цифровых документов, т.к. она представляет самостоятельный практический интерес и при ее решении были открыты новые методы, которые в дальнейшем стали использоваться во многих криптографических системах.

Рассмотрим некоторое сообщество, участники которого время от времени производят документы  $x_1, x_2, \dots$ , представленные в виде цифровых файлов. Например, это может быть биржа, участники которой заключают договоры о купле-продаже, или бухгалтерия, где создаются финансовые документы и т.п. Наша задача — построить систему, в которой каждый документ был бы датирован, т.е. снабжен сведениями о времени его создания, причем так, чтобы никто не мог создать документ «из прошлого», т.е. с датой, соответствующей прошедшему времени.

Сначала рассмотрим решение этой задачи с участием доверенной третьей стороны (trusted third party, TTP), называемой в данном случае «лицо, наносящее временной штамп» (Time Stamping Authority, в дальнейшем — TSA). Будем считать, что все участники и TSA связаны компьютерной сетью, что они договорились использовать одну и ту же хеш-функцию  $h$  и что TSA имеет свою электронную подпись, подлинность которой может проверить любой участник. (Мы подробно разбирали методы электронной, или циф-

ровой подписи в главе 4. Напомним, что для использования этих методов TSA должно иметь секретный и соответствующий ему открытый ключи. При подписании текста TSA задействует секретный ключ. Открытый ключ TSA должен быть достоверно известен всем участникам сообщества, и с его помощью каждый участник может проверить подлинность подписи TSA.) Обозначим подпись TSA к документу  $u$  через  $\Sigma_{\text{TSA}}(u)$ , тогда подписанный документ будет выглядеть как  $u \parallel \Sigma_{\text{TSA}}(u)$ .

Опишем теперь протокол датирования документа. Пусть один из участников, Алиса, создала документ  $x$  и хочет добавить к нему сведения о дате. Опишем работу протокола по шагам:

**Шаг 1.** Алиса вычисляет хеш-функцию  $h(x)$  и отправляет ее к TSA.

**Шаг 2.** TSA, получив  $h(x)$ , формирует слово (файл)  $h(x) \parallel t_x$ , где  $t_x$  — время получения документа  $x$  (скажем, в заранее обусловленном формате: ГОД, МЕСЯЦ, ДЕНЬ, ЧАС, МИНУТА, СЕКУНДА).

**Шаг 3.** TSA подписывает слово  $h(x) \parallel t_x$ , т.е. вычисляет подпись  $\Sigma_{\text{TSA}}(h(x) \parallel t_x)$ , и высыпает ее и  $t_x$  Алисе.

**Шаг 4.** Алиса, получив время и подпись TSA, формирует документ с «заверенной» датой  $w(x) = x \parallel t_x \parallel y$ , где  $y = \Sigma_{\text{TSA}}(h(x) \parallel t_x)$ .

Любой участник, скажем Боб, может проверить подлинность даты в некотором документе  $w^* = x^* \parallel t_x^* \parallel y^*$ . Для этого Боб должен вычислить  $h(x^*)$  и проверить подлинность подписанного сообщения  $h(x^*) \parallel t_x^* \parallel y^*$ , исходя из того что  $y^* = \Sigma_{\text{TSA}}(h(x^*) \parallel t_x^*)$ .

Если это сообщение выдерживает проверку, то Боб делает вывод о том, что действительно документ  $x^*$  был датирован в момент  $t_x^*$ ; в противном случае Боб делает вывод о том, что  $w^*$  подделан, т.е.  $x^*$  был создан после даты  $t_x^*$ . Подтверждает этот вывод следующее

**Утверждение 9.3.** В описанной системе никто из участников не может сформировать документ в момент времени  $T_1$  с датой  $T_2$ , предшествующей  $T_1$ .

**Доказательство.** Дату каждому документу присваивает только TSA, который, по определению, делает это честно. Поэтому единственный путь — «приделать» дату  $T_2$  к данному тексту  $x$  без

участия TSA. Однако это невозможно, т.к. для этого потребуется подделать подпись TSA.  $\square$

Таким образом, мы решили задачу надежного датирования. Полученная система довольно проста и находит практическое применение. Однако надежность этой системы зависит не только от качества используемых криптографических методов (хеш-функция, электронная подпись), но и от честности «лица, наносящего временной штамп» (TSA). Поэтому возникает совершенно естественный для современной криптографии вопрос: нельзя ли построить надежную систему датирования, независящую от «честности» TSA. Положительное решение было предложено в первых работах по методом датирования [63, 34], причем было описано два различных метода построения системы датирования, в одной из которых нечестность TSA сразу обнаруживается (и поэтому он обязан быть честным), тогда как в другой системе TSA просто отсутствует.

В следующем разделе мы рассмотрим предложенную в [63, 34] систему датирования, в которой TSA не может «приделать» фальшивое время, т.е. обязан быть честным. Важно отметить, что в этой системе впервые появилась цепочка связанных данных (или блоков данных — blockchain), нашедшая широкое применение в криптовалютах и многих других приложениях.

## 9.4. Блокчейн (blockchain)

Описываемая в этом разделе система находит самое широкое практическое применение, причем разработкой конкретных приложений занимаются крупные информационные компании, а внедрением — гигантские банки и даже правительства некоторых стран. В нашем описании блокчейн будет построен в процессе развития системы датирования. Полученная система датирования будет обладать уникальным свойством — никто, в том числе и TSA, не сможет изменить (подделать, заменить) ни один документ. Точнее, такая попытка сразу будет обнаружена всеми участниками. Понятно, что такая система представляет интерес как основа бухгалтерской книги, ведущейся, скажем, в банке или фирме для учета финансовых операций, или в больнице, где хранимые документы — последовательно назначаемые больному процедуры и лекарства, и во многих других ситуациях.

Итак, пусть Алиса создала документ  $x$  и хочет снабдить его заверенной датой. В рассматриваемой схеме датирование документа будет проводить TSA, которое вышлет Алисе соответствующий сертификат. В качестве примера можно считать, что TSA — это банк, Алиса — клиент банка и у нее есть там счет, документ — это распоряжение о покупке какого-либо товара или оплаты услуг. Считаем, что банк ставит на документе дату его исполнения.

В рассматриваемом протоколе TSA присваивает каждому документу индивидуальный порядковый номер ( $n$ ) и нам будет удобно обозначать имя  $n$ -ого клиента как  $ID_n$  (идентификатор клиента), а высылаемый им файл — как  $x_n$ . Представим последовательность действий следующим образом:

**Шаг 1.** Алиса вычисляет хеш-функцию  $h(x)$  и сообщает в TSA о желании датировать документ.

**Шаг 2.** TSA, получив  $h(x)$ , присваивает номер данному запросу (пусть это будет  $n$ ) и вычисляет сертификат

$$C_n = (n, t_n, ID_n, y_n, L_n), \quad s_n = \Sigma_{\text{TSA}}(C_n), \quad (9.6)$$

где, как и ранее,  $t_n$  — время получения  $h(x)$ ,  $s_n$  — подпись TSA к  $C_n$ ,  $y_n = h(x)$ ,  $ID_n = \text{Alice}$ , а  $L_n$  — «связующая информация» (linking information) или, короче, «связка»:

$$L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, h(L_{n-1})). \quad (9.7)$$

Затем TSA дожидается поступления следующего запроса, которому присваивается номер  $n + 1$ , и высылает Алисе  $C_n$ ,  $s_n$  и имя последующего клиента  $ID_{n+1}$ .

**Шаг 3.** Алиса, получив  $C_n$ ,  $s_n$ ,  $ID_{n+1}$ , может проверить правильность датирования, убедившись, что время  $t_n$  правильное,  $y_n$  совпадает с отправленным ей  $h(x)$  и подпись TSA  $s_n$  верна.

Заметим, что в этой схеме все документы как бы связаны в одну цепь в порядке их формирования TSA через  $L_n$ :  $n$ -й документ привязан к  $(n - 1)$ .

Допустим теперь, что Боб решил проверить подлинность даты на документе Алисы  $x$ . Для этого он запрашивает у Алисы сертификат  $C_n$ ,  $s_n$  и  $ID_{n+1}$ , проверяет подпись TSA  $s_n$ . Если подпись не

верна, он делает вывод о попытке подделки сертификата. Верности подписи было бы достаточно для проверки подлинности документа, если бы Боб доверял TSA. Но Боб предполагает, что не только Алиса, но и TSA могут не быть честными. По этой причине Боб проводит еще одну проверку: сначала он вычисляет  $h(L_n)$ , которое мы обозначим через  $h^*$ , а затем запрашивает сертификат у  $ID_{n+1}$ , который по запросу вышлет ему  $C_{n+1}, s_{n+1}, ID_{n+2}$ . Из  $C_{n+1}$  Боб найдет  $L_{n+1}$ , по которому, в свою очередь, найдет  $h(L_n)$ , см. (9.7). Теперь он может сравнить это значение с ранее вычисленным  $h^*$ . Если они не совпадают, то Алиса и TSA пытались подделать дату. Однако даже в случае совпадения Боб может провести еще одну дополнительную проверку: аналогично только что проведенной он может вычислить  $h^*(L_{n+1})$ , затем запросить сертификат у  $ID_{n+2}$ , который по запросу вышлет ему  $C_{n+2}, s_{n+2}, ID_{n+3}$ . По этим данным он точно также найдет  $h(L_{n+1})$  и сравнит это значение с  $h^*(L_{n+1})$ . Если они совпадают, а у Боба все же остаются сомнения, он может аналогично сравнить  $h(L_{n+2})$  и  $h^*(L_{n+2})$ . При желании он может двигаться по цепочке «информационно связанных» документов все дальше, переходя от  $n+2$  к  $n+3$  и т.д., до последнего датированного документа. Важно отметить, что двигаясь по цепочке сертификатов, Боб не получает сведений о содержании датированных документов, т.к. сертификаты содержат только значения их хеш-функций (но не сами документы).

Таким образом, описанное соединение документов в цепочку при помощи связок  $(L_n)$  в случае изменения (искажения, подделки)  $i$ -ого сертификата  $C_i$  приводит к тому, что все последующие сертификаты  $C_{i+1}, C_{i+2}, \dots$  становятся недействительными. Это объясняется тем, что связка  $L_{i+1}$  «привязана» к предыдущей,  $L_i$ , см. (9.7).

Остановимся теперь на двух обобщениях описанной системы. Во-первых, можно формировать сертификат не для одного документа  $x_n$ , а для целой группы, объединяя в один «блок» сообщения, поступившие за определенный период времени (скажем, за 10 минут). В этом случае сертификат (9.6) может содержать несколько  $ID$  и соответствующих им  $y$ , снабженных одним  $L$ . В этой ситуации естественно использовать название «блокчейн» (blockchain), или цепочка блоков.

Другое обобщение связано с заменой цепочки на дерево. Точнее, представим себе, что последовательность блоков и соответствующих им сертификатов растет как бинарное дерево, так, что у некоторых

узлов дерева, соответствующим блокам, имеются два сына. Каждый блок, соответствующий сыну, привязан при помощи связки (9.7) к  $L$  отца, но не связан с  $L$  своего «брата». Такие деревья с узлами, связанными при помощи хеш-функций, называются деревьями Меркля (Merkle Tree). В отличие от системы, когда все блоки связаны в одну цепь, древовидная структура позволяет просматривать при проверке не все данные, следующие за проверяемым блоком, а только часть дерева, выходящую из проверяемого узла. Возможны ситуации, когда такое уменьшение проверок дает некоторое уменьшение трудоемкости, правда за счет уменьшения надежности — короткую цепочку легче подделать.

В построенной цепочке блоков изменение одного элемента приводит к тому, что у всех более поздних блоков сертификаты становятся недействительны, что позволяет обнаружить искаженный блок. Если кто-либо хочет скрыть изменения в блоке, ему необходимо пересчитать сертификаты всех последующих. Для того чтобы сделать решение этой задачи вычислительно трудным, в [70] было предложено использовать хешкэш. Для этого мы изменим формирование связывающей информации в (9.7) следующим образом:

$$L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, h(L_{n-1}), \text{nonce}_n). \quad (9.8)$$

где  $\text{nonce}_n$  — такое двоичное слово, что первые  $k$  символов  $h(L_n)$  равны нулю. Другими словами, мы использовали метод хешкэш, для которого в данном случае среднее количество слов, которые надо перебрать для нахождения  $\text{nonce}_n$ , равно  $2^k$ . Теперь после изменения  $n$ -ого блока в цепочке злоумышленнику придется находить последовательно  $\text{nonce}_{n+1}$ ,  $\text{nonce}_{n+2}$ ,  $\text{nonce}_{n+3}$  и т.д., до конца цепочки, что при больших  $k$  требует большого времени вычислений. В свою очередь, это повышает устойчивость описанной системы блокчейн.

В настоящее время известны несколько практически реализованных систем блокчейн, отличающиеся правилами формирования новых блоков и рядом других деталей. В следующем разделе мы подробно опишем блокчейн, используемый в криптовалюте биткоин. Важно отметить, что ее надежность доказана не только теоретически, но и практически — система успешно работает с 2009 года.

## 9.5. Биткоин (bitcoin) и криптовалюты

Описанная в основополагающей работе Сатоши Накомото (Satoshi Nakamoto, 2008) [70] криптовалюта биткоин имеет довольно интересную историю появления и превращения в мощную систему финансовых расчетов. В [71] можно найти описание истории возникновения многих идей и конструкций, связанных с появлением криптовалют.

В этой книге мы встречались с задачей организации анонимных расчетов с использованием компьютерных сетей и нашли решение, когда даже банк не знает, на какие цели расходуют свои средства его клиенты. Однако в этой системе интересы клиентов не защищены от «некорректного» поведения банка. Например, банк может разориться и сбережения клиентов пропадут. Центральный банк может «допечатывать» деньги, что приводит к их обесцениванию и, тем самым, уменьшает реальные сбережения клиентов. (В истории многих стран известны такие «допечатывания», приводившие к росту инфляции и, как следствие, потере сбережений.) Поэтому возникает задача построения системы расчетов, где никто из участников не может как-либо нарушить интересы остальных (речь, конечно, идет о расчетах по компьютерным сетям, а не, скажем, о введении золотых монет). Такую систему — биткоин — предложил в статье [70] и практически построил С. Накамото и данный раздел посвящен ее описанию.

Совсем кратко систему биткоин можно описать так: это блокчейн, поддерживаемый многими участниками. «Физически» участники являются узлами так называемой пиринговой (peer-to-peer) компьютерной сети, где каждый из них предоставляет доступ к хранимым данным для всех других. При этом все узлы сети обладают равными правами и отсутствует какой-либо центр управления. Описываемый блокчейн может рассматриваться как бухгалтерская книга (ledger), хранящая сведения о счетах участников и позволяющая им вести взаимные финансовые расчеты, причем копия этой книги хранится у каждого участника.

При проведении расчетов участники пользуются валютой, единица измерения которой называется биткоин (bitcoin), а 0.00000001 часть биткоина называется Сатоши (Satoshi). Так как и вся система расчетов, рассматриваемая криптовалюта и денежная единица называются одинаково — биткоин, мы для избежания путаницы будем часто использовать для обозначения одной «монеты» довольно рас-

пространенную аббревиатуру ВТС. Вопрос о возникновении монет («штамповке, отливке») будет рассмотрен позже. Перейдем к более подробному описанию системы, следя, в основном, [70].

## Транзакции и биткоины

У каждого участника есть открытый и закрытый ключи, которые мы обозначим через  $okey_{name}$  и  $skey_{name}$  соответственно. Все участники используют один метод цифровой подписи ECDSA и одну хеш-функцию SHA-256 (точнее, так называемую SHA-256d, когда SHA-256 применяется последовательно два раза — сначала к исходному сообщению, затем к полученному значению хеш-функции, т.е. для сообщения  $x$  вычисляется  $h(h(x))$ ).

Расчеты оформляются в виде транзакций, большая часть из которых суть записи о переводе биткоинов от одного участника к другому (при покупке какого-либо товара, при оплате услуг и т.п.). Участник получатель платежа идентифицируется его открытым ключом. Ниже мы покажем, что одна монета (один биткоин) определяется как последовательность электронных подписей всех транзакций, когда монета переходит от одного владельца к другому.

Скажем несколько слов об анонимности участников. Фактически, каждый из них представлен только своим открытым ключом. Более того, один «физический» участник может быть представлен несколькими открытыми ключами. Но в дальнейшем, для удобства, мы будем использовать имена Алиса и Боб.

Опишем теперь процесс проведения транзакции по передаче денег в виде протокола. Пусть Алиса (покупатель) хочет передать один ВТС (биткоин) Бобу (продавцу). При этом считаем, что Алиса получила этот биткоин при выполнении транзакции  $t_{n-1}$ , где  $n-1$  — порядковый номер.

**Шаг 1.** Алиса формирует следующий текст:

$$u_n = (t_{n-1}, 1 \text{ ВТС}, okey_{Bob}). \quad (9.9)$$

Этот текст фактически означает: «взять один биткоин, полученный мною в транзакции  $t_{n-1}$ , и передать его Бобу».

**Шаг 2.** Алиса подписывает  $u_n$ , используя свой секретный ключ  $skey_{Alice}$ , и формирует транзакцию

$$t_n = (u_n, s_n), \quad (9.10)$$

где,  $s_n$  — подпись Алисы к  $u_n$ .

**Шаг 3.** Алиса пересыпает  $t_n$  Бобу и другим участникам сети. (В дальнейшем, после проверки, это слово будет включено в общую бухгалтерскую книгу; см. следующий раздел).

Покажем теперь, как Боб (и любой другой участник!) может проверить подлинность  $t_n$ . Сначала он находит в доступной всем участникам бухгалтерской книге транзакцию  $t_{n-1}$  (на которую ссылается  $t_n$ , см. (9.9)) и узнает, что в ней 1 BTC был направлен Алисе. После этого он проверяет подпись в транзакции  $t_n$ , используя в алгоритме проверки открытый ключ Алисы  $okey_{Alice}$ .

Важно отметить, что только Алиса могла использовать биткоин, переданный ей в транзакции  $t_{n-1}$ , т.к. никто другой не знает секретный ключ  $skey_{Alice}$ , необходимый для создания корректной подписи в  $t_n$ . Отметим также, что Боб использовал только общедоступную информацию (открытые ключи и транзакции из общедоступной бухгалтерской книги), поэтому и любой другой участник совершенно аналогично может проверить подлинность транзакции  $t_n$ .

Другое важное замечание — теперь мы видим, что в рассматриваемой схеме одна монета BTC является в действительности цепочкой транзакций  $t_1, t_2, \dots, t_n$  (о «производстве» биткоинов в  $t_1$  будет сказано ниже). Важно отметить, что указанные номера относятся к транзакциям, в которых участвует данная монета. При этом, скажем, между  $t_2$  и  $t_3$  могли быть выполнены тысячи транзакций, в которых данная монета не участвовала.

Скажем также, что возможны транзакции, в которых участвуют несколько отправителей и/или получателей. Такие транзакции позволяют разменивать и, наоборот, объединять имеющиеся биткоины. Например, возможно 10 BTC в одной транзакции превратить в «монеты» 5 BTC, 3 BTC и 2 BTC и передать их разным участникам. Более того, предусмотрены транзакции, объединяющие перевод нескольких сумм в BTC нескольким адресатам. Например, 3 BTC и 5 BTC могут быть переведены трем адресатам по 4 BTC, 2 BTC и 2 BTC. Для корректных транзакций должно обязательно выполняться одно условие — сумма на входе транзакции должна быть не меньше суммы на выходе (т.е. деньги не должны возникать «из ничего».)

К сожалению, описанной проверки недостаточно для того, чтобы исключить возможность жульничества. У Алисы остается еще

одна возможность обмана — она может попытаться рассчитаться одной и той же своей электронной монетой несколько раз. Мы видели, что эта проблема легко решается при наличии банка, который запоминает номера всех использованных монет (тогда при получении монеты с ранее использованным номером банк обнаруживает попытку обмана). В системе биткоин нет банка, а проблема двойного использования монет решается за счет того, что список всех транзакций (т.е. копия бухгалтерской книги) хранится у каждого участника. Следовательно, если один из них попытается совершить недопустимую, жульническую, операцию, это будет замечено всеми участниками и попытка будет пресечена.

Наконец, отметим что система биткоин развивается, изменяется, обновляется ее программное обеспечение и появляются новые предложения по ее совершенствованию (Bitcoin Improvement Proposal). Кроме описанной нами транзакции по переводу денег появились и новые, которые предоставляют, в частности, большие возможности для повышения уровня анонимности. Однако все эти варианты транзакций предоставляют возможности проверки их корректности способами, близкими к вышеописанному.

Перейдем теперь к более подробному описанию бухгалтерской книги.

### **Формирование бухгалтерской книги и производство биткоинов**

В процессе функционирования платежной системы биткоин вновь появляющиеся транзакции объединяются в блоки, из которых формируется блокчейн, снабженный системой датирования и доказательства работой, описанных в предыдущих разделах.

Остановимся подробнее на формировании новых блоков в блокчейне. Каждый участник хранит копию всей бухгалтерской книги и может самостоятельно формировать очередной блок. При этом он должен найти такое слово попсе, что значение хеш-функции не превышает заданную величину  $\Omega$ , см. (9.5), что, естественно, требует использования его компьютера и занимает определенное время.

Участник, занимающийся формированием блоков, называется майнер (miner), а процесс формирования — майнинг (mining). Для того чтобы работа по формированию блоков транзакций была выгодна майнерам, она оплачивается, во-первых, отчислениями от тран-

закций, включаемых в блок, и, во-вторых, от прямого вознаграждения в виде некоторого количества биткоинов. Транзакция, порождающая новые биткоины, записывается первой в формируемый блок.

Таким образом, мы описали способ «штамповки» новых монет в описываемой платежной системе. Для получения и, тем самым, введения в обиход новых монет, майнер должен загрузить работой свои компьютеры и затратить на это электроэнергию. (Здесь можно провести очевидную аналогию с добычей золота или серебра для штамповки монет — работа старателей превращается в деньги. Понятно, что и послужило введением терминов майнер и майнинг: *miner* — шахтер или горнорабочий, *mining* — добыча руды, металлов и т.п.) Однако в системе биткоин процесс производства новых монет ограничен — после формирования каждого 210000 блоков вознаграждение майнера уменьшается вдвое. Так, первоначальный размер вознаграждения майнера был 50 BTC за блок, в 2008 году он уменьшился до 25 BTC, в 2012 — до 12,5 BTC. Так как величина вознаграждения уменьшается в геометрической прогрессии, то наступит момент, когда она станет меньше одного Сатоши ( $10^{-8}$  BTC) и поэтому не будет выплачиваться и их общее количество к тому моменту не превысит 21 миллион BTC. После этого работа майнеров по производству новых блоков будет вознаграждаться только за счет платы за транзакции. Для этого предусмотрено, что в транзакциях «входная» сумма будет превосходить «исходящую», а разница будет начисляться майнера. Например, если Алиса хочет перевести 1 BTC Бобу, то она может указать отправляемую величину как 1.001 BTC, а получаемую — 1 BTC; в этом случае 0.001 BTC получит майнер. (Стоит отметить, что и в обычных банках клиенты платят за перевод денег).

