EXPERT INSIGHT

SPRINT book

Алгоритмы криптографии

Bookauthority
BEST CRYPTOGRAPHY ALGORITHMS BOOKS OF ALL TIME
WINNER

Второе издание

Массимо Бертаччини

packt

Cryptography Algorithms

Second Edition

Explore New Algorithms in Zero-knowledge, Homomorphic Encryption, and Quantum Cryptography

Massimo Bertaccini

‹packt›

Алгоритмы криптографии

Второе издание

Массимо Бертаччини

SPRINT book 2026

Массимо Бертаччини

Алгоритмы криптографии

Перевел с английского С. Черников

Научный редактор Т. Квист

ББК 32.973.23-082.3
УДК 004.056.5+ 003.26

Бертаччини Массимо

Б26 Алгоритмы криптографии. — Астана: «Спринт Бук», 2026. — 352 с.: ил.

ISBN 978-601-12-6909-4

Изучите математическую логику шифрования и дешифрования сообщений, постепенно переходя от базовых принципов ко все более сложным концепциям. Освоив эллиптические кривые, протоколы с нулевым разглашением, гомоморфное шифрование и основы квантовых вычислений, вы заложите прочный фундамент для дальнейшего развития в криптографии.

Познакомьтесь с самыми инновационными криптографическими алгоритмами и подготовьтесь к работе в стремительно развивающейся сфере кибербезопасности и защиты данных. Уделяя особое внимание современным трендам и практическим вызовам, включая квантовую криптографию, вы получите знания, которые помогут оставаться на передовой этой динамичной области.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-1835080030 англ.

ISBN 978-601-12-6909-4

Изготовлено в России. Изготовитель: ТОО «Спринт Бук». Место нахождения и фактический адрес: 010000, Казахстан, город Астана, район Алматы, проспект Ракымжан Кошкарбаев, д. 10/1, н. п. 18.

Дата изготовления: 02.2026. Наименование: книжная продукция. Срок годности: не ограничен.

Подписано в печать 28.01.26. Формат 70x100/16. Бумага офсетная. Усл. п. л. 28,380. Тираж 700. Заказ 0000.

Краткое содержание

Оглавление

Часть II
Классическая криптография

Часть III
Новые криптографические алгоритмы и протоколы

Часть IV
Квантовая криптография

Моим маме и папе.
Массимо

Предисловие

В наш век огромных возможностей связи, облачных вычислений, программ-вымогателей и хакеров цифровые активы меняют наш образ жизни. Этим фактом обусловлена важность криптографии и кибербезопасности в современном мире. Изменения в обработке и хранении данных потребовали соответствующей эволюции криптографических алгоритмов, которая позволила продвинуться в вечной борьбе с информационным пиратством.

Начав с основ симметричных и асимметричных алгоритмов, я расскажу о современных техниках аутентификации, передачи и поиска зашифрованных данных, позволяющих укрыть их от шпионов и хакеров. Вы познакомитесь с алгоритмами доказательства с нулевым разглашением, эллиптическими кривыми, гомоморфным поиском и квантовой криптографией.

Возможности квантовой криптографии продолжают расширяться и меняют представление о параметрах, которые ранее считались безопасными. Инновации в квантовой криптографии находятся на переднем плане криптографической мысли, поэтому во второе издание я постарался включить начальный курс квантовой криптографии и введение в квантовый поиск на примере алгоритма Гровера.

Я рассматриваю эту книгу в качестве инструмента для студентов и специалистов, которые хотят сосредоточиться на следующем поколении криптографических алгоритмов. Я хочу помочь вам быть в курсе современных разработок в области криптографии. Тем не менее сначала мы сфокусируемся на математической логике этих алгоритмов, чтобы вы понимали основы. Надеюсь, что по мере чтения вы откроете для себя те области практической реализации, которые будут наиболее вам интересны.

Для кого эта книга

Эта книга предназначена для студентов, ИТ-специалистов, энтузиастов кибербезопасности и всех тех, кто хочет развить свои навыки в сфере современной криптографии и построить успешную карьеру в области кибербезопасности.

Структура издания

Часть I. Краткая история и основы криптографии

В главе 1 «Погружение в мир криптографии» рассказывается о криптографии и о том, для чего она нужна и почему важна в ИТ. В этой главе также дается общий обзор основных алгоритмов в истории криптографии.

Часть II. Классическая криптография

В главе 2 «Алгоритмы симметричного шифрования» анализируется симметричное шифрование. Мы сосредоточимся на таких алгоритмах, как DES, AES, а также изучим булеву алгебру, которая широко применяется для реализации киберсистем. В конце продемонстрированы атаки на представленные алгоритмы.

В главе 3 «Алгоритмы асимметричного шифрования» анализируются классические алгоритмы асимметричного шифрования, такие как RSA и алгоритм Диффи — Хеллмана, а также основные алгоритмы шифрования с закрытым и открытым ключом.

Глава 4 «Хеш-функции и цифровые подписи» посвящена хеш-функции SHA-1 и цифровым подписям — одному из столпов современной криптографии. Мы рассмотрим наиболее важные и известные подписи, в частности анонимные и слепые.

Часть III. Новые криптографические алгоритмы и протоколы

В главе 5 «Доказательство с нулевым разглашением» рассматриваются протоколы доказательства с нулевым разглашением — один из новых фундаментальных протоколов шифрования для блокчейна. Они очень полезны для аутентификации людей и машин без раскрытия конфиденциальных данных в небезопасном канале связи. На таких алгоритмах основаны новые протоколы, используемые в блокчейне, в частности zk-SNARK. Глава завершится представлением изобретенного мной нового доказательства с нулевым разглашением Z/K13.

В главе 6 «Новые изобретения в криптографии и логические атаки» описаны три изобретенных мной алгоритма. MB09 основан на последней теореме Ферма. MB11 может стать альтернативой RSA. Также представлены цифровые подписи,

связанные с этими алгоритмами. Кроме того, мы рассмотрим MBXX — новый протокол, который может быть использован для реализации консенсуса.

В главе 7 «Эллиптические кривые» рассматривается новый рубеж децентрализованных финансов — эллиптические кривые. Для реализации передачи цифровой валюты в систему «Биткойн» Сатоши Накамото использовал особую эллиптическую кривую под названием SECP256K1. Мы посмотрим, как она работает и каковы основные характеристики этого очень надежного способа шифрования.

В главе 8 «Гомоморфное шифрование и система защищенного поиска» рассматривается система криптовалютного поиска, которая является примером использования гомоморфного шифрования. Гомоморфное шифрование — это система поиска запроса в зашифрованном тексте. Вы увидите, как она реализована, узнаете историю ее разработки и возможные способы применения этого прорывного механизма для обеспечения безопасности и конфиденциальности данных.

Часть IV. Квантовая криптография

В главе 9 «Основы квантовой криптографии» рассказывается, что с появлением квантовых вычислений большинство рассмотренных нами алгоритмов окажутся под серьезной угрозой атак методом грубой силы. Возможным решением является квантовая криптография — один из самых захватывающих и фантастических видов шифрования, изобретенных человеческим разумом. Мы находимся в самом начале пути развития квантовой криптографии, но в скором времени она получит широкое распространение.

Глава 10 «Алгоритмы квантового поиска и квантовые вычисления» представит алгоритм Гровера в качестве примера квантового поиска и атаки на алгоритмы классического симметричного шифрования. Изучив логику этого алгоритма, вы увидите, как некоторые элементы квантовых вычислений применяются в криптографии, в частности, для решения задач, связанных со случайным поиском и атаками грубой силы.

Как получить максимальную пользу от книги

В издании систематически разбирается математика, стоящая за алгоритмами. Однако для полного понимания материала необходимо знание математики университетского уровня, включая алгебру, модульную арифметику и теорию конечных полей. Полезными также будут знания об эллиптических кривых и квантовых вычислениях, особенно в области таблиц и построения кривых.

Мы также предоставляем PDF-файл с цветными версиями изображений и схем из оригинальной книги. Вы можете скачать его здесь: https://packt.link/gbp/9781835080030.

Условные обозначения

В книге используются следующие условные обозначения.

Курсив

Курсивом выделены новые термины и важные понятия.

Шрифт без засечек

Используется для обозначения URL.

Так оформляется примечание.

Так оформляется совет или предложение.

Над книгой работали

Об авторе

Массимо Бертаччини — доктор наук, генеральный директор и соучредитель компании Cryptolab Inc. Будучи исследователем, получил несколько патентов в области криптографии, квантовой криптографии и искусственного интеллекта. Начав свою карьеру с преподавания математики и статистики, Массимо основал Cryptolab Inc. — стартап в области криптографических решений обеспечения кибербезопасности. Вместе с командой инженеров разработал и внедрил первую в мире поисковую систему, работающую с зашифрованными данными.

Массимо стал лауреатом нескольких международных премий, включая Silicon Valley Inventors, получил знак качества Европейского союза и награду Security Solutions Provider of the Year (США, 2023 год). Сейчас он преподает криптографию в рамках курса по кибербезопасности и активно публикует статьи по криптографии и блокчейну.

Первое издание этой книги на протяжении 40 недель оставалось на десятом месте среди бестселлеров в своей категории на Amazon и было признано BookAuthority лучшей книгой 2023 года в области гомоморфного и квантового шифрования.

Особую благодарность я выражаю Марко Массари и Даниэле Сартини, а также от всего сердца благодарю Леона Ксу и других рецензентов.

Кроме того, я хотел бы поблагодарить Пола Хегера, Маттео Сарика, Давиде Кармечи, Майкола Ломбарди, Джека Волосевича, Пьерджорджио Монтанари, Гауденцио Гаравини, Ицзин Гу и Сару Раухвергер за их поддержку на протяжении моей профессиональной карьеры.

Наконец, я хотел бы поблагодарить всех сотрудников Packt за помощь в написании этой книги.

О научных редакторах

Дэвид Тиллеманс более 20 лет работает в области обеспечения безопасности. В течение 10 лет занимался криптографией и смарт-картами в ходе разработки средств инфраструктуры открытых ключей (public key infrastructure, PKI). Позже он перешел с позиции инженера по криптографии на должность инженера по безопасности приложений, в качестве которого внедрил процесс безопасной разработки, сменив специализацию с криптографа на этичного хакера и консультанта по процессам безопасной разработки.

Полученный опыт позволил Дэвиду стать специалистом в области, сочетающей криптографию, этический хакинг и практику безопасной разработки. Сегодня он является независимым консультантом в финансовом и правительственном секторах, где выступает в качестве архитектора ИОК, разработчика безопасных приложений и консультанта по обеспечению безопасности бизнеса.

Доктор Пол Дюплис — исследователь в области обеспечения безопасности и руководитель исследовательской программы «Безопасность, конфиденциальность и защита» в подразделении корпоративных исследований компании Robert Bosch GmbH — крупнейшего в мире поставщика автомобилей и производителя промышленных, бытовых и потребительских товаров. Пол занимается прикладными исследованиями в различных областях информационной безопасности с 2007 года. Его текущие исследовательские интересы включают автоматизацию безопасности, безопасность программного обеспечения, сетевую безопасность, обнаружение вторжений и ловушек для хакеров[1], применение ИИ для обеспечения безопасности и безопасность самого ИИ, проектирование и технологии сохранения конфиденциальности. Пол получил степень доктора наук по информатике в Тюбингенском университете.

Леон Сюй — инженер-программист в TVU Networks — ведущей компании в области инновационных технологий передачи медиаданных. В рамках своей должности он занимается оптимизацией алгоритмов сетевой передачи данных и разработкой приложений для обнаружения и распознавания объектов в реальном времени.

Леон получил степень магистра прикладной физики в Стэнфордском университете. Являясь страстным энтузиастом квантовых вычислений, он участвует в таких мероприятиях, как IBM Quantum Challenge и QHack.

Для меня большая честь быть рецензентом этой книги и привнести свой технический опыт в успех этого издания.

---

[1] Такие ловушки еще называются ханипотами (от англ. honeypot). Это специально созданная система или сервис, которые имитируют уязвимую цель, чтобы привлечь злоумышленников. — *Здесь и далее примеч. науч. ред.*

О бета-читателях

Далее перечислены те, кто принял участие в нашей программе бета-тестирования книги. Мы хотели бы поблагодарить за ценную помощь в рецензировании:

- Томаса Морриса — AgileDinosaur;
- Анджали Латхию — Archangel;
- Арьяна Пандея — oumuamua.

От издательства

Мы выражаем огромную благодарность клубу рецензентов ИТ-литературы ReadIT Club за помощь в работе над русскоязычным изданием книги и их вклад в повышение качества переводной литературы.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@sprintbook.kz (издательство SprintBook, компьютерная редакция).

Мы будем рады узнать ваше мнение!

О научном редакторе русскоязычного издания

Татьяна Квист — специалист по информационной безопасности и разработчик с опытом работы в академической и прикладной среде. Автор научных публикаций по криптографическому алгоритму RSA, микросервисной архитектуре и схеме разделения секрета Шамира, а также соавтор зарегистрированного программного обеспечения, созданного для обучения методам защиты данных.

С 2020 года Татьяна организует и проводит образовательные мероприятия в области информационной безопасности, в том числе готовит лекции, разрабатывает учебные материалы и создает практические задания для CTF-соревнований (жанр соревнований по ИБ). С 2022 года читает открытые лекции в Петрозаводском государственном университете, выступает на конференциях и помогает студентам осваивать криптографию. В своей практике Татьяна любит приводить аналогии для сложных тем, чтобы изучение было проще и интереснее, поэтому иногда в примечаниях вы можете увидеть сравнение с бытовыми историями.

Часть I

Краткая история и основы криптографии

В первой, вводной, главе вы познакомитесь с базовыми определениями и понятиями, а также историей криптографии и ее алгоритмами.

1 Погружение в мир криптографии

Добро пожаловать в мир криптографии. Данная книга раскроет секреты этой потрясающей науки, которые могут стать полезными как для вашей карьеры, так и для общего развития. Вы познакомитесь с ключевыми алгоритмами, которыми славится криптография, а также откроете для себя некоторые новые алгоритмы, которые я разработал и применил на практике. Я надеюсь, что чтение не только доставит вам удовольствие, но и поможет в достижении ваших академических и профессиональных целей.

В главе 1 вы узнаете, что такое криптография, для чего она нужна и почему имеет такое большое значение в ИТ. Здесь будет представлен краткий обзор основных алгоритмов, разработанных за все время существования криптографии: от шифра Цезаря до шифра Вернама и других менее известных алгоритмов, например криптограммы Бейла. Далее будут подробно описаны алгоритм Ривеста, Шамира и Адлемана (RSA), алгоритм Диффи — Хеллмана, расширенный стандарт шифрования (AES), доказательство с нулевым разглашением, эллиптические кривые, гомоморфное шифрование, квантовая криптография и другие известные алгоритмы.

Эта глава позволит вам понять суть криптографии и основы обеспечения безопасности.

Введение в криптографию

Одним из наиболее важных аспектов изучения криптографии является понимание определений и обозначений. Мне никогда не нравились определения и обозначения, прежде всего потому, что я один использую обозначения, которые сам и ввел. Однако я осознаю, что это очень важное условие для того, чтобы понимать друг друга, особенно когда мы говорим о математике. Поэтому сейчас я представлю основные сведения и понятия, связанные с криптографией.

Начнем с определения алгоритма.

В математике и информатике *алгоритмом* называется конечная последовательность точно заданных компьютеризируемых инструкций.

Здесь возникает важный вопрос: а что такое шифр?

Шифр — это любая система, позволяющая преобразовать открытый текст (сообщение) в недоступный для понимания (шифротекст или криптограмму) (рис. 1.1).

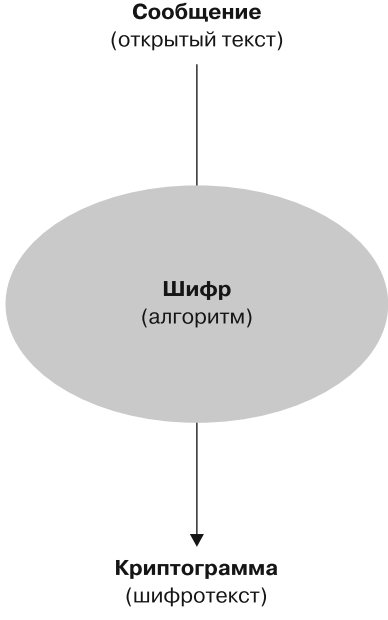


Рис. 1.1. Процесс шифрования

Для того чтобы шифр работал, нам нужно определить две операции: шифрование и расшифрование. Проще говоря, необходимо обеспечить секретность и безопасность сообщения в течение некоторого времени.

Определим M как совокупность всех сообщений (messages) и C как совокупность всех криптограмм (cryptograms).

Шифрование (encryption) — это операция, которая преобразует общее сообщение m в криптограмму с помощью функции E:

$$m \to f(E) \to c.$$

Расшифрование (decryption), или расшифровка[1], — это операция, которая возвращает криптограмме *c* ее первоначальный вид открытого текста *m* с помощью функции *D*:

$$c \rightarrow f(D) \rightarrow m.$$

Математически расшифровку можно выразить так:

$$D(E(m)) = m.$$

Следовательно, *E* и *D* являются обратными друг другу функциями, при этом функция *E* должна быть инъективной. Под *инъективностью* подразумевается то, что разные значения *M* должны соответствовать разным значениям *C*.

Обратите внимание: неважно, использую я прописные или строчные буквы, например (*M*) или (*m*), так как это не имеет никакого значения. Сейчас я добавил круглые скобки без какого-то скрытого смысла, однако дальше секретные параметры функции будут обозначаться квадратными скобками, чтобы можно было отличать их от известных параметров. Таким образом, секретное сообщение *M* будет обозначено [*M*], как и любой другой секретный параметр. В данном случае наша цель заключается в том, чтобы просто показать, как работают алгоритмы, а их реализацию оставим инженерам.

Шифрование и расшифрование имеют еще один важный, можно сказать, ключевой компонент. Для того чтобы зашифровать или расшифровать сообщение, необходим ключ. В криптографии ключом называется параметр, который определяет функциональный выход криптографического алгоритма или шифра. Без него алгоритм бесполезен.

Мы определяем *K* как набор всех ключей (key), используемых для шифрования и расшифрования сообщений *M*, а *k* — как единственный ключ шифрования или расшифрования, который называется также сеансовым ключом. Эти два способа определения ключа (набор ключей — *K*, один ключ — *k*) будут использоваться всегда, независимо от типа ключа — закрытый или открытый.

Теперь, когда мы разобрались с основными криптографическими обозначениями, пришло время объяснить разницу между закрытыми и открытыми ключами.

- В криптографии *закрытый*, или *секретный*, *ключ*, обозначаемый [*K*] или [*k*], — это параметр шифрования или расшифрования, известный только одной или нескольким сторонам, участвующим в обмене секретными сообщениями.
- В криптографии *открытый ключ*, или (*K*), — это ключ шифрования, известный всем, кто хочет отправить секретное сообщение или аутентифицировать пользователя.

---

[1] В русском языке понятия расшифровки и дешифровки отличаются. Расшифрование — это восстановление открытого текста из шифротекста при наличии ключа. Дешифрование — это попытка восстановить текст без ключа, то есть взлом шифра.

В чем же основная разница между закрытыми и открытыми ключами?

Закрытый ключ используется как для шифрования, так и для расшифровки сообщения, а открытый — только для шифрования сообщения и проверки личности (цифровых подписей) людей и компьютеров. Это существенная и очень важная деталь, поскольку она определяет разницу между симметричным и асимметричным шифрованием.

Дам общее определение этих двух методов шифрования.

- В *симметричном шифровании* используется только один общий ключ как для шифрования, так и для расшифровки сообщения.
- В *асимметричном шифровании* для генерирования открытого ключа (для шифрования сообщения) задействуются несколько параметров, в то время как для расшифровки сообщения требуется только один закрытый ключ.

Как вы увидите далее, в симметричном шифровании один и тот же закрытый ключ применяется для шифрования и расшифрования сообщения, а в асимметричном шифровании закрытый ключ используется только для расшифрования. Открытые ключи применяются только в асимметричном шифровании для шифрования сообщения и обработки цифровой подписи. Функции этих двух типов ключей будут раскрыты позже, а пока необходимо запомнить, что закрытый ключ используется как в симметричном, так и в асимметричном шифровании, а открытый ключ — только в асимметричном. В мои планы не входит обсуждение академических определений и обозначений, поэтому, пожалуйста, постарайтесь просто понять, какова область применения каждого элемента и как они используются.

Одна из основных проблем в криптографии заключается в передаче ключа или обмене ключами. В свое время она вызвала бурную полемику в сообществе математиков и криптографов, поскольку было очень сложно определить, как передать ключ без физического контакта.

Например, если Алиса и Боб хотели обменяться ключом (до появления асимметричного шифрования), единственным надежным способом сделать это была физическая встреча. С массовым внедрением телекоммуникационных систем и Интернета это условие создало множество проблем. Первая проблема состояла в том, что связь через Сеть осуществлялась посредством обмена данными по небезопасным каналам. Как можно легко понять, если Алиса общается с Бобом по небезопасному публичному каналу связи, существует большая вероятность компрометации[1] закрытого ключа, что крайне опасно для безопасности и конфиденциальности коммуникаций.

---

[1] Компрометация — это ситуация, при которой конфиденциальная информация, ключи, пароли или системы становятся доступными посторонним лицам или теряют свою безопасность.

По этой причине возникает вопрос: *если мы используем симметричный шифр для защиты секретной информации, как обеспечить безопасность при обмене секретным ключом?*

Простой ответ таков: мы должны обеспечить *безопасность канала* связи для обмена ключом.

Кто-то может спросить: *а как обеспечить безопасность канала?*

Ответ, а точнее, несколько ответов вы найдете далее в этой книге. Даже в сложных военных ситуациях, например во время холодной войны, в легендарной красной линии между лидерами США и СССР использовались симметричные ключи связи. В наши дни принято применять асимметричное шифрование для обмена ключом. Затем, в следующем сеансе связи, этот ключ объединяется с симметричным шифрованием для шифрования передаваемых сообщений.

По многим причинам асимметричное шифрование является хорошим способом обмена ключами и подходит для аутентификации и цифровых подписей. С вычислительной точки зрения симметричное шифрование лучше, поскольку оно работает с ключами небольшой длины, что позволяет экономить производительные ресурсы и время. Так, алгоритмы симметричного шифрования эффективно обеспечивают безопасность, применяя ключи длиной всего 256–512 бит. Сравните это с более чем 4000 бит асимметричного шифрования RSA, например[1]. Подробнее о том, почему и как это возможно, я расскажу во время анализа алгоритмов асимметричного и симметричного шифрования.

В этой книге я проанализирую множество криптографических техник, однако, по большому счету, все алгоритмы можно разделить на два больших семейства: симметричное и асимметричное шифрование.

Для полного понимания криптографии будут полезны еще несколько определений.

- *Открытый текст.* В криптографии это незашифрованный текст или все, что можно представить в открытом доступе. Например, *(встретимся завтра в 10 утра)* — это открытый текст.
- *Шифротекст.* В криптографии это результат шифрования текста. Например, *«встретимся завтра в 10 утра»* может быть представлено в виде шифротекста *[x549559*ehebibcm3494]*.
- Как уже говорилось, для обозначения открытого текста и шифротекста я использую *разные виды скобок.* В частности, круглые скобки (...) обозначают

---

[1] В асимметричном шифровании можно использовать ключи длиной 1024 и менее бит. Но это уже небезопасно, есть риск дешифровки. В 2025 году рекомендуется использовать минимум 2048-битные ключи. В качестве примера можно привести рекомендации NIST (США): https://doi.org/10.6028/NIST.SP.800-57pt1r5.

открытый текст, а квадратные скобки [...] — шифротекст. Таким образом, сообщение, о котором говорилось ранее, — *[x549559*ehebibcm3494]* — считается секретным.

Двоичные числа, код ASCII и условные обозначения

Когда мы работаем с данными на компьютерах, мы обычно используем их в виде строк, состоящих из 0 и 1 (бит). Таким образом, числа из обычной системы счисления с основанием 10 можно перевести в биты (система счисления с основанием 2). Посмотрим, как работает механизм такого преобразования. Например, число 123 можно записать с основанием 10 как[1]:

$$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0.$$

Аналогичным образом можем перевести число с основанием 10 в число с основанием 2. Возьмем в качестве примера число 29 (рис. 1.2)[2].

Перевод числа 29 в число с основанием 2

Шаг	Операция	Результат	Остаток	Перевод (основание 2)
Шаг 1	29 / 2	14	1	**(11101)2**
Шаг 2	14 / 2	7	0	
Шаг 3	7 / 2	3	1	
Шаг 4	3 / 2	1	1	
Шаг 5	1 / 2	0	1	

Рис. 1.2. Перевод числа 29 в число с основанием 2 (биты)

[1] Такой формат записи числа называется позиционным разложением, или разложением числа по основанию системы счисления.

[2] При переводе числа в другую систему счисления его последовательно делят на основание новой системы до тех пор, пока частное не станет нулем. Запись числа в новой системе формируется из остатков, выписанных в обратном порядке.

Деление с остатком — очень популярная операция в криптографии, потому что *модульная арифметика* основана на концепции остатков. Мы углубимся в эту тему в следующем разделе, когда будем говорить о простых числах и модульной арифметике.

Чтобы компьютеры могли кодировать буквы, Американская ассоциация стандартов в 1960 году изобрела код ASCII, переводящий символы в двоичную систему.

На сайте ASCII имеется следующее определение:

> *«ASCII расшифровывается как American Standard Code for Information Interchange (Американская стандартная кодировка для обмена информацией). Это семибитный символьный код, в котором каждый бит представляет собой уникальный символ».*

На рис. 1.3 приведен пример таблицы кодов ASCII с первыми десятью знаками и управляющими символами[1].

DEC	OCT	HEX	BIN	Символ	HTML-код	Описание
0	000	00	00000000	NUL	&#000;	Нулевой байт
1	001	01	00000001	SOH	&#001;	Начало заголовка
2	002	02	00000010	SIX	&#002;	Начало текста
3	003	03	00000011	ETX	&#003;	Конец текста
4	004	04	00000100	EOT	&#004;	Конец передачи
5	005	05	00000101	ENQ	&#005;	«Прошу подтверждения!»
6	006	06	00000110	ACK	&#006;	«Подтверждаю!»
7	007	07	00000111	BEL	&#007;	Звуковой сигнал — звонок
8	010	08	00001000	BS	&#008;	Возврат на один символ (BACKSPACE)
9	011	09	00001001	HT		Табуляция
10	012	0A	00001010	LF
Перевод строки

Рис. 1.3. Первые десять знаков и символов, выраженных в коде ASCII

В главе 4 вы познакомитесь с шестнадцатеричной (hex) и восьмеричной (octal) системами счисления, а пока ограничимся двоичной системой.

Обратите внимание на то, что в своих реализациях, выполненных с помощью исследовательского программного обеспечения *Wolfram Mathematica*, я буду часто использовать символ 88 в качестве X для обозначения номера шифруемого

[1] Управляющие символы — это специальные символы в кодировках, которые не отображаются как обычный знак, а используются для управления устройствами ввода-вывода или форматированием данных.

сообщения. Как можно увидеть в следующем примере, в коде ASCII число 88 соответствует символу *X*:

88 130 58 01011000 *X* X прописная *X*.

Последняя теорема Ферма, простые числа и модульная арифметика

Говоря о криптографии, мы всегда должны держать в голове, что эта дисциплина, по сути, связана с математикой и логикой[1]. Прежде чем приступить к объяснению *последней теоремы Ферма*, я бы хотел ввести еще несколько основных обозначений, которые будут использоваться в книге, что поможет избежать путаницы и лучше понимать текст. Важно знать, что символы =, ≡ (эквивалент) и := (его можно найти в системе Mathematica для операции =) нужны только для того, чтобы сказать, что два элемента соответствуют друг другу в равной степени вне зависимости от того, в каком контакте они представлены: в конечном поле (не волнуйтесь, я объясню этот термин), информатике или обычной алгебре. Математики могут прийти в ужас от такого определения, но я надеюсь, что вы будете искать суть, а не единообразие.

Другой символ, ≈ (приблизительно), может использоваться для обозначения подобных приближенных элементов.

Вы также будете встречать символ ^ (возведение в степень) в качестве классического способа выражения степени: $a\hat{}x$ обозначает «a в степени x», или a^x.

Символ ≠, как вы должны помнить из курса средней школы, означает «не равно», что в модульной арифметике представлено в виде ≢, то есть «не эквивалентно».

Тем не менее вы всегда получите объяснение уравнений. Даже если вы не очень хорошо ладите с математическими и логическими обозначениями, то можете положиться на описания уравнения. В любом случае я буду объяснять каждое новое определение или символ.

Как вы, скорее всего, знаете, простое число — это целое число, которое может делиться только на себя и на 1, например 2, 3, 5, 7... 23... 67... *p*.

Простые числа являются краеугольным камнем математики, поскольку все остальные числа происходят от них. Числа, которые имеют делители, отличные от 1 и самого себя, называются составными. Это, например, 4, 6, 8, 9 и т. д. Вы увидите, что использование в криптографии составного числа вместо простого может стать причиной серьезного снижения безопасности или возникновения атаки (см. главу 5).

---

[1] Под логикой здесь и далее в книге подразумевается математическая логика. Это раздел математики, который формально описывает правильные рассуждения и правила вывода новых утверждений из уже известных.

Теперь посмотрим, что такое последняя теорема Ферма, где она применяется и чем полезна для нас.

Последняя, или великая, теорема Ферма — одна из лучших и самых красивых теорем классической математики, строго связанная с простыми числами, которая имеет основополагающее значение для криптографии. Согласно Википедии,

> *«в теории чисел последняя теорема Ферма (иногда называемая догадкой Ферма, особенно в старых текстах) утверждает, что для любого натурального числа $n > 2$ уравнение $a^n + b^n = c^n$ не имеет решений в целых ненулевых числах a, b, c. С древности было известно, что случаи $n = 1$ и $n = 2$ имеют бесконечно много решений».*

Другими словами, для любой степени $n \geqslant 3$ не существует целого числа a, b или c, которое удовлетворяет уравнению[1]:

$$a^n + b^n = c^n.$$

Почему эта теорема так важна для нас? С каждым новым изученным алгоритмом вы будете все лучше понимать ее значение. По сути, последняя теорема Ферма имеет непосредственное отношение к простым числам. Учитывая свойства простых чисел, продемонстрировать последнюю теорему Ферма можно с помощью уравнения:

$$a^p + b^p \neq c^p,$$

где p — любое простое число больше 2.

Сам Ферма в одной из своих работ отметил, что у него было прекрасное подтверждение теоремы, *которое было слишком обширным, чтобы поместиться на полях его заметок*, однако оно так и не было найдено.

Доказательство Уайлса заняло более 200 страниц. В последней версии оно было сокращено до 130 страниц и оказалось чрезвычайно сложным для понимания. Основано доказательство на эллиптических кривых, которые характеризуются тем, что имеют особую модулярную форму. Уайлс пришел к этому выводу спустя семь лет работы и в 1994 году представил свое доказательство на одном из математических конгрессов. Вы частично познакомитесь с логикой, использованной в доказательстве Уайлса, в главе 7. В данный момент мы просто примем тот факт, что Уайлс подтвердил последнюю теорему Ферма с помощью гипотезы Таниямы — Шимуры, утверждающей, что *эллиптические кривые над полем рациональных чисел связаны с модулярными формами*. Опять же не волнуйтесь, если эта гипотеза покажется вам слишком сложной: по мере нашего продвижения вперед она начнет обретать смысл.

---

[1] При $n = 2$ уравнение принимает вид теоремы Пифагора и имеет бесконечно много целочисленных решений (например, $3^2 + 4^2 = 5^2$). Ключевое отличие последней теоремы Ферма состоит в том, что она касается степеней строго больше 2.

Мы тщательно проанализируем последнюю теорему Ферма в главе 6, где я представлю основанный на ней алгоритм MB09, а также другие инновационные алгоритмы открытых и закрытых ключей. Кроме того, в главе 7 мы разберем криптографическое применение эллиптических кривых.

Ферма, как и многие другие математики, был одержим простыми числами. Он изучал простые числа и их свойства всю жизнь. Он пытался найти общую формулу для обозначения всех простых чисел во Вселенной, но, к сожалению, как и многие другие математики, сумел создать формулу только для некоторых из них. Далее приводится формула, по которой находят простые числа Ферма, где n — некоторое целое положительное число:

$$p = 2^{2^n} + 1.$$

Если вместо n подставить целые числа, то можно получить простые числа:

- $n = 1, p = 5$;
- $n = 2, p = 17$;
- $n = 3, p = 65$ (не простое число);
- $n = 4, p = 257$.

Возможно, наиболее известной, но очень похожей на формулу Ферма является формула поиска простых чисел Мерсенна, где n — некоторое целое положительное число:

$$p = 2^n - 1.$$

В результате получается:

- $n = 1, p = 1$;
- $n = 2, p = 3$;
- $n = 3, p = 7$;
- $n = 4, p = 15$ (не простое число);
- $n = 5, p = 31$.

Несмотря на бесчисленные попытки найти формулу, в которой были бы представлены все простые числа, до сих пор никому не удалось сделать это.

Great Internet Mersenne Prime Search (GIMPS) — это исследовательский проект, цель которого состоит в нахождении новых и самых больших простых чисел с помощью формулы Мерсенна.

На сайте GIMPS можно узнать следующее.

- Все экспоненты ниже 53 423 543 были проверены и подтверждены.
- Все экспоненты ниже 92 111 363 были проверены хотя бы раз.
- Найдено 51-е простое число Мерсенна!

- Сообщение от 21 декабря 2018 года: «В рамках проекта Great Internet Mersenne Prime Search (GIMPS) обнаружено самое большое из известных простых чисел — $2^{82\,589\,933} - 1$, состоящее из 24 862 048 цифр. Компьютер, добровольно предоставленный Патриком Ларошем из Окалы, штат Флорида, сделал эту находку 7 декабря 2018 года. Новое простое число, также известное как M82589933, вычисляется путем умножения 82 589 933 двоек и вычитания единицы из полученного произведения. Оно более чем на полтора миллиона цифр больше предыдущего рекордного простого числа».

Кроме того, GIMPS — это, вероятно, первый децентрализованный пример того, как можно разделить мощности процессора и компьютера для достижения общей цели. Почему же поиск больших простых чисел вызывает у людей такой интерес?

На этот вопрос есть как минимум три ответа: страсть к чистому исследованию, денежное вознаграждение, причитающееся тем, кто находит бóльшие простые числа, и, наконец, потому, что простые числа так же важны для криптографии, как кислород для человека. Именно по этой причине за открытие больших простых чисел полагается вознаграждение.

Вскоре вы поймете, что большинство алгоритмов нового поколения работают с простыми числами. Но как определить, является ли число простым?[1]

В математике существует значительная разница в вычислительной сложности операций умножения и деления. Деление требует гораздо больше вычислительных ресурсов, чем умножение. Другими словами, довольно просто возвести $2x$ (где x — огромное число) в степень и в то же время крайне сложно найти делители этого числа.

Из-за этого математики, такие как Ферма, пытались найти алгоритмы, которые облегчили бы эти вычисления.

В области простых чисел Ферма вывел еще одну очень интересную теорему, известную как *малая теорема Ферма*. Прежде чем мы перейдем к ней, необходимо прояснить, что такое модульная арифметика и как с ее помощью выполнять вычисления.

Самый простой способ понять суть модульной арифметики — вспомнить часы. Когда мы говорим: «Эй, мы можем встретиться в час», мы имеем в виду, что 1 — это первый час после 12, когда часы заканчивают круговой оборот.

Таким образом, можно сказать, что мы бессознательно вычисляем по модулю 12, что записывается как mod 12, где отсчет целых чисел *циклически повторяется*

---

[1] Если вам нужно проверить, является ли число простым, можете воспользоваться базой известных простых чисел на сайте factordb.com. Обозначения можно посмотреть здесь: factordb.com/status.html.

после достижения определенного значения (в данном случае 12), называемого модулем.

Технически результат вычисления с модулем состоит из остатка от деления числа на модуль.

Так, в нашем примере с часами:

$$13 \equiv 1 \pmod{12}.$$

Это означает, что 13 *конгруэнтно* с 1 по модулю 12. Здесь *конгруэнтность* означает равенство. Другими словами, мы можем сказать, что остаток от деления 13 / 12 равен 1 (рис. 1.4).

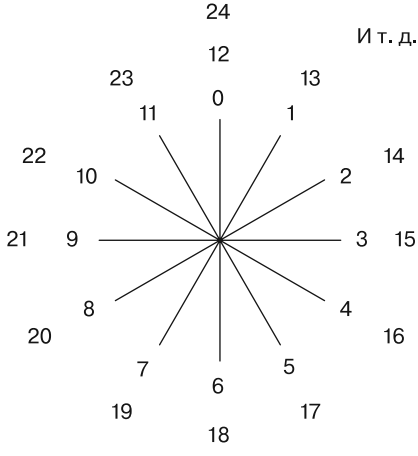


Рис. 1.4. Пример модульной арифметики с часами

Малая теорема Ферма гласит:

> *«Если p — простое число, то любое целое число a, возведенное в степень простого числа p, будет результатом уравнения $a^p \equiv a \pmod{p}$».*

Например, если $a = 2$ и $p = 3$, то $2^3 = 2 \pmod 3$. Другими словами, мы определяем остаток от деления 8 / 3 = 2 с остатком 2.

Малая теорема Ферма лежит в основе теста простоты Ферма и является одним из фундаментальных разделов элементарной теории чисел.

Эта теорема утверждает, что число *p*, *вероятно, является простым* в следующем случае: $a^p \equiv a \pmod{p}$.

Теперь, когда вы освежили свои знания о переводе числа в биты, увидели, как выглядит код ASCII, и изучили основные понятия математики и логики, можем начать путешествие в криптографию.

Краткая история и общий обзор криптографических алгоритмов

Никто, наверное, не знает, какой была первая криптограмма. Криптография используется уже очень давно — около 4000 лет, и за это время она успела сменить множество обликов. Сначала это был своего рода скрытый язык, затем ее основой стала перестановка букв механическим способом, и, наконец, для решения сложных задач стали использовать математику и логику. Что ждет нас в будущем? Вероятно, будут изобретены новые методы сокрытия наших секретов: например, уже ведутся эксперименты с квантовой криптографией. На протяжении всей книги я буду рассказывать о новых алгоритмах и методах шифрования, но позвольте мне сперва показать вам несколько интересных шифров, относящихся к классическому периоду. Несмотря на всю вычислительную мощь, которой мы сейчас обладаем, некоторые из этих алгоритмов до сих пор не взломаны.

Розеттский камень

Одним из первых выдающихся примеров криптографии была иероглифика. Криптография означает «скрытые слова» и происходит от соединения двух греческих слов: κρυπτός («крипто») и γράφω («графо»). Среди множества определений этого слова мы находим следующее: *преобразование обычного текста в неразборчивый и наоборот*. Мы смогли включить иероглифы в это определение, когда узнали, как *преобразовать* их скрытый смысл в понятный текст. Это произошло только тогда, когда был найден Розеттский камень.

Как вы наверняка помните из начальной школы, надписи на Розеттском камне были сделаны на трех языках: древнеегипетском (с помощью иероглифов), демотическом и древнегреческом (рис. 1.5).

Розеттский камень удалось расшифровать только благодаря древнегреческому языку, который был хорошо известен на тот момент.

Иероглифы были формой общения между жителями Древнего Египта и некоторых стран, расположенных поблизости. В начале 1822 года Жан-Франсуа Шампольон смог расшифровать написанное на Розеттском камне. Однако ученый-энциклопедист Томас Юнг признан египтологами первым, кто опубликовал частично правильный перевод текста с Розеттского камня. Мы еще не раз встретимся с Томасом Юнгом в главе 9. Он был первым, кто открыл двойственность свойств фотонов, которые могли вести себя и как волны, и как частицы. Это открытие — важный элемент квантовой механики.

Та же проблема расшифровки неизвестного языка может возникнуть в будущем, если мы вступим в контакт с инопланетянами. Этому посвящен проект «Институт поиска внеземного разума» (SETI) (https://www.seti.org/):

> *«От микробов до инопланетного разума, Институт SETI — единственная в Америке организация, полностью посвященная поиску жизни во Вселенной».*

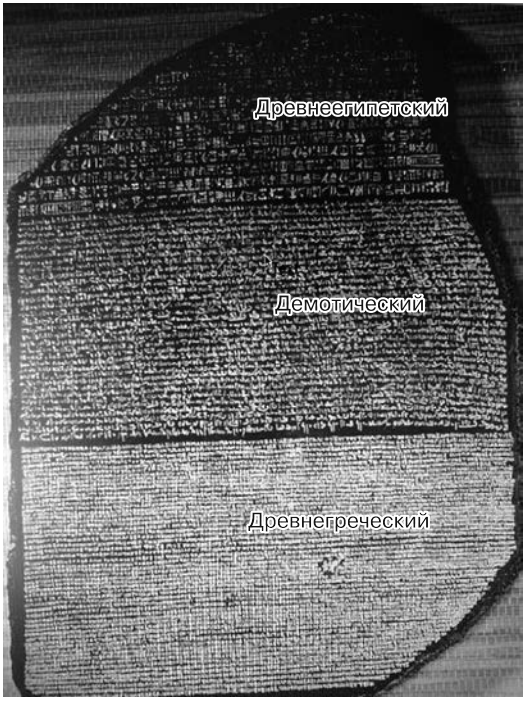


Рис. 1.5. Розеттский камень с текстом на трех языках

Возможно, если мы однажды вступим в контакт с инопланетными существами, то со временем поймем их язык. Вы можете себе представить, что иероглифы в свое время казались столь же непостижимыми, как и язык инопланетян, для тех, кто никогда не сталкивался с этой формой общения.

Шифр Цезаря

Продолжая путешествие по истории, мы увидим, что во времена Римской империи криптография применялась для передачи сообщений от генералов к командирам и солдатам. Фактически мы видим знаменитый *шифр Цезаря*. Почему этот метод шифрования так хорошо известен в истории криптографии?

Не только потому, что его использовал Юлий Цезарь — один из самых выдающихся римских государственных деятелей и военачальников, но и потому, что этот метод был, вероятно, первым, в котором применена математика.

Этот шифр широко известен как шифр сдвига. Техника сдвига очень проста: достаточно сдвинуть каждую шифруемую букву на постоянное число позиций в алфавите так, чтобы в итоге каждая буква была заменена другой. Так, например, если я решу сдвинуть все буквы вправо на три позиции, то А станет Г, Д — Ж и т. д.

В данном случае, переставляя каждую букву на три позиции, мы неявно создали секретный криптографический ключ [K = 3] (рис. 1.6).

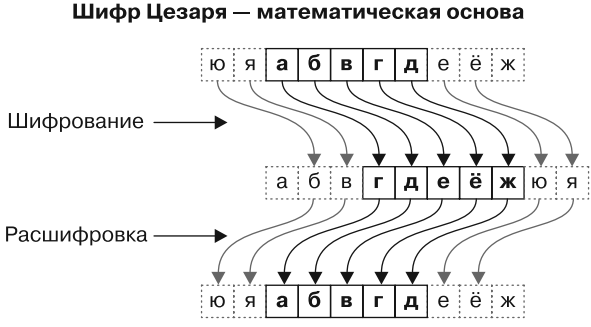


Рис. 1.6. Перестановка букв в шифре Цезаря в процессе шифрования и расшифровки

Очевидно, что перед нами метод шифрования с симметричным ключом. В этом случае алгоритм работает следующим образом:

- используется ключ (+3);
- сообщение — ПРИВЕТ;
- для шифрования каждая буква сдвигается на три позиции вправо (+3);
- для расшифровки каждая буква сдвигается на три позиции влево (–3).

На рис. 1.7 показано, как происходит процесс шифрования и расшифрования шифра Цезаря с использованием *ключа* = + 3. Как можно заметить, после шифрования слово «ПРИВЕТ» превращается в «ТУЛЕЗХ», а после расшифрования — обратно в «ПРИВЕТ»[1].

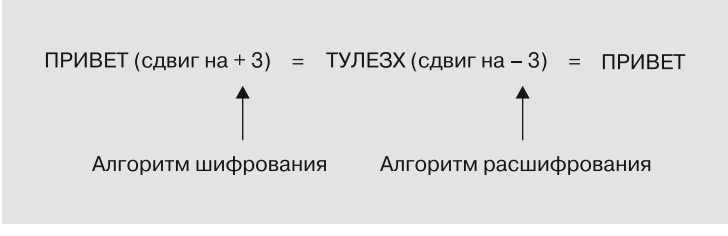


Рис. 1.7. Шифрование и расшифрование с помощью шифра Цезаря

[1] Можно встретить обозначение ROT3, это то же самое, что и ключ = +3.

Шифр Цезаря очень легко взломать с помощью обычного компьютера, если будет задан фиксированный ключ, как в нашем примере[1]. Простота схемы не является проблемой для криптографического алгоритма. А главная проблема заключается в крайней линейности лежащей в ее основе математики. С помощью метода грубой силы, то есть теста, который перебирает все комбинации для обнаружения ключа после угадывания используемого алгоритма (в данном случае шифра сдвига), можно легко взломать код. В примере нужно проверить не более 32 комбинаций: все буквы русского алфавита (33) минус одна (исходная форма открытого текста). Это ничто по сравнению с миллиардами попыток, которые должен сделать компьютер, чтобы взломать современный криптографический алгоритм.

Существует и более сложная версия этого алгоритма, значительно повышающая эффективность шифрования.

Если я буду менять ключ для каждой буквы и использовать его для замены букв и генерации шифротекста, то дело примет интересный оборот. Это будет уже новый вид шифра, называемый транспозиционным[2].

Давайте посмотрим, что произойдет, если мы зашифруем «СЕАНС» с помощью такого метода. Нам нужно сделать следующее.

1. Прописать алфавит.
2. Выбрать парольную фразу (также известную как ключевая фраза), например [ЮЛИЙЦЕЗАРЬ], в качестве второй строки и, повторяя ее несколько раз, расположить так, чтобы каждой букве алфавита соответствовал символ из ключевой фразы, как показано на рис. 1.8.
3. Для каждого символа шифруемого сообщения (в первой строке) выбрать соответствующий символ ключевой фразы (во второй строке).
4. Создать шифротекст, подбирая соответствующие символы из второй строки.

Сложновато? Не волнуйтесь, рисунок *все* прояснит.

Рассматриваемый шифротекст — это простой пример того, как происходит шифрование с помощью парольной фразы. Безусловно, существует много способов его расшифровки, но здесь мы рассматриваем только шифрование.

---

[1] В шифре Цезаря алфавит является общей информацией, известной и отправителю, и получателю. Чтобы успешно расшифровать сообщение, получатель должен использовать тот же алфавит, что и отправитель. Поэтому если злоумышленник сможет перехватить ключ, то, скорее всего, он знает и исходный алфавит.

[2] В этом случае получатель и отправитель должны иметь информацию об одних и тех же ключевых фразах и алфавитах.

Зашифруем «СЕАНС» с помощью ключевой фразы [ЮЛИЙЦЕЗАРЬЮЛИЙ...] (рис. 1.8).

[алфавит]	**А** Б В Г Д **Е** Ё Ж З И Й К Л М **Н** О П Р **С** Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я
[парольная фраза]	**Ю** Л И Й Ц **Е** З А Р Ь Ю Л И Й **Ц** Е З А **Р** Ь Ю Л И Й Ц Е З А Р Ь Ю Л И
[шифротекст]	**Ю** **Е** **Ц** **Р**

СЕАНС = **РЕЮЦР**

Рис. 1.8. Шифрование «СЕАНС» с помощью ключевой фразы атаковать сложнее

Таким образом, зашифровав открытый текст СЕАНС с помощью алфавита и ключа (а лучше парольной или ключевой фразы) ЮЛИЙЦЕЗАРЬ, повторенного без пробелов несколько раз, мы получим шифротекст РЕЮЦР.

Таким образом, С становится Р, Е остается Е, А меняется на Ю, а Н превращается в Ц.

Если в исходном шифре Цезаря для поиска ключа нужно было проверить всего 32 комбинации, то здесь ситуация иная, так как существуют (33!) возможности для обнаружения ключа, или 8 683 317 618 811 890 000 000 000 000 000 000 000 комбинаций (результат умножения $1 \cdot 2 \cdot 3 \cdot ... \cdot 33$).

Еще одним плюсом такой криптограммы является то, что ключевое слово или ключевую фразу легко запомнить, следовательно, вычислить шифротекст не составляет труда. Теперь перейдем к шифру с аналогичной техникой выполнения, который используется в коммерческих целях.

ROT13

Современный пример алгоритма, который используется в Интернете, — ROT13, где ROT — сокращение от слова rotate (в данном контексте «циклический сдвиг»)[1]. По сути, это просто вариант шифра Цезаря со сдвигом на (+13) с *применением одного алгоритма для шифрования и расшифрования*. С вычислительной точки зрения его так же легко взломать, как и шифр Цезаря, но у него есть интересный эффект: при сдвиге и влево, и вправо получается один и тот же результат.

Как и в предыдущем примере, в ROT13 необходимо выбрать буквы, соответствующие заранее выбранному ключу. Разница заключается в том, что для шифрования вместо выбора ключевой фразы мы будем использовать 13 букв

[1] Этот шифр предназначен только для английского алфавита, так как в нем всего 26 букв. Для русского алфавита, который состоит из 33 букв (или 32 без учета «ё»), можно создать аналогичную схему. Например, ROT16.

английского алфавита в качестве *генератора ключей*. При шифровании ROT13 берет только буквы, встречающиеся в английском алфавите, — цифры, символы и другие знаки остаются неизменными. Функция ROT13, по сути, шифрует открытый текст с помощью ключа, который состоит из первых 13 букв алфавита, преобразованных во вторые 13 букв, и наоборот.

Лучше понять схему шифрования поможет следующий пример (рис. 1.9).

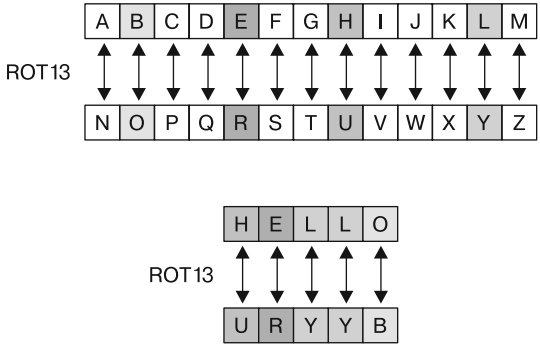


Рис. 1.9. Схема шифрования в ROT13

Здесь можно увидеть, что H становится U, E превращается в R, L — в Y (дважды), а O — в B:

HELLO = URYYB.

Ключ состоит из первых 13 букв английского алфавита до M, которая становится Z. Затем начинается новая последовательность: N становится A, O — B и т. д. до Z, которая становится M.

ROT13 использовался для того, чтобы скрыть потенциально оскорбительную шутку или ответ в новостной группе net.jokes в начале 1980-х годов.

Кроме того, хотя ROT13 не предназначен для обеспечения высокой степени секретности, он все еще используется в некоторых случаях для сокрытия адресов электронной почты от примитивных спам-ботов. ROT13 применяется также для обхода спам-фильтров, например для сокрытия содержимого писем. Задействовать последнюю функцию не рекомендуется из-за крайней уязвимости этого алгоритма.

ROT13 использовался в *Netscape Communicator* — браузере организации, выпустившей https://www.mozilla.org, — для хранения паролей электронной почты. Более того, ROT13 применяется в Windows XP для сокрытия некоторых ключей реестра, так что теперь становится понятно, как иногда даже в крупных корпорациях могут возникать проблемы с безопасностью и конфиденциальностью связи.

Криптограммы Бейла

Возвращаясь к истории криптографии, я хотел бы представить вам удивительный метод шифрования. Несмотря на огромную вычислительную мощность современных технологий, этот шифр до сих пор не раскрыт. Очень часто криптография используется для сокрытия ценной информации или захватывающих сокровищ. Именно такая загадка скрывается за криптограммами *Бейла*.

Чтобы лучше понять метод шифрования, задействованный в этих криптограммах, думаю, будет интересно узнать историю (или легенду) о Бейле и его сокровищах.

Эта история о зарытых сокровищах стоимостью более 20 млн долларов, таинственных зашифрованных документах, ковбоях Дикого Запада и владельце гостиницы, который посвятил свою жизнь расшифровке этих бумаг. Ее полная версия изложена в брошюре, опубликованной в 1885 году.

История (ее полную версию вы найдете здесь: http://www.unmuseum.org/bealepap.htm) начинается в январе 1820 года в Линчбурге, штат Вирджиния, в гостинице «Вашингтон», куда заселился человек по имени Томас Дж. Бейл. Он подружился с владельцем гостиницы Робертом Моррисом. Поскольку господин Моррис считался человеком, заслуживающим доверия, Бейл отдал ему коробку, в которой лежали три загадочных листа бумаги, испещренных числами.

В результате многолетних усилий и бесчисленных провалов удалось расшифровать только второй из трех зашифрованных листов.

Как же выглядят криптограммы Бейла?

Они представляют собой три страницы, содержащие только числа, расположенные в случайном порядке.

Первая страница выглядит следующим образом:

71, 194, 38, 1701, 89, 76, 11, 83, 1629, 48, 94, 63, 132, 16, 111, 95, 84, 341, 975, 14, 40, 64, 27, 81, 139, 213, 63, 90, 1120, 8, 15, 3, 126, 2018, 40, 74, 758, 485, 604, 230, 436, 664, 582, 150, 251, 284, 308, 231, 124, 211, 486, 225, 401, 370, 11, 101, 305, 139, 189, 17, 33, 88, 208, 193, 145, 1, 94, 73, 416, 918, 263, 28, 500, 538, 356, 117, 136, 219, 27, 176, 130, 10, 460, 25, 485, 18, 436, 65, 84, 200, 283, 118, 320, 138, 36, 416, 280, 15, 71, 224, 961, 44, 16, 401, 39, 88, 61, 304, 12, 21, 24, 283, 134, 92, 63, 246, 486, 682, 7, 219, 184, 360, 780, 18, 64, 463, 474, 131, 160, 79, 73, 440, 95, 18, 64, 581, 34, 69, 128, 367, 460, 17, 81, 12, 103, 820, 62, 116, 97, 103, 862, 70, 60, 1317, 471, 540, 208, 121, 890, 346, 36, 150, 59, 568, 614, 13, 120, 63, 219, 812, 2160, 1780, 99, 35, 18, 21, 136, 872, 15, 28, 170, 88, 4, 30, 44, 112, 18, 147, 436, 195, 320, 37, 122, 113, 6, 140, 8, 120, 305, 42, 58, 461, 44, 106, 301, 13, 408, 680, 93, 86, 116, 530, 82, 568, 9, 102, 38, 416, 89, 71, 216, 728, 965, 818, 2, 38, 121, 195, 14, 326, 148, 234, 18, 55, 131, 234, 361, 824, 5, 81, 623, 48, 961, 19, 26, 33, 10, 1101, 365, 92, 88, 181, 275, 346, 201, 206, 86,

36, 219, 324, 829, 840, 64, 326, 19, 48, 122, 85, 216, 284, 919, 861, 326, 985, 233, 64, 68, 232, 431, 960, 50, 29, 81, 216, 321, 603, 14, 612, 81, 360, 36, 51, 62, 194, 78, 60, 200, 314, 676, 112, 4, 28, 18, 61, 136, 247, 819, 921, 1060, 464, 895, 10, 6, 66, 119, 38, 41, 49, 602, 423, 962, 302, 294, 875, 78, 14, 23, 111, 109, 62, 31, 501, 823, 216, 280, 34, 24, 150, 1000, 162, 286, 19, 21, 17, 340, 19, 242, 31, 86, 234, 140, 607, 115, 33, 191, 67, 104, 86, 52, 88, 16, 80, 121, 67, 95, 122, 216, 548, 96, 11, 201, 77, 364, 218, 65, 667, 890, 236, 154, 211, 10, 98, 34, 119, 56, 216, 119, 71, 218, 1164, 1496, 1817, 51, 39, 210, 36, 3, 19, 540, 232, 22, 141, 617, 84, 290, 80, 46, 207, 411, 150, 29, 38, 46, 172, 85, 194, 39, 261, 543, 897, 624, 18, 212, 416, 127, 931, 19, 4, 63, 96, 12, 101, 418, 16, 140, 230, 460, 538, 19, 27, 88, 612, 1431, 90, 716, 275, 74, 83, 11, 426, 89, 72, 84, 1300, 1706, 814, 221, 132, 40, 102, 34, 868, 975, 1101, 84, 16, 79, 23, 16, 81, 122, 324, 403, 912, 227, 936, 447, 55, 86, 34, 43, 212, 107, 96, 314, 264, 1065, 323, 428, 601, 203, 124, 95, 216, 814, 2906, 654, 820, 2, 301, 112, 176, 213, 71, 87, 96, 202, 35, 10, 2, 41, 17, 84, 221, 736, 820, 214, 11, 60, 760.

Вторая (расшифрованная) страница:

115, 73, 24, 807, 37, 52, 49, 17, 31, 62, 647, 22, 7, 15, 140, 47, 29, 107, 79, 84, 56, 239, 10, 26, 811, 5, 196, 308, 85, 52, 160, 136, 59, 211, 36, 9, 46, 316, 554, 122, 106, 95, 53, 58, 2, 42, 7, 35, 122, 53, 31, 82, 77, 250, 196, 56, 96, 118, 71, 140, 287, 28, 353, 37, 1005, 65, 147, 807, 24, 3, 8, 12, 47, 43, 59, 807, 45, 316, 101, 41, 78, 154, 1005, 122, 138, 191, 16, 77, 49, 102, 57, 72, 34, 73, 85, 35, 371, 59, 196, 81, 92, 191, 106, 273, 60, 394, 620, 270, 220, 106, 388, 287, 63, 3, 6, 191, 122, 43, 234, 400, 106, 290, 314, 47, 48, 81, 96, 26, 115, 92, 158, 191, 110, 77, 85, 197, 46, 10, 113, 140, 353, 48, 120, 106, 2, 607, 61, 420, 811, 29, 125, 14, 20, 37, 105, 28, 248, 16, 159, 7, 35, 19, 301, 125, 110, 486, 287, 98, 117, 511, 62, 51, 220, 37, 113, 140, 807, 138, 540, 8, 44, 287, 388, 117, 18, 79, 344, 34, 20, 59, 511, 548, 107, 603, 220, 7, 66, 154, 41, 20, 50, 6, 575, 122, 154, 248, 110, 61, 52, 33, 30, 5, 38, 8, 14, 84, 57, 540, 217, 115, 71, 29, 84, 63, 43, 131, 29, 138, 47, 73, 239, 540, 52, 53, 79, 118, 51, 44, 63, 196, 12, 239, 112, 3, 49, 79, 353, 105, 56, 371, 557, 211, 505, 125, 360, 133, 143, 101, 15, 284, 540, 252, 14, 205, 140, 344, 26, 811, 138, 115, 48, 73, 34, 205, 316, 607, 63, 220, 7, 52, 150, 44, 52, 16, 40, 37, 158, 807, 37, 121, 12, 95, 10, 15, 35, 12, 131, 62, 115, 102, 807, 49, 53, 135, 138, 30, 31, 62, 67, 41, 85, 63, 10, 106, 807, 138, 8, 113, 20, 32, 33, 37, 353, 287, 140, 47, 85, 50, 37, 49, 47, 64, 6, 7, 71, 33, 4, 43, 47, 63, 1, 27, 600, 208, 230, 15, 191, 246, 85, 94, 511, 2, 270, 20, 39, 7, 33, 44, 22, 40, 7, 10, 3, 811, 106, 44, 486, 230, 353, 211, 200, 31, 10, 38, 140, 297, 61, 603, 320, 302, 666, 287, 2, 44, 33, 32, 511, 548, 10, 6, 250, 557, 246, 53, 37, 52, 83, 47, 320, 38, 33, 807, 7, 44, 30, 31, 250, 10, 15, 35, 106, 160, 113, 31, 102, 406, 230, 540, 320, 29, 66, 33, 101, 807, 138, 301, 316, 353, 320, 220, 37, 52, 28, 540, 320, 33, 8, 48, 107, 50, 811, 7, 2, 113, 73, 16, 125, 11, 110, 67, 102, 807, 33, 59, 81, 158, 38, 43, 581, 138, 19, 85, 400, 38, 43, 77, 14, 27, 8, 47, 138, 63, 140, 44, 35, 22, 177, 106, 250, 314, 217, 2, 10, 7, 1005, 4, 20, 25, 44, 48, 7, 26, 46, 110, 230, 807, 191, 34, 112, 147, 44, 110, 121, 125, 96, 41, 51, 50, 140, 56, 47, 152, 540, 63, 807, 28, 42, 250, 138, 582, 98, 643, 32, 107, 140, 112, 26, 85, 138, 540, 53, 20, 125, 371, 38, 36, 10, 52, 118, 136, 102, 420, 150, 112, 71, 14, 20, 7, 24, 18, 12, 807, 37, 67, 110, 62, 33, 21, 95, 220,

511, 102, 811, 30, 83, 84, 305, 620, 15, 2, 10, 8, 220, 106, 353, 105, 106, 60, 275, 72, 8, 50, 205, 185, 112, 125, 540, 65, 106, 807, 138, 96, 110, 16, 73, 33, 807, 150, 409, 400, 50, 154, 285, 96, 106, 316, 270, 205, 101, 811, 400, 8, 44, 37, 52, 40, 241, 34, 205, 38, 16, 46, 47, 85, 24, 44, 15, 64, 73, 138, 807, 85, 78, 110, 33, 420, 505, 53, 37, 38, 22, 31, 10, 110, 106, 101, 140, 15, 38, 3, 5, 44, 7, 98, 287, 135, 150, 96, 33, 84, 125, 807, 191, 96, 511, 118, 40, 370, 643, 466, 106, 41, 107, 603, 220, 275, 30, 150, 105, 49, 53, 287, 250, 208, 134, 7, 53, 12, 47, 85, 63, 138, 110, 21, 112, 140, 485, 486, 505, 14, 73, 84, 575, 1005, 150, 200, 16, 42, 5, 4, 25, 42, 8, 16, 811, 125, 160, 32, 205, 603, 807, 81, 96, 405, 41, 600, 136, 14, 20, 28, 26, 353, 302, 246, 8, 131, 160, 140, 84, 440, 42, 16, 811, 40, 67, 101, 102, 194, 138, 205, 51, 63, 241, 540, 122, 8, 10, 63, 140, 47, 48, 140, 288.

Третья страница:

317, 8, 92, 73, 112, 89, 67, 318, 28, 96,107, 41, 631, 78, 146, 397, 118, 98, 114, 246, 348, 116, 74, 88, 12, 65, 32, 14, 81, 19, 76, 121, 216, 85, 33, 66, 15, 108, 68, 77, 43, 24, 122, 96, 117, 36, 211, 301, 15, 44, 11, 46, 89, 18, 136, 68, 317, 28, 90, 82, 304, 71, 43, 221, 198, 176, 310, 319, 81, 99, 264, 380, 56, 37, 319, 2, 44, 53, 28, 44, 75, 98, 102, 37, 85, 107, 117, 64, 88, 136, 48, 151, 99, 175, 89, 315, 326, 78, 96, 214, 218, 311, 43, 89, 51, 90, 75, 128, 96, 33, 28, 103, 84, 65, 26, 41, 246, 84, 270, 98, 116, 32, 59, 74, 66, 69, 240, 15, 8, 121, 20, 77, 89, 31, 11, 106, 81, 191, 224, 328, 18, 75, 52, 82, 117, 201, 39, 23, 217, 27, 21, 84, 35, 54, 109, 128, 49, 77, 88, 1, 81, 217, 64, 55, 83, 116, 251, 269, 311, 96, 54, 32, 120, 18, 132, 102, 219, 211, 84, 150, 219, 275, 312, 64, 10, 106, 87, 75, 47, 21, 29, 37, 81, 44, 18, 126, 115, 132, 160, 181, 203, 76, 81, 299, 314, 337, 351, 96, 11, 28, 97, 318, 238, 106, 24, 93, 3, 19, 17, 26, 60, 73, 88, 14, 126, 138, 234, 286, 297, 321, 365, 264, 19, 22, 84, 56, 107, 98, 123, 111, 214, 136, 7, 33, 45, 40, 13, 28, 46, 42, 107, 196, 227, 344, 198, 203, 247, 116, 19, 8, 212, 230, 31, 6, 328, 65, 48, 52, 59, 41, 122, 33, 117, 11, 18, 25, 71, 36, 45, 83, 76, 89, 92, 31, 65, 70, 83, 96, 27, 33, 44, 50, 61, 24, 112, 136, 149, 176, 180, 194, 143, 171, 205, 296, 87, 12, 44, 51, 89, 98, 34, 41, 208, 173, 66, 9, 35, 16, 95, 8, 113, 175, 90, 56, 203, 19, 177, 183, 206, 157, 200, 218, 260, 291, 305, 618, 951, 320, 18, 124, 78, 65, 19, 32, 124, 48, 53, 57, 84, 96, 207, 244, 66, 82, 119, 71, 11, 86, 77, 213, 54, 82, 316, 245, 303, 86, 97, 106, 212, 18, 37, 15, 81, 89, 16, 7, 81, 39, 96, 14, 43, 216, 118, 29, 55, 109, 136, 172, 213, 64, 8, 227, 304, 611, 221, 364, 819, 375, 128, 296, 1, 18, 53, 76, 10, 15, 23, 19, 71, 84, 120, 134, 66, 73, 89, 96, 230, 48, 77, 26, 101, 127, 936, 218, 439, 178, 171, 61, 226, 313, 215, 102, 18, 167, 262, 114, 218, 66, 59, 48, 27, 19, 13, 82, 48, 162, 119, 34, 127, 139, 34, 128, 129, 74, 63, 120, 11, 54, 61, 73, 92, 180, 66, 75, 101, 124, 265, 89, 96, 126, 274, 896, 917, 434, 461, 235, 890, 312, 413, 328, 381, 96, 105, 217, 66, 118, 22, 77, 64, 42, 12, 7, 55, 24, 83, 67, 97, 109, 121, 135, 181, 203, 219, 228, 256, 21, 34, 77, 319, 374, 382, 675, 684, 717, 864, 203, 4, 18, 92, 16, 63, 82, 22, 46, 55, 69, 74, 112, 134, 186, 175, 119, 213, 416, 312, 343, 264, 119, 186, 218, 343, 417, 845, 951, 124, 209, 49, 617, 856, 924, 936, 72, 19, 28, 11, 35, 42, 40, 66, 85, 94, 112, 65, 82, 115, 119, 236, 244, 186, 172, 112, 85, 6, 56, 38, 44, 85, 72, 32, 47, 63, 96, 124, 217, 314, 319, 221, 644, 817, 821, 934, 922, 416, 975, 10, 22, 18, 46, 137, 181, 101, 39, 86, 103, 116, 138, 164, 212, 218, 296, 815, 380, 412, 460, 495, 675, 820, 952.

Вторая криптограмма была успешно расшифрована примерно в 1885 году. Здесь я расскажу об основных соображениях, связанных с этим видом шифра.

Поскольку количество чисел в шифре значительно превышает количество букв в алфавите, это не подстановочный и не транспозиционный шифр. Таким образом, можно предположить, что каждое число представляет собой букву из слова из внешнего текста. Шифр, работающий по такому принципу, называется *книжным*. В нем в качестве ключа может быть использована книга или любой другой текст. Эффективным ключом здесь является метод получения букв из текста.

Понимание этой системы позволило расшифровать вторую криптограмму с помощью Декларации независимости Соединенных Штатов. Присвоив каждому слову ссылочного текста (Декларации независимости США) номер и взяв первую букву каждого слова, выделенного в ключе (в списке номеров, в данном случае относящемся ко второму шифру), удалось вывести открытый текст. Исключительная изобретательность этого шифра заключается в том, что текст-ключ (Декларация независимости США) находился в открытом доступе, но в то же время о нем знали только те, кому предназначалось сообщение. Лишь тот, кто владеет ключом (списком чисел) и текстом-ключом, может легко расшифровать такое сообщение.

Рассмотрим процесс расшифровки второй криптограммы.

1. Каждому слову текста, от первого до последнего, присваивается номер.
2. Числа из криптограммы замещаются первыми буквами соответствующих им слов.
3. Получается открытый текст.

Далее представлена первая часть Декларации независимости США (до 115-го слова), где каждому слову присвоен соответствующий номер:

when(1) *in*(2) *the*(3) *course*(4) *of*(5) *human*(6) *events*(7) *it*(8) *becomes*(9) *necessary*(10) *for*(11) *one*(12) *people*(13) *to*(14) *dissolve*(15) *the*(16) *political*(17) *bands*(18) *which*(19) *have*(20) *connected*(21) *them*(22) *with*(23) *another*(24) *and*(25) *to*(26) *assume*(27) *among*(28) *the*(29) *powers*(30) *of*(31) *the*(32) *earth*(33) *the*(34) *separate*(35) *and*(36) *equal*(37) *station*(38) *to*(39) *which*(40) *the*(41) *laws*(42) *of*(43) *nature*(44) *and*(45) *of*(46) *nature's*(47) *god*(48) *entitle*(49) *them*(50) *a*(51) *decent*(52) *respect*(53) *to*(54) *the*(55) *opinions*(56) *of*(57) *mankind*(58) *requires*(59) *that*(60) *they*(61) *should*(62) *declare*(63) *the*(64) *causes*(65) *which*(66) *impel*(67) *them*(68) *to*(69) *the*(70) *separation*(71) *we*(72) *hold*(73) *these*(74) *truths*(75) *to*(76) *be*(77) *self*(78) *evident*(79) *that*(80) *all*(81) *men*(82) *are*(83) *created*(84) *equal*(85) *that*(86) *they*(87) *are*(88) *endowed*(89) *by*(90) *their*(91) *creator*(92) *with*(93) *certain*(94) *unalienable*(95) *rights*(96) *that*(97) *among*(98) *these*(99) *are*(100) *life*(101) *liberty*(102) *and*(103) *the*(104) *pursuit*(105) *of*(106) *happiness*(107) *that*(108) *to*(109) *secure*(110) *these*(111) *rights*(112) *governments*(113) *are*(114) *instituted*(115)...

Следующие числа — это первые строки второй криптограммы. Как можно увидеть, выделенные слова (вместе с их порядковыми номерами) соответствуют числам из шифротекста:

115, 73, 24, 807, 37, 52, 49, 17, 31, 62, 647, 22, 7, 15, 140, 47, 29, 107, 79, 84, 56, 239, 10, 26, 811, 5, 196, 308, 85, 52, 160, 136, 59, 211, 36, 9, 46, 316, 554, 122, 106, 95, 53, 58, 2, 42, 7, 35...

Далее представлен результат расшифрования, выполненного с использованием шифра в сочетании с текстом-ключом (Декларация независимости США), то есть открытый текст. Решение этой загадки заключалось в выборе первой буквы каждого слова открытого текста, например:

- 115 = *instituted* = *I*;
- 73 = *hold* = *h*;
- 24 = *another* = *a*;
- 807 *(отсутствует в списке выше)* = *v*;
- 37 = *equal* = *e*;
- 52 = *decent* = *d*;
- 49 = *entitle* = *e*.

Здесь я привел не всю Декларацию независимости Соединенных Штатов, а ограничился первыми 115 словами. Однако, если хотите, можете зайти на сайт https://www.nsa.gov/Helpful-Links/NSA-FOIA/Declassification-Transparency-Initiatives/Historical-Releases/Beale-Papers/ и попробовать восстановить весь открытый текст.

Вот (с некоторыми пропущенными буквами) реконструкция первого предложения:

> *I have deposited in the county of Bedford...*[1]

Если мы продолжим и сравним числа с соответствующими номерами начальных букв слов Декларации независимости США, то расшифровка будет выглядеть следующим образом:

> *I have deposited in the county of Bedford, about four miles from Buford's, in an excavation or vault, six feet below the surface of the ground, the following articles, belonging jointly to the parties whose names are given in number «3», herewith:*
>
> *The first deposit consisted of one thousand and fourteen pounds of gold, and three thousand eight hundred and twelve pounds of silver, deposited November, 1819. The second was made December, 1821, and consisted of nineteen hundred and*

[1] «В округе Бедфорд я спрятал...»

seven pounds of gold, and twelve hundred and eighty-eight pounds of silver; also jewels, obtained in St. Louis in exchange for silver to save transportation, and valued at $13,000.

The above is securely packed in iron pots, with iron covers. The vault is roughly lined with stone, and the vessels rest on solid stone, and are covered with others. Paper number «1» describes the exact locality of the vault so that no difficulty will be had in finding it[1].

Многие криптографы и криптологи безуспешно пытались расшифровать первую и третью криптограммы Бейла. Некоторые, например кладоискатель Мел Фишер, обнаруживший под водой ценности на сотни миллионов долларов, отправился в Бедфорд, чтобы прочесать местность в поисках сокровищ, но потерпел неудачу.

Возможно, история Бейла — всего лишь легенда. Может, это и правда, но никто никогда не узнает место, где скрыты сокровища, поскольку никто не расшифрует первый шифр. Или же сокровища никогда не будут найдены, потому что кто-то уже нашел их.

В любом случае что действительно интересно в этой истории, так это реализация такого сильного шифра без помощи каких-либо компьютеров или электронных машин. Он был создан с помощью простых мозгов, ручки и листа бумаги.

Парадоксально, но количество попыток, необходимых для взлома шифра, возрастает от 1 до бесконечности, если предположить, что злоумышленник работает методом грубой силы и исследует все тексты, написанные в мире на данный момент. Помимо прочего, что, если текст-ключ не находится в открытом доступе, а написан самим передатчиком и хранится в секрете? В этом случае, если у криптолога нет ключа (а значит, нет текста-ключа), вероятность того, что он расшифрует шифр, равна нулю.

---

[1] «В округе Бедфорд, в четырех милях от Бафорда, в некой заброшенной выработке или тайнике на глубине шести футов я спрятал следующие ценности, принадлежащие исключительно людям, чьи имена приводятся в бумаге, помеченной номером 3.

Первоначальный вклад составили 1014 фунтов золота и 3812 фунтов серебра, доставленные туда в ноябре 1819 года. Второй вклад, сделанный в декабре 1821 года, состоял из 1907 фунтов золота и 1288 фунтов серебра, а также из драгоценных камней, вырученных в Сент-Луисе в обмен на серебро, для облегчения процесса доставки, чья общая стоимость составляла 13 тыс. долларов.

Все вышеперечисленное надежно скрыто в железных горшках, закрытых железными крышками. Местонахождение тайника отмечают несколько выложенных вокруг него камней, сосуды покоятся на каменном основании и засыпаны сверху также камнями. Бумага под номером 1 описывает точное местонахождение тайника, так что найти его не составит труда».

Шифры Бейла интересны еще и тем, что в будущем этот вид алгоритма может найти новые варианты применения в современной криптографии. Некоторые из этих вариантов могут быть связаны с методами исследования зашифрованных данных в облачных вычислениях.

Шифр Вернама

Шифр Вернама обладает наивысшей степенью защиты, поскольку теоретически обеспечивает полную безопасность. Он использует действительно случайный ключ той же длины, что и открытый текст, поэтому его называют *идеальным*. Его идеальность обусловлена энтропией и случайностью, которые следуют из принципа информационной энтропии Шеннона, определяющего равную вероятность каждого бита шифротекста. Мы вернемся к этому алгоритму в главе 9, где поговорим о квантовом распределении ключей и связанном с ним методе шифрования открытого текста после определения квантового ключа. Еще одной интересной реализацией является проект Hyper Crypto Satellite, который использует шифр Вернама для шифрования открытого текста с помощью случайного ключа, передаваемого по спутниковой сети и выраженного в виде бесконечной строки битов.

Давайте в общих чертах рассмотрим этот алгоритм.

Важным условием алгоритма является использование ключа только один раз за сеанс. Еще одно требование — ключ должен быть той же длины, что и сообщение. Эти особенности делают алгоритм неуязвимым для атак на шифротекст, и даже в том маловероятном случае, что ключ будет украден, он будет заменен при следующей передаче. Ключ той же длины, что и сообщение, позволяет избежать проблемы коротких сообщений, как мы увидим позже. Наконец, ключ должен быть абсолютно случайным.

Метод шифрования очень прост: складывая каждый бит ключа с соответствующим битом сообщения по модулю 2, мы получаем шифротекст. Этот метод под названием *«исключающее ИЛИ»* (exclusive OR, XOR) появится в книге еще не раз, особенно когда речь пойдет о симметричном шифровании в главе 2. Сейчас же следует помнить только то, что ключ должен быть той же длины, что и сообщение.

Численный пример алгоритма выглядит следующим образом:

- 00101001 (открытый текст);
- 10101100 (ключ) — побитовое сложение (mod 2)[1];
- 10000101 (шифротекст).

---

[1] В приведенном примере: третий бит открытого текста 1 и ключа 1 дают в сумме 2. Но сумма должна быть по модулю 2, следовательно, на третьем бите шифротекста мы получаем 0, так как 2 = 0 (mod 2).

Шифрование Вернама состоит из четырех этапов.

1. Преобразование открытого текста в строку битов с помощью кода ASCII.
2. Генерация случайного ключа той же длины, что и открытый текст.
3. Шифрование сообщения путем побитового сложения открытого текста с ключом по модулю 2 (XOR) и получение шифротекста.
4. Расшифровка сообщения выполнением обратной операции сложения шифротекста с ключом и восстановление открытого текста.

Для наглядного примера с числами и буквами снова возьмем слово «ПРИВЕТ». Предположим, что каждой букве соответствует число, начиная с 0 = А, 1 = Б, 2 = В, 3 = Г, 4 = Д и так далее до 32 = Я.

Случайный ключ — [ГЁЖБВА].

Шифрование будет представлять собой перестановку (рис. 1.10).

Открытый текст		П	Р	И	В	Е	Т
		16	17	9	2	5	19
Ключ	=	Г	Ё	Ж	Б	В	А
	+	3	6	7	1	2	0
	=	19	23	14	3	7	19
Шифротекст		Т	Ц	Н	Г	Ж	Т

Рис. 1.10. Схема шифрования в алгоритме Вернама

Таким образом, после перестановки букв результатом шифрования [ПРИВЕТ] будет (ТЦНГЖТ).

Вы можете самостоятельно расшифровать шифротекст (ТЦНГЖТ) с помощью обратного алгоритма Вернама: применив $f[-K]$ к шифротексту, получите открытый текст [ПРИВЕТ].

Одна из атак, которой подвергаются многие алгоритмы, известна *как атака только на шифротекст*. Ее проведение считается успешным, если злоумышленник может вычислить открытый текст или, что еще хуже, ключ с помощью шифротекста или его фрагментов. Наиболее распространенными методами являются частотный анализ и анализ трафика. С их помощью я собираюсь рассмотреть использование букв или групп букв в шифротексте. Например, буква «о» — одна из самых часто употребляемых в русском языке, поэтому мы можем спланировать атаку, основываясь на частоте ее использования. Другими словами, если мы проанализируем трафик зашифрованных данных и обнаружим частое повторение какого-то символа в шифротексте, то его можно будет обозначить буквой «о».

Шифр Вернама неуязвим для атак только на шифротекст. Даже если вскроется часть ключа, то расшифровать можно будет только соответствующую этим

битам часть шифротекста. Остальная его часть будет сложна для расшифровки при условии достаточной длины. Однако для получения абсолютной неуязвимости условия реализации этого алгоритма очень жесткие. Во-первых, генерация ключа должна быть абсолютно случайной. Во-вторых, ключ и сообщение должны быть одинаковой длины. В-третьих, всегда существует проблема передачи ключа.

Последняя проблема затрагивает все симметричные алгоритмы и, по сути, является тем, что подтолкнуло криптографов к изобретению асимметричного шифрования для обмена ключами между Алисой и Бобом (об этом поговорим в дальнейшем).

Вторая проблема связана с длиной ключа: если сообщение слишком короткое — например, слово «два», обозначающее время начала военной атаки, — противник может положиться на свой здравый смысл или удачу. Наличие случайного ключа для короткого сообщения не имеет значения, поскольку сообщение может быть расшифровано интуитивно, если злоумышленник знает тему передаваемых сообщений. В то же время длинное сообщение вынуждает использовать очень длинный ключ, который очень дорого генерировать и передавать. Кроме того, поскольку при каждой новой передаче ключ необходимо менять, стоимость реализации шифра в коммерческих целях оказывается очень высокой.

Именно поэтому такие *одноразовые строки* применялись в военных целях во время Второй мировой войны и после нее. Как я уже говорил, этот легендарный алгоритм использовался в красной линии между Вашингтоном и Москвой для шифрования сообщений между лидерами США и СССР во время холодной войны.

Далее в книге мы проанализируем реализацию алгоритма Вернама. Генерация и передача случайного ключа могут вызвать сложности, даже если безопасность метода очень высока. В последней главе книги я покажу новый метод передачи и реализации ключей посредством шифра Вернама в сочетании с другими алгоритмами и методами. Эта новая система *одноразовых шифроблокнотов* (one-time pad, OTP) под названием Hyper Crypto Satellite может использоваться как для аутентификации, так и для шифрования сообщений.

Я также расскажу о возможных уязвимостях системы и о том, как сгенерировать случайный ключ[1]. Этот метод был кандидатом на шифр Вернама на Международной конференции по космосу, однако я решил не представлять его публике в тот момент.

---

[1] В большинстве случаев генераторы случайных чисел псевдослучайны. Проблема истинного источника энтропии — одна из самых сложных и интересных тем в криптографии и цифровой среде в целом. Некоторые компании даже используют лавалампы для более естественной генерации случайных данных.

Обеспечение безопасности и вычислений

Все алгоритмы, которые мы рассмотрели в этой главе, симметричные. Основной нерешенной проблемой таких алгоритмов является передача ключа. Здесь, как я уже упоминал, на помощь приходит асимметричная криптография, о которой мы поговорим в главе 3. В этом разделе в общих чертах рассмотрим вычислительную проблему, связанную с безопасностью криптографических алгоритмов. В дальнейшем сосредоточимся на безопасности каждого анализируемого алгоритма.

Если использовать аллегорию, то можно сказать, что в криптографии слабое звено разрушает всю цепь. Это то же самое, что использовать очень сильный криптографический алгоритм для защиты данных, но при этом оставлять пароль на мониторе компьютера. Другими словами, криптографический алгоритм должен иметь соответствующий уровень безопасности по отношению к математическим задачам. Например, задачи факторизации и дискретного логарифма пока обладают схожими вычислительными чертами, однако если завтра одна из них будет решена, то алгоритм, основанный на обеих, станет бесполезным.

Углубимся в анализ некоторых общепризнанных в криптографии принципов. Первое утверждение гласит: *криптография должна быть с открытым исходным кодом*.

Говоря об *открытом исходном коде*, я, конечно же, имею в виду сам алгоритм, а не ключ. Другими словами, мы должны опираться на *принцип Керкхоффа*, утверждающий следующее.

> *Криптосистема должна быть безопасной, даже если о ней общеизвестно все, кроме ключа.*
>
> *Принцип Керкхоффа применим не только к кодам и шифрам, но и к системам безопасности в целом: каждый секрет создает потенциальную точку отказа. Другими словами, секретность является основной причиной хрупкости, и поэтому она может привести к катастрофическому краху системы. И наоборот, открытость обеспечивает гибкость системы.*
>
> *Брюс Шнайер*

На практике алгоритм, лежащий в основе шифровального кода, должен быть известен. Полагаться на секретность алгоритма при обмене секретными сообщениями не только бесполезно, но и опасно. Причина в том, что, по сути, если алгоритм должен использоваться открытым сообществом (как в Интернете), его невозможно сохранить в тайне.

Второе утверждение заключается в следующем: *безопасность алгоритма во многом зависит от лежащей в его основе математической задачи*.

Например, RSA, один из самых известных и наиболее широко используемых алгоритмов в истории криптографии, опирается на математическую задачу факторизации.

Если коротко, факторизация — это разложение числа на простые множители. Возьмем в качестве простого примера следующее выражение:

$$21 = 3 \cdot 7.$$

Множители для небольших целых чисел найти очень легко. Для 21 ими являются 3 и 7. Однако известно, что увеличение количества цифр экспоненциально увеличивает сложность факторизации. В главе 3 мы подробно рассмотрим RSA. Однако здесь я хотел бы объяснить, почему этот асимметричный алгоритм используется для защиты финансовых данных, секретов разведки и другой конфиденциальной информации.

Причина в том, что математическая задача, лежащая в основе RSA (факторизация), до сих пор остается трудноразрешимой для современных компьютеров. Во вводной главе я не буду углубляться в анализ этого алгоритма, а только ограничусь замечанием, что слабым местом RSA является не только задача факторизации, которая может стать точкой атаки, но и другая, не менее серьезная с вычислительной точки зрения проблема — задача *дискретного логарифма*. Позже в книге мы подробно изучим обе эти трудноразрешимые задачи. Итак, мы предполагаем (что неправильно, как и в 99 % криптографических текстов), что ядром безопасности RSA является факторизация. В главе 6 я продемонстрирую атаку на алгоритм RSA, основанную не на факторизации, по аналогии со слабым звеном цепи, о котором упоминал в начале раздела. Если что-то в алгоритме идет не так, базовая безопасность алгоритма нарушается.

Все же давайте посмотрим, что произойдет, если мы попытаемся взломать RSA методом грубой силы, полагаясь только на задачу факторизации. Чтобы вы имели представление о вычислительной мощности, необходимой для декомпозиции числа RSA из 250 цифр, следует сказать, что факторизацию большого полупростого числа[1] провести совсем не просто, если мы имеем дело с сотнями или тысячами цифр.

Для примера: RSA-250 — это 829-битное число, состоящее из 250 десятичных цифр, которое очень сложно взломать современному компьютеру.

Это целое число было разложено на множители в феврале 2020 года, и общее время вычисления составило около 2700 лет ядра Intel Xeon Gold 6130 с частотой 2,1 ГГц. Как и многие другие рекорды факторизации, этот был достигнут с использованием сетки из нескольких машин и оптимизационного алгоритма, повышающего эффективность их работы.

Третье утверждение заключается в следующем: *защита на практике всегда менее надежна, чем в теории*.

---

[1] Полупростое число — это натуральное число, которое является произведением двух простых чисел.

Например, если проанализировать шифр Вернама, становится вполне очевидно, что реализовать его на практике очень сложно. Таким образом, можно сказать, что защита Вернама неуязвима, но только с точки зрения теории, а не практики. Следствием этого предположения является следующее: реализация алгоритма подразумевает применение на практике его теоретической схемы с добавлением к ней большого уровня сложности. Таким образом, *сложность — это враг безопасности*. Чем сложнее система, тем больше в ней точек атаки.

Еще одно соображение связано с уровнем безопасности алгоритма. Мы можем лучше понять эту концепцию, рассмотрев теорию Шеннона и концепцию *совершенной секретности*. Определение совершенной секретности, данное Клодом Шенноном в 1949 году, основано на статистике и вероятностях. Согласно его теории, шифротекст обладает абсолютной секретностью, если знания злоумышленника о содержании сообщения одинаковы до и после его анализа, проводимого с помощью атак с неограниченными ресурсами. Другими словами, само сообщение не дает противнику никакой информации о своем содержании.

Чтобы лучше понять эту концепцию, ее можно рассмотреть с точки зрения различных уровней, или степеней, безопасности, каждый из которых обеспечивает защиту, но с уменьшающимся уровнем надежности. Другими словами, самый высокий уровень безопасности — самый надежный, а самый низкий — самый слабый. Между ними существует промежуточная зона с неоднозначным уровнем безопасности, который зависит от технологических и вычислительных возможностей противника.

Не столь важно, сколько существует уровней безопасности, которые могут считаться надежными, а сколько — ненадежными. По моему мнению, нам главным образом необходимо уметь оценивать, что является абсолютно безопасным, а что — безопасным только в определенный момент времени. Учитывая это, давайте посмотрим, в чем разница между совершенной секретностью и простой безопасностью.

- Криптосистема может считаться *совершенно секретной*, если удовлетворяет как минимум двум условиям:
  - она не может быть взломана, даже если противник обладает неограниченной вычислительной мощностью;
  - невозможно получить какую-либо информацию о сообщении $[m]$ и ключе $[k]$ с помощью анализа шифротекста $[c]$ (то есть система Вернама теоретически является совершенной системой секретности, но только при определенных условиях).
- Шифротекст *безопасен*, даже если теоретически противник может взломать криптосистему (то есть если у него есть квантовая вычислительная мощность и эффективный алгоритм факторизации), но ее основная математическая задача на данный момент считается крайне трудноразрешимой. Таким образом, шифры (например, RSA, Диффи — Хеллмана и Эль-Гамаля) подходят для

использования при некоторых обстоятельствах, поскольку, судя по эмпирическим (полученным на практике) данным, факторизация и дискретные логарифмы по-прежнему являются трудноразрешимыми задачами.

Понятие безопасности изменчиво и нечетко. То, что безопасно сейчас, может оказаться небезопасным завтра. Что произойдет с RSA и всей классической криптографией, если станут эффективными квантовые компьютеры или будет открыт мощный алгоритм, способный решить задачу факторизации? К этим вопросам мы вернемся в главе 9. Сейчас скажу лишь то, что разрушительная вычислительная мощь квантовых компьютеров станет угрозой для классических криптографических алгоритмов, но сейчас мы не знаем, когда именно это произойдет.

Мы увидим, что при некоторых условиях *квантовый обмен ключами* можно считать *идеальной системой секретности*. Однако она не всегда работает и поэтому пока не используется. На данный момент некоторые системы OTP можно считать высокозащищенными (возможно, они обладают полусовершенной секретностью), но все зависит от практической реализации. И наконец, не забывайте важное правило: слабое звено в цепи разрушает все.

Итак, в заключение можно отметить следующее.

- Криптография должна быть с открытым исходным кодом (алгоритмы должны быть известны), за исключением ключа.
- Безопасность алгоритма во многом зависит от математической задачи, лежащей в его основе.
- Сложность — враг безопасности.
- Безопасность является изменчивой концепцией: идеальная безопасность — это лишь теоретический вопрос.

Резюме

В этой главе мы рассмотрели основные определения криптографии, вы освежили свои знания о двоичной системе и коде ASCII, а также изучили простые числа, уравнения Ферма и модульную арифметику. Кроме того, мы рассмотрели классические криптографические алгоритмы: шифр Цезаря, криптограммы Бейла и шифр Вернама.

В последнем разделе мы проанализировали безопасность с философской и технической точек зрения, разделив такие криптографические понятия, как уровень безопасности и уровень сложности.

В следующей главе рассмотрим симметричное шифрование и погрузимся в такие алгоритмы, как семейства шифров *Data Encryption Standard (DES)* и *AES*, а также затронем некоторые вопросы, упомянутые в этой главе.

Часть II

Классическая криптография

В этой части мы подробно разберем классическую криптографию: симметричное и асимметричное шифрование, хеш-функции и цифровые подписи. Вы познакомитесь с самыми известными алгоритмами, используемыми в кибербезопасности и информационных технологиях.

2 Алгоритмы симметричного шифрования

Настало время представить основные алгоритмы симметричного шифрования и их логико-математические принципы.

В главе 1 вы познакомились с некоторыми симметричными криптосистемами, такими как ROT13 и шифр Вернама. Прежде чем перейти к описанию современных симметричных алгоритмов, необходимо рассмотреть построение классических блочных шифров.

Если вы помните, симметричное шифрование осуществляется с помощью ключа, которым обладают и отправитель, и получатель. Тем не менее остается вопрос: как реализовать симметричные алгоритмы, которые были бы одновременно надежными (в смысле безопасности) и простыми в выполнении (в вычислительном плане)? Давайте посмотрим, как можно ответить на этот вопрос, сравнив асимметричное и симметричное шифрование.

Одна из главных проблем асимметричного шифрования — непростое выполнение операций, особенно расшифровки, из-за высокой вычислительной мощности, необходимой для таких алгоритмов на рекомендуемых уровнях безопасности. Из этого следует, что асимметричное шифрование не подходит для передачи длинных сообщений, и лучшим решением является обмен ключами. Таким образом, используя симметричное шифрование и расшифрование, выполняемые с одним и тем же общим ключом, мы получаем более гладкую схему обмена зашифрованными сообщениями.

К концу главы вы поймете, как реализовывать симметричные алгоритмы, управлять ими и атаковать их.

Понятия и операции в булевой алгебре

Для того чтобы понять механизм работы симметричных алгоритмов, необходимо рассмотреть некоторые понятия булевой алгебры (логики высказываний) и ее операции над двоичной системой.

Как мы видели в главе 1, двоичная система работает с набором битов {0, 1}. Таким образом, работа с булевыми функциями означает выполнение логических вычислений над последовательностью битов для получения одного из двух ответов: ИСТИНА (TRUE) или ЛОЖЬ (FALSE).

Наиболее часто используемыми функциями являются И (конъюнкция), ИЛИ (дизъюнкция) и XOR (исключающее ИЛИ). Кроме того, есть и несколько других обозначений, о которых мы скоро расскажем.

Цель булевой алгебры — определить, удовлетворяет ли переменная *x* в сочетании с другой переменной *y* условию ИСТИНА или ЛОЖЬ. Эта задача называется *задачей выполнимости булевых формул* (boolean satisfiability problem, SAT или B-SAT) и имеет особое значение в информатике. Задача SAT была первой, для которой доказывалось свойство NP-полноты.

NP-полная задача — это классическая задача из класса NP в *теории сложности вычислений*. Если на группу вопросов можно ответить в течение разумного времени, то мы обозначаем ее *P* (polynomial time — *полиномиальное время*). Если время ответа — *NP*, то есть оно *недетерминированное полиномиальное* (nondeterministic polynomial), то мы говорим, что эта группа вопросов не поддается решению за разумное машинное время. Таким образом, эти вопросы являются NP-полными. Следовательно, в общем случае NP-полные задачи трудноразрешимы.

Но я скажу, что это трудная задача только для классического компьютера. Примером может служить *задача факторизации полупростого числа RSA*, которая может быть охарактеризована как NP-полная задача. Мы увидим, что теоретически RSA не вызывает проблем у квантового компьютера, применяющего соответствующий квантовый алгоритм с нужным количеством квабитов (см. главу 9).

Вопрос заключается в следующем: если задана некоторая функция, можно ли присвоить ее переменным такие значения ИСТИНА или ЛОЖЬ, чтобы результатом выражения была ИСТИНА?

Формула *логики высказываний является выполнимой*, если существует набор значений переменных, при котором можно установить, что высказывание ИСТИННО. Если при всех возможных значениях переменных результатом будет ЛОЖЬ, то говорят, что формула невыполнима. Это имеет большое значение в теории алгоритмов, например, при реализации поисковых систем и даже при проектировании аппаратуры или электронных схем.

Приведу пример логики высказываний.

- *Высказывание 1:* «Если небо ясное, то на улице солнечно».
- *Высказывание 2:* «На небе нет облаков».
- *Вывод:* солнечная погода — это ИСТИНА.

Как видно на рис. 2.1, начиная с входных значений и развивая логическую схему с помощью алгоритма, мы получаем заключение ИСТИНА или ЛОЖЬ.

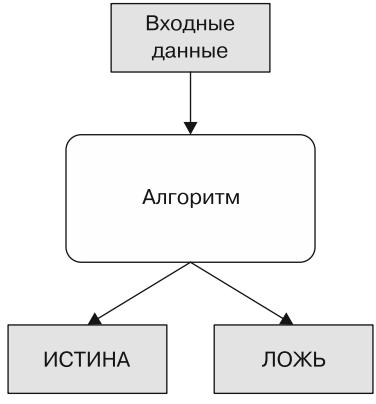


Рис. 2.1. Результатами булевой алгебры являются две противоположные переменные

Все эти понятия будут особенно полезными в последующих главах книги, особенно в главе 5, где мы поговорим о *доказательстве с нулевым разглашением*, и в главе 8, где речь пойдет о поисковой системе, работающей с зашифрованными данными.

Основные логические операции, выполняемые в булевой алгебре.

- *И (конъюнкция)*. Обозначается символом $\wedge$ ($x \wedge y$). Это условие выполняется, если x вместе с y истинны. Здесь мы имеем дело с такими высказываниями, как *«груша И яблоко»*. Если выполняем поиск, допустим, по базе данных, содержащей предложения и слова, то при установке оператора И будут выбраны все элементы, содержащие оба слова, «груша» и «яблоко», а не одно из них.

  Давайте рассмотрим, как этот оператор работает в математическом контексте. В математике оператор И представляет собой произведение $x \cdot y$. На рис. 2.2 показана *таблица значений истинности* для всех логических комбинаций двух элементов. Как видите, условие конъюнкции $x \wedge y$ считается выполненным, только когда $x \cdot y = 1$.
- *ИЛИ* (*дизъюнкция*). Обозначается символом $\vee$ ($x \vee y$). Это условие выполняется, если хотя бы один из элементов, x или y, истинен. Здесь мы имеем дело с такими высказываниями, как *«груша ИЛИ яблоко»*. В примере поиска в базе данных будут выбраны все элементы, содержащие хотя бы одно из двух слов — «груша» или «яблоко».

Таблица значений ИСТИНА для оператора И

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

Рис. 2.2. Математическая таблица для оператора И

В таблице на рис. 2.3 показано, как оператор ИЛИ превращается в математическую операцию $x + y$. Когда хотя бы одна из переменных принимает значение 1, условие дизъюнкции $x \vee y$, представленной суммой двух переменных, считается выполненным.

Таблица значений ИСТИНА для оператора ИЛИ

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

Рис. 2.3. Математическая таблица для оператора ИЛИ

Идемпотентность (от idem + potence — «тот же самый» + «сила») — это свойство некоторых операций в математике и информатике, означающее, что при повторном применении они дадут тот же результат, что и при первом. Как логическое И, так и логическое ИЛИ обладает свойством идемпотентности. Логическое И с двумя входными значениями *A* тоже будет иметь результат *A* (1 И 1 = 1, 0 И 0 = 0). Логическое ИЛИ обладает идемпотентностью, поскольку 0 ИЛИ 0 = 0 и 1 ИЛИ 1 = 1.

- *НЕ (отрицание)*. Обозначается символом $\neg x$, то есть исключающее *x*. Здесь мы имеем дело с такими высказываниями, как *«НЕ яблоко»*. Например, в базе данных мы ищем документы, не содержащие слово («яблоко»). В следующей таблице на рис. 2.4 показан оператор НЕ, обозначаемый словом отрицания (not). Он представляет собой унитарную операцию, которая возвращает значение, противоположное входному.

Таблица значений ИСТИНА для оператора НЕ

x	(НЕ) x
0	0
0	1

Рис. 2.4. Математическая таблица для оператора НЕ

Основные булевы операторы И, ИЛИ и НЕ можно представить в виде диаграммы Венна (рис. 2.5). На иллюстрации с отрицанием показан случай, когда нас интересует все, что не связано с В (то есть только то, что имеет прямое отношение к А).

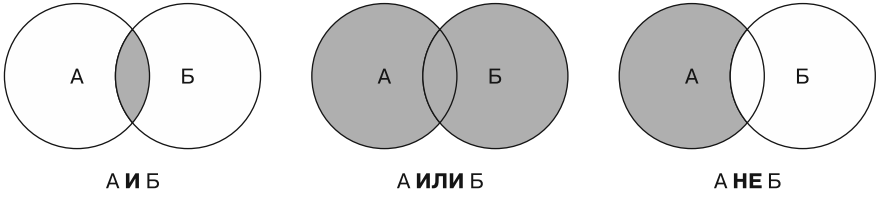


Рис. 2.5. Булевы операторы в виде диаграммы Венна

Помимо трех основных операций, которые мы только что рассмотрели, существуют и другие логические операции, в том числе *И-НЕ*, *ИЛИ-НЕ* и *исключающее ИЛИ*. Все эти операции имеют основополагающее значение в крипто-

графии. Логический оператор *И-НЕ*, например, используется в *гомоморфном шифровании*. Сейчас же нас интересует только исключающее ИЛИ.

Исключающее ИЛИ обозначается символом ⊕.

Операция $A \oplus Б$ возвращает логическое значение 1, если число переменных, принимающих значение 1, нечетно. В математическом смысле XOR — это *сложение по модулю* 2, то есть сложение комбинаций из 1 и 0 по модулю 2 (рис. 2.6). Такая операция называется исключающим ИЛИ (часто сокращенно XOR от exclusive OR).

Таблица XOR

A	⊕	B	[A (XOR) B]
0	⊕	0	= 0
0	⊕	1	= 1
1	⊕	0	= 1
1	⊕	1	= 0

Рис. 2.6. Представление операций XOR над 0 и 1

Логический оператор XOR используется не только в криптографических алгоритмах, но и как средство проверки четности. Если запустить XOR в логической схеме для проверки битов четности в слове из 8 бит, он сможет проверить, является ли общее количество единиц в слове четным или нет.

Теперь, когда мы изучили операции, лежащие в основе булевой алгебры, пришло время рассмотреть первый алгоритм симметричного семейства — DES.

Алгоритмы DES

Первый алгоритм, который мы рассмотрим в этой главе, — DES. Его история началась в 1973 году, когда *Национальному бюро стандартов США* (National Bureau of Standards, NBS), которое позже стало *Национальным институтом стандартов и технологий* (National Institute of Standards and Technology, NIST), потребовался алгоритм в качестве национального стандарта. В 1974 году IBM предложила симметричный алгоритм *Lucifer* («Люцифер»), который NIST передал в Агентство национальной безопасности (АНБ) на проверку. После анализа и некоторых модификаций этот алгоритм был переименован в DES. В 1977 году он был принят в качестве национального стандарта и стал широко

использоваться в электронной торговле (например, в финансовой сфере) для шифрования данных.

В академическом и профессиональном сообществах криптологов разгорелись нешуточные дебаты по поводу криптостойкости DES. Критика основывалась на малой длине ключа и подозрениях в том, что во время проверки АНБ могло создать в алгоритме специальную лазейку для шпионажа за зашифрованными сообщениями.

Тем не менее DES был утвержден в качестве федерального стандарта в ноябре 1976 года и опубликован 15 января 1977 года как *FIPS PUB 46*, разрешенный к использованию для всех несекретных данных. Впоследствии его повторно утверждали в качестве стандарта в 1983, 1988 (исправленная версия *FIPS-46-1*), 1993 *(FIPS-46-2)* и 1999 *(FIPS-46-3)* годах, причем последняя версия основывалась на шифре Triple DES (также известном как 3DES, о котором мы поговорим чуть позже в этой главе). Двадцать шестого мая 2002 года по результатам международного конкурса DES был окончательно вытеснен *AES*, который тоже будет рассмотрен в этой главе. DES представляет собой *блочный шифр*, принцип работы которого заключается в том, что открытый текст сначала делится на блоки по 64 бита и каждый блок шифруется отдельно. Процесс шифрования также называют *сетью Фейстеля* в честь одного из членов команды IBM, разработавшей «Люцифера», Хорста Фейстеля.

Теперь, немного узнав историю этого прародителя современных симметричных алгоритмов, мы можем углубиться в его логическую и математическую схемы.

Простой DES

Прежде чем приступить к разбору полноценного алгоритма DES, посмотрим на его упрощенную версию — простой DES.

Как и DES, этот упрощенный алгоритм является блочным шифром, то есть перед шифрованием открытый текст делится на блоки. Поскольку каждый блок шифруется отдельно, рассмотрим здесь только один из них.

Ключ [*K*] состоит из 9 бит, а сообщение [*M*] — из 12 бит.

Основной частью алгоритма, как и в DES, является *S-блок*, где S означает *подстановку* (substitution). Именно в нем кроются истинная сложность и нелинейная функция симметричных алгоритмов. В остальном это только перестановки и сдвиги битов, что обычный компьютер может сделать автоматически, так что нет причин зацикливаться на этой части алгоритма.

S-блок присутствует во всех современных алгоритмах симметричного шифрования: DES, Triple DES, Blowfish (и более современном Twofish), а также AES.

S-блок в DES представляет собой таблицу 4 × 16, состоящую из 6 бит входных данных и 4 бит выходных данных, которая вводит нелинейное отображение между входом и выходом и вызывает путаницу в данных.

Четыре строки представлены возрастающими 2 битами:

00
01
10
11

А 16 строк столбцов состоят из 4 бит в последовательности:

0000 0001 0010 ... 1111.

Ячейки таблицы состоят из случайных чисел от 0 до 15. Таким образом, они никогда не повторяются в одной и той же строке.

Чтобы вы лучше поняли принцип реализации и работы S-блока, приведу пример: 011011. Эта строка состоит из двух крайних битов, 0 и 1, и четырех битов посередине, 1101 (рис. 2.7).

Рис. 2.7. Биты строки в S-блоке

Итак, сформируем из крайних битов значение 01, а из средних битов — 1101. В двоичной системе, где используется запись вида N_2, $(01)_2$ соответствует 2-й строке таблицы, а $(1101)_2$ — 13-му столбцу. Представление таблицы S-блока в двоичной системе счисления можно увидеть на рис. 2.8.

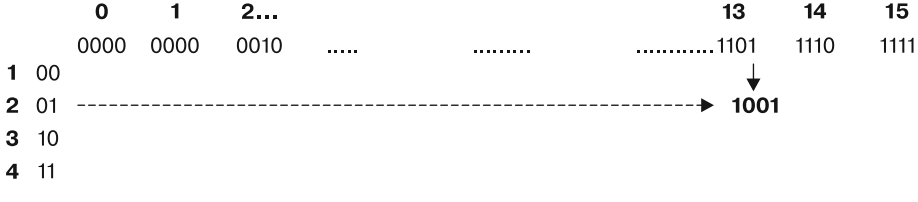


Рис. 2.8. Таблица S-блока (пересечение) 4 × 16, представленная в двоичной системе счисления

Как видите, на пересечении столбца и строки находится $(1001)_2$.

Такую же таблицу можно представить в десятичной системе счисления (рис. 2.9).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
2	14	11	2	12	4	7	13	1	5	0	15	10	3	**9**	8	6
3	3	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
4	11	8	12	7	11	14	2	3	6	15	0	9	10	4	5	3

Рис. 2.9. Таблица S-блока 4 × 16, представленная в десятичной системе счисления

Здесь на пересечении строки 2 и столбца 13 находится десятичное число 9, которому и соответствует двоичное число $(1001)_2$.

Теперь, когда мы поняли, что такое S-блок и как он устроен, можем посмотреть, как работает алгоритм.

Инициализация битов

Сообщение M содержит 12 бит и делится на две равные части, L_0 и R_0, где левая (left) половина L_0, состоит из первых 6 бит, а правая (right) половина R_0 — из последних 6 бит (рис. 2.10).

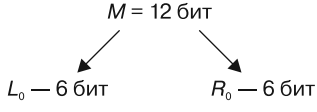


Рис. 2.10. Сообщение (M) разбито на две половины по 6 бит

Теперь, когда у вас есть четкое представление об S-блоке и инициализации битов, перейдем к таким этапам процесса, как расширение битов, генерирование ключа и шифрование битов. Как вы увидите далее, L_0 и R_0 имеют одинаковую схему расширения битов — они просто делят M.

Расширение битов

Каждый блок битов, левая и правая части, расширяется с помощью определенной функции, которая обычно обозначается f.

Алгоритм DES применяет расширение на 8 битах (1 байте), начиная с шестибитных входных данных для каждого блока открытого текста.

Кроме того, DES использует методику сегментирования битов, называемую *режимом электронной кодовой книги* (Electronic Code Book, ECB), чтобы разделить 64 бита открытого текста на восемь блоков по 8 бит, над каждым из которых будет отдельно выполнена функция шифрования (E_k).

Функция f может быть реализована по-разному. Например, вполне может быть, что первый входной бит становится первым выходным, третий бит становится четвертым и шестым и т. д. Допустим, мы хотим расширить шестибитный вход L_0 011001 с помощью функции расширения Exp (от expansion) по схеме, приведенной на рис. 2.11.

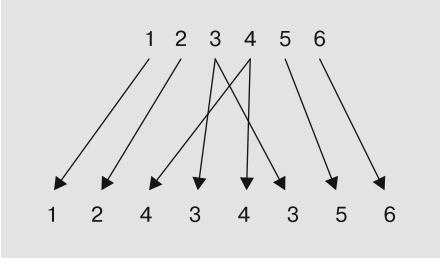


Рис. 2.11. Функция расширения битов

Как можно увидеть на предыдущем изображении, $L_0 = (011001)_2$ был расширен с помощью функции f [12434356].

Расширим L_0 011001 до $(01010101)_2$, как показано на рис. 2.12.

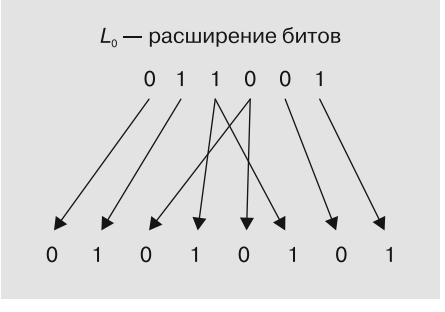


Рис. 2.12. Расширение L_0 $(011001)_2$ до 8 бит

Расширив шестибитное входное значение R_0 $(100110)_2$ до 8 бит по той же схеме f [12434356] в $R_{i-1} = (100110)_2$, мы получим следующий результат (рис. 2.13).

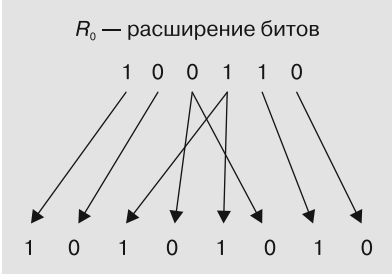


Рис. 2.13. Расширение R_0 $(100110)_2$ до 8 бит

Таким образом, после расширения R_{i-1} имеет вид $(10101010)_2$.

Генерирование ключей

Как мы уже сказали, первичный ключ $[K]$ состоит из 9 бит. Для каждого раунда у нас есть отдельный ключ шифрования $[K_i]$, сгенерированный из 8 бит первичного ключа. Отсчет битов начинается с i-го (от iteration — «циклическое повторение») раунда шифрования.

Рассмотрим пример генерации ключа K_4 (для четвертого раунда):

- K = 010011001 (девятибитный первичный ключ);
- K_4 = 01100101 (8 бит взяты из K).

Лучше понять этот процесс вам поможет рис. 2.14.

Рис. 2.14. Пример генерации ключа

Поскольку ключ $[K_4]$ создается для четвертого раунда шифрования, мы начинаем считать с четвертого бита первичного ключа $[K]$.

Шифрование битов

Для выполнения шифрования (E) мы применяем функцию XOR между расширенным $R_{i-1} = \text{Exp}(R_{i-1}) = (10101010)_2$ и $K_i = (01100101)_2$ (см. рис. 2.6).

Я называю результат этой операции $E(K_i)$:

$$\text{Exp}(R_{i-1}) \oplus K_i = E(K_i) = (11001111)_2.$$

В этот момент мы разбиваем $E(K_i)$, состоящий из 8 бит, на две части: четырехбитные левую и правую половины:

$$L(E(K_i)) = (1100)_2;\ R(E(K_i)) = (1111)_2.$$

Далее обрабатываем 4 бита слева и 4 бита справа двумя S-блоками 2×8, каждый элемент которого состоит из 3 бит. На входе у нас 4 бита: первый — строка, а последние три — двоичное число, обозначающее столбец (то же самое, что и раньше, только с меньшим количеством битов). Так, 0 означает первую строку, а 1 — вторую. Аналогично 000 означает первый столбец, 001 — второй и так далее до 111.

Назовем эти два S-блока S_1 и S_2. На рис. 2.15 представлены элементы каждого из них.

	101	010	001	110	011	100	111	000
S1	001	100	110	010	**000**	111	101	011
S2	100	000	110	101	111	001	011	010
	101	011	000	111	110	010	001	**100**

Рис. 2.15. Примеры S-блоков

$L(E(K_i)) = (1100)_2$ обрабатывается блоком S_1. Выходным параметром этой функции является число $(000)_2$, которое находится на пересечении элемента второй строки $(1)_2$ и четвертого столбца $(100)_2$.

$R(E(K_i)) = (1111)_2$ обрабатывается блоком S_2. Выходным параметром этой функции является число $(111)_2$, которое находится на пересечении элемента второй строки $(1)_2$ и четвертого столбца $(100)_2$.

Последний шаг — конкатенация двух полученных выходных значений, показанная здесь знаком $\|$, которая и станет шифротекстом:

$$S_1(L(E(K_i))) = (000)_2 \,\|\, S_2(R(E(K_i))) = (100)_2;$$

$$000 \,\|\, 100 = (000100)_2 f(R_{i-1}, K_i) = (000100)_2.$$

На рис. 2.16 показано, как математически работает шифрование первого раунда (правая часть) функции *f*.

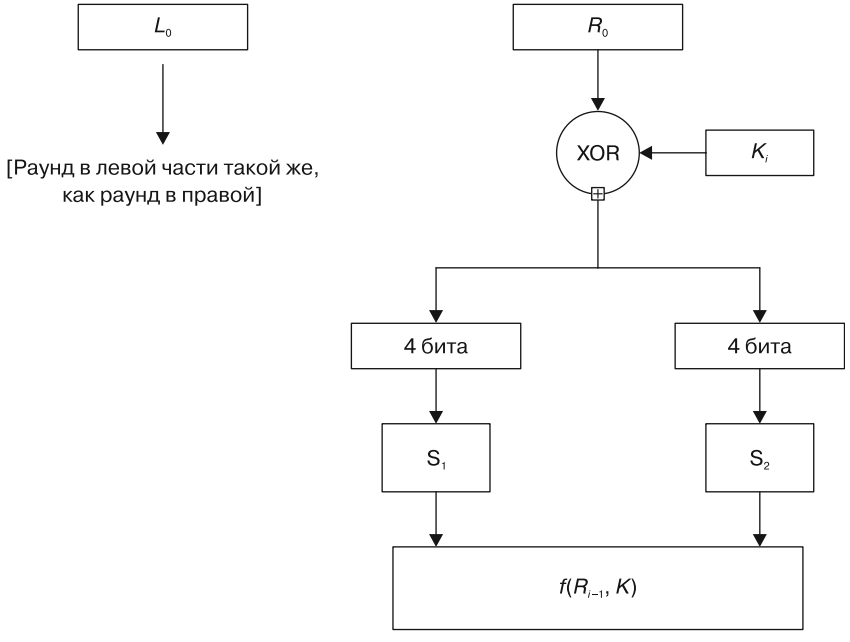


Рис. 2.16. Математическая схема простого шифрования DES в первом раунде *(правая часть)*

Теперь, когда мы рассмотрели, как работает *простой DES*, а также основы симметричного шифрования, вам будет легче разобраться в том, как работает все семейство алгоритмов DES.

Как вы уже могли понять, комбинация операций перестановок, XOR и *сдвига* является основой структуры *системы Фейстеля*.

DES

DES представляет собой 16-раундовый симметричный алгоритм шифрования и расшифрования. DES — это 64-битный шифр во всех отношениях. Операции выполняются путем разбиения сообщения [*M*] на 64-битные блоки. Ключ также имеет размер 64 бита, хотя фактически равен 56 битам (плюс 8 бит четности: 8, 16, 24-й...). Биты четности нужны для обнаружения ошибок. Размер выходных данных алгоритма (*c*) — тоже 64 бита. Чтобы полностью понять шифрование DES, еще раз посмотрите на рис. 2.16.

Генерирование ключа в DES

В 1945 году в докладе «Математическая теория криптографии» (https://www.iacr.org/museum/shannon45.html) Клод Шеннон ввел понятия *запутывания* (confusion) и *рассеивания* (diffusion). DES, как и большинство симметричных алгоритмов, использует перестановку битов, чтобы приобрести эти два свойства.

Под *запутыванием* Шеннон понимал *создание как можно более сложной и запутанной связи между шифротекстом и симметричным ключом*, а под рассеиванием — *рассредоточение статистической структуры открытого текста по всему объему шифротекста*. Эта сложность обычно реализуется с помощью четко определенной повторяющейся серии подстановок и перестановок.

Как упоминалось ранее, первичный ключ DES состоит из 64 бит. Биты ключа перечисляются с 1-го по 64-й, при этом каждый восьмой бит игнорируется. Обратите внимание на выделенный столбец в таблице на рис. 2.17.

Входной ключ DES

1	2	3	4	5	6	7	**8**
9	10	11	12	13	14	15	**16**
17	18	19	20	21	22	23	**24**
25	26	27	28	29	30	31	**32**
33	34	35	36	37	38	39	**40**
41	42	43	44	45	46	47	**48**
49	50	51	52	53	54	55	**56**
57	58	59	10	61	62	63	**64**

Рис. 2.17. Биты, игнорируемые в первичном ключе DES

Таким образом, размер нового ключа — 56 бит.

На этом этапе происходит *первая перестановка* в 56-битном ключе. Она также известна как *начальная перестановка* и является важнейшим этапом DES.

Результатом этой операции является *запутывание* позиций битов. Затем ключ делится на два подключа, C_0 и D_0, по 28 бит каждый.

После этой операции (всегда в одной строке для создания путаницы и рассеивания) выполняется циклический сдвиг (рис. 2.18).

Раунд	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Количество сдвинутых битов	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Рис. 2.18. Количество сдвигаемых в каждом раунде битов ключа в DES

Как показано в таблице, в раундах 1, 2, 9 и 16 сдвиг происходит только на 1 бит влево, в то время как в остальных раундах — на 2 бита влево.

Возьмем для примера C_0 и D_0 (левую и правую части ключа по 28 бит), записанные в двоичном виде:

$$C_0 = 1111000011001100101010101001;$$
$$D_0 = 0101010101100110011110001111.$$

Из C_0 и D_0 генерируются значения C_1 и D_1:

$$C_1 = 1110000110011001010101010011;$$
$$D_1 = 1010101011001100111100011110.$$

Если внимательно посмотреть на генерацию $C_0 \rightarrow C_1$, то легко понять, как она происходит, — это простой зацикленный сдвиг влево всех битов C_0:

$$C_0 = 1111000011001100101010101001;$$
$$C_1 = 1110000110011001010101010011.$$

Следующим этапом после циклического сдвига будет выбор 48 из 56 бит ключа, который представляет собой простую перестановку позиций: допустим, бит 14 перемещается на первую позицию, а бит 32 — на последнюю, 48-ю. Как показано в таблице (рис. 2.19), некоторые биты, такие как 18-й, исключены из новой схемы. В конце процесса сжатия битов остаются только 48 бит. Следовательно, 8 бит отбрасываются. На изображении отсутствуют биты 9, 18, 22, 25, 35, 38, 43 и 54. Стоит отметить, что нумерация битов сбросилась после удаления битов четности (см. рис. 2.17).

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Рис. 2.19. Преобразование и сжатие ключа до 48 бит

На рис. 2.20 показан весь процесс *генерирования ключа*, который включает *удаление битов четности*, *сдвиг влево* и *сжатие*.

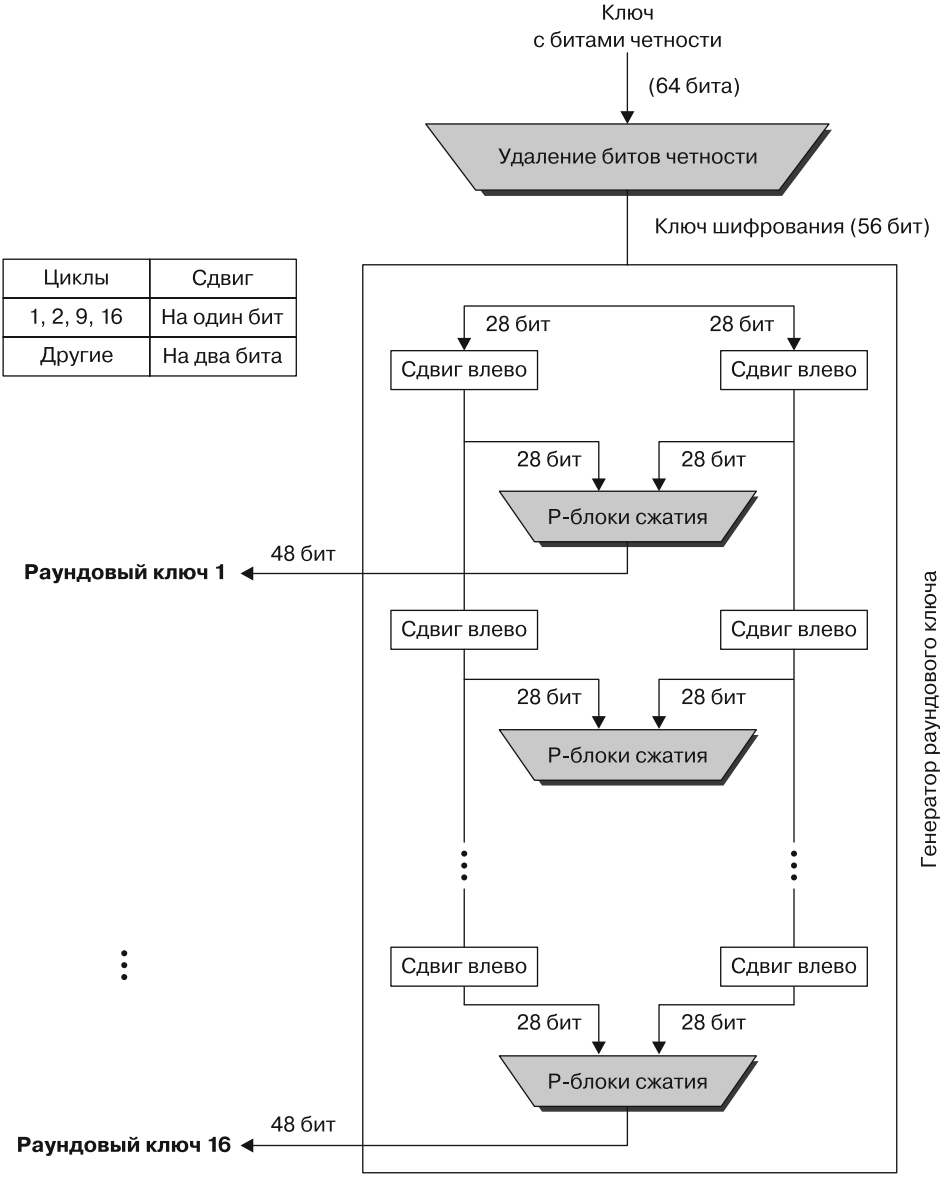


Рис. 2.20. Схема генерации ключа

Техника сжатия — запутывания — перестановки позволяет DES определять различные подключи размером 48 бит, по одному на каждый раунд. Благодаря этому данный алгоритм трудно взломать.

Шифрование

После того как мы сгенерировали ключ, можно приступать к шифрованию сообщения [M].

Схема шифрования DES состоит из трех этапов.

1. *Начальная перестановка* (initial permutation, IP). Сначала биты сообщения [M] переставляются с помощью функции, которую мы называем IP. Следует помнить, что эта операция с криптографической точки зрения не увеличивает безопасность алгоритма. После перестановки 64 бита делятся на две части так же, как в упрощенном DES: 32 бита L_0 и 32 бита R_0.
2. *Раунды шифрования*. Для $0 \leqslant i \leqslant 16$ выполняются следующие операции.
   - $L_i = R_{i-1}$.
   - Вычисление $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$.

     Здесь K_i — это 48-битная строка, полученная из ключа K (раундового ключа j), а f — функция расширения, аналогичная описанной ранее функции f для простого DES.
   - По сути, для $i = 1$ (первого раунда) мы имеем:

$$L_1 = R_0;$$

$$R_1 = L_0 \oplus f(R_0, K_1).$$

3. *Конечная перестановка*. Последняя часть алгоритма на 16-м раунде (последнем) состоит в следующем:
   - левая (L_{16}) и правая (R_{16}) части меняются местами, чтобы получилось (R_{16}, L_{16});
   - применяется перестановка, обратная к IP, — INV (inverse) — для получения шифротекста c, где c = INV (IP(R_{16}, L_{16})).

На рис. 2.21 представлена понятная схема шифрования DES.

Таким образом, мы выполнили сложный процесс генерации ключа в DES, в котором сформировали множество 48-битных подключей из 64-битного первичного ключа. Кроме того, мы прошли через три этапа шифрования: начальную перестановку, раунды шифрования и конечную перестановку.

Теперь, когда мы проанализировали процесс шифрования, можем перейти к расшифровке DES.

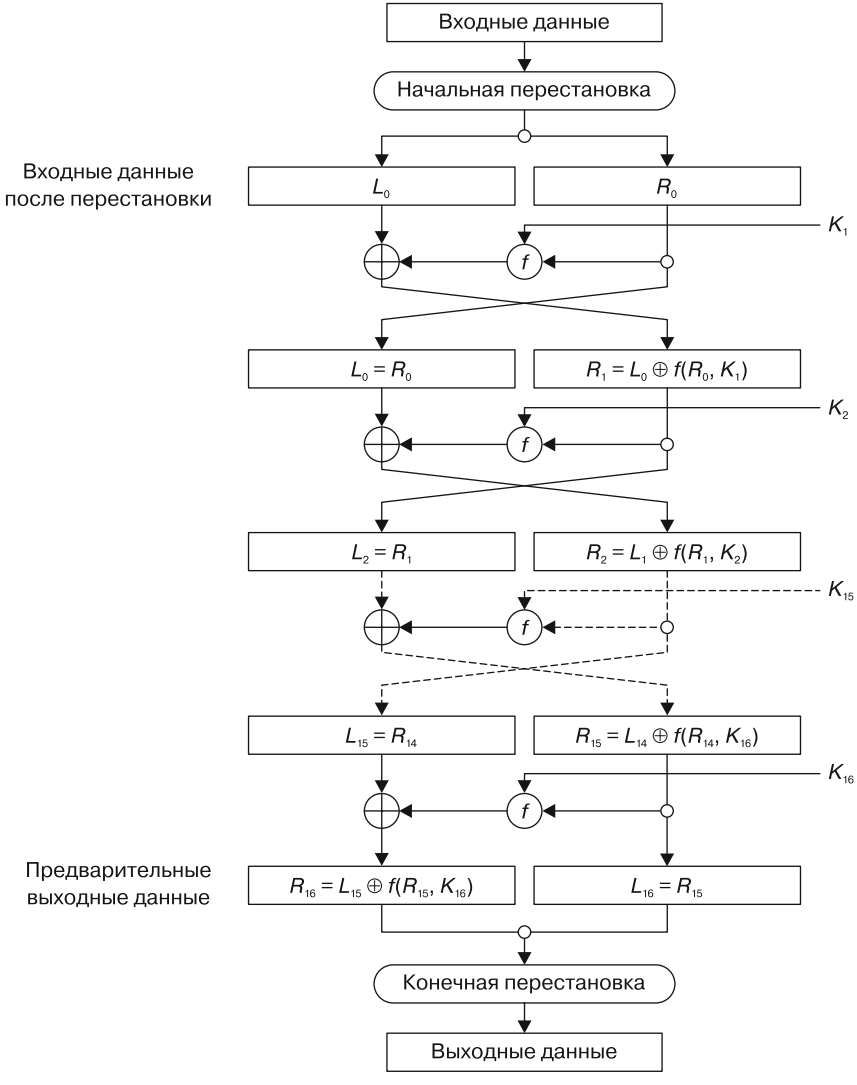


Рис. 2.21. Шифрование DES

Расшифровка

Расшифровка DES очень проста: чтобы получить открытый текст, нужно выполнить процесс шифрования в обратном порядке.

Расшифрование выполняется точно так же, как и шифрование, но с изменением порядка ключей с (K_1... K_{16}) на (K_{16}... K_1).

На рис. 2.22 процесс расшифровки изображен в виде блок-схемы.

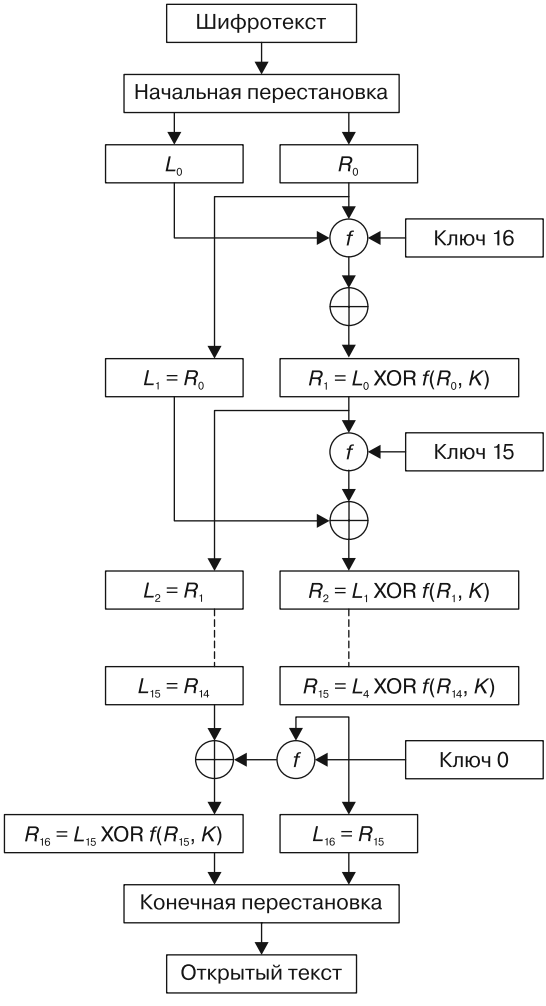


Рис. 2.22. Процесс расшифрования DES

Таким образом, описать процесс расшифрования можно так: над шифротекстом выполняется начальная перестановка, затем L_0 (левая часть) складывается с $R_0 = f(R_0, K_{16})$ с помощью операции XOR.

Эти действия производятся каждый раунд, затем выполняется конечная перестановка, и в итоге получается открытый текст.

Теперь, когда мы закончили с процессом работы алгоритма DES, перейдем к его анализу и уязвимостям.

Анализ алгоритма DES

Если немного углубиться в DES, то можно обнаружить несколько интересных вещей.

Одним из наиболее интересных этапов этого алгоритма является операция XOR, выполняемая между подключами (K_1, K_2... K_{16}) и половиной сообщения [M] в каждом раунде.

На этом этапе работает S-блок — функция f, описанная ранее в разделе об упрощенном DES.

Как мы уже знаем, S-блок является особой таблицей в DES, размеры которой были установлены АНБ: она состоит из 4 строк и 16 столбцов (S-блок 4 × 16). На рис. 2.23 показаны таблицы восьми S-блоков.

	S_i															
1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	0	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Рис. 2.23. Таблицы S-блоков в DES

Как можно заметить, в нее включены числа от 0 до (R_{i-1}), 16 – 1 = 15.

Рассмотрим подробнее S-блок для пятого раунда DES (рис. 2.24).

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	11	14	2	13	6	15	0	9	10	4	5	3

Рис. 2.24. S-блок для пятого раунда

Внимательный читатель заметит, что в 14-м столбце все числа однозначные: 0, 9, 3 и 4. Такая комбинация может представлять угрозу для безопасности.

Вас приведет в недоумение то, что я сейчас скажу, но поиграть с однозначными цифрами внутри S-блока не составит труда.

Внезапно у вас может возникнуть вопрос: почему ключ состоит только из 56 бит, а не из 64 бит? Потому что остальные 8 бит используются для проверки четности. Важное напоминание: ключ и S-блоки никак не связаны. Ключ используется для XOR с частью расширенного блока текста. Результат этого XOR попадает на этап использования S-блоков.

На самом деле исходный первичный ключ имеет длину 64 бита, при этом каждый 8-й бит отбрасывается. В итоге у ключа остается 56 бит (рис. 2.25).

1	2	3	4	5	6	7	**8**	9	10	11	12	13	14	15	**16**
17	18	19	20	21	22	23	**24**	25	26	27	28	29	30	31	**32**
33	34	35	36	37	38	39	**40**	41	42	43	44	45	46	47	**48**
49	50	51	52	53	54	55	**56**	57	58	59	60	61	62	63	**64**

Рис. 2.25. Биты, отбрасываемые при генерировании ключа DES

Вспомните момент, когда мы из 56 бит первичного ключа выбирали 48 бит для формирования подключей. Таким образом, 48 бит на входе дадут ровно 48 бит на выходе после выполнения операции XOR над [K_i].

Существует еще одна проблема, которая может возникнуть из-за метода шифрования, применяемого в DES. После начальной перестановки биты шифруются только в правой части с помощью функции $f(R_{i-1}, K_i)$. Вы можете спросить, не менее ли безопасен такой подход по сравнению с шифрованием всех битов. Если внимательно изучить схему, то можно заметить, что в каждом раунде биты перемещаются слева направо или справа налево, а затем шифруются. Эта техника похожа на процесс обертывания, она позволяет зашифровать все биты, а не только правую часть, как может показаться на первый взгляд.

Может возникнуть еще один вопрос: разве выполнение начальной и конечной перестановки, которая обратна начальной, даст полезный результат? Как я уже

говорил, в такой перестановке битов нет никакого криптографического смысла. Причина ее существования заключается в том, что в 1970-е годы вставка битов в аппаратуру была гораздо сложнее, чем сейчас. В заключение могу сказать, что в целом процесс шифрования в DES — с подстановками, перестановками, экспоненциальным расширением и сдвигом битов — порождает запутанность и рассеивание. Я изложил эту концепцию в начале раздела, когда упоминал принцип безопасности шифра, определенный Клодом Шенноном в его «Математической теории связи».

Атаки на DES

DES имеет богатую историю атак, предпринятых для его взлома с момента создания. В 1975 году в академическом сообществе появились сомнения по поводу стойкости ключа длиной 56 бит. На эту тему было опубликовано множество статей. В частности, одно из очень интересных предсказаний *Уитфилда Диффи* и *Мартина Хеллмана* (тех самых создателей алгоритма *обмена ключами Диффи — Хеллмана*, о котором пойдет речь в главе 3) заключалось в том, что компьютер стоимостью 20 млн долларов (в 1977 году) сможет взломать DES всего за 1 день.

Более 20 лет спустя, в 1998 году, *Фонд электронных рубежей* (Electronic Frontier Foundation, EFF) разработал специальный компьютер под названием «Взломщик DES» для конкурса DES Challenge 2. Призом за взлом DES и расшифровку шифротекста были 10 000 долларов. Фонд потратил чуть меньше 250 000 долларов и использовал 37 050 устройств, встроенных в 26 электронных плат, для создания суперкомпьютера, которому удалось расшифровать сообщение спустя 56 часов. В ходе DES Challenge 3 на взлом ушло всего 22 часа 15 минут: был использован простой *перебор всех возможных комбинаций битов*, задаваемых 256 возможными ключами (около 72 квадриллионов). Фонду удалось взломать DES с помощью аппаратного обеспечения, включающего 1500 микрочипов, работающих на частоте 40 МГц, за 4,5 машинного дня. Только представьте, что одному микрочипу понадобилось бы 38 лет для изучения всего набора ключей.

В этот момент правительство решило заменить DES новым алгоритмом с симметричным ключом. Так появился AES. Однако, прежде чем перейти к AES, проанализируем некоторые возможные атаки на DES.

Я выбрал несколько методов атаки на DES с учетом того, что именно они используются в большинстве случаев для взлома симметричных алгоритмов. Одни предназначены для атак на блочные шифры, другие применимы и к потоковым. Разница в том, что в потоковом шифре за один раз шифруется 1 байт, а в блочном шифре — около 128 бит (блок).

- *Метод грубой силы*. Этот базовый метод атаки, подразумевающий перебор всех возможных параметров для поиска ключа, может быть применен к любому известному шифру. Если вы помните, длина ключа DES

составляет 56 бит. В его случае не нужно перебирать все возможные ключи, поскольку, как доказал Мицуру Мацуи, статистически 16 раундов DES можно взломать с помощью 247 открытых текстов. Несмотря на сложность вычислений, DES признан взламываемым алгоритмом с начала 1990-х годов. Если интересно, можете прочитать статью Мицуру Мацуи здесь: https://link.springer.com/content/pdf/10.1007/3-540-48285-7_33.pdf.

- *Линейный криптоанализ*. Это статистический метод атаки по открытому тексту. Он не гарантирует стопроцентного успеха, но работает в большинстве случаев. Суть метода заключается в том, что атакующий пытается определить ключ шифрования посредством известного входного текста, а затем восстановить все выходные данные, сгенерированные этим ключом[1].
- *Дифференциальный криптоанализ*. Это технический метод, которому требуется анализ уязвимостей внутри DES (и других симметричных алгоритмов). Атакующий пытается выяснить открытый текст или ключ посредством выбранного открытого текста. В отличие от линейного криптоанализа, который работает с маловероятным известным открытым текстом, при дифференциальном криптоанализе используется заранее выбранный открытый текст.

И последнее, но не менее важное: уязвимым местом DES являются *слабые ключи*. Такие ключи просто не способны выполнить какое-либо шифрование. Они очень опасны, поскольку шифрование с их помощью возвращает обратно открытый текст. Слабые ключи хорошо известны в криптографии, и их следует избегать.

Слабые ключи возникают, когда на 16-м этапе генерации ключа получается 16 идентичных ключей.

Рассмотрим пример этой проблемы (для ясности уточню, что мы используем двоичную систему, без битов четности).

- Последовательность одинаковых битов — 0000000000000000 или 1111111111111111.
- Последовательность чередующихся битов — 0101010101010101 или 1010101010101010.

Во всех четырех случаях оказывается, что шифрование является автообратимым. Другими словами, если вы дважды выполните шифрование одного и того же шифротекста, то получите исходный открытый текст.

---

[1] На соревнованиях по информационной безопасности в формате Task-based CTF можно встретить задачи, основанные на подобных атаках. Обычно в качестве исходного текста передают строку из одних и тех же символов (например, «ааааа...» или «00000...»), чтобы упростить анализ и поиск ключа.

Triple DES

Как я упоминал ранее, одним из главных недостатков DES является длина ключа 56 бит. Таким образом, чтобы увеличить объем ключей и продлить их жизнь, была предложена новая версия DES в виде алгоритма *Triple DES*.

Логика 3DES такая же, как и у DES, с той лишь разницей, что здесь мы запускаем алгоритм три раза с тремя разными ключами. Схема 3DES показана на рис. 2.26.

Посмотрим, как работают этапы шифрования и расшифрования в DES, основываясь на схеме, приведенной на рис. 2.26.

Шифрование в 3DES происходит следующим образом.

1. Шифрование блоков открытого текста с помощью обычного шифра DES и ключа [K_1].
2. Расшифровка результата шага 1 с помощью обычного шифра DES и ключа [K_2].
3. Шифрование результата шага 2 с помощью обычного шифра DES и ключа [K_3].

Результатом шага 3 является шифротекст (*C*).

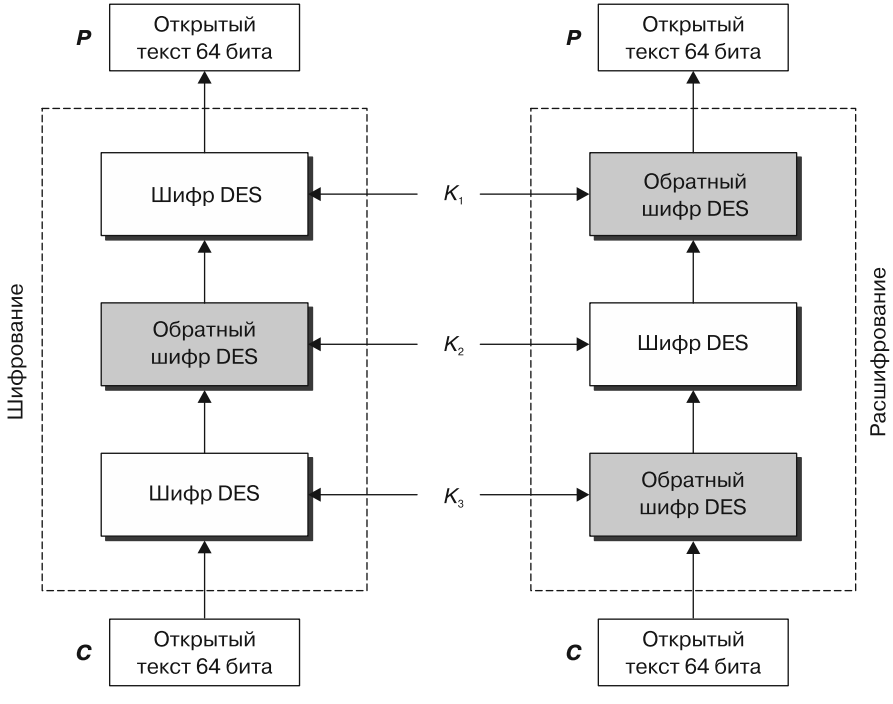


Рис. 2.26. Схема шифрования и расшифрования Triple DES

Расшифровка в 3DES

Расшифровка представляет собой обратный процесс. Сначала пользователь выполняет расшифровку с помощью $[K_3]$, затем шифрование с помощью $[K_2]$ и в конце расшифровку с помощью $[K_1]$.

DESX

Последний алгоритм семейства DES — *DESX*. Он представляет собой усиление ключа DES, предложенное Рональдом Ривестом (тем самым соавтором RSA).

Сам процесс шифрования и расшифрования в DESX такой же, как и в DES, при этом используются три выбранных ключа: $[K_1]$, $[K_2]$ и $[K_3]$.

Шифрование выполняется следующим образом:

$$C = [K_3] \oplus EK_1 ([K_2] \oplus [M]).$$

Сначала мы выполняем шифрование (E_K), объединяя $[K_2]$ и сообщение $[M]$ операцией XOR. К получившемуся результату применяем DES, используя $[K_1]$ размером 56 бит. Затем складываем E_{K_1} и $[K_3]$ с помощью операции XOR. Этот метод позволяет увеличить виртуальный ключ до 64 + 56 + 64 = 184 бит вместо обычных 56 бит (рис. 2.27).

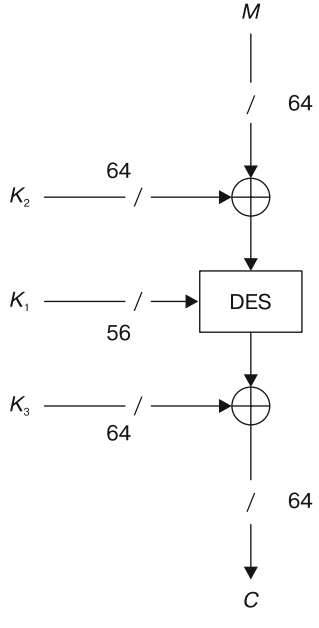


Рис. 2.27. Схема шифрования DESX

Изучив алгоритмы DES, 3DES и DESX, перейдем к еще одному столпу симметричного шифрования – AES.

AES Rijndael

В 2001 году после трехлетнего криптологического тестирования NIST (Национальный институт стандартов и технологий США) признал *AES*, также известный как *Rijndael*, самым надежным алгоритмом.

Среди 15 кандидатов, которые боролись за звание лучшего алгоритма, были выбраны пять финалистов: MARS (IBM), RC6 (RSA Laboratories), Rijndael (Джоан Даемен и Винсент Риймен), Serpent (Росс Андерсон и др.) и Twofish (Брюс Шнайер и др.). Все кандидаты были очень сильны, но в итоге Rijndael стал безусловным победителем.

Первый любопытный вопрос связан с его названием: как произносится Rijndael?

Произносить это название довольно сложно. На веб-странице двух авторов мы можем прочитать, что существует несколько способов произношения в зависимости от национальности и родного языка того, кто это делает[1].

Для начала скажу, что AES представляет собой блочный шифр, поэтому он может работать в разных режимах: *ECB* (уже встречающийся в DES), *сцепление блоков* (cipher block chaining, CBC), *блок обратной связи по шифротексту* (cipher feedback block, CFB), *блок обратной связи по выходу* (output feedback block, OFB) и *режим счетчика* (counter, CTR). В этом разделе рассмотрим основные различия между реализациями.

AES поддерживает разные размеры ключа: 128, 192 и 256 бит. С помощью конкурса NIST хотел найти очень сильный алгоритм, соответствующий таким характеристикам, как способность работать с блоками по 128 бит на входе и на различных типах аппаратного обеспечения, от 8-битных процессоров (также задействуемых в смарт-картах) до 32-битных архитектур, обычно используемых в персональных компьютерах. Помимо этого, алгоритм должен быть быстрым и надежным.

По моему мнению, при определенных условиях (о которых вы узнаете позже) это один из лучших алгоритмов на свете. Более того, я решил реализовать AES 256 в нашей системе защищенного поиска (CSE). Мы еще увидим CSE в главе 8. В настоящее время в Cryptolab мы используем AES для защиты симметричного шифрования данных, передаваемых между виртуальными машинами, которые шифруют и хранят данные.

---

[1] На русском языке можно произносить это как «рейндал».

Описание AES

Обсуждение одного только AES заслуживает отдельной главы. Однако я ограничусь общим описанием алгоритма и выскажу свои замечания и предложения. Если вам интересно узнать о нем больше, можете прочитать документ *FIPS PUB 197*, представленный NIST (опубликован 26 ноября 2001 года).

Для того чтобы избежать путаницы, необходимо отметить важный момент: я буду разбирать AES особым способом, не встречающимся в других работах. Его описание будет основано на разделении алгоритма на этапы, которые я назвал так: *расширение ключа* (key expansion, KE) и *первое добавление раундового ключа* (first add round key, F-ARK). Как вы увидите позже, каждый этап будет делиться на четыре шага: *преобразование SubBytes* (SB), или замена байтов, *преобразование ShiftRows* (SR), или сдвиг строк, *MixColumns* (MC), или смешивание столбцов, и *AddRoundKey* (ARK), или добавление раундового ключа. Главное — понять схему работы алгоритма. Все раунды работают аналогично, поэтому вы легко сможете сориентироваться в механизме из 10 раундов для 128-битного ключа, 12 раундов — для 192 бит и 14 раундов — для 256 бит.

Расширение ключа (KE) работает следующим образом.

1. Фиксированный ключ длиной 128 бит расширяется до длины ключа в зависимости от версии AES: 128, 192 или 256 бит.
2. Затем создаются подключи $[K_1]$, $[K_2]$... $[K_r]$ для шифрования каждого раунда (обычно с добавлением операции XOR к раунду).
3. AES использует особый метод, называемый *развертыванием ключа* Rijndael, для расширения короткого первичного ключа до определенного количества раундовых ключей.

Первое добавление раундового ключа (F-ARK) работает следующим образом.

Это первая операция. Алгоритм берет первый ключ, $[K_1]$, и добавляет его к функции *AddRoundKey*, используя побитовую операцию XOR текущего блока с частью расширенного ключа.

Раунды с R_1 *по* R_{n-1} работают следующим образом.

Каждый раунд, кроме последнего, делится на четыре шага, называемых уровнями, и включает следующие действия.

1. *Преобразование SB*. Это фундаментальный нелинейный шаг, выполняемый посредством определенного S-блока (принцип его работы мы видели в разделе, посвященном DES). S-блок AES показан на рис. 2.28.

2. *SR-преобразование*. Это перестановка бита, которая приводит к его рассеиванию на несколько раундов.
3. *MC-преобразование*. Этот шаг аналогичен SR, но применяется к столбцам.
4. *ARK*. Ключ раунда складывается с результатом предыдущего уровня с помощью операции XOR.

На рис. 2.28 представлен S-блок Rijndael в шестнадцатеричной системе счисления.

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
X	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	a	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	b	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	c	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	d	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	e	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	f	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Рис. 2.28. S-блок Rijndael

В последнем раунде выполняются все операции предыдущих раундов, кроме третьего уровня — MC. В результате выполнения всех перечисленных процессов в каждом раунде *n* раз (в зависимости от размера ключа — 14 раундов при 256-битном ключе) получается шифротекст *C*, зашифрованный AES (рис. 2.29).

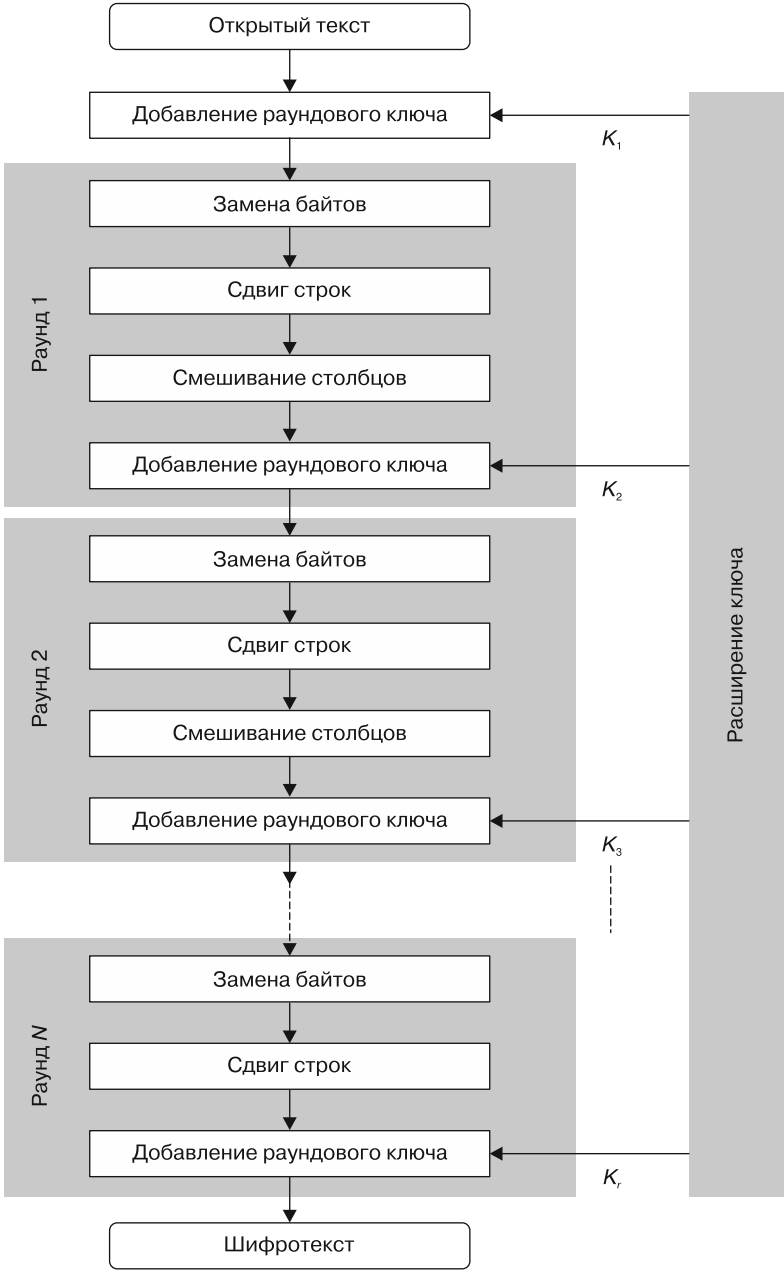


Рис. 2.29. Схема шифрования AES

Таким образом, весь процесс шифрования AES можно представить в виде математической функции (рис. 2.30).

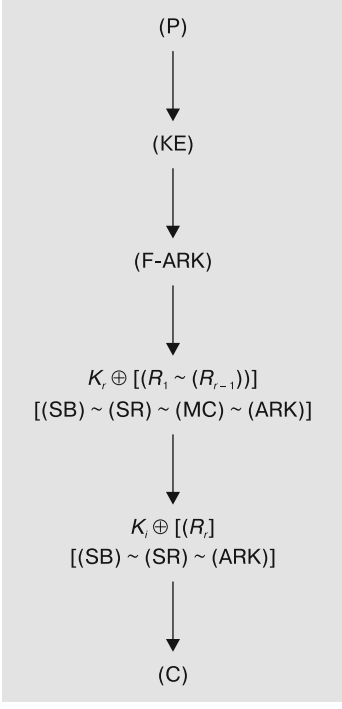


Рис. 2.30. Представление блок-схемы AES в виде математической функции

Согласно предложенной схеме, мы имеем следующее.

- Выражение $K_r \oplus [(R_1 \sim (R_{r-1}))]$ описывает все математические процессы, которые происходят между ключами каждого раунда (K_r). С помощью операции XOR эти ключи объединены с внутренними функциями каждого раунда, начиная с первого (после F-ARK) и заканчивая последним (исключенным). Таким образом, внутри $[(R_1 \sim (R_{r-1}))]$ находятся функции $[(SB) \sim (SR) \sim \sim (MC) \sim (ARK)]$.
- В последнем раунде этап *MC* отсутствует.

На рис. 2.31 показан весь процесс шифрования и расшифрования в AES.

После схематичного описания операций шифрования и расшифрования в AES рассмотрим атаки и уязвимости этого алгоритма.

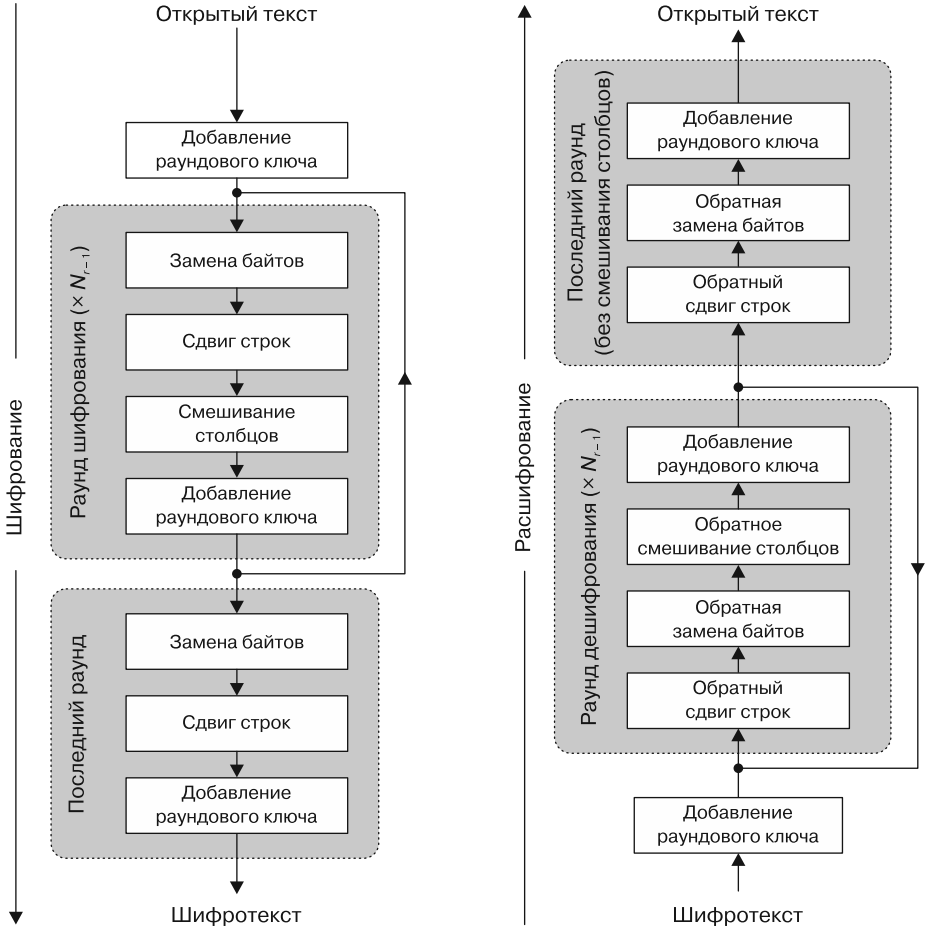


Рис. 2.31. Схема шифрования и расшифрования в AES

Атаки и уязвимости AES

В публикациях АНБ и NIST AES называют неуязвимым для всех известных видов атак.

Однако AES имеет уязвимости, как, собственно, и любая реализуемая система.

Второго октября 2000 был опубликован отчет NIST о возможных уязвимостях AES. Его можно прочитать здесь: https://www.nist.gov/news-events/news/2000/10/commerce-department-announces-winner-global-information-security.

В отчете NIST говорится следующее:

> *«Каждый из алгоритмов-кандидатов должен был поддерживать размеры ключей 128, 192 и 256 бит. Для 128-битного ключа существует примерно 340 000 000 000 000 000 000 000 000 000 000 000 (340 с 36 нулями) возможных ключей».*

Но, несмотря на то что теоретически AES нельзя взломать ($-x$ %), в нем, как и в любом алгоритме, всегда можно найти брешь, если использовать всю вычислительную мощность мира (компьютерный анализ AES вы увидите в главе 8). Обычно уязвимости обнаруживаются на этапе реализации. Обратите внимание на то, что произойдет, если вы реализуете, например, AES в режиме электронной кодовой книги (ECB). (Мы уже встречали режим ECB в DES.) Эта базовая реализация заключается в том, чтобы разделить открытый текст на блоки и для каждого блока открытого текста P вычислить шифротекст C:

$$C = \text{Шифр}(P).$$

Схема шифрования в режиме ECB показана на рис. 2.32.

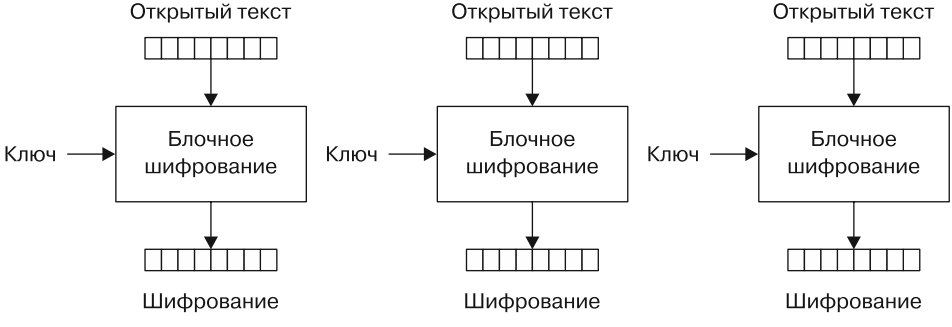
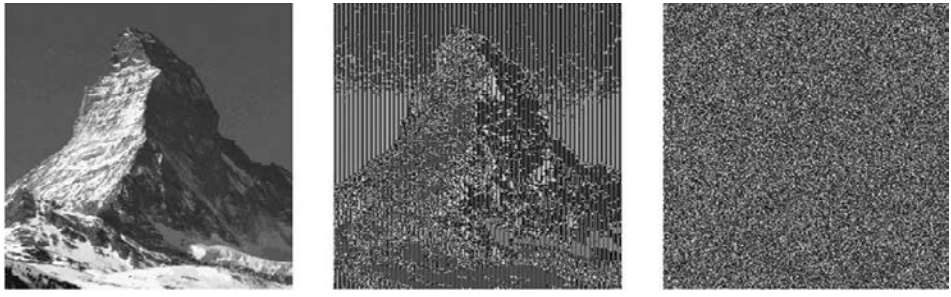


Рис. 2.32. Шифрование в режиме ECB

Реализация AES в режиме ECB может вызвать серьезные проблемы. Например, на рис. 2.33 видно, что при шифровании изображения горы Червино (или Маттерхорн) в режиме ECB можно распознать содержимое, даже если оно зашифровано.

Другими словами, при использовании режима ECB исчезает эффект шифрования, который должен иметь такой же вид, как на третьем изображении (результат шифрования в *другом режиме*).

При другой атаке на ECB, известной как *повтор блока*, знание пары «открытый текст — шифротекст» позволяет злоумышленнику без ключа многократно пересылать известный шифротекст.



Оригинальное изображение | Результат шифрования в режиме ECB | Результат шифрования в другом режиме

Рис. 2.33. Изображение горы Червино: оригинальное, зашифрованное в режиме ECB (неудачное) и зашифрованное в другом режиме

Интересный пример этой уязвимости режима ECB привел Кристофер Свенсон в своей книге «Современный криптоанализ». Для того чтобы обмануть Боба и Алису на этапе обмена информацией, Ева (злоумышленник)[1] может повторно отправить блок известного открытого текста и получить значительную выгоду.

Рассмотрим гипотетический сценарий, связанный с атакой на ECB. Алиса подходит к банкомату, чтобы снять деньги. Предполагается, что связь между ней и банком через банкомат зашифрована с использованием режима AES-ECB.

Зашифрованное сообщение между Алисой (с ключом [*K*]) и банком выглядит следующим образом.

1. Банкомат: шифрование [*K*] (имя: Алиса Смит, номер счета: 123456, сумма: 200 долларов).
2. Предположим, что это сообщение, зашифрованное с помощью AES, на выходе имеет такой вид:

   CF A3 1C F4 67 T3 2D M9...

3. Ответ банка после проверки счета: [OK].

Если Ева перехватывает зашифрованное сообщение, она может просто повторить эту операцию несколько раз, пока не украдет все деньги со счета Алисы. А банк подумает, что Алиса снимает деньги несколько раз. Если не предпринять никаких действий против этой атаки, то жертва (Алиса) потеряет все деньги, ко-

---

[1] В Интернете вы можете найти списки персонажей, которые используются в информационной безопасности. У каждого персонажа есть имя, которое может описать его роль. Например, Ева (Eve) — пассивный злоумышленник, от англ. eavesdropper («подслушивающий»).