Сформировав новый блок, майнер высылает его всем участникам сети. Получив блок, каждый участник проверяет условие

$$h(h(\text{block} || \text{nonce})) \leq \Omega, \quad (9.11)$$

где  $\Omega$  — величина, определяющая объем перебора в доказательство проведения работы, см. определение в (9.5) (здесь, как обычно, значение хеш-функции рассматривается как число). Если проверяемое условие не выполняется, блок вообще не рассматривается. При выполнении этого условия участник проверяет корректность всех содержащихся в блоке транзакций (как это описано в предыдущем

разделе), а также отсутствие двойного использования денег. Если блок выдерживает эти проверки, то участник добавляет его в блокчейн (т.е. копию хранимой бухгалтерской книги). После этого каждый участник майнер начинает формировать новые блоки, «привязывая» их к полученному в блокчейн.

В системе биткоин формируемый блокчейн не может иметь разветвлений, т.е. это действительно должна быть цепочка, а не дерево. Однако возможна ситуация, когда разные майнеры прислали сформированные ими блоки почти одновременно, а в блокчейн может быть включен только один из них. Для разрешения этой коллизии используется так называемая сложность блока. Для ее описания напомним, что для каждого блока выполняется условие (9.11). Величину

$$\alpha = 1/h(h(\text{block} \parallel \text{nonce}))$$

мы назовем сложностью блока, но заметим, что иногда сложностью называют некоторую монотонно возрастающую функцию от этой величины. Название «сложность» для величины  $\alpha$  объясняется тем, что, как мы знаем, среднее количество перебираемых слов попсе до выполнения неравенства (9.11) равно  $1/\alpha$ .

Итак, пусть майнер получил почти одновременно два блока (скажем,  $B^1$ ,  $B^2$ ). В этом случае он включает в цепь тот блок, сложность которого больше. Если в дальнейшем он получает блоки, показывающие, что цепочка, начинающаяся с  $B^2$  растет быстрее, чем с  $B^1$ , то этот майнер начинает работать с ней, оставляя  $B^1$ . При этом транзакции, содержащиеся в  $B^1$  не исчезают, а включаются майнерами в последующие блоки. Однако транзакции, перечисляющие вновь «отштампованные» биткоины майнеру за формирование  $B^1$  исчезнут и лишние деньги не появятся. Вообще говоря, может возникнуть ситуация, когда участник видит, что сформированы две цепочки

$$\dots B^* \rightarrow B_1^1 \rightarrow B_2^1 \rightarrow \dots B_m^1$$

и

$$\dots B^* \rightarrow B_1^2 \rightarrow B_2^2 \rightarrow \dots B_n^2$$

и он должен решить, какую из них включать в блокчейн. В этом случае участник подсчитывает суммарную сложность всех блоков в первой цепочке, а затем во второй. Если первая величина больше, то он включает в блокчейн первую цепочку, в противном случае — вторую.

## Надежность системы биткоин

Известно, что вычислительная техника развивается очень быстро и возможна такая ситуация, что решение некоторой задачи сегодня требует, скажем, 10 минут, а через год появляется компьютер того же класса, затрачивающий на ее решение 7 минут. Для того чтобы время вычисления попсе оставалось примерно постоянным и не зависело от появления со временем все более мощных вычислительных средств и появления или исчезновения майнеров, в системе биткоин предусмотрена возможность изменения среднего числа перебираемых вариантов для нахождения попсе. Делается это следующим образом: после добавления очередных 2016 блоков оценивается среднее время, которое было затрачено на производство одного из них. Если эта величина стала меньше 10 минут, то величина  $\Omega$  уменьшается и, тем самым, увеличивается среднее количество перебираемых значений, которое требуется рассмотреть при нахождении попсе (напомним, это среднее равно  $1/\Omega$ , см. (9.5)). Благодаря этой процедуре, среднее время порождения одного блока остается неизменным (10 минут), несмотря на появление все более мощных вычислительных средств, используемых для нахождения попсе. Аналогичные преобразования проводятся и в случае, если среднее время добавления одного блока стало больше 10 минут (скажем, из-за уменьшения количества майнеров). Однако в этом случае  $\Omega$  не уменьшается, а увеличивается.

Другая возможная угроза системе — появление злоумышленника (Евы), пытающегося генерировать ложные блоки быстрее, чем честные майнеры. Если это ей удается, то цепь в блокчейне будет прирастать только ее блоками, а блоки честных майнеров не будут добавляться к цепи. Это возможно только в том случае, если Ева вычисляет попсе для своих блоков быстрее, чем честные майнеры. В свою очередь, это возможно только тогда, когда суммарные вычислительные мощности всех честных майнеров меньше, чем у Евы. Вообще говоря, можно рассмотреть ситуацию, когда Ева не один человек, а целая группа нечестных майнеров. В этом случае система будет устойчива к описанной атаке, если суммарные мощности честных майнеров будут больше, чем нечестных. Иногда это условие формулируется короче: система устойчива, если честных майнеров больше, чем нечестных.

# Глава 10. СЛУЧАЙНЫЕ ЧИСЛА В КРИПТОГРАФИИ

## 10.1. Введение

Мы видели в предыдущих главах, что случайные и псевдослучайные числа играют довольно важную роль в криптографии. Например, при описании многих протоколов мы встречались с необходимостью генерировать большие случайные числа или слова, используемые как секретные ключи. Вспомним также и шифр Вернама, совершенная секретность которого базируется на том, что ключевая последовательность состоит из равновероятных и независимых символов. Естественно, что задача генерирования последовательностей случайных чисел представляет большой интерес для разработчиков крипtosистем. Более того, с развитием криптографии выяснилось, что многие фундаментальные проблемы этой науки тесно связаны с генерированием и тестированием случайных чисел. Например, возможность построения надежных генераторов псевдослучайных чисел связана с проблемой существования односторонних функций, а одна из атак на блоковые шифры базируется на статистических тестах, предназначенных для выявления отклонений от случайности.

В данной главе мы попытаемся познакомить читателя с основными задачами, идеями и методами, связанными с генерированием и тестированием случайных чисел в криптографии. Мы начнем с основного вопроса: что такое случайное число или, более общо, последовательность случайных чисел? Пожалуй, первое, что приходит в голову — это последовательность, полученная при помощи подбрасываний симметричной монеты, если ее стороны пометить, скажем, нулем и единицей. Именно это определение, или, скорее понятие, и взято за основное, «первичное». Более формально, в качестве случайных чисел в криптографии рассматриваются последовательности из нулей и единиц, в которых появление символов независимо и их вероятности равны. (Мы иногда для краткости будем называть такие последовательности «абсолютно» случайными или просто случайны-

ми, если это не будет приводить к путанице.) Другие случайные числа, скажем, целые числа из некоторого диапазона, легко получить из последовательности случайных нулей и единиц и мы не будем останавливаться на этом вопросе.

Следующий вопрос, на котором мы остановимся — как генерировать случайные числа? В «обычной» жизни люди иногда используют случайные числа, получая их путем подбрасывания монеты или игральной кости, «вытягивания» игральной карты из колоды и т.п. Очевидно, эти способы мало пригодны в криптосистемах, где требуется производить большие последовательности случайных чисел с высокой скоростью. Для этой цели можно использовать другие физические процессы, обладающие высокой производительностью и сравнительно легко сочетающиеся с компьютерными системами. Например, в качестве физических процессов, содержащих «случайную» составляющую, можно использовать шумы, возникающие в электрических цепях и их элементах, счетчики физических частиц, движения манипулятора-мыши в руке человека, работающего за компьютером, и т.д. и т.п. Одна из основных задач, возникающих при использовании «физических» генераторов — преобразование порождаемых ими последовательностей в абсолютно случайные. Она будет кратко рассмотрена в следующем разделе.

Пожалуй самый популярный способ получения случайных чисел не связан с наблюдениями за каким-либо сложным физическим процессом, а базируется на проведении вычислений, (например, как в рассмотренных ранее генераторах для потоковых шифров). Получаемые таким способом числа называются псевдослучайными, что подчеркивает некоторое присущее им радикальное отличие от «истинно» случайных чисел. Действительно, мы не можем предсказать значение «истинно» случайного числа (генерируемого, скажем, физическим датчиком) до его появления, однако можем легко вычислить все значения, порождаемые псевдослучайным генератором. Это кардинальное различие легко видеть в случае генерирования ключа в системах передачи информации: даже если передатчик и приемник используют идентичные физические генераторы для символов ключа, они не получат одинаковые последовательности чисел (а если это маловероятное событие и произойдет, то они не узнают о его наступлении). Поэтому-то и требуется передавать порожденную физическим генератором последовательность от передатчика к приемнику по защищенному каналу. С другой стороны, используя одинак-

ковые генераторы псевдослучайных чисел, приемник и передатчик могут легко получить одинаковые последовательности. В разд. 10.3 мы рассмотрим некоторые вопросы, связанные с псевдослучайными числами.

К качеству случайных чисел, используемых в криптографии, предъявляются высокие требования. Прежде всего, требуется исключить возможные статистические отклонения от эталона: вероятности порождения нуля и единицы должны быть в точности равны  $1/2$  и генерируемые числа должны быть независимы. Для выявления таких отклонений используются специальные статистические методы, или тесты, которых к настоящему времени разработано довольно много. Так, Национальный институт стандартов и технологий США (NIST) рекомендует 16 таких тестов для применения в криптосистемах [76]. Мы также рассмотрим некоторые тесты в разд. 10.4.

Как мы уже отмечали ранее, многие разделы криптографии идеино связаны, причем иногда эти связи далеко не очевидны. Пример такой взаимосвязи дан в последнем разделе этой главы, где мы опишем атаку на блоковые шифры, которая тесно связана со случайными числами и статистическими тестами.

## 10.2. Задачи, возникающие при использовании физических генераторов случайных чисел

Мы будем считать, что физический генератор первоначально порождает последовательность из нулей и единиц, которые, вообще говоря, могут быть неравновероятны и/или зависимы. Такие последовательности необходимо преобразовывать в равновероятные и независимые и только после этого применять в криптосистемах. (Иногда такое преобразование называется «очисткой» или «стандартизацией».) При этом обычно рассматриваются отдельно методы преобразования для двух случаев: в первом считают, что символы исходной последовательности независимы, но, возможно, не равновероятны, а во втором случае предполагают, что символы могут быть и зависимы. Мы рассмотрим только методы решения первой задачи, что объясняется двумя причинами: во-первых, для многих физических генераторов (скажем, базирующихся на оценке интенсивности какого-либо физического процесса) показания, полученные в разные, непересекающиеся, интервалы времени, можно считать независимыми. Во-вторых,

известные методы преобразования зависимых последовательностей в независимые являются только приближенными — у них символы получаемой последовательности только «почти» независимы. Кроме того, авторам неизвестны примеры практического применения в криптосистемах методов преобразования зависимых последовательностей символов в независимые. Хотя отметим, что эта задача привлекает внимание многих исследователей и для ее решения предложено много интересных и остроумных алгоритмов.

Итак, в оставшейся части этого раздела мы рассмотрим методы преобразования последовательностей независимых нулей и единиц, вероятности порождения которых, возможно, не равны.

Фон Нейман (John von Neumann) впервые рассмотрел эту задачу и предложил первый алгоритм для ее решения. Для его описания мы введем следующие обозначения: дан бернуlliевский источник, порождающий символы из алфавита  $\{0, 1\}$  с вероятностями  $1 - p$  и  $p$  соответственно,  $0 < p < 1$ , причем  $p$  может быть неизвестно. Требуется преобразовать (или закодировать) порождаемую источником последовательность в такую, где вероятности появления нуля и единицы равны, т.е. в абсолютно случайную.

Фон Нейман предложил следующий метод: исходная последовательность разбивается на блоки (слова) длины 2, которые кодируются по следующему правилу:

$$00 \rightarrow \Lambda, \quad 01 \rightarrow 0, \quad 10 \rightarrow 1, \quad 11 \rightarrow \Lambda, \quad (10.1)$$

где  $\Lambda$  обозначает пустое слово. Например, порождаемая последовательность  $00\ 01\ 11\ 10\ 00\ 00\ 01$  будет трансформирована в абсолютно случайную последовательность  $010$ . Здесь первый нуль соответствует второму блоку, т.е.  $01$ , единица соответствует четвертому блоку, а второй нуль — последнему блоку. Так как вероятности порождения слов  $01$  и  $10$  совпадают (они равны  $(1 - p)p$  и  $p(1 - p)$ ), то, очевидно, в результате преобразования получается абсолютно случайная последовательность.

Из приведенного примера виден и недостаток метода, задаваемого правилом (10.1) — результирующая последовательность намного короче исходной. Точнее, легко видеть, что из  $t$  исходных символов получается последовательность из  $tp(1 - p)$  независимых символов. Например, если  $p$  близко к  $1/2$ , то из  $t$  символов в среднем получается  $t/4$ .

Элайес (Peter Elias) предложил метод преобразования, более экономно расходующий символы исходной последовательности, что достигается за счет перехода к кодированию блоков длины  $n$ ,  $n > 2$  (при  $n = 2$  методы Элайеса и фон Неймана совпадают). Для количественной оценки эффективности метода Элайес ввел величину  $\eta_n$ , определяемую как отношение среднего значения длины получаемого кодового слова к длине блока  $n$ . Он показал, что естественной верхней границей для величины  $\eta_n$  является двоичная энтропия источника  $h(p)$ , которая, напомним, определяется равенством  $h(p) = -(p \log p + (1 - p) \log(1 - p))$ . (Впрочем, легко понять, что величина  $h(p)$  — это максимально возможное значение отношения длин выходной и входной последовательностей, так как, неформально, энтропия есть мера неопределенности, или случайности исходной последовательности, с другой стороны, энтропия выходной абсолютно случайной последовательности равна ее длине, т.к. для нее энтропия одного символа составляет 1 бит.)

В методе Элайеса величина  $\eta_n$  приближается к энтропии Шеннона с ростом длины блока  $n$ , но при этом и сложность алгоритма быстро возрастает. Точнее, объем используемой памяти растет как  $2^n$ , что делает его практически неприменимым при длине блока несколько десятков. В работе [81] предложен метод, сложность которого существенно меньше — объем требуемой памяти растет как  $n \log^2 n$ , что позволяет применять его и при больших  $n$ . В этой же работе можно найти более подробное описание метода Элайеса.

### 10.3. Генераторы псевдослучайных чисел

Мы уже говорили о генераторах псевдослучайных чисел, когда рассматривали потоковые шифры (разд. 8.4). Практически все используемые в криптографии генераторы псевдослучайных чисел могут быть схематически представлены как некоторая функция от двух переменных  $F(s, i)$ , где  $i$  — номер генерируемого случайного двоичного слова некоторой фиксированной длины, а  $s$  — параметр или секретный ключ, часто называемый «семенем» (seed). Обычно  $s$  — двоичное слово, выбираемое случайно. Генерируемая псевдослучайная последовательность образуется как цепочка двоичных слов  $F(s, 1)$ ,  $F(s, 2)$ ,  $F(s, 3)$ , ...,  $F(s, k)$ , где  $k$  — требуемая длина последовательности. В разд. 8.4 были даны примеры таких генераторов (линейный конгруэнтный, RC4, HC-128 и получаемые при использовании блоко-

вых шифров в режимах OFB и CTR), поэтому мы не будем рассматривать новые примеры, а сформулируем более точно требования, предъявляемые к генераторам.

Любой генератор должен удовлетворять двум основным требованиям: во-первых, генерируемая псевдослучайная последовательность должна быть статистически неотличима от абсолютно случайной и, во-вторых, знание какой-либо начальной части последовательности (скажем,  $F(s, 1), F(s, 2), \dots, F(s, k-1)$ ) не должно позволять предсказывать следующий бит этой последовательности (точнее, наилучшее возможное предсказание должно быть таким: "очередной бит последовательности с вероятностью 0.5 принимает значение ноль и с вероятностью 0.5 – единица"). Здесь сразу следует отметить, что эти требования невыполнимы даже при неизвестном значении слова  $s$ , если не ограничивать сложность тестов и методов прогноза, измеряемую объемом памяти и/или временем вычислений. Для объяснения этого явления мы сначала заметим, что представляют интерес только такие генераторы, у которых длина выходной последовательности больше длины семени  $s$  (в противном случае генератор был бы не нужен – можно было бы использовать  $s$  как случайную последовательность). Теперь предположим, что дана последовательность бит  $z_1, z_2, \dots, z_k$ , порожденная генератором, причем ее длина ( $k$ ) больше длины слова  $s$ . Тогда для предсказания следующего символа  $z_{k+1}$  можно поступить следующим образом: будем поочередно подавать на вход генератора всевозможные значения ключа  $s$  (скажем, в лексикографическом порядке) и генерировать первые  $k$  бит. Если полученные символы при некотором  $s = u$  совпадают с  $z_1, z_2, \dots, z_k$ , то значение ключа равно  $u$  (точнее, это справедливо при выполнении некоторых естественных условий, на которых мы не будем останавливаться, ограничившись неформальным рассмотрением). Естественно, при найденном ключе можно вычислить следующий порождаемый символ  $z_{k+1}$ , т.е. однозначно предсказать его значение.

Таким образом, мы видим, что метод прямого перебора ключей позволяет предсказать следующий бит последовательности, порождаемой генератором псевдослучайных чисел, поэтому-то она и не является «истинно» случайной. Однако время прямого перебора ключей пропорционально  $2^{|s|}$ , где  $|s|$  – длина слова  $s$ , и, очевидно, оно может быть сделано сколь угодно большим при увеличении  $|s|$ . (Это и определяет выбор длины ключа – время выполнения  $2^{|s|}$

операций должно быть гигантским для лучших современных компьютеров.) Поэтому в криптографии оба вышеупомянутых требования к генераторам псевдослучайных чисел несколько модифицированы: во-первых, генерируемая псевдослучайная последовательность должна быть статистически неотличима от абсолютно случайной *за небольшое время вычислений* и, во-вторых, знание какой-либо начальной части последовательности не должно позволить предсказать следующий бит этой последовательности *за небольшое время вычислений* (здесь «большое время» обычно означает «требуются годы вычислений на суперкомпьютерах»).

Для многих, если не для всех реально используемых генераторов предполагается, что оба эти условия выполняются, но это предположение не доказано математически, а базируется на здравом смысле и опыте (это относится, например, к RC4 и HC-128).

С другой стороны, существует изящная теория, позволяющая связать задачу генерирования псевдослучайных чисел с другими разделами криптографии и с теорией сложности. В рамках этой теории «небольшое время вычислений» означает «вычисления, проводимые за полиномиальное время». Изложение этой теории может быть найдено, например, в [61], а здесь мы только отметим ряд основных результатов. Во-первых, показано, что существуют генераторы псевдослучайных чисел, чьи последовательности не отличимы от случайных за полиномиальное время любым тестом. Это, в частности, означает, что такие последовательности можно использовать как «истинно» случайные при решении любой задачи, требующей не более чем полиномиальное время вычислений. Во-вторых, показано что оба требования, предъявляемые к генерируемым последовательностям (статистическая неотличимость от абсолютно случайной последовательности и непредсказуемость), эквивалентны. В-третьих, установлена связь между генераторами псевдослучайных чисел и односторонними функциями, которая, в частности, позволяет строить «доказуемо надежные» генераторы на основе односторонних функций. Кавычки здесь означают, что доказательства справедливы при некоторых условиях, типичных для теории сложности алгоритмов, например, базируются на недоказанных предположениях об односторонности некоторых функций.

Мы остановимся на описании нескольких генераторов, базирующихся на односторонних функциях. Прежде всего отметим, что два класса таких генераторов описаны ранее в разд. 8.4. Действительно,

режимы OFB и CTR описаны в общем виде и могут применяться для любых блоковых шифров, которые, в свою очередь, могут рассматриваться как односторонние функции. Другие конструкции генераторов базируются на теоретико-числовых односторонних функциях, знакомых нам по предыдущим разделам. В качестве примера мы рассмотрим генератор, базирующийся на системе RSA.

Параметры генератора — два больших простых числа  $P$  и  $Q$  ( $P \neq Q$ ), их произведение  $N = PQ$  и число  $e > 1$ , взаимнопростое с  $(P-1)(Q-1)$ . Семя  $x_0$  — случайно выбранное число из диапазона  $(1, N-1)$ . Генератор формирует последовательность бит  $z_1, z_2, \dots, z_k$  по следующей схеме:

$$\begin{aligned} x_i &\leftarrow x_{i-1}^e \bmod N, \\ z_i &\leftarrow \text{младший бит } x_i, \quad i = 1, 2, \dots, k. \end{aligned}$$

Отметим, что число  $e$  (при надлежащем выборе  $P$  и  $Q$ ) можно взять равным трем, что упрощает операцию возведения в степень.

Целый ряд других генераторов, базирующихся на теоретико-числовых односторонних функциях описан в литературе; см., например, [61, 68].

## 10.4. Тесты для проверки генераторов случайных и псевдослучайных чисел

Случайные и псевдослучайные числа находят самое широкое применение не только в криптографических системах, но и в вычислительных методах, и при имитационном моделировании. Это делает актуальной задачу построения эффективных статистических тестов, предназначенных для выявления возможных отклонений от случайности. Так, Национальный институт стандартов и технологий США (NIST) провел исследование известных статистических тестов для проверки случайных и псевдослучайных чисел, результаты которого вместе с рекомендациями по практическому применению опубликованы в [76]. В ходе этого исследования были выделены 16 методов, которые NIST рекомендует для применения в системах криптографии.

В серии статей [21, 23, 82] авторы, во-первых, показали теоретически и экспериментально, что эффективные тесты могут быть построены на основе так называемых архиваторов и, во-вторых, пред-

ложили три теста, мощность которых существенно выше, чем у всех методов, рекомендованных NIST. В этом разделе мы опишем один из этих трех тестов.

Предлагаемый статистический тест базируется на конструкции адаптивного кода «стопка книг», предложенного в [16] (позднее этот код был переоткрыт и в англоязычной литературе получил название «move-to-front»).

Для описания теста нам понадобятся некоторые определения. Пусть источник порождает буквы из алфавита  $A = \{a_1, a_2, \dots, a_S\}$ , и требуется по выборке  $x_1, x_2, \dots, x_n$  проверить гипотезу

$$H_0 : p(a_1) = p(a_2) = \dots = p(a_S) = 1/S,$$

против альтернативной гипотезы  $H_1$ , являющейся отрицанием  $H_0$ .