торые были у нее на счету. Этот трюк работает потому, что Ева повторно отправляет одно и то же сообщение несколько раз. Если в каждом сеансе шифрования используется один и тот же ключ [*K*], Ева может попытаться осуществить эту атаку. Одним из эффективных способов защиты информации является использование уникальных криптограмм для каждой сессии. Это устраняет необходимость в применении симметричного шифрования для многократной передачи данных. В качестве альтернативного решения по предотвращению этого вида атак можно реализовать AES (как и другие блочные шифры) в режиме CBC.

CBC выполняет шифрование блока, генерируя выходные данные на основе значений предыдущих блоков.

В главе 8, в разделе «Анализ вычислительной эффективности CSE», я представлю систему защищенного поиска, в которой реализовано шифрование AES в режиме CBC.

Здесь же просто объясню принцип работы режима CBC.

Метод CBC использует блоки размером 64 бита для открытого текста, шифротекста и *вектора инициализации* (initialization vector, IV). По сути, IV — это случайное число, иногда называемое *солью*, которое добавляется к открытому тексту с помощью операции XOR для вычисления блока.

Сейчас просто запомните:

- *E* (Encryption) — шифрование;
- *D* (Decryption) — расшифровка;
- *C* (Ciphertext) — шифротекст.

Шифрование происходит следующим образом.

1. Начальный блок C_0 вычисляется добавлением первого блока открытого текста P_0 к IV с помощью операции XOR:

$$C_0 = E(P_0 \oplus \text{IV}).$$

2. Каждый последующий блок вычисляется добавлением предыдущего блока шифротекста к блоку открытого текста с помощью операции XOR, после чего шифруется:

$$C_i = E(P_i \oplus C_{i-1}).$$

Расшифровка происходит следующим образом.

1. Чтобы получить обратно открытый текст P_0, необходимо объединить расшифровку первого блока полученного шифротекста C_0 и IV с помощью операции XOR:

$$P_0 = D(C_0) \oplus \text{IV}.$$

2. Для получения всех остальных открытых текстов, P_i, необходимо выполнить операцию XOR между расшифровкой полученного шифротекста C_i и всеми остальными шифротекстами, кроме первого:

$$P_i = D(C_i) \oplus C_{i-1}.$$

На рис. 2.34 представлена схема шифрования в режиме CBC.

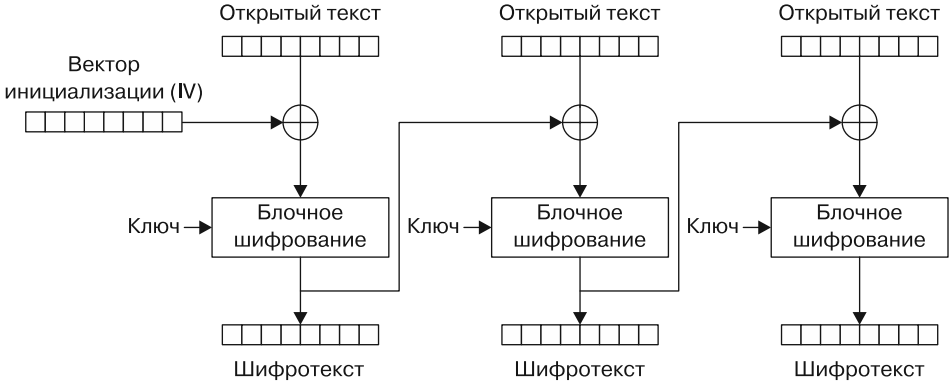


Рис. 2.34. Схема шифрования в режиме CBC

AES — надежный симметричный алгоритм, и до 2009 года единственными успешными атаками на него были так называемые атаки по сторонним каналам. Эти виды атак в основном связаны с реализацией AES в некоторых конкретных приложениях.

Далее приведен список атак по сторонним каналам.

- *Атака на кэш*. Обычно некоторая информация хранится в кэш-памяти (разновидность памяти второго порядка в компьютере). Если злоумышленник удаленно отслеживает доступ к кэш-памяти, он может украсть ключ или открытый текст. Чтобы не допустить этого, необходимо регулярно чистить кэш-память.
- *Атака по времени*. Это метод, использующий время выполнения шифрования. Другими словами, он основан на корреляции между временем шифрования и значениями параметров. Если злоумышленник знает часть сообщения или часть ключа, он может сравнить реальное и смоделированное время выполнения.

  По сути, это скорее не логическая, а *физическая* атака на плохую реализацию кода. В любом случае она относится не только к AES, но и к RSA, алгоритму Диффи — Хеллмана и другим алгоритмам с корреляцией параметров.

- *Атака по замерам потребляемой мощности.* Как и в случае с атакой по времени, здесь могут быть использованы потенциальные уязвимости, присущие аппаратной реализации. Перейдя по ссылке https://www.tandfonline.com/doi/full/10.1080/23742917.2016.1231523, вы можете прочитать об интересном эксперименте, связанном с корреляцией энергопотребления 128-битной реализации AES на Arduino Uno. С помощью функций ARK и SB алгоритма атакующий смог отследить энергопотребление устройства и получить полный 16-байтный ключ шифрования. Любителям аппаратного обеспечения эта атака очень понравится.
- *Электромагнитная атака.* Это еще один вид атаки, осуществляемой на реализацию алгоритма. Одна из таких атак была предпринята на *полевую программируемую вентильную матрицу (ППВМ)* — с помощью осциллографа измерялось излучение, исходящее от антенны.

Наконец, настоящей проблемой AES является обмен ключами. Поскольку это симметричный алгоритм, Алиса и Боб должны договориться об общем ключе, чтобы выполнить необходимое шифрование и расшифрование. Существуют реализации AES без обмена ключами, однако большинство из них нуждаются в асимметричных алгоритмах, чтобы компенсировать отсутствие передачи ключей. Вы лучше поймете эту концепцию в следующей главе, когда мы будем рассматривать асимметричное шифрование. Более того, в главе 8 вы увидите приложение, не нуждающееся в обмене ключами.

Резюме

Мы рассмотрели некоторые из лучших симметричных алгоритмов и их особенности. Тем не менее вопрос, поставленный во введении к этой главе, остается открытым: «Как реализовать симметричные алгоритмы, которые были бы одновременно надежными (в смысле безопасности) и простыми в выполнении (в вычислительном плане)?» Один из возможных ответов — алгоритм AES: он надежен и одновременно прост в вычислительном плане. Изучив симметричное шифрование, вы сделали первые шаги к его пониманию.

Сначала вы изучили основы симметричного шифрования. Были рассмотрены булевы операции, необходимые для понимания симметричного шифрования, расширение ключа и функции S-блока. Затем мы подробно разобрали, как работают простой DES, обычный DES, 3DES и DESX, а также каковы основные уязвимости и атаки, которым подвержено это семейство алгоритмов.

Мы проанализировали AES (Rijndael), включая схему его реализации и логику выполнения шагов, которые делают этот алгоритм таким стойким. Что касается уязвимостей AES и атак на него, было показано, как разница между режимами

ECB и CBC может сделать этот алгоритм уязвимым для атак на реализацию блочного шифра.

Наконец, мы рассмотрели некоторые из наиболее известных атак по сторонним каналам, применимых к большинству криптографических алгоритмов.

Эти темы очень важны, и теперь вы знаете, как реализовать криптографический симметричный алгоритм, и лучше понимаете его особенности. Следующие главы очень тесно связаны с этой частью книги. Глава 8 расскажет о системе защищенного поиска, при котором AES используется в качестве одного из алгоритмов для передачи зашифрованных файлов в облачное хранилище.

Теперь, когда вы узнали об основах симметричного шифрования, пришло время рассмотреть асимметричное шифрование.

3 Алгоритмы асимметричного шифрования

Асимметричное шифрование подразумевает использование разных пар ключей для шифрования и расшифрования сообщения. Этот метод также называют шифрованием с открытым и закрытым ключами. Однако асимметричное шифрование и шифрование с открытым и закрытым ключами имеют кое-какие различия, о которых будет рассказано в этой главе. Мы начнем с истории этого революционного метода шифрования и расшифрования, а затем рассмотрим различные типы асимметричных алгоритмов и то, как они помогают защищать наши кредитные карты, личные и другие данные.

Введение в асимметричное шифрование

Важнейшей функцией шифрования с закрытым и открытым ключами является обмен ключами между двумя сторонами, а также обеспечение безопасности передачи данных.

Чтобы получить исчерпывающее представление о том, что такое асимметричное шифрование, необходимо рассмотреть основные этапы его развития. Этот вид криптографии занимает важное место в нашей повседневной жизни. Именно асимметричное шифрование используется для защиты наших финансовых секретов (номеров кредитных карт и банковских данных в Интернете), создания паролей, которыми мы постоянно пользуемся, а также в целом для безопасного обмена конфиденциальной информацией и защиты частной жизни.

Познакомимся с историей этого увлекательного способа шифрования.

История асимметричной криптографии началась в конце 1970-х годов, а получила развитие в 1980-х, когда с возникновением Интернета и цифровой экономики компьютеры начали появляться в домах обычных людей. Конец 1970-х и 1980-е годы — это период, когда Стив Джобс основал компанию Apple Inc., а между США и СССР все еще шла холодная война. Это также было временем экономического бума во многих западных странах, в частности в Италии, Франции и Германии. Именно тогда появился Интернет.

Противостояние западного и восточного блоков — США и их союзников, с одной стороны, и советского блока — с другой, — стало причиной появления противоборствующих шпионских сетей, центры которых находились в разделенном Берлине. В этот период ситуация с обменом ключами для симметричного шифрования дошла до критической точки: служба безопасности связи США (COMSEC), отвечающая за передачу криптографических ключей, ежедневно передавала тонны ключей, что в конечном счете привело к серьезным угрозам безопасности.

Так, в 1970-х годах банки использовали алгоритм DES и отправляли ключи с курьером, который должен был передавать их лично в руки клиенту. Несмотря на доступ к самым большим вычислительным ресурсам в мире, Агентство национальной безопасности США долго билось над проблемой распределения ключей. Эта задача казалась неразрешимой даже для крупных корпораций, занимающихся решением сложнейших проблем, связанных с будущим мира. С ней не смогла справиться даже RAND — организация с широкими вычислительными возможностями, целью которой было решение проблем будущего и предотвращение критических сбоев. По моему мнению, иногда «критический сбой» действительно необходим просто для того, чтобы прояснить ситуацию, а не игнорировать ее. Ту или иную проблему можно решить разными способами. В случае с асимметричным шифрованием никакие государственные средства или суперкомпьютеры с бесконечными вычислительными возможностями и множеством задействованных умов не смогли решить проблему, которая на первый взгляд кажется довольно простой.

Теперь, когда вы имеете представление о главной проблеме, которую решает асимметричное шифрование, — обмене ключами (на самом деле вы увидите, что в RSA она превращается в прямую передачу сообщения), давайте познакомимся с первопроходцами этого чрезвычайно интригующего вида криптографии.

Первопроходцы

В криптографическом обществе, как и в любом другом, можно встретить как интровертов, так и экстравертов. К числу последних я бы отнес Уитфилда Диффи, независимого вольнодумца, не работающего на правительство или какую-либо крупную корпорацию. Я впервые встретил Уитфилда на конференции в Сан-Франциско в 2016 году, когда он обсуждал криптографию со своими знаменитыми коллегами Мартином Хеллманом и Рональдом Ривестом. Одной из самых ярких деталей, врезавшихся в мою память, был элегантный белый костюм. Будучи довольно высоким, с длинными светлыми волосами и бородой, в нем Диффи был похож на вечно молодого парня из 1960-х годов, чей современник мог бы быть агентом на фондовой бирже Уолл-стрит или индийским

праведником. Уитфилд Диффи — один из отцов современной криптографии, и его имя навсегда останется в истории шифрования с открытым и закрытым ключами. Диффи родился в 1944 году. В 1965-м он окончил Массачусетский технологический институт в Бостоне. После завершения учебы работал в области кибербезопасности, а затем стал одним из самых авторитетных независимых криптографов 1970-х годов. Его называли *киберпанком* в честь *новой волны* научной фантастики того периода, в которой кибернетика, искусственный интеллект и культура хакеров слились в один антиутопический футуристический антураж, который, как правило, фокусировался на *сочетании низкого уровня жизни и высоких технологий*.

В 1960-х годах Министерство обороны США основало новую исследовательскую программу в области связи — *Управление передовыми исследовательскими проектами* (Defense Advanced Research Projects Agency, DARPA), также известное как *ARPA*. Основная задача ARPA заключалась в объединении военных компьютеров в общую сеть для обеспечения более надежного уровня безопасности передачи данных. Этот проект был призван предотвратить отключение связи в случае ядерной атаки. Однако созданная сеть позволила также пересылать сообщения между учеными и выполнять расчеты, используя свободные мощности подключенных компьютеров. Сеть *ARPANET* была официально запущена в 1969 году и объединила тогда всего четыре сайта. Тем не менее она так быстро развивалась, что в 1982 году уже превратилась в Интернет. В конце 1980-х годов к Интернету было подключено множество обычных пользователей, и их число стремительно увеличивалось.

Еще в период развития ARPANET Уитфилд Диффи начал задумываться о том, что однажды у каждого человека будет компьютер, с помощью которого он сможет обмениваться электронными письмами с другими людьми. Он также представлял себе мир, в котором товары продаются через Интернет, а вместо реальных денег используются пластиковые карты. В центре внимания Диффи находились вопросы обеспечения конфиденциальности и безопасности данных. Он постоянно размышлял о проблемах дистанционного общения, когда человек не имеет ни малейшего представления о том, кто находится на противоположном конце провода. Более того, шифрование сообщений и документов часто использовалось при пересылке очень ценной информации. Шифрование стало общедоступным способом скрывать информацию и делиться секретами, и криптография перестала быть уделом только военных, правительства или ученых.

Основной вопрос заключался в следующем: если два совершенно незнакомых человека встретятся друг с другом в Интернете, как с помощью математических параметров они могут зашифровать (или расшифровать) передаваемый документ, не обмениваясь никакой дополнительной информацией, кроме самого документа? В этом и заключается суть проблемы обмена ключами.

Однажды в 1974 году Диффи выступал с докладом в исследовательском центре IBM имени Томаса Дж. Уотсона, где рассказал о различных стратегиях решения проблемы распределения ключей. Слушатели отнеслись к предложенным Диффи методам весьма скептически, и лишь один человек, старший криптограф IBM, поддержал его. От него Диффи узнал, что недавно лабораторию посетил профессор из Стэнфорда, у которого было похожее видение проблемы, — Мартин Хеллман.

Воодушевленный Диффи в тот же вечер поехал на противоположный конец США в Пало-Альто (Калифорния), чтобы навестить профессора Хеллмана. Сотрудничество Диффи и Хеллмана останется известным в криптографии благодаря созданию одного из самых красивых алгоритмов в этой области — *обмена ключами Диффи — Хеллмана*.

Мы проанализируем этот новаторский алгоритм в следующем разделе.

Алгоритм Диффи — Хеллмана

В понимании сути алгоритма Диффи — Хеллмана нам помогут так называемые *мысленные эксперименты*, или *ментальная репрезентация* теории, — то, что часто применял в своей работе Эйнштейн.

Мысленный эксперимент — это гипотетический сценарий, в котором вы мысленно переноситесь в более реальную ситуацию, чем при чисто теоретическом подходе к вопросу. Например, Эйнштейн объяснил теорию относительности с помощью очень популярного мысленного эксперимента: он использовал метафору движущегося поезда, за которым наблюдают зрители с разных позиций — изнутри и снаружи поезда.

В этой книге я буду часто прибегать к таким образным репрезентациям.

Представим, что Алисе и Бобу нужно обменяться каким-то сообщением (на бумаге), но главный почтамт города проверяет содержимое всех писем. Алиса и Боб всячески стараются придумать, как отправить письмо тайно, избежав лишних глаз. В какой-то момент они понимают, что Алиса может положить сообщение в металлический ящик и отправить его Бобу. Но, поскольку у Боба не будет ключа от этого ящика, нашим героям придется сначала где-то встретиться для передачи ключа. Снова мы возвращаемся к проблеме обмена ключами.

Многочисленные попытки решить эту проблему не давали результата. Казалось, что логического решения вовсе не существует, пока однажды Диффи при поддержке Хеллмана не нашел выход, о котором Хеллман позже скажет: «Бог награждает дураков!»

Представим, что должны сделать Алиса и Боб, чтобы решить проблему с обменом ключами.

1. Алиса кладет свое секретное сообщение в металлический ящик и запирает его железным замком. Затем она отправляет этот ящик Бобу, оставляя ключ от него у себя. Обратите внимание: Алиса запирает ящик своим ключом и не отдает его Бобу.
2. Боб устанавливает на ящик еще один замок, используя свой ключ, и отправляет его обратно Алисе. Таким образом, получив ящик в первый раз, Боб не может его открыть. Он просто добавляет к нему еще один замок.
3. Получив обратно ящик, Алиса может снять свой замок. Ящик остается запертым только на замок Боба, как показано на рис. 3.1. Затем Алиса отправляет его Бобу в последний раз.
4. Когда Боб снова получает ящик, он может открыть его и наконец прочитать сообщение от Алисы, которое все это время находилось внутри.

Таким образом, Алиса и Боб ни разу не встретились друг с другом, чтобы обменяться своими ключами от замка. Обратите внимание: ящик дважды перемещался между ними. Алгоритм в этом моменте работает иначе (рис. 3.1).

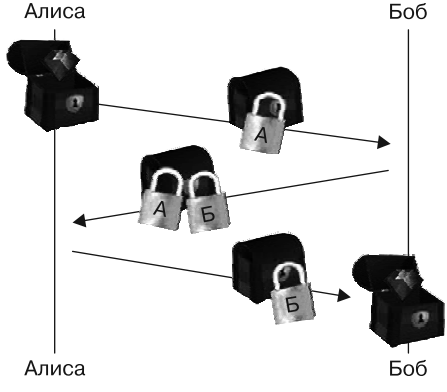


Рис. 3.1. Алгоритм Диффи — Хеллмана на примере Алисы и Боба

Приведенный пример не совсем точно описывает работу алгоритма. Однако он дает объяснение практического решения проблемы обмена ключами, которую считали неразрешимой.

Теперь мы должны перенести это практическое обоснование в логическое математическое представление.

Прежде всего вернемся к модульной арифметике, воспользовавшись некоторыми особыми свойствами операций, выполняемых в конечных полях.

Задача дискретного логарифма

Я постараюсь объяснить математику, лежащую в основе асимметричного шифрования, без излишних обозначений, просто потому, что не хочу вас путать или превращать эту книгу в обширную диссертацию по математике.

Когда мы говорим о конечном (дискретном) поле, то подразумеваем конечную группу (n) целых чисел (Z), лежащих на кольце, — условно (Z_n). На эту группу чисел распространяются все известные нам математические законы, например операции со стандартными целыми числами. Поскольку мы работаем в конечном поле, называемом *модулем n*, нам придется рассмотреть некоторые важные вопросы, связанные с модульной арифметикой. Как вы видели в предыдущей главе, выполнение операций *по модулю* означает возврат отсчета к первому числу каждый раз, когда мы доходим до конца заданного множества. В качестве примера можно привести часы, на которых после 12 отсчет времени снова начинается с 1.

По сути, в конечном поле существует *числовой период*, в рамках которого числа и результаты операций повторяются. Если конечное поле состоит из семи целых чисел {0, 1, 2, 3, 4, 5, 6} (обозначается Z_7), то все результаты операций, выполняемых внутри него, остаются в пределах этого множества.

Приведу небольшой пример операций сложения и умножения в конечном поле (Z_7, +, ·). Здесь важно учитывать, что все результаты операций (по модулю 7), даже если они превышают число 7, будут оставаться в пределах заданного диапазона:

$$1 \cdot 1 \equiv 1 \pmod 7;$$
$$2 \cdot 4 \equiv 1 \pmod 7;$$
$$3 + 5 \equiv 1 \pmod 7;$$
$$3 \cdot 5 \equiv 1 \pmod 7.$$

Модальность обусловливает использование знака «≡» вместо «=». Знак «≡» говорит о том, что математика работает не так, как нам рассказывали в начальной школе. Например, 2 + 2 = 4 в конечном поле по модулю 3 будет иметь вид:

$$2 + 2 \equiv 1 \pmod 3.$$

Из школьной математики мы знаем, что логарифм $\log a(z)$ — это функция с основанием a. Необходимо определить, в какую степень нужно возвести a, чтобы получить число z. Например, при $a = 10$ и $z = 100$ логарифмом является число 2. Тогда мы говорим, что логарифм числа 100 по основанию 10 равен 2, и записываем это выражение как $\log [10, 100] = 2$.

В ходе работы в дискретном поле все становится сложнее, поскольку вместо обычного логарифма необходимо вычислить дискретный логарифм. Допустим, нам нужно решить уравнение:

$$a^{[x]} \equiv b \pmod{p}.$$

Даже если мы знаем значения (a) и (b), решить эту задачу очень трудно, потому что не существует эффективного алгоритма для вычисления дискретного логарифма, то есть [x].

Квадратные скобки обозначают секретность [x]. Технически [x] является дискретной степенью. Однако задача дискретного логарифма и задача поиска дискретной степени имеют одинаковую вычислительную сложность.

Давайте копнем поглубже и посмотрим, как работает эта математическая операция. Возьмем следующий пример:

$$2^4 \equiv (x) \pmod{13}.$$

Сначала мы вычисляем $2^4 = 16$, затем делим $16 / 13 = 1$ с остатком 3. Таким образом, $x = 3$.

Задача дискретного логарифма представляет собой обратную операцию:

$$2^{[y]} \equiv (x) \pmod{13}.$$

В данном случае нам нужно вычислить [y], зная, что основание равно 2. Существует бесконечное множество решений для [y], порождающих (x).

Преобразуя предыдущий пример, мы получаем:

$$2^{[y]} \equiv 3 \pmod{13}$$

для [y].

Одно из решений — $y = 4$, но оно не единственное.

Результат 3 получается при всех целых числах (n) уравнения:

$$[y] = [y + (p - 1)n].$$

Например, при $n = 1$:

$$2^{[4 + (13 - 1) \cdot 1]} \equiv 2^{16} \equiv 3 \pmod{13}.$$

При $n = 2$:

$$2^{[4 + (13 - 1) \cdot 2]} \equiv 2^{28} \equiv 3 \pmod{13}.$$

И так далее.

Следовательно, уравнение имеет бесконечное количество решений для всех целых чисел, то есть ($n \geqslant 0$):

$$[y] \equiv 2^{[4 + 12n]} \pmod{13}.$$

Пока не существует метода решения задачи дискретного логарифма за полиномиальное время. В криптографии, как и в математике, эта задача считается очень трудноразрешимой даже при больших вычислительных возможностях.

Наконец, необходимо ввести определение и обозначение образующего элемента (g). Элемент (g) представляет собой конкретное число, которое порождает всю группу (Z_p). Если (p) — простое число, следовательно, (g) может принимать любое значение от 1 до $p - 1$.

Объяснение алгоритма Диффи — Хеллмана

Алгоритм Диффи — Хеллмана не совсем асимметричный алгоритм шифрования. Правильнее будет назвать его алгоритмом с открытым и закрытым ключами или алгоритмом согласования ключей. Разница заключается не только в названии, но и в сути. Более понятными эти различия станут в разделе, посвященном чисто асимметричному алгоритму RSA. Особенностью алгоритма Диффи — Хеллмана является то, что он позволяет создать общий ключ, который затем используется для симметричного шифрования сообщения [M].

Алгоритм симметричного шифрования комбинируется с передачей общего ключа Диффи — Хеллмана для создания криптограммы C:

$$\text{симметричный алгоритм } E([k], [M]) = C.$$

Другими словами, мы используем алгоритм Диффи — Хеллмана для создания общего секретного ключа [k], а затем с помощью AES или другого симметричного алгоритма шифруем сообщение [M].

Алгоритм Диффи — Хеллмана не шифрует секретное сообщение напрямую, а лишь определяет общий ключ между двумя сторонами. Как вы увидите далее, это очень важный момент.

Дискретные поля и задача дискретного логарифма, защищающая общий ключ от злоумышленников, позволили Диффи и Хеллману реализовать один из самых надежных и известных алгоритмов в криптографии.

Давайте посмотрим, как он работает.

1. Сначала Алиса и Боб договариваются о следующих параметрах: образующем элементе (g) в кольце (Z_p) и простом числе p (для проведения операций mod p).

   Алиса выбирает секретное число [a], а Боб — секретное число [b].

   Алиса вычисляет $A \equiv g^a \pmod{p}$, а Боб — $B \equiv g^b \pmod{p}$.

2. Алиса посылает (A) Бобу, а Боб посылает (B) Алисе.
3. Алиса вычисляет $k_A \equiv B^a \pmod p$, а Боб вычисляет $k_B \equiv A^b \pmod p$. Таким образом, [$k_A = k_B$] будет общим секретом Алисы и Боба.

Далее приведен численный пример этого алгоритма.

1. Алиса и Боб договариваются об используемых параметрах: $g = 7$ и $p = 11$. Алиса выбирает секретное число (3), а Боб — секретное число (6). Алиса вычисляет $7^3 \pmod{11} \equiv 2$, а Боб — $7^6 \pmod{11} \equiv 4$.
2. Алиса посылает (2) Бобу, а Боб посылает (4) Алисе.
3. Алиса вычисляет $4^3 \pmod{11} \equiv 9$, а Боб вычисляет $2^6 \pmod{11} \equiv 9$.

Число [9] является общим секретным ключом [k] Алисы и Боба.

Анализ алгоритма

На шаге 1 Алиса вычисляет $A \equiv g^a \pmod p$, а Боб — $B \equiv g^b \pmod p$. Алиса и Боб обменялись (A) и (B) — открытыми параметрами односторонней функции. *Односторонняя функция* названа так потому, что задача дискретного логарифма, лежащая в ее основе (см. подраздел «Задача дискретного логарифма» ранее), делает невозможным ($-x$ %) вычисление секретного закрытого ключа [a] на основании открытого параметра (A). То же самое верно для [b] и (B).

Еще одно свойство модулярных сил заключается в том, что можно записать B^a и $A^b \pmod p$ следующим образом:

$$B^a \equiv (g^b)^a \pmod p;$$
$$A^b \equiv (g^a)^b \pmod p.$$

Таким образом, применив свойство возведения в степень по модулю, мы имеем следующее:

$$g^{(b \cdot a)} \equiv g^{(a \cdot b)} \pmod p.$$

Отсюда:

- Алиса — $(7^6)^3 \equiv 7^{(6 \cdot 3)} \equiv 9 \pmod{11}$;
- Боб — $(7^3)^6 \equiv 7^{(3 \cdot 6)} \equiv 9 \pmod{11}$

Этот математический трюк и обеспечивает работоспособность алгоритма Диффи — Хеллмана.

Теперь, когда мы разобрались с алгоритмом, отметим его недостатки и возможные атаки на него.

Возможные атаки на алгоритм Диффи — Хеллмана и криптоанализ

Наиболее распространенной атакой на алгоритм Диффи — Хеллмана является атака посредника (man-in-the-middle, MitM).

Когда злоумышленник осуществляет атаку посредника, он проникает в канал связи между отправителем и получателем, где следит за обменом данными, а также блокирует или изменяет его. Как правило, он притворяется одним из настоящих участников диалога. В нашем примере Ева выдает себя за Алису (рис. 3.2).

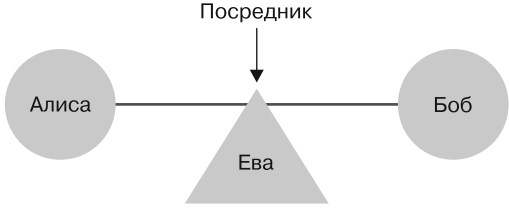


Рис. 3.2. Ева — посредник

На шаге 2 алгоритма Диффи — Хеллмана Алиса и Боб обменялись своими открытыми параметрами (A) и (B).

В процесс обмена данными, когда Алиса отправляет (A) Бобу, а Боб отправляет (B) Алисе, вмешивается Ева, выдавая себя за Алису.

Атака посредника выглядит следующим образом.

- На шаге 2 Алиса отправляет (A) Бобу, а Боб отправляет (B) Алисе.
- В этот момент происходит атака посредника — Ева перехватывает (A) и вместо него отправляет Бобу (E). (Здесь (E) представляет собой не результат шифрования, а открытый параметр Евы, сгенерированный с использованием ее закрытого ключа.) Затем Боб отправляет (B) Еве, полагая, что это Алиса.

Проанализируем функцию Алисы $A \equiv g^a \pmod{p}$ и функцию Боба $B \equiv g^b \pmod{p}$.

Эти функции играют очень важную роль, если выполняются в обычной арифметике, а не в арифметике конечных полей. Например, существует еще одна возможная атака, которая называется атакой дней рождения. Это одна из самых известных атак на дискретные логарифмы. Считается, что в группе людей как минимум у двоих будет один день рождения. Таким образом, в циклической группе можно будет найти несколько одинаковых значений (коллизий), что упрощает решение задачи дискретного логарифма.

В заключительной части алгоритма можно увидеть результат атаки посредника.

Предположим, что Алиса и Боб используют алгоритм Диффи — Хеллмана для генерации общего закрытого ключа и шифрования следующего сообщения: *«Боб, пожалуйста, переведи 10 000 долларов на мой счет № 1234567»*.

После того как Боб и Ева (выдающая себя за Алису) обменялись своими открытыми параметрами, Ева отправляет Бобу измененное сообщение (перехваченное у Алисы), которое было зашифровано общим ключом Евы и Боба.

Предположим, что оно имеет вид *bu3fb3440r3nrunfjr3umi4gj57*je*.

Боб получает это зашифрованное сообщение (предположительно, от Алисы) и расшифровывает его с помощью общего ключа Диффи — Хеллмана.

В результате измененное сообщение от Евы выглядит так: *«Боб, пожалуйста, переведи 10 000 долларов на мой счет № 3217654»*.

Ева указала свой счет, что делает такую атаку потенциально разрушительной.

Как мы уже говорили, шаг 2 не определяет успешность атаки посредника, поскольку сообщения (*A*) и (*B*) передаются в открытом виде. Важен следующий вопрос: как Боб может быть уверен, что сообщение (*A*) действительно исходит от Алисы?

Алгоритм Диффи — Хеллмана не дает Бобу полной гарантии того, что сообщение (*A*) отправлено Алисой, а не Евой (злоумышленником). Аналогично Алиса не может с уверенностью утверждать, что сообщение (*B*) пришло от Боба. Полагаясь только на полученные параметры и не имея дополнительной информации об идентичности обеих сторон, алгоритм Диффи — Хеллмана подвержен риску атаки с подменой идентичности.

Этот пример показывает, что отправителю (Алисе) и получателю (Бобу) необходим способ убедиться в том, что их собеседники являются теми, за кого себя выдают, и что их открытые ключи (*A*) и (*B*) действительно исходят от них. Одной из наиболее широко применяемых техник предотвращения атак посредника и идентификации пользователей канала связи является цифровая подпись. Мы рассмотрим цифровые подписи в главе 4, которая посвящена объяснению этих криптографических методов.

Вместе с тем алгоритмы с открытым и закрытым ключами могут идентифицировать стороны связи и преодолевать атаку посредника. В части IV этой книги я покажу вам несколько алгоритмов нового поколения с открытым и закрытым ключами, которые, хотя и неасимметричны, имеют несколько способов подписания.

Кроме прочего, существует версия алгоритма Диффи — Хеллмана, реализуемая с помощью эллиптических кривых. Мы рассмотрим ее в главе 7.

RSA

Среди криптографических алгоритмов RSA сияет как звезда. Красота этого алгоритма сопоставима с его логической простотой, а внутри него скрыта такая сила, что даже спустя 40 лет после появления он по-прежнему используется для защиты более 80 % коммерческих транзакций в мире.

RSA — это аббревиатура, составленная из фамилий трех изобретателей алгоритма: *Ривеста* (Rivest), *Шамира* (Shamir) и *Адлемана* (Adleman). RSA — это то, что мы называем идеальным асимметричным алгоритмом. В 1997 году английское криптографическое агентство CESG приписало изобретение шифрования с открытым ключом Джеймсу Аллису, который разработал его в 1970 году. Кроме того, CESG заявило, что в 1973 году Клиффорд Кокс написал документ, демонстрирующий аналогичную версию алгоритма RSA.

Основная концепция асимметричного алгоритма заключается в том, что для шифрования и расшифрования используются разные ключи.

Из аналогии с замками из раздела «Алгоритм Диффи — Хеллмана» понятно, что запереть ящик может любой, не только Алиса и Боб. В этом и заключается настоящая проблема атаки посредника: невозможно распознать, кому именно принадлежит замок.

Решение этой проблемы можно найти с помощью еще одного интересного мысленного эксперимента. Мы будем использовать похожую аналогию, но немного изменим детали.

Предположим, Алиса сделала множество копий своего замка и разослала их по всем почтовым отделениям страны, при этом ключ, открывающий все замки, сохранила в секретном месте.

Если Боб хочет отправить Алисе секретное сообщение, он может пойти на почту, положить сообщение в ящик и запереть его с помощью замка Алисы.

Таким образом, закрыв ящик, Боб (отправитель) не сможет его открыть. При этом, когда Алиса получит ящик, она сможет открыть его своим уникальным закрытым ключом. Мы вернемся к этой концепции в следующей главе.

В отличие от алгоритма Диффи — Хеллмана в RSA Боб шифрует сообщение с помощью открытого ключа Алисы. После шифрования даже Боб не способен расшифровать сообщение, а Алиса может сделать это с помощью своего закрытого ключа.

Именно это обеспечило концепции асимметричного шифрования практическое применение. RSA реализует шифрование с использованием открытого ключа получателя и его расшифровку с помощью закрытого ключа получателя. Для

этого необходима определенная математическая функция, о которой я расскажу позже, когда мы будем подробно изучать алгоритм.

Как мы упоминали ранее, изобретателей этого алгоритма было трое. Все они в то время работали в Массачусетском технологическом институте Бостона. Мы говорим о конце 1970-х годов. После изобретения алгоритма Диффи — Хеллмана Рональд Ривест очень увлекся этим новым видом криптографии. Сначала к нему присоединился математик Леонард Адлеман, а затем еще один коллега из отдела компьютерных наук, Ади Шамир. Ривест пытался найти математический способ отправить секретное сообщение, зашифрованное с помощью открытого ключа получателя, и расшифровать его с помощью закрытого ключа получателя. В алгоритме Диффи — Хеллмана сообщение можно зашифровать только после обмена ключами, используя один и тот же общий ключ. Задача заключалась в том, чтобы найти способ отправить сообщение, которое было зашифровано с помощью открытого ключа и может быть расшифровано с помощью закрытого ключа. Однако, как я уже говорил, для этого нужна специфическая обратная математическая функция. В этом и заключается настоящая ценность изобретения RSA, которая вскоре станет вам понятной.

История этого открытия, рассказанная самим Ривестом, довольно забавна. Это был апрель 1977 года. Ривест и Адлеман встретились в доме одного из студентов на Пасху. Выпив очень много вина, Ривест вернулся домой только к полуночи. Он начал обдумывать проблему, которая мучила его почти год. Лежа в кровати, Ривест открыл книгу по математике и наткнулся на функцию, которая могла бы идеально подойти для решения поставленной задачи.

Это была особая обратная функция модульной арифметики, связанная с задачей факторизации.

Как я рассказывал в главе 1, задача факторизации большого числа, полученного перемножением двух больших простых чисел, считается очень трудноразрешимой даже для компьютера с огромной вычислительной мощностью.

Объяснение алгоритма RSA

Чтобы лучше понять алгоритм, рассмотрим, как происходит обмен секретным сообщением между Алисой и Бобом. Предположим, Боб хочет отправить Алисе секретное сообщение. Пусть:

- M — секретное сообщение;
- e — открытый параметр (обычно это постоянное число);
- c — шифротекст или криптограмма;
- p, q — два больших случайных простых числа (закрытые ключи Алисы).

Далее описывается этап создания открытого и закрытого ключей. Как вы увидите, суть RSA (ее магическая функция) заключается в генерировании закрытого ключа Алисы [d].

Генерирование ключей

Открытый ключ Алисы (N) задается следующим образом:

$$N = p \cdot q.$$

Как уже говорилось, произведение двух больших простых чисел (N) очень трудно разложить на множители и в целом злоумышленнику очень сложно найти [p] и [q]. Закрытый ключ Алисы [d] задается следующим образом[1]:

$$[\boldsymbol{d}] \cdot e \equiv 1 \pmod{[\boldsymbol{p} - \mathbf{1}] \cdot [\boldsymbol{q} - \mathbf{1}]}.$$

Выделенные полужирным элементы защищены и держатся в секрете.

Ключ генерируется в два этапа.

Боб выполняет шифрование:

$$c \equiv \boldsymbol{M}^e \pmod{N}.$$

Затем он отправляет шифротекст (c) Алисе. Та может расшифровать (c) с помощью своего закрытого ключа [d]:

$$C^d \equiv \boldsymbol{M} \pmod{N}.$$

Вот и все!

Числовой пример. Давайте возьмем следующие числа:

- $M = 88$;
- $e = 9007$;
- $p = 101$;
- $q = 67$;
- $N = 6767$.

---

[1] Иногда вместо произведения $(p - 1)$ и $(q - 1)$ можно встретить запись $\phi(p \cdot q)$. Функция Эйлера $\phi(n)$ — функция, которая показывает количество натуральных чисел, меньше или равных n и взаимно простых с ним. Поскольку p и q — простые числа, то произведение, записанное в этой формуле, эквивалентно результату функции Эйлера для числа $n = p \cdot q$.

Теперь можем вернуться к двум шагам генерирования ключа.

Боб выполняет шифрование:

$$88^{9007} \equiv 6621 \pmod{6767}.$$

Алиса получает криптограмму, то есть $c = 6621$.

Она выполняет расшифровку:

$$9007 \cdot d \equiv 1 \pmod{(101 - 1) \cdot (67 - 1)};$$
$$d = 3943;$$
$$6621^{3943} \equiv 88 \pmod{6767}.$$

Как видите, Алиса возвращает секретное сообщение $[M] = 88$ с помощью своего закрытого ключа $[d] = 3943$.

Анализ RSA

У RSA есть несколько ключевых элементов, которые необходимо рассмотреть. Однако самое главное — понять, как работает функция, используемая для расшифровки шифротекста (c) и получения исходного текста $[M]$:

$$\boldsymbol{M} \equiv c^{[d]} \pmod{N}.$$

Это шаг 2 из предыдущего раздела, то есть функция дешифрования. Я просто изменил порядок записи, поместив $[M]$ слева.

Причина, по которой эта функция работает, заложена в уравнении, которое используется для генерации ключа:

$$[d] \cdot e \equiv 1 \pmod{[p - 1] \cdot [q - 1]}.$$

Здесь $[d]$ — закрытый ключ Алисы. Согласно теореме Эйлера, скорее всего, это уравнение будет выполняться, поскольку числа $[p]$ и $[q]$ очень велики, а $[M]$, вероятно, является взаимно простым с $[N]$. Если это уравнение выполняется, то мы можем переписать этап шифрования следующим образом:

$$(M^e)^d \pmod{N}.$$

Согласно свойству степеней и теореме Эйлера, имеем следующее:

$$M^{(e \cdot d)} \pmod{N};$$
$$de \equiv 1 \pmod{(p - 1) \cdot (q - 1)}.$$

Это выражение можно также записать как $M^1 = M \pmod{N}$.

Таким образом, применив $[d]$ на этапе расшифрования, Алиса может получить $[M]$.

Типичные атаки на алгоритм

Все атаки, которые будут описаны в этом разделе, хорошо изучены. Именно поэтому мы относим их к типичным атакам на RSA.

Первые три метода атаки связаны с открытым параметром (mod N). Чтобы выполнить атаку на $N = p \cdot q$, злоумышленник может сделать следующее:

- использовать *эффективный* алгоритм факторизации для определения p и q;
- задействовать новые алгоритмы, которые при определенных условиях могут найти числа;
- использовать квантовый компьютер для факторизации N. В главе 9 вы увидите алгоритм, способный решить задачу факторизации RSA. В недалеком будущем квантовые вычисления будут становиться все более мощными и, несомненно, станут главным соперником классической криптографии.

Рассмотрим эти три сценария.

1. Эффективный алгоритм факторизации, упомянутый в первом сценарии, еще неизвестен. Наиболее широко распространены следующие методы атаки:
   - общий метод решета числового поля;
   - метод квадратичного решета;
   - алгоритм Полларда.
2. Во втором сценарии, если (n) — это количество цифр числа $N = p \cdot q$ и атакующему известны первые ($n/4$) или последние ($n/4$) цифры числа [p] или [q], то он может успешно выполнить факторизацию (N). Существует очень небольшая вероятность того, что эти данные будут известны злоумышленнику. Например, если [p] и [q] состоят из 100 цифр, при этом известны первые или последние 25 цифр числа [p], то факторизация N становится возможной.

   Факторизация Копперсмита еще лучше раскрывает этот метод атаки. Атаки Копперсмита работают в сценариях, когда экспоненты (e) или [d] и даже открытый текст [M] слишком коротки. Мы вернемся к этому совсем скоро.
3. Если злоумышленник использует квантовый компьютер, то теоретически он может за короткое время факторизовать N с помощью алгоритма Шора. Я убежден, что в будущем появятся другие, более эффективные квантовые алгоритмы. Более подробно объясню эту теорию в главе 9, где мы поговорим о квантовых вычислениях и квантовой криптографии.

Очень короткий открытый текст [M] и небольшая экспонента (e) повышают вероятность успешного взлома RSA. Это связано с тем, что операция возведения в степень M^e остается внутри модуля N. Таким образом, на этом этапе шифрования мы имеем следующее:

$$M^e < N.$$

Здесь достаточно e-го корня из (c), чтобы найти]M[.

Я использовал открытые квадратные скобки]*M*[для того, чтобы показать, что сообщение было расшифровано.

Числовой пример. Возьмем следующие числа:

- $M = 2$;
- $e = 3$;
- $N = 77$;
- $2^3 \equiv 8 \pmod{77}$.

Поскольку $e = 3$, то, найдя простой кубический корень $\sqrt[3]{\ }$, мы можем получить сообщение в открытом виде:

$$8^{(1/3)} = 2.$$

Здесь мы работаем в линейной, а не модульной арифметике.

Решить эту проблему можно, удлинив сообщение добавлением к нему случайных битов. Этот метод очень широко распространен в области криптографии и обеспечения кибербезопасности и известен как *дополнение*, или *паддинг* (padding).

Существуют различные способы дополнения, однако мы говорим именно о битовом дополнении. Как говорилось в главе 1, мы можем использовать код ASCII для преобразования текста в двоичную систему счисления. Таким образом, сообщение [*M*] можно представить строкой битов. Если мы добавим случайные биты (обычно в конец строки, хотя можно и в начало), то получим что-то вроде:

... | 1011 1001 1101 0100 0010 0111 **0000 0000** |.

Выделенные жирным цифры представляют собой дополнение.

Этот метод можно использовать для заполнения сообщения, длина которого составляет любое количество битов, не обязательно целое число байтов. Например, к сообщению длиной 23 бита можно добавить 9 бит для заполнения 32-битного блока.

Теперь, когда вы лучше знакомы с RSA и свойствами модульной арифметики, рассмотрим первое интересное применение этого алгоритма.

Применение RSA для проверки международных соглашений

Допустим, страна Альфа заинтересована в мониторинге сейсмических данных страны Бета, чтобы удостовериться, что там не ведутся эксперименты с ядерными бомбами. На территории страны Бета установлен набор датчиков для мониторинга сейсмической активности, которые записывают и шифруют

полученные данные. Выходные данные передаются в страну Альфа, скажем, через спутник.

Это интересное применение RSA работает следующим образом.

- Страна Альфа (А) хочет быть уверена в том, что страна Бета (Б) никак не меняет данные.
- Страна Бета хочет проверить сообщение перед отправкой (в целях шпионажа).

Назовем собранные с датчиков данные $[x]$. Протокол будет работать следующим образом.

1. Страна Альфа выбирает параметры ($N = p \cdot q$) как произведение двух больших простых чисел и параметр (e).
2. Страна Альфа отправляет (N, e) стране Бета.
3. Страна Альфа хранит закрытый ключ $[d]$ в секрете.

Протокол проверки угроз атомных экспериментов разработан следующим образом.

1. Расположенный глубоко в земле датчик собирает данные $[x]$, выполняя шифрование с использованием закрытого ключа $[d]$:
$$x^d \equiv y \pmod{N}.$$
2. Сначала датчик отправляет параметры (x) и (y) стране Бета, чтобы та проверила достоверность информации. Бета проводит следующую проверку:
$$y^e \equiv x \pmod{N}.$$
3. После получения положительного результата проверки страна Бета пересылает (x, y) стране Альфа, которая может управлять результатом (x):
$$x \equiv y^e \pmod{N}.$$

(x) — это набор данных, собранных с датчика, а $[d]$ — закрытый ключ страны Альфа, хранящийся внутри защищенного программного датчика, который собирает данные.

Это шифрование выполняется в порядке, обратном порядку алгоритма RSA.

Если уравнение $y^e \equiv x \pmod{N}$ проходит проверку, то страна Альфа может быть уверена, что полученные данные достоверны, а страна Бета не изменила сообщение и не вмешивалась в работу датчика. Это связано с тем, что зашифрованное сообщение (x), соответствующее криптограмме (y), может быть сгенерировано только теми, кто знает закрытый ключ $[d]$.

Если страна Бета пыталась манипулировать шифром внутри ящика, в котором находится датчик, меняя значение (x), то ей будет очень сложно получить значимое сообщение.

Как мы уже говорили, в этом протоколе алгоритм RSA инвертирован, то есть шифрование выполняется с помощью закрытого ключа [d], а не открытого параметра (e).

По сути, сложность модификации шифрования для страны Бета заключается в том, чтобы получить значимое число или сообщение. Изменить криптограмму (y) даже при известном параметре (x) так же сложно, как и найти дискретный логарифм, что, как мы уже видели, является трудноразрешимой задачей.

Мы можем наглядно представить этот процесс с помощью следующей схемы (рис. 3.3).

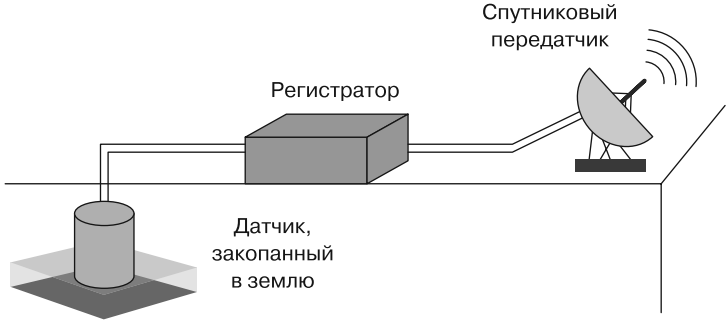


Рис. 3.3. Система записи сейсмических данных

Теперь, когда вы узнали, как используется алгоритм RSA в международных соглашениях, я хочу представить такую тему, как нетипичные атаки на алгоритм RSA. Более подробно она будет раскрыта в главе 6.

Нетипичные атаки

Я назвал эти алгоритмы *нетипичными*, потому что они были реализованы мной, но до сих пор не подтверждены профессиональным сообществом.

По мере чтения вы увидите, что такие *нетипичные атаки* на RSA применимы и к другим асимметричным алгоритмам шифрования. Они были реализованы в период с 2011 по 2014 год и позволили восстановить секретное сообщение [M] без ключа расшифрования Алисы [d] и простых секретных чисел [p] и [q], составляющих (N). Я покажу эти атаки в текущем разделе, но более подробно о них расскажу в главе 6.

Среди этих алгоритмов есть несколько атак на RSA, но их можно применить на большинстве асимметричных алгоритмов, рассматриваемых в этой главе.

Новый алгоритм, который в будущем сможет решить задачу факторизации, называется NextPrime. Он основан на *генетическом алгоритме*, открытом моим близким другом Герардо Иоване. Он объяснил мне этот механизм в 2009 году.

В своей статье *The Set of Prime Numbers* («Набор простых чисел») Герардо описал, как можно получить все простые числа с помощью простого алгоритма, основанного на исключении непростых чисел из определенного шаблона.

Чтобы лучше понять генетический алгоритм Герардо Иоване, можно прочитать его статью The Set of Prime Numbers на https://arxiv.org/abs/0709.1539.

Спустя многие годы работы, пережив множество приступов головной боли, я пришел к математической функции, которая представляет собой кривую, каждая позиция на которой — простое число. Между этими позициями лежат полупростые числа (N), порожденные двумя простыми числами. Эта кривая геометрически представляет все простые числа во Вселенной. Оказывается, что (N) на кривой всегда равноудалено от двух простых чисел [p] и [q]. Также можно доказать, что простые числа имеют четкий порядок, а не располагаются случайным образом, как считалось ранее.

Расстояние между двумя простыми числами, [p] и [q], в произведении, определяющем (N), равно количеству простых чисел, лежащих между [p] и [q]. Например, *расстояние между простыми числами* 17 и 19 равно 0, между числами 1 и 100 — 25, а между 10 000 и 10 500 — всего 55.

На данный момент этот алгоритм эффективен только при определенных условиях, например, когда [p] и [q] находятся *довольно близко* друг к другу (на полиномиальном расстоянии). Однако не имеет значения, насколько велики оба простых числа. Я провел несколько тестов с этим алгоритмом, используя простые числа, состоящие из 101 000 цифр.

Чтобы вы поняли, насколько велики такие числа: 10^{80} — это количество частиц обычной материи во Вселенной. Для сравнения: полупростые числа из $10^{101\,000}$ цифр соответствуют длине открытого ключа RSA около 3000 бит, что становится стандартом наряду с открытыми ключами длиной 4000 бит. Если два простых числа близки друг к другу, то алгоритм NextPrime сможет вычислить ключ за несколько секунд.

Во время написания этой книги я работаю над версией алгоритма NextPrime, основанной на работе квантового компьютера. Он может стать следующим поколением алгоритмов факторизации для квантовых вычислений, подобно алгоритму Шора, о котором речь пойдет в главе 9.

Рассмотрим другие атаки на RSA. Как показано на следующей схеме (рис. 3.4), алгоритм имеет две точки атаки: одна — задача факторизации (N), другая — дискретная степень [M^e].

Как мы видели, бóльшая часть типичного криптоанализа сосредоточена на факторизации (N). Однако, кроме уязвимости со стороны факторизации, RSA имеет проблему, связанную с экспонентой (e). Эти уязвимости, по сути, явля-

ются бэкдорами, которые позволяют восстановить сообщение [M] без секретных параметров отправителя [d], [p] и [q]. В главе 6 вы увидите, что существуют методы атак, эквивалентные созданию бэкдора внутри алгоритма RSA и его основной библиотеки OpenSSL.

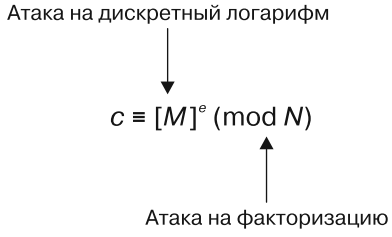


Рис. 3.4. Возможные точки атаки на RSA

В рамках стандартной парадигмы RSA Боб (отправитель) не может вернуть сообщение после того, как оно было доставлено. *Однако нетипичные методы взлома RSA способны оспорить это утверждение*. Боб может выполнить фальшивое шифрование и вернуть сообщение [M], зашифрованное открытым ключом Алисы, а фальшивая криптограмма может быть расшифрована Алисой на этапе расшифрования RSA.

Является ли эта атака сомнительной моделью, не отражающей реальные ситуации, или же имеет практическое применение?

Ответ на этот вопрос я дам в главе 6. А сейчас рассмотрим другой протокол, который стал популярным программным обеспечением, — PGP, основанный на реализации RSA.

PGP

Pretty Good Privacy (PGP) — это, вероятно, самая широко используемая криптографическая программа в мире.

PGP была создана Филиппом Циммерманом во время холодной войны. Филипп планировал перевезти свою семью в Новую Зеландию, поскольку считал, что в случае ядерной атаки эта страна, столь изолированная от остального мира, меньше пострадает от атомных разрушений. Однако в какой-то момент что-то заставило его передумать и остаться в США.

Активисту антиядерного движения Циммерману необходимо было поддерживать связь с единомышленниками. Для защиты сообщений и файлов, передаваемых через Интернет, и была разработана PGP — бесплатная программа с открытым исходным кодом, предназначенная для некоммерческого использования.

В то время криптосистемы размером более 40 бит рассматривались как боеприпасы. Даже сегодня криптография все еще считается оружием. Если вы решите запатентовать новую криптосистему, вам необходимо будет получить специальную лицензию. Требуется разрешение Министерства обороны на публикацию или распространение этой криптосистемы. С этой проблемой столкнулось PGP, в котором никогда не использовались ключи размером меньше 128 бит. Штрафы и уголовное преследование за нарушение этого закона очень суровы, поэтому Циммерман много лет находился в неопределенном правовом статусе, пока американское правительство не решило завершить расследование и закрыть дело.

PGP — это даже не алгоритм, а протокол. Его новизна заключается в объединении асимметричного и симметричного видов шифрования. Протокол использует алгоритм асимметричного шифрования для обмена ключами и алгоритм симметричного шифрования для получения шифротекста. Кроме того, для идентификации пользователя и предотвращения атак посредника он требует цифровую подпись.

Далее перечислены этапы выполнения протокола.

1. Ключ передается с помощью асимметричного алгоритма шифрования (схемы Эль-Гамаля, RSA).
2. Ключ, переданный с помощью асимметричного шифрования, становится сеансовым ключом для симметричного шифрования (уязвимый к взлому DES, Triple DES, IDEA и AES рассматривались в главе 2).
3. Для идентификации пользователей применяется цифровая подпись (мы рассмотрим цифровые подписи RSA в главе 4).
4. С помощью симметричного ключа выполняется расшифровка.

PGP — хороший протокол, обеспечивающий очень высокую степень конфиденциальности и защиту передачи коммерческих секретов.

Схема Эль-Гамаля

Эта схема является асимметричной версией алгоритма Диффи — Хеллмана. Схема Эль-Гамаля направлена на преодоление проблем, связанных с атаками посредника и невозможностью создания цифровой подписи для подтверждения владения ключом в алгоритме Диффи — Хеллмана. Более того, схема Эль-Гамаля, как и RSA, — это подлинный асимметричный алгоритм, поскольку шифрует сообщение без предварительного обмена ключом.

Сложность схемы Эль-Гамаля, как правило, связана с решением задачи дискретного логарифма. Как мы увидим далее, в алгоритме есть задача факторизации.

В схеме Эль-Гамаля присутствует новый элемент, о котором мы еще не говорили, — целое случайное число [k], которое выбирает и держит в секрете отправитель[1]. Это важный элемент, поскольку он придает шифрованию эфемерный характер в том смысле, что [k] делает функцию шифрования непредсказуемой. Более того, мы часто будем видеть этот новый элемент в главе 5, посвященной протоколам доказательства с нулевым разглашением.

Рассмотрим реализацию этого алгоритма и его применение для передачи секретного сообщения [M].

Алиса и Боб — два участника передачи данных. Алиса — отправитель, а Боб — получатель.

На рис. 3.5 показан процесс работы схемы Эль-Гамаля.

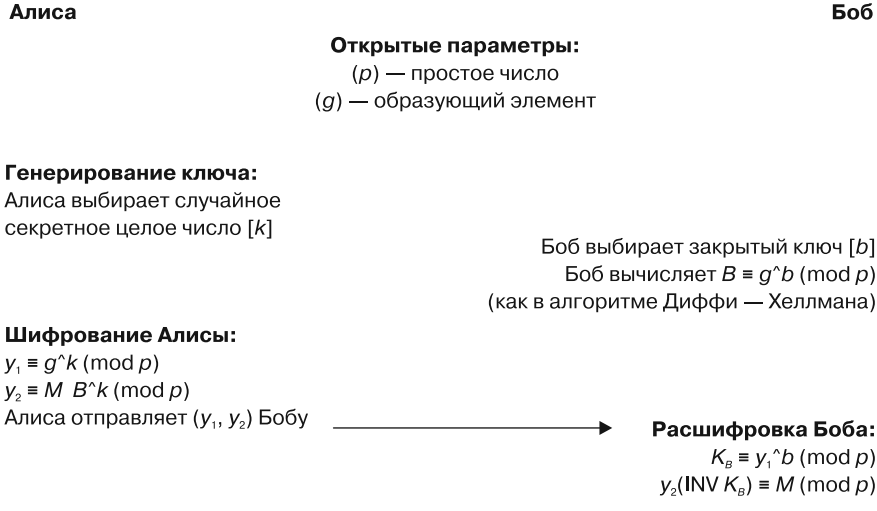


Рис. 3.5. Шифрование/дешифрование в схеме Эль-Гамаля

На последнем шаге расшифровки Боба мы видим умножение на обратное значение (INV) по (mod p). Такая операция, по сути, является делением, выполняемым в конечном поле. Так, если обратным числом A является B, то $A \cdot B = 1 \pmod{p}$. В следующем примере показана реализация этой инвертированной модульной функции в системе Mathematica.

---

[1] В шифровании по схеме Эль-Гамаля у отправителя нет собственного долговременного приватного ключа: он использует случайное секретное число k только для одноразового сеанса шифрования. Далее в книге автор рассматривает применение этой схемы для создания цифровой подписи, где у отправителя уже появляется постоянный приватный ключ (обычно обозначаемый a), а величина k также выступает как одноразовый секрет, но выполняет иную роль.

Теперь, чтобы лучше понять алгоритм, рассмотрим числовой пример.

Открыто определенные параметры. Открытыми параметрами являются p (большое простое число) и g (образующий элемент):

- $p = 200\,003$;
- $g = 7$.

Генерирование ключа. Алиса выбирает случайное число [k], которое будет храниться в секрете, $k = 23$.

Боб вычисляет свой открытый ключ (B) на основе своего закрытого ключа [b], $b = 2367$ (закрытый ключ Боба):

$$B \equiv 7^{2367} \pmod{200\,003} \equiv 151\,854.$$

Шифрование Алисы. Алиса создает секретное сообщение [M]:

$$M = 88.$$

Затем она вычисляет два открытых параметра (y_1) и (y_2), которые будут отправлены Бобу:

$$y_1 \equiv 7^{23} \pmod{200\,003} = 90\,914;$$
$$y_2 \equiv 88 \cdot 151\,854^{23} \pmod{200\,003} = 161\,212.$$

Алиса отправляет Бобу $y_1 = 90\,914$ и $y_2 = 161\,212$.

Расшифровка Боба. Сначала Боб вычисляет значение (K_b), возводя параметр $y_1 = 90\,914$ в степень своего закрытого ключа $b = 2367$:

$$K_b \equiv 90\,914^{2367} \pmod{200004} = 10\,923.$$

Затем происходит вычисление обратного значения (K_b) по (mod p) (это операция упрощения Reduce [$K_b \cdot x = = 1, x$, Modulus $\rightarrow p$] в системе компьютерной алгебры Wolfram Mathematica):

$$\text{INV } K_b = 192\,331 \pmod{200\,003}.$$

Наконец, Боб может расшифровать сообщение [M].

С помощью умножения (y_2) на [INV K_b] он получает сообщение [M]:

$$y_2 \cdot \text{INV } K_b \equiv M \pmod{200\,003}.$$

Конечным результатом всего процесса является сообщение [M]:

$$161\,212 \cdot 192\,331 \equiv 88 \pmod{200\,003}.$$

Шифрование Эль-Гамаля используется в бесплатном программном обеспечении *GNU Privacy Guard* (GnuPG). За годы своего существования эта программа приобрела широкую популярность и стала де-факто стандартом бесплатного программного обеспечения для приватной переписки и создания цифровых под-

писей. Для обмена криптографическими ключами GnuPG использует последнюю версию PGP. Больше об этой программе можно узнать на ее веб-странице: https://gnupg.org/software/index.html.

Как я говорил ранее, в основе схемы Эль-Гамаля лежит задача дискретного логарифма. Это обусловлено тем, что открытые параметры и ключи определяются уравнениями дискретного логарифма.

Например, $B \equiv g^b \pmod{p}$, $y_1 \equiv g^k \pmod{p}$ и $K_b \equiv y_1^b \pmod{p}$ — функции дискретного логарифма.

Хотя задача дискретного логарифма считается основой схемы Эль-Гамаля, важную роль в ней играет задача факторизации, которую я сейчас продемонстрирую. Вернемся к функции шифрования (y_2):

$$y_2 \equiv M \cdot B^k \pmod{p}.$$

Поскольку функция представляет собой операцию умножения, злоумышленник может попытаться восстановить сообщение посредством факторизации (y_2).

Упростив функцию до вида:

$$H \equiv B^k \pmod{p},$$

мы получим уравнение:

$$y_2 \equiv M \cdot H \pmod{p}.$$

Его можно переписать следующим образом:

$$M \equiv \frac{y_2}{H} \pmod{p}.$$

Как видите, (y_2) является произведением $[M \cdot H]$. Если кто-то сможет определить множители (y_2), он, вероятно, сможет найти $[M]$.

Резюме

В этой главе мы рассмотрели основные темы, связанные с асимметричным шифрованием. В частности, вы изучили принцип работы дискретного логарифма, а также познакомились с наиболее известными алгоритмами асимметричного шифрования: алгоритмами Диффи — Хеллмана, RSA и схемой Эль-Гамаля. Кроме того, мы рассмотрели интересный пример применения RSA для обмена конфиденциальными данными между двумя странами. Теперь, когда вы познакомились с основами асимметричного шифрования, настало время перейти к цифровым подписям. В главе 4 поговорим о том, как применять цифровые подписи в изученных алгоритмах. Как вы уже могли убедиться на примере PGP, все эти темы тесно взаимосвязаны.

4 Хеш-функции и цифровые подписи

С незапамятных времен большинство договоров, то есть любых соглашений между людьми, составлялись на бумаге и подписывались вручную. Подпись в конце документа подтверждала подлинность личности подписавшего. Это было возможно, поскольку подписывающие физически находились в одном месте в момент подписания. Как правило, они могли доверять друг другу, так как их личности подтверждало третье, доверенное лицо (нотариус или юридическое лицо), выступающее в качестве *незаинтересованной стороны*.

В наше время люди, желающие подписать договор, часто даже не знакомы друг с другом. Они обмениваются документами для подписания по электронной почте. При этом нет надежной третьей стороны, подтверждающей их личность.

Представьте, что вы подписываете договор и отправляете его по Интернету другой стороне. Эта сторона *не заслуживает доверия*, поэтому вы не хотите раскрывать содержимое документа неизвестному лицу в таком незащищенном канале связи, как Интернет. Как в этом случае можно проверить корректность подписи?

Кроме того, как можно скрыть содержимое документа и в то же время разрешить ставить на нем подпись?

В такой ситуации на помощь приходят цифровые подписи. В этой главе будет показана польза хеш-функции для создания цифровой подписи зашифрованных документов. Цифровые подписи позволяют любому человеку идентифицировать подписывающих и в то же время гарантируют, что документ не попадет в поле зрения посторонних.

Итак, начнем со знакомства с хеш-функциями и их основными областями применения. Затем подробно рассмотрим алгоритмы, выполняющие цифровые подписи.

Общее объяснение хеш-функций

В криптографии хеш-функции используются для множества целей. Как вы уже видели в главе 3, одна из них — *сокрытие* содержимого сообщения, на котором ставится цифровая подпись. Что это значит? Обычно, когда происходит обмен сообщением посредством любого асимметричного алгоритма, для идентификации отправителя необходима подпись. В общем случае можно сказать, что подпись выполняется над сообщением [*M*]. Однако по ряду причин, о которых будет рассказано позже, не рекомендуется подписывать секретное сообщение напрямую. Вместо этого отправитель должен сначала преобразовать сообщение [*M*] в функцию (*M'*), которая будет видна каждому. Эта функция называется *хешем сообщения M*, и в текущей главе она будет обозначаться $f(H)$ или $h[M]$.

Далее мы подробнее рассмотрим взаимосвязь хешей и цифровых подписей. Стоит отметить, что я включил хеш-функции в главу о цифровых подписях главным образом потому, что они играют основную роль в процессе подписания сообщения. Однако хеш-функции активно используются и вне области цифровых подписей. Например, они применяются в *распределенных хеш-таблицах* в технологии блокчейн и поисковых системах.

Итак, первый вопрос, который нас интересует: что такое хеш-функция?

Ответ можно найти в значении слова «хеш». Одно из определений хеша — *разбиение на части*, что в данном случае означает разделение содержимого сообщения или любой другой информации на более мелкие сегменты.

Приняв сообщение произвольной длины [*M*] в качестве входных данных и выполнив над ним хеш-функцию $f(H)$, мы получим выходные данные (дайджест сообщения) определенного фиксированного размера (*M'*).

Хеши концептуально близки к *функции расширения битов*, о которой говорилось в главе 2. Как вы помните, эта функция увеличивает количество битов входных данных. Хеш-функции выполняют противоположную задачу — битовое значение хеша становится меньше входных данных.

Хеш-функция, или просто хеш, является *однонаправленной функцией*. Позже мы увидим, что это свойство играет очень важную роль в классификации хеш-функций.

Однонаправленность, или односторонность, функции означает, что вычислить результат в одном направлении легко, но вернуться к исходному сообщению, используя выход функции, очень трудно, если не невозможно.

Посмотрим, какими свойствами должна обладать хеш-функция.

- При наличии входного сообщения $[M]$ его дайджест[1] $h(M)$ может быть *очень быстро* вычислен.
- Должно быть практически *невозможно* (–x%) вернуться к исходному сообщению $[M]$, используя выходное сообщение (M'), вычисленное через $h(M)$.
- Поиск двух различных входных сообщений $[m_1]$ и $[m_2]$ должен быть *неразрешимой* задачей с вычислительной точки зрения при условии

$$h(m_1) = h(m_2).$$

В этом случае можно сказать, что функция $f(h)$ устойчива к коллизиям.

В качестве примера хеш-функции можно привести дайджест всего содержимого Википедии, который представляет собой бит фиксированной длины (рис. 4.1).

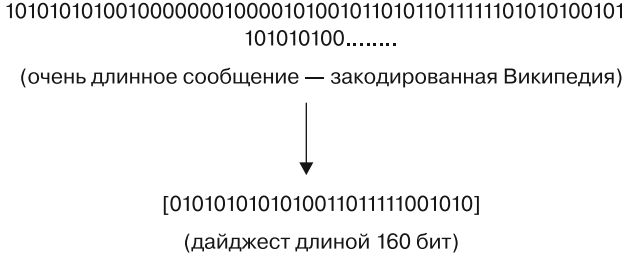


Рис. 4.1. Пример дайджеста

Другой отличной метафорой хеш-функций является воронка-мясорубка, которая перемалывает входящий открытый текст и выдает дайджест фиксированной длины (рис. 4.2).

Следующий вопрос: зачем и где используются хеш-функции?

Как говорилось в начале этой главы, хеш-функции широко применяются в самых разных областях криптографии.

- Хеш-функции обычно используются в асимметричном шифровании, чтобы избежать раскрытия исходного сообщения $[M]$ при создании цифровых подписей (мы увидим это позже в данной главе). Хеш исходного сообщения подтверждает личность передающего, используя (M') в качестве замены.
- Хеши задействуются для проверки целостности сообщения. На основании цифрового хеша (M') получатель может легко обнаружить любые изменения исходного сообщения $[M]$. Даже если кто-то изменит всего 1 бит содержи-

[1] Дайджест — результат работы хеш-функции.

мого Википедии [M], его хеш-функция $h(M)$ приобретет совершенно другое хеш-значение $h(M')$. Эта особенность хеш-функций очень важна, поскольку нам нужна сильная функция для проверки целостности оригинального содержимого.

- Кроме того, хеш-функции используются для индексирования баз данных. Вы будете встречаться с этими функциями на протяжении всей книги: например, с их помощью реализована система защищенного поиска, подробно описанная в главе 8, увидите их вы и в главе 9. Следует отметить, что хеш-функции потенциально обладают *квантовой стойкостью*, то есть *при определенных условиях* способны выдержать атаку квантового компьютера.
- Основополагающая концепция безопасности хеш-функции заключается в сложности получения исходного сообщения на основании выходного значения $h(M)$.

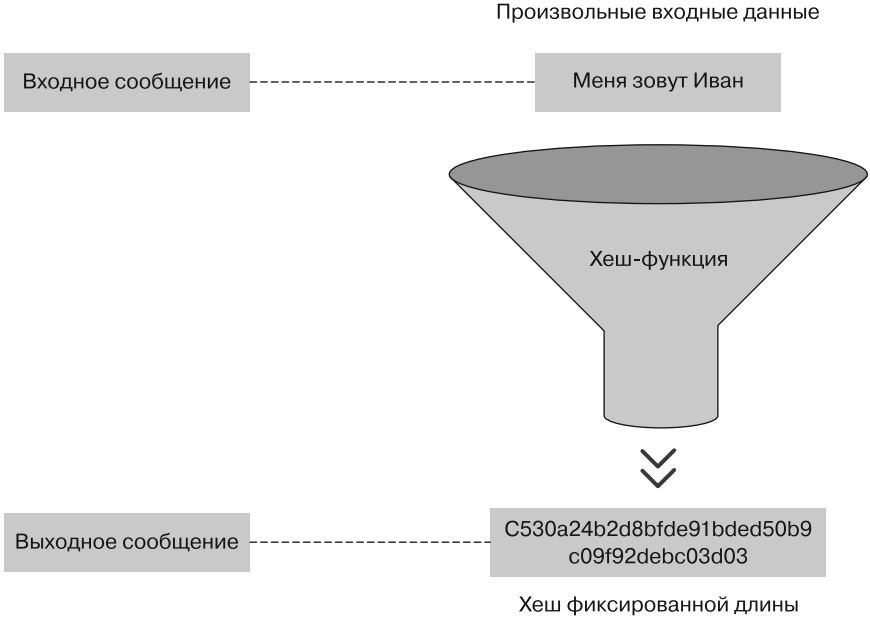


Рис. 4.2. Хеш-функция, изображенная в виде воронки-мясорубки

Это свойство обусловлено задачей *дискретного логарифма*, которую мы разобрали при изучении асимметричного шифрования:

$$g^{[a]} \equiv y \pmod{p}.$$

Как вы можете помнить, даже зная выходное значение (y) и образующий элемент (g), очень сложно найти секретный закрытый ключ [a].

Однако алгоритмы, основанные на задаче дискретного логарифма, могут быть слишком медленными и слабыми для практических реализаций. Хеши, в свою очередь, отличаются высокой скоростью вычисления. Рассмотрев хеш-функции и их свойства, можно перейти к основным алгоритмам, которые их реализуют.

Обзор основных хеш-алгоритмов

Хеш-алгоритм — это особый вид математической функции, которая при вводе переменной производит фиксированное количество битов выходных данных. Таким образом, он должен обладать устойчивостью к коллизиям. Другими словами, должно быть сложно создать две хеш-функции для одного и того же входного значения и наоборот.

Существует множество хеш-алгоритмов, но наиболее распространенными и важными являются MD5, SHA-2 и CRC32. В этой главе мы сосредоточимся на *семействе алгоритмов безопасного шифрования* (Secure Hash Algorithm, SHA).

Просто чтобы вы знали: в начале 1990-х годов европейские ученые разработали алгоритм *RIPEMD-160* в трех вариантах: помимо 160-битной версии, существовали также 256- и 320-битная. Хотя этот алгоритм не имел такого успеха, как семейство SHA, он хорошо подходит для обеспечения безопасности, поскольку до сих пор ни разу не был взломан.

В этой главе мы рассмотрим семейство алгоритмов SHA, разработанное АНБ. Я расскажу все, что нужно знать о том, как работает этот вид алгоритма. В частности, система «Биткойн» использует хеш-функцию SHA-256 в качестве *доказательства выполнения работы* (Proof of Work, PoW) при майнинге криптовалюты.

Процесс создания новых биткойнов называется майнингом. Он осуществляется путем решения чрезвычайно сложной математической задачи, основанной на хеш-функции SHA-256. Если не вдаваться в подробности, то майнинг биткойнов — это система, в которой все транзакции обрабатываются майнерами. Выбирая транзакции объемом 1 Мбайт, майнеры связывают их и передают на вход SHA-256, а затем пытаются найти определенный результат, который будет принят сетью. Первый майнер, нашедший такие выходные данные, получает вознаграждение в виде определенного количества биткойнов. Мы поговорим о семействе SHA более подробно позже, а сейчас в общих чертах рассмотрим другую хеш-функцию — MD5.

Познакомиться с хеш-функциями нам поможет простой пример хеша MD5.

Вы можете попробовать воспользоваться *генератором* MD5 для хеширования своих файлов. Это бесплатный онлайн-ресурс, на который можно перейти по

ссылке https://www.md5hashgenerator.com/. Мое имя, захешированное и преобразованное в шестнадцатеричный вид с помощью MD5, выглядит следующим образом (рис. 4.3).

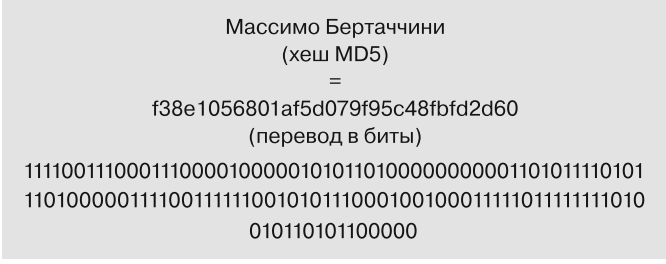


Рис. 4.3. Пример хеш-функции MD5

Семейство MD было признано небезопасным. Обнаруженные атаки на MD5, актуальные и для алгоритма RIPEMD, основаны на дифференциальном анализе и создании коллизии между двумя различными входными сообщениями.

Итак, перейдем к математическим основам и реализации хеш-функций.

Операции и обозначения для реализации хеш-функций

В этом разделе вы узнаете больше об операциях, выполняемых внутри хеш-функций, и обозначениях, используемых для их представления.

Для этого нам необходимо вернуться к булевой алгебре из главы 2. Однако здесь я представлю еще больше символов, а также дополню некоторые базовые понятия.

Как говорилось в главе 2, булева алгебра подразумевает выполнение операций над битами. Если взять два десятичных числа, а затем перенести математические и логические операции на соответствующие им двоичные числа, то в результате получится следующее.

- $X \wedge Y$ = логическая конъюнкция И — побитовое умножение (mod 2). Когда обе переменные равны 1, результат будет равен 1. В противном случае результат равен 0:

$$X = 60 \rightarrow 00111100;$$
$$Y = 240 \rightarrow 11110000;$$
$$\text{И} = 48 \rightarrow 00110000.$$

- $X \vee Y$ = логическая дизъюнкция ИЛИ — побитовая операция (mod 2), в которой результат равен 1, если хотя бы одна переменная в операции равна 1. В противном случае результат равен 0:

$$X = 60 \rightarrow 00111100;$$
$$Y = 240 \rightarrow 11110000;$$
$$\text{ИЛИ} = 252 \rightarrow 11111100.$$

- $X \oplus Y$ = XOR — побитовая сумма (mod 2). Мы уже знаем, как работает операция XOR: когда биты различны, результат равен 1. В противном случае результат равен 0:

$$a = 60 \rightarrow 00111100;$$
$$b = 240 \rightarrow \mathbf{11110000};$$
$$\text{XOR} = 204 \rightarrow 11001100.$$

Кроме того, для реализации хеш-функций используются и другие полезные операции.

- $\neg X$. Эта операция (оператор НЕ, или оператор *инверсии* ~) преобразует 1 в 0 и 0 в 1. Например, двоичная строка:

$$01010101 —$$

принимает вид:

$$10101010.$$

- $X << r$. Операция сдвига битов влево, то есть сдвиг (X) бит влево на r позиций.

На рис. 4.4 показано, что происходит при сдвиге битов влево на одну позицию.

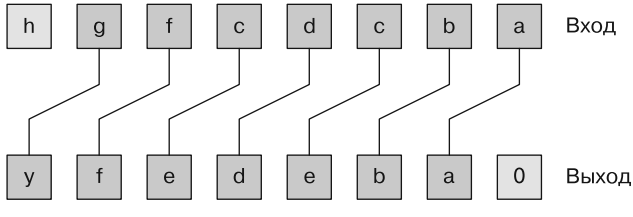


Рис. 4.4. Операция сдвига битов влево

Допустим, у нас есть десятичное число 23, или 00010111 в двоичной системе счисления. Если мы выполним сдвиг влево, то получим изображенное на рис. 4.5.

Результатом является $(00101110)_2$, или 46 в десятичной системе счисления.

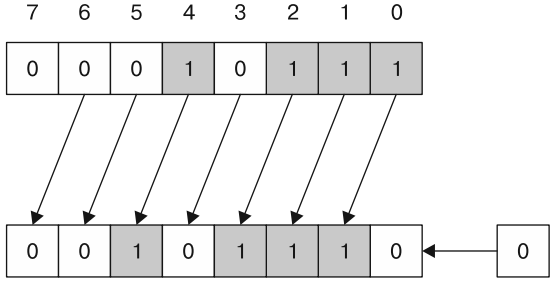


Рис. 4.5. Пример операции сдвига битов влево (на одну позицию)

- $X >> r$ — операция сдвига битов вправо. Она аналогична предыдущей с тем отличием, что сдвигает биты вправо.

  На рис. 4.6 показан сдвиг битов вправо.

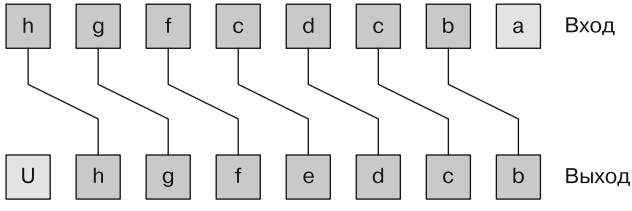


Рис. 4.6. Операция сдвига битов вправо (на одну позицию)

Рассмотрим эту операцию на том же примере десятичного числа 23, или 00010111 в двоичной системе счисления. Если мы выполним сдвиг битов вправо, то получим десятичное число 11 (рис. 4.7).

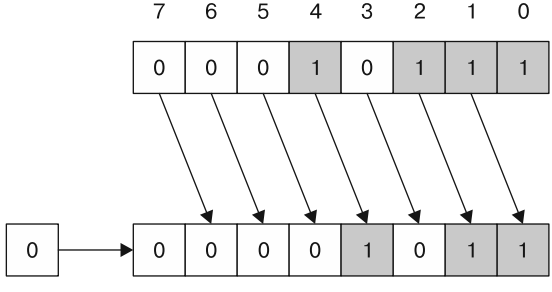


Рис. 4.7. Пример операции сдвига битов вправо (на одну позицию)

- $X \hookleftarrow r$. Циклический сдвиг битов влево. Этот оператор (также обозначается как <<<) подразумевает циклическое движение битов, аналогичное сдвигу, но с тем ключевым отличием, что начальный бит становится конечным

(рис. 4.8). В алгоритме SHA-1 он используется для циклического сдвига переменных *A* и *B* на 5 и 30 позиций соответственно (*A* <<< 5, *B* <<< 30), о чем будет рассказано в следующем подразделе.

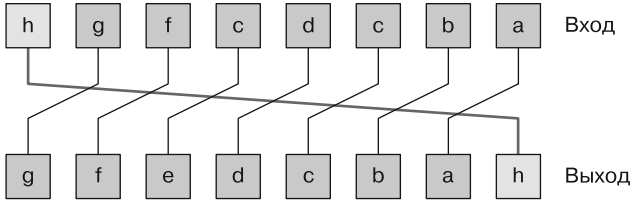


Рис. 4.8. Циклический сдвиг битов влево (на одну позицию)

Если мы применим циклический сдвиг битов влево к числу 23 в двоичной системе счисления (00010111), то получим изображенное на рис. 4.9.

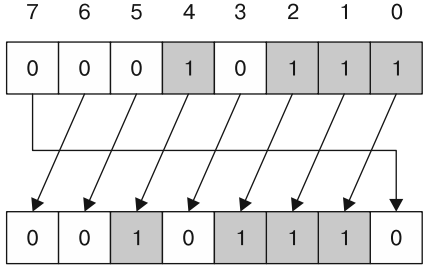


Рис. 4.9. Пример циклического сдвига битов влево

Результат этой операции совпадает с результатом операции сдвига битов влево — $(00101110)_2 = 46$. Однако если мы выполним циклический сдвиг битов вправо, то получим изображенное на рис. 4.10.