При тестировании по предлагаемому методу буквы алфавита  $A$  упорядочены (и занумерованы в соответствии с этим порядком от 1 до  $S$ ), причем этот порядок меняется после анализа каждого выборочного значения  $x_t$ , следующим образом: буква  $x_t$ , которую мы обозначим через  $a$  ( $x_t = a$ ), получает номер 1, номера тех букв, которые были меньше, чем номер  $a$ , увеличиваются на 1, а у остальных букв номера не меняются. Для более формального описания этого преобразования обозначим через  $\nu^t(a)$  номер буквы  $a \in A$  после анализа  $x_1, x_2, \dots, x_{t-1}$ , и пусть начальный порядок  $\nu^1(\cdot)$  на буквах  $A$  задан произвольно. Тогда нумерация после анализа  $x_t$  определяется следующим образом:

$$\nu^{t+1}(a) = \begin{cases} 1, & \text{если } x_t = a, \\ \nu^t(a) + 1, & \text{если } \nu^t(a) < \nu^t(x_t), \\ \nu^t(a), & \text{если } \nu^t(a) > \nu^t(x_t). \end{cases} \quad (10.2)$$

(Все происходит как в стопке книг, если считать, что номер книги совпадает с положением в стопке. Книга извлекается и кладется на верх. Ее номер становится первым; книги, которые первоначально были над ней, сдвигаются вниз, а остальные остаются на месте.)

Основная идея метода — подсчитывается не частота встречаемости букв в выборке  $x_1, x_2, \dots, x_n$ , а частота встречаемости номеров букв (при описанном упорядочивании). В том случае, когда выполнена гипотеза  $H_1$ , вероятность (и частота встречаемости в выборке) некоторых букв больше  $1/S$  и их номера, в среднем, будут меньше, чем у букв с меньшими вероятностями. (Другими словами, книги,

к которым обращаются чаще, проводят в верхней части стопки значительно большее время, чем остальные. И, следовательно, вероятность обнаружить требуемую книгу в верхней части стопки больше, чем в нижней.) Если же выполнена гипотеза  $H_0$ , то, очевидно, вероятность появления в выборке буквы с любым номером равна  $1/S$ .

При применении описываемого теста множество всех номеров  $\{1, \dots, S\}$  заранее, до анализа выборки, разбивается на  $r$ ,  $r > 1$ , непересекающихся частей  $A_1 = \{1, 2, \dots, k_1\}$ ,  $A_2 = \{k_1 + 1, \dots, k_2\}$ ,  $\dots$ ,  $A_r = \{k_{r-1} + 1, \dots, k_r\}$ . Затем по выборке  $x_1, x_2, \dots, x_n$  подсчитывается количество номеров  $\nu^t(x_t)$ , принадлежащих подмножеству  $A_j$ , которое мы обозначим через  $n_j$ ,  $j = 1, \dots, r$ . При выполнении  $H_0$  вероятность того, что  $\nu^t(x_t)$  принадлежит множеству  $A_j$ , пропорциональна количеству элементов этого подмножества, т.е. равна  $|A_j|/S$ . Затем по значениям  $n_1, \dots, n_r$  проверяется гипотеза

$$H_0^* : P\{\nu^t(x_t) \in A_j\} = |A_j|/S \quad (10.3)$$

против альтернативной гипотезы  $H_1^* = \neg H_0^*$ . Очевидно, что при выполнении исходной гипотезы  $H_0$  выполняется и  $H_0^*$  и, наоборот, при выполнении гипотезы  $H_1^*$  выполняется  $H_1$ . Поэтому применение описанного критерия корректно.

При проверке гипотез используется хорошо известный в статистике критерий хи-квадрат ( $\chi^2$ ), описание которого может быть найдено, например, в [13]. При применении критерия  $\chi^2$  вычисляется величина (статистика)

$$x^2 = \sum_{j=1}^r \frac{(n_j - nP_j^0)^2}{nP_j^0}, \quad (10.4)$$

где  $P_j^0 = |A_j|/S$ , см. (10.3). Известно, что распределение случайной величины  $x^2$  асимптотически приближается к распределению  $\chi^2$  с  $(r - 1)$  степенью свободы при выполнении  $H_0$ .

Мы не описываем какой-либо алгоритм выбора количества подмножеств  $A_j$  и их величины, а предлагаем определять эти характеристики экспериментально. Дело в том, что при тестировании генераторов случайных и псевдослучайных чисел, как и в некоторых других задачах, возможно проведение специальных экспериментов для определения значений этих величин, а затем проведение проверки гипотез по независимым данным (или по участкам последова-

тельности псевдослучайных чисел, не использовавшимся при подборе указанных значений). Такие эксперименты направлены на поиск значений параметров, позволяющих выявить отклонения от случайности (т.е. от  $H_0$ ) при приемлемых объемах выборки (или за приемлемое время вычислений).

В качестве примера покажем, что дает описанный метод тестирования применительно к генераторам, предложенным в [13]. Д. Кнут [13] применяет различные статистические тесты к нескольким генераторам псевдослучайных чисел, которые он обозначил буквами  $B$ ,  $C$ ,  $D$ ,  $E$ ,  $F$ . Генераторы  $B-E$  задаются так называемыми линейными конгруэнтными последовательностями по формуле

$$x_{i+1} = (ax_i + b) \bmod m, \quad (10.5)$$

где  $a$ ,  $b$ ,  $m$  и  $x_0$  — параметры метода, а генератор  $F$  основан на так называемом методе Фибоначчи (полное описание всех генераторов дано в [13]).

Все эти генераторы предназначены для получения равномерно распределенных целых чисел из множества  $\{0, \dots, m-1\}$ , где  $m$  — параметр метода, см. (10.5). Известно, что младшие знаки порождаемых по (10.5) чисел часто далеки от абсолютно случайных, поэтому обычно рекомендуется использовать только старшие знаки в качестве случайных чисел [13]. Следуя этой рекомендации, из порождаемых генератором значений выделялся старший бит, или старший байт (варианты, обозначаемые в дальнейшем  $R_1$  и  $R_8$  соответственно). Точнее, в режиме  $R_1$  для четного  $m$  старший бит  $z_i$  вычислялся по формуле

$$z_i = \begin{cases} 0, & \text{если } x_i < m/2, \\ 1, & \text{если } x_i \geq m/2, \end{cases}$$

а для нечетного — как

$$z_i = \begin{cases} 0, & \text{если } x_i < (m-1)/2, \\ 1, & \text{если } x_i > (m-1)/2, \\ \Lambda, & \text{если } x_i = (m-1)/2, \end{cases}$$

где  $\Lambda$  — пустое слово.

В режиме  $R_8$  восьмibитовое слово  $\hat{z}_i$  «извлекается» из  $x_i$  по формуле

$$\hat{z}_i = \begin{cases} \lfloor 256x_i/m^* \rfloor, & \text{если } x_i < m^*, \\ \Lambda, & \text{если } x_i \geq m^*, \end{cases}$$

где  $m^* = 256[m/256]$ , а целое число  $[256x_1/m^*]$  записано как 8-битовое слово.

Последовательность  $z_i \in \{0, 1\}$  разбивалась на блоки длины  $s$  и при тестировании рассматривалась как выборка из алфавита размера  $S = 2^s$ , состоящего из всех двоичных слов длины  $s$ . Множество всех позиций в «стопке книг» разбивалось или на два подмножества  $A_1 = \{a_1, \dots, a_{k_1}\}$ ,  $A_2 = \{a_{k_1+1}, \dots, a_S\}$ , или на три:  $A_1 = \{a_1, \dots, a_{k_1}\}$ ,  $A_2 = \{a_{k_1+1}, \dots, a_{k_2}\}$ ,  $A_3 = \{a_{k_2+1}, \dots, a_S\}$ .

В табл. 10.1 приведены данные по тестированию датчиков, описанных в книге Д. Кнута [13]. При проведении экспериментов каждый генератор тестировался в режиме  $R_1$  при разных длинах выборки. Если удавалось найти длину выборки, при которой генерируемые последовательности отличались от случайных (при применении «стопки книг»), все вычисления повторялись при этой длине выборки по другим 100 последовательностям, ранее не использовавшимся для тестирования. При этом вычислялись величины  $Q_\alpha$ , равные количеству тех случаев, когда вычисленное по выборке значение  $x^2$  (см. (10.4)) превосходило значение квантиля уровня  $\alpha$  распределения  $\chi^2$  с соответствующим числом степеней свободы для двух значений  $\alpha = 0,5$  и  $\alpha = 0,95$ . Если же не удавалось найти отклонений на исследуемых длинах выборок, то все вычисления повторялись в режиме  $R_8$ .

Д. Кнут [13] протестировал указанные датчики другими методами и пришел к выводу, что генераторы  $D$  и особенно  $E$  и  $F$  должны быть забракованы,  $B$  прошел испытания удовлетворительно, а  $C$  «находится на грани». Полученные нами данные хорошо согласуются с этими результатами. Действительно, применение нового теста показывает, что генератор  $F$  должен быть забракован уже при длине выборки 240 бит: при ста испытаниях вычисленное по (10.4) значение  $x^2$  превосходит квантили порядка 0,95 и 0,5 соответственно 80 и 83 раза из 100 (тогда как для «идеальных» случайных бит это должно происходить, в среднем, в 5 и в 50 из 100 случаев). Генераторы  $C$  и  $D$  также бракуются описанным тестом, правда при больших длинах выборки. Это можно показать строго, используя, скажем, тот же критерий  $\chi^2$ . Как мы отмечали, при абсолютно случайных числах вероятность «попадания» в колонку  $Q_{0,95}$  должна быть равна 0,05. Прямое вычисление показывает, что гипотеза о равенстве вероятности попадания 0,05 должна быть отвергнута по критерию  $\chi^2$  при уровне значимости 0,01, если число попаданий превышает 11 из

100. Из табл. 10.1 видно, что во всех случаях гипотеза об абсолютной случайности чисел, порождаемых генераторами  $C-F$  должна быть отвергнута при уровне значимости 0,01, т.к.  $Q_{0,95}$  во всех случаях больше 11. Интересно отметить, что датчик  $C$  не бракуется ни одним статистическим тестом из [13]. Однако описанный нами тест забраковал этот датчик даже в режиме  $R_1$ . Таким образом, предложенный тест оказался более мощным, чем методы, традиционно используемые для тестирования генераторов случайных чисел.

Таблица 10.1. Тестирование генераторов из [13]

Генератор	Режим	$s$	Длина выборки, бит	$k_1$	$k_2$	$Q_{0,5}$	$Q_{0,95}$
$B$	$R_8$	24	1680000	$2^{17}$	-	52	4
$C$	$R_1$	24	7920000	$2^{17}$	-	70	17
$D$	$R_1$	16	160000	$2^{10}$	$2^{14}$	74	16
$E$	$R_1$	12	12000	$2^3$	$2^5$	99	97
$F$	$R_1$	8	240	$2^1$	$2^2$	83	80

Как уже отмечалось, в [21, 82] тест «стопка книг» сравнивался с методами, рекомендованными Национальным институтом стандартов и технологий США (NIST). Оказалось, что его мощность существенно выше, чем у всех 16 методов, рекомендованных NIST; подробности можно найти в [82].

## 10.5. Статистическая атака на блочные шифры

Как отмечалось в главе 8, блочные шифры с секретным ключом находят самое широкое применение в системах защиты передаваемой и хранимой информации, что делает актуальными как задачи построения надежных блочных шифров, так и поиск эффективных криптологических атак на эти шифры (т.е. методов определения секретного ключа шифра на основе экспериментов с зашифрованными сообщениями). Исследования в этих областях ведутся параллельно и часто одними и теми же специалистами и, как правило, изобретение новой атаки приводит к появлению шифров, к ней устойчивых. Отметим сразу, что для криптографии представляют интерес атаки, которые менее трудоемки, чем метод прямого перебора ключей.

Мы видели в предыдущей главе, что блоковые шифры описываются как функция, определенная на множестве всех двоичных слов длины  $(n+k)$  и принимающая значения в множестве двоичных слов длины  $n$ , где  $n$  — размер блока, а  $k$  — длина (секретного) ключа. Мы видели, что в современных шифрах длина блока обычно составляет 128 или 64 бита, а длина ключа принимает значения от нескольких десятков до нескольких тысяч бит. Например, у шифра AES (Rijndael) длина блока  $n = 128$  бит, а длина ключа может принимать три значения — 128, 196 и 256 бит. У других популярных шифров RC5 и RC6, предложенных Р. Ривестом, длина блока может быть 32, 64 или 128 бит, а длина ключа в разных вариантах принимает значения от 64 до нескольких тысяч бит.

Как было показано в главе 8, процесс шифрования в современных блоковых шифрах обычно разбивается на последовательность сравнительно простых этапов, называемых раундами. В ходе каждого нового раунда проводится шифрование данных, полученных на предыдущем этапе с так называемым ключом раунда. В RC5, RC6 и многих других шифрах количество раундов является параметром и часто криptoаналитики исследуют стойкость шифров как функцию числа раундов. Одна из целей такого анализа — нахождение числа раундов, гарантирующих высокую надежность шифра. Таким образом, схематично процесс шифрования можно представить как цепочку «элементарных» этапов (или раундов) шифрования

$$x_1 = E_1(x_0, k_1), \quad x_2 = E_2(x_1, k_2), \dots, \quad x_r = E_r(x_{r-1}, k_r), \quad (10.6)$$

где  $x_0$  — исходное  $n$ -битовое слово, которое необходимо зашифровать,  $E_i$  — операция (функция) шифрования на  $i$ -м этапе,  $k_i$  — ключ, используемый на  $i$ -м этапе,  $x_i$  —  $n$ -битовое слово, являющееся «выходом»  $i$ -го этапа и «входом»  $(i+1)$ -го, и, наконец,  $x_r$  — результат шифрования.

Как отмечалось при изучении блоковых шифров, для них известны *специальные* виды атак, такие как дифференциальный и линейный криptoанализ и их разновидности. Для иллюстрации связи случайных чисел с многими задачами криптографии в данном разделе мы опишем *универсальную* атаку на блоковые шифры, названную градиентной статистической атакой, и в качестве примера исследуем возможность ее применения для криptoанализа шифра RC5. Приведенные экспериментальные данные позволяют сделать вывод о том, что эта атака может быть применена, и что для некоторых режи-

мов шифра ее трудоемкость может быть существенно меньше, чем у прямого перебора ключей.

Описываемый нами метод относится к классу атак с выбираемым текстом (*chosen plaintext attack*), см. главы 1 и 8. При реализации этой атаки криптоаналитик может подавать на вход шифра любой текст и анализировать полученное зашифрованное сообщение. Цель атаки — нахождение секретного ключа, причем при этом предполагается, что криптоаналитик знает все характеристики шифра, кроме этого ключа. Такие атаки представляют практический интерес и, как отмечалось в главе 8, современные блоковые шифры должны быть стойки к ним.

У большинства современных шифров начальный ключ  $K$  длиной в  $|K|$  бит преобразуется в последовательность так называемых раундовых ключей  $k_1, k_2, \dots, k_r$ , которые используются последовательно для шифрования на разных этапах, см. (10.6). В разных шифрах эта процедура осуществляется по разному; причем это зависит не только от шифра, но и от значений длин блока, ключа и числа раундов, которые для многих шифров являются параметрами. Например, для шифра RC5 длина блока может принимать значения 32, 64 или 128 бит, количество раундов может быть любым целым числом, а длина ключа должна быть кратна 8 и может принимать любое значение, начиная с 8 бит. Отметим, что значения  $n = 64$ ,  $r = 12$ ,  $|K| = 128$  рекомендованы разработчиками и широко исследованы. Часто рассматриваются и схемы, в которых длина ключа  $K$  равна суммарной длине ключей раундов,  $|K| = \sum_{i=1}^r |k_i|$ .

Расшифрование проводится по схеме, обратной к шифрованию (10.6):

$$x_{r-1} = D_r(x_r, k_r), \quad x_{r-2} = D_{r-1}(x_{r-1}, k_{r-1}), \dots, \quad x_0 = D_1(x_1, k_1), \quad (10.7)$$

где используются те же ключи раундов, а операции  $D_i$  обратны этапам шифрования  $E_i$ .

Оценим трудоемкость атаки путем полного перебора ключей. Для ее проведения достаточно иметь одно зашифрованное сообщение (двоичное слово), длина которого не меньше длины ключа. Затем необходимо пытаться дешифровать это зашифрованное сообщение, последовательно перебирая все возможные ключи в каком-либо порядке и сравнивая полученный результат с исходным, незашифрованным, текстом; совпадение означает, что неизвестный ключ най-

ден. Обычно предполагается, что ключ принимает любое значение из множества всех двоичных слов длины  $|K|$  с вероятностью  $2^{-|K|}$ , поэтому среднее значение числа перебираемых ключей равно  $2^{|K|}/2$ .

Одно из требований, предъявляемых к блоковым шифрам, можно сформулировать следующим образом: любое зашифрованное сообщение должно быть неотличимо от абсолютно случайной последовательности. В частности, все шифры, принимавшие участие в конкурсе на шифр 21-го века, проводившемся в США в 1999–2001 годах, проверялись на выполнение этого условия. Мы не будем останавливаться на логическом анализе этого требования (которое, в некотором смысле, вообще не выполнимо), а приведем пример, поясняющий его смысл. Для этого мы определим  $n$ -битовое слово  $\alpha_i$  как двоичную запись числа  $i$ ,  $i = 0, 1, 2, \dots, 2^n - 1$ , где, как и ранее,  $n$  — длина блока рассматриваемого шифра (т.е.  $\alpha_0$  состоит из  $n$ -битовой цепочки нулей,  $\alpha_1$  из  $(n-1)$  нуля и единицы,  $\alpha_2$  — из  $(n-2)$  нулей, после которых идет последовательность 10 и т.д.). От современного блокового шифра требуется, чтобы при любом значении ключа последовательность  $n$ -битовых слов  $E(\alpha_0), E(\alpha_1), E(\alpha_2), \dots$ , рассматриваемая как двоичная последовательность, была статистически неотличима от случайной. (Здесь  $E(\alpha_i)$  означает зашифрованное слово  $\alpha_i$ .) Это требование, в частности, позволяет использовать блоковые шифры как генераторы псевдослучайных чисел, как было показано в разд. 8.4.

Перейдем теперь к описанию статистической атаки на блоковые шифры, у которых шифрование и расшифрование разбивается на последовательность раундов (10.6) и (10.7), начав с неформального предварительного рассмотрения. При этом мы будем использовать совершенно не строгие термины «более» и «менее» случайные последовательности, понимая под этим, что некоторая последовательность более случайна, чем другая, если отклонения от случайности у первой достоверно выявляются при большей длине, чем у второй. (При этом предполагается, что используется некоторый статистический тест при одном и том же уровне значимости. Другое «определение» более случайной последовательности — величина статистики критерия для этой последовательности меньше, чем для менее случайной.) Предположим, что на вход шифра, ключ которого неизвестен, подаются последовательно слова  $\alpha_0, \alpha_1, \alpha_2, \dots$ . Очевидно,

эта последовательность очень неслучайна. Последовательность

$$E_1(\alpha_0, k_1), E_1(\alpha_1, k_1), E_1(\alpha_2, k_1), \dots$$

после первого раунда шифрования, которую мы обозначим через  $\beta_0, \beta_1, \dots$ , более случайна, чем исходная; получаемая после второго раунда последовательность

$$E_2(\beta_0, k_2), E_2(\beta_1, k_2), E_2(\beta_2, k_2), \dots$$

еще более случайна и т.д. Наконец, полученная после последнего раунда последовательность  $\omega_0, \omega_1, \omega_2, \dots$  более случайна, чем предыдущая. Это неформальное утверждение подтверждается экспериментально в приведенных ниже данных для шифра RC5 и в довольно многочисленных работах по анализу блочных шифров. Объяснение этого факта довольно очевидно: шифрование на каждом раунде приводит к «перемешиванию» и, тем самым, повышает «случайность» шифруемых данных. Отметим и очевидное следствие — при расшифровании последовательности  $\omega_0, \omega_1, \omega_2, \dots$  по схеме (10.7) случайность получаемых данных последовательно уменьшается. Это, конечно, справедливо только в том случае, когда при расшифровании используются «истинные» ключи раундов. Если же при дешифровании, скажем,  $x_{r-1} = D_r(x_r, k_r)$  вместо истинного ключа  $k_r$ , используется какое-либо другое слово  $k_r^*$  той же длины, то эффект преобразования  $D_r(x_r, k_r^*)$  будет таков же, как при шифровании — выходная последовательность станет более случайна, чем входная. Это важное для нас наблюдение в общем виде состоит в следующем: при дешифровании на  $j$ -м раунде при использовании «неправильного» ключа  $k_j^*$  (вместо «правильного» ключа  $k_j$ ) случайность выходной последовательности возрастает, тогда как при использовании правильного  $k_j$  — убывает. На этом наблюдении и базируется предлагаемая атака, которую мы теперь можем схематично описать.

Вначале уточним формулировку задачи. Дан шифр, для которого шифрование и дешифрование проводятся по схемам (10.6), (10.7) соответственно. Предполагается, что все параметры шифра, кроме ключа  $K$ , известны. Цель атаки — найти неизвестные ключи раундов  $k_1, k_2, \dots, k_r$ , где, как и ранее,  $r$  — число раундов (что эквивалентно нахождению  $K$ , т.к. дает возможность дешифровать любое сообщение, зашифрованное с этим ключом).

При проведении описываемой атаки сначала на вход шифра подается «простая» последовательность из  $m_r$   $n$ -битовых слов (на-

пример, вышеописанная  $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{m_r}$ ), где  $m_r$  — параметр метода. Обозначим полученную на выходе зашифрованную последовательность через  $\omega_0, \omega_1, \omega_2, \dots, \omega_{m_r}$ . Предполагается, что используется некоторая количественная мера случайности, которую мы обозначим через  $\gamma(w)$ , где  $w$  — двоичная последовательность, причем, чем больше  $\gamma(w)$ , тем последовательность менее случайна (в дальнейшем в качестве такой меры будет использоваться статистика  $x^2$ ).

После этого для всех возможных значений ключа  $r$ -го раунда  $k_r$  поочередно вычисляем последовательность  $\Gamma_r(u)$ , определяемую как

$$\Gamma_r(u) = D_r(\omega_0, u), D_r(\omega_1, u), D_r(\omega_2, u), \dots, D_r(\omega_{m_r}, u), \quad (10.8)$$

где  $u \in \{0, 1\}^{|k_r|}$ , и оцениваем степень ее случайности. Затем находим такое значение  $u^*$ , для которого статистика  $\gamma(\Gamma_r(u^*))$  максимальна среди всех значений  $\gamma(\Gamma_r(u))$ ,  $u \in \{0, 1\}^{|k_r|}$ , и полагаем, что (неизвестный) ключ  $r$ -го раунда равен  $u^*$ :  $k_r = u^*$ . Отметим сразу, что количество операций дешифрования на этом этапе пропорционально  $2^{|k_r|} m_r$ .