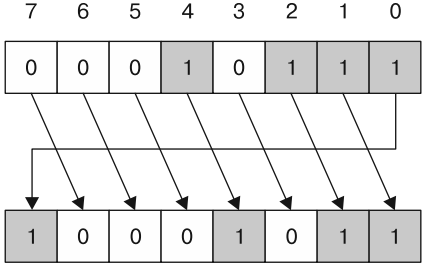


Рис. 4.10. Пример циклического сдвига битов вправо

В этом случае результатом операции будет $(10001011)_2$, или 139 в десятичной системе счисления.

- $\boxplus$. Этот оператор представляет модульную сумму $(X + Y) \pmod{2^{32}}$ в SHA, как вы увидите позже при рассмотрении алгоритма SHA-1 (рис. 4.14).

Кроме логических операторов, используемых для выполнения хеш-функций, необходимо обратить внимание на две детали, связанные с их реализацией.

- В SHA есть константы (K_t) от 0 до n (мы рассмотрим их в следующем подразделе).
- Как правило, записи ведутся в шестнадцатеричной системе счисления.

Посмотрим, как работает шестнадцатеричная система счисления. Я предполагаю, что вы с ней уже знакомы. Тем не менее на рис. 4.11 сравниваются двоичная, шестнадцатеричная и десятичная системы.

Двоичная система, b = 2	Шестнадцатеричная система, b = 16	Десятичная система, b = 10
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Рис. 4.11. Сравнение двоичной, шестнадцатеричной и десятичной систем

Как правило, шестнадцатеричная система счисления использует двоичные числа от 0 до 9, каждое из которых обозначено четырьмя битами, например:

- 0 = 0000;
- 1 = 0001;
- 2 = 0010;
- ...
- 9 = 1001.

Числа от 10 до 16 обозначены шестью прописными буквами: A, B, C, D, E и F, и в общей сложности получается 16 шестнадцатеричных чисел:

- A = 1010;
- B = 1011;
- ...
- F = 1111.

Для представления байтов используется шестнадцатеричная система счисления, где 1 байт — это восьмиразрядное двоичное число.

Например, 1 байт (8 бит) — это $(1111\ 0010)_2 = [F2]_{16} = 242$.

Теперь, когда у нас есть инструменты для определения хеш-функции, приступим к разбору алгоритма SHA-1 — самой простой модели семейства SHA.

Объяснение алгоритма SHA-1

Алгоритм безопасного хеширования 1 (Secure Hash Algorithm 1, SHA-1) разработан АНБ. В 2005 году SHA-1 был взломан и заменен последующими версиями SHA-2 и SHA-3. Тем не менее в этом разделе мы рассмотрим именно SHA-1 — просто в качестве примера реализации хеш-функции.

Выходными данными SHA-1 является 160-битное хеш-значение, полученное в результате итерационный процедуры. Эта концепция станет более понятной далее.

Как и в других хеш-функциях, сообщение $[m]$, состоящее из входных переменных битов, разбивается на 512-битные блоки фиксированной длины: $m = [m_1, m_2, m_3 ... m_l]$.

В конце этого раздела вы увидите, как 2800-битное входное сообщение $[m]$ будет разбито на блоки $[m_1, m_2 ... m_l]$ по 512 бит каждый.

Блоки создаются с помощью *функции сжатия* $f(H)$, которую мы подробнее рассмотрим на шаге 3 алгоритма. Пока же достаточно знать, что она объединяет блок текущего раунда с результатом, полученным в предыдущем раунде. Процесс обработки формируется из четырех раундов, обозначаемых переменной t. Как показано на рис. 4.12, раунд t состоит из 20 шагов (всего получается 80 шагов). Каждую итерацию можно рассматривать как счетчик, который проходит по 20 значениям каждого раунда. Как видно на рис. 4.12 и 4.13, в каждой итерации используются константы (K_t) и операции $f_t(B, C, D)$, которые зависят от текущего раунда.

В каждом раунде обновляются подрегистры (A, B, C, D, E). В конце четвертого раунда, когда $t = 79$, подрегистры (A, B, C, D, E) добавляются к подрегистрам $(H_0, H_1, H_2, H_3, H_4)$, в результате чего генерируется окончательное хеш-значение размером 160 бит.

Теперь рассмотрим, что такое константы в SHA-1.

Константами называются фиксированные шестнадцатеричные числа, определенные специальными критериями. Важным критерием при выборе констант в SHA является предотвращение коллизий. Коллизии происходят, когда два разных блока входных данных с разными константами имеют одинаковое хеш-значение. Таким образом, даже если кому-то может показаться, что было бы неплохо изменить константы, не следует этого делать, потому что это может привести к возникновению коллизии.

Например, в алгоритме SHA-1 заданные константы выглядят следующим образом (рис. 4.12).

$$K_t = \begin{cases} \text{5A827999} & \text{если} \quad 00 \leqslant t \geqslant 19 \\ \text{6ED9EBA1} & \text{если} \quad 20 \leqslant t \geqslant 39 \\ \text{8F1BBCDC} & \text{если} \quad 40 \leqslant t \geqslant 59 \\ \text{CA62C1D6} & \text{если} \quad 60 \leqslant t \geqslant 79 \end{cases}$$

Рис. 4.12. Константы K_t в SHA-1

Как видите, в разных раундах t константам соответствуют разные значения.

Помимо констант, у нас есть функция $f_t(B, C, D)$, определенная следующим образом (рис. 4.13).

$$f_t(B, C, D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{если} \quad 00 \leqslant t \geqslant 19 \\ B \oplus C \oplus D & \text{если} \quad 20 \leqslant t \geqslant 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{если} \quad 40 \leqslant t \geqslant 59 \\ B \oplus C \oplus D & \text{если} \quad 60 \leqslant t \geqslant 79 \end{cases}$$

Рис. 4.13. Функция f_t

Первым, начальным регистром SHA-1 является X_0 — 160-битная хеш-функция, сгенерированная пятью подрегистрами (H_0, H_1, H_2, H_3, H_4), каждый из которых состоит из 32 бит. Инициализация этих подрегистров происходит на шаге 2 с использованием констант, выраженных шестнадцатеричными числами.

Рассмотрим алгоритм SHA-1, разделив процесс получения конечного 160-битного хеш-значения на четыре шага.

- *Шаг 1.* Сначала выполняется конкатенация битов сообщения $[m]$:

  $$y = m_1 \| m_2 \| m_3 \| \ldots \| m_L,$$

  где символ $\|$ обозначает конкатенацию битов, выраженную каждым блоком сообщения $[m_L]$, состоящим из 512 бит.

- *Шаг 2*. Инициализация подрегистров: H_0 = 67452301, H_1 = EFCDAB89, H_2 = 98BADCFE, H_3 = 10325476, H_4 = C3D2E1F0.

  Эти константы, выраженные в шестнадцатеричной системе счисления, были выбраны разработчиками алгоритма из АНБ.
- *Шаг 3*. Помните, что функция сжатия объединяет блок текущего раунда с результатом, полученным в предыдущем раунде? Посмотрим, как это выглядит на практике. При j = 0, 1... L – 1 выполняются следующие инструкции.
  - $m_i = W_0 \| W_1 \| ... \| W_{15}$,

    где каждый (W_j) состоит из 32 бит.
  - Для t = 16... 79 вычисляем:

    $$W_t = (W_{t3} \oplus W_{t8} \oplus W_{t14} \oplus W_{t16}) \hookleftarrow 1e.$$
  - В начале раунда устанавливаем:

    $$A = H_0; B = H_1; C = H_2; D = H_3; E = H_4.$$

    Каждая переменная (A, B, C, D, E) имеет длину 32 бита, а общая длина подрегистра составляет 160 бит.
  - В 80 итерациях, где $0 \leqslant t \leqslant 79$, последовательно выполняются такие операции:

    $$T = (A \hookleftarrow 5) + f_t(B, C, D) + E + W_t + K_t = D;$$
    $$D = C; C = (B \hookleftarrow 30); B = A; A = T.$$
  - Подрегистры (A, B, C, D, E) добавляются к подрегистрам (H_0, H_1, H_2, H_3, H_4):

    $$H_0 = H_0 + A; H_1 = H_1 + B; H_2 = H_2 + C;$$
    $$H_3 = H_3 + D; H_4 = H_4 + E.$$
- *Шаг 4*. На выходе получаем:

  $$H_0 \| H_1 \| H_2 \| H_3 \| H_4.$$

  Это хеш-значение длиной 160 бит.

На рис. 4.14 представлена схема SHA-1 (с подрегистрами A, B, C, D, E).

Из предыдущего раздела мы знаем следующее:

$X \hookleftarrow r$ — это циклический сдвиг битов влево, также обозначаемый <<<. Следовательно, $A \hookleftarrow 5$ — это циклический сдвиг на 5 позиций влево, а $B \hookleftarrow 30$ — циклический сдвиг на 30 позиций влево.

Оператор $\boxplus$ представляет модульную *сумму* $(X + Y) \pmod{2^{32}}$.

В качестве примера можно рассмотреть схему, на которой представлен один раунд X_j (рис. 4.15).

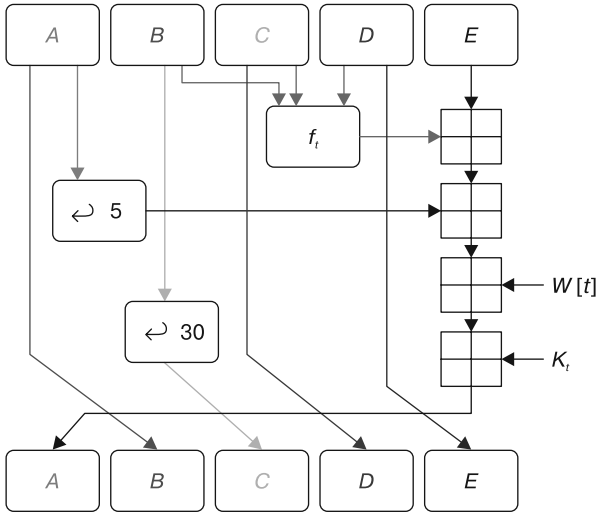


Рис. 4.14. Схема SHA-1 с операциями в каждом подрегистре

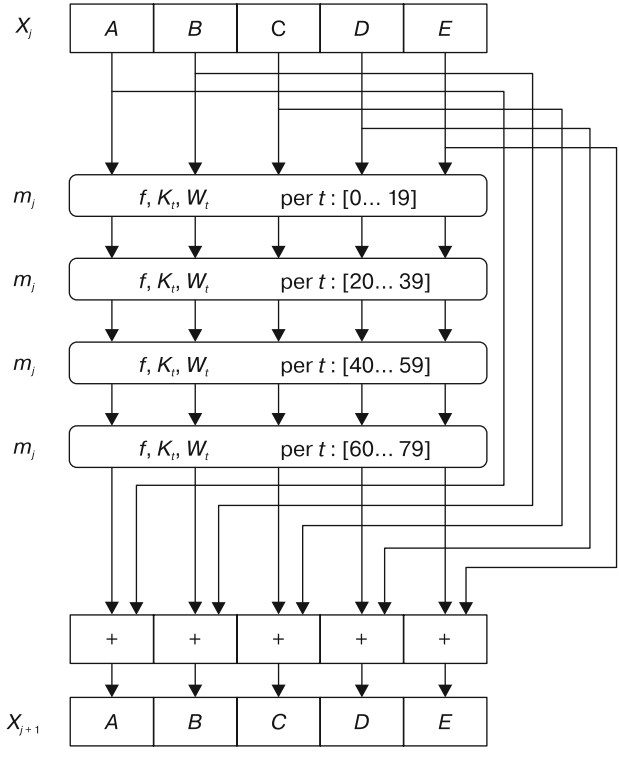


Рис. 4.15. Один раунд X_j

Заметки и пример по SHA-1

В этом разделе разберем алгоритм SHA-1 немного подробнее.

Основной частью алгоритма является шаг 3, так как шаги 1 и 2 — это просто инициализация сообщений и подрегистров. Шаг 3 содержит серию математических операций, состоящих из конкатенации битов, операции XOR, сдвига, транспонирования и сложения битов.

На шаге 4 происходит уменьшение выходных данных до 160 бит только потому, что размер каждого подрегистра — H_0, H_1, H_2, H_3 и H_4 — составляет 32 бита. Описывая шаг 1, мы упоминали, что минимальный размер блока входного сообщения должен составлять 512 бит. Сообщение [m] разбивается на блоки по 512 бит. Если же исходное сообщение меньше 512 бит, необходимо применить операцию добавления битов для заполнения блока.

Алгоритм SHA-1 вычисляет 160-битовый дайджест произвольного входного сообщения. Входное сообщение имеет вид битовой строки, поэтому его длина измеряется количеством битов, то есть длина пустого сообщения — 0. Если длина сообщения кратна 8 битам (1 байту), его можно представить в компактном шестнадцатеричном виде. В противном случае к сообщению необходимо применить дополнение. Цель дополнения заключается в том, чтобы общая длина сообщения стала кратной 512. Поскольку SHA-1 вычисляет дайджест сообщения путем последовательной обработки блоков длиной 512 бит, ключевым моментом для получения сильного дайджеста является обеспечение высокой степени *запутывания* и *рассеивания* битов в итерационных операциях хеш-функции.

Мы говорили о запутывании и рассеивании по теореме Шеннона в главе 2. Вкратце напомню: запутывание связано с увеличением энтропии шифротекста в отношении ключа, что увеличивает неоднозначность самого шифротекста. Рассеивание означает, что изменение одного бита в шифротексте (статистически) изменит половину битов в открытом тексте, что усложняет дешифрование.

Дополнение состоит из такой последовательности шагов.

1. SHA-1 берет исходное сообщение и добавляет к нему один бит 1, за которым следует последовательность битов 0.
2. Биты 0 добавляются для того, чтобы длина нового сообщения стала кратной 512 битам.
3. Таким образом, новое сообщение будет иметь конечную длину $n \cdot 512$.

Например, если исходное сообщение имеет длину 2800 бит, его необходимо дополнить одним битом 1 в конце, а затем еще 207 битами 0. Таким образом, 2800 + 1 + 207 = 3008 бит. Затем с помощью алгоритма деления проверяется кратность полученного результата: $3008 = 5 \cdot 512 + 448$. В конец этой строки до-

бавляют длину исходного сообщения. Если двоичное представление этой длины меньше 64 бит, то перед ней добавляют нули. Таким образом, 3008 + 64 = 3072, что кратно 512 (битам блока)[1].

Пример кодирования одного блока с помощью SHA-1

Подкрепим представленную информацию практическим примером.

- *Шаг 1. Дополнение сообщения*. Предположим, мы хотим закодировать сообщение *abc* с помощью SHA-1. В двоичной системе счисления оно представлено в виде:

  $$abc = 01100001\ 01100010\ 01100011.$$

  В шестнадцатеричном виде это такая строка:

  $$abc = 616263.$$

  Как видно на рис. 4.16, сообщение дополняется битом 1, а затем добавляется такое количество битов 0, чтобы длина сообщения составила 448 бит. Поскольку в исходном сообщении *abc* 24 бита, к нему добавляется еще 423 бита 0. В конце добавляется длина сообщения в 64-битном формате. В результате получается сообщение длиной 512 бит.

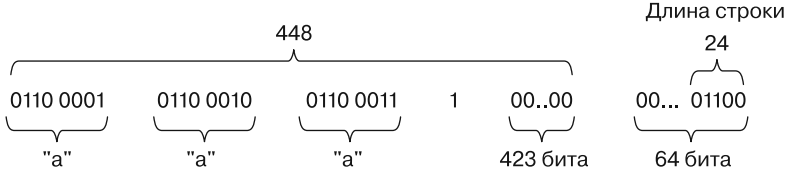


Рис. 4.16. Дополнение сообщения в SHA-1

- *Шаг 2. Инициализация подрегистров*. Как мы знаем, подрегистрам H_0, H_1, H_2, H_3 и H_4 уже заданы хеш-значения:

  $$H[0] = 67452301;$$
  $$H[1] = EFCDAB89;$$
  $$H[2] = 98BADCFE;$$
  $$H[3] = 10325476;$$
  $$H[4] = C3D2E1F0.$$

[1] Число 448 бит в процедуре дополнения SHA-1 не является произвольным. Каждый блок сообщения в SHA-1 имеет размер 512 бит, из которых последние 64 бита строго зарезервированы для записи длины исходного сообщения. Поэтому перед добавлением этого 64-битного поля длины общая длина данных должна составлять 512 – 64 = 448 бит.

- *Шаг 3. Содержимое блока*[1]*:*

$W[0]$ = 61626380; $W[1]$ = 00000000; $W[2]$ = 00000000; $W[3]$ = 00000000;
$W[4]$ = 00000000; $W[5]$ = 00000000; $W[6]$ = 00000000; $W[7]$ = 00000000;
$W[8]$ = 00000000; $W[9]$ = 00000000; $W[10]$ = 00000000; $W[11]$ = 00000000;
$W[12]$ = 00000000; $W[13]$ = 00000000; $W[14]$ = 00000000; $W[15]$ = 00000018.

Выполнение цикла операций на подрегистрах:

A B C D E

t = 0: 0116FC33 67452301 7BF36AE2 98BADCFE 10325476;
t = 1: 8990536D 0116FC33 59D148C0 7BF36AE2 98BADCFE;
...
T_t = 79: 42541B35 5738D5E1 21834873 681E6DF6 D8FDF6AD.

Сложение подрегистров:

$H[0]$ = 67452301 + 42541B35 = A9993E36;
$H[1]$ = EFCDAB89 + 5738D5E1 = 4706816A;
$H[2]$ = 98BADCFE + 21834873 = BA3E2571;
$H[3]$ = 10325476 + 681E6DF6 = 7850C26C;
$H[4]$ = C3D2E1F0 + D8FDF6AD = 9CD0D89D.

- *Шаг 4. Результат.* После выполнения четырех раундов итоговый дайджест сообщения строки *abc* 160-битного хеша имеет вид (рис. 4.17):

A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D.

Прежде чем приступить к изучению мира цифровых подписей, я хочу уделить внимание взаимосвязи хеша и цифровой подписи.

Хеш-функции, как вы увидите позже, широко используются при выполнении цифровых подписей. Так, в RSA хеш-функция необходима для того, чтобы предотвратить раскрытие содержимого сообщения при проверке цифровой подписи. Поэтому обычно сообщение [*M*] заменяется его хешем.

Теперь мы готовы к изучению цифровых подписей.

---

[1] Все блоки записаны в шестнадцатеричном виде; 80 в такой записи — это 1000 0000 в двоичном представлении — добавочная единица и далее нули для «добивки» в 448 бит. В блоке W[15]: 18 — это 24 в десятичной системе счисления (длина исходного сообщения).

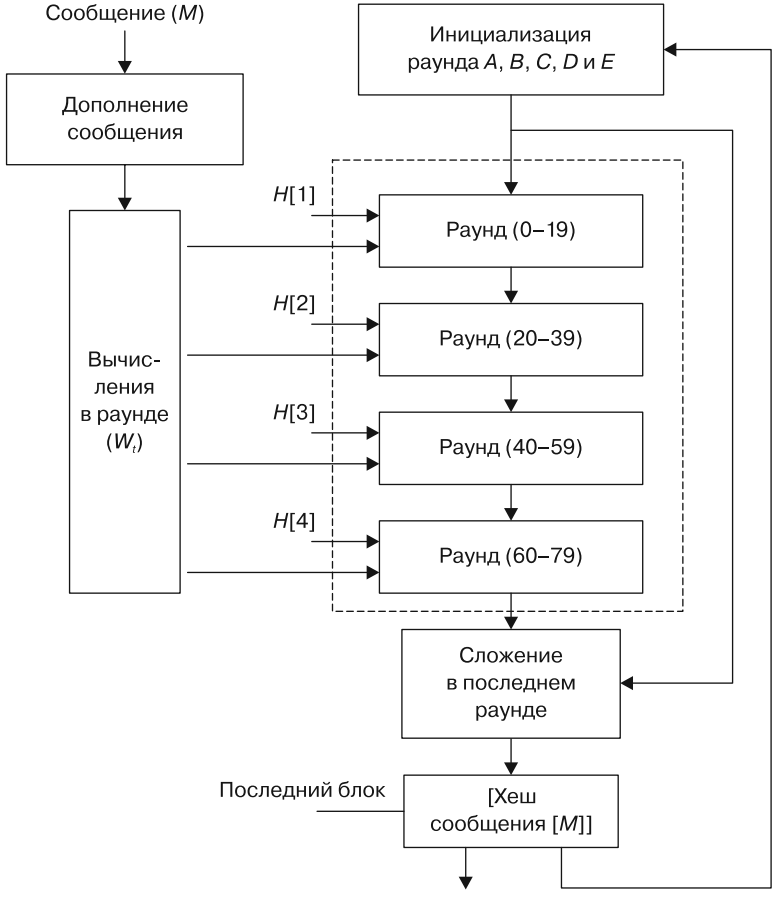


Рис. 4.17. Полный раунд в SHA-1

Аутентификация и цифровые подписи

В криптографии *аутентификация* — одна из самых интересных и сложных проблем. Она является также одной из наиболее важных и часто используемых при управлении доступом функций.

Существует три основных метода аутентификации:

- знание — что-то, что *знает только пользователь* (например, пароль);
- владение — что-то, что *есть только у пользователя* (смарт-карта, устройство или метка);

- свойство — что-то, что *характеризует пользователя* (например, отпечатки пальцев, сканирование радужной оболочки глаза и другие биометрические характеристики человека).

Как вы увидите далее, аутентификация может выполняться не только с помощью цифровых подписей, но и другими методами, например на основе доказательства с нулевым разглашением, которому посвящена глава 5.

Посмотрим, как происходит аутентификация с использованием цифровой подписи поверх шифрования с открытым и закрытым ключами.

Например, Алиса хочет передать сообщение Бобу. Она знает его открытый ключ, поэтому шифрует сообщение $[M]$ с помощью алгоритма RSA и отправляет его Бобу.

Передача сообщения сопряжена с риском возникновения следующих распространенных проблем.

1. Как Боб может быть уверен, что сообщение пришло от Алисы *(проблема аутентификации)*?
2. После получения сообщения Бобом Алиса всегда имеет возможность опровергнуть то, что именно она его отправила *(отказ от информации)*.
3. Злоумышленник может перехватить и подделать сообщение $[M]$ (атака *посредника* — MitM), изменив часть его содержимого. В таком случае сообщение *теряет* свою *целостность*.
4. Мы об этом еще не говорили, однако злоумышленник может перехватить и просмотреть сообщение, не меняя его. Как отправитель и получатель могут быть уверены, что этого не произойдет? Например, как можно гарантировать, что провайдер связи или облако, в котором хранятся ваши данные, не шпионит за вами? Все это можно отнести к проблеме шпионажа, и, как вы увидите позже, существует решение, позволяющее выявить и избежать ее.

Отвечая на последний вопрос, можно справедливо заметить, что злоумышленнику трудно, если не невозможно восстановить секретное сообщение, не зная ключа, поскольку сообщение $[M]$ зашифровано.

Тем не менее я покажу, что при некоторых условиях можно увидеть содержание зашифрованного сообщения, даже не зная секретного ключа, необходимого для расшифровки.

В этой главе мы обсудим первые три проблемы. Что касается четвертой, то я расскажу о шпионских атаках и способе их предотвращения в главе 6. Там я представлю несколько собственных алгоритмов и поделюсь тем, с какими проблемами мне пришлось столкнуться во время работы над решениями для защиты.

Цифровые подписи в RSA

Как я уже говорил, подписать сообщение можно разными способами. Сейчас мы рассмотрим, как осуществляется подписание с математической точки зрения и как любой человек может проверить подпись.

Цифровая подпись является доказательством того, что отправитель сообщения $[M]$ поручает получателю выполнить следующие действия:

- подтвердить свою личность;
- подтвердить, что над сообщением $[M]$ не было проведено никаких манипуляций;
- обеспечить невозможность отказаться от сообщения.

Давайте разберемся, как цифровые подписи могут помочь решить все эти задачи и каким образом ставится подпись на сообщении в зависимости от типа алгоритма и метода подписания.

Начнем с рассмотрения цифровых подписей в RSA, а затем перейдем к другим способам подписания в алгоритмах с открытым и закрытым ключами.

Из главы 3 вы знаете, что со стороны Боба процесс шифрования RSA выглядит следующим образом.

Шифрование выполняется с использованием открытого ключа Алисы (N_A). Также определяются секретные элементы внутри квадратных скобок $[M]$:

$$[M]^e \equiv c \pmod{N_A}.$$

Таким образом, Боб шифрует сообщение $[M]$ с помощью открытого ключа Алисы, а Алиса расшифровывает $[M]$ своим закрытым ключом $[d_A]$, выполнив следующую операцию:

$$c^{[d_A]} \equiv [M] \pmod{N_A},$$

где параметр $[d_A]$ создается с помощью операции:

$$\text{INV}\ (e) \equiv [d_A] \bmod (p-1)(q-1).$$

Процесс подписания и проверки личности Боба выглядит следующим образом.

- *Шаг 1.* Боб выбирает два больших простых числа, $[p_1]$ и $[q_1]$, и держит их в секрете.
- *Шаг 2.* Боб вычисляет $N_B = [p_1] \cdot [q_1]$ и публикует его в качестве своего открытого ключа:

$$\text{INV}\ (e) \equiv [d_B] \bmod (p_1-1)(q_1-1).$$

- *Шаг 3.* Боб вычисляет свой закрытый ключ $[d_B]$.

- *Шаг 4*. Боб создает подпись:

$$S \equiv [M]^{[d_B]} (\mathrm{mod}\, N_B),$$

где (S) и (N_B) — открытые параметры, а [d_B] держится в секрете.

А что же [M]?

Предполагается, что сообщение [M] должно оставаться в секрете и передаваться только между Бобом и Алисой. Однако цифровая подпись должна быть доступна для проверки любому человеку. Мы вернемся к этому вопросу позже, а сейчас проверим подпись (S) для сообщения между Бобом и Алисой.

- *Шаг 5*. Проверка подписи. Это обратный процесс. Алиса или любой другой человек может выполнить следующую проверку:

$$S^e \equiv [M] \ (\mathrm{mod}\ N_B).$$

Если это уравнение проходит проверку, то Алиса принимает сообщение [M].

Прежде чем приступить к проблеме, связанной с [M], рассмотрим этот алгоритм на примере.

Числовой пример. Возьмем числовые параметры из примера RSA, который мы разбирали в главе 3:

$$[M] = 88;$$
$$e = 9007.$$

Параметры Алисы:

$$[p] = 101;$$
$$[q] = 67;$$
$$N_A = 6767.$$

Выполним все шаги RSA, чтобы показать полный процесс подписания сообщения.

- *Шаг 1*. Боб шифрует сообщение:

$$[M]^e \equiv c \ (\mathrm{mod}\ N_A);$$
$$88^{9007} \equiv 6621 \ (\mathrm{mod}\ 6767);$$
$$c = 6621.$$

- *Шаг 2*. Боб генерирует подпись (S) для сообщения [M]. Он выбирает два простых числа, [p_1] и [q_1]:

$$[p_1] = 211;$$
$$[q_1] = 113;$$
$$N_B = 211 \cdot 113 = 23\,843;$$
$$9007 \cdot [d_B] \equiv 1 \ (\mathrm{mod}\ (211 - 1) \cdot (113 - 1).$$

Решив модульное уравнение для $[d_B]$, он получает:

$$[d_B] = 9103.$$

Теперь Боб может подписать сообщение:

$$[M]^{[d_B]} \equiv S \pmod{N_B};$$
$$88^{9103} \equiv 19\,354 \pmod{23\,843};$$
$$S = 19\,354.$$

Он отправляет Алисе пару $(c, S) = (6621, 19\,354)$.

- *Шаг 3*. Алиса расшифровывает сообщение:

$$c^{[d_A]} \equiv M \pmod{N_A}.$$

Она вычисляет $[d_A]$:

$$9007 \cdot [d_A] \equiv 1 \pmod{(101 - 1) \cdot (67 - 1)};$$
$$[d_A] \equiv 3943.$$

Затем расшифровывает криптограмму (c) и получает сообщение $[M]$:

$$c^{[d_A]} \equiv M \pmod{N_A};$$
$$6621^{3943} \equiv 88 \pmod{6767}.$$

- *Шаг 4*. Проверка личности Боба. Если в результате возведения подписи (S) в степень параметра (e) получается сообщение $[M]$, то Алиса может быть уверена в том, что сообщение действительно было отправлено Бобом:

$$S^e \equiv [M] \pmod{N_B};$$
$$19\,354^{9007} = 88 \pmod{23\,843}.$$

Так и есть, Алиса получает сообщение $M = 88$.

Надеюсь, кто-то заметил, что сообщение $[M]$ может быть проверено кем угодно, не только Алисой, потому что (S), (e), (N_B) — это открытые параметры. Таким образом, если при создании своей подписи (S) Боб будет использовать исходное сообщение $[M]$, а не его хеш $h[M]$, то каждый, кто знает открытый ключ Боба, сможет легко восстановить сообщение $[M]$!

Чтобы восстановить секретное сообщение $[M]$, достаточно решить уравнение:

$$S^e \equiv x \pmod{N_B},$$

где параметры (S), (e), (N_B) известны.

Вот здесь и приходят на помощь *хеш-функции*. Выполнив хеш-функцию $h[M] = (m)$, Боб отправляет пару (c, S), подписав (m) вместо $[M]$.

Алиса уже получила зашифрованное сообщение [M], поэтому проверить его может только она:

$$h[M] = m.$$

Результат ИСТИНА подтверждает личность Боба, а результат ЛОЖЬ означает отказ в принятии его заявления личности.

Почему цифровые подписи работают

Если подпись Боба попытается использовать кто-то другой, он столкнется с необходимостью решить задачу дискретного логарифма. Генерирование значения [d_B] в следующем уравнении — серьезная проблема для злоумышленника:

$$m^{[d_B]} \equiv S \pmod{N_B}.$$

Кроме того, даже если злоумышленнику известны значения (m) и (S), вычисление [d_B] все равно остается очень сложной задачей. Мы рассматривали задачу дискретного логарифма в главе 3.

Посмотрим на примере, что произойдет, если Ева (злоумышленник) попытается изменить подпись.

Ева подменяет [d_B] на (d_E), вычисляя поддельную цифровую подпись (S'):

$$m^{(d_E)} \equiv S' \pmod{N_B}.$$

Затем ей необходимо заставить Алису принять подпись (S'). Для этого она совершает атаку посредника, притворившись Бобом и подменив настоящую подпись (S) своей (S').

Но когда Алиса проверит подпись (S'), она увидит, что та не соответствует настоящей подписи Боба, так как хеш сообщения отличается — вместо (m) она получила (m'):

$$(S')^e \equiv m' \pmod{N_B};$$

$$m' \not\equiv m.$$

Таким образом, Алиса отклоняет цифровую подпись и не будет открывать сообщения, пришедшие с этого фиктивного адреса. Другими словами, когда Алиса получает (m') вместо (m), она понимает, что есть какая-то проблема, и не принимает сообщение [M].

Именно поэтому криптографам необходимо уделять большое внимание коллизиям между хешами.

Так действует подпись (S).

Предыдущая атака — это довольно простой трюк. Однако существуют более умные и хитрые атаки, которые мы увидим в главе 6, когда будем изучать не-

стандартные методы взлома. При этом следует отметить, что обмен открытыми ключами играет важную роль в предотвращении атак посредника на инфраструктуру открытых ключей.

Цифровые подписи в схеме Эль-Гамаля

В главе 3 мы познакомились со *схемой Эль-Гамаля* — с алгоритмом с открытым и закрытым ключами, основанным на *обмене ключами Диффи — Хеллмана*.

Способы подписания сообщения в схеме Эль-Гамаля и в алгоритме RSA отличаются друг от друга, но одинаково эффективны. Выбор в пользу одного из них обусловлен тем, что в основе RSA лежит задача факторизации, а в основе схемы Эль-Гамаля — математическая задача дискретного логарифма.

Как вы помните, шифрование в схеме Эль-Гамаля осуществляется с помощью следующих элементов:

- (g) и (p) — открытые параметры;
- $[k]$ — закрытый временный ключ Алисы;
- $[M]$ — секретное сообщение;
- $B \equiv g^{[b]} \pmod{p}$ — открытый ключ Боба, b — закрытый ключ Боба;
- $A \equiv g^{[a]} \pmod{p}$ — открытый ключ Алисы, a — закрытый ключ Алисы.

Шифрование Алисы:

$$y_1 \equiv g^{[k]} \pmod{p};$$
$$y_2 \equiv [M] \cdot B^{[k]} \pmod{p}.$$

Не забывайте, что элементы в квадратных скобках — это секретные параметры. Все остальное носит открытый характер.

Теперь, если Алиса хочет добавить к сообщению свою цифровую подпись, она защищает сообщение $[M]$ с помощью *хеширования* $h[M]$ и передает результат Бобу, чтобы подтвердить свою личность.

Чтобы подписать сообщение $[M]$, Алиса сначала должна сгенерировать хеш сообщения $h[M]$:

$$h[M] = m.$$

Теперь она может оперировать открытым значением дайджеста $[M] \rightarrow (m)$, поскольку, как мы уже выяснили, восстановить $[M]$ с помощью его криптографического хеша (m) практически невозможно.

Алиса создает подпись (S) следующим образом.

- *Шаг 1.* Вычисляет обратное значение $[k]$ по (mod ($p - 1$)):

$$[\text{INV } k] \equiv k^{(-1)} \pmod{(p-1)}.$$

- *Шаг 2.* Решает уравнение:

$$S \equiv [\text{INV } k] \cdot (m - [a] \cdot y_1) \pmod{(p-1)} -$$

и отправляет Бобу открытые параметры (m), (y_1), (S).

- *Шаг 3.* При первой проверке Боб вычисляет V_1:

$$V_1 \equiv A^{(y_1)} \cdot y_1^{\wedge(S)} \pmod{p}.$$

После этапа дешифрования, если $h[M] = m$, Боб получает второй параметр проверки V_2:

$$V_2 \equiv g^m \pmod{p}.$$

Если $V_1 = V_2$, Боб принимает сообщение.

Числовой пример. Предположим, что секретное сообщение имеет вид $[M] = 88$.

Остальным открытым параметрам присвоим следующие значения:

- $g = 7$;
- $p = 200\,003$;
- $h[M] = 77$.

Первым шагом является инициализация закрытого и открытого ключей:

- $[b] = 2367$ (закрытый ключ Боба);
- $[a] = 5433$ (закрытый ключ Алисы);
- $[k] = 23$ (случайное секретное число Алисы);
- $B = 151\,854$ (открытый ключ Боба);
- $A = 43\,725$ (открытый ключ Алисы);
- $y_1 \equiv g^{[k]} \pmod{p} = 723 \pmod{200\,003} = 90\,914$.

После инициализации Алиса вычисляет обратное значение ключа (шаг 1), а затем подпись (шаг 2).

- *Шаг 1.* Алиса вычисляет обратную величину $[k]$ по (mod $p - 1$):

$$[\text{INV } k] \equiv [k]^{(-1)} \pmod{(p-1)} = 23^{-1} \pmod{(200\,003 - 1)} = 34\,783.$$

- *Шаг 2.* Теперь Алиса может получить подпись (S):

$$S \equiv [\text{INV } k] \cdot (m - [a] \cdot y_1) \pmod{(p-1)};$$

$$S \equiv 34\,783 \cdot (77 - (5433 \cdot 90\,914)) \pmod{(200\,003 - 1)} = 72\,577.$$

Она отправляет Бобу открытые параметры (m, y_1, S) = (77, 90 914, 72 577).

- *Шаг 3*. Эти параметры позволяют Бобу выполнить первую проверку V_1, а затем вторую — V_2. Если $V_2 = V_1$, Боб принимает цифровую подпись (S):

$$V_1 \equiv A^{(y_1)} \cdot y_1^{(S)} \pmod{p};$$

$$V_1 \equiv 43\,725^{(90\,914)} \cdot 90\,914^{72\,577} \pmod{200\,003} = 76561.$$

  Проверка (V_2):

$$V_2 \equiv g^m \pmod{p};$$

$$V_2 \equiv 7^{77} \pmod{200\,003} = 76\,561.$$

  Боб подтверждает, что $V_1 = V_2$.

Можно сказать, что задача, лежащая в основе алгоритма, аналогична нахождению дискретного логарифма. Проанализируем функцию проверки:

$$V_1 \equiv A^{(y_1)} \cdot y_1^{(S)} \pmod{p}.$$

Все элементы формируются с помощью дискретных степеней. Как вы уже знаете, возврат к исходному сообщению с помощью показателя дискретной степени, даже если известны экспонента или основание, пока сложная задача. Таким образом, будет неправильно утверждать, что дискретные степени и логарифмы обеспечивают безопасность этого алгоритма.

Как вы видели в главе 3, проблему для безопасности может нести и функция:

$$y_2 \equiv [M] \cdot B^{[k]} \pmod{p}.$$

Она представляет собой произведение, и, если злоумышленнику удастся восстановить $[k]$, он узнает $[M]$.

Таким образом, уязвимость алгоритма заключается не только в задаче дискретного логарифма, но и в задаче факторизации.

Теперь, когда вы узнали о применении и реализации цифровых подписей, перейдем к другому интересному криптографическому протоколу — слепым подписям.

Слепые подписи

Слепые подписи изобрел *Дэвид Чаум*. Он долго бился над разработкой криптографической системы для обезличивания *электронных платежей*. В 1990 году Дэвид профинансировал систему eCash, работающую с неотслеживаемой валютой. К сожалению, в 1998-м проект обанкротился, но Чаум вошел в историю криптографии как один из пионеров цифровых денег и один из отцов современной концепции криптовалюты, которая была реализована в «Биткойне».

Чаум хотел решить следующие проблемы:

- найти алгоритм, позволяющий избежать *проблемы двойного расходования* в электронных платежах;
- сделать цифровую систему *безопасной* и *анонимной* и тем самым гарантировать *конфиденциальность* пользователя.

В 1982 году Чаум написал статью «Слепые подписи для неотслеживаемых платежей». Далее объясняется, как работают слепые подписи и как их реализовать.

Поставить на сообщении слепую подпись означает подписать его, не зная о его содержании. Такую подпись можно использовать не только в электронных платежах, но и, например, для открытой регистрации изобретения без раскрытия его деталей. Еще одно применение слепых подписей — электронные машины для голосования, например, на выборах президента или партии. В этом случае результат голосования (переданное сообщение) должен быть известен получателю, но личность избирателя должна оставаться в тайне, чтобы голос был засчитан (в этом и состоит функция слепых подписей).

Инновационная схема слепой подписи для *шифра MBXI*, изобретенного и запатентованного мной в 2011 году, будет представлена в главе 6, где я расскажу о новых шифрах с закрытым и открытым ключами.

А сейчас посмотрим, как работает протокол Дэвида Чаума, создав слепую подпись с помощью RSA.

Слепая подпись в RSA

Предположим, у Боба есть важный секрет, который он не хочет раскрывать публике до определенного времени. Например, он разработал мощное лекарство от рака и претендует на Нобелевскую премию.

Алиса представляет комиссию по присуждению Нобелевской премии.

Она выбирает два больших секретных числа $[p_A, q_A]$:

- $[p_A] \cdot [q_A] = N_A$ — открытый ключ Алисы;
- (e) — открытый параметр, определенный в RSA.

Затем Алиса создает закрытый ключ $[d_A]$:

$$\text{INV}\ (e) \equiv [d_A] \bmod (p_A - 1)\ (q_A - 1).$$

Обозначим секрет Боба $[M_1]$, чтобы можно было отличать его от обычного сообщения $[M]$.

Боб выбирает случайное число $[k]$ и держит его в тайне.

Теперь он может выполнить протокол *слепой* подписи на $[M_1]$.

- *Шаг 1*. Боб *скрывает* содержимое сообщения $[M_1]$ с помощью шифрования (t):

$$t \equiv [M_1] \cdot [k]^e \pmod{N_A}.$$

  Затем отправляет (t) Алисе.

- *Шаг 2*. Алиса создает слепую подпись:

$$S \equiv t^{[d_A]} \pmod{N_A}.$$

  Она отправляет (S) Бобу, который может проверить, соответствует ли эта слепая подпись сообщению $[M_1]$.

- *Шаг 3*. Проверка. Боб вычисляет (V):

$$S / k \equiv V \pmod{N_A}.$$

  Затем он может выполнить следующую проверку:

$$V^e \equiv [M_1] \pmod{N_A}.$$

Результат ИСТИНА подтверждает, что Алиса успешно поставила *слепую* подпись на $[M_1]$. В этом случае Боб может быть уверен, что:

$$[M_1]^{[d_A]} \equiv V \pmod{N_A}.$$

Поскольку никому, кроме Алисы, не разрешается выполнить функцию (S) без решения задачи *дискретного логарифма* (это было показано в главе 3), можно с уверенностью сказать, что подпись поставила именно она. Это утверждение верно до тех пор, пока не появятся новые обстоятельства. Например, в будущем кто-то может найти эффективный способ решения задачи дискретного логарифма или квантовый компьютер достигнет достаточного количества кубитов, чтобы взломать алгоритм, как вы увидите в главе 9.

Числовой пример

1. Алиса задает следующие параметры:
   - $p_A = 67$;
   - $q_A = 101$;
   - $m = 88$ (как и в других примерах).

   Таким образом, открытым ключом (N_A) является:

$$67 \cdot 101 = N_A = 6767;$$
$$d_A \equiv 1/e \pmod{(p_A - 1) \cdot (q_A - 1)};$$
$$\text{Reduce}\,[e \cdot x = = 1, x, \text{Modulus} \to (p_A - 1)\,(q_A - 1)]^1;$$
$$[d_A] = 1553;$$
$$e = 17.$$

[1] Эта формула сработает для системы Mathematica.

2. Боб выбирает случайное число $[k] = 29$. Он вычисляет (t):

$$t \equiv M_1 \cdot k^e \equiv 88 \cdot 29^{17} = 3524 \pmod{6767}.$$

3. Боб отправляет (t) Алисе. Теперь она может поставить слепую подпись на (t):

$$S \equiv t^{[d_A]} \equiv 3524^{1553} = 1533 \pmod{6767}.$$

4. Боб может проверить (S). Алиса посылает $(S) = 1553$ Бобу:

$$S / k \equiv V \pmod{N_A};$$
$$1533 / 29 = 2853 \pmod{6767};$$
$$\text{Reduce}\,[k \cdot x = = S, x, \text{Modulus} \to N_A];$$
$$x = 2853;$$
$$V = 2853.$$

   Если:

$$V^e \equiv [M_1] \pmod{N_A};$$
$$2853^{17} \equiv 88 \pmod{6767},$$

   то Боб принимает подпись (S).

Как видно из изложенного, Алиса уверена, что разработка Боба (лекарство от рака) принадлежит именно ему, а Боб может сохранить детали своего изобретения в тайне до определенного времени.

Замечания по протоколу слепой подписи

Дополнительная проверка $[M_1]$ позволяет убедиться в том, что Алиса действительно подписала $[M_1]$, ничего не узнав о его содержимом:

$$[M_1]^{[d_A]} \equiv V \pmod{N_A};$$
$$88^{1553} \equiv V \pmod{N_A}.$$

Необходимо быть осторожными со слепыми подписями, ведь Алиса не знает, что она собирается подписать, поскольку $[M_1]$ скрыто внутри (t). Например, Боб может попытаться убедить Алису подписать чек на 1 млн долларов. Принятие таких протоколов таит в себе множество опасностей.

Кроме того, не следует забывать о возможных атаках.

На этапе шифрования мы столкнулись с задачей факторизации:

$$t \equiv [M_1] \cdot [k]^e \pmod{N_A}.$$

Мы знаем, что (t) — это произведение $[X \cdot Y]$:

$$X = M_1 \leftarrow \text{задача факторизации;}$$

$$Y = k^e.$$

Тот факт, что злоумышленник не может определить $[k]$, не имеет значения, ведь он всегда может попытаться найти $[M_1]$ путем факторизации (t) при условии, что $[M_1]$ будет небольшим числом. Если $M_1 = 0$, то, конечно, и (t) будет равно нулю, а значит, атакующему станет понятно, что сообщение $M = 0$.

В то же время если мы предположим, что (k^e) — небольшое число, поскольку $[k]$ носит случайный характер, то злоумышленник сможет выполнить операцию:

$$\text{Reduce } [(k^e) \cdot x = = t, x, \text{ Modulus} \rightarrow N_A];$$

$$x = \text{сообщение.}$$

В этом случае, если, к сожалению, $[k^e] \pmod{N_A}$ окажется небольшим числом, злоумышленник сможет восстановить сообщение $[M_1]$.

Как вы поймете из следующей главы, слепые подписи — это предшественники протоколов доказательства с нулевым разглашением. Некоторые элементы, которые мы здесь рассмотрели, такие как случайное число $[k]$, слепые подписи и последний шаг проверки $V = S / k$, выполняемый получателем сообщения, основаны на той же логике, что и протоколы доказательства с нулевым разглашением.

Резюме

В этой главе мы проанализировали хеш-функции, цифровые и слепые подписи. Мы начали с объяснения хеш-функций и математических операций, лежащих в их основе, а также подробно рассмотрели алгоритм SHA-1. Затем разобрали цифровые подписи в алгоритме RSA и схеме Эль-Гамаля, используя практические числовые примеры, и обсудили их возможные уязвимости.

Кроме того, был представлен протокол слепой подписи как важный криптографический инструмент реализации систем электронного голосования и электронных платежей.

Теперь вы знаете, что такое хеш-функция и как она работает. А также понимаете, что представляет собой цифровая подпись и как ее реализовать в алгоритме RSA и в схеме Эль-Гамаля. Мы также обсудили уязвимости, которые могут привести к раскрытию цифровых подписей, и методы их устранения.

И последнее: вы узнали, для чего нужны слепые подписи и в каких областях они применяются.

Эти темы очень важны, поскольку мы будем часто возвращаться к ним в следующих главах. Они особенно полезны для того, чтобы понять доказательство с нулевым разглашением, о котором я расскажу в главе 5, а также другие алгоритмы, представленные в главе 6. В главе 6 будут рассмотрены также новые методы атак на цифровые подписи.

Теперь, когда вы изучили основы хеш-функций и цифровых подписей, настало время перейти к следующей главе и подробно рассмотреть доказательства с нулевым разглашением.

Часть III

Новые криптографические алгоритмы и протоколы

В этой части мы рассмотрим новые протоколы и алгоритмы, появляющиеся для защиты данных в новой среде кибербезопасности, блокчейна, ИКТ и криптопоисковиков. Среди них будет несколько моих патентов и изобретений.

5 Доказательство с нулевым разглашением

Как вы видели на примере цифровой подписи, аутентификация — одна из самых важных, сложных и увлекательных проблем криптографии будущего. Представьте, что вам нужно идентифицировать себя в Интернете перед кем-то, кто вас не знает. Сначала вас просят указать имя, фамилию и адрес. Затем, возможно, потребуется предоставить номер социального страхования и другие конфиденциальные данные. Раскрытие такой информации в Сети несет в себе большие риски, так как злоумышленники могут украсть персональные данные и использовать их в преступных целях.

Хакеры знают все о жизни своих жертв, включая финансовое состояние, имеющиеся активы и даже номера кредитных карт. С помощью личных данных преступник легко сможет узнать о значительной части вашей цифровой жизни.

Вы, возможно, слышали о том, как преступники установили фальшивый банкомат в торговом центре. Каждый раз, когда кто-то вставлял карту и вводил PIN-код, компьютер фиксировал эти данные, при этом банкомат отказывал в операции. Затем мошенники смогли сделать копии карт с PIN-кодами и беспрепятственно снять деньги уже в настоящих банкоматах.

Во многих ситуациях требуется ввод конфиденциальных данных, таких как пароли или PIN-коды. Если хакер получит доступ к информации, связанной с человеком или его цифровыми устройствами, он легко сможет украсть личность жертвы и тем самым нанести ей серьезный ущерб. Как можно защититься от подобных мошеннических схем?

Один из способов — не раскрывать никаких конфиденциальных данных, к которым относятся имя, фамилия, адрес, номер социального страхования, PIN-код, но это не всегда возможно. Другим подходом является применение протоколов *доказательства с нулевым разглашением* (Zero-Knowledge Protocols, ZKP).

Доказательство с нулевым разглашением — это метод, позволяющий подтвердить обладание определенной информацией, *не раскрывая ее*. Фактически пользователь может идентифицировать себя без ввода пароля, PIN-кода или других

конфиденциальных данных. Эти криптографические протоколы и являются центральной темой главы 5.

Теперь, когда вы имеете общее представление о доказательстве с нулевым разглашением и его практических аспектах, углубимся в анализ связанных с ним ключевых сценариев и криптографических протоколов.

Основной сценарий ZKP — цифровая пещера

Представьте такую ситуацию: Пегги должна доказать Виктору, что она может открыть запертую дверь в *пещере Али-Бабы*. Как видно на рис. 5.1, в пещере есть только одна дверь, к которой можно подойти с двух сторон. Обратите внимание: стандартные имена Алиса и Боб были изменены на Пегги и Виктор, поскольку в данном контексте важно обозначить роль *доказывающего* (*Peggy* — от *P*rover) и *проверяющего* (*Victor* — от *V*erifier).

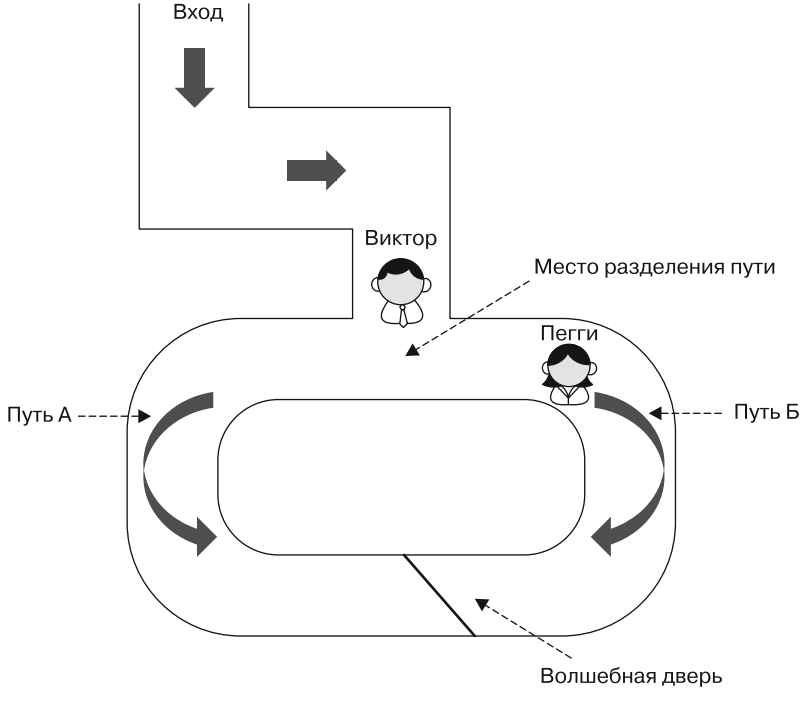


Рис. 5.1. Пещера Али-Бабы

В этом примере представлена современная версия пещеры Али-Бабы, где дверь заперта на надежный электронный замок, который не позволяет войти никому, кто не знает секретного кода.

Предположим, Пегги нужно доказать Виктору, что она знает код от этой двери, не раскрывая его. В этом случае задача Пегги заключается в том, чтобы *не* раскрывать Виктору код, поскольку она не уверена, знает ли он его, и не хочет делиться с ним этой информацией. Ей нужно лишь продемонстрировать, что она может пройти через дверь с другой стороны.

Иными словами, ZKP представляет собой проверку, при которой не раскрывается сам секрет, а лишь доказывается возможность решения задачи с его помощью. Обычный способ доказать знание чего-либо — просто сообщить эту информацию, однако в таком случае можно случайно раскрыть больше, чем необходимо. ZKP позволяет Пегги избежать раскрытия кода, при этом Виктор может убедиться в том, что она действительно его знает, если увидит, как она проходит через дверь.

Существует множество способов реализации ZKP, поскольку такая проверка требуется в большом количестве сценариев. Представьте, что Пегги — человек, а Виктор — какое-либо устройство (банкомат или сервер). Пегги должна подтвердить свою личность, но при этом она не хочет раскрывать личные данные, такие как имя или фамилия. Ей просто нужно доказать устройству, что она является той, за кого себя выдает. В этом случае на помощь приходит ZKP.

Еще один пример — аутентификация виртуальных машин в компьютерной сети. Эту тему мы подробно разберем в главе 8, где будет показано использование ZKP для защиты от атак посредника.

Теперь углубимся в реализацию ZKP. Начнем с анализа неинтерактивных протоколов.

Неинтерактивные протоколы доказательства с нулевым разглашением

В этом разделе мы проанализируем неинтерактивный протокол доказательства с нулевым разглашением. Доказывающий должен продемонстрировать истинность утверждения, предполагая, что проверяющий не знает правильного ответа (содержания утверждения), а проверка осуществляется без какого-либо обмена информацией между ними.

Схема протокола выглядит следующим образом:

доказывающий (утверждение) → [доказательство знания] → проверяющий (проверка).

Рассмотрим следующую задачу: Пегги утверждает, что знает [m], зашифрованное с использованием алгоритма RSA:

$$m^e \equiv c \pmod{N}.$$

Важно помнить, что Пегги не хочет раскрывать содержимое [m].

Здесь (N), (c), (e) — это открытые параметры, а [m] — секретное значение.

Помните, что секретные элементы функции обозначаются квадратными скобками [...].

Чтобы можно было доказать утверждение, выполняется следующий протокол.

1. Пегги выбирает случайное число [r_1], которое хранится в тайне, и умножает его обратное значение (INV [r_1]) на [m] (по модулю N):

$$r_2 \equiv [m] \cdot r_1^{-1} \pmod{N}.$$

2. Не раскрывая значение r_2, она вычисляет (x_1) и (x_2):

$$x_1 \equiv r_1^e \pmod{N}; x_2 \equiv r_2^e \pmod{N}.$$

   Затем она отправляет x_1 и x_2 Виктору.

3. Виктор выполняет проверку:

$$x_1 \cdot x_2 \equiv c \pmod{N}.$$

Положительный результат проверки $x_1 \cdot x_2 \equiv c \pmod{N}$ дает основание полагать, что Пегги действительно известно [m].

Таким образом, Пегги демонстрирует Виктору, что она действительно знает сообщение [m], не раскрывая его содержимого. Причем ей неважно, знает Виктор [m] или нет, так как это не играет никакой роли при использовании данного протокола.

Фактически основная сложность выполнения протокола заключается в том, что Пегги должна решить задачу RSA, не раскрывая [m].

Если Пегги знает [m], скрытое в шифротексте (c), она может вычислить $x_1 \cdot x_2 \equiv c \pmod{N}$.

Важно также учитывать, что если (c) представляет собой большое число, то предполагается, что вычислить x_1 и x_2, два множителя (c), без знания [m] будет так же затруднительно, как и выполнить факторизацию (N).

Исходя из предыдущего уравнения, (r_2) вычисляется с использованием [m]:

$$r_2 \equiv [m] \cdot r_1^{-1} \pmod{N}.$$

Таким образом, произведение $x_1 \cdot x_2$, как видно из последующего доказательства, исключает (r_1), в результате чего остается только $m^e \pmod{N}$, чем и является (c).

Даже если Виктор не знает значения [m], он может верить, что Пегги его знает, потому что она *предоставила множители* (c).

Поскольку алгоритм RSA основан на задаче факторизации, как вы видели в главе 3, здесь применима следующая функция:

$$x_1 \cdot x_2 \equiv c \pmod{N}.$$

Таким образом, можно утверждать, что вычисление двух чисел, x_1 и x_2, являющихся множителями (c), — трудноразрешимая задача.

Позже я докажу, что этот протокол уязвим перед атакой, которая обходит задачу факторизации и позволяет обмануть Виктора.

Числовой пример. Рассмотрим пример применения протокола со следующими числовыми параметрами:

- $m = 88$;
- $N = 2\,430\,101$;
- $e = 9007$;
- $m^e \equiv c \pmod{N}$;
- $88^{9007} \equiv 160\,613 \pmod{2\,430\,101}$.

Сначала Пегги выбирает случайное число $r_1 = 67$.

- *Шаг 1.* Пегги вычисляет r_2:

  $$r_2 \equiv [m] \cdot r_1^{-1} \pmod{N}.$$

  Она умножает [INV (r_1)] — значение, обратное $r_1 \pmod{N}$, на [m]:

  $$67 \cdot x \equiv 1 \pmod{2\,430\,101};$$
  $$x = 217\,621;$$
  $$[m] \cdot x \equiv r_2 \pmod{N};$$
  $$88 \cdot 217\,621 \equiv 2\,139\,941 \pmod{2\,430\,101}.$$

  В результате получает значение $r_2 = 2\,139\,941$.

Далее она вычисляет x_1 и x_2:

$$x_1 \equiv r_1^e \pmod{N};$$

$$x_2 \equiv r_2^e \pmod{N};$$

$$67^{9007} = 1\,587\,671 \pmod{2\,430\,101};$$

$$x_1 = 1\,587\,671;$$

$$2\,139\,941^{9007} \equiv 374\,578 \pmod{2\,430\,101};$$

$$x_2 = 374\,578.$$

Пегги отправляет Виктору значения x_1 и x_2, то есть (1 587 671, 374 578).

- *Шаг 2*. Виктор выполняет проверку:

$$x_1 \cdot x_2 \equiv c \pmod{N};$$

$$1\,577\,671 \cdot 374\,578 \equiv 160\,613 \pmod{2\,430\,101};$$

$$160\,613 = c.$$

Теперь разберемся в математическом аспекте этого протокола.

Демонстрация работы неинтерактивного протокола ZKP

Необходимо продемонстрировать, что:

$$x_1 \cdot x_2 \equiv c \pmod{N},$$

где $c \equiv [m^e] \pmod{N}$. Если в предыдущей функции сделать замены $(x_1 = r_1^e)$ и $(x_2 = r_2^e)$, то мы получим:

$$x_1 \cdot x_2 \equiv r_1^e \cdot r_2^e \equiv c \pmod{N}.$$

Замена r_2 дает следующий результат:

$$x_1 \cdot x_2 \equiv (r_1)^e \cdot (\boldsymbol{m} \cdot (r_1^{-1})^e) \equiv (r_1)^e \cdot m^e \cdot (\mathbf{INV}\ (r_1)^e) \equiv c \pmod{N}.$$

Исходя из свойств модульной степени (собрав вместе два выделенных множителя), мы имеем:

$$r_1^e \cdot \left(\mathrm{INV}\left(r_1\right)\right)^e \equiv 1 \pmod{N}.$$

Таким образом, после сокращения r_1^e и $(\mathrm{INV}\ (r_1))^e$ в итоговом уравнении останется только m^e:

$$x_1 \cdot x_2 \equiv m^e \equiv c \pmod{N}.$$

Как мы уже знаем, (m^e) — это просто результат шифрования секретного сообщения $[m]$ с помощью RSA, то есть криптограмма (c). Вот почему $x_1 \cdot x_2 = c$. Это именно то, что мы хотели продемонстрировать.

У этого протокола есть важная особенность — он выполняется всего в два шага.

1. Пегги вычисляет параметры.
2. Виктор проверяет правильность значений.

Принципиальным моментом является то, что в этом процессе между Пегги и Виктором нет прямого взаимодействия.

В следующем разделе будет показано, как можно атаковать протокол RSA ZKP.

Атака на неинтерактивный протокол RSA ZKP

Если вы дошли до этого момента, я надеюсь, что останетесь со мной и позволите мне продемонстрировать, как использовать протокол для обмана проверяющего.

Обратите внимание: я разработал эту атаку в конце 2018 года. С тех пор появилось еще больше интересных атак на ZKP.

Цель моей атаки — доказать, что Ева (злоумышленник), даже не зная значения $[m]$, может вычислить два *фальшивых* числа (x_1, x_2), которые используются для факторизации (c).

Посмотрим, как работает атака и к какому результату приводит.

1. Ева выбирает случайное число $[r]$ и вычисляет:

$$[r] \cdot (v_1) \equiv e \pmod{N}.$$

   Элемент (e) — это открытый параметр RSA, как мы видели в разделе о RSA в главе 3, то есть значение (e) известно каждому. Из этой функции Ева извлекает (v_1).
2. Параллельно она вычисляет:

$$e \cdot x \equiv c \pmod{N}.$$

3. Параметры (e, c) находятся в открытом доступе. Это уравнение, как и в шаге 1, является обратным умножением (по модулю N). С помощью этой операции Ева получает $[x]$. Затем она умножает $[x]$ на $[r]$ и получает (v_2):

$$x \cdot r \equiv v_2 \pmod{N}.$$

   Ева отправляет (v_1, v_2) Виктору. Он, в свою очередь, выполняет следующую проверку:

$$v_1 \cdot v_2 \equiv c \pmod{N}.$$

Таким образом Ева может выдать себя за Пегги и заявить, что знает $[m]$, даже если это неправда!

Числовой пример. Здесь $r = 39$ — секретное число, выбранное Евой, (N, c, e) — открытые параметры из предыдущего примера (N = 2 430 101, c = 160 613, e = 9007):

- *Шаг 1.* Ева вычисляет v_1:

$$e \cdot r^{-1} \equiv v_1 \pmod{N};$$
$$9007 \cdot 436\,172 = 1\,557\,988 \pmod{2\,430\,101};$$
$$v_1 = 1\,557\,988.$$

  Затем она переходит к вычислению v_2.

  Следующая операция позволяет получить обратную величину (e) за счет (c).

  С помощью обратного модульного умножения $e \cdot x \equiv c \pmod{N}$ Ева получает значение (x):

$$9007 \cdot x = 160\,613 \pmod{2\,430\,101};$$
$$x = \mathbf{2\,031\,892}.$$

  Полученное x позволяет Еве сгенерировать v_2:

$$x \cdot r \equiv v_2 \pmod{N};$$
$$v_2 = 1\,480\,556.$$

  Ева отправляет (v^1 = 1 557 988, v_2 = 1 480 556) Виктору.

- *Шаг 2.* Виктор может выполнить проверку:

$$v_1 \cdot v_2 = c;$$
$$1\,557\,988 \cdot 1\,480\,556 = 160\,613 \pmod{2\,430\,101}.$$

  Атака прошла успешно!

Основным действующим лицом этой атаки могла быть сама Пегги, если бы она не знала [m], но хотела убедить Виктора в обратном.

Успешность атаки обусловлена тем, что [m] является частью (c). В неинтерактивном протоколе не требуется показывать [m] или его хеш [$H(m)$], так как Пегги не должна раскрывать никакой информации об [m] Виктору. Помните, что это не протокол аутентификации, а доказательство знания [m].

В приведенном примере вы могли заметить, что (c) не является результатом произведения двух больших простых чисел, как в случае с N. Таким образом, протокол имеет довольно слабую логическую основу: найти два числа, произведением которых является (c), — не такая уж сложная задача, даже если (c) большое число.

В качестве гипотетического примера можно представить себе сценарий, в котором страна (A) должна продемонстрировать стране (B), что она владеет формулой атомной бомбы. С помощью доказательства с нулевым разглашением страна (A) может заявить, что знает $[m]$, то есть формулу атомной бомбы, хотя на самом деле это ложное утверждение.

При определенных условиях эту атаку можно предотвратить.

Например, если Виктор уже знает $[m]$, он может потребовать от Пегги отправить ему хеш сообщения $H\ [m]$. Сравнив значения хеша и $[m]$, он может проверить корректность параметров (x_1 и x_2).

Однако здесь есть проблема: цель протокола не в том, чтобы доказать что-то уже известное, а в том, чтобы доказать что-то независимо от того, известно это или нет.

Предотвратить подобные проблемы поможет интерактивный протокол, который мы рассмотрим в следующем разделе.

Интерактивный протокол ZKP Шнорра

В предыдущем разделе мы рассмотрели неинтерактивный протокол, где между Пегги и Виктором нет прямого взаимодействия, а есть только *фиксированное обязательство*. Пегги доказывает Виктору, что знает сообщение $[m]$, не раскрывая его содержимого. Таким образом, она доказывает свою честность демонстрацией того, что может решить задачу алгоритма RSA (или другую сложную математическую задачу). Однако мы также увидели, что этот протокол можно *обойти* с помощью математического трюка.

Посмотрим, надежен ли интерактивный ZKP и устойчив ли он к потенциальным атакам.

Основными участниками интерактивного протокола снова выступают Пегги и Виктор. Предположим следующее:

- p — большое простое число;
- g — образующий элемент группы (Z^p);
- $B \equiv g^a \pmod p$ — открытый параметр Пегги;
- (p, g, B) — открытые параметры;
- $[a]$ — секретное число, являющееся объектом обязательства.

Пегги утверждает, что знает $[a]$. Предположим, что $[a]$ — это пароль для доступа к определенной сумме денег. Чтобы подтвердить это, Пегги и Виктор выполняют следующий интерактивный протокол.

- *Шаг 1.* Пегги выбирает случайное целое число $[k]$ так, чтобы $1 \leqslant k < p - 1$. Это основа модульной арифметики.

  Она вычисляет (V):

  $$V \equiv g^k \pmod{p}$$

  и отправляет его Виктору.
- *Шаг 2.* Виктор выбирает случайное целое число (r) так, чтобы $1 \leqslant r < p - 1$, и отправляет его Пегги.
- *Шаг 3.* Пегги вычисляет (w):

  $$w \equiv (k - a \cdot r) \pmod{(p - 1)}$$

  и отправляет его Виктору.
- *Шаг 4.* Виктор выполняет проверку:

  $$V \equiv g^w \cdot B^r \pmod{p}.$$

Положительный результат проверки доказывает Виктору, что Пегги действительно знает $[a]$.

Давайте посмотрим, что обусловливает эффективность протокола и почему последняя функция (V) является доказательством знания $[a]$ (обязательством) Пегги.

Однако прежде всего я покажу, почему протокол работает с математической точки зрения, а затем приведу соответствующий числовой пример.

Демонстрация интерактивного ZKP

Напомню начальные условия:

$$V \equiv g^k \pmod{p};$$
$$B \equiv g^a \pmod{p};$$
$$w \equiv k - a \cdot r \pmod{(p - 1)}.$$

Теперь подставим эти значения в последнюю проверку из шага 3:

$$V \equiv g^w \cdot B^r \pmod{p};$$
$$V \equiv g^k \pmod{p}.$$

Замена функций в (V) меняет уравнение:

$$V \equiv g^{(k - a \cdot r \pmod{p - 1})} \cdot ((g^a)^r) \pmod{p}.$$

Согласно свойствам экспоненциального множителя:

$$g^k \equiv g^{(k - [ar])} \cdot g^{[ar]} \pmod{p};$$
$$V \equiv g^k \equiv g^{(k - ar + ar)}.$$

После сокращения [$-ar$] и [$+ar$] получается:

$$g^k \equiv g^k \pmod{p}.$$

Что и требовалось доказать.

Числовой пример. Пусть:

- $p = 1987$;
- $a = 17$;
- $g = 3$,

где $p = 1987$ и $g = 3$ — это открытые параметры.

Пегги утверждает, что знает секретное число [a] = 17:

$$B \equiv g^a \pmod{p}.$$

Пегги вычисляет свой открытый ключ (B):

$$3^{17} \equiv 1059 \pmod{1987}.$$

Она выбирает случайное число [k] = 67 и вычисляет (V):

$$V = 3^{67} = 1753 \pmod{1987}.$$

И отправляет (V) Виктору.

Виктор выбирает случайное число ($r = 37$) и отправляет его Пегги, которая может выполнить следующие вычисления:

$$k - a \cdot r \equiv w \pmod{p - 1};$$
$$67 - 17 \cdot 37 \equiv 1424 \pmod{1987 - 1};$$
$$w = 1424.$$

Пегги отправляет ($w \equiv 1424$) Виктору.

Наконец, Виктор проверяет, соответствует ли (V) = 1753 результату следующей операции:

$$g^w \cdot B^r \equiv V \pmod{p};$$
$$3^{1424} \cdot 1059^{37} \equiv 1753 \pmod{1984};$$
$$V = 1753.$$

Таким образом, соответствие подтверждено.

Этот протокол можно использовать для *аутентификации*. К примеру, Виктор — это банк, хранящий открытый параметр (B) клиента банка Пегги. Секретное число [a] — это секретный код Пегги, то есть PIN-код. Чтобы получить доступ к своему счету, Пегги должна подтвердить, что знает [a].

В качестве другого примера может выступать такой сценарий использования, в котором Виктор — это центральное вычислительное устройство (сервер), а Пегги — пользователь, который хочет подключиться к серверу по незащищенной линии, или другой сервер.

Смысл использования ZKP заключается в том, чтобы помочь Пегги сохранить свои конфиденциальные данные в тайне. Поэтому она должна продемонстрировать Виктору, что знает дискретный логарифм (B).

Как вы видели в главе 3, знания (B) и (g) недостаточно для вычисления $[a]$ в уравнении:

$$B \equiv g^{[a]} \pmod{p}.$$

Это обусловлено тем, что мы работаем в области модульной арифметики.

Для решения проверочного уравнения (w) Пегги необходимо знать $[a]$:

$$w \equiv k - \boldsymbol{a} \cdot r \pmod{p-1}.$$

Нет никакого способа обмануть Виктора, так как он посылает параметр (r) Пегги, который используется вместе с (w) и (B) для выполнения проверки. Кроме того, параметр (r) лишает Пегги возможности блефовать:

$$V \equiv g^{w} \cdot \boldsymbol{B}^{r} \pmod{p}.$$

Надеюсь, я убедил вас в том, что в данном случае у Пегги нет возможности обмануть Виктора.

Проверка разрушительной атаки на интерактивный ZKP

Теперь, когда мы убедились, что Пегги не может обмануть Виктора, рассмотрим атаку на этот протокол, которую я разработал в конце 2018 года, и увидим, работает она или нет.

Здесь задача атакующего (Евы) — предоставить доказательство проверки, не зная секретного числа $[a]$.

- *Шаг 1*. Пегги выбирает случайное целое число $[k]$ так, чтобы $1 \leqslant k < p - 1$.

  Затем она вычисляет:

  $$V \equiv g^{[k]} \pmod{p}.$$

  Пегги отправляет (V) Виктору. В этот момент Ева может попытаться провести атаку посредника.

  В этот момент Ева вводит $V_1 \equiv g^{[k_1]} \pmod{p}$, где $[k_1]$ — число, придуманное Евой взамен $[k]$.

- *Шаг 2.* Виктор выбирает случайное целое число (r) так, чтобы $1 \leqslant r < p - 1$. Отправляет его Пегги. А в нашем случае с атакой его получает Ева.
- *Шаг 3.* В корректном протоколе Пегги получает ответ: $w \equiv k - a \cdot r \pmod{(p-1)}$. Но так как сообщение перехватила Ева, которая заменила V на V_1, то ей надо вычислить w_1 таким образом, чтобы $V_1 \equiv g^{w_1} \cdot \boldsymbol{B}^r \pmod{p}$.

Вот так работает атака.

- Атака считается успешной, если можно исключить (r) из конечной функции проверки (V_1).
- Атакующий должен найти значение (w_1):

$$w_1 = [x] \rightarrow V_1 = g^{k_1}.$$

Следует обратить внимание на некоторые моменты.

- Не обязательно вычислять (w_1) так же, как (w). Параметру (w_1) можно придать любое значение.
- Самой первой точкой атаки является подмена (V) на (V_1), но это не обязательно. Поскольку параметр (r) неизвестен при передаче (V_1) Виктору, он уже не может быть изменен.

Удачи! Если вам удастся найти способ обмануть интерактивный протокол Шнорра, пожалуйста, сообщите мне о своих результатах, и вы получите должность криптографа-исследователя.

С представленным интерактивным протоколом связана еще одна проблема. Представим, что два человека живут в разных часовых поясах, например в Европе и Австралии. Если один человек *онлайн*, то другой, скорее всего, *не в сети*, потому что спит. В таком случае этот протокол не особо подходящий способ коммуникации.

Кроме того, протокол Шнорра не очень подходит для совершения криптовалютных транзакций. Большинство криптовалютных протоколов используют внутри своих архитектурных структур алгоритмы доказательства с нулевым разглашением для обезличивания данных.

Теперь, когда мы знаем, как реализовать интерактивный протокол, можем перейти к знакомству с zk-SNARKs.

Однораундовый ZKP

В этом разделе мы рассмотрим малоизвестный ZKP, состоящий всего из одного раунда шифрования, который был представлен исследователями Султаном Альмухаммади и Клиффордом Нейманом из Университета Южной Калифор-

нии. Он предназначен для доказательства знания всего за один раунд. В статье *Security and privacy using one-round zero-knowledge proofs* («Обеспечение безопасности и конфиденциальности с помощью одноразундового доказательства с нулевым разглашением», 2005) говорится следующее: *«Предложенный подход создает новые протоколы, которые позволяют доказывающему подтвердить знание секрета, не раскрывая его»*.

Исследователи доказали, что неинтерактивные протоколы доказательства с нулевым разглашением более эффективны с точки зрения вычислительных и коммуникационных затрат, поскольку экономят время выполнения и уменьшают задержки при передаче данных.

ZKP применяются во многих областях информационных технологий: в приложениях для онлайн-торговли, смарт-картах, электронных деньгах, для обеспечения анонимной связи и электронного голосования. Альмухаммади и Нейман искали способ выполнить ZKP всего за один раунд без использования итеративной математической схемы, влекущей за собой большие затраты вычислительных и коммуникационных ресурсов.

Давайте погрузимся в эту тему глубже и посмотрим, как работает одноразундовый ZKP.

Допустим, Пегги хочет доказать Виктору, что она знает дискретный логарифм (при этом протокол может работать и с другими задачами). Для этого ей нужно показать, что она знает $[x]$:

$$g^{[x]} \equiv b \pmod{p}.$$

Виктор запускает вызов (c), чтобы проверить, действительно ли Пегги знает $[x]$. Он выбирает случайное значение $[y]$ и вычисляет:

$$c \equiv g^{[y]} \pmod{p}.$$

Затем он отправляет (c) Пегги. Она добавляет к нему параметр $[x]$ и вычисляет (r):

$$c^{[x]} \equiv r \pmod{p}.$$

Пегги отправляет (r) Виктору. С его помощью он может выполнить проверку:

$$r \equiv b^{[y]} \pmod{p}.$$

Если (r) соответствует $V = b^{[y]} \pmod{p}$, Виктор принимает доказательство.

Этот простой и незамысловатый протокол чем-то похож на алгоритм Диффи — Хеллмана, о котором мы говорили в главе 3. Он основан на вычислительной сложности дискретного логарифма. Однако для того, чтобы вы лучше поняли суть выполняемых операций, я покажу на числовом примере математическую сторону работы протокола.

Работа протокола с математической точки зрения

Первый вопрос: почему параметры (r) и (V) равны?

Ответ можно найти здесь:

$$r \equiv c^{[x]} \equiv g^{[y][x]} \equiv \boldsymbol{g}^{[y \cdot x]} \left(\mathrm{mod}\ p\right);$$

$$V \equiv b^{[y]} \equiv g^{[x][y]} \equiv \boldsymbol{g}^{[x \cdot y]} \left(\mathrm{mod}\ p\right).$$

Как можно увидеть, $r \equiv V \equiv b^{[y]} \pmod{p}$.

Числовой пример

- $p = 2741$;
- $g = 7$;
- $x = 88$ — это секретное число, знание которого нужно доказать Пегги.

Утверждение имеет вид:

$$g^x \equiv b \pmod{p};$$

$$7^{88} \equiv 1095 \pmod{2741};$$

$$b = 1095.$$

Виктор выбирает случайное число $y = 67$.

Затем он выполняет следующие вычисления:

$$g^y \equiv c \pmod{p};$$

$$7^{67} \equiv 1298 \pmod{2741};$$

$$c = \mathbf{1298}.$$

Получив (c), Пегги вычисляет:

$$c^x \equiv r \pmod{p};$$

$$1298^{88} \equiv 361 \pmod{2741};$$

$$r = \mathbf{361}.$$

Она отправляет (r) Виктору, который, в свою очередь, выполняет проверку:

$$b^y \equiv V \pmod{p};$$

$$1095^{67} \equiv 361 \pmod{2741};$$

$$\boldsymbol{V} = \mathbf{361} = \boldsymbol{r}.$$

Таким образом, мы показали работу однораундового ZKP на числовом примере. В следующем разделе рассмотрим детали, которые еще нагляднее демонстрируют сильное сходство этого протокола с алгоритмом Диффи — Хеллмана.

Детали одноraундового протокола

Вы могли отметить сходство одноraундового протокола с обменом ключами Диффи — Хеллмана. Это не случайное совпадение. Несмотря на различие целей, которые преследуют эти алгоритмы, у них много общего. Давайте посмотрим, чем именно они похожи.

Потом мы рассмотрим эффективность одноraундового протокола. Всего за два шага Пегги может доказать Виктору корректность утверждения $[x]$.

Воссоздадим одноraундовый протокол.

- *Шаг 1*. Пегги вычисляет:

  $$g^x \equiv b \pmod{p}.$$

  Так же как в алгоритме Диффи — Хеллмана:

  $$g^a \equiv A \pmod{p}.$$

- *Шаг 2*. Виктор вычисляет:

  $$g^y \equiv c \pmod{p}.$$

  Так же как в алгоритме Диффи — Хеллмана:

  $$g^b \equiv B \pmod{p}.$$

- *Шаг 3*. Пегги вычисляет:

  $$c^x \equiv r \pmod{p}.$$

  В алгоритме Диффи — Хеллмана это значение соответствует общему ключу H:

  $$B^a \equiv H \pmod{p}.$$

- *Шаг 4*. Виктор вычисляет:

  $$b^y \equiv r \pmod{p}.$$

  В алгоритме Диффи — Хеллмана это значение тоже соответствует общему ключу H:

  $$A^b \equiv H \pmod{p}.$$

Таким образом, $[H]$ является общим закрытым ключом, который Алиса и Боб (в нашем случае Пегги и Виктор) используют в алгоритме Диффи — Хеллмана. Здесь $[r = H]$ служит для Виктора доказательством.

Так, можно утверждать, что протокол Султана и Клиффорда действительно идентичен алгоритму Диффи — Хеллмана, который был рассмотрен в главе 3.

Благодаря одноraундовому протоколу Пегги может доказать, что знает значение $[x]$. Виктор может быть уверен в этом, даже если сам не знает значение $[x]$.

Наилучшее представление о необходимых шагах дает упрощенная версия протокола, представленная далее. По сути, их всего два:

- инициализированы параметры g, b, p и x для Пегги;
- Виктор генерирует случайное значение y.

Итак, рассмотрим эту версию протокола.

- *Шаг 1*. Виктор посылает Пегги результат следующего вычисления:

$$c \equiv g^y \pmod{p}.$$

- *Шаг 2*. Пегги посылает Виктору результат следующего вычисления:

$$r \equiv c^x \pmod{p}.$$

- Далее Виктор сразу может выполнить проверку:

$$r \equiv b^y \pmod{p}.$$

Как видите, для выполнения этого протокола и проверки утверждения [x] через подтверждение параметра (r) Виктором требуется всего два шага.

Одноraундовый протокол вдохновил меня на создание собственного протокола, который мы рассмотрим в следующем разделе. Мои исследования позволили сократить количество шагов до одного.

Введение в протоколы zk-SNARK — мистическая математика

Мы переходим к новому типу протоколов — *zk-SNARK*. Это разновидность неинтерактивных ZKP, которые считаются сложными и часто относятся к *мистической математике*. В следующем разделе рассмотрим интересные возможные атаки на эти протоколы.

В этом же разделе сосредоточимся на протоколах zk-SNARK. *Неинтерактивные доказательства с нулевым разглашением* (zk-SNARK или zk-STARK) — это ZKP, которые не требуют прямого взаимодействия между доказывающим и проверяющим, что роднит его с первым протоколом, рассмотренным в этой главе.

Аббревиатура zk-SNARK расшифровывается как *Zero-Knowledge Succinct Non-Interactive Argument of Knowledge* (краткий неинтерактивный аргумент знания с нулевым разглашением). Эти схемы требуют всего одного взаимодействия между доказывающим и проверяющим.