Затем повторяем аналогичные вычисления для поиска ключа  $(r-1)$ -го раунда ( $k_{r-1}$ ), используя в качестве исходной последовательность  $\Gamma_r(k_r) = \Gamma_r(u^*)$ , см. (10.8). Точнее, вычисляем последовательность

$$\Gamma_{r-1}(u) = D_{r-1}(D_r(\omega_0, k_r), u), D_{r-1}(D_r(\omega_1, k_r), u), \dots, \quad (10.9)$$

где теперь  $u \in \{0, 1\}^{|k_{r-1}|}$  и оцениваем случайность этой последовательности. Мы считаем, что количество  $n$ -битовых слов в этой последовательности, которое мы обозначим через  $m_{r-1}$ , не превосходит  $m_r$  (в противном случае можно вычислить недостающие слова, хотя, как будет видно из последующих данных,  $m_{r-1}$  будет обычно меньше  $m_r$ , т.к. последовательность  $\Gamma_{r-1}(u)$  менее случайна, чем  $\Gamma_r(u)$ ). Слово  $u^{**}$ , минимизирующее случайность последовательности  $\Gamma_{r-1}(u)$  и будет значением ключа  $(r-1)$ -го раунда. Отметим, что на этом этапе количество операций дешифрования пропорционально  $2^{|k_{r-1}|} m_{r-1}$ .

Повторяя описанные вычисления последовательно, мы найдем значения ключей раундов  $k_{r-1}, k_{r-2}, k_{r-3}, \dots, k_1$ . Суммарное количество операций при нахождении всех ключей раундов пропорционально  $\sum_{i=1}^r 2^{|k_i|} m_i$  или в типичном случае, когда длины ключей

раундов равны ( $|k_i| = |k|$ ), количество операций пропорционально величине  $rm_{\max}2^{|k|}$ , тогда как для прямого перебора это величина порядка  $2^{|K|}$  (где  $m_{\max} = \max_{i=1,\dots,r} m_i$  и  $K$  — ключ шифра). Эта разница в показателях степени и определяет область применимости предлагаемой атаки: если  $rm_{\max} < 2^{|K|-|k|}$ , то количество операций у предлагаемого метода меньше, чем у полного перебора ключей.

Теперь обсудим возможные модификации, параметры и варианты описанного метода. Мы описали идею метода в «чистом» виде, а здесь остановимся на возможных вариантах его реализации.

Во-первых, мера случайности  $\gamma(\cdot)$  является параметром метода, причем можно использовать не только различные меры для разных шифров, но и для разных раундов. Как указано выше, любой статистический тест, который применим для проверки основной гипотезы ( $H_0$ ) о том, что двоичная последовательность порождается бернуlliевским источником с равными вероятностями для нуля и единицы, против альтернативной гипотезы  $H_1$ , являющейся отрицанием  $H_0$ , может быть использован для этой цели. При этом  $\gamma(\cdot)$  может быть равна величине статистики критерия (скажем, величине  $x^2$ , используемой в критерии хи-квадрат; именно такая мера будет использована в нижеприведенных примерах.)

Во-вторых, в отличие от описанного выше варианта, при поиске ключа  $j$ -го раунда можно выбирать не один истинный ключ, а несколько «подозрительных» значений  $u$ , т.е. таких слов, у которых мера случайности  $\gamma(\Gamma_j(u))$  минимальна (среди  $u \in \{0,1\}^{|k_j|}$ ). Кроме того, при поиске простых последовательностей и ключей раундов естественно использовать последовательные методы, аналогичные последовательным критериям в математической статистике.

В третьих, начальная «очень неслучайная» последовательность  $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_m$  может выбираться различными способами. Например, кажется разумным выбирать последовательность, в которой соседние слова  $(\alpha_i, \alpha_{i+1})$  не только содержат много одинаковых символов, но и отличаются только одним знаком (так называемый код Грея). Наконец, часть двоичных символов в словах последовательности  $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_m$ , могут выбираться случайно, а оставшиеся полагаться равными нулю и т.д.

Последняя модификация связана с тем фактом, что у многих современных шифров при большом числе раундов даже «очень» неслучайная последовательность после шифрования статистически не от-

личима от случайной (при использовании известных статистических тестов и при приемлемом времени вычислений). Пусть, например, шифр использует  $r$  раундов и для некоторой «простой» начальной последовательности  $\alpha^0 = \alpha_0^0, \alpha_1^0, \alpha_2^0, \dots, \alpha_m^0$  последовательности

$$\begin{aligned}\alpha^1 &= E_1(\alpha_0^0, k_1), E_1(\alpha_1^0, k_1), E_1(\alpha_2^0, k_1), \dots, E_1(\alpha_m^0, k_1) \\ \alpha^2 &= E_2(\alpha_0^1, k_2), E_2(\alpha_1^1, k_2), E_2(\alpha_2^1, k_2), \dots, E_2(\alpha_m^1, k_2) \\ &\dots \\ \alpha^d &= E_d(\alpha_0^{d-1}, k_d), E_d(\alpha_1^{d-1}, k_d), \dots, E_d(\alpha_m^{d-1}, k_d)\end{aligned}$$

неслучайны при всех ключах раундов  $k_1, \dots, k_d$ ,  $d < r$ . Тогда описанная выше атака может быть модифицирована следующим образом: для каждого набора ключей  $k_{d+1}, \dots, k_r$ , соответствующих раундам  $d+1, \dots, r$ , повторяя описанную выше процедуру нахождения неизвестных ключей  $k_1, \dots, k_d$ . Другими словами, ключи  $k_{d+1}, \dots, k_r$  находим полным перебором, а  $k_1, \dots, k_d$  — по описанному выше методу. Для проведения такой комбинированной атаки потребуется количество операций

$$m2^{\sum_{j=d+1}^r |k_j|} \sum_{j=1}^d 2^{|k_j|},$$

что при некоторых соотношениях параметров может быть меньше, чем количество операций, необходимое при прямом переборе всех ключей.

В качестве объекта экспериментального исследования предложенной атаки был взят шифр RC5. В этом шифре, как видно из его описания, каждый раунд может быть легко разделен на две части, называемые полураундами. Мы в дальнейшем часто будем использовать выражения типа «3,5 раунда» вместо «7 полураундов» (эта терминология является общепринятой в работах, касающихся RC5, RC6 и ряда других шифров). Описание экспериментального исследования мы начнем с анализа степени случайности зашифрованных сообщений в зависимости от числа раундов шифра.

Первый вопрос, который исследовался экспериментально<sup>1</sup>, касался возможности различия зашифрованных при помощи RC5

<sup>1</sup>Вычисления проводились на суперкомпьютерах Института вычислительных технологий СО РАН и Новосибирского государственного университета.

«простых», явно не случайных, последовательностей при разном числе (полу)раундов. Для этого мы использовали в качестве исходной вышеупомянутую последовательность  $\alpha_i$ ,  $i = 0, 1, \dots$ , где  $\alpha_i$  — запись числа  $i$  в двоичной системе счисления, под которую отводится 64 бита. (Напомним, что мы рассматриваем RC5 с длиной шифруемого блока 64 бита.) Во всех случаях указанная последовательность зашифровывалась при помощи указанного шифра с заданным количеством полураундов и по полученной последовательности проверялась гипотеза  $H_0$  о том, что двоичная последовательность порождается бернуlliевским источником с равными вероятностями нуля и единицы, против альтернативной гипотезы  $H_1$ , являющейся отрицанием  $H_0$ . В дальнейшем, для того чтобы избегать длинных повторов, мы будем называть эту задачу «гипотезой о случайности». Для проверки этой статистической гипотезы использовался описанный в [23] тест «адаптивный хи-квадрат».

В табл. 10.2 приведены данные по проверке гипотезы о случайности при использовании адаптивного критерия хи-квадрат для RC5 с разным количеством раундов.

Таблица 10.2. Количество последовательностей (из 100), для которых гипотеза о случайности была отвергнута

Номер ключа	Число раундов $r$ и длина последовательностей $t$				
	$r = 1$ $t = 2^{18}$	$r = 1,5$ $t = 2^{18}$	$r = 2$ $t = 2^{18}$	$r = 2,5$ $t = 2^{20}$	$r = 3$ $t = 2^{20}$
1	100	63	64	51	52
2	100	100	100	74	70
3	100	61	61	17	17
4	100	81	78	62	64
5	100	100	100	65	6
6	100	85	86	12	9
7	100	100	100	11	8
8	100	98	99	99	99
9	100	80	79	14	14
10	100	100	100	7	5

Все вычисления проводились для 10 случайно выбранных ключей и повторялись 100 раз при шифровании 100 следующих после-

довательностей слов длины  $t$  :

$$\alpha_0, \alpha_1, \dots, \alpha_{t-1}, \quad \alpha_t, \alpha_{t+1}, \dots, \alpha_{2t-1}, \quad \dots, \\ \dots, \alpha_{99t}, \alpha_{99t+1}, \dots, \alpha_{100t-1}. \quad (10.10)$$

В табл. 10.2 указано количество случаев, когда гипотеза о случайности отвергалась при уровне значимости 0,001. Например, мы видим из таблицы, что гипотеза о случайности была отвергнута 100 раз из 100 при использовании первого (случайно выбранного) ключа и при длине последовательности  $t = 2^{18}$  слов. Таким образом, мы видим из этой таблицы, что зашифрованные последовательности явно не случайны, т.к. в противном случае в среднем гипотеза отвергалась бы приблизительно в  $0,001 \cdot 100 = 0,1$  случаях из 100.

Для большего числа раундов вычисления проводились с меньшим числом вариантов (или повторностей), т.к. в этом случае требовались последовательности большей длины и соответственно большее время вычислений. Снова проверялась гипотеза о случайности для той же зашифрованной последовательности  $\alpha_0, \dots, \alpha_{t-1}$  с разными (случайно выбранными) ключами и разным числом раундов; результаты приведены в табл. 10.3. Мы видим, что зашифрованная

Таблица 10.3. Проверка гипотезы о случайности для большого числа раундов при уровне значимости 0,01

Число раундов	$t$	Количество тестов	Количество случаев, в которых гипотеза о случайности отвергнута
5	$2^{28}$	30	30
5,5	$2^{29}$	22	10
6	$2^{31}$	6	6
6,5	$2^{32}$	6	6
7	$2^{32}$	6	5
7,5	$2^{33}$	3	3
8	$2^{37}$	3	2

последовательность  $\alpha_0, \alpha_1, \dots, \alpha_{t-1}$  довольно надежно отличается от случайной до восьмого раунда.

Как уже было сказано, главное предположение, без выполнения которого градиентная статистическая атака была бы невозможна, состоит в следующем: при дешифровании на любом раунде при

использовании «неправильного» ключа  $k^*$  (вместо «правильного» ключа  $k$ ) случайность выходной последовательности возрастает, тогда как при использовании «правильного»  $k$  — убывает. Выполнение этого предположения проверялось экспериментально по следующей схеме: для трех случайно выбранных ключей 100 вышеописанных сообщений (10.10) шифровались при помощи RC5 до  $j$ -го полураунда. Затем проводилось дешифрование на один полураунд с «истинным» ключом полураунда и с 10 случайно выбранными «неправильными» ключами, и для всех 11 последовательностей оценивалась степень случайности полученных после этого преобразования данных. Отметим, что, в соответствии в нашей гипотезой, разница в степени случайности последовательности, дешифрованной с «правильным» ключом и 10 других, дешифрованных со случайными ключами, должна соответствовать различиям в случайности, получаемым при шифровании в один дополнительный раунд. (Действительно, правильный ключ уменьшает случайность на полраунда, а неправильный — увеличивает случайность на полраунда.)

В табл. 10.4 приведены данные экспериментов для различного количества раундов ( $r$ ) при уровне значимости 0,001. Параметры теста и длины последовательностей  $t$  определялись в ходе предварительных экспериментов, проводимых по независимым данным, полученным с использованием других случайных ключей.

Поясним результаты первой серии опытов для 2,5 раундов. Оказалось, что среди 100 последовательностей, расшифрованных с правильным ключом полураунда, неслучайными признаны 54 из 100 (при уровне значимости 0,001), тогда как из 100 последовательностей дешифрованных с первым «неправильным» ключом признано неслучайными только 9 последовательностей, со вторым — 10, с третьим — 10 и т.д.; т.е. последовательности, дешифрованные с правильным ключом, менее случайны, чем последовательности, дешифрованные с неправильными ключами.

Мы видим, что данные, приведенные в таблицах 10.2—10.4, подтверждают предположения, выполнение которых необходимо для принципиальной возможности проведения градиентной статистической атаки: во-первых, случайность зашифрованной последовательности возрастает при увеличении числа полураундов и, во-вторых, случайность последовательности, дешифрованной с неправильным ключом полураунда, больше, чем у дешифрованной с правильным ключом.

**Таблица 10.4. Количество «неслучайных» последовательностей (из 100) при дешифровании с истинным и 10 случайными ключами ползураундов**

$r = 2,5, t = 2^8$

Серия	Истинный ключ	Случайные ключи									
		1	2	3	4	5	6	7	8	9	10
1	54	9	10	10	15	13	13	8	10	19	12
2	69	34	34	35	34	36	33	34	36	39	33
3	87	44	37	36	38	38	37	41	42	42	41

$r = 3, t = 2^{16}$

Серия	Истинный ключ	Случайные ключи									
		1	2	3	4	5	6	7	8	9	10
1	97	81	84	84	81	84	83	84	82	83	83
2	73	35	39	36	32	36	32	33	40	39	42
3	94	0	1	2	0	1	1	0	4	0	1

$r = 3,5, t = 2^{19}$

Серия	Истинный ключ	Случайные ключи									
		1	2	3	4	5	6	7	8	9	10
1	100	28	16	23	9	15	26	18	17	22	22
2	48	9	10	9	11	10	8	9	10	10	11
3	65	20	21	18	20	19	20	19	19	18	17

Таким образом, градиентная атака на шифр RC5 была реализована. Оказалось, что она позволяет достаточно надежно находить ключи четырех раундов при приемлемом времени вычислений.

### Описание шифра RC5

Ввиду того что градиентная атака рассматривалась по отношению к шифру RC5, мы для полноты изложения приводим описание этого шифра, достаточное для его компьютерной реализации.

Шифр RC5 является предшественником шифра RC6, рассмотренного нами в разд. 8.2. При описании RC5 будем использовать те же обозначения параметров шифра и операций, как ранее для RC6.

В RC5 пользователь задает размер слова  $w$  (16, 32 или 64 бита), количество раундов  $r$  и длину ключа  $l$ . Размер блока составляет два слова. Шифрование и расшифрование блока данных производится с

использованием одного и того же раундового ключа  $W$  длиной  $2r+2$  слова (нумерация слов с нуля), получаемого из секретного ключа  $K$ .

### Алгоритм 10.1. RC5: шифрование блока данных

**ВХОД:** Блок из двух слов  $(a, b)$ , раундовый ключ  $W$ .

**ВЫХОД:** Зашифрованный блок  $(a, b)$ .

1.  $a \leftarrow a + W_0, b \leftarrow b + W_1;$
2. FOR  $i = 1, 2, \dots, r$  DO
3.  $a \leftarrow ((a \oplus b) \leftrightarrow b) + W_{2i},$
4.  $b \leftarrow ((b \oplus a) \leftrightarrow a) + W_{2i+1};$
5. RETURN  $(a, b)$ .

Для расшифрования «прокручиваем» этот процесс в обратном порядке.

### Алгоритм 10.2. RC5: расшифрование блока данных

**ВХОД:** Блок из двух слов  $(a, b)$ , раундовый ключ  $W$ .

**ВЫХОД:** Дешифрованный блок  $(a, b)$ .

1. FOR  $i = r, r - 1, \dots, 1$  DO
2.  $b \leftarrow ((b - W_{2i+1}) \leftrightarrow a) \oplus a,$
3.  $a \leftarrow ((a - W_{2i}) \leftrightarrow b) \oplus b;$
4.  $b \leftarrow b - W_1, a \leftarrow a - W_0;$
5. RETURN  $(a, b)$ .

Алгоритм формирования раундового ключа практически полностью совпадает с соответствующим алгоритмом для RC6. Мы здесь его не приводим, т.к. для рассмотренной градиентной статистической атаки этот алгоритм не существенен.

## 10.6. Атака различия на потоковые шифры

В разделах, посвященных потоковым шифрам и генераторам псевдослучайных чисел (8.4 и 10.3 соответственно), мы говорили о том, что генерируемые в них псевдослучайные последовательности должны быть статистически неотличимы от последовательностей равновероятных и независимых нулей и единиц – такие последовательности

мы условились называть «абсолютно случайными». Атака различия (distinguishing attack) направлена на обнаружение отклонения генерируемых последовательностей от абсолютной случайности. Если такое отклонение удается обнаружить за приемлемое время, то соответствующий генератор или потоковый шифр считаются слабыми, непригодными для криптографических применений. Поэтому все известные и вновь разрабатываемые потоковые шифры (точнее, лежащие в их основе генераторы псевдослучайных чисел) подвергаются анализу на отклонение от случайности, для чего используются различные статистические тесты.

В данном разделе мы познакомим читателя с результатами атак различия [50], проведенных с помощью теста «стопка книг» (разд. 10.4) на шифры RC4 и ZK-Сгурт. Шифр RC4 был описан в разд. 8.4. Что касается другого шифра – ZK-Сгурт, то он был предложен в качестве участника в конкурсе потоковых шифров ECRYPT Stream Cipher Project (eSTREAM), проводимом Евросоюзом с 2004 по 2008 год в целях способствования разработкам эффективных и компактных потоковых шифров для широкого применения [53].

Ключевая последовательность шифра RC4 тестирулась на случайность для наиболее популярного режима работы шифра, когда длина слова в нем составляет 8 бит. Исследовались генерируемые последовательности для 100 случайно выбранных ключей. Для применения теста «стопка книг» последовательности разбивались на слова длиной 16 бит, сама «стопка» строилась из двух частей с размером верхней части 16 (в этом случае оставшиеся  $2^{16} - 16$  элементов можно не хранить в памяти). Были сгенерированы файлы различной длины для каждого из случайно выбранных ключей. К каждому файлу был применен тест с уровнем значимости  $\alpha = 0.05$ . Это означает, что если нулевая гипотеза  $H_0$  верна, то, в среднем, 5 файлов из 100 должны были быть признанными неслучайными. Полученные результаты приведены в табл. 10.5, числа во второй строке показывают количество файлов (из 100), признанных неслучайными.

Таблица 10.5. Тестирование RC4

Длина файла, бит	$2^{31}$	$2^{32}$	$2^{33}$	$2^{34}$	$2^{35}$	$2^{36}$	$2^{37}$	$2^{38}$	$2^{39}$
Количество неслучайных	6	12	17	23	37	74	95	99	100

Принимая во внимание, что в среднем только 5 файлов из 100 должны признаваться неслучайными, если последовательность абсолютно случайна, мы видим, что ключевые последовательности RC4 далеки от случайности, если их длина превышает  $2^{32}$  бит.

Ключевая последовательность шифра ZK-Crypt также исследовалась для 100 случайно выбранных ключей. Но в этот раз последовательность разбивалась на слова длиной 32 бита и верхняя часть «стопки» устанавливалась равной  $2^{16}$  элементов. Генерировались файлы различной длины для каждого ключа шифра и к ним применялся тест с уровнем значимости  $\alpha = 0.001$ . Таким образом, если бы гипотеза  $H_0$  была верна, то только один из 1000 файлов должен был признаваться неслучайным. Полученные результаты приведены в табл. 10.6. Мы видим, что ключевые последовательности, генерируемые ZK-Crypt, далеки от случайности уже при длине порядка  $2^{24}$  бит.

Т а б л и ц а 10.6. Т естирование ZK-Crypt

Длина файла, бит	$2^{24}$	$2^{26}$	$2^{28}$	$2^{30}$
Количество неслучайных	25	51	97	100

Результаты тестов, посланные в адрес оргкомитета eSTREAM, послужили весомым фактором в отклонении шифра ZK-Crypt. Справедливости ради отметим, что попытки найти отклонения с помощью теста «стопка книг» и других тестов, разработанных авторами, таких как «аддитивный хи-квадрат» [23], у других потоковых шифров – участников конкурса, не привели к успеху. В частности, не найдено никаких отклонений от случайности в последовательностях, генерируемых финалистами конкурса, см. [53]. Что касается шифра RC4, то он браковался и в ряде других работ, см., например, [44], но при значительно больших длинах генерируемых последовательностей порядка  $2^{55}$  бит, что подвластно только суперкомпьютерам.

# Глава 11. СТЕГАНОГРАФИЯ И СТЕГОАНАЛИЗ

## 11.1. Назначение и применение стеганографии в современных информационных технологиях

Стеганография — это наука о передаче некоторого «секретного» сообщения от отправителя к получателю так, что сам факт передачи секретного сообщения должен оставаться скрытым от наблюдателя. В англоязычной литературе для методов скрытой передачи информации часто используется название *data hiding*, что можно примерно перевести как «скрытие» или «сокрытие» данных. Условие скрытой передачи данных отличает стеганографию от криптографии, где сам факт передачи секретного (зашифрованного) сообщения не скрывается и известен Еве. Заметим, что это различие весьма существенно, т.к. во многих случаях сам факт обмена зашифрованными сообщениями является доказательством незаконной деятельности или, как минимум, вызывает подозрения. Принято считать, что криптография возникла практически одновременно с письменностью и, очевидно, не может существовать без нее. Стеганография же могла возникнуть до появления письменности и, судя по художественной литературе, могла довольно широко использоваться в течении столетий людьми, не знающими письменности. Пример такой стеганографической системы — наличие или отсутствие цветов на подоконнике у Алисы, сообщающей Бобу о возможности или невозможности свидания (мы по-прежнему будем называть участников, обменивающихся сообщениями, Алисой и Бобом, а наблюдателя — Евой).

Интересный пример реального применения стеганографии описан в [66]. Во время первой мировой войны немецкий шпион передал из Нью-Йорка следующее сообщение на английском: «Apparently neutral's protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets

and vegetable oils». В [66] утверждается, что это сообщение выглядит совершенно «безобидно», однако, если из каждого слова взять вторую букву, то получится следующий «шпионский» текст: «Pershing sails from NY June I. (Першинг отплывает из Нью-Йорка 1 июня)».