Протоколы zk-SNARK высоко ценятся за свою способность обезличивать транзакции и идентифицировать пользователей в криптовалютных операциях.

Кроме того, они используются в блокчейне в качестве инструмента аутентификации для создания *консенсуса*.

В дальнейшем вы увидите, что применение zk-SNARK в блокчейне особенно важно для функционирования интерактивных контрактов — криптовалютных эскроу-счетов, доступ к которым открывается после выполнения определенного условия.

Несмотря на то что интерактивные контракты и блокчейн выходят за рамки этой книги, на их примере я хочу показать применение zk-SNARK в криптовалютной области.

Предположим, Пегги осуществляет платеж в Ethereum для выполнения интерактивного контракта, заключенного с Виктором. Они оба хотят быть уверены в успешности выполнения обязательств контракта (со стороны Пегги) и получения оплаты (со стороны Виктора). При этом детали контракта остаются скрытыми. Здесь zk-SNARK играет ключевую роль в обеспечении секретности и исполнения интерактивного контракта. Таким образом, он должен быть быстрым, безопасным и простым в применении.

Как мы уже обсуждали, цель ZKP заключается в упрощении взаимодействия в ненадежной среде. В данном случае речь идет о блокчейне и виртуальных платежах, но суть процесса остается неизменной.

В блокчейне zk-SNARK скрывает условия интерактивного контракта и одновременно доказывает их выполнение, тем самым обеспечивая конфиденциальность пользователей и компаний.

Однако следует помнить (с точки зрения не крайнего скептицизма, но реализма), что это верно только при определенных условиях, которые можно сформулировать так.

- Доказательство, представленное доказывающим, должно обладать такой же вычислительной сложностью, что и алгоритм, выбранный в качестве основы доказательства.
- Не должно существовать математических способов обмануть проверяющего с помощью поддельных доказательств или упрощенных вычислений (в качестве примера можно привести подмену параметров в уравнении $V_1 \equiv g^{(v_1)} \cdot B^r \pmod{p}$, которая позволяет не знать $[a]$).

Посмотрим, как работает протокол zk-SNARK.

Как работает протокол zk-SNARK

В начале этого подраздела я покажу работу протоколов zk-SNARK в общем, а затем отдельно рассмотрим их применение в доказательстве знания на основе дискретного логарифма.

Как вы уже видели в других ZKP, zk-SNARK состоит из трех частей, или элементов: (G), (P) и (V).

- G — это образующий элемент (generator) ключей, состоящий из секретного параметра (утверждения или другого случайного ключа), который генерирует открытые параметры на основе закрытых ключей.
- P — это алгоритм доказательства (proof), в котором указывается, что именно хочет продемонстрировать доказывающий.
- V — это алгоритм проверки (verification), который возвращает проверяющему булеву переменную ИСТИНА или ЛОЖЬ. Также я покажу, что ZKP и, в частности, протоколов zk-SNARK *недостаточно*, чтобы сохранить значение [w] в тайне, но можно прийти к доказательству истинности утверждения, если оно также является ложным.

Давайте изучим пример протокола, похожий на тот, что мы рассматривали в разделе «Интерактивный протокол ZKP Шнорра», и разберем его работу в неинтерактивном режиме (zk-SNARK).

Здесь Анна выступает в роли доказывающего, а Карл — в роли проверяющего.

Анна должна доказать, что ей известно значение [a].

- Она вычисляет свой открытый ключ (y):

$$y \equiv g^a \pmod{p}.$$

  (g) — образующий элемент, как в алгоритме Диффи — Хеллмана и других алгоритмах с закрытым и открытым ключами, которые мы уже видели в этой книге.
- Анна выбирает случайную величину [v] в пределах множества $p - 1$, которую держит в секрете, и вычисляет:

$$t \equiv g^v \pmod{p}.$$

- Затем она вычисляет (c) как хеш-функцию трех параметров (g, y, t), что дает ей возможность получить (r):

$$r \equiv v - c \cdot a \pmod{p - 1}.$$

- Карл может выполнить следующую проверку:

$$t \equiv g^r \cdot y^c \pmod{p}.$$

- Наконец, если равенство выполняется, то Карл подтверждает, что утверждение Анны [a] является ИСТИННЫМ.

Теперь, когда мы проанализировали, как этот ZKP работает в среде zk-SNARK, рассмотрим атаку на этот протокол, а затем числовой пример.

Демонстрация атаки на протокол zk-SNARK

В июне 2019 года я провел атаку, которая показывает, что ничего нельзя считать абсолютно безопасным. На практике успешность этой атаки зависит от защищенности информации, поэтому здесь я просто покажу, как ее можно осуществить с математической точки зрения.

Предположим, Ева — это сервер, который перехватывает открытую хеш-функцию $H(g, y, t)$ и осуществляет атаку посредника.

В момент, когда Анна отправляет (V) Карлу, Ева меняет $(c) = H(g, y, t)$ на $(c_1) = H_1(g, y, t_1)$, где (H) — это хеш-функция, а (t_1) задается как:

$$t_1 \equiv g^{v_1} \pmod{p}.$$

Как вы, вероятно, заметили, замена (v) на (v_1) — это тот же прием, что и замена (k) на (k_1) в атаке, описанной ранее.

Проще говоря, Ева может ввести (r_1) как:

$$r_1 = v_1.$$

Теперь Ева приказывает третьему серверу, подключенному к Интернету, отправить значения (r_1, v_1, c_1) Карлу для следующей проверки:

$$t_1 \equiv g^{r_1} \cdot y^{c_1} \pmod{p}.$$

Легко подтвердить, что $t_1 = g^{v_1}$, так как:

$$y^{(p-1)} \equiv 1 \pmod{p}.$$

Поскольку $c_1 = p - 1$ и $r_1 = v_1$, конечный результат будет выглядеть следующим образом:

$$t_1 \equiv g^{v_1} \equiv g^{v_1} \cdot 1 \pmod{p}.$$

Числовой пример. Дано:

- $p = 3571$;
- $g = 7$;
- $x = 23$.

Открытый ключ Анны вычисляется следующим образом:

$$y \equiv g^x \pmod{p};$$
$$7^{23} = 907 \pmod{3571};$$
$$y = \mathbf{907}.$$

Теперь рассмотрим, как Анна может продемонстрировать Карлу вычисление секретного числа $[x]$.

Она выбирает $v = 67$ и вычисляет:

$$t \equiv g^v \pmod{p};$$
$$7^{67} = 584 \pmod{3571};$$
$$t = \mathbf{584}.$$

Предположим, что хеш (g, y, t) равен $c = \mathbf{37}$.

Анна вычисляет r:

$$r \equiv v - c \cdot x \pmod{p - 1};$$
$$(67 - 37 \cdot 23) \equiv 2786 \pmod{3570 - 1};$$
$$r = \mathbf{2786}.$$

Затем она отправляет $(r, t, c) = (2786, 584, 37)$ Карлу, который, в свою очередь, может выполнить проверку:

$$g^r \cdot y^c \equiv t \pmod{p - 1};$$
$$7^{2786} \cdot 907^{37} = 584 \pmod{3571}.$$

Однако Ева перехватывает открытые параметры (y), (t) и (r) и осуществляет атаку посредника, оставив (y) неизменным, но заменив (t) на (t_1) и (r) на (r_1):

$$v_1 = 57.$$

Ева выполняет следующие вычисления:

$$t_1 \equiv 7^{57} \pmod{3571};$$
$$\boldsymbol{t_1} = \mathbf{712};$$
$$r_1 = v_1 = 57;$$
$$c_1 = p - 1 = 3570.$$

Она отправляет $(r_1, t_1, c_1) = (57, 712, 3570)$ Карлу.

Карл выполняет проверку:

$$\boldsymbol{t_1} \equiv g^{r_1} \cdot y^{c_1} \pmod{p}.$$

Я выделил параметр, который подменила Ева, — (t_1, r_1, c_1). При этом она оставила неизменными (y, g, p). Подставив новые параметры в уравнение проверки, мы получили следующий результат:

$$7^{57} \cdot 907^{3570} \equiv 712 \pmod{3571}.$$

Карл может проверить, что $t_1 = 712$ соответствует параметрам, полученным от Евы.

По сути, если Карл не может определить, что $r_1 = v_1$, или не принимает $c = p - 1$, то атака считается успешной и Ева может выдать себя за Анну.

Как же защититься от этой атаки? Если ее реализация будет технически более сложной, то предотвратить ее, вероятно, будет очень тяжело.

Обратите внимание на то, что во время атаки открытый ключ Анны (y), который «заворачивает» закрытый ключ [a] объекта утверждения, не был изменен.

В любом случае протоколы zk-SNARK могут быть реализованы с использованием других методов и протоколов для доказательства утверждений. Что это за алгоритмы и протоколы, мы увидим в следующем разделе. Область блокчейна и криптовалют быстро развивается в поисках новых методов анонимной аутентификации пользователей. Но поскольку эта тема относительно новая, лучше сосредоточиться на том, чтобы найти все возможные атаки и методы их устранения.

ZK13 — протокол ZKP для аутентификации и обмена ключами

Теперь мы рассмотрим протокол *ZK13*, который я запатентовал в 2013 году. Это неинтерактивный протокол, решающий важную задачу в проекте системы защищенного поиска (Crypto Search Engine, CSE), который подробно описан в главе 8. Основная задача, которую он решает, заключается в обеспечении аутентификации без использования открытого ключа.

В этом разделе мы разберем ZKP, используемый для аутентификации, которую можно назвать *аутентификацией с нулевым разглашением*. Представьте, что Алиса и Боб хотят передать друг другу общий секрет, известный только им. Пусть этот секрет — количество птиц, замеченных на берегу озера сегодня. Это число известно только Алисе и Бобу до тех пор, пока они сами об этом никому не рассказали. Мы можем рассматривать это количество как общий секретный ключ, который не нужно передавать, так как он уже известен обеим сторонам. Таким образом, кроме задачи аутентификации, возникает также задача проверки закрытого, *предварительно выданного ключа* (pre-shared key, PSK). В протоколе ZK13 Алиса просит Боба использовать этот секрет в качестве пароля — [*закрытого ключа*]. Главное отличие ZK13 от алгоритма Диффи — Хеллмана заключается в том, что секретный ключ *фактически* не передается, так как обе стороны его знают и только проверяют друг друга.

Протокол ZK13 может применяться сразу для нескольких целей. В данном разделе рассмотрим только аутентификацию. Позже в книге мы разберем, как использовать ZKP для обмена общим закрытым ключом.

В 2013 году, разрабатывая архитектуру CSE, о которой будет подробно рассказано в главе 8, я столкнулся с одной из самых сложных проблем — поиском

криптографического метода идентификации сети *виртуальной машины* (virtual machine, VM), который можно было бы использовать вместе с алгоритмом симметричного шифрования.

Как было рассказано в главе 2, симметричные алгоритмы не поддерживают цифровую подпись для аутентификации, так как не используют открытые ключи. На первый взгляд, найти другой метод аутентификации нелегко, однако это возможно, если процесс начинается с общего секрета. Основной целью было предотвращение атак посредника на сеть со стороны внешней враждебной VM.

Чтобы преодолеть все эти сложности, я задумался о реализации нового протокола ZKP. Ознакомившись с наиболее популярными ZKP, сначала выбрал протокол Шнорра (представлен ранее в этой главе). Однако в моем случае *интерактивный протокол* не очень подходил, так как требовал большого количества шагов между доказывающим и проверяющим, что приводило к задержкам при обмене данными. Поэтому я решил реализовать собственный неинтерактивный протокол доказательства с нулевым разглашением.

В результате долгих исследований и применения изобретательности появился ZK13. Перед тем как перейти к анализу этого протокола, я объясню, каким требованиям он должен был отвечать.

- Секретный общий ключ (вызов) должен быть встроен в базу данных VM. Это позволяет *внедрить* секретный параметр [s] в обе VM без обмена ключами с помощью асимметричного алгоритма.
- Цель ZK13 — предоставить участникам коммуникации возможность идентифицировать друг друга, обмениваясь лишь небольшим объемом конфиденциальной информации. Например, вместо передачи самого сообщения [m] осуществлять обмен хешем $H(m)$. Важно отметить, что чем больше объем обмениваемой информации, тем выше уязвимость протокола к потенциальным атакам.
- ZK13 должен был быть простым и, как я уже говорил, *неинтерактивным* протоколом, для работы которого проверяющему требуется лишь одна порция информации. На это есть две причины. Первая — необходимо избегать избыточного обмена информацией (о чем говорилось в предыдущем пункте), так как это может нарушить безопасность системы. Вторая причина связана с целью приложения: платформа CSE предназначена для поиска и извлечения зашифрованных данных с помощью облака или внешних серверов. Поскольку для поисковой системы важна высокая скорость работы, обработка запросов и выдача ответов должны занимать минимально возможное время. Таким образом, особенно важно предотвратить задержку на этапе аутентификации.

- Еще одним условием создания ZK13 было использование лучших и наиболее безопасных методов аутентификации. В то время, когда он был задуман (2011–2013 годы), эра квантовых вычислений еще не представляла никакой опасности для криптографии. Поэтому система опиралась на трудноразрешимую задачу дискретного логарифма.

Объяснение ZK13

Протокол ZK13 подразумевает только одну процедуру передачи данных и использование общего секрета, что можно представить следующим образом (рис. 5.2).

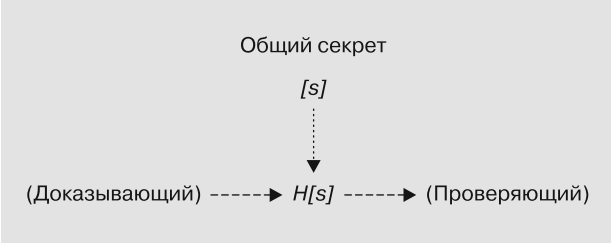


Рис. 5.2. Схема использования хеша общего секрета [*s*]

Перед тем как Боб (VM-1) отправит Алисе (VM-2) набор зашифрованных файлов с помощью системы CSE, он должен доказать ей, что знает секрет [*s*]. Обратите внимание на то, что знание [*s*] уже встроено в «мозги» обеих виртуальных машин.

Боб выбирает случайное число [*k*] (образующий элемент (*G*) протокола zk-SNARK или генератор случайных ключей). Это число генерируется и уничтожается в каждой сессии.

- Открытые параметры:
  - p — простое число;
  - g — образующий элемент.
- Инициализация ключа:
  - [k] — случайное число, выбранное Бобом.
- Секретные параметры:
  - [s] — общий секрет;
  - H[s] — хеш от секрета [s].

- *Шаг 1а*. Боб вычисляет значение (r):

$$r \equiv g^k \pmod{p}.$$

  Общий секрет — это [s], но фактически VM работает с хеш-функциями $H[s]$.

- *Шаг 1б*. Боб вычисляет секретный параметр [F], меняющийся в каждой сессии:

$$\boldsymbol{H[s]} \cdot \boldsymbol{k} \equiv \boldsymbol{F} \pmod{p-1}.$$

  Далее посредством возведения (g) в степень [F] он задает второй элемент zk-SNARK (P):

$$g^F \equiv P \pmod{p}.$$

  Боб отправляет пару (P, r) Алисе.

- *Шаг 2*. Этап проверки (V) подтверждает достоверность доказательства (P) на основе функции:

$$[s] \rightarrow H[s];$$
$$r^{|H[s]|} \equiv g^F \pmod{p},$$

  если:

$$V \equiv r^{|H[s]|} = P \pmod{p}.$$

  Алиса выполняет проверку на основе созданного хеша $H[s]$. Она принимает аутентификацию, только если $V = P$.

  В этом случае, как мы и предполагали в начальных условиях, значение [s] должно быть ей известно.

Как видите, выполнение ZK13 подразумевает всего два шага, но проверяющий, в данном случае Алиса, должен знать секрет [s], иначе проверить доказательство невозможно.

Числовой пример. Открытые параметры:

- $p = 2741$;
- $g = 7$.

Секретные параметры:

- $H[s] = 88$;
- $k = 23$.

Отсюда:

$$g^k \equiv r \pmod{p};$$
$$7^{23} \equiv 2379 \pmod{2741};$$
$$r = 2379.$$

Боб вычисляет [*F*], а затем (*P*):

$$H[s] \cdot k \equiv F \pmod{p-1};$$
$$88 \cdot 23 \equiv 2024 \pmod{2741-1};$$
$$F = \mathbf{2024};$$
$$g^F \equiv P \pmod{2741};$$
$$7^{2024} \equiv 132 \pmod{2741};$$
$$P = \mathbf{132}.$$

Алиса выполняет проверку:

$$r^{[H[s]]} \equiv P \pmod{p};$$
$$2379^{88} \equiv 132 \pmod{2741}.$$

Она еще раз проверяет, что $H[s_A] = H[s_B]$. Если это утверждение ИСТИННО, то Боб действительно знает секрет [*s*]. Теперь, когда мы подтвердили работоспособность ZK13 на числовом примере, я хотел бы продемонстрировать его математические детали.

Демонстрация протокола ZK13

Если $P \equiv g^F \pmod{p}$, нам нужно доказать, что:

$$P \equiv g^F \equiv r^s \pmod{p}.$$

Здесь я специально использую [*s*] вместо *H[s]*.

Поскольку $r \equiv g^k \pmod{p}$, в результате подстановки (*r*) в предыдущее уравнение мы получаем:

$$P \equiv g^F \equiv (g^k)^s \pmod{p}.$$

Мы также знаем, что:

$$F \equiv s \cdot k \pmod{p-1}.$$

Таким образом, согласно свойствам модульной степени:

$$P \equiv g^{s \cdot k} \equiv (g^k)^s \equiv g^{k \cdot s} \pmod{p}.$$

По сути, я заменил параметр (*P*) — доказательство, созданное Бобом, — элементами самого параметра, продемонстрировав, что секрет [*s*] содержится внутри (*P*). Параметр (*P*) должен совпадать с *эфемерным* параметром $(r)^{[s]}$, который Боб создает и отправляет Алисе вместе с доказательством (*P*). Если Алиса знает [*s*], то она может быть уверена в том, что Боб тоже знает [*s*], который является частью (*P*). Именно это мы и хотели продемонстрировать.

Особенности протокола ZK13

Я думаю, вы согласитесь со мной: протокол ZK13 позволяет определить доказательство знания секрета [s] всего за одну процедуру передачи данных.

При объяснении алгоритма я разделил этот процесс на три шага. На самом деле он состоит из двух шагов (с одной процедурой передачи данных), поскольку операции генерирования ключа (G) выполняются в автономном режиме. Поэтому шаги 1а и 1б можно объединить.

Возможные атаки на ZK13

Допустим, Ева (злоумышленник) хочет выдать себя за Алису или Боба, выполнив атаку посредника. Чтобы сделать это, она заменяет (r) на (r_1), который вычисляет с помощью поддельного (k_1):

$$r_1 \equiv g^{k_1} \pmod{p}.$$

Однако при вычислении [F] Ева сталкивается с проблемой, так как не знает $H[s]$, поскольку предполагается, что [s] останется в тайне. Таким образом, ей не удается осуществить атаку.

Тем не менее она может перехватить (r, P) и воспроизвести эти параметры в следующей сессии, начав так называемую *атаку повторного воспроизведения*. Эта атака предотвратима: поскольку (r) генерируется случайным [k], можно блокировать повторяющееся значение (r).

Итак, мы рассмотрели возможную атаку и способ ее предотвращения.

Резюме

Теперь у вас есть четкое понимание того, что такое протоколы ZKP и для чего они нужны.

В этой главе мы подробно разобрали интерактивные и неинтерактивные протоколы ZKP. Помимо прочего, рассмотрели ZKP на основе алгоритма RSA и оригинальный способ его взлома.

Мы обсудили аутентификацию с помощью интерактивного протокола Шнорра и рассмотрели попытку атаки на него.

Были разобраны протоколы zk-SNARK и связанную с ними мистическую математику, чтобы продемонстрировать сложность некоторых других задач, включая интересный способ атаки на zk-SNARK на основе дискретного логарифма.

Далее мы проанализировали неинтерактивный протокол, основанный на алгоритме Диффи — Хеллмана, вы также познакомились с протоколом ZK13 и использованием общих секретов для аутентификации виртуальных машин.

Наконец, мы рассмотрели применение протоколов zk-SNARK в криптовалютной области, уделив особое внимание их применению для обезличивания транзакций.

Вы познакомились с некоторыми схемами атак на ZKP и увидели, как можно поэкспериментировать с такими протоколами, используя некоторые математические хитрости.

Затронутые в этой главе темы должны помочь вам глубже понять ZKP и ее функции. Многое из того, что будет рассмотрено в последующих главах, пересекается с изложенной здесь информацией. В следующей главе мы проанализируем некоторые алгоритмы с закрытым и открытым ключами, которые были разработаны лично мной.

6 Новые изобретения в криптографии и логические атаки

В главе 5 я рассказал о своем неинтерактивном протоколе ZK13, запатентованном в 2013 году. Эта глава посвящена двум алгоритмам, запатентованным и опубликованным мной в 2008–2012 годах, — MB09 и MBXI. Их названия составлены из моих инициалов и годов разработки. Мы также рассмотрим третий протокол — MBXX, запатентованный в 2020 году, который является объединением MB09 и MBXI с новой концепцией использования в сферах блокчейна и цифровых валют. Кроме того, я представлю новый вид шифрования — легковесное шифрование, предназначенное для защиты данных в IoT и смарт-устройствах.

Мы подробно разберем каждый из этих алгоритмов, начиная с истории создания и заканчивая их схемами работы, сильными и слабыми сторонами.

Сначала рассмотрим алгоритм MB09, который представляет собой вариант алгоритма с открытым и закрытым ключами, используемый в основном для передачи электронных денег. Мы посмотрим на схемы выполнения цифровой подписи MBXI и сравним их с алгоритмами Диффи — Хеллмана и RSA.

В конце обсудим криптографический протокол MBXX, который вполне пригоден для достижения консенсуса в сфере блокчейна.

Некоторые из рассматриваемых алгоритмов связаны с электронными платежами, поэтому мы вкратце затронем такие темы, как блокчейн и криптовалюта.

История создания алгоритма MB09 и блокчейна

Начиная работу над проектом MB09, а затем и MBXI, я не знал, для чего именно они будут нужны. На самом деле я даже не знал, найдут ли они применение в области криптографии или кибербезопасности.

В период с 2007 по 2008 год я решил подать заявку на свой первый патент — систему, которую позже назвал MB09. В то время я изучал проблемы, связанные с электронными платежами, и стремился создать алгоритм, который мог бы эффективно работать в этой среде. Когда я наконец завершил процесс подачи заявки, на дворе был июнь 2009 года. Возможно, тогда лишь немногие знали о новой волне криптовалют, возглавляемой киберпанками и криптоанархистами, которая впоследствии превратилась в одну из самых инновационных технологий XXI века.

Штаб-квартира этой группы находилась в богемном здании своеобразного псевдоотеля на Мишн-стрит в центре Сан-Франциско (рис. 6.1). Именно там недавно сформированное сообщество организовало «дом хакеров», где проходили обсуждения нового технофинансового направления. Когда несколько лет назад я посетил это место, группа уже покинула его, но истории о легендарных мозговых штурмах и встречах сохранили в нем атмосферу таинственности.

Рис. 6.1. «Дом хакеров» на Мишн-стрит, 20, в Сан-Франциско — штаб-квартира биткойна (фотография автора)

В эту группу входил Сатоши Накамото — легендарная фигура, таинственный автор опубликованного в ноябре 2008 года проектного документа *A Peer-to-Peer Electronic Cash System* («Одноранговая электронная денежная система»). Именно этот документ, основанный на более ранней идее криптовалюты b-money Вэй Дая, положил начало новой эре, которая навсегда изменила систему электронных платежей.

В аннотации Сатоши заявил:

> *«...мы предлагаем решение проблемы двойного расходования с помощью одноранговой сети. Сеть отмечает транзакции временными метками, хешируя их в непрерывную цепочку доказательства работы и формируя запись, которую нельзя изменить без повторного выполнения доказательства работы. Самая длинная цепочка служит не только доказательством последовательности произошедших событий, но и подтверждением того, что она была сформирована крупнейшим пулом вычислительной мощности».*

Хотя в этом документе ни разу не употребляется термин «блокчейн», выражение «самая длинная цепочка» подразумевает именно его.

Этот документ оказался преддверием эры криптовалют, кульминацией которой стало появление и повсеместное использование одной из самых революционных виртуальных валют — биткойна (Bitcoin). В наши дни вряд ли найдется человек, который не слышал бы о биткойне. Возможно, кто-то немного знает даже о Zcash и множестве других виртуальных монет, выпущенных в период *первичного предложения монет* (Initial Coin Offering, ICO). ICO, широко распространившееся с 2016 года, представляет собой аналог IPO (Initial Public Offering — *первичное публичное предложение акций*), но в криптовалютном формате.

Эта глава посвящена алгоритмам, связанным с безопасными и анонимными платежами, созданным с 2008 по 2012 год. Важно понимать контекст, в котором возникла идея нового свободного подхода к деньгам. Также необходимо разобраться в лежащей в основе этой концепции технологии — блокчейне, ставшем центральным элементом электронных платежных систем. В настоящее время блокчейн находит применение в финтехе, искусственном интеллекте, здравоохранении и других сферах.

Однако для полного понимания происходящего нам нужно вернуться на несколько лет назад. В главе 4 я рассказывал о Дэвиде Чауме, который разработал слепые подписи в 1980-х годах. Система слепых подписей обезличила этот вид электронных платежей, что, безусловно, можно считать первой попыткой создать электронные деньги.

В среду, 17 сентября 2008 года, рухнул финансовый гигант Lehman Brothers[1], и волна банкротств и долговых обязательств захлестнула многие банки. Именно этот системный финансовый кризис, известный как *ипотечный кризис в США*, стал основой для новой волны криптовалют. Крах традиционной финансовой

---

[1] Lehman Brothers объявил себя банкротом 15 сентября; 17-го числа последствия захлестнули рынок.

системы и глубокая неопределенность в обществе подтолкнули Сатоши Накамото к переосмыслению создания, передачи и использования денег. В своем документе он предложил три принципиальных столпа создания идеальной валюты: *отсутствие правительства*, *отсутствие банков* и *отсутствие доверенных третьих сторон*. Третьего января 2009 года, когда Сатоши создал *первый нулевой блок* биткойна, он встроил в *блокчейн* сообщение рядом с номерами, сгенерированными первым хешем цепочки. Сообщение гласило: *Chancellor on brink of second bailout for banks* («Канцлер готов во второй раз спасти банки»), отражая всю его ненависть к банковской системе.

На рис. 6.2 можно увидеть, как выглядел нулевой блок биткойна.

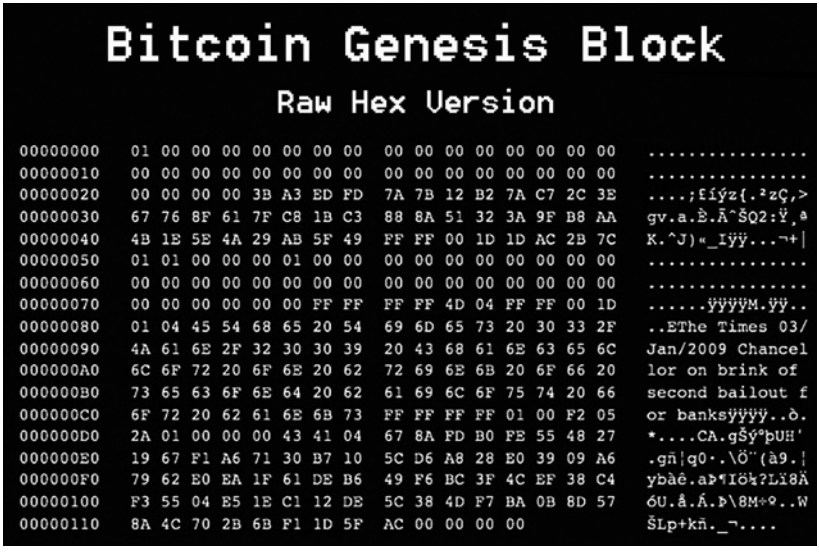


Рис. 6.2. Нулевой блок Сатоши Накамото

Здесь мы не будем детально разбирать процесс обработки транзакций в блокчейне биткойна. Вместо этого сосредоточимся на криптографических алгоритмах, лежащих в его основе. Главная сложность, с которой столкнулся Сатоши Накамото, заключалась в поиске метода проверки соответствия при передаче цифровой валюты, то есть в решении *проблемы консенсуса*. Позже в этой главе я вернусь к концепции консенсуса и покажу, что в определенных условиях можно обойтись без него, используя исключительно криптографические алгоритмы. Я расскажу о том, что блокчейн избыточен для обработки цифровых транзакций, так как все можно сделать с помощью одной криптографии.

На разработку алгоритмов шифрования с открытым и закрытым ключами меня подвигли две причины. Первая была связана с платежным методом *M-PESA*, с которым я познакомился во время своих исследований в 2007–2008 годах. Вторая — со стремлением бросить вызов устоявшимся стандартам.

Остановимся на первой причине. Возможно, многие не знают, что представляет собой система M-PESA и как она работает. Это довольно простой платежный метод, поэтому я не буду углубляться в его детали. Здесь более важен следующий вопрос: почему две трети людей, рожденных в Африке (более 1,2 миллиарда), используют M-PESA для передачи денег через мобильные телефоны? Посредством M-PESA мигранты, находящиеся в других странах, отправляют на родину почти 500 миллиардов долларов.

Ответ прост: причины такой ситуации — низкие комиссии за транзакции по сравнению с более высокими тарифами MoneyGram и Western Union и простота использования приложения на телефоне. Известно, что в Африке мобильных телефонов больше, чем автомобилей, домов или банковских счетов. В будущем Африка станет огромным цифровым рынком финансовых технологий, поскольку невозможно создать достаточное количество банковских отделений или банкоматов, чтобы покрыть огромную территорию континента. Таким образом, мобильный телефон остается наиболее распространенным способом оплаты и перевода средств. Это явление настолько привлекло мое внимание, что я посвятил несколько месяцев изучению платежных систем, адаптированных для мобильных устройств.

В то время крупная телекоммуникационная компания Vodafone владела 40 % акций крупнейшей телекоммуникационной компании Кении Safaricom. В 2007 году Safaricom запустила экспериментальную программу, позволяющую пользователям отправлять деньги через мобильный телефон. Vodafone внедрила этот продукт сначала в Кении, а затем в Танзании, ЮАР, Мозамбике, Египте, Индии, Румынии и на Фиджи.

В мае 2011 года, после создания первой криптографической системы электронных платежей и подтверждения ее работоспособности, я подписал контракт с упомянутой крупной телекоммуникационной компанией. Целью контракта было изучение возможности реализации платформы для электронных платежей. Эта компания также привлекла меня и мою команду к исследовательскому проекту по разработке нового алгоритма, который впоследствии стал системой MBXI. Теперь, спустя годы, я могу сказать, что платежная система, которую я разработал, внедрил и протестировал вместе со своими коллегами Тицианой Ланди и Алессандро Пассерини, стала одним из самых значимых достижений в моей карьере.

Далее я раскрою суть первого из двух разработанных алгоритмов — MB09. Этот метод основан на последней теореме Ферма и переработан специально для криптографических целей. Хотя платежная система MB09 так и не стала

стандартом, у нее есть интересные свойства, знание которых может быть полезно для вашего развития в области криптографии. Затем мы рассмотрим MBXI вместе с интересной атакой на RSA.

Введение в алгоритм MB09 и доказательство последней теоремы Ферма

Сначала рассмотрим некоторые детали алгоритма и вспомним о последней теореме Ферма. Изначально я представил MB09 как алгоритм шифрования, но на самом деле он больше подходит *для электронных платежей*. До того как технологии блокчейна и криптовалют стали широко известны, MB09 разрабатывался как метод шифрования и расшифрования для обмена сообщениями между двумя участниками. Спустя некоторое время я доработал его, превратив в *полностью гомоморфный алгоритм шифрования* MB23. В конечном счете в 2020 году на его основе была создана новая версия, получившая название MBXX. Основной целью MBXX было преодоление проблемы консенсуса, на которую указал Сатоши Накамото, и связанных с ней технологий блокчейна биткойна.

Давайте посмотрим, как работала первая версия алгоритма MB09.

1. Для этого вспомним последнюю теорему Ферма:

$$a^n + b^n = z^n.$$

   Здесь показатели степени n представляют собой любые множества положительных целых чисел.

   Как нам уже известно, это уравнение невозможно выполнить ни при каких значениях степени n, кроме $n = 2$, что приводит к известной теореме Пифагора:

$$3^2 + 4^2 = 5^2.$$

2. Углубившись в анализ этого одновременно простого и сложного уравнения, мы можем обнаружить, что в модульном представлении, если заменить показатели степени n простыми числами p, согласно свойствам модульной арифметики, уравнение будет иметь следующий вид:

$$a^p + b^p \equiv z^p \pmod{p}$$

   для всех $p > 2$.

   Проще говоря, это уравнение показывает, что сумма $a^p + b^p$ всегда равна z^p для всех простых чисел $p > 2$. Доказательство его выполнения заключается в следующем.

   - Малая теорема Ферма утверждает, что $a^p \equiv a \pmod{p}$, $b^p \equiv b \pmod{p}$ и так далее до z. Здесь простое число p — это один и тот же параметр как в степени, так и в модуле.

- Согласно основному свойству сложения:

$$a + b = z.$$

- Если поменять a на $a^p \pmod p$, b на $b^p \pmod p$, а z на $z^p \pmod p$, то выражение $a^p + b^p \equiv z^p \pmod p$ всегда выполняется.

3. Таким образом, можно заключить, что, в отличие от линейного представления, для всех показателей степени (p) сумма $(a + b)^p$ всегда выполняется и равна z^p. Другими словами:

$$a^p + b^p \equiv z \pmod p.$$

В качестве примера можно привести модульное сложение:

$$3^{17} + 5^{17} \equiv 8 \pmod{17}.$$

Таким образом, можно переписать уравнение из шага 2 в следующей форме:

$$a^p + b^p \equiv (a + b)^p \pmod p.$$

Свойство модульного сложения позволяет записать это уравнение в упрощенной форме:

$$a + b \equiv z \pmod p.$$

Это означает, что если взять простое число (p) как показатель степени, а также использовать его в качестве модуля, то предыдущее модульное уравнение приобретает линейный вид, где $(a + b)$ всегда равно его сумме (z).

Сейчас я попытаюсь привести доказательство сказанного.

1. На шаге 3 мы получили уравнение $a^p + b^p \equiv z \pmod p$.
2. На шаге 2 показали (согласно малой теореме Ферма), что уравнение всегда выполняется для всех $p > 2$.

   Например, если $p = 11$, то:

$$3^{11} + 5^{11} \equiv 8 \pmod{11}.$$

3. Поскольку сумма $a^p + b^p$ всегда равна $z = a + b$, это уравнение (в модульной форме) всегда выполняется, в отличие от линейного уравнения, где $a^n + b^n$ не равно z^n.
4. Однако возникает проблема: показатель степени n — это целое число, а не простое число p. Действительно ли это проблема? Если доказано выполнение уравнения для любого p, будет ли оно выполняться для всех целых чисел?
5. Предположим, что следующее уравнение (с использованием p' вместо p в качестве модуля) всегда выполняется:

$$a^p + b^p \equiv z^p \pmod{p'},$$

где (p') — множество всех простых чисел больше z^p. Таким образом, по определению $p' > z$.

Улавливаете суть?

При $p' > z^p$ мы имеем дело с суммой $a^p + b^p$, выраженной в линейной, а не модульной форме, поскольку результаты операции суммы попадают обратно внутрь кольца (Zp').

Давайте предположим, что первым простым числом после $z^p = 8^{11}$ является $p' = 8\,589\,934\,609$.

Вычисляем уравнение по (mod 8 589 934 609):

$$3^{11} + 5^{11} = 8^{11} = 8\,589\,934\,592 \pmod{8\,589\,934\,609}.$$

В итоге уравнение по модулю p' идентично линейному уравнению.

6. По факту оба уравнения не имеют решения из-за малой теоремы Ферма. Опять же мы имеем:

$$z^p = z \pmod{p}.$$

Следовательно:

$$z^p \neq z \pmod{p'}.$$

Таким образом, на шаге 5 мы ошибочно предположили, что $z = a^p + b^p \pmod{p'}$.

Теперь должны предположить, что:

$$z^p \neq a^p + b^p \pmod{p'}.$$

Последнее уравнение эквивалентно записи:

$$z^p \neq a^p + b^p.$$

Именно это мы и хотели продемонстрировать в контексте версии теоремы Ферма с простыми числами в качестве экспонент.

Теперь, возвращаясь к алгоритму MB09, отметим, что уравнение

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

является примером действия малой теоремы Ферма и определяет предыдущую линейную функцию.

Рассмотрим следующий пример:

$$3^7 = 3 \pmod{7}.$$

Заметьте, что здесь используется знак «=», а не «≡», что указывает на абсолютное равенство, а не просто модульную эквивалентность. В данном случае это очень важно, потому что мы имеем в виду равенство по абсолютной величине, то есть на самом деле 3 равно 3, а не конгруэнтно.

Я понимаю, что все это может показаться немного запутанным, но постарайтесь не терять мою мысль, и неясность должна исчезнуть.

Малая теорема Ферма утверждает, что $a^p = a \pmod p$, если $a < p$. Что происходит при $a > p$?

Например, попробуем взять $a = 5$ и $p = 3$:

$$5^3 \equiv 2 \pmod 3.$$

Как видите, если $a > p$, то (a) уже не равно (a) по абсолютной величине, то есть результат только конгруэнтен.

Это интригующее свойство заставило меня вспомнить множество соображений относительно малой теоремы Ферма, наиболее важным из которых здесь является формулировка следующей гипотезы:

> *«Если я возьму a >> p (a намного больше p) и буду держать [a] в секрете, то это будет односторонняя функция, в которой будет очень трудно вернуться к исходным параметрам на основе результата, скажем, от (A) к [a]».*

Рассмотрим пример.

Можно проверить, что $5^3 \equiv 2 \pmod 3$, а также $8^3 \equiv 2 \pmod 3$, 11^3... 14^3, 17^3 — все они конгруэнтны, и так до бесконечности.

Любопытно, что, если к начальному числу 5 прибавлять 3 + 3 + 3... (по сути, прибавлять 3 одно за другим), всегда получается одно и то же значение (2) по (mod 3), которое по абсолютной величине отличается от (5), что подтверждает малую теорему Ферма для $p > a$.

Наконец, в то время я хотел продемонстрировать, что очень трудно вычислить секретное число $[a]$ на основе открытого числа (A), если $p < a$.

Позвольте мне объяснить эту концепцию на примере большого значения $[a]$ = 96 269 369 030 336 679 694 019 965 478 670.

Иными словами, это число легко свести к соответствующему ему модулю $p = 3$.

Фактически мы имеем следующее:

$$96\,269\,369\,030\,336\,679\,694\,019\,965\,478\,670 \equiv 2 \pmod 3.$$

И наоборот, если известно открытое число $(A) = 2$, где верно следующее:

$$A \equiv [a]^p \pmod p,$$

то даже когда злоумышленник знает $p = 3$, ему будет очень трудно попытаться получить закрытый ключ $[a]$, если $a >> p$:

$$[a] = 96\,269\,369\,030\,336\,679\,694\,019\,965\,478\,670.$$

Это объясняется тем, что $p = 3$ — это наше кольцо (Z^p), но закрытый секретный ключ [a] находится вне кольца (Z^3) и вы не знаете, когда остановить итерацию поиска числа [a]. Теоретически [a] может быть любым числом от (p) до бесконечности, и неважно, насколько оно велико, потому что очень легко вычислить (A) на основе [a], но не наоборот. Однако все это работает лишь в некоторых случаях.

Теперь, когда вы познакомились с основами алгоритма MB09, вам, вероятно, будет интересно узнать, как он был реализован.

Строчной [a] и прописной (A) я обозначил скрытые и известные элементы уравнения соответственно. Однако ошибочно полагать, что $A > a$. На самом деле все наоборот: с точки зрения математики значение A гораздо меньше значения [a].

Подробное объяснение алгоритма MB09

Как я упомянул ранее, система MB09 основана на принципах шифрования с открытым и закрытым ключами. Тем не менее она *не* предназначена для передачи сообщений между двумя или более участниками. MB09 отлично подходит для создания протокола анонимного управления и передачи электронных денег, а также обеспечения безопасности транзакций.

Одной из целей внедрения такой схемы является обеспечение работы сети в качестве *автономной системы*. Я подробнее объясню эту концепцию в следующем разделе, когда буду представлять MBXX — улучшенную версию MB09 в децентрализованной среде. Изначально алгоритм MB09 функционировал в централизованной системе, но со временем перешел в децентрализованную распределенную архитектуру.

В следующем разделе я объясню основные концепции сети, управляемой криптографическими алгоритмами, в которой Z (централизованный администратор) должен подтвердить правильность и допустимость всех транзакций пользователей.

Z является *централизованным администратором* сети, который может представлять собой телекоммуникационный провайдер или банк. Алиса и Боб — два участника сети.

Боб — отправитель, а Алиса — получатель. Они могут быть виртуальными машинами, серверами или компьютерами. В нашем примере они являются

частью сети с большим количеством пользователей. Как показано на рис. 6.3, администратор сети (*Z*) — это сервер, связанный с большим количеством пользователей.

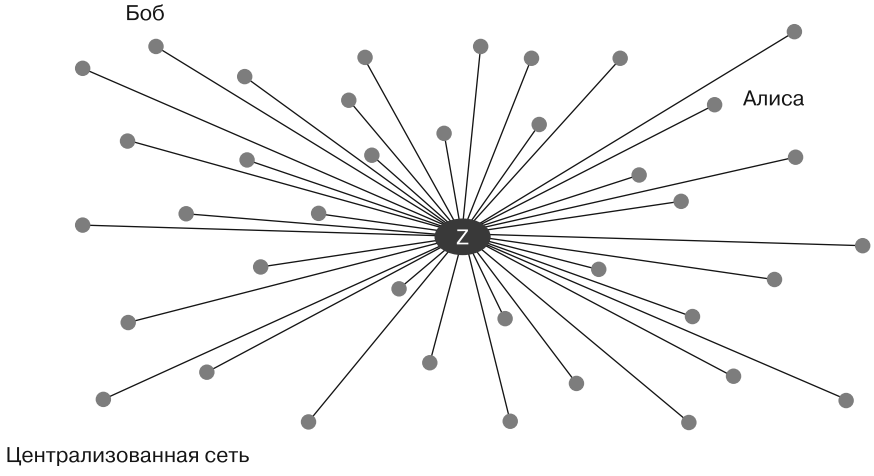


Рис. 6.3. Централизованная сеть

Мы рассмотрим алгоритм только для этих двух участников, но, как я уже упоминал, его сила заключается в возможности работы с большим количеством пользователей, которые не знают друг друга, а также не доверяют друг другу и третьим сторонам.

Z уже опубликовал некоторые параметры, такие как простое число (*p*) и открытые ключи (*A*) и (*B*) для Алисы и Боба соответственно.

На самом деле последняя теорема Ферма (в модульном виде), примененная к MB09, может работать с бесконечным числом пользователей сети, что математически выражается как:

$$a^p + b^p + c^p + n^p \equiv z^p \pmod{p}.$$

В примере я использовал сложение (+), однако вместо него могут быть XOR, вычитание или умножение.

Сообщение [*M*] было предварительно закодировано в формате ASCII (как это можно сделать, вы узнали в главе 1), и его можно рассматривать как сумму электронных денег, передаваемую между (*A*) и (*B*).

Теперь с помощью математики докажем, что Алиса и Боб могут использовать этот алгоритм для передачи денег друг другу.

Каждый шаг алгоритма после *инициализации ключей* состоит из следующих рекурсивных итераций:

- задействование открытых параметров пользователей;
- операции с параметрами — передача сообщений (например, передача электронных денег);
- пересчет новых параметров.

Инициализация ключей:

- $[a]$ — закрытый секретный ключ Алисы;
- $[b]$ — закрытый секретный ключ Боба.

Предполагается, что $[a]$ и $[b]$ — два случайных больших простых числа.

Открытые параметры

Алиса и Боб вычисляют свои открытые ключи, используя свои закрытые ключи:

- $A \equiv a \pmod{p}$ $[a >> p]$;
- $B \equiv b \pmod{p}$ $[b >> p]$;
- $C \equiv c \pmod{p}$ $[c >> p]$;
- $Z \equiv z \pmod{p}$ $[z >> p]$.

На этом этапе можно вывести две функции, $f(z)$ и $f(Z)$. Первая представлена в модульном виде, вторая — в линейном:

$$a^p + b^p + c^p + ... + n^p \equiv z^p \pmod{p};$$
$$A + B + C + ... + N = Z.$$

Соответствие элементов первой и второй формул друг другу играет ключевую роль в алгоритме. Как показано на рис. 6.4, каждый элемент первого уравнения соответствует каждому элементу второго уравнения.

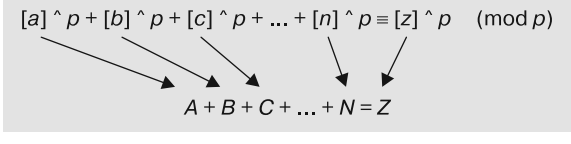


Рис. 6.4. Соответствие между элементами двух уравнений

Особенность этого соответствия заключается в том, что в первом уравнении операции выполняются с закрытыми ключами пользователей, а во втором — с открытыми. На рисунке видно, что стрелки направлены от $[a]$ к (A), и, как мы уже знаем, обратный переход от (A) к $[a]$ затруднителен.

Здесь (Z) является, по сути, *изоморфным балансом* между операциями, выполняемыми с явно выраженными параметрами, а не внутри квадратных скобок, представляющих собой секретные параметры функции $f[z]$.

На этом этапе MB09 (версии 2009 года) переходит к стандартной криптографической передаче сообщения между Алисой и Бобом на основе пары ключей.

Помните, что здесь $[a]$ и $[b]$ — это секретные значения Алисы и Боба. Кроме того, не следует полагать, что (A) больше $[a]$ только потому, что оно обозначено прописной буквой.

Сначала необходимо сгенерировать секретный ключ передачи $[K]$. Это общий ключ, который позволяет передавать сообщение $[M]$ между Алисой и Бобом. Он может быть сгенерирован любым алгоритмом с открытым и закрытым ключами, например алгоритмом Диффи — Хеллмана.

В первой версии MB09 я использовал ключ $[K]$, сгенерированный с помощью алгоритма Диффи — Хеллмана. Однако, как вы могли догадаться, этот ключ может сформировать любой другой алгоритм, использующий секретные ключи Алисы и Боба в качестве входных данных и возвращающий криптограмму (c) на выходе.

(c) обозначает криптограмму, представленную внутри круглых скобок. Здесь я использую (c), потому что у нас только два действующих лица. В других случаях буду задействовать иное обозначение — не путайте его с элементом сети.

Представление алгоритма с двумя участниками (как уже говорилось, он предназначен для многоканальной связи) выглядит так:

$$a^p + b^p \equiv z^p \pmod{p};$$
$$(A) + (B) = (Z).$$

Алиса шифрует $[M]$ с помощью своего секретного ключа $[K_A]$, который был сгенерирован алгоритмом с открытым и закрытым ключами, в результате чего получает криптограмму (c):

$$[M] \cdot [K_A] \equiv (c) \pmod{p}.$$

Здесь, как я уже говорил, (c) — это криптограмма, которая получается в результате умножения секретного сообщения $[M]$ на закрытый ключ Алисы $[K_A]$. В первоначальной версии алгоритма для создания (c) я использовал операцию умножения. Однако его можно реализовать и с помощью другого метода шифрования, например побитовой операции XOR или скалярного умножения.

Боб расшифровывает $[M]$, генерируя свой секретный ключ $[K_B]$:

$$c(\text{INV})\,[K_B] = [M]\ (\text{mod}\ p).$$

Если рассматривать этот алгоритм в контексте электронных денег, $[M]$ будет представлять собой реальную сумму передаваемых электронных денег, а (Hm) — хеш-значение $[M]$:

$$[a \pm M]^p + [b \pm M]^p \equiv [z \pm M]^p\ (\text{mod}\ p);$$
$$(A \pm Hm) + (B \pm Hm) = (Z).$$

Результатом операций в первом уравнении шифрования является значение $[z]$, связанное с сообщением $[M]$ вслепую. Оно соответствует результату операций, выполненных во втором уравнении, и совпадает с результатом линейного уравнения (Z). Позже мы рассмотрим похожий протокол при обсуждении алгоритма MBXX.

По сути, (Z) представляет собой *гомоморфный баланс* системы — передаваемые электронные деньги не меняют значение (Z). Использование такого гомоморфного баланса позволяет администратору всегда знать общий баланс электронных денег в виде изоморфных значений и контролировать точность транзакций, даже если передаваемая сумма $[M]$ неизвестна. Мы вернемся к (Z), когда будем изучать протокол MBXX, в частности в пункте «Детали протокола MBXX».

Выполним упражнение с использованием обмена ключами Диффи — Хеллмана, где $[a]$ и $[b]$ — секретные параметры Алисы и Боба соответственно.

Как мы знаем, в алгоритме Диффи — Хеллмана имеются такие открытые параметры:

- p — большое простое число;
- g — образующий элемент кольца (Z^p).

Обмен $[M]$ осуществляется следующим образом.

Шаг 1. Шифрование. Алиса генерирует открытый ключ (Ag) с помощью своего секретного параметра $[a]$:

$$Ag \equiv g^{[a]}\ (\text{mod}\ p).$$

Боб генерирует открытый ключ (Bg) с помощью своего секретного параметра $[b]$:

$$Bg \equiv g^{[b]}\ (\text{mod}\ p).$$

Алиса вычисляет:

$$Bg^{[a]} \equiv [K_A]\ (\text{mod}\ p) —$$

и шифрует $[M]$ с помощью своего ключа $[K_A]$:

$$[M] \cdot [K_A] \equiv (c)\ (\text{mod}\ p).$$

Шаг 2. Расшифровка. Алиса отправляет (c, Ag) Бобу.

Она также отправляет хеш-значение $[M] = (Hm)$ администратору.

Боб вычисляет свой ключ $[K_B]$ и расшифровывает (c), возвращая значение $[M]$:

$$Ag^{[b]} \equiv [K_B] \pmod{p}$$
$$(c) \cdot [(\text{INV})K_B] \equiv [M] \pmod{p}.$$

Обратите внимание, что $[(\text{INV})K_B]$ является обратной модульной операцией умножения, которую также можно представить в виде деления $c / K_B = M \pmod{p}$. Полная схема алгоритма передачи секретного сообщения приведена на рис. 6.5.

Если рассматривать закрытые ключи Алисы $[a]$ и Боба $[b]$ как значения их счетов, а $[M]$ — как количество переданных электронных денег, то необходим третий шаг, представляющий собой гомоморфный баланс.

Шаг 3. Гомоморфный баланс. Подставляя значение $[M]$ как в первое, так и во второе уравнение, администратор может подтвердить корректность транзакции между (A) и (B):

$$[a \pm M]^p + [b \pm M]^p \equiv [z \pm M]^p \pmod{p};$$
$$(A \pm Hm) + (B \pm Hm) = (Z).$$

Обратите внимание на следующую деталь в функции шифрования:

$$[M] \cdot [K] \equiv (c) \pmod{p}.$$

Как говорилось ранее, я выполнил шифрование, умножив сообщение $[M]$ на значение ключа $[K]$. Однако это не единственный способ зашифровать $[M]$ — например, между $[M]$ и $[K]$ можно использовать операцию побитового XOR. Обратите внимание, что здесь не так важен алгоритм передачи, как сама логическая структура протокола для реализации множественных передач.

Фактически логическая структура этого алгоритма разработана не для двух, а для большего количества пользователей.

В примере, приведенном на рис. 6.5, предполагается, что администратор знает изначальное количество денег у пользователей.

В то время вместе с коллегами я разработал проверку концепции (proof of concept, POC), чтобы доказать, что MB09 — хороший инструмент для обмена цифровыми деньгами. Мы реализовали передачу криптовалюты между мобильными телефонами. Используемый протокол был основан на модифицированной версии MB09, которую я здесь представил. Он был добавлен в действующие элементы для инженерной реализации.

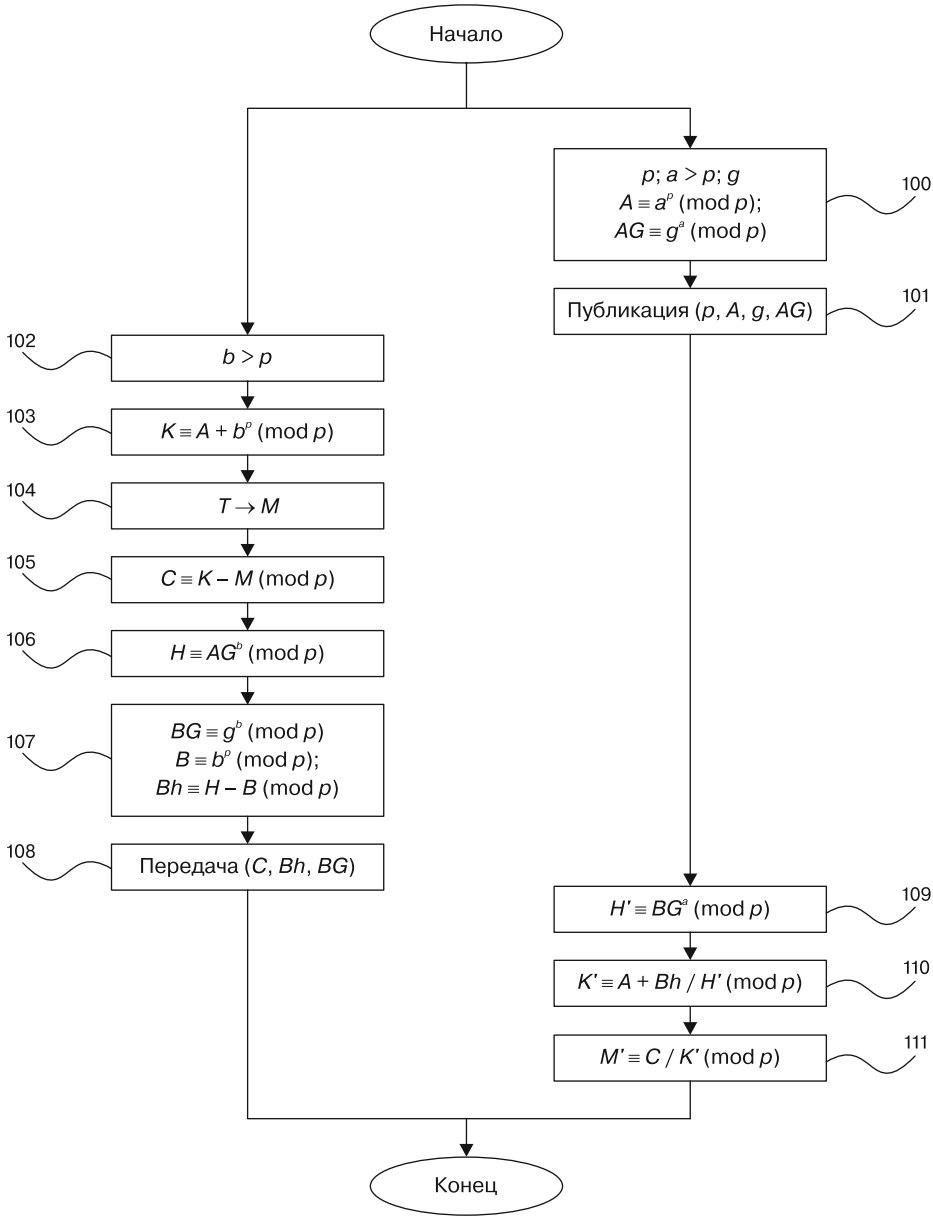


Рис. 6.5. Первоначальная версия запатентованного алгоритма MB09, опубликованная в Договоре о патентной кооперации

В 2010 году платежная система на базе MB09 была отобрана телекоммуникационной компанией в качестве кандидата для проведения электронных платежей. Впоследствии эта компания приобрела часть прав на патент MB09 и поручила нашей исследовательской лаборатории разработку нового алгоритма с закрытым и открытым ключами MBXI, который мы и рассмотрим далее.

Введение в алгоритм MBXI

Я начал работу над проектом MBXI в 2010 году. Схема алгоритма уже давно существовала у меня в голове, но оставались некоторые неразрешенные вопросы. Работая над проектом MB09 для создания сети электронных платежей, я столкнулся с необходимостью разработки проприетарного алгоритма, который позволял бы передавать сообщение $[M]$. В этом контексте $[M]$ представляет собой сумму электронных денег, передаваемую между двумя или более участниками сети.

Таким образом, я предложил, а в ноябре 2011 года запатентовал новый алгоритм с открытым и закрытым ключами — MBXI. Спустя десять лет патент все еще остается в силе.

MBXI представляет собой криптографический процесс, который совмещает в себе черты алгоритмов симметричного и асимметричного шифрования. В главах 2 и 3 мы изучили принципы работы симметричного и асимметричного видов шифрования. MBXI основан на задаче дискретного логарифма, которая уже рассматривалась в других алгоритмах, например в алгоритме Диффи — Хеллмана, схеме Эль-Гамаля, доказательствах с нулевым разглашением. Как известно, нахождение дискретного логарифма по-прежнему остается очень трудной задачей.

Надежность этого алгоритма обусловлена использованием модульных экспоненциальных уравнений, которые работают как односторонние функции при определении показателей степени в уравнениях шифрования и расшифрования. Решение таких уравнений, в которых используется большое простое число (например, для модуля p), — крайне тяжелая или даже практически невозможная задача.

В MBXI единственным способом определения закрытого ключа является нахождение дискретного логарифма, в отличие от других асимметричных алгоритмов, например RSA или схемы Эль-Гамаля, уязвимых со стороны факторизации и дискретного логарифма (как объяснялось в главе 3).

Фактически MBXI нельзя однозначно отнести ни к асимметричному, ни к симметричному виду шифрования. Это алгоритм с открытым и закрытым ключами. Причины я поясню позже.

Давайте углубимся в схему работы MBXI.

Алиса и Боб остаются основными участниками протокола. В данном сценарии предположим, что Боб хочет отправить Алисе секретное сообщение [M].

Первый шаг алгоритма — генерирование закрытых и открытых ключей на основе общих открытых параметров, опубликованных в сети:

- p — большое простое число;
- g — образующий элемент кольца (Z^p).

Шаг 1. Генерирование ключей. Здесь:

- [a] — закрытый секретный ключ Алисы (большое число, выбранное в пределах $p - 1$);
- [b] — закрытый секретный ключ Боба (большое число, выбранное в пределах $p - 1$);
- открытый ключ Алисы — $K_A \equiv g^a \pmod{p}$;
- открытый ключ Боба — $K_B \equiv g^b \pmod{p}$.

Шаг 2. Шифрование. В MBXI оно выполняется на основе обратного модульного уравнения:

$$\left\{\left[K_A^b + e_B\right] \pmod{p}\right\} \cdot x \equiv 1 \pmod{p-1}.$$

В этом уравнении Боб берет открытый ключ Алисы (K_A), возводит его в степень своего закрытого ключа [b] и добавляет параметр (e_B). Параметр (e_B) выбирается из множества простых чисел так, чтобы значения $\left[K_A^b + e_B\right] \pmod{p}$ и ($p - 1$) были взаимно простыми.

Затем, решив обратное модульное уравнение для [x], Боб вычисляет криптограмму (C):

$$C \equiv M^x \pmod{p}.$$

Боб отправляет Алисе тройку параметров (C, e_B, K_B).

Шаг 3. Расшифровка. Алиса может расшифровать (C), используя открытый ключ Боба (K_B), свой закрытый ключ [a] и переданный Бобом параметр (e_B). Функции параметра (e_B) будут описаны позже при подробном объяснении алгоритма.

Расшифровка выполняется путем решения следующего уравнения, результатом которого является закрытый ключ расшифрования Алисы [y]:

$$y \equiv \left\{\left[K_B^a + e_B\right] \pmod{p}\right\}.$$

Наконец, сообщение [M] восстанавливается путем возведения (C) в степень [y]:

$$M \equiv C^y \pmod{p}.$$

Здесь [y] является закрытым ключом расшифрования Алисы, который задается путем решения обратного модульного уравнения.

На рис. 6.6 представлена схема процесса шифрования и расшифрования в алгоритме MBXI. Здесь сообщение [T] разбивается на блоки подсообщений [M].

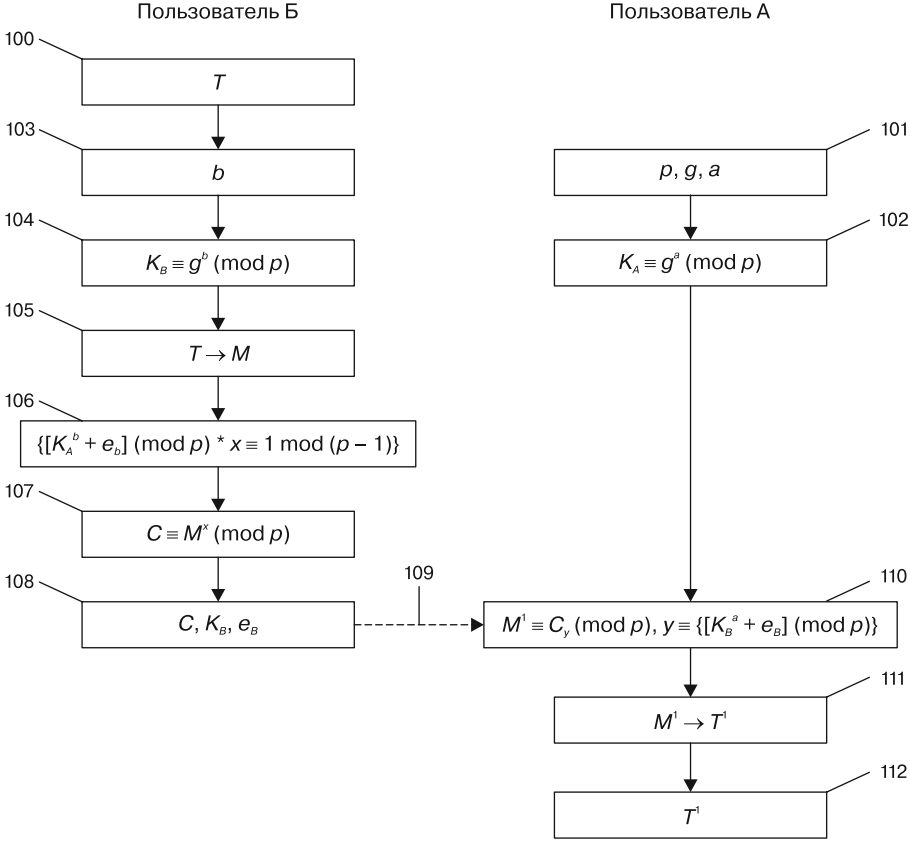


Рис. 6.6. Процесс шифрования и расшифровки в MBXI

Числовой пример. Пример иллюстрирует криптографическую коммуникацию между отправителем (Бобом) и получателем (Алисой). Разумеется, как и во всех других примерах, мы используем относительно небольшие числа, но в реальных приложениях закрытые ключи имеют минимальный размер около 3000 бит (то есть тысячи цифр), например простое число (p).

Мы начинаем с двух исходных параметров:

- $p = 7919$;
- $g = 7$ (образующий элемент).

Алиса выбирает закрытый ключ [a]:

$$a = 123\,456.$$

Боб выбирает закрытый ключ [b]:

$$b = 543\,210.$$

Как вы могли заметить, я взял обычную последовательность цифр $a = 123\,456$ и $b = 543\,210$. Это просто пример, и в реальном мире никогда нельзя использовать подобную последовательность при выборе пароля, так как это первое, что пробует ввести злоумышленник.

Первым шагом алгоритма является генерирование ключей, Алиса вычисляет свой открытый ключ:

$$K_A \equiv 7^{123\,456} \pmod{7919} = 7036.$$

Боб вычисляет свой открытый ключ:

$$K_B \equiv 7^{543\,210} \pmod{7919} = 4997.$$

Боб выполняет шифрование с помощью следующего уравнения:

$$\{[7036^{543\,210} + 1] \pmod{7919}\} \cdot x \equiv 1 \pmod{7919 - 1}.$$

Здесь $(e_B) = 1$ — это самое маленькое целое число, которое обеспечивает выполнение уравнения.

Таким образом, Боб определяет закрытый ключ шифрования [x]:

$$x = 3009.$$

С помощью этого параметра он вычисляет криптограмму (C):

$$C \equiv 88^{3009} \pmod{7919} = 2760.$$

Боб отправляет Алисе тройку параметров $(C, K_B, e_B) = (2760, 4997, 1)$.

Алиса использует эти параметры для дешифрования:

$$M = 2760\text{^}\{[4997^{123\,456} + 1] \pmod{7919}\} \pmod{7919} = 88.$$

$M = 88$ совпадает с исходным сообщением, отправленным Бобом.

Таким образом я продемонстрировал работоспособность алгоритма на реальных числах. Теперь рассмотрим некоторые его функции и общие черты с алгоритмом RSA.

Особенности алгоритма MBXI и предисловие к атаке на RSA

Прежде всего я хотел бы прояснить смысл параметра (e_B). Он открыто передается от отправителя к получателю. Однако в других версиях он может изменяться в зависимости от случайных факторов.

Параметр (e_B) выбирается случайным образом в диапазоне (1, 10). Пусть (e_B) = 6, тогда уравнение шифрования будет иметь следующий вид:

$$\{[7036^{54\,3210} + e_B] \pmod{7919}\} \cdot x \equiv 1 \pmod{7919 - 1}.$$

Обратите внимание, что при (e_B) = 6, уравнение *не* выполняется!

Действительно, если мы попытаемся подставить в предыдущую функцию $e_B = 6$ вместо (e_B) = 1, используя функцию упрощения Reduce в системе компьютерной алгебры Wolfram Mathematica, то получим вот такой результат:

$$\text{Reduce}[4960 \cdot x = = 1, x, \text{Modulus} \rightarrow 7919 - 1];$$

ЛОЖЬ.

Другими словами, обратного числа для 4960 (mod 7919) не существует. При этом случайно выбранное из заданного диапазона число увеличивается на 1, и процесс проверки повторяется с использованием (e_B) = 7, что удовлетворяет условию взаимной простоты чисел, при котором $x = 5575$.

Таким образом, мы получим следующий процесс шифрования:

$$C \equiv 88^{5575} \pmod{7919} = 2195.$$

В этом случае Боб передает Алисе тройку параметров (C, K_B, e_B) = (2195, 4997, 7).

Расшифровка будет выглядеть так:

$$M \equiv 2195\text{^}\{[4997^{123\,456} + 7] \pmod{7919}\} \pmod{7919}.$$

Из этого уравнения следует, что $M = 88$ соответствует исходному блоку сообщения [M], который был передан Бобом.

Еще один важный момент относится скорее к реализации MBXI, в частности к нескольким проблемам, связанным с длиной сообщения [M].

Сообщение [$M = 1$] означает, что и шифротекст ($C = 1$) независимо от ключа шифрования.

Аналогичное свойство имеет алгоритм RSA в его простейшей форме. Большинство практических реализаций RSA включают в себя дополнение сообщений, которое устраняет предсказуемость. То же самое можно сделать и в MBXI. Однако для некоторых узкоспециализированных целей сохранение длины может быть

полезным. Технический термин — *гомоморфное шифрование*. Частичную гомоморфность алгоритма RSA мы рассмотрим в главе 8.

Как говорилось в главе 5, для устранения (в общем случае) подобных нежелательных свойств необходимо *дополнение (паддинг)*, которое удлиняет строку шифротекста.

Возможно, вы сейчас задаетесь вопросом, является ли MBXI подлинным алгоритмом асимметричного шифрования. Мой ответ: нет, это именно алгоритм шифрования с открытым и закрытым ключами. Данная особенность MBXI имеет несколько интересных вариантов применения, например для совместного использования ключа, однако она не может заменить мощь такого симметричного алгоритма, как AES.

Чтобы продемонстрировать схему общего ключа, я переформулирую уравнение шифрования:

$$C \equiv M^x \pmod{p}.$$

Неожиданно мы замечаем сходство со структурой шифрования RSA:

$$C \equiv M^e \pmod{N}.$$

Достаточно поменять местами параметры $[x]$ и (e), и окажется, что две схемы шифрования похожи.

В RSA, напомню, (e) — это открытый параметр, а (N) — открытый ключ получателя, задаваемый как $N = [p] \cdot [q]$.

В отличие от RSA в MBXI $[x]$ является секретным ключом (как в алгоритме Диффи — Хеллмана), а (p) — большое открытое простое число.

Это сравнение будет полезно для объяснения некоторых интересных атак на RSA, на которые меня вдохновила схожесть схем шифрования RSA и MBXI.

Нетрадиционные атаки и саморасшифровка в RSA

Понятие асимметричного бэкдора было введено Адамом Янгом и Моти Юнгом в статье *Proceedings of Advances in Cryptology* («Достижения в криптологии»). Асимметричный бэкдор может использовать только тот, кто его внедрил, даже если его полная реализация находится в общем доступе (например, через публикацию, обнаружение и раскрытие методом обратного проектирования). Такой вид атак назван *клептографией* (kleptography), они могут выполняться на уровне программного обеспечения, аппаратного обеспечения (например, на

смарт-картах) или их комбинации. Теория асимметричных бэкдоров теперь является частью более широкой области, называемой *криптовирусологией* (cryptovirology). Примечательно, что АНБ США упомянуло клептографические бэкдоры в стандарте Dual EC DRBG.

Существует экспериментальный асимметричный бэкдор в процессе генерирования ключей в алгоритме RSA. Этот бэкдор, внедренный в OpenSSL RSA и опубликованный Янгом и Юнгом, использует скрученные пары эллиптических кривых. В этом разделе мы рассмотрим несколько иной подход, который предполагает внедрение параметра внутрь *модуля N* функции шифрования RSA. Кроме того, можно изменить всего несколько строк кода в библиотеке OpenSSL, и в результате мы получим возможность создать бэкдор в RSA.

Бэкдор позволяет злоумышленнику шпионить за коммуникацией между двумя и более людьми. Он также дает возможность взломать не поддающийся криптоанализу шифр с использованием математических или аппаратных методов.

В обоих случаях создание бэкдора требует определенных ресурсов, навыков и подготовки для внедрения вредоносного кода. В конечном счете это позволяет третьей стороне (Еве) видеть или изменять зашифрованное сообщение. Однако есть еще один вариант, позволяющий создать более изощренный бэкдор: отправитель сам может встроить его в собственное шифрование.

Реализация бэкдора в злонамеренных целях противозаконна. Поэтому информацию, изложенную в этом разделе, следует воспринимать исключительно в качестве теории, которая говорит о том, что в алгоритме RSA существует такая возможность.

Пришло время вернуться к вопросу из главы 3: является ли метод *автообратимого* расшифрования (self-reverse decryption)[1] нереалистичной моделью, или он может применяться на практике? В каких случаях оправданно применение этого метода атаки на сеть?

Представьте, что Боб является администратором телекоммуникационной компании, социальной сети или облачного провайдера и хочет шпионить за

[1] В контексте автообратимого расшифрования может возникнуть вопрос: если речь идет о бэкдорах, то почему мы называем это расшифровкой (то есть чем-то легитимным), а не дешифровкой? Ответ заключается в том, что мы не будем «ломать» сам шифр или выполнять криптоанализ. Вместо этого мы слегка модифицируем сам алгоритм, внедрив в него бэкдор. И уже этот модифицированный шифр сможем расшифровывать штатной операцией. Далее в тексте будет пример.

своими пользователями. В криптовалютной сфере также возможны интересные сценарии, когда Боб (отправитель) может создать поддельную подпись на передаваемой сумме. Это, вероятно, позволит ему выполнить *двойное расходование* электронных денег. При этом метод *автообратимого* расшифрования можно использовать и в благих целях — например, для восстановления зашифрованных файлов после атаки программ-вымогателей.

Как написал Саймон Сингх в книге «Книга шифров» (1998), в отчете Wayne Madsen Report утверждалось, что швейцарская криптографическая компания Crypto AG внедрила бэкдоры в некоторые свои продукты и предоставила доступ к ним правительству США. В результате американские спецслужбы могли читать зашифрованные сообщения пользователей из различных стран. В 1991 году убийцы бывшего премьер-министра Ирана Шапура Бахтияра были пойманы благодаря перехвату и расшифровке сообщений из Ирана, которые передавались с помощью оборудования Crypto AG.

Теперь рассмотрим, как создать логический бэкдор с помощью функций, представленных в этой книге.

Один из главных принципов шифрования RSA заключается в том, что отправитель (Боб) не может повторно расшифровать криптограмму (c) после ее отправки. Это обусловлено тем, что только получатель (Алиса), владеющая закрытым ключом (p, q), может вычислить (N). Соответственно, она единственная, кто может расшифровать криптограмму (c) и получить сообщение [M].

Освежим знания, полученные в главе 3. На рис. 6.7 изображен этап шифрования в алгоритме RSA.

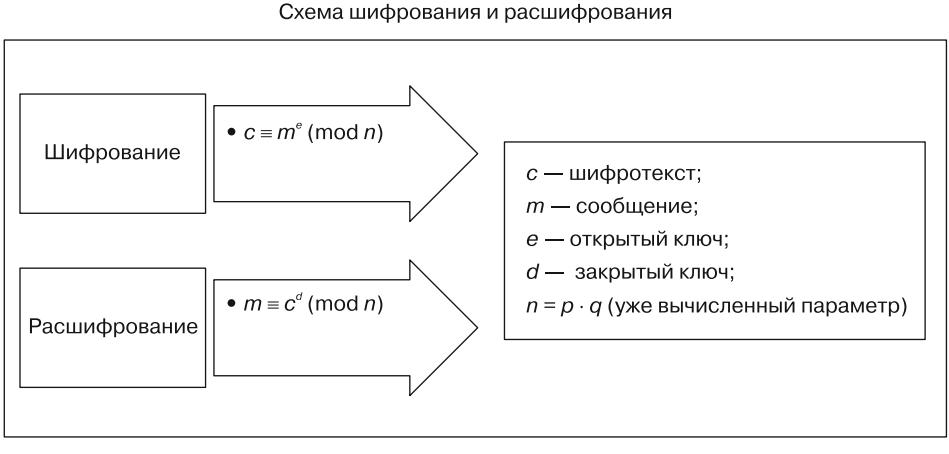


Рис. 6.7. Этап шифрования RSA

На этом этапе мы имеем следующие параметры:

- $[M]$ — секретное сообщение;
- $(N) = [p] \cdot [q]$ — открытый ключ Алисы;
- e — открытый параметр.

Следует отметить, что шифрование выполняется отправителем (Бобом) с использованием открытого ключа Алисы, который мы обозначим (N_A):

$$[M]^e \equiv c \pmod{N_A}.$$

Боб знает секретное сообщение $[M]$, но он *не может* математически повторно *расшифровать* криптограмму (c) после того, как передаст ее Алисе, поскольку предполагается, что только Алиса знает свой закрытый ключ $[d_A]$.

Генерация ключа. Открытый ключ Алисы (N_A) задается следующим образом:

$$N_A = [p] \cdot [q].$$

$[d_A]$ вычисляется так:

$$[d_A] \cdot e \equiv 1 \pmod{[p-1] \cdot [q-1]}.$$

Как рассказывалось в главе 3, обратное умножение является основой алгоритма RSA, так как позволяет получить секретное сообщение $[M]$, скрытое в криптограмме (c), с помощью следующей операции дешифрования:

$$M \equiv c^{d_A} \pmod{N_A}.$$

Таким образом, одна из основных парадигм алгоритма RSA заключается в том, что отправленная криптограмма (c) не может быть расшифрована никем, кто не является владельцем закрытого ключа $[d_A]$.

Однако это утверждение можно опровергнуть с помощью примера с внедренным бэкдором.

Давайте посмотрим, как реализовать псевдоатаку, которая позволяет сгенерировать автообратимое расшифрование криптограммы Бобом в RSA.

Еще раз напомню: здесь мы предполагаем, что автором атаки является сам Боб (отправитель). Как вы увидите позже, внешний злоумышленник Ева тоже может легко выдать себя за Боба.

Открытый и закрытый ключи:

- (N_A) — открытый ключ Алисы (получателя);
- (N_B) — открытый ключ Боба (отправителя);
- (e) — открытый параметр (задается системой);
- $[d_A]$ — закрытый ключ Алисы;
- $[d_B]$ — закрытый ключ Боба.

Боб выполняет «атаку», самостоятельно изменяя параметры.

Шаг 1. Шифрование. Боб выбирает такое простое число $[P_B] > [M]$, которое позволит ему выполнить модифицированное шифрование (M) и *получить новую криптограмму* (c_1):

$$[M]^e \equiv c_1 \pmod{N_A \cdot P_B}.$$

Он отправляет (c_1) Алисе.

Параметр $[P_B]$ — это тот самый бэкдор, который позволяет Бобу вернуть сообщение $[M]$.

Шаг 2. Расшифровка Боба. На этом этапе Боб вычисляет параметр $[x]$, который задается функцией обратного умножения:

$$e \cdot x \equiv 1 \pmod{P_B - 1}.$$

Результат обратного умножения очень важен, так как операция возвращает параметр $[x]$, который является специальным закрытым ключом Боба, дающим доступ к расшифровке (c_1). Это и есть магическая функция, позволяющая получить сообщение $[M]$.

Теперь Боб может расшифровать (c_1), используя $[x]$ в качестве своего закрытого ключа:

$$(c_1)^x \equiv [M] \pmod{P_B}.$$

Боб может математически извлечь сообщение $[M]$ из шифротекста (c_1). Это опровергает общепринятое утверждение о том, что в RSA отправитель не может расшифровать криптограмму после ее отправки, так как не владеет закрытым ключом получателя.

Кто-то может возразить, сказав, что мы изменили шифрование RSA, поэтому имеем дело с другим алгоритмом, а не с RSA. В какой-то степени это верное замечание, но давайте посмотрим, что произойдет, когда Алиса расшифрует (c_1).

Шаг 3. Расшифровка Алисы. Здесь начинается самая интересная часть атаки: как только Алиса получает поддельную криптограмму (c_1), она может расшифровать ее с помощью своего закрытого ключа $[d_A]$ и получить секретное сообщение $[M]$. Таким образом, этап расшифровки RSA остается неизменным:

$$(c_1)^{d_A} \equiv [M] \pmod{N_A}.$$

В результате Алиса получает сообщение $[M]$, не зная о фальшивой криптограмме (c_1).

Как показывает рис. 6.8, Боб может выполнить автообратимое расшифрование измененной криптограммы, в то время как Алиса использует свой закрытый ключ для расшифровки поддельной криптограммы (c_1):

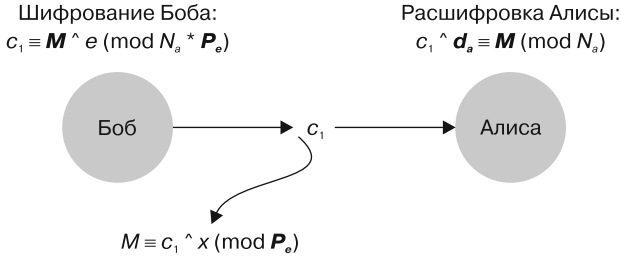


Рис. 6.8. Схема атаки автообратимого расшифрования Боба

Так же легко продемонстрировать, что если эту атаку выполнит Ева (внешний злоумышленник), внедрив свой параметр $[P_e]$ (бэкдор) в этап шифрования Боба, то она сможет перехватить сообщение $[M]$, которым обмениваются Боб и Алиса.

На рис. 6.9 показана схема шпионской атаки, предпринятой Евой.

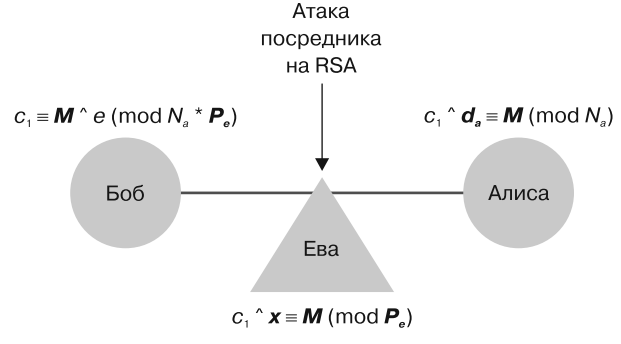


Рис. 6.9. Схема шпионской атаки Евы (атака посредника)

С помощью этой схемы Ева может не только получить сообщение $[M]$, но и изменить его, отправив Алисе поддельное новое сообщение $[M_1]$, а затем воссоздав новое (подлинное) RSA-шифрование после прочтения сообщения, отправленного Бобом.