Первый текст называется в стеганографии сообщением-контейнером (или просто контейнером; по английски *cover*, или *covertext*, или *coverimage*, или *coveraudio* и т.п.), а второй текст — встраиваемой, или скрытой, или секретной информацией (*embedded message*, *hidden information*, *secret information*).

Другой известный многим пример использования стеганографии — акrostих. Так, в стихотворении Н. Гумилева «Анна Ахматова»

Ангел лег у края небосклона,  
 Наклонившись, удивлялся бездне;  
 Новый мир был синим и беззвездным.  
 Ад молчал, не слышалось ни стона.  
 Алой крови робкое биение,  
 Хрупких рук испуг и содроганье  
 Миру снов досталось в обладанье  
 Ангела святое отраженье.  
 Тесно в мире, пусть живет, мечтая  
 О любви, о свете и о тени,  
 В ужасе предвечном открывая  
 Азбуку своих же откровений

последовательность первых букв строк образует имя АННА АХМАТОВА.

Акrostихи и другие подобные им методы встраивания скрытой информации были довольно популярны в течение тысячелетий. Считается, что Эпихарм, древнегреческий филолог и драматург, живший в Сиракузах в V в. до н.э., был изобретателем акrostиха. Он использовал акrostих для защиты авторских прав, «пряча» свое имя в написанные им тексты. Хотя различные методы стеганографии были открыты и использовались еще до нашей эры в Китае, Греции и Риме, настоящий расцвет этой науки произошел в средние века в Европе. Тогда были изобретены методы встраивания скрытых сообщений в музыкальные произведения, картины, самые разнообразные тексты и т.п. При этом было придумано много технических средств (таких как невидимые чернила), а в 16–17 веках было опубликовано несколько книг, посвященных стеганографии. Большое внимание

привлекал и ее «антитип» — стегоанализ, предназначенный для обнаружения скрытой информации. Подробный и интересный обзор достижений стеганографии в «доинформационную» эпоху приведен в статье [73], авторы которой считают, что знакомство с историей изобретения, а часто и крушения, различных методов скрытия информации может оказаться полезным и разработчикам современных стеганографических методов.

В дальнейшем мы будем рассматривать традиционную схему использования стеганографии, когда предполагается, что Алиса отправляет Бобу «невинное» сообщение, в котором спрятан секретный текст, однако все рассматриваемые методы относятся и к ситуации, когда Алиса ничего не пересыпает, а размещает некоторые сведения так, что их могут увидеть (или просматривать) многие, в том числе и Боб. Например, Алиса пишет комментарий к статье в электронной газете, а Боб, как один из пользователей Интернета, просматривает это сообщение и извлекает из него скрытую информацию. Конечно, Алиса и Боб должны договориться заранее о том, какая газета используется, о ключе, позволяющем прочитать скрытые сведения (в частности, о признаках, выделяющих тексты Алисы из всех прочих комментариев). В этом случае скрывается не только факт пересылки некоторого секретного сообщения от Алисы к Бобу, но Ева не знает даже о том, что Алиса общается с Бобом. (Вспомните пример с цветами на подоконнике). В средствах массовой информации имеются упоминания о том, что некоторые группы террористов использовали именно такую схему для координации совместной деятельности.

В наше время методы стеганографии, или более обще, встраивания скрытой информации, находятся в центре внимания многих исследователей и инженеров. Ежегодно проводятся многочисленные конференции по защите информации, где многие доклады посвящены стеганографии и стегоанализу, издаются международные научные журналы, печатающие статьи о методах скрытой передачи информации и их применении. Среди довольно многочисленных обзорных статей и монографий по стеганографии мы отметим работы [12, 73], которые могут дополнить данный раздел. Кроме того, существуют свободно доступные через Интернет компьютерные программы как для встраивания скрытой информации, так и для ее обнаруживания, предназначенные для данных разных типов (таких как цифровые фотографии, фильмы, аудиофайлы, программы).

Рассмотрим теперь основные сферы применения методов «скрытия» данных. Самое очевидное использование методов встраивания данных — скрытая передача некоторых сведений от Алисы к Бобу. (Иногда именно эта сфера применения называется стеганографией, а другие — скрытием или встраиванием данных.) В настоящее время стеганографические методы используются при передаче данных через Интернет и подобные компьютерные сети путем встраивания секретной информации в аудио и видео файлы, в тексты на естественных языках и другие типы данных.

Другое, возможно, наиболее распространенное применение скрытия данных — защита авторских прав. Как и Эпихарм из Сиракуз, авторы или владельцы цифровых фильмов, фотографий, музыкальных произведений и даже компьютерных программ «вставляют» в них скрытые метки, которые называются или «водяными знаками» или «отпечатками пальцев» (watermark и fingerprint соответственно). В первом случае вставляемая метка одна и та же для всех копий (как, скажем, водяной знак на бумажных купюрах одного номинала), а во втором — метки разные в различных копиях; они являются как бы идентификатором копии открытого текста (или контейнера). Часто встраиваемый в цифровые файлы водяной знак называют «цифровой меткой» или цифровым водяным знаком. Их назначение не изменилось со времен Эпихарма — в случае необходимости законный пользователь может доказать свое авторство (или право собственности), продемонстрировав водяной знак, скажем, в суде. Применение цифровых отпечатков пальцев дает и некоторые дополнительные возможности и особенно широко применяется при защите от несанкционированного копирования. Поясним это на примере. Пусть хозяин цифрового фильма продает его копии различным владельцам кинотеатров с правом показа на определенной территории, разной для разных покупателей. При этом хозяин встраивает в каждую копию индивидуальную метку — «отпечаток пальца». Если один из купивших фильм будет незаконно продавать его копии для показа на других территориях, то по любой такой копии можно будет найти злоумышленника (и, скажем, привлечь его к суду).

К методам скрытия данных, применяемым для получения водяных знаков и отпечатков пальцев, обычно предъявляются специфические требования, отличные от систем скрытой передачи сообщений. Дело в том, что «злоумышленник» обычно заранее знает, что в тексте содержится скрытая информация — водяной знак или

отпечаток пальца, и его задача — не подтверждение этого факта, а уничтожение встроенных данных. Естественно, что для уничтожения скрытой информации применяются специфические приемы, отличные от методов обнаружения.

Третья сфера применения методов встраивания скрытых данных — проверка целостности документа, или, более обще, цифрового объекта. Простейший пример — файл, являющийся «цифровым носителем» некоторого документа, содержит скрытую встроенную цифровую метку, равную, скажем, значению хэш-функции текста документа. Тогда, если некоторый злоумышленник изменит часть текста документа, то новое значение хэш-функции будет отличаться от прежнего. Однако это прежнее значение спрятано, злоумышленнику неизвестно, и он его изменить не может. Поэтому-то это изменение документа будет обнаружено его составителем или другим лицом, знающим ключ для обнаружения метки.

Можно выделить и другие области применения методов встраивания данных, которые в значительной степени близки к выше-перечисленным. Например, в базе данных, содержащей историю болезней пациентов, разумно хранить их имена в «спрятанном» виде; это же относится и к базам данных, хранящим персональные данные любого другого типа. Считается, что хранение имен в скрытом, а не просто зашифрованном виде, в некоторых случаях повышает надежность защиты.

Рассмотрим более формально общую схему стегосистемы и некоторые типы атак. Итак, Алиса и Боб имеют секретный ключ и могут обмениваться «невинными» сообщениями (цифровыми фотографиями, аудиофайлами, электронными письма и т.д. и т.п.). У Евы может быть несколько различных задач и, тем самым, атак на стегосистему.

Во-первых, Еву может интересовать, содержат ли сообщения, пересылаемые Алисой, некоторую секретную (встроенную) информацию. Это, собственно, и есть задача стегоанализа. Для ответа на этот вопрос Ева может использовать самые разнообразные методы, часть из которых базируется на здравом смысле, другие — на алгоритмах спектрального анализа, трети используют аппарат проверки гипотез математической статистики и т.п.

Во-вторых, Ева может попытаться подвергнуть передаваемые данные некоторому воздействию, уничтожающему спрятанную информацию, если она есть в сообщении. Пусть, например, Алиса пе-

пересыпает Бобу фотографию в формате JPEG, в которой, возможно, содержится спрятанная информация. Ева может сначала сжать фотографию с некоторой потерей качества, а затем вернуться к прежнему размеру. При таком преобразовании некоторые мелкие детали, вообще говоря, могут исчезнуть или измениться, что может исказить и скрытые данные (если они были). Другими словами, Ева преобразует пересыпаемый файл-контейнер так, чтобы, с одной стороны, почти его не испортить, а с другой стороны существенно исказить встроенные данные. Такие атаки часто разрабатывают для уничтожения цифровых водяных знаков и отпечатков пальцев.

Третий тип атак — Ева подменяет сообщения, передаваемые Алисой, заменяя их на другие. Например, если Алиса пересыпает фотографии зданий города, в котором она живет, то Ева может заменять их на другие, сделанные в том же месте и при помощи такого же фотоаппарата. В этом случае Боб может не заметить подмены и будет введен в заблуждение.

Возможно выделить и другие типы атак, доступных Еве, но мы кратко остановимся только на одной, придуманной для уничтожения электронных отпечатков пальцев, т.к. она находит широкое применение и привлекает внимание многих исследователей, изобретающих способы защиты от несанкционированного копирования данных. Напомним, электронные отпечатки пальцев — это специальные метки, включаемые в защищаемые данные (пусть, для определенности, в цифровой фильм). Цифровой фильм, как и любой другой файл, может рассматриваться как текст в двоичном алфавите. Для проведения рассматриваемой атаки Ева должна найти еще одного злоумышленника (Адама), имеющего другую защищенную копию фильма, а затем Ева и Адам должны сравнить принадлежащие им копии и обнаружить различающиеся двоичные символы. Понятно, что эти символы принадлежат встроенным меткам, поэтому злоумышленники могут создать третью копию фильма, в которой символы, не совпадающие в первых двух, заменяются, скажем, на случайно порожденные. При этом они могут рассчитывать, что если законный владелец фильма обнаружит эту копию, то по ней он не сможет определить ни Еву, ни Адама, т.к. полученная «случайная» метка будет отличаться от меток из их копий.

Естественно, что устойчивость к атакам различных типов — наиболее важная характеристика стеганографических систем. Другой важный вопрос — скорость передачи скрытой информации для рас-

сматриваемой стегосистемы, которая может измеряться количеством бит в единицу времени или отношением длин скрытого сообщения и содержащего его контейнера. В настоящее время есть некоторые подходы и математические модели, предназначенные для решения вопроса о максимально возможной скорости передачи скрытой информации [69], но для большинства практически используемых методов стеганографии они не применимы.

В завершение этого раздела мы сформулируем правило Керкхоффса применительно к стеганографии: предполагается, что при проведении атак на систему Ева знает метод, который, возможно, используется Алисой и Бобом для скрытой передачи информации, но не знает секретного ключа, известного только им.

Ниже мы с разной степенью подробности рассмотрим как основные методы встраивания данных, так и методы их обнаружения, уделяя главное внимание атаке, направленной на выявление самого факта присутствия скрытой информации. Особое внимание будет уделено описанию и исследованию свойств совершенной стегосистемы, в которой факт наличия скрытой информации в принципе не обнаруживаем.

## 11.2. Основные методы встраивания скрытых данных

Мы начнем с описания одного из старейших и широко распространенных методов стеганографии, который по английски называется Least Significant Bit (LSB), что можно перевести как встраивание в младшие знаки или в младшие биты. В настоящее время он применяется для встраивания скрытых данных в цифровые фотографии, аудиофайлы и фильмы, но, как и многие другие алгоритмы стеганографии, возник еще в средние века. До появления компьютеров при проведении расчетов широко использовались таблицы логарифмов и других математических функций. Эти таблицы состояли из многих тысяч значений, скажем, десятичного логарифма, представленных обычно в виде многостраничных книжек. Вычисление табличных значений отнимало много времени и сил у их разработчиков, что собственно и определяло их высокую стоимость и делало весьма актуальной проблему защиты авторских прав. Дело в том, что злоумышленник мог скопировать такую таблицу и продавать

копии по более низкой цене, в результате чего истинный автор нес потери. Для защиты от этого разработчики в каждую продаваемую таблицу вносили небольшие искажения, заменяя последнюю цифру в значении логарифма на неверную. (Например, значение 0,2284938 могли заменить на 0,2284939). Подчеркнем, что в разных продаваемых копиях таблицы изменения вносились в различные значения, поэтому, если покупатель какой-либо копии начинал ее тиражировать и продавать, то законный владелец таблиц мог по любой такой «пиратской» копии найти злоумышленника и, скажем, обратиться в суд. Для доказательства своей правоты в суде было достаточно продемонстрировать истинные значения и внесенные искажения.

Интересно, что в наше время для защиты цифровых фотографий, аудиофайлов и других данных от пиратского копирования используется точно такой же метод. В графическом формате BMP, аудиоформате WAW и целом ряде других данные представлены в виде двоичных чисел, под запись каждого из которых отводится слово фиксированной длины (как правило, кратной байту). Для записи «секретной» информации используются последние биты в указанных словах, соответствующие младшим цифрам чисел. Отсюда и произошло название метода — встраивание в наименее значимые биты (Least-Significant Bit — LSB). При использовании этого метода качество, скажем, фотографий почти не ухудшается и вносимые изменения, как правило, не видны человеческим глазом. Однако если для записи скрытой информации использовать числа, характеризующие яркость всех пикселей фотографии, то современные методы стегоанализа смогут обнаружить наличие встроенной информации. Поэтому в современных вариантах описываемого метода для встраивания информации используются последние биты только небольшой доли пикселей, причем они выбираются, как правило, при помощи генератора псевдослучайных чисел. Как известно, последовательность, вырабатываемая генератором псевдослучайных чисел, определяется начальным значением, и во многих методах встраивания это значение является секретным ключом, известным Алисе и Бобу. В том случае если метод используется для защиты авторских прав, лучше считать, что есть законный владелец, который встраивает индивидуальные метки (электронные отпечатки пальцев) в места, определяемые при помощи секретного ключа. При обнаружении пиратской копии он при помощи того же самого ключа прочтет спрятанный номер и определит виновного в пиратском копировании.

Приведем более подробно описание формата BMP, для которого существует довольно много программ встраивания скрытых данных, основанных на методе LSB. Аббревиатура BMP означает BitMap («битовая карта», «битовая матрица»). BMP относится к числу тех графических форматов, где данные представлены «как есть», без каких-либо преобразований, сжатия и т.д. По этой причине файлы BMP имеют довольно большой размер, например, изображение разрешением  $1280 \times 800$  будет занимать 2,92 Мб при кодировании одного пикселя («точки») 24-битовым словом. У этого формата довольно много вариантов, но мы рассмотрим только так называемый неиндексированный BMP. Этот способ представления данных в основном предназначен для полноцветной графики и в настоящее время широко распространен. Основным представителем неиндексированных форматов является 24-битовый. При представлении данных в этом формате в начале файла располагается заголовок, а основной частью является массив, содержащий информацию о всех точках изображения. В неиндексированных форматах каждая точка изображения кодируется тремя цветовыми компонентами: синей, зеленой и красной (R, G и B). Под каждую компоненту 24-битового BMP отводится по одному байту, следовательно, один пиксель изображения занимает 3 байта или 24 бита. Для встраивания скрытой информации методом LSB используются последние биты байтов, кодирующих цветовые компоненты. В настоящее время существуют десятки различных методов встраивания скрытых данных и методов стегоанализа для файлов в формате BMP, причем появление новых методов стегоанализа приводит к совершенствованию алгоритмов встраивания и наоборот.

Среди различных вариантов LSB метода мы остановимся на так называемом LSB matching, который был описан сравнительно недавно и считается наиболее устойчивым ко многим методам стегоанализа [64]. К особенностям этого метода относится то, что, во-первых, при кодировании используется последовательность случайных двоичных чисел и, во-вторых, при замене последнего бита используются сложение и вычитание, причем эти операции проводятся в «обычной» двоичной арифметике (с переносами единиц).

Итак, будем считать для определенности что под запись каждого значения отводится один байт (т.е. восьмибитовая строка) и обозначим число, записанное в  $i$ -ом байте через  $x_i$ , последний знак этого числа — через  $LSB(x_i)$ , встраиваемый бит — через  $m_i$  (его

значение равно 0 или 1), и случайное двоичное число, принимающее с равной вероятностью значения  $-1$  или  $+1$ , пусть будет  $r_i$ . Тогда LSB matching задается равенством

$$\bar{x}_i = \begin{cases} x_i, & \text{если } \text{LSB}(x_i) = m_i \\ x_i + r_i, & \text{если } \text{LSB}(x_i) \neq m_i, \quad x_i \neq 0, \quad x_i \neq 255 \\ x_i + 1, & \text{если } \text{LSB}(x_i) \neq m_i, \quad x_i = 0 \\ x_i - 1, & \text{если } \text{LSB}(x_i) \neq m_i, \quad x_i = 255 \end{cases}$$

Здесь  $\bar{x}_i$  — значение после встраивания, а последние две строчки посвящены тем довольно редким случаям, когда  $x_i$ , рассматриваемое как число в двоичной системе счисления, принимает минимальное и максимальное значение. Заметим, что последний знак  $\bar{x}_i$  совпадает со значением встраиваемого  $m_i$ .

Другой распространенный класс методов встраивания скрытой информации предназначен для форматов, в которых данные представлены в виде набора коэффициентов при разложении по множеству ортогональных функций. К таким форматам относится JPEG и MP3, предназначенные для представления изображений и аудиоданных соответственно. В настоящее время многие такие методы реализованы в виде программ и широко доступны; описание многих из них можно найти в [12]. Основная идея довольно проста: последовательность встраиваемых символов записывается вместо младших знаков коэффициентов разложения. При этом, как и в LSB, предполагается, что изменения младших цифр вносят незначительные искажения и поэтому их трудно обнаружить. При встраивании учитываются свойства представления данных в конкретном формате и особенности человеческого восприятия. Так, для изображений известно, что при разложении в ряд Фурье искажения в высокочастотных гармониках менее заметны, чем в низкочастотных, поэтому последние не используются для встраивания скрытой информации.

Остановимся кратко на встраивании скрытой информации в текстовые данные. Во-первых, стоит отметить методы, основанные на использовании синонимов. Идея таких методов довольно проста и мы поясним ее на примере. Пусть Алиса и Боб договорились заранее о том, что в парах синонимов «*большой* — *крупный*», «*умный* — *способный*», «*летчик* — *пилот*» первое слово кодирует 0, а второе — 1. Пусть Алиса отправляет текст ... *умный пилот* ... *крупный* ..., в котором через ... обозначены участки текста, не содержащие синонимов из списка, заранее согласованного Алисой и Бобом. После

получения текста Боб прочитает скрытую последовательность 011.

Другой способ, описанный в литературе и реализованный в виде свободно доступных программ — внесение грамматических ошибок и опечаток. Алиса и Боб договариваются о списке ошибок и опечаток и их скрытых значениях, а затем действуют примерно так же, как и в случае синонимов.

Существуют методы встраивания скрытой информации, использующие алгоритмы автоматического генерирования текстов на естественных языках. При использовании таких методов на вход программы подается последовательностей нулей и единиц, а на выходе появляется кодирующий ее текст. Боб при получении такого текста производит обратное преобразование и извлекает скрытый текст. Важно отметить, что такие тексты являются грамматически корректными наборами слов, но человек сразу поймет их ненатуральность, искусственность. (Это относится к современным методам; возможно, что со временем появятся алгоритмы, генерирующие осмысленные тексты). Смысл использования этих методов в том, что при автоматическом их анализе (т.е. без участия человека) обнаружение наличия скрытой информации может быть трудной задачей.

Интересно, что компьютерные программы могут также использовать для встраивания скрытой информации. На первый взгляд кажется, что это невозможно — ведь изменение символа в тексте программы должно ее «испортить». Однако оказывается что это не всегда так: в программах существуют такие последовательности команд, изменение порядка выполнения которых не меняет результата их работы. Например, следующие фрагменты программы на языке высокого уровня `c1 := 3; ff := 17` и `ff := 17; c1 := 3` эквивалентны. Алиса и Боб могут использовать такие последовательности для передачи скрытой информации. Скажем, они могут договориться о том, что если тексты таких пар операций расположены в алфавитном порядке, то они кодируют 0, в противном случае — 1. Тогда в нашем примере первая пара кодирует 0, вторая — 1. Более подробно с этой тематикой можно познакомиться по работам [30, 72].

### 11.3. Стегоанализ на основе сжатия данных

Методы стегоанализа, т.е. методы обнаружения скрытой информации во внешне «невинных» данных, активно разрабатываются и исследуются многими специалистами; см., например, работы [29, 46, 59, 65, 66] и ссылки, в них приведенные. Как большинство современных методов скрытия данных, так и большинство методов стегоанализа основываются на различных приемах, базирующихся на здравом смысле и экспериментальной проверке. Это обусловлено отсутствием развитой теории, которая могла бы непосредственно применяться к различным типам контейнеров, в которые встраиваются секретные данные. Большинство практически используемых методов скрытия данных меняют вероятностное распределение, или статистическую структуру, исходного контейнера. Например, использование младших знаков для скрытия данных (LSB метод) приводит к внесению некоторого «шума» и тем самым меняет распределение вероятностей. Это же относится и к встраиванию скрытой информации в тексты — использование, скажем, синонимов для встраивания скрытой информации может привести к изменению частот встречаемости слов и их сочетаний. Другими словами, распределения вероятностей пустых и заполненных контейнеров различны и именно это свойство используется для стегоанализа. Известны многочисленные методы стегоанализа, предназначенные для различных типов данных-контейнеров, описание которых может быть найдено, например, в [12, 29, 46, 59, 65, 66] и в цитируемых там работах.

Мы остановимся в данном разделе только на одном общем подходе, который базируется на использовании методов сжатия данных и рассмотрим его применение на одном примере. Неформальное обоснование предлагаемого метода основывается на том, что скрыто включаемые данные статистически независимы от содержимого контейнера, поэтому при его сжатии объем полученного файла будет больше, чем при сжатии пустого контейнера. (Это относится к «идеальному сжатию», или к так называемым методам универсального кодирования, в пределе сжимающим данные до минимально возможной величины — энтропии Шеннона. Отметим, что многие реально используемые методы сжатия данных основаны на универсальных кодах).

На этом наблюдении основывается следующий метод, или, скопее, схема: контейнер, возможно содержащий скрытые, или встроенные

ные, данные сжимается сначала в «исходном» виде. Затем в контейнер встраиваются данные при помощи одного из известных методов и полученный заполненный контейнер снова сжимается. Если разность в длинах первого и второго сжатых файлов велика, то разумно предположить, что в первом, исходном, контейнере не содержались скрытые данные. Если же разность длин не велика, то вероятно в исходном контейнере содержались скрытые данные. Отметим сразу, что для выбора наиболее эффективного метода сжатия и для определения того, какая разность велика, а какая не велика, необходимо проводить предварительные эксперименты. При этом исследуемый контейнер может разбиваться на части для повышения эффективности метода. Важно также отметить, что при применении описанного подхода можно использовать широко распространенные так называемые архиваторы, предназначенные для сжатия данных.