Атака возможна только в том случае, если Ева подсчитает биты в библиотеке OpenSSL. Это связано с тем, что на этапе фальшивого шифрования для криптограммы (c_1) генерируется слишком много битов, что делает ее значительно длиннее подлинной криптограммы (c). Говоря математическим языком, $c_1 >> c$. В алгоритме может быть установлен *предел* битов, заданный конкретными параметрами, например длиной ключа или модулем N. Таким образом, если $N = 2000$ бит, то (c_1) не может содержать более 2000 бит, иначе система отклонит ее.

Я нашел способ избежать этой проблемы, разработав алгоритм сжатия, который генерирует выходную криптограмму на этапе шифрования $(c_x) < (N_A)$. Таким образом, криптограмма (c_1), сформированная на этапе фальшивого шифрования, сокращается до (c_x) и библиотека OpenSSL принимает ее в качестве *корректного* параметра. Этот алгоритм сжатия может быть использован и в других целях — например, для сжатия и перекомпоновки криптограммы при передаче для экономии пропускной способности.

Как видно на рис. 6.10, Ева может перехватить передаваемое Бобом сообщение $[M]$ и заменить его на $[M_1]$, отправив Алисе новую криптограмму (c_E), сгенерированную с помощью подлинного этапа шифрования RSA.

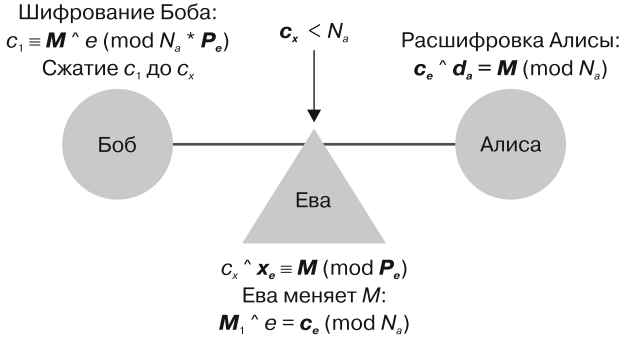


Рис. 6.10. Атака Евы с помощью алгоритма сжатия

Теперь мы можем ответить на другой вопрос: будет ли атака Евы успешной, если Алиса запросит постановку цифровой подписи на сообщении $[M]$?

Другими словами, существует ли способ противостоять таким атакам?

Ответ на этот вопрос мы получим в следующем разделе, где я расскажу о новом методе создания цифровой подписи, который применим не только к RSA, но и к большинству алгоритмов с закрытым и открытым ключами.

Новый протокол защиты RSA и асимметричные алгоритмы защиты от шпионажа

Существует множество способов защиты RSA от автообратимого расшифрования и вообще от шпионских атак, и выбор того или иного варианта зависит от того, кто проводит атаку. Один из них связан с цифровыми подписями (их мы рассматривали в главе 4). Для противодействия фальшивому шифрованию и бэкдорам, упомянутым в предыдущем разделе, требуется особый вид цифровой подписи.

В связи с этим я придумал новый протокол, гарантирующий, что получатель расшифрует неизмененную криптограмму. С его помощью отправитель может поставить цифровую подпись на криптограмме.

Действительно, если Боб (отправитель) подпишет криптограмму (c) личной подписью [d_B], то Алиса (получатель) может быть уверена, что никто не перехватывал и не изменял сообщение [M], а криптограмма (c) действительно была создана с использованием подлинного шифрования RSA и отправлена Бобом.

Посмотрим, как работает этот протокол.

Боб подписывает криптограмму (c), используя свой закрытый ключ [d_B]:

$$c^{[d_B]} \equiv S_B \pmod{N_B}.$$

Он отправляет свою подпись (S_B) Алисе, которая проверяет выполнение условия[1]:

$$(S_B)^e \equiv cv \pmod{N_B}.$$

С помощью этой операции Алиса (и все желающие) может найти криптограмму через значение cv, что является проверкой криптограммы.

Если Алиса обнаружит, что сообщение [M], возведенное в степень открытого параметра (e), соответствует полученной криптограмме так, что $cv = c$, она может быть уверена в отсутствии бэкдора в алгоритме:

$$[M]^e \equiv c \pmod{N_B}.$$

- Если $cv = c$, то Алиса примет сообщение.
- Если $cv \neq c$, то Алиса отклонит сообщение.

Выполняя операцию $cv = c$, Алиса сравнивает криптограмму от Боба с криптограммой, полученной при расшифровке подписи (S_B). Если они совпадают, значит, никакого бэкдора нет. (Конечно, стоит отметить, что в криптографии никто не может быть на 100 % уверен в отсутствии шпионажа.) Если в результате вместо (c) получается (c_1), (c_x) или любая другая криптограмма, Алиса может сделать вывод: что-то идет не так.

Здесь подпись была поставлена на криптограмме с открытым параметром (c), а не на сообщении [M], как делается обычно. Таким образом, функция может быть выполнена подписанием непосредственно (c), а не его хеш-значения. На практике абсолютной рекомендацией является создание цифровой подписи для [M].

---

[1] В этом случае Алиса проверяет выполнение условия с публичной экспонентой Боба (eB). В формуле eB пишется как обычная e.

Этот протокол работает для всех криптограмм (c) независимо от выбранного асимметричного алгоритма. Достаточно использовать закрытый ключ отправителя в качестве степени, в которую возводится криптограмма (c).

В следующем разделе мы рассмотрим различные способы применения цифровых подписей к алгоритму MBXI.

Цифровые подписи в MBXI

Вернемся к MBXI. Замечу, что для шифрования может использоваться измененный ключ шифрования [x]:

$$C \equiv M^{x} \pmod{p}.$$

[x] является обратным значением ключа расшифровки [y], представленного в следующем уравнении:

$$C^{y} \equiv M \pmod{p}.$$

В математическом выражении шифрование выглядит так:

$$\left\{\left[K_A^b + e_B\right] \pmod{p}\right\} x \equiv 1 \pmod{p-1}.$$

Результатом этого выражения является [y] — значение, обратное к уравнению расшифровки:

$$y \equiv \left\{\left[K_B^a + e_B\right] \pmod{p}\right\}.$$

Чтобы лучше понять, проведем числовую проверку:

- $x = 3009$;
- $y = 4955$.

Если подставить $x = 3009$ в обратную функцию (mod $p - 1$), можно найти значение [y] с помощью системы Mathematica:

$$\text{Reduce}[3009 \cdot x = = 1, y, \text{Modulus} \to p - 1];$$
$$y = = 4955.$$

Таким образом, если Боб отправляет сообщение, используя MBXI, он фактически делится секретным ключом с Алисой.

Однако возникает другая проблема: как защититься от атаки посредника в симметричном алгоритме?

Как видно, MBXI обладает бóльшим сходством с асимметричными алгоритмами, чем с симметричными, поэтому рассмотрим цифровые подписи для MBXI.

Мы уже знаем, что MBXI может выступать как алгоритм обмена ключами, аналогичный алгоритму Диффи — Хеллмана. Однако, в отличие от последнего,

в MBXI доступна функция цифровой подписи. Более того, он предполагает несколько способов создать цифровую подпись.

Вспомним основные требования, которым должны отвечать цифровые подписи.

- Аутентификация — получатель может подтвердить личность отправителя.
- Невозможность отказаться — отправитель не может отказаться от отправленного сообщения.
- Целостность — получатель не может изменить или сфабриковать уже подписанный документ.

Кроме того, зная о возможных уязвимостях цифровых подписей, добавим еще одно важное требование — *защиты от шпионажа*: никто не должен иметь возможности тайно получить доступ к содержимому документа.

Последнее требование станет основным объектом дальнейшего изучения. Создание цифровой подписи состоит из трех шагов.

1. Алгоритм генерирования ключей (G) производит пару ключей {(Pk) и [Sk]}, где (Pk) — это открытый ключ для проверки подписи, а [Sk] — секретный ключ отправителя, используемый для подписания сообщения [M].
2. Алгоритм подписания (S) использует сообщение [M] (как правило, его хеш) в качестве входного значения и секретный ключ [Sk], чтобы создать цифровую подпись (s).
3. Алгоритм проверки подписи (V) подтверждает или отклоняет подпись, используя сообщение [M] (как правило, его хеш) в качестве входного значения, открытый ключ (Pk) и подпись (s).

Учитывая все эти требования и условия, наши цифровые подписи могут быть двух типов.

- *Прямая подпись.* Это схема получения сообщения [M] непосредственно из функции подписания (S). Она обычно используется в RSA для создания цифровой подписи (S) на сообщении, как мы видели в главе 4:

  $$[M]^d \equiv S \pmod{N}.$$

  Она также предполагает этап проверки $S^e \equiv M \pmod{N}$.
- *Подпись с дополнением.* Для проверки ее действительности не требуется исходное сообщение (примером может служить подпись Эль-Гамаля, в которой для проверки подписи не нужно напрямую подписывать сообщение [M]).

Теперь, когда мы разобрались с методами создания цифровой подписи, можем пойти дальше и проанализировать какой-нибудь отдельный случай применения MBXI.

Создание прямой цифровой подписи в MBXI

Прямая подпись напрямую связана с сообщением $[M]$. Однако, как мы выяснили в главе 4, в некоторых случаях лучше использовать хеш сообщения $H(m)$, так как в противном случае (S) позволяет восстановить оригинальное сообщение $[M]$.

Посмотрим, как работает этот метод в алгоритме MBXI. У нас есть следующие параметры:

- закрытый ключ Алисы $[a]$ (SK_A);
- открытый ключ Алисы (K_A) (PK_A);
- закрытый ключ Боба $[b]$ (SK_B);
- открытый ключ Боба (K_B) (PK_B).

Открытый ключ Боба определяется как:

$$K_B \equiv g^b \pmod{p}.$$

Если Боб отправляет сообщение $[M]$, он выбирает параметр $[e_B]$, известный только ему и Алисе. Это важно, так как шифрование возможно только при определенных значениях $[e_B]$:

$$\left\{\left[K_A^b + e_B\right] \pmod{p}\right\} x \equiv 1 \pmod{p-1}.$$

Боб и Алиса собираются вместе, чтобы автономно определить $[e_B]$ в том смысле, что $[e_B]$ генерируется шаг за шагом в ходе процесса, который я назвал *совместной итерацией*. Он состоит из последовательной итерации $[e_B]$, продолжающейся до тех пор, пока не будут выполнены функции результатов шифрования.

Итак, мы можем привести предыдущее уравнение шифрования в следующий вид:

$$C \equiv M^x \pmod{p}.$$

Здесь $[x]$ — это результат предыдущих функций шифрования.

Если $[M]$ — это передаваемое сообщение, а $H(m)$ — его хеш-значение, то Боб может подписать $H(m)$ тем же способом, каким выполнял шифрование:

$$S \equiv H(m)^x \pmod{p}.$$

Здесь (S) — это цифровая подпись.

Алиса получает криптограмму (C) вместе с подписью (S), вычисленную с помощью хеша сообщения $[M]$.

Она может расшифровать подпись (S) с помощью того же уравнения, которое использовалось для расшифровки [M], но с другим значением [e_B]. Боб повторно вычислил [e_B] и перенес это значение на следующий шаг проверки своей функции шифрования:

$$H(m) \equiv S^y \pmod{p}.$$

Здесь [y] задается функцией расшифрования:

$$y = \left\{\left[K_B^a + e_B\right] \pmod{p}\right\}.$$

Как можно видеть, Алиса проверяет подпись (S) с помощью объединения открытого параметра Боба (K_B) и своего секретного ключа [a].

Никто, кроме Боба, не может быть отправителем, так как для уравнения подписания (S) может использоваться только его закрытый ключ [b]:

$$\left\{\left[K_A^b + e_B\right] \pmod{p}\right\} x \equiv 1 \pmod{p-1}.$$

Злоумышленник может вернуть закрытый ключ Боба [b] из параметра (K_B) с помощью дискретного логарифма. Однако, как вы уже видели, это очень сложная задача.

Действительно, [x] — это результат предыдущей функции (S), и для его получения необходим закрытый ключ Боба [b].

Давайте перейдем к другому методу создания цифровой подписи, который можно применить в MBXI.

Создание подписи с дополнением в MBXI

Второй способ подписания сообщения в MBXI называется *подписью с дополнением*. Этот метод аналогичен подписи Эль-Гамаля (см. главу 4). Алгоритм MBXI приводит к детерминированному равенству результатов, что доказывает истинность подписи.

Как и раньше, мы имеем следующие параметры:

- закрытый ключ Боба [b] (SK_B);
- открытый ключ Боба (K_B) (PK_B);

Они задаются функцией:

$$K_B \equiv g^b \pmod{p}.$$

Задача алгоритма подписания (S) — создать такое доказательство (s), предоставляемое Бобом, чтобы Алиса (получатель) могла проверить его подлинность (V).

Для этого Боб создает хеш секретного сообщения [M] так, чтобы никто не мог восстановить [M] на основе $H(m)$, не зная исходного сообщения.

Следовательно, чтобы продемонстрировать, что он является тем, за кого себя выдает, Боб подписывает $H(m)$.

Шаг 1. Генерирование ключей (G). Боб выбирает случайное число [k] в кольце (Zp). Затем он вычисляет (r):

$$r \equiv g^k \pmod{p}.$$

Шаг 2. Создание цифровой подписи (s). Боб вычисляет цифровую подпись (s) на основе хеша сообщения $H(m)$, объединенного со случайным ключом [k] и закрытым ключом [b]:

$$s \equiv H(m) \cdot [k + b] \pmod{p - 1}.$$

Боб отправляет тройку параметров ($H(m)$, s, r) Алисе.

Шаг 3. Проверка подписи (V). Алиса проверяет функции (V) и (V_1):

$$(V)g^s \equiv V \pmod{p};$$

$$\left(V_1\right)r^{H(m)} \cdot K_B^{H(m)} \equiv V_1 \pmod{p}.$$

Если $V = V_1$, Алиса принимает подпись Боба. Такой способ можно считать стандартом подписания сообщения M с помощью его хеша.

Случайный ключ [k] держится в секрете и меняется в каждой сессии создания цифровой подписи.

Теперь рассмотрим алгоритм создания цифровой подписи в MBXI с математической точки зрения.

Демонстрация алгоритма создания цифровой подписи MBXI с точки зрения математики

В этом разделе я покажу с математической точки зрения, как работают цифровые подписи в MBXI.

Согласно свойствам возведения в степень, если представить уравнение из шага 2 в качестве степени для образующего элемента (g), то получится следующий результат:

$$g^s \equiv g^{[k+b]^{H(m)}} \equiv \left(g^k\right)^{H(m)} \cdot \left(g^b\right)^{H(m)} \pmod{p}.$$

Таким образом, при $g^k = r$ и $g^b = K_B$:

$$g^s \equiv r^{H(m)} \cdot K_B^{H(m)} \pmod{p}.$$

Именно это мы и хотели продемонстрировать.

Числовой пример. Возьмем в качестве входных данных те же параметры, что и в предыдущем примере шифрования:

- $p = 7919$;
- $g = 7$;
- $e_B = 1$.

Закрытый ключ Алисы $[a]$, где $a = 123456$.

Закрытый ключ Боба $[b]$, где $b = 543210$.

Открытый ключ Алисы:

$$K_A \equiv 7^{123\,456} \pmod{7919} = 7036.$$

Открытый ключ Боба:

$$K_B \equiv 7^{543\,210} \pmod{7919} = 4997.$$

Напомню, что сообщение $[M]$, зашифрованное Бобом, имеет значение $M = 88$.

Сначала рассмотрим создание прямой подписи (S).

Шаг 1. Боб вычисляет хеш сообщения $[M] = 88$. Предположим, что он получил такой результат:

$$H(88) = 1\,305\,186\,650.$$

Используя хорошо известную нам функцию упрощения Mathematica Reduce, получаем секретный ключ шифрования $[x]$:

$$\text{Reduce}\left[\left(\text{mod}\left[K_A^b,\ p\right] + e_B\right) \cdot x == 1,\ x,\ \text{Modulus} \to p - 1\right];$$

$$x = 3009.$$

Боб вычисляет цифровую подпись (S), используя $H(m)$ в сочетании с $[x]$:

$$S \equiv 1\,305\,186\,650^{[3009]} \pmod{7919} = 7734.$$

Боб отправляет $(\boldsymbol{H(m)}, \boldsymbol{S}) = (\mathbf{1\,305\,186\,650}, \mathbf{7734})$ Алисе.

Шаг 2. Далее Алиса может проверить, действительно ли подпись (S) принадлежит Бобу.

Результат возведения значения подписи в степень [y] должен соответствовать $H(m)$ (mod p):

$$S^y \equiv H(m) \pmod{p}.$$

Как мы видели, [y] — это функция расшифрования, заданная открытым ключом Боба (K_B), возведенного в степень секретного ключа Алисы [a], с добавлением параметра [e_B]:

$$y \equiv \left\{\left[K_B^a + e_B\right] \pmod{p}\right\}.$$

Подставив числовые значения в уравнение, получаем:

$$y \equiv \{[4997^{123456} + 1] \pmod{7919}\} = 4955.$$

Алиса возводит (S) в степень [y]. В результате должно получиться значение, равное $H(m)$ (mod p):

$$V \equiv 7734^{4955} = 827 \pmod{7919}.$$

Это соответствует:

$$H(m) \pmod{p};$$

$$V' \equiv 1\,305\,186\,650 \pmod{7919} = 827.$$

Если $V = V'$, то Алиса принимает подпись. Она может проверить, что:

$$V = 827 = V'.$$

Как я уже рассказывал, существует еще один способ подписания и проверки подписи с помощью MBXI. Рассмотрим, как формируется подпись с дополнением.

Все параметры (модуль, закрытый и открытый ключи, образующий элемент и e_B) остаются прежними.

Боб выбирает случайное число [k], $k = 1529$.

Затем вычисляет (r):

$$r \equiv g^k \pmod{p} \equiv 7^{1529} \pmod{7919} = 4551.$$

Теперь Боб может поставить подпись с дополнением (s):

$$H(m) = 827 \pmod{p};$$

$$s \equiv H(m) \cdot [k + b] \pmod{p - 1} \equiv 827 \cdot [1529 + 543\,210] \pmod{7919 - 1} = 4543.$$

Боб отправляет ($H(m)$, s) = (827, 4543) Алисе.

Обратите внимание: здесь я вычислил модуль (p) от $H(m)$, чтобы дальше работать с малыми числами.

Алиса проверяет следующее:

- $V \equiv r^{H(m)} \cdot K_B^{H(m)} \pmod{p}$;
- $V \equiv 4551^{827} \cdot 4997^{827} \pmod{p} = 7147$;
- $V' \equiv g^s \pmod{p}$;
- $V' \equiv 7^{4543} \pmod{7919} = 7147$;
- $V = V'$.

Таким образом, Алиса принимает подпись (s).

Мы рассмотрели методы создания цифровой подписи в MBXI. В следующем разделе обсудим развитие MB09 и MBXI в сфере блокчейна.

Развитие MB09 и MBXI: введение в MBXX

В 2020 году я разработал и запатентовал новый протокол, объединивший в себе алгоритмы MB09 и MBXI.

По моему мнению, в своей статье Сатоши Накамото не смог дать исчерпывающее решение одной проблемы — *проблемы консенсуса*. Другая сложность (она также присутствует в MB09) заключалась в централизованности системы.

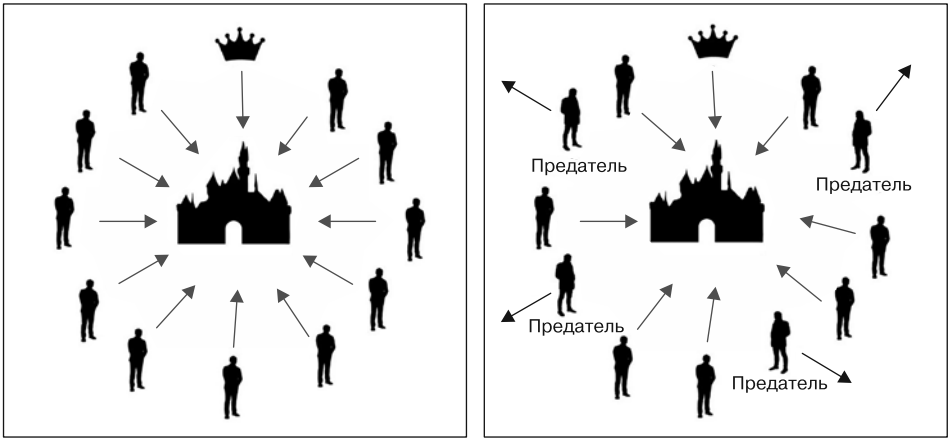
Я хотел решить эти проблемы с помощью реализации схемы, которая соответствовала следующим условиям.

- Протокол должен работать в децентрализованной модели.
- Консенсус о подлинности транзакций должен определяться математической детерминированной функцией, а не статистической вероятностью атаки.

Другими словами, проблема *двойного расходования* электронных денег должна быть решена криптографически, а не с помощью *консенсуса*, основанного на теории игр. Действительно, проблема консенсуса, выбранная Сатоши Накамото, основана на теории проблемы византийских генералов.

В описании был приведен пример с группой генералов, которые должны следовать приказу атаковать врага. Если в атаке участвуют больше трех генералов (генералы = узлы) и некоторые из них отказываются подчиняться приказу, это может привести к провалу всей операции. На рис. 6.11 можно увидеть разницу между согласованной и несогласованной атаками.

Сатоши Накамото использовал эту теорию, предполагая, что узлы системы (сети «Биткойн»), ответственные за контроль двойного расходования, проверяют подлинность транзакций, основываясь на проблеме консенсуса. Он предположил, что система может эффективно противостоять атакам, если *большинство генералов* будут действовать добросовестно.



Согласованная атака
с успешным результатом

Несогласованная атака
с провальным результатом

Рис. 6.11. Проблема византийских генералов

Не все знают, что в биткойне, как и в большинстве других криптовалют, майнеры, которые выполняют проверку транзакций через доказательство работы, не могут гарантировать безопасность системы, если количество благонадежных узлов составляет менее двух третей от их общего числа. Это означает, что для обеспечения защищенности системы необходимо, чтобы благонадежными были минимум 75 %, а не просто 51 % узлов.

Довольно странно, что Сатоши Накамото, разработав столь сложный протокол, в значительной степени основанный на криптографии для создания, передачи и оценки стоимости биткойна, выбрал недетерминированный метод для достижения *консенсуса* при проверке транзакций.

Проблема *консенсуса* Накамото не дает математических доказательств детерминированного результата. Как я уже сказал, она работает только при условии, что около 75 % узлов в системе благонадежны. Более того, если атака все же произойдет, это может привести к краху системы и все транзакции станут недействительными.

Доказательство работы (proof of work) — это метод, изобретенный Сатоши Накамото для добычи и передачи биткойнов. Он заключается в решении задачи, лежащей в основе алгоритма SHA-256 (мы изучали семейство алгоритмов SHA в главе 4).

Чтобы избежать проблемы двойной траты и поддерживать анонимные одноранговые транзакции, мы должны полагаться на математическое доказательство, которое дает детерминированную достоверность. Я представил себе систему, в которой частный блокчейн управляется исключительно компьютерами. Например, вычислительная сеть может почти полностью управлять *децентрализованной автономной организацией*.

Концепция децентрализованной автономной организации (Decentralized Autonomous Organization, DAO) была неизвестна или по крайней мере не использовалась до 2016 года. В данном контексте DAO рассматривается как организация, саморегулируемая компьютерной сетью или системой распределенных вычислений с высокой мощностью, которая способна совершать транзакции и запускать программы для достижения определенной цели. Такая система может поддерживаться искусственным интеллектом, принимающим некоторые решения. Основная цель заключается в создании децентрализованной организации, управляемой исключительно компьютерами, а не людьми. Точнее, роль человека сводится к написанию кода посредством так называемых интерактивных контрактов. При этом искусственный интеллект используется для принятия решений, но не для решения проблемы консенсуса.

Не имеет значения, кто владеет оборудованием или отвечает за его обслуживание. Это связано с тем, что организация представляет собой метаинфраструктуру, не имеющую центрального офиса, совета директоров, срока существования, а в будущем даже избавленная от вмешательства человека. Аппаратные машины, такие как виртуальные серверы, существуют в облаке и выполняют алгоритмы в виртуальной среде, где время и пространство не имеют традиционного смысла. Если отбросить проблемы, связанные с ответственностью такой организации, и философскую концепцию создания сущности, способной жить вечно и непрерывно, остается только чистая математика и логика.

Теоретически работу этой схемы невозможно прервать, потому что, если она будет хорошо развита, эта организация может быть распределена по виртуальной сети, полностью независимой от власти человека и регулируемой алгоритмами.

Теперь перейдем к самому протоколу MBXX и общей схеме его работы. На рис. 6.12 изображена децентрализованная схема, соединенная через несколько крупных центральных узлов-звезд.

Давайте подробнее изучим выполнение протокола для операций с электронными деньгами.

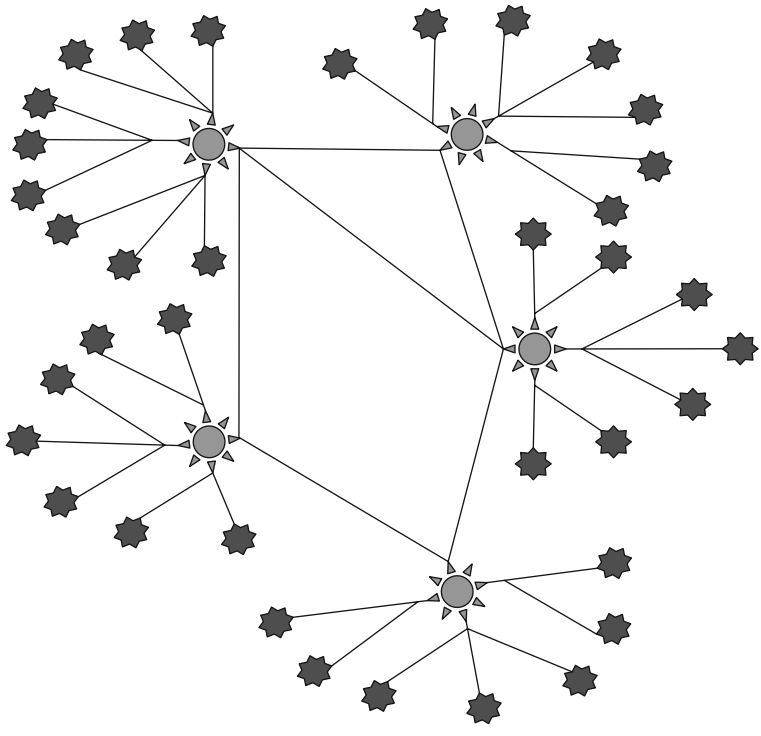


Рис. 6.12. Децентрализованная архитектура

Описание протокола MBXX

Схема протокола MBXX состоит из трех этапов, первый из которых — инициализация всех параметров. Инициализация принятых алгоритмов, параметров и ключей представляет собой условия *первого уровня*.

Этап инициализации. На этом этапе инициализируются все параметры:

- используемые в протоколе алгоритмы, такие как *метаалгоритм* (м/алг.), *алгоритм передачи* (п/алг.) и доказательство с нулевым разглашением (док. нул. разгл.): SHA-256, MB09, RSA, алгоритм Диффи — Хеллмана, MBXI, ZK13, zk-SNARK и др.;
- алгоритм цифровой подписи (по большей части зависит от выбранного алгоритма шифрования);
- параметры алгоритма — закрытые ключи пользователей [*a*], [*b*], [*c*]... [*n*] и соответствующие им открытые ключи (*A*, *B*, *C*... *N*);
- суммы электронных денег на счетах всех пользователей, от которых были получены закрытые ключи;

- образующий элемент (g);
- случайные числа [k_1, k_2... k_n] для создания протоколов доказательства с нулевым разглашением.

После завершения этого этапа приступаем к составлению протокола.

Шаг 1. Проверка обшей суммы с помощью метаалгоритма. Алгоритм первого уровня, или *метаалгоритм*, предназначен для проверки баланса до и после транзакции.

Не обязательно использовать систему, описанную в MB09, для выполнения гомоморфного баланса. Существуют более эффективные инструменты для этой цели. Однако для простоты объяснения я буду использовать соответствие MB09.

Вернемся к расширенной версии последней теоремы Ферма:

$$[a]^p + [b]^p + [c]^p + ... + [n]^p \equiv [z]^p \pmod{p};$$
$$A + B + C + ... + N = Z.$$

Это (м/алг.), который выполняет две функции:

- проверяет изоморфный баланс (Z) в определенный момент времени;
- восстанавливает равновесие системы при поступлении новых электронных денег (то есть сумма электронных денег, выраженных в изоморфном балансе в течение определенного времени t_0, t_1... t_n, меняется, поскольку начальный баланс (Z_0) увеличивается при поступлении новых электронных денег).

Другими словами, *изоморфное вычисление* представляет собой соответствие между закрытыми значениями [a^p, b^p... n^p] и их открытыми эквивалентами (A, B... N).

Итак, как мы уже выяснили, злоумышленнику будет очень сложно восстановить [a], даже если он знает соответствующее ему открытое значение (A). Это связано с тем, что данная функция *однонаправленная*. Таким образом, баланс счетов будет защищен с помощью соответствующего процесса дополнения.

Кроме того, вы можете использовать другую запись (м/алг.) для получения этого соответствия, например *хеш-систему*, где [a] соответствует (A) $\rightarrow$ H[a], или другую систему контрольных сумм.

Здесь [a, b, c... n] — это количество электронных денег в определенный момент времени: t_0, t_1, t_2... t_n. Каждый из этих элементов соответствует количеству денег, принадлежащих человеку или компьютеру, например Алисе, Бобу, Карлу и Нику. Каждому закрытому параметру (количеству электронных денег) [a], [b]... [n] соответствует открытый параметр (A), (B)... (N). Эти открытые параме-

тры представляют собой *изоморфные значения*, транспонированные скрытыми параметрами в публичное пространство.

Другими словами, [*a*] соответствует (*A*), [*b*] — (*B*)... [*n*] — (*N*).

Шаг 2. Транзакции. Транзакции между пользователями происходят в любой момент времени: t_0, t_1... t_n. Здесь [*M*] — это заданная сумма электронных денег, а [*Ma*, *Mb*... *Mn*] — суммы, передаваемые каждым участником сети в конкретный момент времени t_0, t_1... t_n. Например, мы можем считать [a_0] = 10 000 долларов начальным балансом (*A*), а [*Ma*] = 1500 долларов — суммой, переведенной со счета (*A*) на счет (*B*).

Обмен [*Ma*], [*Mb*]... [*Mn*] между пользователями выполняется с помощью криптографического алгоритма, который я назвал *алгоритмом передачи* (п/алг.). Он был создан на основе алгоритма с закрытым и открытым ключами, похожего на MBXI.

После первой транзакции, проведенной, например, между сторонами (*A*) и (*B*) в моменты времени (t_0) и (t_1), суммы на соответствующих счетах меняются с [a_0] и [b_0] на [a_1] и [b_1]. Таким образом, если начальный баланс *A* оставляет 10 000 долларов и сторона (*A*) переводит [*Ma*] = 1500 долларов стороне (*B*), то в момент времени (t_1) баланс (*A*) будет равен:

$$(t_0)\ [a_0] = 10\,000;$$
$$(t_1)\ [a_1] = a_0 - Ma;$$
$$(t_1)\ [a_1] = 10\,000 - 1500 = 8500 \text{ долларов.}$$

Идентификация отправителя должна быть выполнена с помощью цифровой подписи, созданной в (п/алг.).

В нашем случае Алиса добавляет свою цифровую подпись, используя MBXI:

$$S_A \equiv (H[Ma])^x \pmod{p}.$$

Таким образом, (S_A) — это цифровая подпись Алисы, которая была получена с применением хеша сообщения (*H[Ma]*), возведенного в степень ключа шифрования [*x*], где (*H[Ma]*) задается следующей функцией: (*H[Ma]*) = хеш сообщения [*M*].

Как вы помните, закрытый ключ [*x*] получается решением следующего уравнения:

$$\left\{\left[K_A^b + e_B\right] \pmod{p}\right\} x \equiv 1 \pmod{p-1}.$$

Здесь подпись (S_A), переданная вместе с криптограммой (*c*), будет проверена стороной (*B*) обратным способом:

$$(S_A)^y = (H[Ma]) \pmod{p}.$$

После того как Боб убедился, что полученный ($H[Ma]$) соответствует хешу от [M], он принимает подпись и получает платеж.

Шаг 3. Проверка. На этом этапе все транзакции проверяются администратором системы. Процесс может быть реализован автономной организацией, подобной DAO.

Первую проверку система проведет на *гомоморфном балансе*, чтобы удостовериться в отсутствии двойного расходования при расчетах. Если все в порядке, то после совершения транзакций (одноранговых операций через п/алг.) администратор (а лучше несколько администраторов, поскольку система децентрализована) приступает к *слепой проверке* счетов отправителя и получателя, чтобы подтвердить, что на счетах находится правильное количество средств. Проверка будет осуществляться с помощью определенного протокола доказательства с нулевым разглашением.

Доказательство с нулевым разглашением позволяет обнаружить, не пытается ли пользователь обмануть систему и установить двойное расходование, отправив фальшивый параметр проверки. В этом случае пользователь может попытаться совершить платеж, но его баланс не покрывает обрабатываемую сумму. Доказательство с нулевым разглашением позволяет автоматически распознать проблему и отказать в транзакции без какого-либо метода консенсуса, основанного на доказательстве работы или других доказательствах. На рис. 6.13 можно увидеть схему первой транзакции, совершенной в момент времени (t_0–t_1) с помощью протокола MBXX.

Обратите внимание на знак – в скобках (t_0–t_1). Он обозначает не вычитание, а переход от t_0 к t_1.

Если в момент времени (t_1) не будет произведено никаких других транзакций, то протокол будет продолжать работать с заданными метками времени (t_2), (t_3) и т. д.

Детали протокола MBXX

В этом разделе представлено еще больше информации о протоколе MBXX. Здесь мы сосредоточимся на его конкретных функциях.

На рис. 6.13 изображена функция (п/алг.(t_0–t_1)), которая выражает все операции, выполняемые между пользователями в момент времени (t_0–t_1). Она выглядит следующим образом:

$$[a_1] = \{[a_0] \pm [Ma]\} \sim \{[b_1] = [b_0] \pm [Mb]\};$$
$$A_1 = A_0 - (H[Ma]) \text{ или } A_0 + (H[Mb]);$$
$$B_1 = B_0 - (H[Mb]) \text{ или } B_0 + (H[Ma]).$$

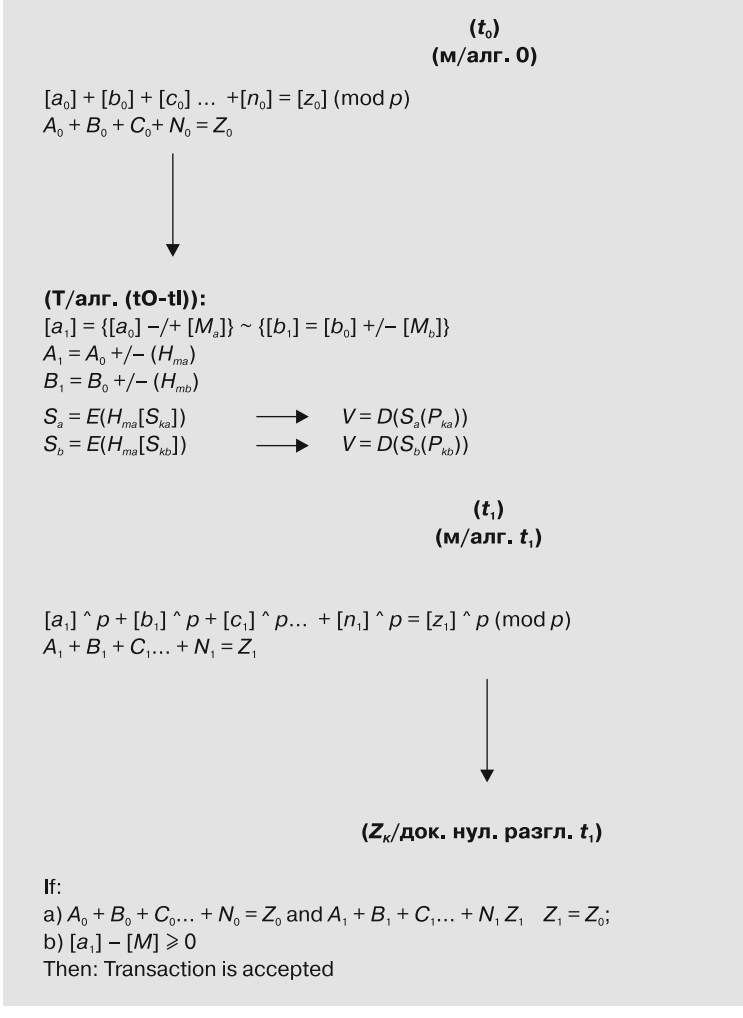


Рис. 6.13. Схема первого этапа протокола MBXX[1]

Таким образом, секретная сумма $[a_1]$ формируется из суммы $[a_0]$, прибавленной к сумме $[Ma]$ или отнятой от нее, что соответствует значению $[b_1]$, сформированному из переданной и полученной суммы $[Mb]$. После этой операции на основании (A_0) генерируется открытый параметр (A_1). (A_0) — это параметр

[1] Sa — подпись Алисы, Hma — хеш сообщения M от Алисы, Ska — секретный ключ Алисы, E — функция создания подписи, V — проверка подписи, D — функция расшифрования (decryption) для проверки подписи с использованием публичного ключа, Pka — публичный ключ Алисы.

предыдущей операции, вычитаемый из (*H[Ma]*), если сумма [*M*] была получена из (*B*). То же самое относится и к параметру (B_1), который генерируется предыдущим параметром (B_0), прибавленным к хешу исходной суммы (она передана параметру (*A*) или получена из него) или отнятым от него.

Эта схема (Z_0) представляет собой количество денег, находящихся в обращении в системе на данный момент (t_0). Она не изменится, если в систему не будут поступать электронные деньги. Если происходит транзакция, то изменяются соответствующие значения (параметры) пользователей, участвующих в транзакции.

Как мы видели на шаге 2, в *процессе транзакции* используется любой алгоритм с открытым и закрытым ключами. Это легко продемонстрировать на примере MBXI, так как система хорошо работает. Поскольку суммы электронных денег передаются непосредственно от пользователя к пользователю, можно говорить об эффекте одноранговой системы, где в транзакциях не участвует третья сторона. Таким образом, теоретически для осуществления транзакций не требуется никакого финансового или банковского учреждения.

Отправитель добавляет цифровую подпись (*S*). Получатель проверяет и принимает ее, если при расшифровке подписи (*V*) получается хеш [*M*] = (*Hm*). Это было продемонстрировано на рис. 6.13, представляющем (п/алг.):

$$S_A = E(Hma[Ska]) \rightarrow V = D(S_A(Pka));$$
$$S_B = E(Hmb[Skb]) \rightarrow V = D(S_B(Pkb)).$$

Здесь (S_A) — подпись Алисы, (S_B) — подпись Боба.

Последний шаг — проверка, и он связан с доказательством с нулевым разглашением. Первая проверка выполняется следующим образом:

$$A_0 + B_0 + C_0 + ... + N_0 = Z_0;$$
$$A_1 + B_1 + C_1 + ... + N_1 = Z_1.$$

Достоверность подтверждается, если получен результат $Z_1 = Z_0$.

Это означает, что открытый параметр (Z_1), являющийся результатом *гомоморфного баланса*, относящегося к моменту времени (t_1), соответствует (Z_0), относящемуся к моменту времени (t_0), даже если изменились значения частичных элементов уравнения. Это условие гарантирует (при отсутствии поступления денег в систему между t_0 и t_1), что сетевая передача денег между пользователями является нейтральной или сбалансированной.

Иными словами, суммы транзакций, совершенных пользователями системы, сбалансированы. Однако проверка этого условия сама по себе не гарантирует отсутствия двойного расходования средств. На самом деле оно вполне возможно, если стороны, работающие в системе, неблагонадежны.

Следовательно, здесь необходимо реализовать *слепую проверку* или *гомоморфную проверку* для подтверждения того, что суммы на отдельных счетах не принимают отрицательных значений. Если это условие также выполняется, то система (DAO) будет гарантировать отсутствие двойного расходования.

Наконец, можно сказать, что обязательным условием для удовлетворения требований системы является нейтральный гомоморфный баланс. Условие, которое необходимо проверить, заключается в том, что любой отдельно взятый баланс никогда не принимает отрицательного значения.

Выводы по протоколу MBXX

MBXX должен преодолеть две проблемы, обозначенные в начале этого раздела.

1. Протокол работает в децентрализованном автономном режиме без участия в транзакциях третьих лиц.
2. Консенсус в отношении подлинности транзакций задается математической функцией, а не статистической вероятностью атаки.

Что касается первой задачи — *децентрализованного автономного режима*, то в MBXX транзакции совершаются посредством криптографического алгоритма, выполняемого между пользователями в одноранговой системе. Более того, проверяющий (компьютер) может заблокировать транзакцию только в том случае, если проверка не завершена.

Проверяющий не знает количества переведенных денег и количества отдельных счетов. Ему известны только первоначальная сумма денег, поступивших в систему, и наличие нарушений баланса.

Что касается второй проблемы — *консенсуса*, то можно продемонстрировать, что ее может решить чистая криптографическая модель, основанная на проверке изоморфного баланса, представленного (м/алг.), и доказательстве с нулевым разглашением. С помощью слепого сравнения доказательство с нулевым разглашением определяет, является ли баланс отдельного счета положительным или отрицательным, так как принимает только положительные балансы.

В ходе работы с узлами проверки может возникнуть вопрос о том, какой блокчейн использовать — открытый или эксклюзивный. На мой взгляд, все зависит от реализации. Давайте не будем забегать далеко вперед: изучить математическую основу протокола и проверять проблему консенсуса — это сложное испытание!

Познакомившись с некоторыми протоколами шифрования с открытым и закрытым ключами, перейдем к интересному алгоритму симметричного шифрования, связанному с новым методом, который назван *легковесным*, или *малоресурсным*, *шифрованием* и особенно часто используется в IoT *(Интернет вещей)* и для потоковой передачи данных.

Легковесное шифрование

Алгоритмы легковесного, или малоресурсного, шифрования (Lightweight Encryption Algorithms, LEA, или просто LE) представляют собой новую группу криптографических алгоритмов с минимальными требованиями к ресурсам и очень низким энергопотреблением. Легковесное шифрование ориентировано на широкий спектр устройств с ограниченными ресурсами, таких как конечные узлы IoT и RFID-метки, и может быть реализовано как в программном, так и в аппаратном обеспечении для работы с различными коммуникационными технологиями.

Создание легковесного шифрования было обусловлено необходимостью минимизировать использование памяти, вычислительных ресурсов и энергопотребления, обеспечивая при этом надежные решения для устройств с ограниченными ресурсами. Предполагается, что легковесная криптография должна оптимизировать затраты на реализацию, скорость, безопасность, производительность и энергопотребление. У таких алгоритмов низкие вычислительные требования, и они потребляют меньше ресурсов при обработке данных, особенно в сфере IoT.

В современном мире передачи и хранения данных IoT будет играть важнейшую роль для общества. Он представляет собой сложную систему, связывающую автомобили и мобильные устройства, управляющую домашними системами и контролирующую инженерно-техническое обеспечение крупных компаний.

Такая чувствительная и критически важная система требует одновременного соблюдения трех основных условий.

- Данные должны передаваться на высокой скорости.
- Вычислительная нагрузка и требования к памяти при шифровании должны быть минимизированы для обеспечения эффективной работы на IoT-устройствах с малым объемом ресурсов и батареями.
- Должны быть обеспечены безопасность и защита передаваемых сообщений, которые нередко содержат конфиденциальную информацию.

Чтобы соответствовать этим требованиям, необходимо найти шифрование, которое будет быстрым, но при этом безопасным.

Рассмотрим новый алгоритм, который привлек мое внимание.

Алгоритм Cybpher

Здесь я предлагаю рассмотреть новый алгоритм, экспертную оценку которого делал в 2022 году. Cybpher — это симметричный алгоритм, разработанный польско-американской компанией CyberusLabs. Он основан на использовании одноразовых шифроблокнотов (one-time pad, OTP). Cybpher работает быстрее

других алгоритмов симметричного шифрования, поскольку его математические операции просты и требуют немного времени на обработку. Он использует минимальное количество памяти, и в нем очень мало вычислительных циклов. Этот алгоритм можно считать безопасным, поскольку он вдохновлен одним из самых стойких методов шифрования — побитовыми транспозициями. Однако, как было показано в этой книге, ни один алгоритм, основанный на классических математических операциях, нельзя считать полностью безопасным.

Общую схему шифрования и расшифрования LE Cybpher можно представить следующим образом (рис. 6.14).

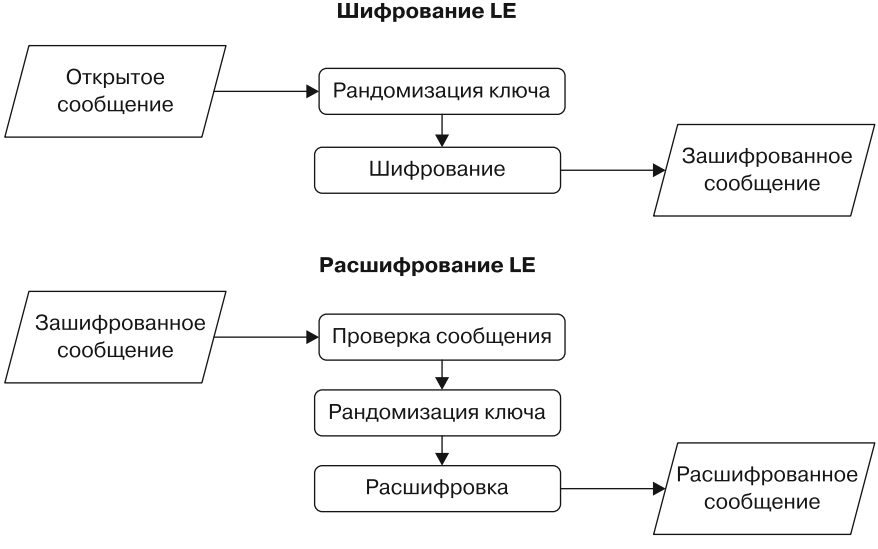


Рис. 6.14. Методы шифрования и расшифрования LE Cybpher

Минимальные требования к процессору — 8 бит, к памяти — чуть более 2 Кбайт.

Логика Cybpher проста, но простота не означает хрупкость. Напротив, в криптографии сложность может скрывать уязвимости. Анализируя исторические криптографические алгоритмы, можно увидеть, что чем сложнее алгоритм, тем выше вероятность появления в нем слабых мест. Принцип Керкхоффа гласит: «Самое слабое звено ломает всю цепь». Иными словами, сложность — это враг безопасности, так как чем длиннее цепь, тем выше вероятность ее разрыва. На данный момент самым безопасным, теоретически неразрушимым в классической криптографии остается алгоритм Вернама. Как мы говорили в главе 2, этот алгоритм использует только одну математическую операцию — XOR (исключающее ИЛИ). Напомню, что операция XOR является булевым оператором, который выражает побитовую сумму между открытым текстом и случайным ключом той же длины.

Алгоритм Cybpher в значительной степени основан на булевых логических операциях. Однако у него есть три ключевых отличия от алгоритма Вернама.

- Ключ изменяется в каждой сессии, генерируясь автоматически.
- Алгоритм способен генерировать бесконечное число ключей.
- Алгоритм шифрует бесконечные объемы данных, начиная с одного входного ключа.

Cybpher использует SWAP-функцию для создания новых ключей шифрования. Каждый новый ключ формируется на основе предыдущего ключа и текущей передачи данных, связывая отправителя и получателя посредством записи всей истории передачи данных между устройствами в текущий ключ.

Этот процесс соединяет устройства отправителя и получателя и создает взаимно аутентифицированную пару устройств. Таким образом, в Cybpher активный ключ, совместно используемый двумя или более IoT-устройствами, сначала обеспечивает аутентификацию для контроля точного соответствия выбранному устройству, а затем безопасность при передаче данных.

Мы проанализируем различия между Cybpher и другими симметричными алгоритмами далее в этой главе. А сейчас примем как факт то, что Cybpher может преодолеть проблему передачи небольших сообщений без использования какой-либо особой техники удлинения. Cybpher хорошо справляется с шифрованием даже очень коротких сообщений, типичных для IoT.

Вам может быть интересно, как такая система будет исправлять ошибки передачи. Это хороший вопрос для этапа реализации. Однако мы ограничимся обсуждением алгоритма с теоретической точки зрения.

Продемонстрирую работу алгоритма на этапе шифрования одной буквы.

Шифрование в Cybpher

Рассмотрим процесс шифрования Cybpher на примере слова hello. Алгоритм шифрует текст посимвольно. Мы возьмем первую букву, h, затем создадим новый ключ для буквы e и т. д. Я покажу обработку первой буквы и генерирование первого и второго ключей. Все остальные шаги будут идентичны им.

Зашифруем символ *h*, учитывая, что в ASCII-кодировке он соответствует значению 104.

Шаг 1. Создаем буфер ключей, перемешав 256 чисел (от 0 до 255) случайным образом.

Буфер ключей (до шифрования):

210 50 213 113 21 239 59 11 164 7 158 172 24 176 68 236 80 215 70 153 139 207 42 223 101 136 78 237 147 132 151 150 127 0 232 233 196 133 85 26 154 222 123 227 209 180 145 225 118 117 126 49 192 19 128 74 111 52 37 30 160 187 39 130 69 46

98 200 81 211 23 142 177 212 189 246 114 71 148 106 61 58 131 152 203 115 156 90 185 205 124 83 103 17 29 108 137 182 165 144 161 67 13 251 97 16 62 107 170 228 149 125 173 234 197 143 167 159 169 32 208 250 63 235 243 193 18 253 178 244 195 254 229 84 226 65 140 8 247 79 186 166 241 76 201 206 48 224 94 15 168 55 220 45 174 109 99 60 34 194 102 146 110 135 44 245 240 12 119 217 157 36 216 28 14 238 41 73 93 91 219 255 25 47 54 198 214 202 191 75 1 96 86 9 138 188 4 104 35 33 27 120 181 199 100 155 179 20 230 6 221 88 112 95 92 134 72 31 231 190 183 163 51 57 248 40 66 10 38 89 105 218 171 121 122 2 204 53 87 129 252 43 184 3 56 64 242 141 249 162 77 116 82 175 22 5

Шаг 2. Случайным образом выбираем число x, которое станет первым ключом (OTP) из созданного буфера. Допустим, $x = 210$, берем 210-й элемент в буфере ключей и записываем $x_1 = 221$ (это значение 210-го элемента в буфере ключей).

Шаг 3. Сгенерируем параметр y, сложив два числа, $x + x_1 \pmod{256}$: $210 + 221 = 175 \pmod{256}$. После чего найдем соответствующее значение для позиции y в другом буфере (буфере смещения), тоже созданном перемешиванием 256 чисел случайным образом. В результате мы определяем, что значение, соответствующее 175 в буфере смещения, — это $K_1 = 6$.

Буфер смещения:

85 29 207 24 82 164 118 35 240 249 248 132 205 126 194 138 20 45 4 162 243 250 7 101 163 222 169 106 211 103 25 43 137 80 14 176 33 47 3 144 86 244 54 107 64 152 220 247 56 44 181 16 251 156 1 175 214 23 239 59 149 13 112 121 238 32 134 210 196 135 102 136 198 52 202 161 108 229 19 216 209 232 120 223 110 204 2 34 26 36 245 70 97 140 30 68 119 113 206 60 96 151 21 200 84 124 81 87 109 95 237 218 31 174 153 187 197 65 46 10 203 122 69 127 182 98 183 159 180 67 147 104 221 166 217 235 0 78 129 39 133 226 79 141 72 49 228 66 94 173 9 139 233 208 190 12 252 105 185 99 155 5 179 177 167 40 77 158 227 88 125 117 37 234 55 6 165 212 199 246 157 15 111 253 143 22 38 172 188 62 28 213 8 91 128 131 142 17 186 191 145 83 224 189 93 192 53 11 170 63 115 123 242 130 76 42 74 90 114 184 231 18 148 41 254 73 201 57 178 255 195 61 241 219 58 89 150 27 236 71 146 230 75 154 171 215 116 51 160 225 92 48 193 100 168 50

Шаг 4. Шифрование символа $h = 104$.

Выполняем операцию XOR между ключом K_1 и открытым текстом h:

$$K_1 \oplus h = c;$$
$$6 \oplus 104 = 110.$$

Затем находим соответствующее число в позиции 110 в буфере ключей и заменяем 110 на 149. Это и будет криптограммой $h = 149$:

$$110 \rightarrow 149 \text{ [в буфере ключей]};$$
$$c = 149 \text{ [шифрование буквы h]}.$$

Теперь наступает самое интересное — генерирование *новой сессии ключей* (для шифрования следующей буквы — e).

Для этого необходимо взять последнее число из буфера ключей:

$$z = 5;$$
$$\text{SWAP } 149 \otimes 5.$$

Заменить число 5 шифротекстом $c = 149$.

Это будет следующий сеансовый ключ $K_2 = 149$.

Тестирование производительности Cybpher

Далее мы проанализируем тесты Cybpher, проведенные для сравнения с другими алгоритмами. В частности, мы решили сравнить скорость Cybpher со стандартными алгоритмами симметричного шифрования AES 256 и AES 128.

Тесты проводились компанией CyberusLabs (владельцем Cybpher) на двух платформах: во встраиваемой системе с 32-битным процессором ARM Cortex A8 на ОС Debian 7.11 и в 64-битной настольной системе Intel i7 на ОС Ubuntu 18.04. Стоит отметить, что на момент написания статьи эти тесты еще не были проверены независимыми экспертами.

Проверка включала в себя несколько разных типов и размеров входных данных, а также различные итерации обработки в диапазоне от 1 до 1000.

На рис. 6.15 приведены результаты тестов на процессоре Intel i7. Cybpher сравнивался с AES 128 и AES 256 (стандартами симметричного шифрования).

Алгоритм	Параметры компилятора	Размер входных данных, байт	Итерации	Среднее время, мс		Общее время обработки, мс	Пропускная способность, Мбит/с
				Шифрования	Расшифровки		
Cybpher	DNDEBUG-03	1 048 576 (1 Мбайт)	1	2,2	1,6	3,8	2105,26
AES 128	DNDEBUG-03	1 048 576 (1 Мбайт)	1	5,2	5,2	10,4	769,23
AES 256	DNDEBUG-03	1 048 576 (1 Мбайт)	1	6,5	6,5	13,0	615,38
Cybpher	DNDEBUG-03	1 048 576 (1 Мбайт)	1000	2221	1569	3790,0	2110,82
AES 128	DNDEBUG-03	1 048 576 (1 Мбайт)	1000	5254	5207	10 461,0	764,75
AES 256	DNDEBUG-03	1 048 576 (1 Мбайт)	1000	6504	6540	13 044,0	613,31

Рис. 6.15. Время выполнения операции LE Cybpher по сравнению с AES 128 и AES 256

Если не указано иное, значения пропускной способности приводятся в мегабитах в секунду (Мбит/с).

Кроме того, была измерена энтропия системы. На рис. 6.16 показана энтропия шифрования, выполненного с помощью Cybpher.

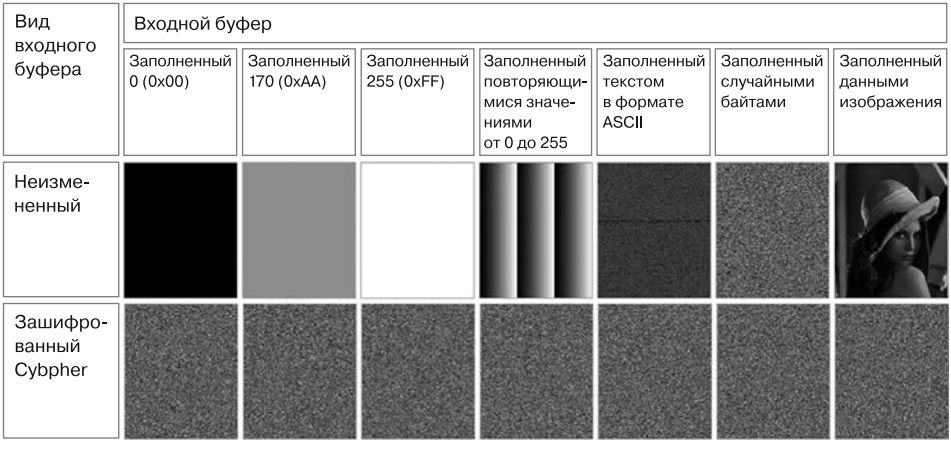


Рис. 6.16. Энтропия шифрования на различных входных данных

Легковесное шифрование — это новый вид шифрования, основанный на высокой скорости работы и потребляющих небольшую мощность вычислениях. Оно в основном используется в IoT и других средах, где для обработки и передачи информации применяются мини-процессоры.

Cybpher представляет новый вид LEA, который может стать стандартом в мире алгоритмов LE. Также я представил техническую базу, архитектуру и пример реализации этого алгоритма. Результаты тестов продемонстрировали, насколько Cybpher быстр и прост в вычислениях.

На данный момент Cybpher можно считать устойчивым к атакам методом перебора и атакам на открытый текст, если будет соблюдаться принцип случайности при создании буфера ключей и буфера смещения. Однако время покажет, насколько эта новая для всех нас технология действительно безопасна.

Резюме

В этой главе мы проанализировали некоторые из моих изобретений. Эти алгоритмы и протоколы были придуманы в первую очередь для решения проблем, связанных с системами шифрования с использованием открытого и закрытого ключей.

Сначала мы рассмотрели MB09, который был призван стать стандартом для обеспечения безопасности электронных платежей, производимых в сфере дистанционной передачи данных.

Затем был описан MBXI — алгоритм шифрования с открытым и закрытым ключами, реализованный в качестве альтернативы RSA. Мы рассмотрели способы использования цифровой подписи с помощью этого алгоритма, а также различные типы подписи: прямую и с дополнением.

Последним был представлен MBXX. Соединив в себе MB09 и MBXI, этот протокол направлен на преодоление проблемы двойного расходования средств и так называемой проблемы консенсуса. Он может быть использован в будущем в качестве альтернативного метода доказательства работы (или доказательства доли, или других доказательств) в децентрализованной платежной системе для подтверждения криптовалютных транзакций, основанного на статистической точности.

Таким образом, вы узнали о новых методах и системах шифрования с открытым и закрытым ключами — MB09, MBXI и MBXX, а также познакомились с некоторыми схемами в централизованных и децентрализованных криптосистемах, связанных с новой эрой цифровой валюты.

Наконец, вы познакомились с легковесным шифрованием — методом шифрования, применяемым для дистанционной передачи данных в среде IoT.

7 Эллиптические кривые

Эллиптические кривые — это новый рубеж сферы децентрализованных финансов. Сатоши Накамото выбрал особый тип эллиптической кривой для реализации передачи электронной валюты — secp256k1. *Эллиптическая криптография* (elliptic curve cryptography, ECC) широко используется во встроенных устройствах благодаря меньшему размеру ключей и сниженным требованиям к вычислительным ресурсам при операциях с закрытым ключом. Давайте разберемся, что представляет собой и как работает этот очень надежный метод шифрования.

В данной главе мы рассмотрим математические основы эллиптической криптографии. Я представлю основные принципы, но стоит понимать, что шифрование на эллиптических кривых — сложная тема, включающая в себя геометрию, модульную арифметику, цифровые подписи и логику.

Математическую часть эллиптической криптографии я покажу на примере использования secp256k1 для создания цифровой подписи в биткойне.

Наконец, мы обсудим возможность атак на эллиптические кривые.

Обзор эллиптических кривых

Примерно в 1985 году Виктор Миллер и Нил Коблиц впервые предложили задействовать *эллиптические кривые* для криптографических целей. Позже Хендрик Ленстра показал, как их можно использовать для факторизации целых чисел.

Эллиптические кривые — это геометрическое представление особых математических равенств на координатной плоскости. Мы начнем их изучение с двумерного пространства, но стоит отметить, что более сложные представления охватывают трех- и четырехмерное пространство, включая иррациональные и мнимые числа. Однако сейчас не стоит об этом думать — со временем все станет понятнее.

Эллиптическая криптография используется как альтернатива алгоритмам асимметричного шифрования, таким как RSA и ElGamal. Вы также увидите, что с помощью эллиптических кривых можно реализовать обмен ключами Диффи — Хеллмана.

Кроме того, после революционного рывка в разработке криптовалют кривая secp256k1 и *алгоритм цифровой подписи на эллиптических кривых* (Elliptic Curve Digital Signature Algorithm, ECDSA) стали ключевыми технологиями в «Биткойне», обеспечивающими безопасность транзакций.

Эта глава шаг за шагом познакомит вас с логикой эллиптических кривых и их применением в цифровом мире.

Предполагается, что 313-битное шифрование на эллиптических кривых обеспечивает тот же уровень безопасности, что и 4096-битное шифрование в традиционных асимметричных системах (см. www.keylength.com). Такой небольшой размер ключей удобен для многих приложений, требующих высокой производительности, например для мобильных устройств.

Давайте разберем основы эллиптических кривых, включая их геометрические, алгебраические и логические свойства.

Операции, выполняемые на эллиптических кривых

Первое наблюдение: эллиптическая кривая не является *эллипсом*. Она представлена следующей математической формой:

$$E: y^2 = x^3 + ax^2 + bx + c.$$

E представляет собой форму эллиптической кривой, а параметры (*a*, *b* и *c*) являются коэффициентами кривой.

Для наглядности попробуем построить график по уравнению кривой:

$$E: y^2 = x^3 + 73.$$

На рис. 7.1 изображен график этой эллиптической кривой в ее геометрической форме, которую я построил с помощью программы WolframAlpha.

Теперь перейдем к анализу функциональных, геометрических и алгебраических свойств эллиптических кривых, а также их преимуществ. Поскольку кривые нелинейны, их легко реализовать в криптографических целях, что делает их адаптируемыми.

Кривая	$y^2 = x^3 + 73$

Предполагаемый график

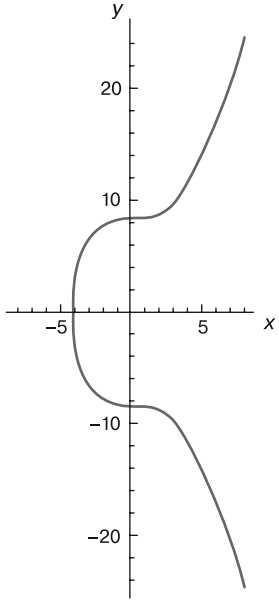


Рис. 7.1. Эллиптическая кривая E: $y^2 = x^3 + 73$

В качестве примера рассмотрим кривую, график которой только что построили:

$$E: y^2 = x^3 + 73.$$

Когда ($y = 0$), кривая пересекает ось X в точке ($x = -4{,}1793...$). Математически это значение определяется подстановкой $y = 0$ в уравнение:

$$y^2 = x^3 + 73.$$

Таким образом, один из трех корней кривой является кубическим корнем из –73.

Когда ($x = 0$), кривая пересекает ось Y в двух точках ($y = \pm 8{,}544003...$). Математически это значение определяется подстановкой $x = 0$ в уравнение:

$$y^2 = x^3 + 73.$$

В результате получается точка пересечения кривой с осью Y, как показано на рис. 7.1.

У эллиптических кривых есть еще одна интересная особенность. Если кривая проходит через две точки, симметрично расположенные относительно оси *Y*, значит, существует третья точка *O*, которую можно представить как бесконечно удаленную точку, *лежащую* на неопределяемом конце оси *Y*. Мы поговорим о ней позже, когда будем обсуждать сложение точек кривой.

Одно из самых интересных свойств эллиптических кривых — это значение суммы двух точек кривой. Здесь мы определяем операцию сложения на самой кривой.

Допустим, нам нужно сложить между собой две точки кривой, *P* и *Q*. Если провести между ними прямую, то она пересечет кривую в третьей точке, $-R$. Затем, если зеркально отразить точку $-R$ относительно оси *X*, то получится еще одна точка R — *это и есть сумма P и Q*.

Следующий график поможет понять геометрическое представление суммы (рис. 7.2).

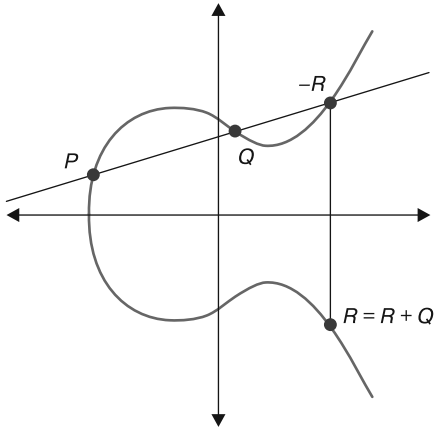


Рис. 7.2. Сложение и удвоение точек на эллиптической кривой

Теперь посмотрим на это сложение с точки зрения алгебры.

Сначала возьмем координаты *P* и *Q* и выполним следующее вычисление (*s*):

$$\boldsymbol{s} = (y_P - y_Q) / (x_P - x_Q).$$

Вычислить x_R, то есть координату точки *R* по оси *X*, можно с помощью операции:

$$\boldsymbol{x}_R = s^2 - (x_P + x_Q).$$

Вычислить y_R, то есть координату точки *R* по оси *Y*, можно с помощью операции:

$$\boldsymbol{y}_R = s(x_P - x_R) - y_P.$$

А как насчет вычисления $P + P = R = 2P$ — так называемого *удвоения точки*?

Как показано на рис. 7.3, геометрическое представление точки P, сложенной с самой собой так, что получается значение $2P$, задается касательной, проходящей через точку P и пересекающей кривую в точке R'. Если отразить эту точку суммы, то получится точка R.

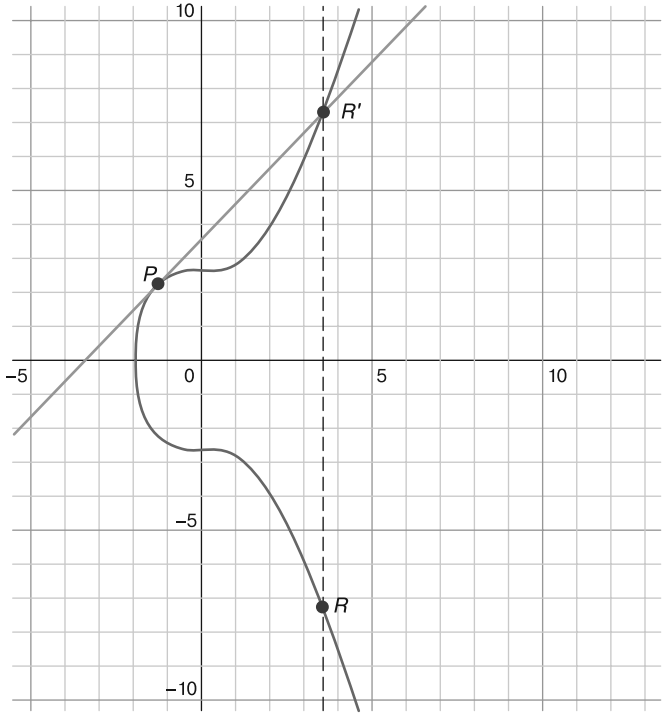


Рис. 7.3. Удвоенная точка $2P$

Это очень похоже на вычисление суммы точки, представленное в предыдущем примере.

Проводим касательную линию в точке P, находим точку пересечения прямой с кривой в точке $R' = (-2P)$. Точка на кривой, отраженная относительно оси X, и является удвоенной точкой P, то есть $R = (2P)$.

Как вычислить $P + P$ алгебраически? В этом случае (t) будет иметь следующий вид:

$$\boldsymbol{t} = \left(3x_P^2 + a\right) \Big/ 2y_P,$$

Помните, что (a) является параметром кривой.

Таким образом, чтобы найти координату точки R на оси X, мы должны вычислить:

$$x_R = t^2 - 2x_P.$$

Найти координаты точки R на оси Y можно с помощью уравнения:

$$y_R = t(x_P - x_R) - y_P.$$

Есть еще один сценарий, связанный с эллиптическими кривыми, определяющий, как складывать точки по вертикали.

Здесь O представлена бесконечно удаленной точкой, заданной значением суммы P и Q, если $x_P = x_Q$. Как показано на рис. 7.4, прямая, полученная в результате соединения P и Q, параллельна оси Y.

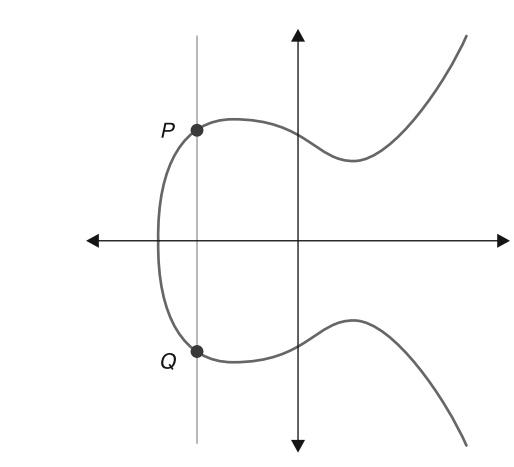


Рис. 7.4. Сложение точек по вертикали

Алгебраически бесконечно удаленная точка задается как:

$$P + Q = \boldsymbol{O}, \text{ если } x_P = x_Q,$$

или:

$$P + P = \boldsymbol{O}, \text{ если } x_P = 0.$$

В качестве примера предположим, что мы складываем две точки, A и B, которые определяются следующим образом:

$$A = (x_A; y_A);$$

$$B = (x_B; y_B).$$

Мы можем сложить эти точки, используя следующие уравнения:

$$x_C = \left(\frac{y_B - y_A}{x_B - x_A}\right)^2 - x_A - x_B;$$

$$y_C = \left(\frac{y_B - y_A}{x_B - x_A}\right)(x_A - x_C) - y_A.$$

Если вместо сложения A и B необходимо удвоение точки, $2A = A + A$, это можно сделать следующим образом:

$$x_{2A} = \left(\frac{3x_A^2}{2y_A}\right)^2 - 2x_A;$$

$$y_{2A} = \left(\frac{3x_A^2}{2y_A}\right)(x_A - x_{2A}) - y_A.$$

Теперь, когда вы увидели, как складывать и удваивать точки, посмотрим, как выполняется другая важная операция криптографии на эллиптических кривых — скалярное умножение.

Скалярное умножение

На этом этапе вы должны познакомиться еще с одной операцией, характерной для эллиптических кривых, — со скалярным умножением. Этот процесс математически представляет собой сумму P и Q и позволяет вычислять $2P$, $3P$... nP на эллиптических кривых.

Логика скалярного умножения несложная, но чтобы лучше понять ее, нужен практический пример.

Давайте выполним умножение на эллиптической кривой:

$$Q = n \cdot P.$$

Это равенство называется *многократным сложением*, и его можно представить следующим образом:

$$Q = \{P + P + P + P + ... + P\}$$

n раз.

Поскольку нам нужно сложить координаты точки с самими собой, мы можем представить скалярное умножение как сложение точки с самой собой n раз:

$$P + P = 2P.$$

Как мы уже видели, это формула удвоения точки (она геометрически представлена на рис. 7.3), перенесенная в алгебраическое представление значения суммы.

Всегда помните, что мы имеем дело с кривой, поэтому координаты новой точки пересечения будут следующими:

$$2P(x_{2P}, y_{2P}).$$

По той же логике можно получить $3P$, $4P$... $7P$. Начнем с $3P$:

$$R = 3P = 2P + P.$$

Далее следует $4P$:

$$R = 4P = 2P + 2P.$$

Затем $5P$:

$$R = 5P = 2P + 2P + P.$$

Затем $6P$:

$$R = 6P = 2P + 2P + 2P.$$

$6P$ можно задать и как:

$$R = 2(3P) = 2(2P + P).$$

Наконец, определяем $7P$:

$$R = 7P = P + (P + (P + (P + (P + (P + (P)))))).$$

В то же время $7P$ можно задать и как:

$$R = 7P = P + 6P = P + 2(3P) = P + 2(P + 2P).$$

Хотя все представленные операции называются умножением, по сути, скалярное умножение — это разложение числа R на скалярные составляющие, которое будет постепенно сводиться к минимальной единице P. Так, например, $7P$ становится умножением P и $2P$.

По тому же принципу умножение числа K на P требует разложения P на простейшие компоненты — например, на сумму $P + P$ или удвоенную точку $2P$, которые в итоге дадут нужный результат. Как показано на рис. 7.4, все вычисления строятся на операциях сложения и умножения.

Например, если разложить $9P$, то мы получим:

$$9P = 2(3P) + 3P = 2(2P + P) + 2P + P = P + 2P + 2(2P + P).$$

Теперь, когда вы поняли суть скалярного умножения, давайте выясним, почему оно так важно для создания системы шифрования на эллиптических кривых.

Как вы видели в предыдущих главах (например, когда мы говорили об алгоритме Диффи — Хеллмана в главе 3), в криптографии мы опираемся на *односторонние* функции, которые позволяют легко производить вычисления в одном направлении, но крайне сложно — в обратном. Другими словами, там необходимо максимально усложнить восстановление исходного сообщения.

По тому же принципу в эллиптических кривых необходимо найти такую *односторонюю* функцию, как, например, дискретный логарифм. Для этой роли идеально подходит *скалярное умножение*. Теперь определим задачу дискретного логарифма в контексте эллиптических кривых.

Рассмотрим эллиптическую кривую E. Известны два параметра: точка P и ее множитель Q. Необходимо найти такое k, чтобы $Q = k \cdot P$. Решить эту задачу очень трудно.

Она называется задачей дискретного логарифма Q по основанию P и считается трудноразрешимой, потому что нахождение k требует очень сложных вычислений. Чтобы лучше понять, с чем мы имеем дело, я приведу пример.

Пример. В группе, заданной эллиптической кривой, чему равен дискретный логарифм k точки $Q = (4, 5)$ по основанию $P = (16, 5)$? Ответ:

$$y^2 = x^3 + 9x + 17$$

на поле F_{23}.

Один из способов найти k — просто вычислять множители P, пока не получим Q. Первые множители P выглядят следующим образом:

$$P = (16, 5); 2P = (20, 20); 3P = (14, 14); 4P = (19, 20); 5P = (13, 10);$$
$$6P = (7, 3); 7P = (8, 7); 8P = (12, 17); 9P = (4, 5).$$

Поскольку $9P = (4, 5) = Q$, дискретный логарифм Q по основанию P составляет $k = 9$.

В реальном приложении значение k будет таким большим, что определить его подобным образом было невозможно.

Это фундаментальный принцип, лежащий в основе алгоритма Диффи — Хеллмана, реализованного на эллиптических кривых, который мы рассмотрим далее.

Реализация алгоритма Диффи — Хеллмана на эллиптических кривых

В этом разделе мы реализуем алгоритм Диффи — Хеллмана, который рассмотрели в главе 3, на эллиптических кривых. Не следует забывать, что основная задача этого алгоритма — нахождение дискретного логарифма. Здесь будет продемонстрировано, что она может быть перенесена и на эллиптические кривые.

Прежде всего рассмотрим эллиптическую кривую по модулю *p*. *Базовая точка*, или *образующий элемент*, является первым компонентом классической версии

алгоритма Диффи — Хеллмана (g) и здесь обозначается (G). Разберем несколько важных элементов.

- G — это точка на кривой, которая образует циклическую группу.

  Циклическая группа означает, что каждая точка на кривой порождается многократным сложением (сложение точек рассмотрено в предыдущем разделе). Вы уже встречались с циклическими группами, когда знакомились с алгоритмом Диффи — Хеллмана в главе 3.
- *Порядок точки G*, обозначаемый через (n):

$$\text{порядок } (G) = n.$$

  Порядок (G) = (n) — это размер группы.

  Порядок (n) является наименьшим положительным целым числом [k], при котором выполняется равенство:

$$kG = O,$$

  где O — бесконечно удаленная точка.
- Следующий важный параметр — кофактор h:

$$h = (\text{количество точек на } E) / n.$$

  Значение (h) определяется количеством точек на кривой E (mod p), разделенным на порядок кривой (n).

  Значение $h = 1$ — оптимальный вариант, тогда как $h > 4$ делает кривую уязвимой к атакам. Более подробную информацию о слабых ключах можно найти в статье https://eprint.iacr.org/2005/030.pdf.

Теперь рассмотрим пошаговую реализацию алгоритма Диффи — Хеллмана на эллиптических кривых.

Шаг 1. Инициализация параметров. Для инициализации алгоритма Диффи — Хеллмана на эллиптической кривой E (mod p) используются открытые параметры {p, a, b, G, n, h}:

- p — это (mod p), который вы уже видели в классической версии алгоритма Диффи — Хеллмана;
- a и b — параметры эллиптической кривой;
- G — образующий элемент;
- n — порядок G;
- h — кофактор.

Шаг 2. Формирование общего ключа [K] на эллиптической кривой E (mod p). После инициализации параметров Алиса и Боб должны выбрать тип кривой $E \pmod p$, с которой они будут работать:

$$y^2 = x^3 + ax + b \pmod p.$$

Боб выбирает закрытый ключ $[\beta]$ так, чтобы $1 \leqslant [\beta] \leqslant n - 1$.

Алиса выбирает закрытый ключ $[\alpha]$ так, чтобы $1 \leqslant [\alpha] \leqslant n - 1$.

После выбора случайных ключей Алиса и Боб вычисляют свои открытые ключи.

Боб вычисляет:

$$B = [\beta] \cdot G.$$

Алиса вычисляет:

$$A = [\alpha] \cdot G.$$

Теперь (аналогично классической версии алгоритма Диффи — Хеллмана) создается общий ключ путем обмена открытыми параметрами.

1. Боб отправляет B (x_B и y_B) Алисе.
2. Алиса получает B.
3. Алиса отправляет A (x_A и y_A) Бобу.
4. Боб получает A.
5. Боб вычисляет:
$$K = [\beta] \cdot A.$$
6. Алиса вычисляет:
$$K = [\alpha] \cdot B.$$
7. В итоге и Алиса, и Боб получают одну и ту же точку $[K]$ на эллиптической кривой E, которая становится их общим секретным ключом.

Для того чтобы вычислить $[K]$, Ева (злоумышленник) должна знать закрытые ключи Алисы или Боба — $[\alpha]$ или $[\beta]$ соответственно, задаваемые функциями $B = [\beta] \cdot G$ или $A = [\alpha] \cdot G$. Для восстановления значения $[K]$ ей необходимо найти дискретный логарифм на кривой $E \pmod p$, что, как вы видели, является трудноразрешимой задачей.

Числовой пример. Допустим, у нас есть эллиптическая кривая:

$$E: y^2 = x^3 + 2x + 2 \pmod{17}.$$

График этой кривой показан на рис. 7.5.

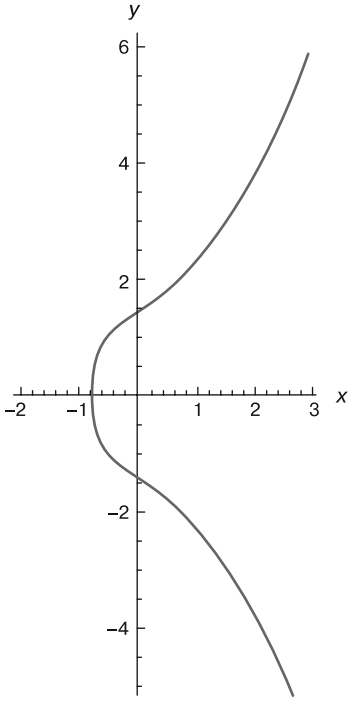


Рис. 7.5. Кривая, построенная для примера реализации алгоритма Диффи — Хеллмана

Образующим элементом возьмем G (5, 1).

Прежде всего необходимо определить порядок кривой, то есть значение n. Для этого вычисляем все точки, пока не найдем минимальное целое число n, при котором группа точек становится циклической.

Начнем с вычисления $2G$. Для этого используем формулу удвоения точки:

$$t = \left(3x_P^2\right) \Big/ 2y_P;$$

$$t = (3 \cdot 5^2 + 2) / (2 \cdot 1) = 77 \cdot 2^{-1} = \text{Reduce}[2 \cdot x = = 77, x, \text{Modulus} \rightarrow 17] =$$

$$= 13 \pmod{17} = 13.$$

Теперь параметр (t) можно использовать для вычисления координат x и y точки $2G$:

$$x_{2G} = s^2 - 2x_G = 13^2 - 2 \cdot 5 = \text{mod}\,[13^2 - 2 \cdot 5, 17] = 6 \pmod{17};$$

$$y_{2G} = s \cdot (x_G - x_{2G}) - y_G = 13 \cdot (5 - 6) - 1 = -13 - 1 = -14 \pmod{17} = 3 \pmod{17}.$$

Таким образом, $2G$ имеет следующие координаты:

$$2G = (6, 3).$$

Далее последовательно вычисляем $3G$, $4G$... nG до тех пор, пока не достигнем бесконечно удаленной точки OG. Это требует большого количества ручной работы, но служит отличным упражнением для закрепления материала.

Я помогу вам выполнить все операции скалярного умножения до точки OG, предоставив координаты некоторых точек:

- $G = (5, 1)$;
- $2G = (6, 3)$;
- $3G = (10, 6)$;
- $9G = (7, 6)$;
- $10G = (7, 11)$.

В итоге мы можем найти бесконечно удаленную точку OG:

$$19G = \boldsymbol{O}G.$$

Следовательно, порядок G:

$$n = 19.$$

Параметры кривой E:

$$G = (5, 1); n = 19.$$

Боб выбирает значение $\beta = 9$.

Алиса выбирает значение $\alpha = 3$.

Боб вычисляет свой открытый ключ (B):

$$B = 9G = (7, 6).$$

Алиса вычисляет свой открытый ключ (A):

$$A = 3G = (10, 6).$$

Алиса отправляет (A) Бобу. Боб считает общий ключ с помощью своей приватной переменной:

$$[\beta] \cdot A = 9A = 9 \cdot (3G) = 8G = (13, 7).$$

Боб отправляет (B) Алисе. Алиса считает общий ключ с помощью своей приватной переменной:

$$[\alpha] \cdot B = 3B = 3 \cdot (9G) = 8G = (13, 7).$$

Таким образом, общим ключом является $[K] = (13, 7)$.

Параметры, использованные в этом примере, слишком малы для реального применения. В реальной криптографии применяются ключи значительно длиннее представленных здесь. При этом алгоритм Диффи — Хеллмана на эллиптических кривых вычислительно проще классической версии и позволяет использовать относительно короткие ключи без ущерба для безопасности.

Необходимо выбирать эллиптические кривые, разработанные профессиональными криптографами и математиками.

Алгоритм Диффи — Хеллмана на эллиптической кривой E (mod p) не предотвращает атаки посредника так же эффективно, как его классическая версия (см. главу 3).

Теперь, когда вы лучше понимаете эллиптические кривые и связанные с ними операции, можем рассмотреть алгоритм цифровой подписи ECDSA, который используется в системе «Биткойн».

Эллиптическая кривая secp256k1 — цифровая подпись для «Биткойна»

ECDSA (алгоритм цифровой подписи на эллиптических кривых) является схемой цифровой подписи, используемой в архитектуре «Биткойна». В ней применяется эллиптическая кривая secp256k1, стандартизированная группой Standards for Efficient Cryptography Group (SECG).

ECDSA предполагает использование параметров ($a = 0$) и ($b = 7$) в уравнении[1]:

$$E: y^2 = x^3 + 7.$$

Более формальное описание можно найти в документе SECG по адресу https://www.secg.org/sec2-v2.pdf, где представлены рекомендуемые параметры для 256-битной кривой Коблица, а также других криптографических кривых той же битовой длины.

На рис. 7.6 показана кривая secp256k1 в вещественной плоскости.

Как мы знаем, эллиптическая кривая представлена как в вещественной, так и в мнимой плоскости. В трехмерном пространстве, когда точки определены в конечном поле, эллиптическая кривая может иметь вид тора (рис. 7.7).

---

[1] Параметры ($a = 0$) и ($b = 7$) нужно подставить в формулу $y^2 = x^3 + ax + b$, которая использовалась ранее. При подстановке получается то же, что и у автора.

Кривая	$y^2 = x^3 + 7$

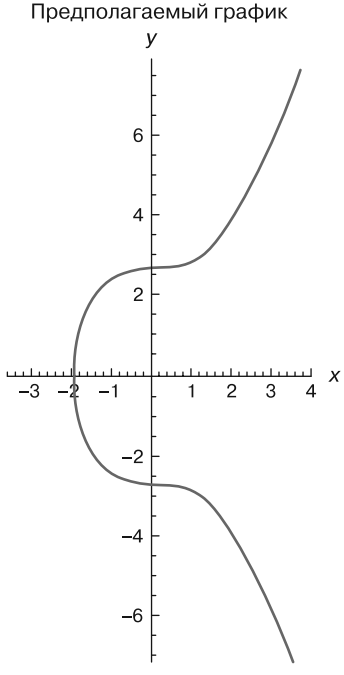


Рис. 7.6. Эллиптическая кривая secp256k1

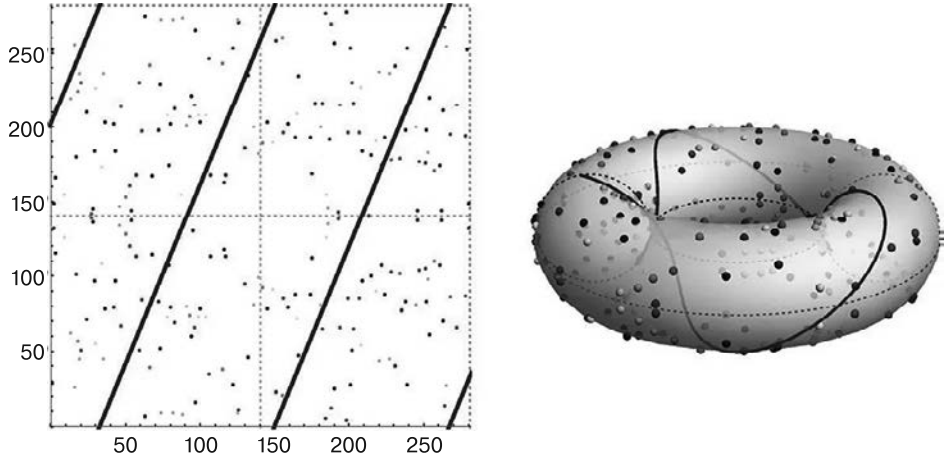


Рис. 7.7. 3D-представление эллиптической кривой в конечном поле

Кривая secp256k1 определяется в Z-поле следующим образом:

$$Z \bmod 2^{256} - 2^{32} - 977.$$

Кроме того, она может быть записана в виде целого числа 115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 908.

Здесь координаты точек представлены 256-битными целыми числами в большом модуле p.

Сначала кривая secp256k1 использовалась редко. Однако после того, как она стала применяться в «Биткойне», ее популярность резко возросла. В отличие от большинства других эллиптических кривых, обычно имеющих *случайную* структуру, secp256k1 была разработана с акцентом на эффективности. В своей правильно оптимизированной реализации она работает на 30 % быстрее аналогов. Более того, константы (a и b) были выбраны так, чтобы уменьшить вероятность появления скрытых уязвимостей, что отличает ее от кривых, предложенных NIST.

Кроме того, в secp256k1 образующий элемент (G), порядок (n) и простое число (модуль p) не выбираются случайным образом, а определяются через другие параметры. Это делает кривую одной из лучших для использования в сфере цифровых подписей.

Именно по этой причине разработчики «Биткойн» предпочли secp256k1. Далее мы увидим, какое большое количество чисел, выбранных в качестве модуля (p), порядка (n) и образующего элемента (G), обеспечивает безопасность этой кривой.

Давайте разберемся, как реализуется ECDSA на кривой secp256k1.

Шаг 1. Генерирование ключей

Закрытый ключ [d] — это число, случайно выбранное в диапазоне $1 \leqslant d \leqslant n - 1$, где ($n$) является порядком G. Оно должно иметь длину 256 бит, чтобы вычислить хеш-значение случайного числа с помощью SHA-256, которое дает на выходе 256-битное число. Таким образом можно подтвердить, что длина числа действительно равна 256 битам. Если на выходе получается число меньше ($n - 1$), то ключ принимается, если нет, то будет предпринята новая попытка проверки.

Открытый ключ (K_{pub}) определяется равенством:

$$K_{pub} = K_{priv} \cdot G.$$

Таким образом, количество возможных закрытых ключей равно порядку G, или n.

Следующее уравнение вычисляет открытый ключ (Q) для кривой secp256k1 с использованием закрытого ключа [d]:

$$Q \equiv [d] \cdot G \pmod{p}.$$

Результатом уравнения будет Q (открытый ключ).

Затем мы можем перейти к вычислению цифровой подписи. Предположим, что Алиса отправляет подпись и ее параметры $[d]$, (G) и (Q) уже рассчитаны. Виктор (майнер) должен проверить подлинность подписи. Рассмотрим далее, как подписать транзакцию «Биткойна» с использованием кривой secp256k1.

Шаг 2. Создание цифровой подписи в secp256k1

Помните, что для подписания документа $[M]$ или, что более актуально для нашего случая, данных о сумме биткойнов, $[B]$ (как мы обсуждали в главе 4), всегда рекомендуется вычислять хеш$[B] = z$.

Таким образом, мы будем подписывать (z), который является хешем сообщения, а не непосредственно само сообщение $[M]$.

Еще один необходимый нам параметр — $[k]$, который представляет собой эфемерный (или сеансовый) ключ.

Процесс создания цифровой подписи (S) имеет много общего с подписанием в алгоритме с открытым и закрытым ключами, с тем отличием, что задача дискретного логарифма здесь решается с помощью скалярного умножения.

1. Алиса выбирает случайный секрет $[k]$ так, чтобы $[1 \leqslant k \leqslant n - 1]$.
2. Алиса вычисляет координаты $R\,(x, y) = k \cdot G$.
3. Алиса находит $r \equiv x \pmod{n}$ — координату x точки $R\,(x, y)$.
4. Алиса вычисляет $S \equiv (z + r \cdot d) \,/\, k \pmod{p}$ (это цифровая подпись).

Алиса отправляет пару параметров (r и S) Виктору для проверки цифровой подписи.

Шаг 3. Проверка цифровой подписи

Получив пару параметров (r и S), Виктор может проверить подлинность подписи Алисы.

Чтобы выполнить проверку (S), Виктор использует протокол, состоящий из следующих действий.

1. Проверка того, что (r) и (S) находятся в диапазоне между 1 и $(n - 1)$.
2. Вычисление $w \equiv S^{(-1)} \pmod{n}$.
3. Вычисление $U \equiv z \cdot w \pmod{n}$.
4. Вычисление $V \equiv r \cdot w \pmod{n}$.
5. Вычисление точки на secp256k1 $R\,(x, y) = UG + VQ$.
6. Проверка того, что $r \equiv x_R \pmod{n}$.

Если (r) соответствует $x_R \pmod n$, то есть координате x точки R, то проверяющий принимает подпись (S), соответствующую данным о сумме биткойнов [B], владелец которых утверждает, что владеет заявленной суммой биткойнов.

Довольно непросто объяснить без практического примера столь сложный протокол создания цифровой подписи для транзакций биткойна с использованием кривой secp256k1. Так что в следующем подразделе представлен пример, который, я надеюсь, прояснит многие нюансы.

Числовой пример цифровой подписи на кривой secp256k1

В этом разделе мы погрузимся в создание цифровой подписи на кривой secp256k1, чтобы понять механизм реализации и проверки цифровой подписи.

Предположим, что кривая имеет следующие параметры:

- $p = 67 \pmod p$;
- $G = (2, 22)$;
- порядок $n = 79$;
- закрытый ключ $[d] = 2$.

Так как мы выбрали очень простой закрытый ключ, то для получения открытого ключа (Q) достаточно вычислить удвоенную точку:

$$Q = d \cdot G.$$

Переходим к вычислению открытого ключа (Q). Для начала воспользуемся следующей формулой для вычисления удвоенной точки:

$$t = \left(3x_P^2\right) \Big/ 2y_P;$$

$t = (3 \cdot 2^2 + 0) / (2 \cdot 22) = 12 / 44 = \text{Reduce}\ [44 \cdot x = = 12, x, \text{Modulus} \rightarrow (67)] = 49.$

В $Q = d \cdot G$ заменим параметры $[d]$ и (G) соответствующими им числами:

$$Q = 2 \cdot (2, 22).$$

С помощью формулы для вычисления удвоенной точки найдем координаты Q (x и y). Начнем с x:

$$x_Q = t^2 - 2x_G;$$

$$x_Q = 49^2 - 2 \cdot 2 = 52 \pmod{67} = 52.$$

Аналогичным образом находим координату y:

$$y_Q = t(x_G - x_Q) - y_G x;$$

$$y_Q \equiv 49(2 - 52) - 22 = 7 \pmod{67} = 7.$$

Таким образом, открытый ключ (Q) имеет следующие координаты:

$$\boldsymbol{Q}(\boldsymbol{x}, \boldsymbol{y}) = (52, 7).$$

Теперь Алиса может выполнить свою цифровую подпись (S).

Сначала она вычисляет хеш $H[M] = z$.

Предположим, что $z = 17$ — это хеш данных о сумме биткойнов [B].

Алиса выбирает случайное секретное число для сеансового ключа [k]:

$$k = 3.$$

Затем вычисляет координаты $R\,(x, y) = k \cdot G$:

$$k \cdot G = 3 \cdot (G) = G + 2G.$$

Мы уже вычислили $2G = (52, 7)$, поэтому можно выполнить дополнительные вычисления $G + 2G = (2, 22) + (52, 7)$.

Вспомним формулу сложения точек на эллиптических кривых:

$$\boldsymbol{t_1} = (y_Q - y_Q) / (x_Q - x_G).$$

Вычислить x_R (координату x точки R) можно с помощью следующей операции:

$$t_1 \equiv (7 - 22)/(52 - 2) \pmod{67} \equiv 60 \pmod{67}.$$

Мы находим [x_R]:

$$x_R \equiv t_1^2 - \left(x_G + x_Q\right) \left(\mathrm{mod}\, 67\right);$$

$$x_R \equiv 60^2 - \left(2 + 52\right) = 62 \left(\mathrm{mod}\, 67\right).$$

Давайте рассмотрим координату x точки R:

$$r = 52 \pmod{79} = 62.$$

На этом этапе можно выполнить подпись, применив формулу

$$S \equiv (z + r \cdot d) / k \pmod{p} \equiv (17 + 62 \cdot 2) / 3 \pmod{67} = 47.$$

В результате мы получили очень важную точку — подпись (S):

$$S = 47.$$

Алиса отправляет пару параметров ($S = 47$, $r = 62$) Виктору.

Для проверки цифровой подписи Виктору нужны значения (S, r) = (47, 62), а также следующие открытые параметры:

- $z = 17$;
- $n = 79$ (порядок G);
- $G = (2, 22)$ — базовая точка;
- $Q = (52, 7)$ — открытый ключ.

Прежде всего Виктор должен проверить следующее:

$$1 \leqslant (47, 62) \leqslant (79 - 1).$$

Таким образом, подпись проходит первую проверку.

Теперь Виктор вычисляет (w), представляющее собой обратное значение цифровой подписи (S):

$$w \equiv S^{(-1)} \pmod{n}.$$

Это, как мы уже неоднократно видели, означает, что выполняются обратные функции по отношению к $S \pmod{n}$:

$$\text{Reduce}[(S) \cdot x = = 1, x, \text{Modulus} \rightarrow (n)] =$$
$$= \text{Reduce}[(47) \cdot x = = 1, x, \text{Modulus} \rightarrow (79)] = 37;$$
$$w = 37.$$

Затем Виктор вычисляет значение (U):

$$U \equiv z \cdot w \pmod{n} \equiv 17 \cdot 37 \pmod{79} = 76.$$

Теперь он может проверить подпись:

$$V \equiv r \cdot w \pmod{n} \equiv 62 \cdot 37 \pmod{79} = 3.$$

Игра еще не окончена: нам предстоит *преобразовать* координаты $V = 3$ на secp256k1c помощью скалярного умножения.

Для этого нужно выполнить следующее равенство:

$$R(x, y) = U \cdot G + V \cdot Q.$$

Разделим его на две части: (UG) и (VQ). Начнем с вычисления UG:

$$U \cdot G = 76G = 2 \cdot 38G = 2 \cdot (2 \cdot 19G) = 2 \cdot (2 \cdot (G + 18G)) = 2 \cdot (2 \cdot (G + 2 \cdot (9G))) =$$
$$= 2 \cdot (2 \cdot (G + 2 \cdot (G + 8G))) = 2 \cdot (2 \cdot (G + 2 \cdot (G + 2 \cdot (4G)))) =$$
$$= 2 \cdot (2 \cdot (G + 2 \cdot (G + 2 \cdot (2 \cdot (2G))))).$$

Чтобы сократить $76G$ посредством скалярного умножения, нам нужно выполнить шесть операций удвоения точки (G) и две операции сложения точек.

Это сокращение пригодится во время работы с очень большими числами. Не существует алгоритма, способного эффективно выполнить подобную операцию, поэтому придется задействовать свои мозги.

Мы уже знаем, что $2G = 2 \cdot (2, 22) = (52, 7)$, поэтому можем переформулировать выполненные ранее операции $2 \cdot (52, 7) = (21, 42)$, что позволяет провести дальнейшее сокращение:

$$UG = 2 \cdot ((52, 7) + 2 \cdot ((2, 22) + (21, 42))) = 2 \cdot ((52, 7) + 2 \cdot (13, 44)) =$$
$$= 2 \cdot (2 \cdot (38, 26)) = 2 \cdot (27, 40) = (62, 4).$$

На следующем этапе вычисляется $VQ = 3Q = Q + 2Q$.

Один из методов выполнения этого скалярного умножения предполагает, что сначала необходимо вычислить $2Q = 2 \cdot (52, 7) = (25, 17)$.

Затем найти решение $Q + 2Q = (52, 7) + (25, 17)$:

$$VQ = (52, 7) + (25, 17) = (11, 20).$$

Наконец, можно сложить $UG + VQ$ и получить уникальную точку сложения:

$$R(x, y) = U \cdot G + V \cdot Q = (62, 4) + (11, 20) = (62, 63).$$

Виктор может выполнить последнюю проверку:

$$r \equiv x_R \pmod{n}.$$

Он получает следующий результат:

$$r = 62 = x_R = 62 \pmod{79}.$$

Виктор принимает подпись!

На этом примере вы увидели, насколько сложен процесс подписания даже при использовании небольших чисел. Поэтому можно представить себе уровень сложности, если длина параметров будет составлять 256 бит и более. Эллиптическая кривая secp256k1 предназначена для защиты прав собственности «Биткойна». Именно для этого она и используется. С помощью подписи (S) пользователь может доказать, что владеет данными [B], которые соответствуют вычисленному хешу (z).

Как упоминалось в начале раздела, secp256k1 — это особая эллиптическая кривая, отличающаяся от других своей эффективностью и тщательно подобранными параметрами, которые обеспечивают ее безопасную и оптимизированную реализацию.

Если вы хотите больше узнать о реализации алгоритма цифровой подписи ECDSA, можете посетить веб-страницы NIST и Института инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers, IEEE).

NIST опубликовал список различных эллиптических кривых и рекомендуемые параметры их реализации. Документ можно найти, перейдя по ссылке https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-78-4.pdf.

IEEE — это частная организация, занимающаяся продвижением публикаций, конференций и стандартов. Она выпустила документ IEEE P1363-2000 (Standards Specification for Public-Key Cryptography) со спецификациями для реализации эллиптической криптографии.

В следующем разделе вы узнаете о взломе закрытых ключей ECDSA, осуществленном хакерской группой fail0verflow. Эта группа объявила, что восстановила

секретные ключи, которые Sony использовала для авторизации в PlayStation 3. Успех атаки был обусловлен тем, что Sony нарушила основные принципы ECDSA, применяя статический закрытый ключ вместо генерации случайного.

Атаки на ECDSA и безопасность эллиптических кривых

Представленная здесь атака на ECDSA позволяет восстановить приватный ключ [d], если случайный (эфемерный) ключ [k] не является совершенно случайным или используется повторно для подписи хеша сообщения (z).

Атака была проведена в 2010 году для извлечения ключей подписи, использовавшихся для игровой консоли PlayStation 3, и привела к компрометации данных более чем 77 миллионов учетных записей.

Чтобы лучше понять механизм этой разрушительной атаки, которая позволяет восстановить не только сообщение, но и приватный ключ [d], мы разделим ее на два этапа. В данном примере рассмотрим ситуацию, когда два сообщения, [M] и [M_1], подписываются с использованием одинаковых закрытых ключей [k] и [d].

Шаг 1. Восстановление случайного ключа [k]

Как мы знаем, подпись ($S = 47$), сгенерированная в момент времени (t_0) из хеша сообщения [M], определяется уравнением:

$$S \equiv (z + r \cdot d) / k \pmod{p}.$$

Представим его в числовом виде:

$$S \equiv (17 + 62 \cdot 2) / 3 \pmod{67} = 47.$$

Теперь предположим, что мы знаем $z_1 = 23$ — хеш второго сообщения [M_1], отправленного в момент времени (t_1), а также вторую подпись (S_1), которая сформирована уравнением:

$$S_1 \equiv (z_1 + r \cdot d) / k \pmod{p}.$$

Используя функцию сокращения Reduce в системе Mathematica, получаем значение (S_1):

$$[k \cdot x = (z_1 + r \cdot d), x, \text{Modulus} \rightarrow (n)] = 49;$$

$$S_1 = 49,$$

где $(S - S_1) / (z - z_1) \equiv k \pmod{n}$.

Специальная функция Reduce позволяет факторизовать большие модульные числа.

Теперь, подставив полученные параметры, мы можем легко найти [k], используя функцию сокращения Reduce:

$$\text{Reduce}[(47 - 49) \cdot x = = (17 - 23), x, \text{Modulus} \rightarrow (79)];$$
$$k = 3.$$

Получив значение [k], можем вычислить закрытый ключ [d].

А теперь посмотрим, что происходит на следующем этапе.

Шаг 2. Восстановление закрытого ключа [d]

Восстановив случайный ключ [k], можно легко вычислить закрытый ключ [d].

Как мы знаем, подпись (S) создается с помощью следующего уравнения:

$$S = (z + r \cdot d) / k.$$

Переносим (k) влево из знаменателя в числитель и переписываем уравнение:

$$S \cdot k = z + r \cdot d.$$

Поскольку нам уже известны (k) = 3 и открытые параметры (S, z и r), можем найти [d].

Мы можем вычислить значение [d], переписав приведенное ранее уравнение следующим образом:

$$d = (S \cdot k - z) / r.$$

Все параметры справа нам уже известны.

Подставив сюда числа, получаем:

$$(47 \cdot 3 - 17) / r = 2.$$

Таким образом, значение закрытого ключа [d] = 2 было восстановлено!

Здесь возникает вопрос: что произойдет, если кто-то найдет способ сгенерировать множество подписей (S_1, S_2 и S_n) на основе разных хешей (z_1, z_2 и z_n) или создать несколько подписей, используя один и тот же хеш (z)?

Другими словами, поскольку (z) является хешем сообщения (данными о сумме биткойнов, хранящихся в кошельке), можно задать вопрос: что произойдет, если удастся сгенерировать новую подпись S_1 с помощью (z)?

Только представьте: если возможно создать новую подпись S_1 на основе (z), тогда становится возможным генерирование множества валидных подписей для одной транзакции.

Теперь кратко проанализируем ключевое преимущество эллиптической криптографии перед другими шифрами. Эффективность эллиптических кривых в сравнении с классическими алгоритмами криптографии с открытым и закрытым ключами показана на рис. 7.8. Вы сразу же заметите, что размер ключей в ECC[1] значительно меньше, чем в алгоритме *Ривеста, Шамира и Адлемана* (RSA) и в *алгоритме цифровой подписи* (DSA). В таблицу также включены размеры ключей симметричной схемы шифрования (расширенного стандарта шифрования). Эти ключи, несомненно, короче, однако, в отличие от алгоритмов симметричного шифрования, эллиптическая кривая используется для цифровых подписей, поэтому лучше сравнивать ее с алгоритмами асимметричного шифрования.

Сравнение размеров ключей, необходимых для обеспечения одинакового уровня защиты

Симметричная схема (размер ключа, битов)	Схема на эллиптических кривых (размер порядка *n*, битов)	RSA и DSA (размер модуля, битов)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15 360

Рис. 7.8. Таблица Уильяма Сталлингса для сравнения эллиптической криптографии и классических методов шифрования

Как видите, 256-битное шифрование на основе эллиптических кривых эквивалентно 3072-битному ключу в RSA. Более того, 512-битный ключ ECC эквивалентен 15 360-битному ключу RSA. В сравнении с алгоритмом симметричного шифрования эллиптическая криптография требует вдвое большего количества битов. Но, поскольку она предназначена для выполнения цифровых подписей, сравнение с алгоритмами асимметричного шифрования (RSA, DSA) более уместно.

[1] ECC (elliptic-curve cryptography) — криптография, основанная на эллиптических кривых.

При выборе той или иной реализации особенно важным становится вопрос вычислительной мощности.

Предупреждение по поводу длины ключа: не имеет значения, насколько длинным является модуль или насколько велик ключ. Логическую уязвимость алгоритма нельзя исправить.

Взгляд на будущее эллиптической криптографии

Теперь, когда вы на практике увидели эффективность атаки на ECDSA, возникает один из самых интересных вопросов: *является ли криптография на эллиптических кривых устойчивой к традиционным и квантовым атакам?*

Большинство эллиптических кривых (при правильной реализации) неуязвимы для традиционных атак, кроме атак на дискретный логарифм (например, Pollard Rho или атака дней рождения) и атак посредника для версии алгоритма Диффи — Хеллмана на эллиптических кривых.

Тем не менее задача, на которой основаны эллиптические кривые, решаема алгоритмом Шора — представителем квантовой криптографии. Мы обсудим этот алгоритм в следующей главе.

Таким образом, если кто-то спросит: «Будут ли мои биткойны в безопасности в течение следующих 10–20 лет?» — можно ответить так: «При некоторых определенных условиях да. Однако, если начало эпохи квантовых вычислений сгенерирует достаточное количество квантовых битов для решения традиционной задачи дискретного логарифма, задача дискретного логарифма на эллиптических кривых также может быть решена за полиномиальное время». Таким образом, я согласен с аспирантом МТИ Джереми Вольвендом, который в своей работе *Elliptic Curve Cryptography: Pre and Post Quantum* («Криптография с эллиптическими кривыми: до и после квантовой эпохи») написал:

> *«К сожалению, день, когда квантовые компьютеры достигнут практически эффективного количества квантовых битов, станет концом эллиптической криптографии в том виде, в котором мы ее знаем».*

Резюме

В этой главе мы проанализировали некоторые из наиболее часто используемых эллиптических кривых. Мы рассмотрели, что представляет собой эллиптическая кривая и как она применяется в криптографических целях.

Эллиптическая криптография (ECC) включает в себя алгоритмы и протоколы, предназначенные в первую очередь для защиты секретной информации в системах шифрования с открытым и закрытым ключами, таких как обмен ключами Диффи — Хеллмана и цифровые подписи.

В частности, мы детально рассмотрели задачу дискретного логарифма, перенесенную на эллиптические кривые, и вы познакомились с базовыми операциями, используемыми в эллиптической криптографии, такими как сложение точек на кривой и скалярное умножение.

Эти операции существенно отличаются от сложения и умножения в традиционной арифметике, и именно в них заключается сила эллиптических кривых.

Исследовав на практическом примере алгоритм Диффи — Хеллмана на эллиптических кривых, мы подробно разобрали эллиптическую кривую secp256k1, которая используется в протоколе «Биткойна» для реализации цифровых подписей с помощью ECDSA.

Теперь, после изучения эллиптических кривых и их применения в системах шифрования с открытым и закрытым ключами, становится понятно, что основным преимуществом этих кривых является использование ими ключей небольшого размера.

В конце главы мы затронули вопрос устойчивости эллиптической криптографии к атакам со стороны квантовых компьютеров. Ответ на него представлен в главе 9, где мы погрузимся в одну из самых интригующих и необычных тем этой книги. Квантовые вычисления и квантовая криптография бросят новый вызов будущему криптографии.

8 Гомоморфное шифрование и система защищенного поиска

В этой главе рассмотрим технику поиска по зашифрованным данным. Я предоставлю основную информацию о поисковых системах в целом и их математической составляющей. Затем вы познакомитесь с инновационной поисковой системой, способной работать с зашифрованными данными, — с *системой зашифрованного поиска* (Crypto Search Engine, CSE).

Эта система была спроектирована и разработана компанией Cryptolab (https://www.cryptolab.us/), которая занимается кибербезопасностью и защитой данных. Я основал эту криптографическую лабораторию вместе с двумя инженерами, работающими в области науки о данных, Алессандро Пассерини и Тицианой Ланди, в 2012 году. Цель Cryptolab — повысить конфиденциальность и безопасность данных, передаваемых и обрабатываемых через Интернет, а затем хранимых и совместно используемых в облаке. Дополнительную информацию можно найти на веб-странице CSE (https://www.cryptolab.cloud/).

Мы поговорим о гомоморфном шифровании, в частности о гомоморфном поиске — одной из самых увлекательных задач, с которыми я сталкивался за свою карьеру. Я с удовольствием расскажу о том, что вдохновило меня на эту работу, и изложу историю создания этой системы. Кроме того, вы узнаете, на каких математических принципах основываются эти поисковые системы, а также познакомитесь с соответствующими алгоритмами.

Введение в CSE — гомоморфизм

CSE зародилась в 2014 году, когда я несколько месяцев бился над новым методом *факторизации*. Как нетрудно заметить, задачи факторизации и *слепого поиска* тесно связаны и обладают схожей вычислительной сложностью.

Более того, эти задачи лежат в рамках проблемы $P = NP$: одни из них решаются *легко* (P), а другие — *крайне трудно* (NP) даже при гипотетически неограниченных вычислительных ресурсах.

Большинство ученых и специалистов по анализу данных убеждены, что $P \neq NP$, а все NP-задачи принципиально неразрешимы полиномиальными алгоритмами.

С этим я не согласен. Меня куда больше увлекает поиск *решений* сложных задач, чем рассуждения о невозможности найти их. Я также уверен, что существуют альтернативные пути решения. Например, последняя теорема Ферма была доказана Эндрю Уайлсом — доказательство заняло около 100 страниц (https://staff.fnwi.uva.nl/a.l.kret/Galoistheorie/wiles.pdf).

Однако я убежден, что решение может уложиться в несколько страниц — у меня даже есть конкретные идеи. К примеру, выполнение алгоритма Шора с помощью эффективного квантового процессора (это будет показано в главе 9) позволит факторизовать большие полупростые числа за считаные секунды. То же самое с поиском в гигантских базах данных: квантовые алгоритмы вроде алгоритма Гровера мгновенно находят решения даже среди триллионов теоретически возможных вариантов.

Возвращаясь к истории CSE, скажу, что после завершения работы над *первой версией* нашей *криптографической библиотеки* в Cryptolab я занялся разработкой алгоритма слепого поиска данных. Цель проекта заключалась в объединении всех наработок, созданных мной и моей исследовательской командой в период с 2009 по 2014 год, в одну библиотеку. Проект *Cryptoon* был разработан на языке C++ и включал не только наши криптографические алгоритмы, но и алгоритмы симметричного и асимметричного шифрования, протоколы доказательств знания и аутентификации с нулевым разглашением, большинство из которых описаны в этой книге.

В то время меня особенно вдохновляли открытия в области *гомоморфного шифрования*. Его особенность заключается в возможности выполнять вычисления над зашифрованными данными и возвращать уже зашифрованный результат. Чтобы получить открытый текст, достаточно расшифровать результат с помощью соответствующего ключа. Для наглядности рассмотрим схему работы этой сложной системы.

Операции над зашифрованными данными генерируют зашифрованный результат $E[Z]$ следующим образом:

открытый текст (A), $(B) \rightarrow$ зашифрованные данные $E[A]$, $E[B] \rightarrow$ операции над зашифрованными данными = зашифрованный результат;

$$E[A] \; \forall o \; E[B] = E[Z].$$

Расшифровка результата $E[Z]$ выполняется следующим образом:

$E[Z] \rightarrow$ расшифровка зашифрованного результата $[Z]$;

$$D(Z) = (A) \; \forall o \; (B) = (Z) \text{ (открытый текст)}.$$

В своем изобретении я воспользовался символом $\forall o$ для обозначения любой математической и логической операции (или набора операций), выполняемой между (A) и (B), где $\forall$ означает «для всех», o — «операции».

Не волнуйтесь: в этой главе мы подробно разберем данные концепции. Теперь обратимся к понятию гомоморфизма.

Гомоморфизмы — это операции, выполняемые над *изоморфными элементами*. Мы можем найти изоморфизмы как в природе, так и в математике. В природе, например, нижние крылья некоторых бабочек имеют концентрический узор, повторяющий форму крыла. Скелет человека соотносится с его телом или его тенью, что является формой изоморфизма: элементы, принадлежащие человеку, животному или чему-то другому, связаны с целым или порождены им.

Даже если с помощью тени нельзя разглядеть лицо человека, мы, безусловно, можем распознать некоторые его черты, например фигуру или волосы, что позволяет идентифицировать этого человека. Мы говорим не о внешнем сходстве, а о структурной связи между частями одного целого или объектами, имеющими общую природу.

На рис. 8.1 приведены две фотографии. Левая представляет изоморфизм, потому что тень на песке, созданная солнечным светом, структурно отражает тело, а вторая — нет, потому что облака над домом лишь случайно повторяют его форму.

Рис. 8.1. Примеры: *слева* — изоморфизма; *справа* — отсутствия изоморфизма

Строго говоря, *гомоморфное шифрование* — это метод шифрования, который позволяет производить вычисления на шифротекстах, создавая зашифрованный результат, который при расшифровке полностью совпадает с результатом тех же операций, как если бы они были выполнены над открытым текстом.

Рассмотрим схему гомоморфного шифрования (рис. 8.2).

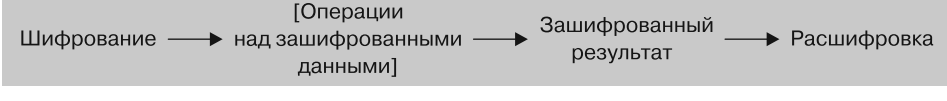


Рис. 8.2. Поток операций в гомоморфном шифровании

Этот процесс выполняется с помощью таких математических операций, как умножение или сложение. Если алгоритм поддерживает все математические и логические операции, его называют *полностью гомоморфным*, в остальных случаях — *частично гомоморфным*. К последним относятся RSA, схема Эль-Гамаля или MBXI. В RSA, как вы увидите далее, гомоморфизм проявляется при умножении.

Частичный гомоморфизм в RSA

Чтобы продемонстрировать соответствие между операциями над шифротекстом и открытым текстом (в данном случае умножением), приведу алгоритм RSA.

Формула шифрования в RSA выглядит следующим образом:

$$[M]^e \equiv c \pmod{N},$$

где $[M]$ — сообщение; (e) — открытый параметр шифрования; (N) — открытый ключ, состоящий из $[p] \cdot [q]$; (c) — криптограмма.

Предположим, что у нас есть два сообщения, $[M_1]$ и $[M_2]$, и мы шифруем их с помощью одних и тех же открытых параметров (e, N), чтобы в результате получить две разные криптограммы (c_1, c_2):

$$c_1 \equiv M_1^e \pmod{N};$$
$$c_2 \equiv M_2^e \pmod{N}.$$

Если перемножить шифры $(c_1 \cdot c_2)$, то получим третью криптограмму (c_3):

$$c_3 = c_1 \cdot c_2;$$
$$c_3 \equiv M_1^e \pmod{N} \cdot M_2^e \pmod{N} \equiv M_1^e \cdot M_2^e \pmod{N}.$$

Я просто заменил операции, выполняемые над зашифрованными сообщениями $[M_1]$ и $[M_2]$, криптограммами (c_1) и (c_2).

Все эти операции можно объединить в криптограмму (c_3):

$$c_3 = c_1 \cdot c_2 \pmod{N},$$

или:

$$c_3 \equiv ([M_1] \cdot [M_2])^e \pmod{N}.$$

Таким образом, мы получили (c_3), который представляет собой результат шифрования с использованием умножения двух криптограмм $(c_1 \cdot c_2)$ на первом уровне и одновременно результат операций над сообщениями $[M_1 \cdot M_2]$ на втором уровне. Теперь посмотрим, что подразумевается под *вторым уровнем*. Расшифровав (c_3) с помощью закрытого ключа $[d]$, мы получим $[M_1 \cdot M_2]$:

$$c_3^d \pmod{N} \equiv M_1 \cdot M_2.$$

Возможным скептикам я предлагаю рассмотреть числовой пример, подтверждающий, что:

$$c_1 \cdot c_2 \equiv ([M_1] \cdot [M_2])^e \pmod{N},$$

или:

$$c_3^d \pmod{N} \equiv M_1 \cdot M_2.$$

Таким образом, расшифровав криптограмму (c_3) своим закрытым ключом [d], Алиса получит результат (в линейной математике) умножения сообщений [M_1] · [M_2], который Боб заранее «зашил» в уникальную криптограмму (c_3) с помощью RSA.

Это называется *частичным гомоморфизмом* умножения в RSA.

Числовой пример. Для проверки частичного гомоморфизма RSA возьмем несколько небольших чисел:

- $M_1 = 11$;
- $M_2 = 8$;
- $e = 7$;
- $p = 13$;
- $q = 17$;
- $N = 221$ [p] · [q].

Вычисляем криптограммы c_1 и c_2:

$$c_1 \equiv 11^7 \pmod{221} = 54;$$
$$c_2 \equiv 8^7 \pmod{221} = 83.$$

Теперь необходимо вычислить (c_3) *первого уровня*:

$$c_3 \equiv c_1 \cdot c_2 \pmod{N}.$$

Затем получаем (c_3) *второго уровня*:

$$c_3 \equiv ([M_1] \cdot [M_2])^e \pmod{N}.$$

Подставив числа, получаем *на первом уровне*:

$$c_3 \equiv 54 \cdot 83 = 62 \pmod{221},$$

на втором уровне:

$$c_3 \equiv ([11] \cdot [8])^7 \pmod{221} = 62.$$

Как видите, мы имеем определенное соответствие между операциями над криптограммами c_1 и c_2 (в данном случае умножение) и операциями умножения над сообщениями [M_1, M_2].

Пока все нормально: мы применили простые математические преобразования к уравнениям первого и второго уровней.

Однако ключевым моментом является расшифровка (c_3). Алиса может расшифровать две переданные Бобом криптограммы (c_1, c_2) с помощью своего уникального закрытого ключа [d]. Найдем закрытый ключ [d], используя известную функцию Reduce в Wolfram Mathematica:

$$\text{Reduce}\,[e \cdot d = 1, x, \text{Modulus} \rightarrow [13 - 1]\,[17 : 1]];$$
$$d = 55.$$

Теперь самое интересное: расшифровав (c_3), являющуюся произведением ($c_1 \cdot c_2$), Алиса может узнать и результат умножения сообщений $[M_1] \cdot [M_2]$.

В RSA функция дешифрования выглядит следующим образом:

$$c^d \equiv M \,(\text{mod}\, N).$$

В нашем примере она принимает вид:

$$c_3^d \equiv M_1 \cdot M_2 \,(\text{mod}\, N).$$

Подставив сюда $c_3 = 62$ и $[d] = 55$:

$$62^{55} \equiv 88 \,(\text{mod}\, 221),$$

получаем следующий результат:

$$M_1 \cdot M_2 = 11 \cdot 8 = 88.$$

Это именно то, что нам нужно!

Таким образом, RSA действительно обладает некоторыми свойствами гомоморфизма, если использует для шифрования умножение. Однако для гомоморфного поиска, в отличие от RSA, требуются иные подходы. Далее углубимся в изучение гомоморфного шифрования и его роли в защите данных.

Изучение гомоморфного шифрования и способы его применения

Анализ частично гомоморфной схемы RSA из предыдущего раздела позволяет выявить интересную взаимосвязь между операциями над криптограммами (открытыми данными) и сообщениями (зашифрованными данными).

В ней и заключается суть гомоморфизма.

Проблема в том, что RSA, как и большинство известных алгоритмов, поддерживает *гомоморфизм* лишь *частично* и может представлять только умножение или сложение. Настоящая трудность заключается в том, чтобы найти эффективный

алгоритм, представляющий все математические и логические операции одновременно.

Представим еще один случай гомоморфизма на примере сложения.

Возьмем два секретных значения, [A] и [B], которые в сумме дают [C]:

$$A + B \equiv C \pmod{Z}.$$

Теперь возьмем два зашифрованных значения, соответствующих A и B. Например, зашифрованные значения $a = 9$ и $b = 2$, сумма которых $c = 11$.

Хотя это упрощенная модель (технически не шифрование), она иллюстрирует частичный гомоморфизм сложения:

$$A = 87 \rightarrow a \equiv 9 \pmod{13};$$
$$B = 93 \rightarrow b \equiv 2 \pmod{13};$$
$$C = A + B = 180 \rightarrow c \equiv a + b \equiv 11 \pmod{13}.$$

Чтобы лучше понимать, что такое гомоморфизм, представим эти операции в виде ряда (рис. 8.3, 8.4).

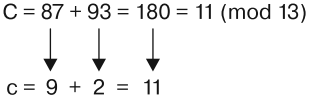


Рис. 8.3. Соответствие гомоморфной суммы открытых и зашифрованных значений

Рис. 8.4. Слепой поиск

Как видите, это простое соответствие позволяет работать *вслепую* с зашифрованными значениями $a + b = 9 + 2$ и получать результат $c = 11$ без раскрытия *конфиденциальных* значений $A = 87$, $B = 93$ и $C = 180$. Вместо них отображается их изоморфное значение $c = 11$, которое не является реальным значением C, но соответствует ему. По условию функция, преобразующая значение, односторонняя, поэтому будет трудно вернуться от (a) к [A], от (b) к [B], а от (c) к его изоморфному значению [C].

Я намеренно выделил термин «конфиденциальность» (знаю, может показаться странным говорить о конфиденциальности числа). Смысл в том, что результат, который мы хотим получить, работая с такими формами изоморфизма, сохраняет конфиденциальность данных. Результаты *слепой* работы с данными могут находиться в открытом доступе без риска раскрытия элементов исходных уравнений.

Именно это вдохновило меня на создание CSE — поисковой системы, способной работать с зашифрованными данными *вслепую*, не зная, что именно ищет, и не зависящей от алгоритмов шифрования и дешифрования получаемых данных.

Прежде всего необходимо разобраться с математической составляющей как традиционных поисковых систем, так и самой CSE. Это поможет лучше понять особенности их реализации.

Пришло время изучить математические и логические основы традиционных поисковых систем и выяснить, насколько сложно осуществлять эффективный поиск по зашифрованным данным.

Математические и логические основы традиционных поисковых систем

Основы *теории сетей* были заложены в 1735 году благодаря знаменитой задаче о кенигсбергских мостах. Первым, кто ее решил, стал математик Леонард Эйлер. Простая на первый взгляд задача состояла в том, чтобы пересечь реку Прегель в прусском Кенигсберге (современный Калининград) по нескольким мостам.

Как видно на рис. 8.5, семь мостов соединяют два берега реки (А и Б) через два острова (В и Г). Суть вопроса: *существует ли маршрут, по которому можно пройти по всем мостам только один раз*?

Вы можете попытаться найти решение самостоятельно, посмотрев на рис. 8.5.

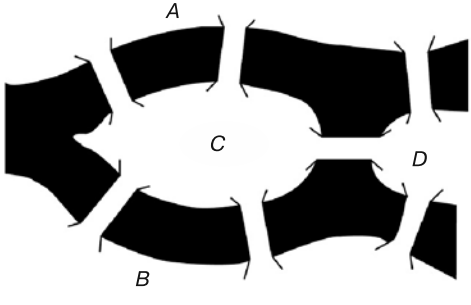


Рис. 8.5. Семь мостов Кенигсберга

Здесь важно понимать, что перед нами сеть из мостов, и, чтобы решить задачу, ее необходимо визуализировать. На рис. 8.6 изображены всего четыре узла — точки, где может находиться путник в конкретный момент времени.

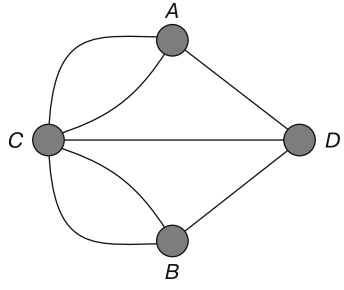


Рис. 8.6. Сеть мостов Кенигсберга

Эйлер показал, что это невозможно, так как сеть мостов состоит из четырех узлов (берега и острова) и семи ребер (мосты). Если число узлов четное, а число ребер нечетное, то задача не имеет решения.

Проблема эффективного прохождения сетей — один из важнейших элементов, который необходимо учитывать при разработке и реализации эффективной *поисковой системы*, *социальной сети* или *анализе больших данных*. Мы должны найти способ получить решение (например, искомое слово), а затем пройти сеть и вернуться обратно с наименьшими усилиями.

Похожая задача возникает при поиске оптимального маршрута среди множества вариантов — например, дорог на карте. Задумывались ли вы когда-нибудь, почему Google лидирует не только в поиске информации, но и в построении лучших маршрутов до места назначения длиной в сотни миль за миллисекунды?

Эта задача связана с *цепями*, в частности *эйлеровыми цепями* и другими подобными математическими задачами. Чтобы глубже изучить данную концепцию, рассмотрим пример работы сети под названием *Papillon* (рис. 8.7).

Сеть Papillon — это эйлерова цепь $A \to B \to C \to D \to E \to C \to A$.

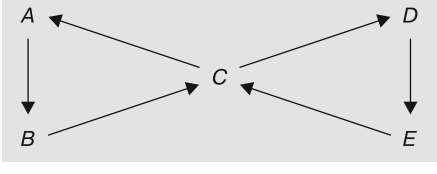


Рис. 8.7. Эйлерова сеть Papillon

Как видите, мы дважды проходим через точку *B*, при этом неважно, с какой точки начнется наш путь.

Однако, немного изменив маршрут, как показано на рис. 8.8, можно пройти через все пять узлов (*A*, *B*, *C*, *D*, *E*) без повторного прохода через *C*.

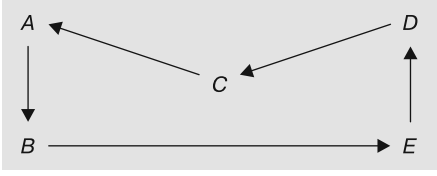
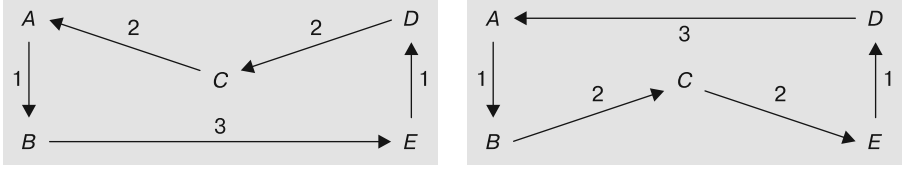


Рис. 8.8. Эйлеров цикл

В этом случае цепь выглядит так: $A \rightarrow B \rightarrow E \rightarrow D \rightarrow C \rightarrow A$.

Это подводит нас к следующему вопросу: *как найти наилучший маршрут по цепи, состоящей из узловых точек*?

Для создания эффективной поисковой системы, способной найти оптимальный маршрут прохождения узлов (как на рис. 8.7 и 8.8), необходимо задать некоторые условия. В реальности такими условиями могут быть пробки, объезд платных дорог, но чаще всего ключевым критерием становится кратчайший путь. В нашем случае таким условием является нахождение кратчайшего пути, пересекающего все точки по часовой стрелке только один раз. Другими словами, поисковой системе необходимо рассчитать минимальное расстояние для маршрута, начинающегося и заканчивающегося в точке *A*, который пересекал бы все пять точек по часовой стрелке только единожды. Одно (но не единственное) из возможных решений показано на рис. 8.9, *а*.



а *б*

Рис. 8.9. Возможные решения для цикла: *а* — M_1; *б* — M_2

На рис. 8.9, *б*, изображено еще одно решение — $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow A$. Схема на рис. 8.7 (сеть Papillon) не удовлетворяет одному из заданных нами условий (однократное посещение узлов), поэтому она не подходит.

Для оптимизации поиска мы должны присвоить значение каждому сегменту, связанному линиями, представляющими собой маршрут, по которому будет

двигаться поисковая система. Предположим, сегменты имеют следующие значения:

- $A \rightarrow B = 1$;
- $Б \rightarrow E = 3$;
- $A \rightarrow C = B \rightarrow C = C \rightarrow E = D \rightarrow C = 2$.

Маршруты, показанные на рис. 8.9, *а* и *б*, одинаково эффективны.

Маршрут M_1 имеет значение 1 + 3 + 1 + 2 + 2 = 9. Он не менее эффективен, чем маршрут M_2, который имеет то же значение: 1 + 2 + 2 + 1 + 3 = 9.

Итак, два решения, M_1 ($A \rightarrow B \rightarrow E \rightarrow D \rightarrow C \rightarrow A$) и M_2 ($A \rightarrow B \rightarrow C \rightarrow E \rightarrow D \rightarrow A$), имеют одинаковое суммарное значение $M = 9$, что является оптимальным вариантом для нашей поисковой системы.

Теперь предположим, что мы хотим вычислить суммарное значение цепи Papillon (см. рис. 8.7). Это значение — $M_p = 10$, что делает Papillon не таким эффективным маршрутом в целом, но лучшим вариантом для нахождения точки C и возвращения в точку A за меньшее время (рис. 8.10). Цепь Papillon позволяет вернуться в точку A непосредственно из B: M_3: $A \rightarrow B \rightarrow C \rightarrow A$.

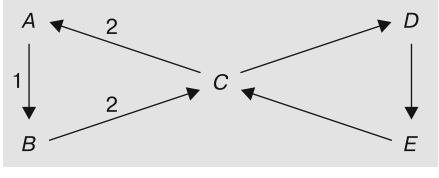


Рис. 8.10. Маршрут M_3

Эти примеры дают понимание логики работы поисковой системы. Одно из главных требований к ней — эффективность. Нельзя игнорировать тот факт, что важным элементом в разработке поисковой системы (зашифрованной или нет) является производительность. Например, путь, выраженный в километрах или других единицах измерения, всегда имеет временной эквивалент. Это как попросить Google найти оптимальный способ добраться до пункта назначения и выбрать в качестве переменной время в пути, а не расстояние.

Время — ключевой фактор при обработке запроса в поисковой системе. Принято считать, что 2 секунды — это максимально возможное время для ответа на запрос (в незашифрованном виде). Пользователь не готов ждать ответа дольше 2 секунд, когда ищет что-то на своем мобильном телефоне. Именно поэтому мы должны спроектировать цепь с упором на максимальную эффективность.

Одно из базовых требований для разработки CSE таково: время обработки запроса не должно превышать 2 секунд. В самом начале это казалось

невыполнимой задачей, учитывая уже существующие поисковые системы вроде TOR (https://www.torproject.org/), которые тоже работают с зашифрованными данными, но решают немного иные задачи.

Работа с зашифрованными данными подразумевает увеличение задержки. Это обусловлено тем, что математические операции, как мы уже видели, требуют большей вычислительной мощности, чем операции над открытым текстом. Многоуровневое шифрование (как в TOR) не всегда повышает безопасность системы, но определенно нагружает ее.

Важно помнить, что в криптографии одно слабое звено ставит под угрозу всю цепь. В следующем разделе мы рассмотрим еще один важный элемент теории графов, который имеет решающее значение для реализации поисковой системы, — древовидные графы.

Введение в деревья — теория графов

Древовидный граф — это дискретная математическая структура, геометрически представленная в форме дерева. Такие графы применяются для принятия решений и поиска оптимальных маршрутов в математике, области искусственного интеллекта и других сферах. Как будет показано далее, код Хаффмана задействует древовидные графы для кодирования текста.

В древовидном графе используются вершины (v), также называемые узлами или точками, которые связаны друг с другом. Каждая пара связанных вершин называется ребром, или связью, или звеном. Две вершины связаны единственным путем.

На рис. 8.11 вы видите дерево, где *1* — корень дерева, *2–7* — узлы, а *8–15* — *листья* (конечные точки). Отрезки, соединяющие все точки, — это ребра.

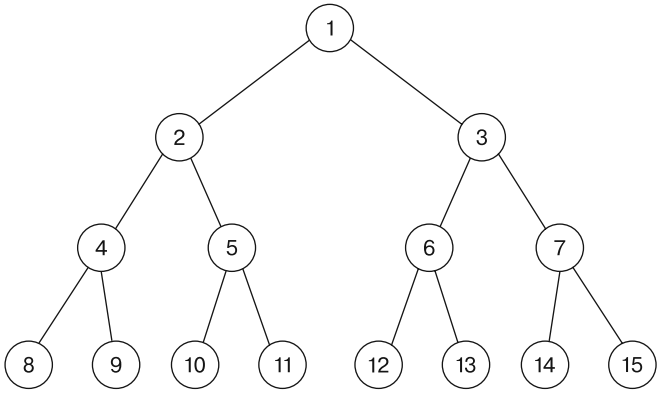


Рис. 8.11. Древовидный граф

В этой книге мы рассмотрели множество криптографических алгоритмов, основная цель которых — обеспечение безопасности данных. А сейчас разберем алгоритмы, также известные как *коды*. Они не только защищают данные, но и оптимизируют хранение информации в компактной и удобной форме. В основе таких алгоритмов лежит эффективность поиска и извлечения информации. Как мы видели на примере кенигсбергских мостов и эйлеровых цепей, древовидные графы могут быть использованы для вычисления наилучшего способа получения информации.

Эта задача носит название *префиксных кодов* и используется в *сжатии данных без потерь* — еще одном ключевом элементе криптографии. В следующем разделе мы рассмотрим кодирование по Хаффману — разновидность сжатия данных, при котором можно извлечь полученную информацию без потери данных.

Важность префиксных кодов обусловлена тем, что они позволяют закодировать слово уникальным образом (табл. 8.1).

Таблица 8.1. Пример префиксных кодов букв

Буква	a	b	c	d	e	f
Код	0	101	100	111	1101	1100

Пример кодирования строки *abc* будет выглядеть как 0·101·100. Таким образом гарантируется, что при поиске строки, начинающейся с *abc*, не будет выдано других кодов, кроме 0101100. Этот метод аналогичен системе телефонных индексов, в которой каждая страна, регион или город имеет уникальный индекс. Например, индекс 001 соответствует Северной Америке, а добавление к нему кода 415 (001-415) сужает область до Сан-Франциско в Калифорнии. Как показано на рис. 8.12, мир разделен на зоны со следующими префиксными кодами: 001 — Америка, 002 — Африка, 003 — Европа и т. д. В результате упрощается поиск номера, расположенного в коде страны — региона — города.

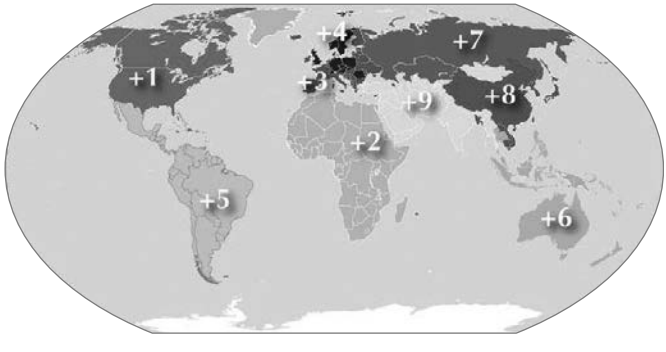


Рис. 8.12. Префиксные коды номеров по всему миру

Поисковые системы работают схожим образом. Как и в случае с префиксами, алгоритмы индексации должны быть сверхэффективными, чтобы находить нужные данные среди миллиардов записей за миллисекунды. Хотя эта книга посвящена криптографическим алгоритмам, я хочу объяснить принцип работы древовидных графов, так как они являются ключом к реализации эффективной поисковой системы для зашифрованных данных. Итак, приступим к реализации дерева.

Код Хаффмана

Предположим, мы хотим решить следующую задачу: представить текст в двоичном коде наиболее компактным способом, минимизируя количество битовых строк.

В главе 1, в подразделе «Двоичные числа, код ASCII и условные обозначения», мы уже рассматривали кодирование символов (букв, цифр, знаков) с помощью ASCII. Однако здесь задача заключается в другом: нам нужно обеспечить высокую скорость и эффективность кодирования текста с использованием как можно меньшего количества битов.

Мы можем воспользоваться элегантным методом, придуманным Дэвидом Хаффманом, который работает за счет реализации специального древовидного графа.

В английском алфавите 26 букв, но не все они встречаются в тексте с одинаковой частотой. Например, мы хотим реализовать древовидный граф, способный кодировать шесть букв: *a*, *o*, *q*, *u*, *y* и *z*. В нашем тексте они имеют следующую частотность:

- *a* = 20;
- *o* = 28;
- *q* = 4;
- *u* = 17;
- *y* = 12;
- *z* = 7.

Задача состоит в том, чтобы определить минимально возможное количество битов, (0) и (1), для кодирования текста, состоящего из этих букв.

Взгляните на рис. 8.13, где показана стратегия реализации этого древовидного графа.

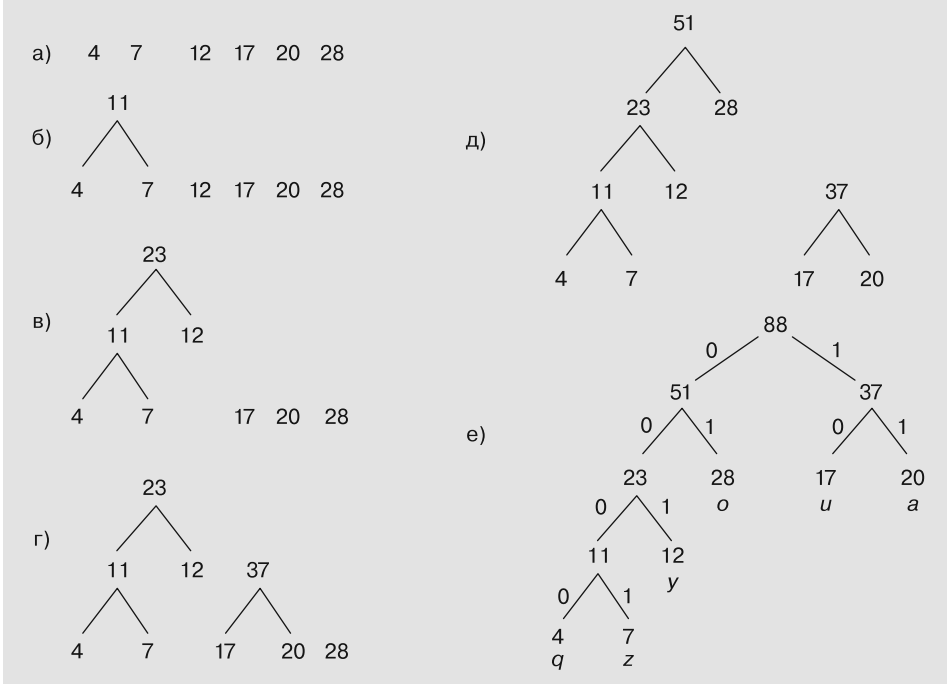


Рис. 8.13. Реализация древовидного графа Хаффмана

Проанализируем последовательность шагов, необходимых для реализации графа.

1. Первым делом (этап *а*) формируем последовательность узлов (в данном случае шесть), располагая их в порядке возрастания частотности букв.
2. На шагах *б* — *д* можно увидеть реализацию основного дерева. Узлы представляют собой частичные деревья одного большого дерева (этап *е*). Чтобы получить два узла, связанных с числом 88, на этапе *г* мы создаем новое дерево, объединив 17 и 20 в 37.
3. Наконец, находим вершину дерева, представленную числом 88 (корень), которое является суммой вершин 51 (23 + 28) и 37 (17 + 20). Теперь, когда мы закончили построение дерева, можем присвоить значение 0 всем левым ребрам и 1 — правым (это работает и в обратном режиме). Кроме того, присваиваем каждой букве число, соответствующее ее частотности: $q = 4$, $z = 7$, $y = 12$, $o = 28$, $u = 17$, $a = 20$.
4. После построения дерева считываем *код строки* для каждой буквы, начиная с корня и добавляя 0 каждый раз, когда переходим к левому дочернему

узлу, и 1, когда переходим к правому дочернему узлу. Так, например, чтобы прочитать букву *z*, мы начинаем с корня-вершины 88 и считаем, сколько нулей встречается на левых ребрах и сколько единиц — на правых, пока не достигнем буквы *z*. Таким образом, для *z* получится 0001, а для других букв — *q* = 0000, *z* = 0001, *y* = 001, *o* = 01, *u* = 10, *a* = 11.

Существуют и другие схемы кодирования дерева. Например, можно присвоить значение 0 правой стороне, а 1 — левой.

Структура дерева гарантирует, что ни один код не повторится. Например, *a* = 11 будет уникальным для любой другой закодированной буквы и ни одна другая буква не будет начинаться с 11. Это упрощает для компьютера чтение данных и ускоряет их декодирование.

Мы также можем убедиться, что для кодирования шести букв с учетом их частотности из примера использовалось минимальное количество битов:

$$(4 \cdot 4) + (4 \cdot 7) + (3 \cdot 12) + (2 \cdot 17) + (2 \cdot 20) + (2 \cdot 28) = 210.$$

Для кодирования буквы *q* нужно 4 бита (*q* = 0000), а ее частотность *q* = 4. Столько же битов нужно для *z* = 0001, умноженной на *z* = 7 (ее частотность), и т. д. Таким образом можно гарантировать, что текст максимально сжат. Для углубленного изучения данной темы читайте статью о сжатии по Хаффману: https://www2.cs.sfu.ca/CourseCentral/365/li/squeeze/Huffman.html.

Префиксные коды значительно упрощают сжатие и поиск данных, но это не единственные элементы, которые необходимо учитывать при создании эффективной зашифрованной поисковой системы.

Хеш-функции и булева алгебра

Еще одним важным элементом являются хеш-функции, интегрируемые с древовидными графами. Мы уже разбирали хеш-функции в главе 4, а в подразделе «Операции и обозначения для реализации хеш-функций» рассмотрели логические операторы, полезные для их построения. Однако если в криптографии хеши используются для создания цифровых подписей, чтобы не раскрывать содержимое сообщения [*M*], то здесь они служат другой цели. *Объект запроса* называется *ключевым словом*. Ключевое слово также является результатом индексирования строки или группы строк в базе данных.

Эффективная поисковая система индексирует минимальное число строк (аналогично кодированию), в также распознает и удаляет дублирующиеся индексы. Кроме того, она шифрует сам запрос в ходе работы с зашифрованными данными.

CSE поддерживает сложные запросы с несколькими ключевыми словами. Для этого используются логические операторы ИЛИ, И и НЕ, изученные в главе 2, в разделе «Понятия и операции в булевой алгебре».

Например, если мы хотим, чтобы поисковая система нашла этот раздел книги, то отправим ей следующий запрос:

[хеш-функции И булева алгебра].

После шифрования всего содержимого книги и разделения его на главы (можно предположить, что каждая из них представляет собой зашифрованный файл) мы ожидаем получения в ответ файла (или файлов), содержащего три ключевых слова: «хеш-функции», «булева» и «алгебра», связанных оператором И:

[хеш-функции И булева И алгебра].

Иначе говоря, система вернет все, что в данный момент содержит ключевые слова «хеш-функции + булева + алгебра» вместе (исключая, например, разделы, содержащие только одно из них).

В этом случае фраза, которая составлена из нескольких ключевых слов, связанных оператором И, вернет именно этот раздел в зашифрованном виде. Используя закрытый ключ расшифрования, мы можем расшифровать и прочитать найденный текст.

Именно так работает CSE. Более того, наша система поддерживает поиск по зашифрованным данным с помощью всех трех операторов — ИЛИ, И и НЕ.

Например, для поиска по зашифрованному тексту мы можем ввести следующий запрос:

[булева И алгебра НЕ хеш-функции].

Поисковая система найдет все элементы, содержащие слова «булева» и «алгебра» вместе, исключая «хеш-функции». В таком случае результатом будет раздел из главы 2, в котором описана булева алгебра.

Запросы в CSE нечувствительны к регистру: неважно, напечатаем мы «Хеш-функции» или «хеш-функции», это не повлияет на результат поиска. Кроме того, как вы могли заметить, я заключил запрос в квадратные скобки, потому что даже сами запросы в CSE зашифрованы для защиты конфиденциальности.

Секрет эффективности поиска ключевого слова заключается в сочетании древовидного индекса, хеш-функций и булевой алгебры для оптимизации пути к *целевому ключевому слову*.

Наша исследовательская группа Cryptolab изобрела уникальную технику, которая обеспечивает поиск зашифрованного ключевого слова в неструктурированных

зашифрованных файлах в среднем за 0,35 секунды практически независимо (–%) от количества обрабатываемых данных.

Пришло время показать, как создавалась система CSE и каковы ее основные функции.

Объяснение CSE

Данные часто передаются по сетям и хранятся на удаленных серверах, которые могут быть незащищенными и неприватными. При этом требуется возможность просматривать, искать и обрабатывать их независимо от физического расположения. В ходе работы с конфиденциальными данными, например, в сфере здравоохранения критически важно сохранять безопасность и секретность на протяжении всего процесса. Современные технологии управления такими данными на удаленных серверах достигают этой цели с помощью неоптимальной комбинации методов: конфиденциальные данные обычно защищают локальным шифрованием, а затем передают на хранение в удаленное хранилище.

Когда поступает запрос на просмотр или поиск данных, они расшифровываются на удаленном сервере, после чего становятся доступными. Если требуется изменить данные, может понадобиться дополнительное шифрование. Такой подход хотя и работает, но не отличается эффективностью, так как тратит вычислительные ресурсы и оставляет конфиденциальные данные в открытом виде на удаленных серверах, часто принадлежащих сторонним облачным сервисам. Эту проблему можно предотвратить, передав зашифрованные данные обратно в локальное хранилище. Там их можно безопасно расшифровать, просмотреть, найти нужное, снова зашифровать и отправить обратно. Понятно, как этот подход защищает данные, однако это довольно долгий процесс, требующий больше пропускной способности и повышающий риск утечки при передаче по ненадежным сетям.

На практике это приводит к двум очень распространенным сценариям: с одной стороны, некоторые пользователи отказываются от своей конфиденциальности, чтобы сэкономить (как правило, облачные сервисы смартфонов привлекательны из-за своей дешевизны до тех пор, пока пользователь готов делиться личными данными). С другой — некоторые учреждения, работающие с конфиденциальными данными, вынуждены иметь собственные центры обработки данных. Для защиты от аварий (что характерно для критически важной инфраструктуры в целом) такие центры должны иметь географическое и технологическое резервирование, что еще больше увеличивает расходы на их содержание.

Отсюда возникает идея *шифровать данные перед отправкой* на серверы и *хранить их только в зашифрованном виде*. Таким образом, даже если хакер преодо-

леет все барьеры безопасности и украдет данные, он не сможет их прочитать. Техническая проблема с этим подходом заключается в том, что зашифрованные данные представляют собой черные ящики, то есть с ними нельзя работать без предварительной расшифровки. По этой причине принято помещать такие данные в защищенную среду, например в частный центр обработки данных, где их можно безопасно расшифровать, не опасаясь кражи хакерами. Однако и это не гарантирует полной безопасности, поскольку *доверенное* лицо может получить доступ к данным, прочитать их и скопировать на какой-нибудь внешний носитель, например на флешку.

На рис. 8.14 показана схема хранения зашифрованных данных во внешнем хранилище. Как можно заметить, ключи шифрования (ключ данных и первичный ключ) зашифрованы и хранятся на внешнем сервере, например в облаке, вместе с зашифрованными данными. При поисковом запросе эти ключи используются для расшифровки данных в зашифрованном хранилище. Этот процесс необходим при обычном шифровании для поиска в базе данных. Недостаток этой системы проявляется в момент расшифровки при выполнении поиска, так как раскрывается содержимое данных.

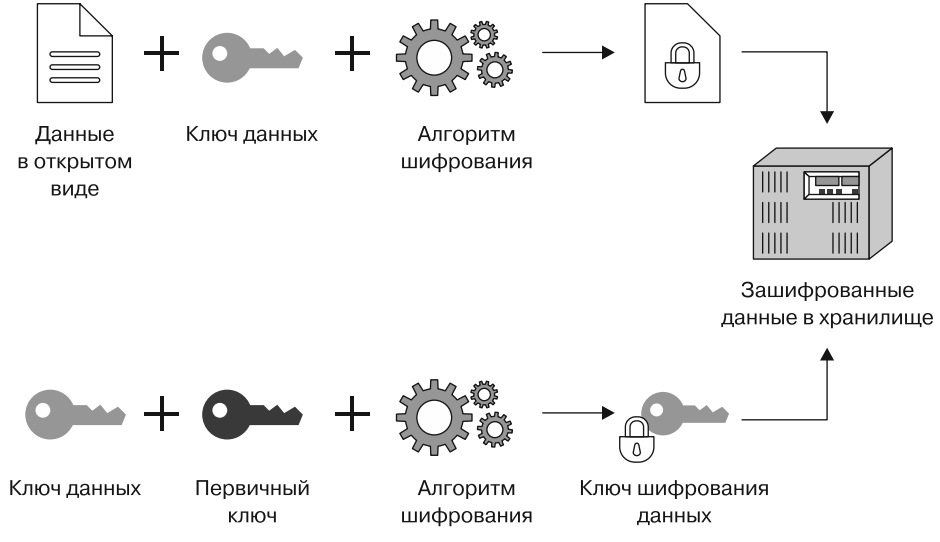


Рис. 8.14. Хранение зашифрованных данных традиционным способом (неэффективно и небезопасно)

Метод слепого поиска делает систему эффективной и совершенно безопасной. Именно это и может сделать CSE. Посмотрим, как работает и какие задачи решает эта поисковая система.

Новаторство CSE

CSE представляет собой технологическое решение, основанное на изоморфизме — *преобразовании, которое защищает информацию*, поскольку позволяет осуществлять поиск и просмотр зашифрованных данных, а также управление ими *без раскрытия информации*. Зашифрованные конфиденциальные данные, хранящиеся в публичном облаке, будут полностью доступны только их владельцам. При этом провайдеры могут обрабатывать данные с помощью зашифрованных запросов, не получая доступа к ним самим. CSE может работать независимо от типа шифрования, выбранного пользователем. С одной стороны, пользователи могут полностью владеть своими данными и обмениваться ими с провайдерами на честных условиях, а не по принципу «бери или уходи», который мы наблюдаем в настоящее время. Это ключевая ценность для следующего поколения цифровых граждан, которые смогут автономно обмениваться своими данными или продавать их. С другой стороны, государственные структуры, учреждения и критически важные инфраструктуры смогут защищать и хранить резервные копии своих данных с помощью множества провайдеров публичных облаков, что значительно снижает затраты и дает доступ к инструментам для анализа больших массивов данных.

Поиск и просмотр зашифрованных данных, а также управление ими — это уникальная технология, способная трансформировать любую бизнес-модель, связанную с созданием, передачей, хранением и анализом конфиденциальной информации.

CSE может стать технологической основой для поисковых систем *следующего поколения*. Данные можно искать и анализировать, при этом они остаются зашифрованными и целостными. Новые поисковые системы будут полностью соответствовать законодательству ЕС и США о конфиденциальности данных, например Общему регламенту по защите данных (General Data Protection Regulation, GDPR), и последним социально-экономическим тенденциям, касающимся целостности, конфиденциальности и безопасности данных. Масштабирование CSE будет иметь критически важное значение для дальнейшего развития поддерживающей инфраструктуры.

А теперь посмотрим, как работает CSE. На рис. 8.15 наглядно представлен цикл обработки запроса по зашифрованным данным, размещенным в публичном облаке.

На схеме изображены две стороны, но в действительности между ними не происходит обмена информацией. Сторона А (здесь представлена человеком) обычно является частным сервером, который хранит зашифрованные ключи и формирует запрос. Этот сервер просит сторону Б (облако) выполнить слепой поиск по зашифрованным данным.

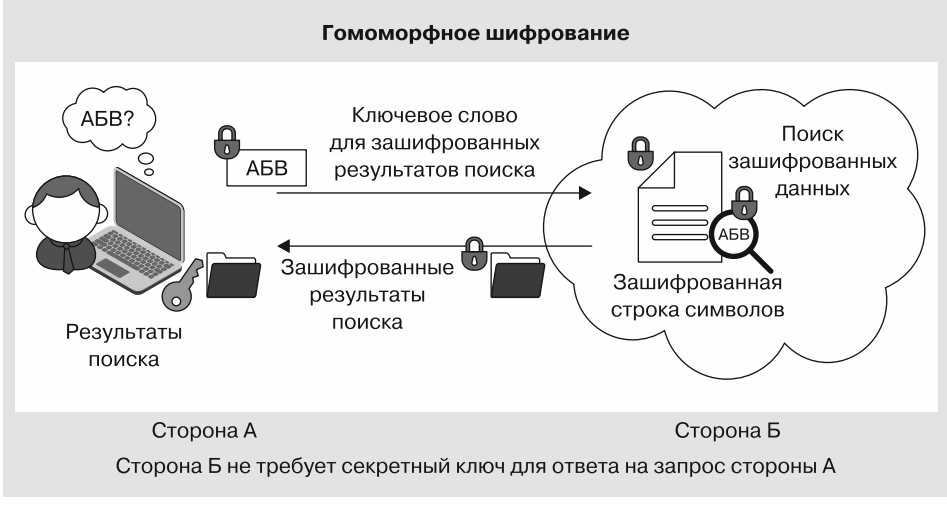


Рис. 8.15. Схема CSE

На рис. 8.16 показана упрощенная схема гибридной облачной архитектуры, применимой к CSE.

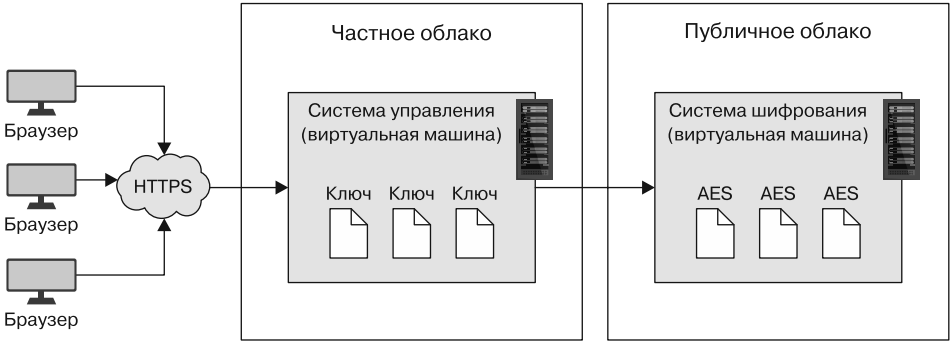


Рис. 8.16. Базовая гибридная архитектура CSE

В этой базовой схеме две виртуальные машины (ВМ) — *система шифрования* [СШ] (частный сервер) и *система управления* (СУ) (публичное облако) — используют симметричный алгоритм шифрования. CSE не привязана к конкретному алгоритму шифрования файлов. В первой версии задействуется алгоритм AES-256 (о нем рассказано в главе 2). В следующей версии к классическому алгоритму добавится алгоритм из квантовой криптографии, что позволит передавать файлы по квантовому каналу между [СШ] и (СУ).

Скобки указывают на вид среды, используемой в виртуальной машине:

- [СШ] обозначает частную защищенную среду (частное облако);
- (СУ) обозначает публичное облако или незащищенную среду.

Внутри обеих ВМ хранятся зашифрованные данные.

Как правило, HTTPS (см. рис. 8.16) используется только внутри защищенной частной зоны для подключения устройств пользователя к [СШ], расположенной в частном облаке, например на сервере компании.

Основная функция [СШ] — генерировать отдельные случайные ключи шифрования без повторений для каждого зашифрованного файла. Таким образом, невозможно ($-x$ %) найти два файла с одинаковым шифром, даже если их открытые тексты идентичны.

В задачи (СУ) входит хранение зашифрованных файлов и поиск по зашифрованным данным. Для большей эффективности поисковая система может быть размещена на третьей ВМ, полностью отделенной от [СШ] и (СУ) (рис. 8.17).

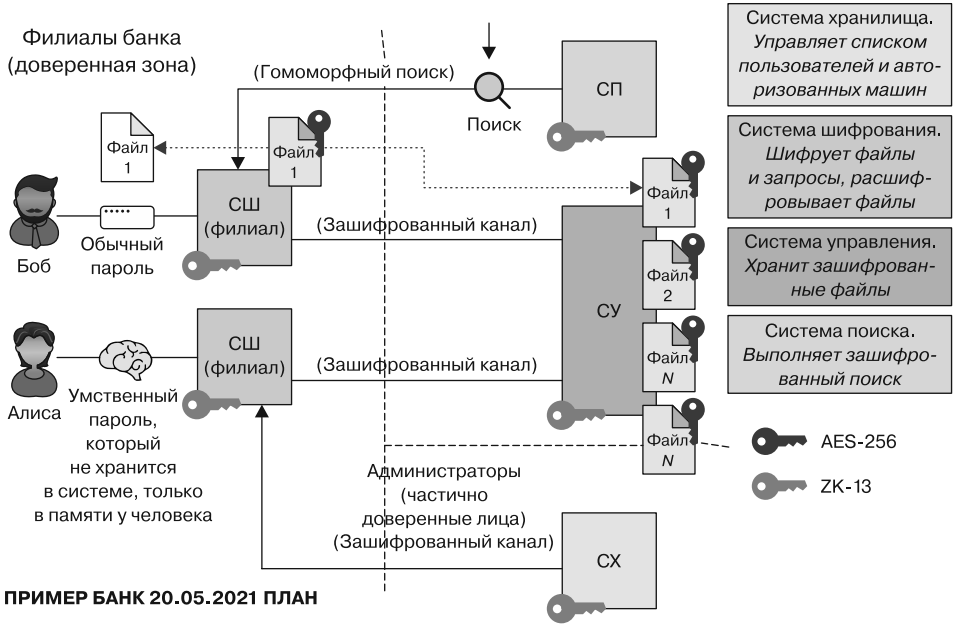


Рис. 8.17. Схема сложной архитектуры в CSE с несколькими [СШ], связанными с системой поиска (СП), системой управления (СУ) и системой хранилища [СХ]

На рис. 8.17 демонстрируется архитектура CSE, где несколько [СШ] подключены к нескольким (СП), которые идентифицированы с помощью доказатель-

ства с нулевым разглашением. Это обеспечивает *доказательство подлинности* в сети виртуальных машин. В первой версии CSE использовался неинтерактивный протокол ZK13 (рассмотрен в главе 5), который контролирует подлинность ВМ, предотвращая возможные атаки посредника. В дальнейшем для идентификации ВМ будет внедрен специальный квантовый протокол аутентификации.

Теперь, когда мы рассмотрели схему работы CSE, предстоит выяснить ее вычислительную эффективность.

Анализ вычислительной эффективности CSE

Секрет слепого поиска заключается в объединении хеш-функций, алгоритмов шифрования и аутентификации, а также пользовательских интерфейсов в одну платформу.

С вычислительной точки зрения задача состоит в том, чтобы совместить максимальную безопасность, обеспечиваемую алгоритмами шифрования, расшифрования и аутентификации, а также политиками входа в систему и управления платформой, с эффективностью поиска в зашифрованных данных. Допустим, нужно выполнить слепой поиск по зашифрованной версии всей Википедии. В этом случае проблема заключается в объеме данных (выраженном в килобайтах, гигабайтах или терабайтах) и количестве обрабатываемых *ключевых слов*. Другими словами, мы имеем дело со сложной системой, представленной совокупностью объединенных в CSE таких элементов, как:

- количество зашифрованных файлов;
- время шифрования файлов;
- биты на файл и биты на весь обрабатываемый массив данных;
- количество обработанных ключевых слов;
- время выполнения одного запроса;
- аутентификация между человеком и виртуальной машиной;
- аутентификация между виртуальными машинами;
- время расшифровки найденного файла;
- время удаления данных из памяти системы.