Описанный подход применялся для стегоанализа данных в форматах BMP и JPEG [98]. Оказалось, что полученные стегодетекторы более эффективны, чем методы, базирующиеся на других подходах; см. [98]. В качестве примера мы остановимся несколько подробнее на стегоанализе данных в формате BMP.

Представим некоторое изображение  $X$  как последовательность  $x_1, \dots, x_N$ , где  $N$  — длина файла в байтах, а  $x_i$  — содержимое  $i$ -го байта. Разобьем последовательность  $x_1, \dots, x_N$  на  $d$  равных отрезков и обозначим каждый отрезок через  $X_j$ , где  $j = 1, \dots, d$ . Пусть  $\Psi$  — некоторый алгоритм сжатия,  $U$  — файл, а  $|\Psi(U)|$  — его длина после сжатия методом  $\Psi$ . Обозначим через  $f(U)$  отношение длины сжатого файла к исходной:

$$f(U) = \frac{|\Psi(U)|}{|U|}.$$

Для файла («изображения»)  $U$  обозначим через  $\varphi(U)$  файл, полученный из  $U$  путем замены последних значений всех байтов на последовательность случайных двоичных символов и определим величину

$$\delta(U) = |f(\varphi(U)) - f(U)|.$$

Поясним неформально смысл введения этой величины. Представим себе, что изображение  $U$ , которое можно рассматривать как контейнер, не использовалось для записи скрытой информации. Тогда при применении преобразования  $\varphi(U)$  последние знаки (восьмые биты)

будут случайной последовательностью и не будут статистически связаны с остальной информацией. Поэтому длина сжатого файла  $\varphi(U)$  будет существенно больше, чем длина сжатого исходного файла  $U$ . Следовательно, и величина  $f(\varphi(U))$  будет больше  $f(U)$ , а значит и разность этих длин  $\delta(U)$  будет относительно велика. С другой стороны, если в исходное изображение уже внесли скрытую информацию, записав ее в последние, восьмые, биты, то вторичная запись в эти знаки, происходящая при применении  $\varphi(U)$  не изменит статистической структуры данных, поэтому длины сжатых файлов  $\Psi(U)$  и  $\Psi(\varphi(U))$ , будут близки и, соответственно, величина  $\delta(U)$  будет относительно мала. На этом наблюдении и построен метод из [98]: вычисляется величина  $\delta(X_j)$  для всех частей  $X_j$ ,  $j = 1, \dots, d$ , и по их максимальному значению делается вывод о наличии или отсутствии скрытой информации в исходном файле  $X$ ; детальное описание можно найти в [98].

В заключение этого раздела отметим, что применение архиваторов дает возможность построить достаточно эффективные алгоритмы стегоанализа текстов, в которых скрытые данные встроены при помощи упоминавшихся в предыдущем разделе методов синонимов, ошибок и опечаток, а также автоматической генерации.

#### 11.4. Асимптотически оптимальные совершенные стеганографические системы

В этом разделе мы рассмотрим конструкцию совершенной стеганографической системы, т.е. такой, где сообщения, несущие и не несущие скрытую информацию (пустые и заполненные контейнеры) подчиняются одному и тому же распределению вероятностей и, следовательно, статистически неразличимы. Другими словами, Ева в принципе не сможет различить пустые и заполненные контейнеры. Эта система была предложена Рябко Б. Я. и Рябко Д. Б. [22, 83].

Во многих случаях использования стегосистем закон распределения вероятностей сообщений, в которые «встраивается» скрытая информация, точно неизвестен, а в случае когда эти сообщения суть цифровые фотографии, фильмы, музыкальные произведения, электронные письма, SMS- и ICQ-сообщения и т.п., закон распределения, по-видимому, и не может быть известен точно. Поэтому довольно естественной кажется рассмотренная в [42] задача построения так

называемых универсальных стегосистем, в которых закон распределения вероятностей «пустых» контейнеров неизвестен.

Приведем некоторые обозначения, используемые в дальнейшем. Мы будем считать, что дан некоторый источник *открытых сообщений*  $\mu$ , порождающий независимые и одинаково распределенные случайные величины, принимающие значения из некоторого, возможно бесконечного, алфавита  $A$ . Есть два участника — Алиса и Боб — и Алиса собирается использовать этот источник для скрытой передачи сообщений, состоящих из последовательности символов алфавита  $B = \{0, 1\}$ , порождаемых независимо и с равными вероятностями. Этот источник мы обозначим через  $\omega$  и в дальнейшем будем называть его *источником секретных сообщений*. Такая модель источника секретных сообщений является фактически общепринятой, т.к. обычно предполагается, что секретные сообщения уже зашифрованы Алисой с ключом, известным только ей и Бобу. Если Алиса использует шифр Вернама, то зашифрованная последовательность состоит из равновероятных и независимых символов; если же используются современные блоковые или потоковые шифры с секретным ключом, то, как мы видели ранее, зашифрованная последовательность должна быть «похожа» на цепочку равновероятных и независимых двоичных символов. («Похожесть» может означать неотличимость за полиномиальное время или подтверждаться экспериментальными статистическими данными, имеющимися для всех современных шифров.) Как обычно, мы предполагаем, что Ева читает все сообщения, передаваемые от Алисы к Бобу и пытается установить, не содержат ли они какую-либо скрытую информацию. Отметим еще раз, что если сообщения, содержащие и не содержащие встроенную секретную информацию, подчиняются одному и тому же закону распределения вероятностей, то Ева (и никто другой) не может различать такие сообщения. Благодаря этому свойству такие системы и названы совершенными.

В данном разделе предлагается конструкция универсальной совершенной стегосистемы и показано, что скорость передачи скрытой информации в ней ограничена предельной величиной — энтропией Шеннона источника  $\mu$ ; затем описаны конструкции стегосистем, где эта скорость приближается к пределу. Важно отметить, что предлагается простой алгоритм кодирования и декодирования, сложность которого растет полиномиально при стремлении скорости передачи спрятанной информации к ее пределу — энтропии Шеннона.

Для того чтобы объяснить основную идею предлагаемой конструкции, мы начнем описание системы с простейшего случая, когда не только источник секретных символов  $\omega$  двоичный, но и источник открытых сообщений  $\mu$  порождает последовательность независимых символов из двоичного алфавита  $A = \{a, b\}$ . Пусть требуется передавать (секретную) последовательность символов  $y^* = y_1 y_2 y_3 \dots$ , порожденную источником независимых и равновероятных двоичных символов  $\omega$  и пусть имеется порожденная источником последовательность символов  $x^* = x_1 x_2 x_3 \dots$ . Например, пусть

$$y^* = 0110 \dots, \quad x^* = aababaaaabbaaaaaabb \dots \quad (11.1)$$

При кодировании последовательности  $y^*$   $x^*$  преобразуется в новую последовательность  $X$ , передаваемую Бобу и такую, что, во-первых, по  $X$  Боб может однозначно восстановить (секретную) последовательность  $y^*$  и, во-вторых, распределение вероятностей символов в  $X$  такое же, как и в  $x^*$ . (Другими словами,  $X$  и  $x^*$  статистически не различимы.) Процесс построения последовательности  $X$  по  $x^*$  и  $y^*$  мы разобьем на этапы. Сначала разделим все символы  $x^*$  на пары и для удобства обозначим все возможные пары следующим образом:

$$aa = u, \quad bb = u, \quad ab = v_0, \quad ba = v_1.$$

Например, последовательность из (11.1) можно представить в виде

$$x^* = aa \; ba \; ba \; aa \; ab \; ba \; aa \; aa \; bb \dots = uv_1v_1uv_0v_1uuv_1 \dots$$

(пробелы поставлены только для удобства чтения.) Затем сформируем последовательность  $X$  следующим образом: все пары букв, соответствующие  $u$ , оставим без изменения, а пары, соответствующие  $v_k$  заменим последовательно на пары букв, соответствующие  $v_{y_1} v_{y_2} v_{y_3} \dots$ . В рассматриваемом примере (11.1) мы получим следующую последовательность  $X$ :

$$X = aa \; ab \; ba \; aa \; ba \; ab \; aa \; aa \; bb \dots$$

Декодирование очевидно: Боб разбивает полученную последовательность символов  $X$  на пары и заменяет пары  $ab$  и  $ba$  на 0 и 1 соответственно, а остальные пары символов просто пропускает.

Свойства описанного метода, который мы обозначим через  $St_2$ , характеризует следующий почти очевидный факт.

**Утверждение 11.1.** Пусть дан источник  $\mu$ , который порождает независимые и одинаково распределенные случайные величины, принимающие значения из алфавита  $A = \{a, b\}$ , и пусть этот источник используется для скрытой передачи сообщений, состоящих из независимых и равновероятных двоичных символов, по описанному методу  $St_2$ . Тогда распределение вероятностей сообщений, получаемых на выходе стегосистемы, то же, что и у источника  $\mu$ .

Мы не будем приводить вполне очевидное доказательство этого утверждения, т.к. оно является частным случаем приводимой ниже теоремы.

В разделе 9.2 мы видели, что близкая конструкция была использована фон Нейманом при построении последовательности равновероятных двоичных символов. Его метод, как и описанная стегосистема, базировался на том, что вероятности появления пар символов  $ab$  и  $ba$  равны.

Описанную выше конструкцию легко обобщить на случай произвольного алфавита  $A$ . Действительно, зададим на множестве всех букв  $A$  какой-либо порядок. (Здесь стоит отметить, что  $A$  может состоять, например, из фотографий, но, в любом случае, все эти и подобные объекты представлены в виде двоичных слов и могут быть упорядочены, скажем, лексикографически). Как и ранее, чтобы передать (секретную) последовательность символов  $y^* = y_1 y_2 y_3 \dots$ , порождаемую источником независимых и равновероятных двоичных символов  $\omega$ , имеющаяся последовательность, порожденная источником независимых символов  $\mu$ ,  $x^* = x_1 x_2 x_3 \dots$ ,  $x_i \in A$ , разбивается на блоки длины 2. Если блок  $x_{2i-1} x_{2i}$  состоит из одинаковых букв, то он не используется для кодирования и передается без изменений; если же блок  $x_{2i-1} x_{2i}$  состоит из разных букв, скажем,  $\alpha$  и  $\beta$ , то он используется для кодирования очередного символа, который мы обозначим через  $y_k$ . Без ограничения общности предположим, что  $\alpha < \beta$  при заданном упорядочивании; тогда в передаваемую последовательность включается слово  $\alpha\beta$ , если  $y_k = 0$ , и  $\beta\alpha$ , если  $y_k = 1$ . Декодирование очевидно: если пара символов  $X_{2i-1} X_{2i}$  в закодированной последовательности состоит из одинаковых букв, то она не кодирует символ из  $y^*$ ; если же  $X_{2i-1} X_{2i}$  различны и  $X_{2i-1} < X_{2i}$  (при заданном упорядочивании), то очередной скрыто передаваемый символ ( $y_k$ ) равен 0, в противном случае  $y_k = 1$ . Обозначим описанную стегосистему через  $St_2(A)$ .

**Теорема 11.2.** Пусть дан источник  $\mu$ , порождающий независимые и одинаково распределенные случайные величины, принимающие значения из некоторого алфавита  $A$ , и пусть этот источник используется для скрытой передачи сообщений, состоящих из независимых и равновероятных двоичных символов при помощи стегосистемы  $St_2(A)$ . Тогда распределение вероятностей сообщений, получаемых на выходе стегосистемы, то же, что и у  $\mu$ , а среднее количество передаваемых букв, приходящихся на один секретно передаваемый бит, равно  $2/(1 - \sum_{a \in A} \mu(a)^2)$ .

**Доказательство.** Для доказательства возьмем произвольные  $\alpha, \beta \in A$  и  $i$  и покажем, что

$$P(X_{2i-1}X_{2i} = \alpha\beta) = \mu(\alpha\beta).$$

Если  $\alpha = \beta$ , то  $P(X_{2i-1}X_{2i}) = P(x_{2i-1}x_{2i})$ , т.е. вероятности в последовательности, содержащей скрытую информацию, и исходной, совпадают. Пусть теперь  $\alpha < \beta$ . Тогда

$$\begin{aligned} P(X_{2i-1}X_{2i} = \alpha\beta) &= \\ &= P(y_k = 0)P(x_{2i}x_{2i+1} = \alpha\beta) + P(y_k = 1)P(x_{2i}x_{2i+1} = \beta\alpha) = \\ &= 1/2 \mu(\alpha)\mu(\beta) + 1/2 \mu(\beta)\mu(\alpha) = \mu(\alpha)\mu(\beta). \end{aligned}$$

Случай  $\beta > \alpha$  разбирается аналогично. Второе утверждение получается прямым вычислением вероятности того, что в блоке обе буквы одинаковы.  $\square$

Отметим, что на практике, когда открыто передаваемые символы из  $A$  являются, например, графическими файлами, и каждый файл практически уникален, алфавит  $A$  огромен, так что среднее количество передаваемых букв (графических файлов), приходящихся на один секретно передаваемый бит, примерно равно 2.

Перейдем к описанию общего метода. Пусть, как и ранее, требуется скрытно передавать (секретную) последовательность символов  $y^* = y_1y_2y_3\dots$ , порожденную источником независимых и равновероятных двоичных символов  $\omega$ , и пусть имеется последовательность символов  $x^* = x_1x_2x_3\dots$ , порожденная источником независимых символов  $\mu$ , где каждый символ  $x_i$  принадлежит алфавиту  $A$ . В предлагаемой стегосистеме последовательность  $x^*$  разбивается на блоки длины  $n$ , где  $n > 1$  — параметр метода.

Каждый блок используется для кодирования некоторого числа символов из  $y^*$  (например, в ранее описанной стегосистеме  $St_2(A)$  каждый блок из двух символов кодировал либо одну букву из  $y^*$ , либо ноль букв). Однако, в общем случае возникает одна особенность, не встречавшаяся в случае двухбуквенного блока. Точнее, возникает задача согласования вероятностей блоков из последовательностей  $x^*$  и  $y^*$ . Дело в том, что вероятности слов, порождаемых источником секретных символов, кратны степени двух, тогда как количество равновероятных блоков может не удовлетворять этому условию.

Перейдем к точному описанию. Обозначим через  $u$  первые  $n$  букв из  $x^*$ :  $u = x_1 \dots x_n$  и пусть  $\nu_u(a)$  — количество встреч буквы  $a$  в слове  $u$ . По определению, множество  $S_u$  состоит из всех слов длины  $n$ , у которых частота встречаемости каждой буквы из алфавита  $A$  та же, что и в слове  $u$ , т.е.  $S_u$  состоит из слов частотного класса слова  $u$ . (Для того чтобы пояснить смысл рассмотрения этого множества, заметим, что вероятности всех его элементов равны, т.к.  $\mu$  — источник независимых и одинаково распределенных случайных величин.) Пусть на множестве слов  $S_u$  задан какой-либо порядок (скажем, лексикографический), известный Алисе и Бобу, и пусть  $S_u = \{s_0, s_1, \dots, s_{|S_u|-1}\}$  при этом упорядочивании.

Обозначим  $m = \lfloor \log_2 |S_u| \rfloor$ , где  $\lfloor y \rfloor$  — наибольшее целое, не превосходящее  $y$ . Рассмотрим двоичное представление числа  $|S_u|$ :

$$|S_u| = (\alpha_m, \alpha_{m-1}, \dots, \alpha_0),$$

причем  $\alpha_m = 1$ ,  $\alpha_j \in \{0, 1\}$ ,  $m > j \geq 0$ . Другими словами,

$$|S_u| = \alpha_m 2^m + \alpha_{m-1} 2^{m-1} + \alpha_{m-2} 2^{m-2} + \dots + \alpha_0, \quad \alpha_m = 1.$$

Обозначим через  $\delta(u)$  номер слова  $u$  (при заданном на  $S_u$  порядке) и пусть  $(\lambda_m, \lambda_{m-1}, \dots, \lambda_0)$  — двоичное представление числа  $\delta(u)$ . Пусть  $j(u)$  — наибольшее из чисел, таких что  $\alpha_j \neq \lambda_j$ . Алиса, определив  $j(u)$ , считывает  $j(u)$  букв из последовательности скрытно передаваемых символов  $y^*$  и пусть эти символы, рассматриваемые как число в двоичной системе счисления, равны  $\tau$ . Алиса находит в множестве  $S_u$  слово  $v$ , номер которого в  $S_u$  равен  $\sum_{j(u) < s \leq m} \alpha_s 2^s + \tau$  и передает слово  $v$  Бобу (или, другими словами,  $v$  помещается в выходную последовательность кодера.)

При декодировании Боб, получив слово  $v$ , определяет множество  $S_v$  (совпадающее с  $S_u$ ), находит так же, как при кодировании,

$j(v)$  (для  $u$  и  $v$  они совпадают:  $j(u) = j(v)$ ) и  $\tau$ , а затем по  $\tau$  определяет  $j(v)$  закодированных символов. Все последующие  $n$ -буквенные слова кодируются Алисой и декодируются Бобом аналогично. Обозначим эту систему через  $St_n(A)$ .

Рассмотрим пример, иллюстрирующий все этапы вычислений.

**Пример 11.1.** Пусть  $A = \{a, b, c\}$ ,  $n = 3$ ,  $u = bac$ . Тогда  $S_u = \{abc, acb, bac, bca, cab, cba\}$ ,  $|S_u| = 6$ ,  $m = 2$ ,  $\alpha_2 = 1$ ,  $\alpha_1 = 1$ ,  $\alpha_0 = 0$ ,  $\delta(u) = 2$ ,  $\lambda_2\lambda_1\lambda_0 = 010$ ,  $j(u) = 2$ . Вычислив эти значения, Алиса считывает  $j(u) (= 2)$  секретно передаваемых символов последовательности  $u^*$ . Пусть, для определенности, эти символы будут 11. После этого Алиса находит  $j(v) = 2$  и номер слова  $v$  в  $S_v (= S_u)$  в данном случае равный  $\sum_{2 < s \leq 2} \alpha_s 2^s + \tau = 0 + 3 = 3$ . Соответствующее ему слово  $v = bca$ . Боб, получив это слово, определяет по нему  $S_v (= S_u)$ ,  $\tau = 3$ , а по значению  $\tau$  он узнает переданные секретные символы 11.  $\square$

**Теорема 11.3.** Пусть дан источник  $\mu$ , порождающий независимые и одинаково распределенные случайные величины, принимающие значения из некоторого алфавита  $A$  и пусть этот источник используется для скрытой передачи сообщений, состоящих из независимых и равновероятных двоичных символов, по описанному выше методу  $St_n(A)$  при длине блока  $n$ ,  $n \geq 2$ . Тогда выполнены следующие утверждения.

- 1) Сообщения, получаемые на выходе стегосистемы подчиняются распределению  $\mu$  (т.е. распределение входной и выходной последовательности кодера одинаковые и, следовательно, система совершенна).
- 2) Среднее число скрытых символов на букву источника ( $L_n$ ) удовлетворяет неравенству

$$L_n \geq \frac{1}{n} \left( \sum_{u \in A^n} \mu(u) \log \frac{n!}{\prod_{a \in A} \nu_u(a)!} - 2 \right),$$

где  $\mu(u)$  есть вероятность порождения слова  $u$  источником  $\mu$ , а  $\nu_u(a)$  – количество встреч буквы  $a$  в слове  $u$ .

- 3) Если алфавит  $A$  конечен и длина блока  $n$  стремится к бесконечности, то среднее число  $L_n$  скрытых символов на букву

стремится к энтропии Шеннона источника сообщений

$$h(\mu) = - \sum_{a \in A} \mu(a) \log \mu(a).$$

**Доказательство.** Для доказательства первого утверждения теоремы достаточно показать, что для каждого  $n$ -буквенного слова  $u$  из входной, исходной, последовательности вероятность появления в кодирующей последовательности любого слова  $v \in S_u$  равна  $1/|S_u|$ . Доказательство, как и ранее, основано на применении формулы полной вероятности. Как видно из описания, при  $\alpha_j = 1$  вероятность того, что из последовательности скрытно передаваемых символов будет считано  $j$  бит,  $j = 0, \dots, m$ , равна  $2^j/|S_u|$ , т.к. номер слова  $u$  в  $S_u$  должен удовлетворять неравенству

$$\sum_{j(u) < s \leq m} \alpha_s 2^s \leq \delta(u) < \sum_{j(u) \leq s \leq m} \alpha_s 2^s.$$

Пусть, для определенности,  $u$  и  $v$  — первые слова в исходной и закодированной последовательности. Тогда

$$P(X_1 \dots X_n = v) = P(u \in S_v \text{ и } j(v) = j(u)) 2^{-j(v)}.$$

Здесь последний множитель равен вероятности считать из последовательности скрытно передаваемых символов  $u^*$  двоичное слово длины  $j(v)$ , кодирующее  $v$ . Из последнего равенства получаем:

$$\begin{aligned} P(X_1 \dots X_n = v) &= P(u \in S_v) P(j(v) = j(u) | u \in S_v) 2^{-j(v)} = \\ &= |S_v| \mu(u) (2^{j(v)} / |S_v|) 2^{-j(v)} = \mu(u). \end{aligned}$$

Так как  $u$  и  $v$  принадлежат к одному частотному классу, из последнего равенства видно, что  $P(X_1 \dots X_n = v) = \mu(v)$ .

Для доказательства второго утверждения определим величину  $\phi = 2^m/|S_u|$  и обозначим через  $L(S_u)$  среднее число скрытно передаваемых символов, приходящихся на одно слово из  $S_u$ :

$$L(S_u) = \frac{1}{|S_u|} \sum_{i=0}^m \alpha_i i 2^i.$$

Справедливы следующие соотношения:

$$\begin{aligned}
 L(S_u) &= \frac{1}{|S_u|} \sum_{i=0}^m \alpha_i 2^i = \frac{1}{|S_u|} \left( m \sum_{i=0}^m \alpha_i i 2^i - \sum_{i=0}^m \alpha_i 2^i (m-i) \right) = \\
 &= m - \left( 2^m \sum_{k=0}^m k \alpha_{m-k} 2^{-k} \right) > m - 2^{m+1} / |S_u| = \\
 &= m - 2/\phi = \log |S_u| - \log \phi - 2/\phi.
 \end{aligned}$$

Можно проверить прямым нахождением максимума, что

$$\log \phi + 2/\phi \leq 2 \text{ при } \phi \in [1, 2].$$

Отсюда получаем, что

$$L(S_u) > \log |S_u| - 2.$$

Отсюда, и из равенства

$$|S_u| = \frac{n!}{\prod_{a \in A} \nu_u(a)!}$$

получаем второе утверждение теоремы.

Последнее утверждение следует из широко известного в теории информации факта, гласящего, что с вероятностью, стремящейся к 1, справедливо неравенство

$$h(\mu) - \delta < \log |S_u|/n < h(\mu) + \delta$$

при любом  $\delta > 0$ , см., например, [45, 4]. □

Во многих реальных стегосистемах алфавит  $A$  огромен (скажем, состоит из всех возможных цифровых фотографий заданного формата или всех возможных электронных писем). В этом случае представляет интерес асимптотическое поведение  $L_n$  при фиксированном  $n$  и  $|A| \rightarrow \infty$ . Для точного рассмотрения этого случая мы будем использовать понятие минимум-энтропии (minimum entropy или minentropy), которая определяется равенством

$$H_\infty(\mu) = \min_{a \in A} \{-\log \mu(a)\}.$$

**Следствие 11.4.** Если выполнены условия теоремы 11.3, длина блока  $n$  конечна и количество букв в  $A$  стремится к бесконечности так, что  $H_\infty(\mu) \rightarrow \infty$ , то величина  $L_n$  удовлетворяет неравенствам

$$\log(n!)/n \geq L_n \geq (\log(n!) - 2)/n,$$

что при больших  $n$  эквивалентно асимптотическому равенству

$$L_n = \log n(1 + o(n)).$$

Это утверждение легко выводится из того, что число перестановок из  $n$  различных элементов равно  $n!$  и второго утверждения теоремы 11.3.

Остановимся теперь кратко на оценке сложности стегосистемы  $St_n(A)$ . Хранение всех слов из множества  $S_u$  потребовало бы порядка  $2^n \log |A|$  бит памяти, что конечно практически нереализуемо при больших  $n$ . В [17] предложен алгоритм быстрой нумерации, который позволяет находить номер блока любого слова  $u$  в множестве  $S_u$  при кодировании и проводить обратную операцию при декодировании, затрачивая  $O(\log^{\text{const}} n)$  операций на символ при объеме памяти  $O(n \log^3 n)$  бит.

Стоит отметить, что основная идея, использованная при построении стегосистемы  $St_n(A)$ , применима и к более общим источникам открытых сообщений, чем источники, порождающие независимые одинаково распределенные сообщения. В самом деле, единственное свойство таких источников, которое мы использовали, заключается в том, что все блоки сообщений, полученные друг из друга перестановками, имеют одинаковую вероятность. Если источник открытых сообщений обладает тем свойством, что на каком-то шаге некоторые сообщения имеют одинаковую (условную) вероятность, то, в случае генерации источником одного из этих сообщений, заменяя его при необходимости на одно из равновероятных, можно передать секретную информацию. Сообщения, не принадлежащие ни к одной группе равновероятных сообщений, для кодирования секретной информации не используются. Источники, порождающие независимые одинаково распределенные сообщения — всего лишь один простой и содержательный пример кодирования такого рода.

# ОТВЕТЫ К ЗАДАЧАМ И УПРАЖНЕНИЯМ

**1.1.** а.  $k = 17$ . б.  $k = 27$ .

**1.2.** а. ПРИВЕТ ( $k = 5$ ). б. ВЕЧНА ( $k = 20$ ).

**2.1.** а.  $5 \equiv 5$ ,  $16 \equiv 6$ ,  $27 \equiv 7$ ,  $-4 \equiv 6$ ,  $-13 \equiv -3 \equiv 7$ ,  $3+8 \equiv 1$ ,  $3-8 \equiv 5$ ,  $3 \cdot 8 \equiv 4$ ,  $3 \cdot 8 \cdot 5 \equiv 4 \cdot 5 \equiv 0 \pmod{10}$ . б.  $5 \equiv 5$ ,  $16 \equiv 5$ ,  $27 \equiv 5$ ,  $-4 \equiv 7$ ,  $-13 \equiv -2 \equiv 9$ ,  $3+8 \equiv 0$ ,  $3-8 \equiv 6$ ,  $3 \cdot 8 \equiv 2$ ,  $3 \cdot 8 \cdot 5 \equiv 2 \cdot 5 \equiv 10 \pmod{11}$ .

**2.2.**  $2^8 \pmod{10} \equiv 6$ ,  $3^7 \pmod{10} \equiv 7$ ,  $7^{19} \pmod{100} \equiv 43$ ,  $7^{57} \pmod{100} \equiv 7$ .

**2.3.**  $108 = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 3$ ,  $77 = 7 \cdot 11$ ,  $65 = 5 \cdot 13$ ,  $30 = 3 \cdot 3 \cdot 5$ ,  $159 = 3 \cdot 53$ .

**2.4.** пары  $(25, 12)$  и  $(40, 27)$  взаимно просты, другие — нет (числа  $(25, 15)$  делятся на 5,  $(13, 39)$  делятся на 13).

**2.5.**  $\varphi(14) = 6$ ,  $\varphi(20) = 8$ .

**2.6.**  $\varphi(53) = 52$ ,  $\varphi(21) = \varphi(7) \cdot \varphi(3) = 6 \cdot 2 = 12$ ,  $\varphi(159) = 2 \cdot 52 = 104$ .

**2.7.**  $3^{13} \pmod{13} \equiv 3 \cdot 3^{12} \pmod{13} \equiv 3$ ,  $5^{22} \pmod{11} \equiv 5^2 \cdot 5^{10} \cdot 5^{10} \pmod{11} \equiv 25 \pmod{11} \equiv 3$ ,  $3^{17} \pmod{5} \equiv 3$ .

**2.8.**  $3^9 \pmod{20} \equiv 3 \cdot 3^8 \pmod{20} \equiv 3$ ,  $2^{14} \pmod{21} \equiv 2^2 \cdot 2^{12} \pmod{21} \equiv 4$ ,  $2^{107} \pmod{159} \equiv 2^3 \cdot 2^{104} \pmod{159} \equiv 8$ .

**2.9.**  $\gcd(21, 12) = 3$ ,  $\gcd(30, 12) = 6$ ,  $\gcd(24, 40) = \gcd(40, 24) = 8$ ,  $\gcd(33, 16) = 1$ .

**2.10.** а.  $x = -1$ ,  $y = 2$ . б.  $x = 1$ ,  $y = -2$ . в.  $x = 2$ ,  $y = -1$ . г.  $x = 1$ ,  $y = -2$ .

**2.11.**  $3^{-1} \pmod{7} \equiv 5$ ,  $5^{-1} \pmod{8} \equiv 5$ ,  $3^{-1} \pmod{53} \equiv 18$ ,  $10^{-1} \pmod{53} \equiv 16$ .

**2.12.** Простые числа, меньшие 100: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 73, 79, 83, 89, 97. Из них числа 5, 7, 11, 23, 47, 59 и 83 соответствуют виду  $p = 2q + 1$ .

**2.13.** При  $p = 11$  в качестве параметра  $g$  могут быть выбраны числа 2, 6, 7 и 8.

**2.14.** а.  $Y_A = 20$ ,  $Y_B = 17$ ,  $Z_{AB} = 21$ . б.  $Y_A = 13$ ,  $Y_B = 14$ ,  $Z_{AB} = 10$ . в.  $Y_A = 21$ ,  $Y_B = 9$ ,  $Z_{AB} = 16$ . г.  $Y_A = 8$ ,  $Y_B = 5$ ,  $Z_{AB} = 9$ . д.  $Y_A = 6$ ,  $Y_B = 17$ ,  $Z_{AB} = 16$ .

**2.15.** а.  $d_A = 11$ ,  $d_B = 13$ ,  $x_1 = 17$ ,  $x_2 = 5$ ,  $x_3 = 6$ ,  $x_4 = 4$ .

б.  $d_A = 3$ ,  $d_B = 19$ ,  $x_1 = 8$ ,  $x_2 = 12$ ,  $x_3 = 3$ ,  $x_4 = 6$ . в.  $d_A = 5$ ,  $d_B = 11$ ,  $x_1 = 14$ ,  $x_2 = 10$ ,  $x_3 = 3$ ,  $x_4 = 10$ . г.  $d_A = 5$ ,  $d_B = 15$ ,  $x_1 = 7$ ,  $x_2 = 21$ ,  $x_3 = 14$ ,  $x_4 = 17$ . д.  $d_A = 11$ ,  $d_B = 5$ ,  $x_1 = 15$ ,  $x_2 = 2$ ,  $x_3 = 8$ ,  $x_4 = 9$ .

**2.16.** а.  $d_B = 13$ ,  $r = 14$ ,  $e = 12$ ,  $m' = 5$ . б.  $d_B = 16$ ,  $r = 9$ ,  $e = 15$ ,  $m' = 10$ . в.  $d_B = 15$ ,  $r = 16$ ,  $e = 14$ ,  $m' = 10$ . г.  $d_B = 21$ ,  $r = 14$ ,  $e = 12$ ,  $m' = 5$ . д.  $d_B = 8$ ,  $r = 5$ ,  $e = 5$ ,  $m' = 10$ .

**2.17.** а.  $N_A = 55$ ,  $\varphi(N_A) = 40$ ,  $c_A = 27$ ,  $e = 23$ ,  $m' = 12$ .

б.  $N_A = 65$ ,  $\varphi(N_A) = 48$ ,  $c_A = 29$ ,  $e = 50$ ,  $m' = 20$ . в.  $N_A = 77$ ,  $\varphi(N_A) = 60$ ,  $c_A = 43$ ,  $e = 52$ ,  $m' = 17$ . г.  $N_A = 91$ ,  $\varphi(N_A) = 72$ ,  $c_A = 29$ ,  $e = 88$ ,  $m' = 30$ . д.  $N_A = 33$ ,  $\varphi(N_A) = 20$ ,  $c_A = 7$ ,  $e = 9$ ,  $m' = 15$ .

**2.18.**  $m = 111$ .

3.1. а.  $x = 17$ . б.  $x = 10$ . в.  $x = 28$ . г.  $x = 14$ . д.  $x = 30$ .

3.2. а.  $x = 20$ . б.  $x = 45$ . в.  $x = 34$ . г.  $x = 53$ . д.  $x = 25$ .

3.3. а.  $x = 10000$ . б.  $x = 20000$ . в.  $x = 1000$ . г.  $x = 12345$ . д.  $x = 25000$ .

**4.1.** а.  $s = 28$ . б.  $s = 30$ . в.  $s = 26$ . г.  $s = 71$ . д.  $s = 18$ .

**4.2.** а.  $\langle 7, 28 \rangle$  подлинно,  $\langle 22, 15 \rangle$  нет,  $\langle 16, 36 \rangle$  подлинно. б.  $\langle 6, 42 \rangle$  нет,  $\langle 10, 30 \rangle$  да,  $\langle 6, 41 \rangle$  да. в.  $\langle 13, 41 \rangle$  да,  $\langle 11, 28 \rangle$  нет,  $\langle 5, 26 \rangle$  да. г.  $\langle 15, 71 \rangle$  да,  $\langle 11, 46 \rangle$  нет,  $\langle 16, 74 \rangle$  да. д.  $\langle 10, 14 \rangle$  нет,  $\langle 24, 18 \rangle$  да,  $\langle 17, 8 \rangle$  да.

**4.3.** а.  $y = 22$ ,  $r = 10$ ,  $u = 15$ ,  $k^{-1} = 15$ ,  $s = 5$ . б.  $y = 9$ ,  $r = 19$ ,  $u = 13$ ,  $k^{-1} = 3$ ,  $s = 17$ . в.  $y = 10$ ,  $r = 21$ ,  $u = 11$ ,  $k^{-1} = 17$ ,  $s = 11$ . г.  $y = 6$ ,  $r = 17$ ,  $u = 7$ ,  $k^{-1} = 19$ ,  $s = 1$ . д.  $y = 11$ ,  $r = 7$ ,  $u = 18$ ,  $k^{-1} = 7$ ,  $s = 16$ .

**4.4.** а.  $\langle 15; 20, 3 \rangle$  да ( $y^r = 1$ ,  $r^s = 19$ ,  $g^h = 19$ ),  $\langle 15; 10, 5 \rangle$  да

$(y^r = 1, r^s = 19, g^h = 19)$ ,  $\langle 15; 19, 3 \rangle$  нет ( $y^r = 22, r^s = 5, g^h = 19 \neq 18$ ). **6.**  $\langle 5; 19, 17 \rangle$  да ( $y^r = 13, r^s = 21, g^h = 20$ ),  $\langle 7; 17, 8 \rangle$  нет ( $y^r = 3, r^s = 18, g^h = 17 \neq 8$ ),  $\langle 6; 17, 8 \rangle$  да ( $y^r = 3, r^s = 18, g^h = 8$ ). **в.**  $\langle 3; 17, 12 \rangle$  да ( $y^r = 17, r^s = 6, g^h = 10$ ),  $\langle 2; 17, 12 \rangle$  нет ( $y^r = 17, r^s = 6, g^h = 2 \neq 10$ ),  $\langle 8; 21, 11 \rangle$  да ( $y^r = 7, r^s = 22, g^h = 16$ ). **г.**  $\langle 5; 17, 1 \rangle$  да ( $y^r = 12, r^s = 17, g^h = 20$ ),  $\langle 5; 11, 3 \rangle$  да ( $y^r = 1, r^s = 20, g^h = 20$ ),  $\langle 5; 17, 10 \rangle$  нет ( $y^r = 12, r^s = 4, g^h = 20 \neq 2$ ). **д.**  $\langle 15; 7, 1 \rangle$  нет ( $y^r = 7, r^s = 7, g^h = 19 \neq 3$ ),  $\langle 10; 15, 3 \rangle$  да ( $y^r = 10, r^s = 17, g^h = 9$ ),  $\langle 15; 7, 16 \rangle$  да ( $y^r = 7, r^s = 6, g^h = 19$ ).

**4.5.** **а.**  $y = 14, r = 3, s = 8$ . **б.**  $y = 24, r = 3, s = 5$ . **в.**  $y = 40, r = 9, s = 2$ . **г.**  $y = 22, r = 9, s = 5$ . **д.**  $y = 64, r = 7, s = 10$ .

**4.6.** **а.**  $\langle 10; 4, 5 \rangle$  нет ( $h^{-1} = 10, u_1 = 6, u_2 = 4, a^{u_1} = 62, y^{u_2} = 25, v = 9 \neq 4$ ),  $\langle 10; 7, 5 \rangle$  да ( $h^{-1} = 10, u_1 = 6, u_2 = 7, a^{u_1} = 62, y^{u_2} = 59, v = 7$ ),  $\langle 10; 3, 8 \rangle$  да ( $h^{-1} = 10, u_1 = 3, u_2 = 3, a^{u_1} = 14, y^{u_2} = 64, v = 3$ ). **б.**  $\langle 1; 3, 5 \rangle$  да ( $h^{-1} = 1, u_1 = 5, u_2 = 8, a^{u_1} = 40, y^{u_2} = 64, v = 3$ ),  $\langle 1; 4, 3 \rangle$  да ( $h^{-1} = 1, u_1 = 3, u_2 = 7, a^{u_1} = 14, y^{u_2} = 25, v = 4$ ),  $\langle 1; 4, 5 \rangle$  нет ( $h^{-1} = 1, u_1 = 5, u_2 = 7, a^{u_1} = 40, y^{u_2} = 25, v = 7 \neq 4$ ). **в.**  $\langle 7; 7, 4 \rangle$  да ( $h^{-1} = 8, u_1 = 10, u_2 = 10, a^{u_1} = 59, y^{u_2} = 62, v = 7$ ),  $\langle 7; 9, 2 \rangle$  нет ( $h^{-1} = 8, u_1 = 5, u_2 = 5, a^{u_1} = 40, y^{u_2} = 14, v = 2 \neq 9$ ),  $\langle 5; 9, 2 \rangle$  да ( $h^{-1} = 9, u_1 = 7, u_2 = 7, a^{u_1} = 9, y^{u_2} = 22, v = 9$ ). **г.**  $\langle 6; 9, 5 \rangle$  да ( $h^{-1} = 2, u_1 = 10, u_2 = 4, a^{u_1} = 59, y^{u_2} = 24, v = 9$ ),  $\langle 8; 8, 3 \rangle$  нет ( $h^{-1} = 7, u_1 = 10, u_2 = 10, a^{u_1} = 59, y^{u_2} = 64, v = 2 \neq 8$ ),  $\langle 7; 4, 1 \rangle$  да ( $h^{-1} = 8, u_1 = 8, u_2 = 1, a^{u_1} = 24, y^{u_2} = 22, v = 4$ ). **д.**  $\langle 10; 7, 3 \rangle$  да ( $h^{-1} = 10, u_1 = 8, u_2 = 7, a^{u_1} = 24, y^{u_2} = 24, v = 7$ ),  $\langle 7; 7, 10 \rangle$  да ( $h^{-1} = 8, u_1 = 3, u_2 = 10, a^{u_1} = 14, y^{u_2} = 22, v = 7$ ),  $\langle 8; 7, 5 \rangle$  нет ( $h^{-1} = 7, u_1 = 2, u_2 = 6, a^{u_1} = 22, y^{u_2} = 59, v = 3 \neq 7$ ).

**5.1.** **а.**  $d_A = 17, d_B = 9$ , Алиса получает  $\gamma$ , Боб —  $\beta$ ; по каналу связи передаются числа  $(11, 20, 21), (11), (14, 10), (17)$ . **б.**  $d_A = 19, d_B = 3$ , Алиса получает  $\gamma$ , Боб —  $\alpha$ ; по каналу связи передаются числа  $(17, 19, 5), (19), (15, 19), (19)$ . **в.**  $d_A = 7, d_B = 15$ , Алиса получает  $\alpha$ , Боб —  $\beta$ ; по каналу связи передаются числа  $(11, 7, 10), (7), (11, 20), (21)$ . **г.**  $d_A = 5, d_B = 19$ , Алиса получает  $\alpha$ , Боб —  $\beta$ ; по каналу связи передаются числа  $(21, 15, 11), (11), (10, 11), (5)$ . **д.**  $d_A = 3, d_B = 9$ , Алиса получает  $\alpha$ , Боб —  $\gamma$ ; по каналу

связи передаются числа (19, 14, 17), (19), (21, 15), (15).

**5.2.** **а.**  $\hat{n} = 103$ ,  $\hat{s} = 52$ ,  $r^{-1} = 24$ , банкнота  $\langle 11, 58 \rangle$ . **б.**  $\hat{n} = 13$ ,  $\hat{s} = 13$ ,  $r^{-1} = 20$ , банкнота  $\langle 99, 22 \rangle$ . **в.**  $\hat{n} = 58$ ,  $\hat{s} = 74$ ,  $r^{-1} = 12$ , банкнота  $\langle 55, 55 \rangle$ . **г.**  $\hat{n} = 37$ ,  $\hat{s} = 46$ ,  $r^{-1} = 8$ , банкнота  $\langle 44, 11 \rangle$ . **д.**  $\hat{n} = 49$ ,  $\hat{s} = 70$ ,  $r^{-1} = 4$ , банкнота  $\langle 77, 42 \rangle$ .

**6.1.** Из указанного списка кривой принадлежат только точки  $(1,1)$ ,  $(2,1)$  и  $(5,8)$ .

**6.2.**  $[2](2,2) = (3,5)$ ,  $[2](4,6) = (1,3)$ ,  $(1,3) + (1,4) = \mathcal{O}$ ,  $(2,2) + (3,2) = (2,5)$ ,  $(3,5) + (5,1) = (3,2)$ .

**7.1.** **а.**  $\bar{e} = 1111001110$ . **б.**  $\bar{e} = 1111110101$ . **в.**  $\bar{e} = 0001000110$ . **г.**  $\bar{e} = 0101011011$ . **д.**  $\bar{e} = 0001010001$ .

**7.2.** **а.**  $P_1 \approx 0.002$ ,  $P_2 \approx 0.006$ ,  $P_3 \approx 0.623$ ,  $P_4 \approx 0.051$ ,  $P_5 \approx 0.311$ ,  $P_6 \approx 0.007$ . **б.**  $P_1 \approx 0.000$ ,  $P_2 \approx 0.009$ ,  $P_3 \approx 0.000$ ,  $P_4 \approx 0.000$ ,  $P_5 \approx 0.892$ ,  $P_6 \approx 0.099$ . **в.**  $P_1 \approx 0.000$ ,  $P_2 \approx 0.697$ ,  $P_3 \approx 0.000$ ,  $P_4 \approx 0.004$ ,  $P_5 \approx 0.299$ ,  $P_6 \approx 0.000$ . **г.**  $P_1 \approx 0.003$ ,  $P_2 \approx 0.000$ ,  $P_3 \approx 0.036$ ,  $P_4 \approx 0.000$ ,  $P_5 \approx 0.801$ ,  $P_6 \approx 0.160$ . **д.**  $P_1 \approx 0.196$ ,  $P_2 \approx 0.000$ ,  $P_3 \approx 0.001$ ,  $P_4 \approx 0.000$ ,  $P_5 \approx 0.018$ ,  $P_6 \approx 0.785$ .

**7.3.** **а.**  $H \approx 1.16$ ,  $n \approx 6.04$ . **б.**  $H \approx 0.52$ ,  $n \approx 2.42$ . **в.**  $H \approx 0.9$ ,  $n \approx 3.76$ . **г.**  $H \approx 1.08$ ,  $n \approx 5.08$ . **д.**  $H \approx 1.16$ ,  $n \approx 6.04$ .

**7.4.** **а.**  $P_1 \approx 0.7$  ( $\bar{m} = bcacbcacc$ ),  $P_2 = 0$ ,  $P_3 \approx 0.3$  ( $\bar{m} = acbcacbcc$ ),  $P_4 = 0$ ,  $P_5 = 0$ ,  $P_6 = 0$ . **б.**  $P_1 = 0$ ,  $P_2 = 0$ ,  $P_3 = 0$ ,  $P_4 \approx 0.21$  ( $\bar{m} = bcccaccac$ ),  $P_5 \approx 0.20$  ( $\bar{m} = abbbcbcb$ ),  $P_6 \approx 0.59$  ( $\bar{m} = acccbccbc$ ). **в.**  $P_1 = 0$ ,  $P_2 = 0$ ,  $P_3 = 0$ ,  $P_4 = 1$  ( $\bar{m} = ccbcabcbb$ ),  $P_5 = 0$ ,  $P_6 = 0$ . **г.**  $P_1 = 0$ ,  $P_2 = 0$ ,  $P_3 \approx 0.000$  ( $\bar{m} = acbbbbccb$ ),  $P_4 \approx 1.000$  ( $\bar{m} = abccccbcc$ ),  $P_5 = 0$ ,  $P_6 = 0$ . **д.**  $P_1 = 0$ ,  $P_2 = 0$ ,  $P_3 \approx 0.009$  ( $\bar{m} = bbbcbccb$ ),  $P_4 \approx 0.970$  ( $\bar{m} = cccbcccac$ ),  $P_5 = 0$ ,  $P_6 \approx 0.021$  ( $\bar{m} = cccacccac$ ).

## СПИСОК ЛИТЕРАТУРЫ

1. **Ахо А., Хопкрофт Дж., Ульман Дж.** Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 383 с.
2. **Введение в криптографию /** Под общ. ред. В. В. Ященко. – М.: МЦНМО: «ЧеРо», 2000. – 287 с.
3. **Виноградов И. М.** Основы теории чисел. – М.: Наука, 1972. – 402 с.
4. **Галлагер Р.** Теория информации и надежная связь. – М.: Советское радио, 1974. – 425 с.
5. **ГОСТ 28147-89.** Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования данных. – Введ. 30.06.1990. – М.: Изд-во стандартов, 1990.
6. **ГОСТ Р 34.10-94.** Информационная технология. Криптографическая защита информации. Процедуры выработки и проверки электронной цифровой подписи на базе асимметричного криптографического алгоритма. – Введ. 01.01.1995. – М.: Госстандарт России, 1995.
7. **ГОСТ Р 34.11-94.** Информационная технология. Криптографическая защита информации. Функция хэширования. – Введ. 01.01.1995. – М.: Госстандарт России, 1995.
8. **ГОСТ Р 34.10-2001.** Процессы формирования и проверки электронной цифровой подписи. – Введ. 01.07.2002. – М.: Госстандарт России, 2001.
9. **ГОСТ Р 34.10-2012.** Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. – Введ. 01.01.2013. – М.: Стандартинформ, 2012.

10. ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Функция хэширования. – Введ. 01.01.2013. – М.: Стандартинформ, 2012.
11. ГОСТ Р 34.12-2015 Информационная технология. Криптографическая защита информации. Блочные шифры. – Введ. 01.01.2016. – М.: Стандартинформ, 2015.
12. Грибунин В. Г., Оков И. Н., Туринцев И. В. Цифровая стеганография. – Солон-Пресс, 2002.
13. Кнут Д. Искусство программирования для ЭВМ. В 3-х томах. Т. 2. Получисленные алгоритмы. – М.: Мир, 1977. – 724 с.
14. Лысяк А. С., Рябко В. Я., Фионов А. Н. Анализ эффективности градиентной статистической атаки на блоковые шифры RC6, MARS, CAST-128, IDEA, Blowfish в системах защиты информации // Вестник СибГУТИ. – 2013. – № 1. – С. 85–109.
15. Монарев В.А., Фионов А.Н., Шокин Ю.И. Обзор современных теоретико-информационных подходов к решению основных задач криптографии и стеганографии // Вычислительные технологии. – 2010. – Т. 15, № 2. – С. 69–86
16. Рябко Б. Я. Сжатие данных с помощью стопки книг // Проблемы передачи информации. – 1980. – Т. 16, № 4. – С. 16–21.
17. Рябко Б. Я. Быстрая нумерация комбинаторных объектов // Дискретная математика. – 1998. – Т. 10, № 2. – С. 101–119.
18. Рябко Б. Я. Просто реализуемая идеальная криптографическая система // Проблемы передачи информации. – 2000. – Т. 36, № 1. – С. 90–104.
19. Рябко Б. Я. Шифр Вернама устойчив к небольшим отклонениям от случайности // Проблемы передачи информации. – 2015. – Т. 51, № 1. – С. 90–95.
20. Рябко Б. Я., Монарев В. А., Фионов А. Н., Шокин Ю. И. Градиентная статистическая атака на блоковые шифры // НТК "Математика и безопасность информационных технологий" (МАБИТ-2005). Москва, МГУ, 2005. С. 43–48.

21. Рябко Б. Я., Пестунов А. И. «Стопка книг» как новый статистический тест для случайных чисел // Проблемы передачи информации. – 2004. – Т. 40, № 1. – С. 73–78.
22. Рябко Б. Я., Рябко Д. Б. Асимптотически оптимальные совершенные стеганографические системы // Проблемы передачи информации. – 2009. – Т. 45, № 2. – С. 119–126.
23. Рябко Б. Я., Стогниенко В. С., Шокин Ю. И. Адаптивный критерий хи-квадрат для различения близких гипотез при большом числе классов и его применение к некоторым задачам криптографии // Проблемы передачи информации. – 2003. – Т. 30, № 2. – С. 53–62.
24. Рябко Б. Я., Фионов А. Н. Быстрый метод полной рандомизации сообщений // Проблемы передачи информации. – 1997. – Т. 33, № 3. – С. 3–14.
25. Рябко Б. Я., Фионов А. Н. Эффективный метод адаптивного арифметического кодирования для источников с большими алфавитами // Проблемы передачи информации. – 1999. – Т. 35, № 4. – С. 1–14.
26. Феллер В. Введение в теорию вероятностей и ее приложения. В 2-х томах. – М.: Мир, 1984. – Т 1. – 527 с.
27. Фионов А. Н. Эффективный метод рандомизации сообщений на основе арифметического кодирования // Дискретный анализ и исследование операций. – 1997. – Т. 4, № 2. – С. 51–74.
28. Шеннон К. Работы по теории информации и кибернетике. – М.: ИЛ, 1963. – С. 333–369 (Теория связи в секретных системах).
29. Anderson R.J., Petricolas F.A. On the limits of steganography // Journal of Selected Areas in Communications. – 1998. – V. 16, N. 4. – P. 474–481.
30. Anckaert B., De Sutter B., Chanet D., De Bosschere K. Steganography for executables and code transformation signatures // C. Park and S. Chee (Eds.): ICISC 2004. – Berlin, Heidelberg: Springer-Verlag. – 2005. – LNCS, V. 3506. – P. 431–445.

31. **Aumasson J.-Ph., Henzen L., Meier W., Phan R. C.-W.** SHA-3 proposal BLAKE – <https://131002.net/blake/>.
32. **Aumasson J.-Ph., Neves S., Wilcox-O'Hearn Z., Winnerlein Ch.** BLAKE2 – fast secure hashing – <https://blake2.net/>.
33. **Back A.** Hashcash – a denial of service counter-measure. <ftp://sunsite.icm.edu.pl/site/replay.old/programs/hashcash/hashcash.pdf> – 2002.
34. **Bayer D., Haber S., Stornetta W. S.** Improving the efficiency and reliability of digital time-stamping // Sequences II. – Springer, New York. – 1993. – P. 329–334.
35. **Bernstein D. J.** ChaCha, a variant of Salsa20 – <https://cr.yp.to/chacha/chacha-20080128.pdf>.
36. **Bertoni G., Daemen J., Peeters M., Van Assche G.** Sponge functions // Ecrypt Hash Workshop, 2007. – <https://keccak.team/files/SpongeFunctions.pdf>.
37. **Bertoni G., Daemen J., Peeters M., Van Assche G.** The Keccak reference – SHA-3 competition (round 3), 2011. – <https://keccak.team/files/Keccak-reference-3.0.pdf>.
38. **Blake I., Seroussi G., Smart N.** Elliptic Curves in Cryptography. – Cambridge University Press, 1999. – 204 p.
39. **Biham E., Dunkelman O.** A Framework for Iterative Hash Functions – HAIFA // Cryptology ePrint Archive. – 2007 (278). – <https://eprint.iacr.org/2007/278.pdf>.
40. **Biryukov A., Perrin L., Udovenko A.** Reverse-engineering the S-box of Salsa20, Kuznyechik and Salsa20/SHA3 // Cryptology ePrint Archive. – 2016 (071). – <https://eprint.iacr.org/2016/071.pdf>.
41. **Borodin M., Rybkin A., Urivskiy A.** High-speed software implementation of the prospective 128-bit block cipher and Salsa20 hash-function // 3rd Workshop on Current Trends in Cryptology (CTCrypt 2014), Moscow, Russia, June 5–6, 2014.

42. **Cachin C.** An information-theoretic model for steganography // 2nd Information Hiding Workshop, 1998. – Springer Verlag. – LNCS V. 1525. – P. 306–318.
43. **Cover T. M., Thomas J. A.** Elements of Information Theory, Wiley-Interscience, New York, NY, USA: 2006.
44. **Crowley P.** Small bias in RC4 experimentally verified. – 2003. – <http://www.ciphergoth.org/crypto/rc4/>.
45. **Csiszar I.** The method of types // IEEE Transactions on Information Theory. – 1998. – V. 44, N. 6. – P. 2505–2523.
46. **Dabeer O., Sullivan K., Madhow U., Chandrasekaran S., Manjunath B. S.** Detection of hiding in the least significant bit // IEEE Transactions on Signal Processing. – 2004. – V. 52. – P. 346–358.
47. **Daemen J., Rijmen V.** The Rijndael Block Cipher – <http://csrc.nist.gov/encryption/aes/rijndael/>.
48. **Difflie W., Hellman M. E.** New directions in cryptography // IEEE Transactions on Information Theory. – 1976. – V. 22. – P. 644–654.
49. **Dobbertin H., Bosselaers A., Preneel B.** RIPEMD-160: A Strengthened Version of RIPEMD // Proceedings of FSE. – LNCS, V. 1039. – P. 71–82. Springer, 1996.
50. **Doroshenko S., Fionov A., Lubkin A., Monarev V., Ryabko B., Shokin Yu. I.** Experimental statistical attacks on block and stream ciphers // 3rd Russian-German Advanced Research Workshop on Computational Science and High Performance Computing. Novosibirsk, July 23–27, 2007. Springer, 2007 (NNFM; V. 101). P. 155–164.
51. **Dwork C., Naor M.** Pricing via processing or combatting junk mail // Advances in Cryptology – CRYPT0'92. – LNCS 740. – Springer-Verlag, 1993. – P. 139–147.
52. **Elias P.** The efficient construction of an unbiased random sequence // The Annals of Math. Statistics. – 1972. – V. 43. N. 3. – P. 864–870.

53. **eSTREAM:** the ECRYPT Stream Cipher Project. – <http://www.ecrypt.eu.org/stream/>.
54. **FIPS PUB 180-1.** Secure hash standard – 1995 – <http://csrc.nist.gov/publications/>.
55. **FIPS PUB 180-4.** Secure hash standard (SHS) – 2015 – <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.
56. **FIPS PUB 186-1.** Digital signature standard – 1998 – <http://csrc.nist.gov/publications/>.
57. **FIPS PUB 186-4.** Digital signature standard – 2013 – <http://csrc.nist.gov/publications/>.
58. **FIPS PUB 197.** Advanced encryption standard – 2001 – <http://csrc.nist.gov/publications/>.
59. **Fridrich J., Goljan M.** Practical steganalysis — state of the art // Proc. SPIE Photonics Imaging 2002, Security and Watermarking of Multimedia Contents. – SPIE Press, 2002. – V. 4675. – P. 11-13.
60. **Gallager R. G.** Information Theory and Reliable Communication, John Wiley & Sons, New York, 1968.
61. **Goldwasser S., Bellare M.** Lecture notes on cryptography – <http://www-cse.ucsd.edu/users/mihir/crypto-lectnotes.html>
62. **Guo J., Jean J., Leurent G., Peyrin Th., Wang L.** The usage of counter revisited: second-preimage attack on new Russian standardized hash function // Cryptology ePrint Archive. – 2014 (675). – <https://eprint.iacr.org/2014/675.pdf>.
63. **Haber S., Stornetta W. S.** How to time-stamp a digital document // Journal of Cryptology. – 1991. – V. 3, N. 2. – P. 99–111.
64. **Ker A. D., Lubenko I.** Feature reduction and payload location with WAM steganalysis // Media Forensics and Security. Proceedings of the SPIE. – 2009. – V. 7254. – 13 p.
65. **Kharrazi M., Sencar H. T., Memon N.** Image steganography: concepts and practice // Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore. – Singapore, 2004.

66. **Lyu S., Farid H.** Steganalysis using higher-order image statistics // IEEE Transactions on Information Forensics and Security. – 2006. – V. 1, N. 1. – P. 111–119.
67. **Menezes A.** Elliptic Curve Public Key Cryptosystems. – Kluwer Academic Publishers, 1993.
68. **Menezes A., van Oorschot P., Vanstone S.** Handbook of Applied Cryptography. – CRC Press, 1996. – 661 p. – <http://www.cacr.math.uwaterloo.ca/hac/>.
69. **Moulin P., O'Sullivan J. A.** Information-theoretic analysis of information hiding // IEEE Transactions on Information Theory. – 2003. – V. 49, N. 3. – P. 563–593.
70. **Nakamoto S.** Bitcoin: a peer-to-peer electronic cash system. – 2008. – <https://bitcoin.org/bitcoin.pdf>.
71. **Narayanan A., Bonneau J., Felten E., Miller A., Goldfeder S.** Bitcoin and cryptocurrency technologies: a comprehensive introduction. – Princeton University Press, 2016.
72. **Nechta I., Ryabko B., Fionov A.** Stealthy steganographic methods for executable files // XII International Symposium on Problems of Redundancy. – St.-Petersburg, May 26-30, 2009. – P. 191–195.
73. **Petitcolas F. A. P., Anderson R. J., Kuhn M. G.** Information hiding – a survey // Proceedings of the IEEE. – 1999. – V. 7, N. 7. – P. 1062–1078.
74. **Preneel B., Chaum D., Fumy W., Jansen C. J. A., Landrock P., Roelofsen G.** Race Integrity Primitives Evaluation (RIPE): A status report // Advances in Cryptology – EUROCRYPT '91. – LNCS, V 547. – Springer, 1991.
75. **Rivest R.** The MD5 message-digest algorithm // Request for Comments: 1321. – 1992. – <https://tools.ietf.org/html/rfc1321>.
76. **Rukhin A. et al.** A statistical test suite for random and pseudorandom number generators for cryptographic applications

- // NIST Special Publication 800-22 (rev. May, 15, 2001). – <http://csrc.nist.gov/rng/SP800-22b.pdf>.
77. **Ryabko B.** Properties of two Shannon's ciphers // *Designs, Codes and Cryptography*. – 2017. – P.1-7. – ISSN 0925-1022. – EISSN 1573-7586.
78. **Ryabko B., Fionov A.** Efficient homophonic coding // *IEEE Transactions on Information Theory*. – 1999. – V. 45, N. 6. – P. 2083–2091.
79. **Ryabko B., Fionov A.** Fast and space-efficient adaptive arithmetic coding // *Cryptography and Coding*. – Berlin: Springer, 1999. – P. 270–279 (Lecture Notes in Computer Science; V. 1746).
80. **Ryabko B., Fionov A., Monarev V., Shokin Yu.** Using information theory approach to randomness testing // 2nd Russian-German Advanced Research Workshop on Computational Science and High Performance Computing. Stuttgart, Germany, March 14–16, 2005. Springer, 2005 (NNFM; V. 91). P. 261–272.
81. **Ryabko B., Matchikina E.** Fast and efficient construction of an unbiased random sequence // *IEEE Transactions on Information Theory*. – 2000. – V. 46, N 3. – P. 1090–1093.
82. **Ryabko B. Ya., Monarev V. A.** Using information theory approach to randomness testing // *Journal of Statistical Planning and Inference*. – 2005.
83. **Ryabko B., Ryabko D.** Information-theoretic approach to steganographic systems // *IEEE International Symposium on Information Theory*, Nice, France, 2007. – P. 2461–2464.
84. **Ryabko B., Ryabko D.** Confidence sets in time-series filtering // *IEEE International Symposium on Information Theory*, Saint-Petersburg, Russia, 2011.
85. **Ryabko B., Ryabko D.** Constructing perfect steganographic systems // *Information and Computation*. – 2011. – V. 209. – P. 1223–1230.
86. **Schneier B.** *Applied Cryptography*, Second Edition: Protocols, Algorithms, and Source Code in C. – Wiley, 1996.

87. **Schneier B.** Self-study course in block cipher cryptanalysis // *Cryptologia*. – 2000. – V. 24, N. 1. – P. 18–34. – <http://www.counterpane.com/self-study.html>.
88. **Shannon C. E.** Communication theory of secrecy systems // *Bell System Technical Journal*. – 1949. – V. 28, N. 4. – P. 656–715.
89. **Shannon C. E.** Prediction and entropy of printed English // *Bell System Technical Journal*. – 1951. – V. 30, N. 1. – P. 50–64.
90. **Shishkin V., Dygin D., Lavrikov I., Marshalko G., Rudskoy V., Trifonov D.** Low-weight and hi-end: draft Russian encryption standard // 3rd Workshop on Current Trends in Cryptology (CTCrypt 2014). – June 5–6, 2014. – Moscow, Russia.
91. **Silverman J. H.** *The Arithmetic of Elliptic Curves*. – Springer-Verlag, GTM 106, 1986.
92. **Takahira R., Tanaka-Ishii K., Dębowksi L.** Entropy rate estimates for natural language – a new extrapolation of compressed large-scale corpora // *Entropy*. – 2016. – V. 18, N. 10. – P. 364.
93. **The MD6 Hash Algorithm** – <http://groups.csail.mit.edu/cis/md6/>.
94. **The Whirlpool Hash Function** – <http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html>.
95. **von Neumann J.** Various techniques used in connection with random digits // *Monte Carlo Method, Applied Mathematics Series*. – 1951. – N. 12. – P. 36–38.
96. **Wang X., Feng D., Lai X., Yu H.** Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD // *Cryptology ePrint Archive*. – 2004 (199). – <https://eprint.iacr.org/2004/199.pdf>.
97. **Wu H.** The Stream Cipher HC-128. – [http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128\\_p3.pdf](http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128_p3.pdf).
98. **Zhilkin M., Melentsova N., Ryabko B.** Data compression based method of revealing hidden information in steganographic systems // XI International Symposium on Problems of Redundancy in Information and Control Systems, Saint-Petersburg, 2007. – P. 42–44.

# ОГЛАВЛЕНИЕ

<b>Предисловие</b>	<b>3</b>
<b>1. Введение</b>	<b>4</b>
Задачи и упражнения	10
<b>2. Крипtosистемы с открытым ключом</b>	<b>11</b>
2.1. Предыстория и основные идеи	11
2.2. Первая система с открытым ключом — система Диффи-Хеллмана	17
2.3. Элементы теории чисел	20
2.4. Шифр Шамира	27
2.5. Шифр Эль-Гамаля	30
2.6. Одностороння функция с «лазейкой» и шифр RSA	33
Задачи и упражнения	37
Темы лабораторных работ	39
<b>3. Методы взлома шифров, основанных на дискретном логарифмировании</b>	<b>40</b>
3.1. Постановка задачи	40
3.2. Метод «шаг младенца, шаг великана»	42
3.3. Алгоритм исчисления порядка	44
Задачи и упражнения	49
Темы лабораторных работ	50
<b>4. Электронная, или цифровая подпись</b>	<b>51</b>
4.1. Электронная подпись RSA	51
4.2. Электронная подпись на базе шифра Эль-Гамаля	54
4.3. Стандарты на электронную (цифровую) подпись	57
Задачи и упражнения	62
Темы лабораторных работ	63

<b>5. Криптографические протоколы . . . . .</b>	<b>65</b>
5.1. Ментальный покер . . . . .	65
5.2. Доказательства с нулевым знанием . . . . .	70
Задача о раскраске графа . . . . .	71
Задача о нахождении гамильтонова цикла в графе . . . . .	74
5.3. Электронные деньги . . . . .	82
5.4. Взаимная идентификация с установлением ключа . . . . .	88
Задачи и упражнения . . . . .	94
Темы лабораторных работ . . . . .	96
<b>6. Криптосистемы на эллиптических кривых . . . . .</b>	<b>97</b>
6.1. Введение . . . . .	97
6.2. Математические основы . . . . .	98
6.3. Выбор параметров кривой . . . . .	106
6.4. Построение криптосистем . . . . .	108
Шифр Эль-Гамаля на эллиптической кривой . . . . .	109
Цифровая подпись по ГОСТ Р 34.10-2012 . . . . .	110
Алгоритм ECDSA . . . . .	111
6.5. Эффективная реализация операций . . . . .	112
6.6. Определение количества точек на кривой . . . . .	118
6.7. Использование стандартных кривых . . . . .	127
Задачи и упражнения . . . . .	130
Темы лабораторных работ . . . . .	130
<b>7. Теоретическая стойкость криптосистем . . . . .</b>	<b>132</b>
7.1. Введение . . . . .	132
7.2. Теория систем с совершенной секретностью . . . . .	133
7.3. Шифр Вернама . . . . .	135
7.4. Элементы теории информации . . . . .	137
7.5. Устойчивость шифра Вернама к небольшим отклонениям ключа от случайности . . . . .	143
7.6. Шифры с бегущим ключом . . . . .	148
7.7. Расстояние единственности шифра с секретным ключом . . . . .	152
7.8. Идеальные криптосистемы . . . . .	158
Задачи и упражнения . . . . .	164

<b>8. Современные шифры с секретным ключом . . . . .</b>	<b>166</b>
8.1. Введение . . . . .	166
8.2. Блоковые шифры . . . . .	169
Шифр Магма . . . . .	171
Шифр RC6 . . . . .	174
Шифр Rijndael (AES) . . . . .	177
Шифр Кузнечик . . . . .	189
8.3. Основные режимы функционирования блоковых шифров . . . . .	195
Режим ECB . . . . .	196
Режим CBC . . . . .	197
8.4. Потоковые шифры . . . . .	197
Режим OFB блокового шифра . . . . .	199
Режим CTR блокового шифра . . . . .	201
Алгоритм RC4 . . . . .	201
Алгоритм HC-128 . . . . .	203
8.5. Криптографические хеш-функции . . . . .	206
<b>9. Криптовалюты и блокчейн . . . . .</b>	<b>216</b>
9.1. Введение . . . . .	216
9.2. Доказательство выполнения работы (proof-of-work) и Хэшкэш (Hashcash) . . . . .	216
9.3. Датирование документов (time-stamping) . . . . .	220
9.4. Блокчейн (blockchain) . . . . .	223
9.5. Биткоин (bitcoin) и криптовалюты . . . . .	227
Транзакции и биткоины . . . . .	228
Формирование бухгалтерской книги и производство биткоинов . . . . .	230
Надежность системы биткоин . . . . .	233
<b>10. Случайные числа в криптографии . . . . .</b>	<b>234</b>
10.1. Введение . . . . .	234
10.2. Задачи, возникающие при использовании физических генераторов случайных чисел . . . . .	236
10.3. Генераторы псевдослучайных чисел . . . . .	238
10.4. Тесты для проверки генераторов случайных и псевдо-случайных чисел . . . . .	241
10.5. Статистическая атака на блоковые шифры . . . . .	246
10.6. Атака различия на потоковые шифры . . . . .	258