Все эти элементы должны быть объединены в оптимальное решение, представленное уравнением для оценки *эффективности* (*E*) CSE:

$$\boldsymbol{E} = \boldsymbol{S} / \boldsymbol{T},$$

где E — эффективность системы; S — уровень безопасности криптографических алгоритмов; T — время обработки запроса.

Таким образом, чем выше уровень безопасности системы и чем меньше времени требуется на обработку запроса, тем эффективнее будет система.

В предложенном уравнении 0 — это минимальный уровень безопасности алгоритма, а 1 означает 100%-ную безопасность. Предположим, что система не использует никакого шифрования для защиты данных, тогда уровень *безопасности* $S = 0$. При квантовом шифровании мы, вероятно, будем иметь очень высокий уровень безопасности $S = 1$. Время поиска T — это переменная, которая принимает значения от 0 до бесконечности, даже если мы знаем, что время поиска ответа на запрос должно составлять не более 2 секунд. Следовательно, максимальной степенью эффективности будет очень высокий уровень безопасности, например $S = 0{,}999999$, а минимально возможное время выполнения одного запроса $T = 0{,}00001$. Таким образом, если мы хотим достичь максимальной эффективности, нам следует увеличить уровень безопасности и сократить время обработки.

Здесь предполагается, что данные передаются и обрабатываются через Интернет, что делает их уязвимыми для внешних атак и шпионажа. Однако нельзя сказать, что шифрование данных гарантирует абсолютную защиту системы. Если система работает в автономном режиме, то есть не подключенный к Интернету компьютер не передает информацию вовне и не обрабатывает ту, что поступает из Сети, она может иметь более высокую степень защиты, даже если данные не зашифрованы.

Обозначение (0 +) означает нижнюю границу переменной *T*, которая стремится к нулю, но никогда его не достигает.

Поскольку время является детерминированной оценочной переменной, проблема заключается в том, как определить наилучшие уровень безопасности и степень эффективности.

В CSE в качестве алгоритма симметричного шифрования для передачи файлов между [СШ] и (СУ) используется AES-256. AES, как мы видели в главе 2, является стандартом симметричного шифрования (действует в данный момент, но нельзя сказать, что останется стандартом в будущем).

Я попытался рассчитать уровень сложности AES-256 с точки зрения устойчивости к атакам и требуемых вычислений, учитывая доступные на сегодня вычислительные мощности. Очевидно, технологии развиваются стремительно и то, что считается безопасным сейчас, может перестать быть таковым через три или пять лет. Здесь мы рассматриваем только вычислительную мощность, но с точки зрения безопасности можно выбрать альтернативу — например, шифрование с помощью Twofish.

В следующем подразделе разберем логику расчета эффективности криптографического алгоритма, такого как AES-256.

Пример взлома методом грубой силы

Предположим, у нас есть четырехъядерный компьютер, скажем MacBook Pro 2015 года с процессором i7. Его производительность равна 1024 Мбайт/с, или 2^{30} байт/с.

Поскольку AES-256 использует блоки по 16 байт (2^4), в среднем один высокопроизводительный компьютер может обработать $2^{30} / 2^4 = 2^{(30-4)} = 2^{26}$ блоков в секунду. Если предположить, что только один компьютер постоянно работает в течение 60 с · 60 мин · 24 ч · 365,25 дня = 31 557 600 (секунд в году), то общее количество операций в год составит 31 557 600 · 2^{26} = 2 117 794 686 566 400.

Таким образом, нам предстоит изучить примерно 2117,8 трлн ключей.

Чтобы провести атаку грубой силой на AES-256, в среднем необходимо перебрать 2^{255} ключей.

Вам может повезти, и вы переберете меньше чем 2^{255} ключей, но это число остается огромным.

Таким образом, нам нужно разделить 2^{255} на 2117,8 трлн. В результате мы получим число в экспоненциальной форме — примерно 2,73 · 10^{61}. В полном виде это 27 337 893 038 406 611 194 430 009 974 922 940 323 611 067 429 756 962 487 493 203 года.

Это число также можно представить как 27 трлн трлн трлн трлн трлн трлн лет. Для сравнения: Вселенная существует всего 15 млрд лет!

Что, если объединить все компьютеры Земли для взлома AES-256?

Если вы увлекаетесь вычислениями или другими странными вопросами, связанными с большими числами, спросите у поисковой системы Wolfram Alpha: «Сколько компьютеров существует на Земле?»

Alpha — высокопроизводительная поисковая система на базе Wolfram Mathematica — даст ответ: 2 млрд компьютеров.

Также с помощью Wolfram Alpha вы можете проверить всю мою предыдущую аргументацию. Идем дальше и делим время, затрачиваемое одним компьютером, на 2 млрд, оптимистично предполагая, что все компьютеры на Земле обладают вычислительной мощностью четырехъядерного MacBook Pro i7.

Результат составит 13 650 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 лет.

Или 1,365 · 10^{52} года. Это огромное время несоизмеримо с тем, что нам нужно, даже если использовать все компьютеры на Земле.

Исходя из этого тезиса, мы должны присвоить AES-256 степень эффективности $S = 1$. Однако все не так просто, и я покажу это на обычном вопросе:

> *«Что, если вместо совокупной мощности всех обычных компьютеров воспользоваться квантовым компьютером?»*

Как мы увидим в главе 9, алгоритм Шора может решить задачу факторизации. Однако AES не основывается на ней. Так не может ли теоретически квантовый компьютер взломать AES?

Как вы, вероятно, догадываетесь, взлом AES — это задача, связанная с поиском чего-то в огромном количестве данных — в 2^{255} ключах, скрытых внутри алгоритма. Для этой цели мог бы подойти *алгоритм Гровера* (если его удастся реализовать на квантовом компьютере в ближайшем будущем). Все алгоритмы вроде AES, которые используют скрытые ключи и основываются на принципе запутывания и рассеивания Шеннона, теоретически уязвимы для квантовых компьютеров.

Я расскажу об алгоритме Гровера и квантовом поиске в главе 10. Возможно, в будущем появится лучший алгоритм для квантового поиска. Если это произойдет, вопрос будет уже не в увеличении или удвоении длины ключа, а в чем-то, что превышает возможности классической логики и математики.

Алгоритм Гровера показывает, как выполнять поиск в неструктурированных данных с помощью квантового компьютера.

Хорошая новость заключается в том, что CSE — это система, которая, по прогнозам, не будет зависеть от используемого алгоритма. Теоретически она может выдерживать квантовые атаки, заменяя некоторые алгоритмы внутри своего ядра, даже *гомоморфный алгоритм поиска, изобретенный командой Cryptolab, — HK16,* — для выполнения запросов к зашифрованным данным. Другими словами, будет заменено ядро системы. Его можно заменить *квантовым гомоморфным поиском*, который сможет более эффективно работать на квантовом компьютере.

Главное преимущество CSE заключается в том, что это система, изначально спроектированная как независимая от используемых алгоритмов. Теоретически она способна противостоять квантовым атакам за счет замены алгоритмов в ее ядре, включая даже гомоморфный алгоритм поиска HK16, разработанный командой Cryptolab для работы с зашифрованными данными. Таким образом, ядро системы можно заменить, например, на квантовый гомоморфный поиск, который работает на квантовом компьютере значительно эффективнее.

Наконец, я хочу ответить на вопрос: какова эффективность CSE сейчас?

Я не могу объективно оценить CSE, так как участвую в ее разработке. Тем не менее могу уверенно заявить, что при выбранных алгоритмах и среднем времени обработки запроса 0,35 секунды CSE демонстрирует превосходную

эффективность и по уровню безопасности, и по быстродействию. Это останется актуальным до тех пор, пока не изменятся условия, как мы уже обсуждали.

Теперь, разобравшись с эффективностью CSE, перейдем к практическому применению этой системы в реальном мире.

Области применения CSE

CSE применяется по большей части к *системам хранения зашифрованных данных* и *синхронизации и обмена файлами* (FSS) в облаке. Как я уже отмечал, облачное хранение зашифрованных данных имеет ключевое преимущество перед другими облачными сервисами — *гомоморфный поиск*. Интересным вариантом использования CSE в блокчейне являются *слепые транзакции*. Здесь гомоморфная поисковая система обезличивает клиентов системы и позволяет безопасно обрабатывать транзакции. В здравоохранении уже протестированы два решения на базе CSE: управление *электронными медицинскими картами* (electronic health records, EHR) в облаке и программа *CovidFree*. Если углубиться в детали, то технология CSE позволяет искать данные и анализировать большие массивы информации в EHR-системах без ущерба безопасности и приватности пациентов. CovidFree — это запатентованное в 2020 году приложение для борьбы с пандемией COVID-19, которое обеспечивает конфиденциальность и безопасность данных для отслеживания вируса. Оно мониторит распространение вируса и сдерживает его за счет контроля доступа в общественные места (аэропорты, поезда, автобусы, рестораны и т. д.), не собирая персональных данных пользователей.

Схема совместного использования данных в зашифрованной среде представлена на рис. 8.18.

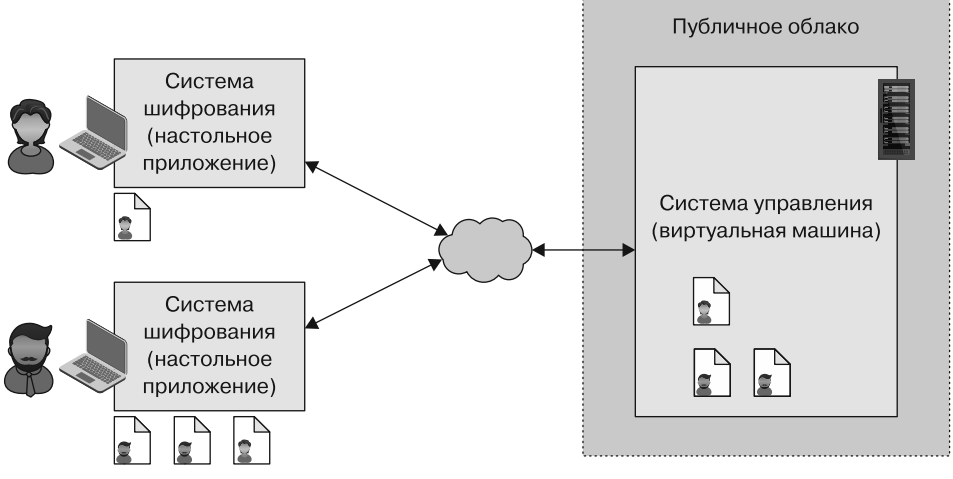


Рис. 8.18. Совместное использование данных в зашифрованном облаке

Другие области применения CSE связаны с видеонаблюдением и распознаванием изображений с помощью искусственного интеллекта. В таких случаях эта технология позволяет напрямую управлять видеоданными и анализировать зашифрованную информацию в реальном времени, обеспечивая защиту конфиденциальных данных в видеосфере.

Как видно на рис. 8.19, CSE применяется также в различных областях ИТ.

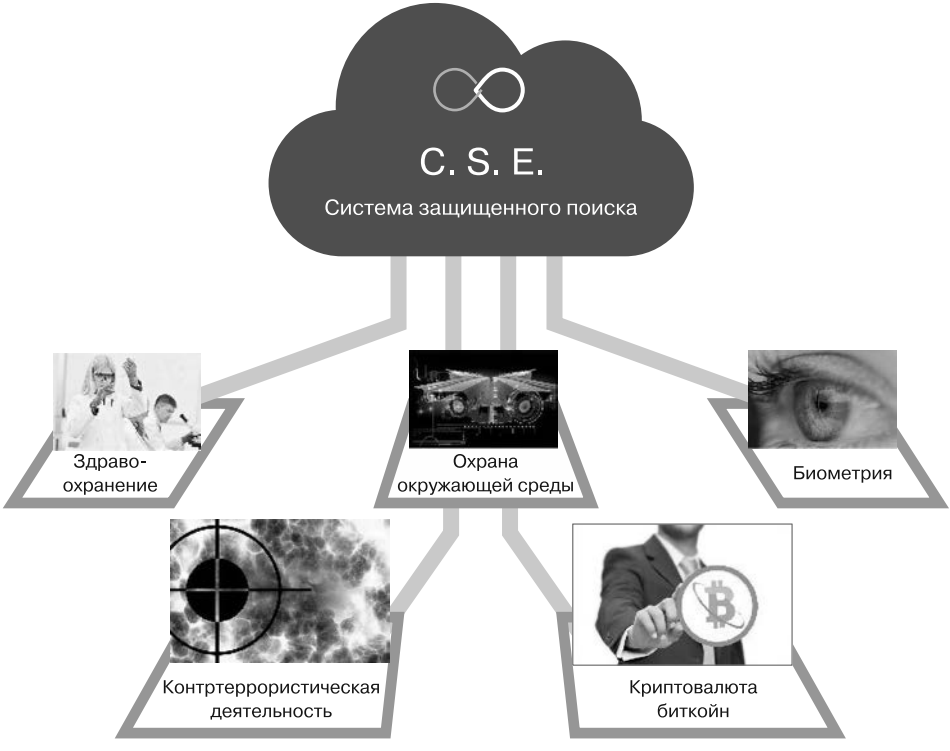


Рис. 8.19. Области применения CSE

Еще один интересный пример использования — автомобильная сфера. В будущем подключенные и автономные автомобили будут активно обмениваться данными, чтобы можно было, например, прогнозировать аварии или пробки на определенных маршрутах. При таком сценарии хакеры легко могут захватить контроль над автомобилем, если данные передаются в открытом виде. CSE предотвращает это (или как минимум снижает риск), поскольку все данные передаются и обрабатываются в зашифрованном виде. Даже при попытке перехвата данных в канале связи злоумышленники обнаружат лишь зашифрованные строки, а не читаемые сообщения.

В биологических науках CSE может быть особенно полезен для хранения и секвенирования ДНК[1] (рис. 8.20). Это крайне чувствительная область, сочетающая в себе возможность выявления болезней и многих личных характеристик человека, поэтому работа с такими данными требует особой осторожности. Как и в случае с информацией о нашем здоровье, физическом и психологическом состоянии, CSE позволяет хранить данные ДНК и осуществлять слепой поиск, предотвращая кражу или несанкционированный шпионаж.

Рис. 8.20. Слепая обработка данных ДНК с использованием CSE

Большинство описанных решений уже протестированы, а некоторые применяются на практике. Есть надежда, что следующее поколение поисковых систем обеспечит конфиденциальность и безопасность для всех, и CSE может стать тому примером.

Новый рубеж CSE и новый квантовый алгоритм передачи сообщений — QTM

В будущем мы планируем внедрить новый квантовый алгоритм, который я назвал *Quantum Transmission Message* (*QTM*), для симметричного шифрования и передачи данных в CSE. Сейчас этот алгоритм находится на этапе тестирования и скоро будет доступен в бета-версии.

Мы заменим AES квантовой передачей данных с помощью фотонов, основываясь на новой, революционной концепции: передавать можно не только ключ через квантовые алгоритмы (например, BB84, как мы увидим в главе 9), но и само сообщение, зашифрованное с помощью QTM.

---

[1] Секвенирование ДНК — это процесс определения точной последовательности нуклеотидов (A, T, G, C) в молекуле ДНК.

Этот алгоритм, изобретенный и запатентованный мной в 2023 году, вскоре будет применен в правительственных, наземных и спутниковых телекоммуникациях. Он подходит не только для шифрования и дешифрования CSE, но и для любых операций QTM — как на Земле, так и в спутниковой связи.

После испытаний в нашей лаборатории квантовых вычислений я представлю результаты работы этого алгоритма научному сообществу. Поэтому приглашаю вас в виртуальный тур в нашу новую лабораторию квантовых вычислений и криптографии, чтобы расшифровать код в этой книге.

Резюме

В этой главе вы познакомились с основными принципами гомоморфного шифрования и его основами, мы также проанализировали частичный гомоморфизм в RSA.

После этого вы изучили математические основы поисковых систем, в частности эйлеровы циклы. Затем мы углубились в теорию древовидных графов, разобрав конкретную структуру — код Хаффмана. В сочетании с хеш-деревьями и другими элементами этот метод отлично подходит для поиска и сжатия неструктурированных данных.

Наконец, мы проанализировали CSE, рассмотрев его ключевые компоненты и возможные способы применения в будущем.

Вы познакомились с основами CSE, которые могут стать отправной точкой для дальнейшего изучения. Если вы решите разобраться в реализации системы, имейте в виду, что поиск ведется не по открытым данным, а по зашифрованному хранилищу. Хранилище шифруется с помощью AES, и для поиска требуется специальная обработка данных. Впрочем, детальное погружение в этот механизм — тема для отдельного разговора.

Теперь, когда вы узнали о CSE и основных классических криптографических алгоритмах, мы переходим к новому измерению в криптографии — квантовой механике и квантовой криптографии. Эти темы раскрыты в следующих двух главах.

Часть IV
Квантовая криптография

Эта часть завершает книгу одной из самых значимых тем в криптографии — квантовыми вычислениями. Способность квантовых вычислений выполнять атаки методом грубой силы ставит под угрозу многие меры безопасности, так что эта тема является важнейшей для любого будущего криптографа. Чтобы вы поняли, как элементы квантовых вычислений применяются в криптографии, мы рассмотрим использование алгоритма Гровера для квантового поиска.

9 Основы квантовой криптографии

В этой главе я объясню основы *квантовой механики (Q-механики)* и *квантовой криптографии (Q-криптографии)*. Эти темы требуют определенных знаний в области физики, модульной арифметики, криптографии и логики.

Q-механика связана с явлениями, которые находятся за пределами обычного человеческого опыта. Поэтому большинству людей может быть сложно понять эту теорию и поверить в нее. Мы начнем со знакомства с причудливым миром Q-механики, а затем рассмотрим Q-криптографию, которая является прямым следствием этой теории.

Мы также проанализируем квантовые вычисления и алгоритм Шора. Последний довольно сложен, поэтому я разбил его на пять этапов: инициализация кубитов, выбор случайного числа, проведение квантового измерения, поиск правильного варианта и факторизация. Мы, конечно же, разберем каждый этап отдельно.

Наконец, мы поговорим о том, какие алгоритмы хорошо работают в рамках постквантовой криптографии.

Введение в Q-механику и Q-криптографию

До сих пор в книге криптография всегда следовала законам логики и строгим законам математики. Теперь мы подойдем к ней с другой стороны: от логики классической арифметики перейдем в новое измерение, где обычные физические события, как известно, совершенно иные и во многих случаях контринтуитивные.

Нам предстоит столкнуться с наукой, о которой Нильс Бор (один из отцов Q-механики) сказал: «Тот, у кого не кружится голова при размышлениях о квантовой механике, еще не понял ее». Эйнштейн был не согласен с Бором и назвал теорию квантовой запутанности воображаемой теорией.

Вот несколько моментов, касающихся Q-криптографии и Q-механики, которые необходимо принимать во внимание.

- Вся криптография даже самая развитая, надежная и сложная, опирается на два элемента.
  - Трудность ее взлома зависит от алгоритма и лежащей в его основе математической задачи: факторизации, дискретного логарифма, умножения полиномов и др.
  - Размер принятого ключа. Иными словами, длина ключа или область чисел, в которой работает алгоритм для генерации ключа, определяет уровень вычислительного предела, ниже которого любой алгоритм оказывается уязвимым. Например, если мы определим открытый ключ (N) в RSA как $N = 21$, то даже ученик начальных классов сможет найти его множители, или два секретных числа (закрытых ключа) — $p = 3$ и $q = 7$, так как $21 = 3 \cdot 7$. И наоборот, если мы зададим область операций 10^{1000}, то задача факторизации становится трудноразрешимой.
- В Q-криптографии проблема не зависит ни от одного из этих элементов. В основе этого вида криптографии (в классическом смысле) лежит не какая-то *математически сложная задача*, а особая и даже странная гипотеза: в Q-механике элемент (частица, например фотон) может находиться в состоянии 1 и 0 одновременно. Он может присутствовать сразу в двух местах. Невозможно определить состояние частицы, пока оно не будет измерено. Как мы скоро узнаем, само понятие времени в Q-механике теряет свой смысл, поскольку свойство обусловленности является не обязательным, а лишь возможным.

Прежде чем приступить к самой гипотезе, выдвинутой здесь, рассмотрим эксперимент, который изменил представление людей о микрочастицах.

Эксперимент, изменивший историю кванта

Первый эксперимент, навсегда изменивший историю науки в этой области, был поставлен ученым-энциклопедистом Томасом Юнгом в конце XVIII века. Кроме того, Юнг был первым, кто расшифровал некоторые иероглифы, поэтому его можно считать и криптографом. Ученый часто гулял возле небольшого озера, где плавали утки. Он заметил, как волны, создаваемые плавающими утками, взаимодействуют и пересекаются, образуя рябь. Это напомнило ему о схожем поведении световых волн, что и вдохновило его на дальнейшие исследования.

Эксперимент, представленный на рис. 9.1, показывает, что световые волны ведут себя как частицы[1] и образуют множество мелких полос на экране позади

[1] Вы также можете самостоятельно ознакомиться с понятием «корпускулярно-волновой дуализм», который хорошо дополнит дальнейший рассказ об этом эксперименте.

двух щелей, через которые проходит свет, вместо образования всего двух крупных полос.

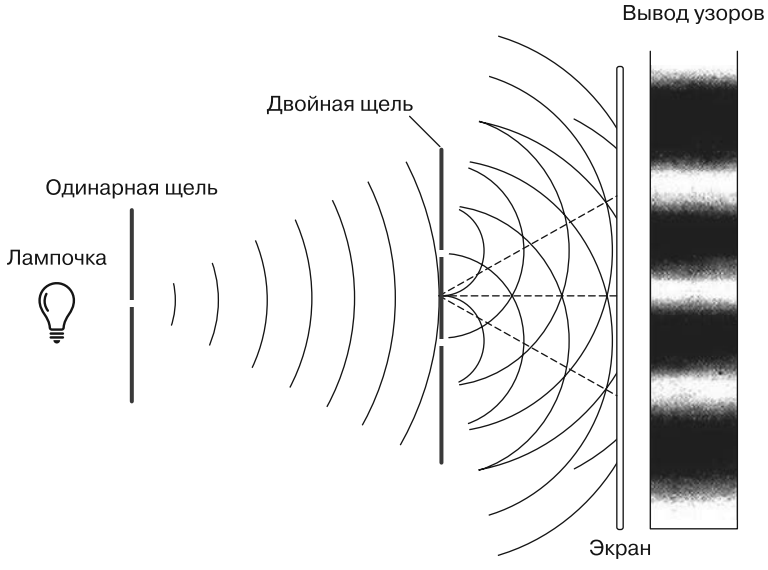


Рис. 9.1. Эксперимент Юнга со светом

Пока в этом эксперименте не видно ничего парадоксального. Однако если провести его, направляя через две щели лишь по одному фотону — частице света — с равными интервалами (например, каждую секунду), результат удивит. Вместо ожидаемых двух темных линий на экране за щелями возникает целый ряд полос. Они напоминают узоры, создаваемые световыми волнами.

В следующем разделе я приведу измененный вариант этого эксперимента, который позволит вам понять работу Q-механики (или, что более вероятно, вскружит вам голову).

Мысленный эксперимент для понимания элементов Q-механики

В этом разделе мы попытаемся копнуть глубже и познакомиться с Q-механикой, проведя мысленный эксперимент. Он похож на оригинальный эксперимент Юнга: стена с двумя щелями и экран на задней стенке. На протяжении всей главы мы поэтапно воссоздадим эксперимент и поймем парадоксы Q-механики.

Предположим, мы выпустили фотоны через две щели, показанные на рис. 9.2. Представим фотоны в виде маленьких шариков.

Этап 1. Суперпозиция

Начнем с того, что выпустим шарики обычного размера (см. рис. 9.2). Каждый шарик с большой вероятностью будет проходить через какую-то из двух щелей. Допустим, что в конце первого раунда большинство шариков попадет на экран, образовав две соответствующие щелям прямые линии.

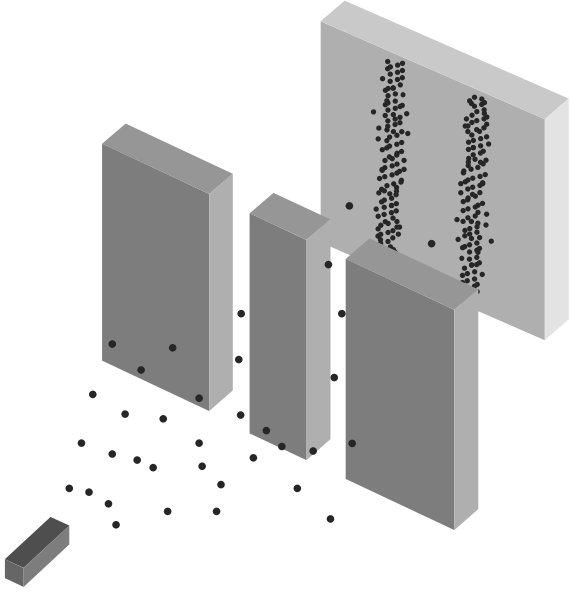


Рис. 9.2. Шарики, ударяющиеся об экран, образуют две линии позади щелей

Теперь предположим, что две щели стали очень узкими, а шарики — очень маленькими. В этом случае мы имеем дело с фотонами. Мы увидим, что результат будет совершенно другим. В отличие от обычных шариков, шарики, размеры которых сопоставимы с размерами фотонов, образуют полосатый узор (рис. 9.3).

Как видите, после измерения частицы создают полосатый узор, при условии, что и щели, и частицы достаточно малы. Результатом этого явления становятся *волны*.

Когда волна проходит через щель, она распространяется с другой стороны. Если щелей две, то образуются две взаимодействующие волны. Они создадут эффект полосатого узора, как мы видим на экране в эксперименте Юнга, воспроизведенном на рис. 9.1.

Проблема заключается в том, что если мы запускаем только один шарик (частицу) за раз, то все равно получается полосатый узор. (Если коротко: причина этого связана с энергией частиц.) Так как же возможно, чтобы один шарик, каким бы маленьким он ни был, прошел одновременно сквозь две щели?

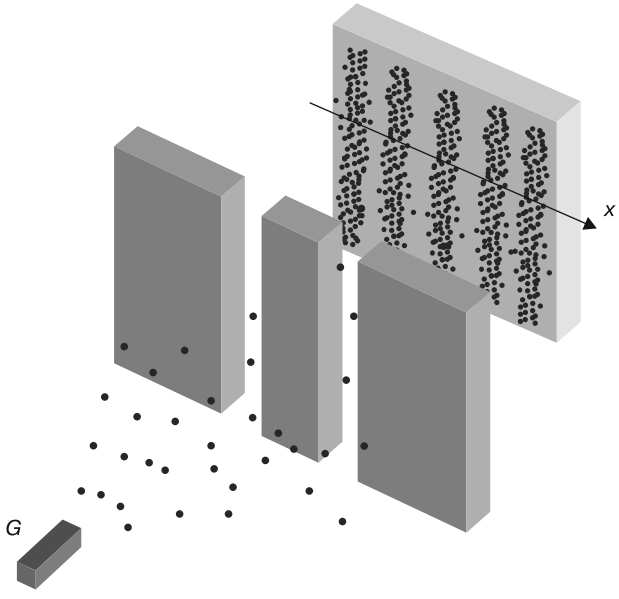


Рис. 9.3. Очень маленькие шарики образуют полосатый узор

Давайте на мгновение приостановимся и поразмыслим над этим первым важным свойством. Если мы обозначим 0 состояние фотона, проходящего через левую щель, и 1 — фотона, проходящего через правую щель, то можем сказать, что эти два фотона проходят одновременно через две щели, находясь сразу в состоянии 0 и 1. Это свойство Q-механики, которое называется *суперпозицией*.

Если закрыть одну щель и запускать шарики через открытую вторую щель, то на экране появится картина, похожая на ту, что вызывает попадание обычных шариков на экран сразу за этой щелью. Однако если открыть обе щели, полосатый узор восстановится.

Суперпозиция может предложить ви́дение, когда частица находится в разных состояниях одновременно. Однако в другой области физики некоторые ученые выдвинули иную интерпретацию этого эксперимента, называемую *копенгагенской интерпретацией* или *интерпретацией многих миров*. Она заключается не в признании существования суперпозиции, а в не менее странной концепции *мультивселенной*. Согласно этой концепции различные состояния частицы представлены множеством вселенных.

Альтернативное ви́дение мультивселенной предлагает философскую теорию квантовой реальности, которую можно даже перенести в нашу жизнь. Считается, что в различных параллельных мирах существуют наши копии во множестве состояний, играющие разные роли.

Продолжим эксперимент и поговорим о другой важной черте Q-механики — неопределенности.

Этап 2. Принцип неопределенности

Теперь проверим, что произойдет, если перед каждой щелью поставить детектор. Ожидается, что на обоих детекторах шарики отобразятся одновременно, потому что, как мы видели ранее, если выпустить только одну частицу, она окажется в состоянии суперпозиции. Однако в данном случае этого не происходит.

Если мы выпустим шарики, достаточно маленькие, чтобы их можно было сравнить с малыми частицами, то увидим, что они отображаются только на одном из детекторов, но никогда на обоих. Любая попытка выяснить, через какую из двух щелей пройдут шарики, приводит к колебанию, что и вызывает исчезновение полосатого узора. Таким образом, эта попытка заставляет шарики проходить только через одну щель, а не через обе.

Это еще одно фундаментальное свойство Q-механики: когда фотоны находятся в состоянии суперпозиции (или в мультивселенной), они одновременно существуют в состояниях 0 и 1. Однако при наблюдении фотоны меняют свое поведение и выбирают только одно состояние, 0 или 1.

Наблюдатель в квантовой механике — это все, что может обнаружить квантовую частицу. Наблюдение также называется измерением, поскольку представляет собой сбор данных.

Свойство неопределенности строго коррелирует с суперпозицией. Мы вернемся к этому позже, а пока продолжим наш эксперимент.

Важно! В этом мысленном эксперименте упростим ситуацию, представив, что видим фотоны глазами. Это, конечно, невозможно в реальном мире, так как фотоны можно наблюдать только с помощью детекторов. В реальности определение состояния суперпозиции происходит с помощью квантового измерения, которое представляет собой взаимодействие между квантовой системой и измерительным устройством.

Если закрыть глаза и не наблюдать за прохождением шариков через щели, то, даже если перед щелями поставить детекторы, шарики все равно будут в суперпозиции. Только если открыть глаза и посмотреть на них, состояние неопределенности прекращается и шарики выбирают состояние 0 или 1. Следовательно, шарики, с которыми мы имеем дело, должны отличаться от волн.

Состояние классической волны *детерминировано* и порождает свои эффекты в любых условиях независимо от того, наблюдаем мы эксперимент или нет. В Q-механике все иначе: частицы ведут себя странно, следуя правилам, которые выходят за рамки классической физики. Эта проблема порождает еще одно интересное следствие в физике, связанное с философией, но оно выходит за рамки тематики книги.

Теперь предположим, что мы поместили два объекта за двумя щелями. Выпуская шарики через щели, мы можем представить, что они сталкиваются с одним из двух объектов. Это действительно так, если наблюдать за происходящим. Однако если закрыть глаза и не смотреть, то вероятность того, что шарики столкнутся с одним или другим объектом, одинакова. Другими словами, эти два объекта можно сравнить со знаменитым мысленным экспериментом с *котом Шредингера*, в котором кот одновременно и жив, и мертв, пока мы на него не посмотрим.

Согласно математике, описывающей вероятность волн, ни один из исходов не является однозначным. Только когда мы откроем глаза и посмотрим, мы сможем узнать, жив объект (1) или мертв (0). Более того, очень трудно принять логику, согласно которой мы можем приписать определенную *вероятность* частице, находящейся в конкретном месте Вселенной, и поверить в то, что она проявляется только во время наблюдения.

Еще раз отмечу, что под наблюдением я подразумеваю человеческие глаза, поскольку речь идет о нашем эксперименте, но в действительности это квантовое измерение, проводимое специальным прибором.

Но, несмотря на безумную логику, все эксперименты, которые мы рассматривали до сих пор, показывают, что свойство неопределенности справедливо! *Вероятность* — еще один фундаментальный элемент, который необходимо учитывать, когда речь идет о Q-механике. Вероятность одновременного нахождения объектов в том или ином состоянии и в том или ином месте побудила Эйнштейна быть ярым противником квантовой теории и, в частности, Шредингера. Относительно этой логики Эйнштейн заметил: «Бог не играет в кости со Вселенной».

Когда мы говорим о вероятности в Q-механике, ключевым свойством, демонстрирующим вероятностную природу частицы, является спин. Давайте разберемся, что такое спин и почему он так важен для наших исследований.

Этап 3. Спин и запутанность

Чтобы объяснить, как частица может находиться в двух состояниях одновременно, нужно ввести понятие *спи́на* (spin).

В литературе под спином понимается *собственный момент импульса тела*. Его можно объединить с орбитальным моментом для вычисления полного момента импульса.

Направление спина частицы можно описать как воображаемую стрелку, пересекающую частицу в разных направлениях. Частицы, вращающиеся в противоположных направлениях, будут иметь стрелки, направленные в противоположные стороны (рис. 9.4).

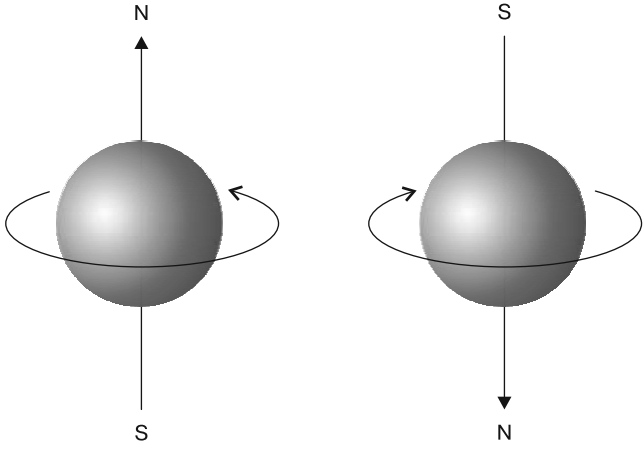


Рис. 9.4. Спин

Как вы поймете позже, когда мы будем разбирать квантовый обмен ключами, спин играет важную роль в Q-криптографии. Содержимое частицы или информация, которую она может нести, зависит от ее спина. Это связано с информацией, заключенной в направлении спина, то есть в том, могут ли Алиса и Боб определить биты квантово распределенного ключа. Мы поговорим об этом позже.

Теперь вернемся к концепции квантовой информации. Мы знаем, что в классическом мире информация состоит из букв и цифр. Однако можно передавать и получать информацию дискретным способом, например: ДА или НЕТ, ВВЕРХ или ВНИЗ, ИСТИНА или ЛОЖЬ. Мы также можем дискретизировать информацию, сократив ее до бита в реальном мире. То же самое можно сделать и в Q-механике, но с использованием *квантового бита (кубита)*.

Рассмотрим пример с монетой, стороны которой называются орлом и решкой, где физический объект может представлять бит в состоянии 0 для орла или 1 для решки. В кубитах мы используем обозначения Дирака, которые описывают состояние бита с помощью значений бра и кет — $|0\rangle$ и $|1\rangle$.

Мы уже знаем, что в суперпозиции состояние частицы может быть одновременно 1 и 0, но при наблюдении она вынуждена выбирать одно из них. Это явление называется *коллапсом волновой функции* и происходит, когда волны, изначально находящиеся в состоянии суперпозиции, сводятся к одному состоянию в результате взаимодействия с внешним миром, когда кто-то следит за каналом связи между двумя частицами.

Когда две частицы взаимодействуют между собой, их спиновые состояния могут запутаться, что ученые назвали *квантовой запутанностью*.

В широком смысле *квантовая запутанность* возникает, когда одна группа частиц начинает зависеть от состояния другой группы. Это происходит, когда группы взаимодействуют друг с другом или находятся в непосредственной близости друг от друга.

Два запутанных электрона могут иметь противоположные спины, как мы вскоре увидим в нашем примере. Это означает, что если один из них имеет спин «вверх», то второй сразу же принимает спин «вниз» (рис. 9.5).

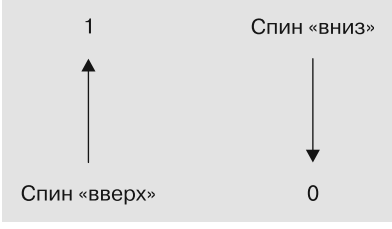


Рис. 9.5. Спины «вверх» и «вниз» электрона

Даже если мы разделим два электрона так, что они окажутся на некотором расстоянии друг от друга, и измерим спин одного из них, то сразу же узнаем спин второго. Например, если мы измерим спин электрона в лаборатории в Калифорнии и увидим, что он имеет значение «вверх», значит, спин другого электрона во второй лаборатории в Нью-Джерси будет иметь значение «вниз». Неважно, как далеко друг от друга находятся эти две частицы.

Таким образом, мы можем сказать, что определение запутанного состояния происходит быстрее скорости света!

У данной теории был яростный противник — Эйнштейн, который назвал это явление жутким действием на расстоянии. Тем не менее, несмотря на слова Эйнштейна, запутанность очень полезна для нашего технологического мира, как мы увидим совсем скоро.

Рассмотрим эксперимент, демонстрирующий реальность запутывания информации.

В 2015 году группа ученых под руководством Рональда Хансона из Делфтского технологического университета (Нидерланды) провела эксперимент, показывающий, что две запутанные частицы могут обмениваться информацией быстрее скорости света.

В ходе эксперимента в лабораториях *А* и *Б*, отстоящих друг от друга на 1280 м, были установлены два алмаза. Из точек *А* и *Б* были выпущены электромагнитные импульсы, вызывающие излучение фотона, находящегося в состоянии запутанности со спином электрона. Эксперимент показал: когда две частицы одновременно достигли пункта назначения *В*, где установлен детектор, их запутанность была передана электронам. Карта эксперимента показана на рис. 9.6.

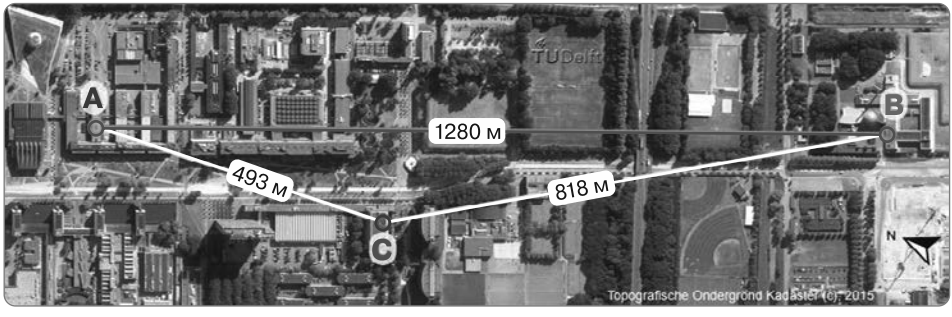


Рис. 9.6. Эксперимент с запутыванием (Делфтский технологический университет)

Недавно группа выполнила важное техническое усовершенствование: экспериментальная установка теперь всегда готова к созданию *запутанности «по требованию»*. Это означает, что состояние запутанности между двумя частицами в будущем может быть получено по запросу, что позволяет разрабатывать квантовые приложения, считавшиеся ранее невозможными. Если вам интересно узнать больше об этом эксперименте, можете перейти на официальный сайт Делфтского университета: https://www.tudelft.nl/2018/tu-delft/delftse-wetenschappers-realiseren-als-eersten-on-demand-quantum-verstrengeling.

Далее приведены примеры сфер, где уже используется странный феномен запутывания:

- часы и все приложения, связанные с фондовым рынком и GPS, которые задействуют свойства запутанных фотонов;
- созданный в Университете Хоккайдо микроскоп, который использует свойства запутанных фотонов для тщательного изучения микроскопических элементов;
- квантовая телепортация, связанная с переносом информации;
- квантовая биология, включая эксперименты с ДНК и другие клинические исследования;
- наш объект изучения — Q-криптография.

Сочетание всех этих элементов и свойств создает прорывное и захватывающее приложение — *квантовое распределение ключей* (quantum key distribution, QKD).

Прежде чем рассмотреть его более подробно, поговорим о том, как возник этот процесс и какие элементы в нем сочетаются.

Происхождение Q-криптографии: квантовые деньги

Теперь, когда вы познакомились с основами Q-механики, перейдем к Q-криптографии. Любопытная история ее применения началась в 70-х годах прошлого века, когда у аспиранта Колумбийского университета Стивена Визнера возникла идея создать особый вид денег, которые теоретически невозможно подделать, — *квантовые деньги*. В основном квантовые деньги Визнера опирались на квантовую физику, связанную с фотонами.

Предположим, у нас есть группа фотонов, движущихся в одном направлении по заранее заданной оси. Перемещаясь в пространстве, фотон имеет направление колебаний, известное как *поляризация*. На рис. 9.7 показано, о чем идет речь[1].

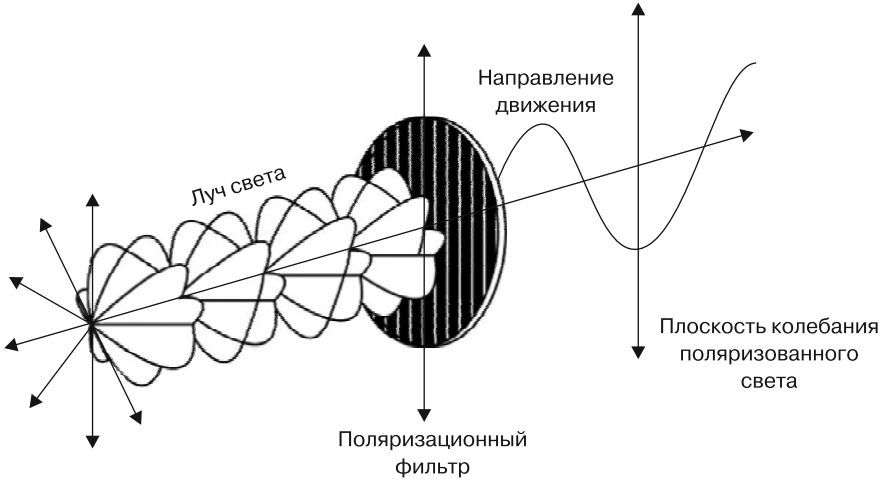


Рис. 9.7. Поляризация фотона

[1] В магазинах вы также могли видеть поляризационные очки или фильтры для объективов фотоаппаратов, которые работают по тому же принципу.

Как видите, фотоны вращаются во всех направлениях, включая диагональные, поскольку исходное состояние неполяризованного фотона — это суперпозиция колебаний по горизонтали и вертикали. Однако если мы поместим фильтр — *поляроид*, ориентированный вертикально, то увидим, что фотоны, ориентированные вертикально, проходят через него в 100 % случаев.

При этом фильтр будет блокировать все фотоны, поляризованные перпендикулярно ему, и около половины всех фотонов, ориентированных диагонально. Более того, фотоны сталкиваются с квантовой дилеммой: проходя через фильтр, они подвергаются измерению, поэтому им приходится *выбирать* ориентацию — вертикальную или горизонтальную.

Эта особенность приводит к странным результатам при рассмотрении эксперимента с поляризованными линзами. На рис. 9.8 показано странное явление: третья линза (поляризатор 3), добавленная к двум имеющимся, позволяет увидеть больше света, чем раньше.

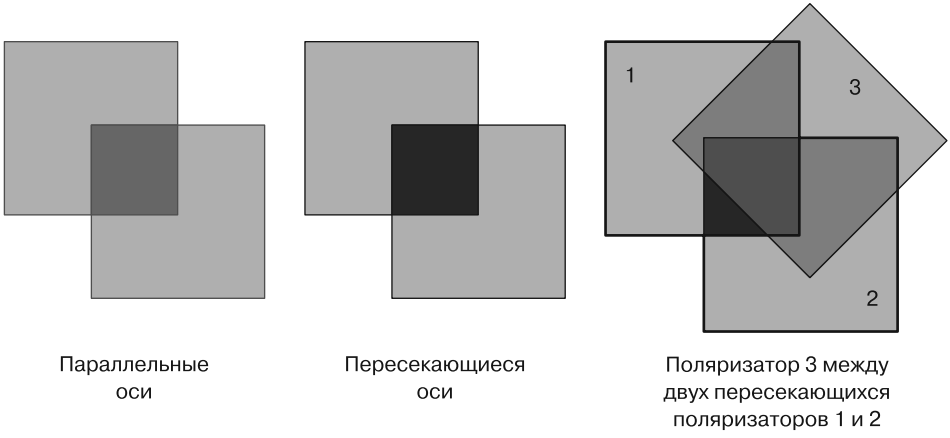


Рис. 9.8. Эксперимент с поляризованным фильтром

В этом и заключается особенность Q-механики: по нашей логике добавление еще одной линзы должно уменьшить количество света и затемнить изображение, но все происходит наоборот.

Теперь, когда вы знаете, что такое поляризация и каково ее странное поведение, когда мы имеем дело с фотонами и квантовой механикой, перейдем к одному из первых приложений, реализованных в квантовой механике, — квантовым деньгам.

Итак, Визнер предложил создать специальную банкноту, которая содержит 20 *световых ловушек*, ведущих себя как фильтры. Эти ловушки расположены так, что поляризации фотонов ориентированы в четырех направлениях, как показано на схеме однодолларовой купюры (рис. 9.9). Фильтры могут определять (путем сопоставления с серийным номером банкноты), являются ли деньги настоящими.

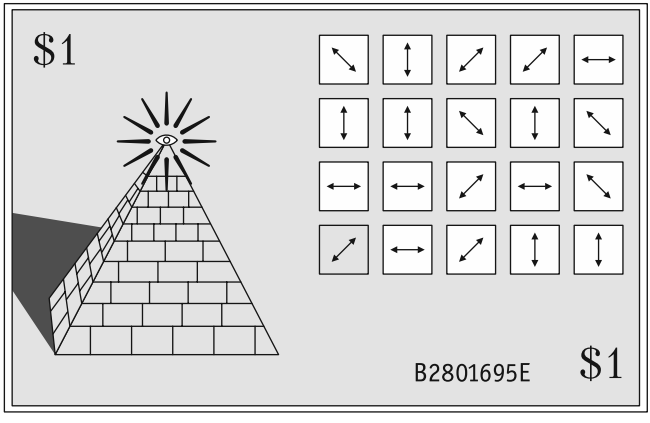


Рис. 9.9. Квантовая однодолларовая банкнота, предложенная Визнером

Только банк, выпускающий банкноты, может определить, настоящие они или нет. Благодаря свойствам Q-механики, в частности принципу неопределенности, при всем желании подделать такую банкноту было бы невозможно. Рассмотрим причину этого.

На рис. 9.9 показано, куда направлены световые ловушки, но в реальности они невидимы. Предположим, Ева хочет скопировать банкноту. Она может скопировать серийный номер на подделываемой банкноте и попытаться выяснить ориентацию ловушек.

Если Ева решит использовать вертикальный фильтр для определения поляризации, она сможет обнаружить все вертикальные фотоны, будучи уверенной, что каждый горизонтальный фотон будет заблокирован. Как быть с остальными? Некоторые из них пройдут через фильтр, но точно узнать их поляризацию невозможно. Поэтому единственное, что может сделать Ева, — случайным образом определить поляризацию всех фотонов так, чтобы она отличалась от вертикальной и горизонтальной.

Теперь возникает проблема: если банк проверит поддельную банкноту, он узнает, что она фальшивая. Из-за принципа неопределенности только банк знает, какой фильтр соответствует конкретному серийному номеру.

Визнера постоянно игнорировали, когда он говорил о своей идее квантовых денег. Его изобретение неосуществимо на практике из-за непомерно высокой стоимости создания такой банкноты. Однако его мысль заложила фундамент следующей темы исследования — *квантового распределения ключей* (quantum key distribution, QKD).

Прежде чем продолжить, сделаем небольшое замечание о квантовых деньгах: в своем изобретении Визнер намеревался дать банкам возможность обнаруживать попытки подделки банкноты, но эта концепция имеет изъян. В реальном мире проблема касается не банков, ведь гораздо важнее, чтобы пользователи при одноранговом взаимодействии сами могли определять подлинность банкнот. Поэтому, на мой взгляд, если квантовые деньги когда-нибудь появятся, они будут реализованы каким-то другим способом.

В конце концов Визнер нашел человека, который прислушался к его мнению, — Чарльза Беннета, старого друга, участвовавшего в различных исследовательских проектах и чрезвычайно любопытного в науке. Чарльз поговорил с Жилем Брассаром, в то время исследователем в области компьютерных наук в Монреальском университете. Заинтересовавшись этой проблемой, они нашли способ реализации QKD, о котором я расскажу далее.

QKD: BB84

В 1984 году Чарльз Беннет и Жиль Брассар разработали версию QKD под названием BB84. Теперь мы можем использовать изученные свойства Q-механики для описания техники распределения битов (или, лучше сказать, кубитов) по квантовому каналу.

Прежде чем углубиться в эту тему, вспомним, что такое кубит. Кубиты несут квантовую единицу информации. Как и обычные биты (0) и (1), кубиты — это математические объекты, над которыми можно производить вычисления и операции.

Для определения кубита мы будем использовать *двумерное векторное комплексное пространство единичной длины*. Проще говоря, кубит — это единичный вектор в двумерном пространстве. Пока мы можем представлять кубит как поляризованный фотон, подобный использованному в эксперименте с запутанностью из предыдущего раздела. Я уже ввел обозначения для представления кубитов в виде бра и кет — $|0\rangle$ и $|1\rangle$.

Не буду слишком углубляться в математику Q-механики, однако она нам понадобится для анализа некоторых свойств кубитов. Если элемент несет некую информацию (как это делают кубиты), он связан с какой-то формой вычислительного процесса, в данном случае — с квантовыми вычислениями. Мы вернемся к этому понятию позже, когда будем говорить о квантовых компьютерах.

Геометрическое представление кубита приведено на рис. 9.10.

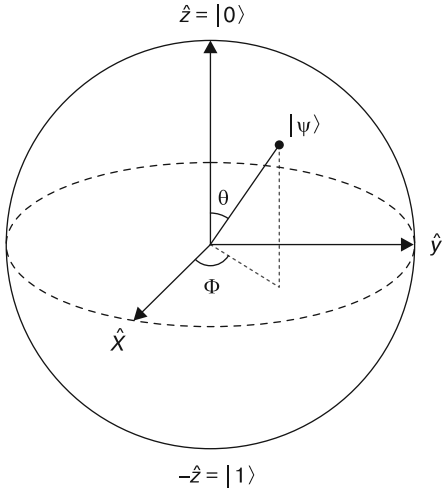


Рис. 9.10. Сфера Блоха — фундаментальное представление кубита

Разница между классическими вычислениями для всех операций с битами на основе машины Тьюринга и лямбда-исчислениями и квантовыми вычислениями существенна. Квантовые вычисления опираются на свойства суперпозиции и запутанности. При рассмотрении суперпозиции мы узнали, что кубиты могут быть представлены одновременно в двух состояниях — 0 и 1, в отличие от обычных битов, которые принимают только одно из двух этих значений.

На рис. 9.11 показано упрощенное математическое представление кубитов, которое позволяет лучше понять природу состояний 0 и 1.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Рис. 9.11. Кубиты в векторной форме

Таким образом, рассматривая кубит как единичный вектор, его можно представить следующим образом (с учетом рис. 9.11):

$$|\psi\rangle = a|0\rangle + b|1\rangle.$$

Здесь $|a|$ и $|b|$ представляют собой комплексные числа a и b, для которых выполняется условие:

$$|a|^2 + |b|^2 = 1.$$

Здесь нужно отметить важный момент: мы знаем только амплитуды вероятностей — коэффициенты $|a|$ и $|b|$, сумма квадратов которых равна 1, но никогда — как фотон окажется в состоянии суперпозиции, поэтому значения коэффициентов $|a|$ и $|b|$ непредсказуемы.

Этого достаточно для понимания алгоритма, лежащего в основе QKD.

Теперь, кратко познакомившись с основными понятиями квантового исчисления и операций с кубитами, вы готовы к изучению алгоритма квантовой передачи ключа.

Шаг 1. Инициализация квантового канала

Для обмена сообщениями Алисе и Бобу необходимы квантовый и классический каналы. Квантовый канал — это среда, по которой можно передавать фотоны. Его особенность заключается в том, что он должен быть защищен от внешней среды. Любая связь с внешним миром должна быть исключена. Как вы уже видели, изолированность необходима для создания суперпозиции и запутанности частиц.

Под классическим каналом подразумеваются обычный Интернет или телефонная линия, по которой Алиса и Боб могут в открытую обмениваться сообщениями.

Предполагается, что Ева (злоумышленник) может прослушивать классический канал, а также посылать и обнаруживать фотоны, как и Алиса с Бобом.

Поскольку интересной и по-настоящему инновационной частью системы является только квантовый канал, по которому Алиса и Боб обмениваются ключом, мы сосредоточимся на процессах, происходящих именно в нем. Затем рассмотрим особенности передачи сообщений по классическому каналу.

Шаг 2. Передача фотонов

Алиса начинает передавать Бобу группу битов. Эти биты кодируются с использованием базиса, который для каждого из них выбирается случайным образом по определенным правилам.

Для каждого бита существует два возможных базиса:

$$B_1 = \{|\uparrow\rangle, |\rightarrow\rangle\};$$

$$B_2 = \{|\nwarrow\rangle, |\nearrow\rangle\}.$$

Если Алиса выбирает B_1, она кодирует следующее:

$$B_1\colon 0 = |\uparrow\rangle \text{ и } 1 = |\rightarrow\rangle,$$

если B_2 —

$$B_2\colon 0 = |\nwarrow\rangle \text{ и } 1 = |\nearrow\rangle.$$

Каждый раз, когда Алиса передает фотон, Боб случайным образом выбирает базис B_1 или B_2 для его измерения. Таким образом, для каждого фотона, полученного от Алисы, Боб запишет бит 0 или 1 в зависимости от того, какой базис измерения был использован.

После завершения измерений Боб засекречивает результаты. Затем он связывается с Алисой по классическому каналу и сообщает, какие базисы, B_1 или B_2, использовал для измерения каждого фотона, но не раскрывает их поляризацию.

Шаг 3. Определение общего ключа

Алиса отвечает на сообщение Боба, указывая правильные базисы для измерения поляризации фотона. Алиса и Боб сохраняют те биты, для которых они выбрали одинаковые базисы, и отбрасывают все остальные. Поскольку существует только два возможных варианта, они выберут один и тот же базис примерно для половины передаваемых битов. Их можно использовать для формирования общего ключа.

Числовой пример. Предположим, Алиса хочет передать такую последовательность битов:

$$[0, 1, 1, 1, 0, 0, 1, 1].$$

Биты имеют следующие поляризации.

Базис	**0**	**1**
B_1	$\|\uparrow\rangle$	$\|\rightarrow\rangle$
B_2	$\|\nwarrow\rangle$	$\|\nearrow\rangle$

Алиса, а затем и Боб случайным образом выбирают базисы для измерения.

Базисы Алисы	B_1	B_2	B_1	B_1	B_2	B_2	B_1	B_2
Последовательность Алисы	0	1	1	1	0	0	1	1
Поляризация Алисы	$\|\uparrow\rangle$	$\|\nearrow\rangle$	$\|\rightarrow\rangle$	$\|\rightarrow\rangle$	$\|\nwarrow\rangle$	$\|\nwarrow\rangle$	$\|\rightarrow\rangle$	$\|\nearrow\rangle$

Базисы Боба	B_2	B_2	B_2	B_1	B_2	B_1	B_1	B_2
Поляризация Боба	$\|\nwarrow\rangle$	$\|\nearrow\rangle$	$\|\nwarrow\rangle$	$\|\rightarrow\rangle$	$\|\nwarrow\rangle$	$\|\uparrow\rangle$	$\|\rightarrow\rangle$	$\|\nearrow\rangle$
Последовательность Боба	0	1	0	1	0	0	1	1

Сравнив выбор Боба с выбором Алисы, мы видим, что вторая, четвертая, пятая, седьмая и восьмая поляризации совпадают. Используя биты, соответствующие этим поляризациям, можно сформировать общий ключ.

Правильные поляризации	$\|\nearrow\rangle$	$\|\rightarrow\rangle$	$\|\nwarrow\rangle$	$\|\rightarrow\rangle$	$\|\nearrow\rangle$
Соответствующие биты	1	1	0	1	1
[Общий ключ]	1	1	0	1	1

На рис. 9.12 наглядно представлено, как выглядит эта схема.

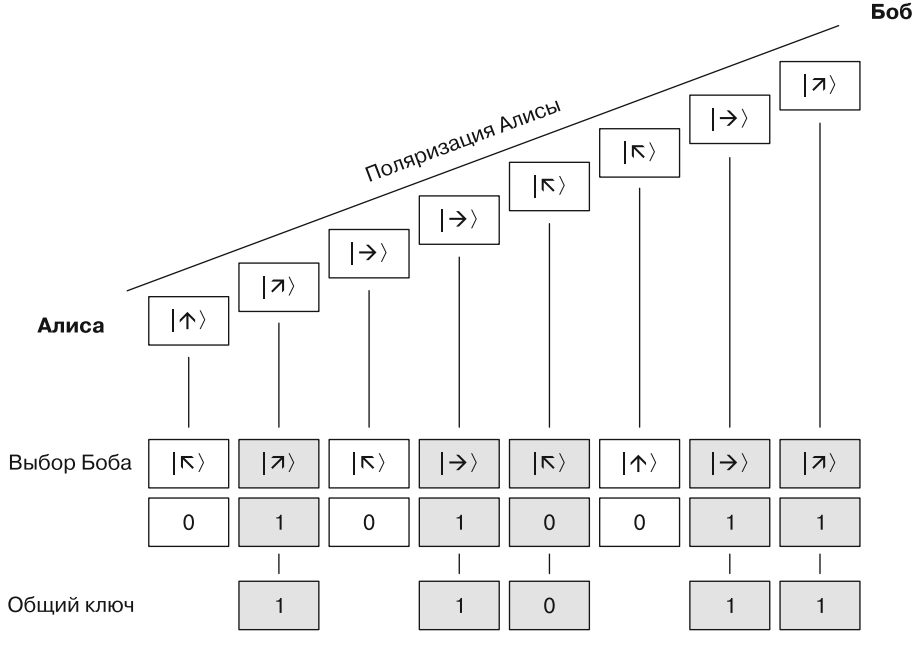


Рис. 9.12. Представление QKD

На данном этапе нам нужно только подтвердить идентичность двух ключей (Алисы и Боба). В рамках проверки четности Алисе и Бобу нужно проверять не всю последовательность, а только ее часть. Очевидно, что если просматривать всю последовательность четности по классическому каналу, то Ева может вмешаться в процесс и украсть ключ. Поэтому лучшим решением будет проверка только части последовательности битов.

Например, если ключ состоит из 1100 цифр, то проверяются только 10 % битов. Таким образом, после проверки четности, выполняемой по определенному алгоритму, отбрасываются 100 обработанных бит, а остальные сохраняются. Этот метод позволяет убедиться, что в процессе передачи и обнаружения битов

не возникло ошибок, а также определить, пыталась ли Ева наблюдать за передачей в квантовом канале. Это обусловлено тем, что в Q-механике невозможно наблюдать фотон, не заставив его принять какое-либо направление поляризации (по принципу неопределенности). Однако если Ева измерит фотоны до Боба и позволит им продолжить движение до измерения Боба, она потерпит неудачу примерно в половине случаев. Когда тот же фотон, измеренный Евой, попадет к Бобу, вероятность того, что при измерении он получит верное значение, составит всего 25 %. Таким образом, любая попытка перехвата увеличивает вероятность ошибки. Вот почему после третьего шага Алиса и Боб, выполняющие проверку четности ошибок, также могут обнаружить Еву.

Теперь вопрос тысячелетия: *может ли Ева восстановить ключ?*

Другими словами, можно ли взломать этот алгоритм?

Попробуем найти ответ на этот вопрос в следующем разделе, где рассмотрим возможные атаки.

Атаки и технические вопросы

При описании этого алгоритма предполагалось, что Ева обладает той же вычислительной мощностью, что и Боб с Алисой, и может следить за их взаимодействием. Если Ева располагает квантовым компьютером и всеми инструментами для определения поляризации фотонов, сможет ли она взломать алгоритм?

Давайте посмотрим, что Ева, с ее вычислительной мощностью и возможностью следить за классическим и квантовым каналами, может сделать.

Она находится в той же позиции, что и Боб (получает фотоны по квантовому каналу). Предполагается, что если кто-то сможет перехватить всю последовательность битов до выбора Боба, то он сможет собрать биты ключа. Это могло бы быть правдой, если бы Ева сделала 100 % правильных измерений, но это невозможно для ключа длиной 1000 бит или более.

При этом, как мы сейчас увидим, атака будет обнаружена Алисой и Бобом.

Посмотрим, что может произойти. Предположим, Алиса передает фотон с поляризацией $|\rightarrow\rangle$, а Боб использует базис B_1, который совпадает с базисом Алисы. Если Ева измерит фотон с тем же базисом B_1, измерение не изменит поляризацию и Боб правильно измерит состояние фотона. Однако если Ева использует B_2, то она измерит два состояния, $|\nwarrow\rangle$ и $|\nearrow\rangle$, с равной вероятностью. Фотон, который попадет к Бобу, только в половине случае будет правильно измерен как $|\rightarrow\rangle$ и будет иметь верное значение. Учитывая два возможных варианта, которые Ева может выбрать для определения правильного или неправильного измерения, Боб измерит верное значение только в 25 % случаев.

Другими словами, каждая попытка Евы выполнить измерения увеличивает вероятность возникновения ошибки во взаимодействии Алисы и Боба. Поэтому, когда Алиса и Боб пойдут проверять биты четности, они обнаружат ошибку и поймут, что предпринимается попытка атаки. В этом случае они повторят процедуру (надеюсь), избежав повторной атаки, или изменят канал связи — например, переключат цветовой код оптического волокна с желтого на зеленый[1].

Я поддерживаю тезис о том, что эта система теоретически неуязвима, но также признаю, что это лишь частичное решение, главным образом потому, что это не квантовая передача сообщения, а только квантовое распределение ключа. Причина, по которой я утверждаю, что она теоретически неуязвима, заключается в том, что Боб и Алиса могут обнаружить Еву, опираясь на принцип неопределенности Гейзенберга. Однако атака посредника, даже если Ева не восстановит ключ (она может попытаться выдать себя за Боба), способна заблокировать обмен ключами между Алисой и Бобом, заставив их повторять операцию бесконечное число раз или, как мы видели, переключиться на другой квантовый канал связи.

В специальном документе Агентства национальной безопасности США можно прочитать о некоторых слабых местах QKD и причинах, по которым его не рекомендуется внедрять: https://www.nsa.gov/Cybersecurity/Quantum-Key-Distribution-QKD-and-Quantum-Cryptography-QC/.

Необходимо понять, связаны ли опасения АНБ с возможностью реализации такой системы или с тем, что злоумышленники могут воспользоваться преимуществами этой сильной технологии.

В документе АНБ выделяются следующие проблемы практической реализации QKD.

- *QKD — это лишь частичное решение*. Предполагается, что протокол не обеспечивает аутентификацию между двумя участниками. Таким образом, для идентификации Алисы и Боба, то есть цифровой подписи, требуется алгоритм асимметричного шифрования, что сводит на нет преимущества квантовой защиты. Эту проблему можно было бы решить с помощью квантового алгоритма аутентификации, реализующего квантовую цифровую подпись, если это возможно. К сожалению, на современном уровне развития техники выполнение цифровой подписи с помощью квантового алгоритма и обеспечение невозможности отречься от сообщения дают лишь вероятностный результат, далекий от детерминированной цифровой подписи классических алгоритмов. Другой вариант — использование MAC (функция

[1] Здесь имеется в виду, что оптоволоконные линии маркируются стандартными цветовыми кодами, соответствующими разным типам волокон. Поэтому смена «желтого на зеленый» означает переход на другой физический канал связи, а не буквальную смену цвета кабеля.

аутентификации для симметричных алгоритмов), но и она может в будущем оказаться *не* квантово устойчивой.

- *Затраты и специализированное оборудование*. Эта проблема связана с более высокими затратами и трудностями внедрения технологии для пользователей. Кроме того, систему сложно интегрировать с существующим сетевым оборудованием. Следует отметить, что документ АНБ, вероятно, был написан до массового распространения оптического волокна. Сейчас наши инфраструктуры все шире используют оптоволокно для передачи информации, а данный вид связи применяется и для передачи света или фотонов.
- *Безопасность и проверка*. Проблемы здесь связаны с аппаратным обеспечением для реализации квантовой системы. Специфическое и сложное оборудование, используемое для QKD, может иметь уязвимости. Однако этот момент также обсуждаем, поскольку качество современного оборудования значительно улучшилось.
- *Риск отказа в обслуживании*. Это, пожалуй, самая серьезная проблема, связанная с QKD. Как уже упоминалось, Ева может повторять атаку на квантовый канал, не давая Алисе и Бобу сформировать общий ключ. Мы изучаем новый метод преодоления этой проблемы с помощью разноцветных волокон, которые будут автоматически переключаться в случае повторного обнаружения аномалии в квантовом канале.

Чтобы оставаться в курсе событий, посетите наш замечательный лабораторный центр «Квантовая криптография»: www.quantumlab.science и www.quantumlab.educational.

Q-лаборатория работает с ноября 2024 года.

Как мы уже убедились, QKD нельзя назвать идеальным методом. Существует отдаленная вероятность того, что кто-то сможет взломать эту систему.

Важно! QKD представляет только одну часть алгоритма криптографической передачи данных. Другая часть — это алгоритм шифрования, используемый для передачи сообщения [*M*].

QKD — это фундаментальная часть развития Q-криптографии. Система должна гарантировать передачу сообщения [*M*] по квантовому каналу, а для этого нужен еще один квантовый алгоритм. Более того, остается нерешенным вопрос о квантовом протоколе аутентификации.

По сути, QKD можно сравнить с алгоритмом Диффи — Хеллмана, так же как RSA можно сравнить с *квантовой системой передачи сообщений*.

Есть разные варианты, когда речь идет о передаче сообщений в классической криптографии: мы можем использовать *шифр Вернама* (см. главу 1), который в сочетании со случайным ключом (QKD) теоретически безопасен, или другой алгоритм симметричного шифрования. Опять же мы не работаем с полностью квантовой системой шифрования — она зависит также от классических математических задач.

Возвращаясь к алгоритму QKD, мы должны выяснить, почему его теоретически невозможно взломать даже с помощью квантового компьютера.

Если злоумышленник расшифрует ключ, сгенерированный с помощью Q-криптографии, результат этого действия, вероятно, будет иметь разрушительные последствия для всей Q-механики. Это произошло бы из-за противоречия одной из аксиом Q-механики — мы говорим о принципе неопределенности. Если бы нечто подобное произошло, то вся структура Q-механики, вероятно, была бы вовлечена в процесс пересмотра, который потрясет основы теории и все ее физические и философские следствия.

Чтобы утвердить эту концепцию, я предлагаю рассмотреть такую мысль: предположим, в будущем кто-то обнаружит, что фотоны, проходящие между точкой передачи Алисы и точкой измерения Боба, не являются действительно случайными, а следуют заранее определенной схеме. Предположим, что эта схема — число π или последовательность Фибоначчи. В этом случае биты фактически случайны, но их случайность будет известна, так что Ева сможет предсказать, какую схему поляризации следует использовать.

Что произойдет, если мы узнаем, что последовательность цифр, например 1415, входит в число π?

Как показано далее, число π — это бесконечная последовательность цифр, стоящих за числом 3 после десятичной точки, начинающаяся с 1415: π = 3,1415926 53589793238462643383279502884197169399375105820974944592307816406286 20899862803482534211706798214808651328230664709384460955058223172535 94081284811174502841027019385211055596446229489549303819644288109 75665933446128475648233786783165271201 90...

Таким образом, мы можем восстановить все остальные цифры числа π после 1415: 92653589...

Несомненно, я согласен с докладом АНБ, в котором говорится, что QKD — это неполный алгоритм. В нем отсутствует вторая часть, шифрующая сообщение [*M*] в квантовом канале. При использовании BB84 этот этап сводится к классической криптографии, поскольку после обмена ключами (в квантовом

канале) необходимо зашифровать сообщение с помощью классического алгоритма шифрования. Здесь есть две проблемы.

1. Как уже было сказано, на этом этапе используется классический алгоритм, даже если он теоретически безопасный.
2. Используя классический алгоритм, мы не сможем обнаружить вмешательство Евы. Сделать это можно только с помощью квантового алгоритма передачи.

Поскольку я всегда был против гибридных схем, то в 2023 году разработал новый алгоритм шифрования *QTM23*, пригодный для квантовой системы передачи сообщений. Надеюсь, он станет стандартом для передачи данных, выполняемой в рамках Q-криптографии.

В Cryptolab мы сейчас находимся на стадии создания прототипа; 19 июня 2024 года мы передали сообщение HELLO WORLD! по квантовому каналу с помощью QTM23, используя то же самое оборудование, что и в экспериментах с BB84. Боб (возьмем имена, к которым привыкли) получил сообщение примерно через 6 минут после того, как его передала Алиса. В настоящее время мы применяем механические моторизованные детекторы лучей, которые посылают фотоны по одному. Технически это очень медленный метод передачи, но в будущем мы планируем увеличить скорость с помощью светодиодных детекторов лучей.

Из-за патентных вопросов я не могу раскрывать детали реализации этого алгоритма. Я надеюсь, что вскоре смогу обновить данный раздел более подробной информацией об этом инновационном решении.

Теперь, когда мы рассмотрели распределение ключей в Q-криптографии, нас ждет еще один вызов — исследование квантовых вычислений и возможностей, которые у них появятся с ростом мощности.

Квантовые вычисления

В этом разделе я расскажу о *квантовых вычислениях* и *квантовых компьютерах*. Разница между ними заключается в том, что квантовые вычисления — это вычислительная мощность квантовой системы, а квантовый компьютер — физическая реализация системы, состоящая из аппаратного и программного обеспечения. Впервые идея квантового компьютера была предложена Ричардом Фейнманом в 1982 году. Затем, в 1985-м, Дэвид Дойч сформулировал первую теоретическую модель такого компьютера. В последние десятилетия эта область претерпела значительную эволюцию: частные компании начали работать и экспериментировать с новыми моделями квантовых компьютеров. Некоторые из них, такие как *D-Waves* и *Righetti Computing*, собрали миллионы долларов на исследования и разработку этих машин (рис. 9.13) и соответствующего программного обеспечения.

Рис. 9.13. Гибридный квантовый компьютер D-Waves

Как вы уже знаете, обычные компьютеры работают с битами, а квантовые — с кубитами. Особенность кубита заключается в том, что он может одновременно находиться в состояниях 0 и 1. Уже одно это говорит о более широких возможностях вычислений по сравнению с обычным компьютером.

Обычный компьютер выполняет каждую операцию по очереди, пока не получит конечный результат. Квантовый же компьютер может выполнять несколько операций одновременно. Более того, многие кубиты могут быть связаны друг с другом в состоянии запутанности.

Это явление создает новое состояние суперпозиции над множеством комбинаций запутанных кубитов, экспоненциально увеличивая вычислительную мощность квантового компьютера по сравнению с обычным. Речь идет не только о распараллеливании вычислений, но и об экспоненциальном росте вычислительной способности при увеличении количества кубитов. Представьте вычислительную мощность квантовых компьютеров: n кубитов обеспечивают обработку информации в 2^n раза эффективнее обычных битов.

Рассмотрим этот тезис подробнее на примере. В состоянии суперпозиции или запутанности 8 кубитов могут одновременно представлять любое число от 0 до 255. Это связано с тем, что $2^8 = 256$, то есть существует 256 возможных конфигураций, которые в двоичной системе счисления выглядят как:

$$0 = (0)^2 \rightarrow 255 = (11111111)^2.$$

Преимущество квантовых компьютеров становится особенно очевидным при решении двух криптографически значимых задач: факторизации и поиска в больших массивах данных.

Эти две задачи считаются очень трудноразрешимыми для обычного компьютера. Мы видели, как алгоритм RSA и другие криптографические алгоритмы используют факторизацию для генерации односторонней функции, способной определить защищенную криптограмму. Однако криптографическое

сообщество постоянно задается вопросом: *как долго классическая криптография сможет сопротивляться, когда квантовые компьютеры обретут достаточно кубитов для выполнения немыслимых сегодня операций?*

Мы попытаемся ответить на этот вопрос, изучив потенциал и возможности квантового компьютера, адаптированного к нашим условиям.

Как я уже упоминал, квантовый компьютер может одновременно обрабатывать все базисы состояний линейной комбинации. В этом смысле его можно рассматривать как мощную параллельную машину, способную одновременно вычислять огромные состояния комбинаций.

Например, чтобы понять, на что способен квантовый компьютер, возьмем три частицы: первую с ориентацией 1 и две с ориентацией 0 относительно некоторого базиса. Мы можем представить это как $|100\rangle$.

Квантовый компьютер может принять $|100\rangle$ в качестве входного значения и выдать некоторый результат, но в то же время он может принять на вход *нормализованную комбинацию* из трех частиц с базисом состояний:

$$1/\sqrt{3}\left(|100\rangle + |100\rangle + |100\rangle\right).$$

Таким образом, он выдает результат только за одну операцию, поскольку вычисляет лишь базис состояний.

Квантовый компьютер не знает, находится фотон в одном базисе состояний или в нескольких комбинациях состояний, пока не произойдет измерение (принцип неопределенности).

Эта способность к одновременному развитию комбинаций линейных состояний и определяет превосходство квантовых компьютеров.

Хотя обычный компьютер (даже самый мощный суперкомпьютер на Земле) требует в качестве входных данных строку битов и возвращает строку битов (даже если распараллеливает миллиарды операций), он не сможет выполнить функцию со следующими входными значениями:

$$1/C\sum|x\rangle.$$

Здесь C — коэффициент нормализации всех возможных состояний входных данных. На выходе функция возвращает:

$$1/C\sum\left|x, f\left(x\right)\right\rangle,$$

где $|x, f(x)\rangle$ — это последовательность битов большей длины, чем $|x|$, которая представляет и $|x|$, и $f(x)$.

Для понимания этой концепции можно представить, что у вас есть сумма битов (C) и в ней скрыты все возможные состояния (m) кубитов в суперпозиции.

Таким образом, мы можем сказать, что (C) представляет собой число всех состояний, в которых одновременно находятся кубиты.

Представьте, насколько это может быть фантастическим! Однако остается проблема — измерение. Мы видели, что при измерении квантового вычисления кубиты *«замораживаются»* в одном состоянии, которое определяется случайностью. Поэтому можно заставить систему *находиться* в каком-либо состоянии следующим образом:

$$|x_0, f(x_0)\rangle.$$

Это происходит для некоторого случайно выбранного значения x_0, и в результате все остальные состояния будут уничтожены.

В этом случае возможность программирования квантового компьютера позволяет создать квантовое вычисление, которое обеспечивает очень высокую вероятность получения желаемого результата.

Чтобы понять процесс реализации квантового алгоритма, рассмотрим алгоритм, который теоретически может уничтожить значительную часть классической криптографии, — алгоритм Шора.

Алгоритм Шора

В одном интервью Дэвиду Дойчу, считающемуся одним из отцов квантовых вычислений, задали вопрос о философии суперпозиции: *«В каком смысле квантовое сообщество рассматривает гипотезу о существовании мультивселенной?»*

Ответ Дэвида относится к Q-криптографии, в частности, он попытался продемонстрировать существование мультивселенной с помощью задачи факторизации. Я попробую перефразировать ответ Дэвида:

> *«Представьте, что вам нужно факторизовать целое число, состоящее из 10 000 цифр, которое представляет собой произведение двух больших простых чисел. Ни один обычный компьютер не сможет выразить его как произведение его простых множителей. Даже если мы возьмем всю материю, содержащуюся во Вселенной, и превратим ее в суперкомпьютер, который будет работать в течение всего времени существования Вселенной, он не сможет даже прикоснуться к решению задачи факторизации. У квантового компьютера это займет всего несколько минут или секунд. Как такое возможно?*
>
> *Каждый, кто не является солипсистом, должен признать, что ответ связан с каким-то физическим процессом. Мы знаем, что во Вселенной недостаточно энергии, чтобы вычислить и получить ответ, поэтому, помимо того что мы можем видеть непосредственно, должно происходить что-то еще. На данный момент мы уже приняли структуру мультивселенной».*

Это отступление о *квантовой вычислительной мощности подчеркивает потенциал квантовых компьютеров*. Давайте погрузимся в математическую и логическую суть алгоритма Шора, поскольку этот алгоритм — истинное представление того, что происходит в квантовом компьютере.

Гипотеза и тезис

Гипотеза (в контексте алгоритма Шора) формулируется следующим образом: предположим, мы нашли два нетривиальных значения (a) и (r), удовлетворяющих условию:

$$a^r \equiv 1 \pmod{n}.$$

Это создает возможность факторизации числа (n).

Начнем со случайного выбора числа (a) и рассмотрим такую последовательность чисел:

$$1, a, a^2, a^3 \ldots a^n \equiv 1 \pmod{n}.$$

Тезис состоит в том, что мы можем найти период для этой последовательности, который обозначим (r). Этот период будет повторяться (r) раз при каждом возведении (a) в степень (r), тем самым решая уравнение для (r), которое всегда дает в результате (1):

$$a^r \equiv 1 \pmod{n}.$$

Поэтому нам нужно запрограммировать свой квантовый компьютер на поиск (r), чтобы он мог *с высокой вероятностью* факторизовать (n).

Шаг 1. Инициализация кубитов

Мы выбираем число (m) так, чтобы выполнялось условие:

$$n^2 \leqslant 2^m < 2^n.$$

Предположим, что мы выбрали $m = 9$, то есть у нас есть 9 бит, инициализированных в состоянии (0). Математически это выглядит так:

$$m = 9 : |000000000\rangle.$$

Меняя оси битов, мы можем преобразовать первый бит в линейную комбинацию $|0\rangle$ и $|1\rangle$, что дает следующий результат:

$$1/\sqrt{2}\left(|000000000\rangle + |100000000\rangle\right).$$

Выполнив аналогичное преобразование для каждого бита из (m) внутри бра и кет до *m бит для получения квантового состояния*, мы получим последовательность, подобную этой:

$$1/\sqrt{2^m}\left(|000000000\rangle + |000000001\rangle + |000000010\rangle + \ldots + |111111111\rangle\right).$$

Поскольку $m = 9$, то $2^m = 512$: у нас будет 512 комбинаций умножения битов, заданных первым значением $|000000000\rangle = 0$ и последним значением $|2^{m-1}\rangle = |111111111\rangle = 511$ (в десятичной системе счисления).

В такой сумме все возможные состояния m кубитов накладываются друг на друга в суперпозиции. Теперь, чтобы упростить обозначения, перепишем m кубитов:

$$1/\sqrt{2^m}\left(|0\rangle + |1\rangle + |2\rangle + \ldots + |2^{m-1}\rangle\right).$$

Эти функции аналогичны предыдущим, но представлены количественными числами.

Шаг 2. Выбор случайного числа (a)

Теперь нам нужно выбрать случайное число (a) так, чтобы выполнялось условие:

$$1 < a < n.$$

Квантовые вычисления позволяют вычислить функцию:

$$f(x) \equiv a^x \pmod{n} —$$

и возвращают следующий результат:

$$1/\sqrt{2^m}\left(|0, a^0 \pmod{n}\rangle + |1, a^1 \pmod{n}\rangle + |2, a^2 \pmod{n}\rangle + \ldots + |2^{m-1}, a^{2m-1} \pmod{n}\rangle\right).$$

Эта функция может показаться довольно сложной. Поэтому, чтобы упростить ее, забудем о модуле n, который повторяется в каждой операции:

$$1/\sqrt{2^m}\left(|0, a^0\rangle + |1, a^1\rangle + |2, a^2\rangle + \ldots + |2^{m-1}, a^{2m-1}\rangle\right).$$

Не забываем, что все операции происходят по (mod n), но сосредоточимся на смысле функции: она говорит, что мы должны возвести (a) во все (2^{m-1}) состояний m кубитов.

Однако на данном этапе это квантовое состояние не дает никакой дополнительной информации по сравнению с обычным компьютером.

Теперь рассмотрим пример, который поможет лучше понять наши действия.

Пусть $n = 21$ (очень простое число для факторизации). Здесь, впрочем, важен не размер числа, а сам метод.

Здесь мы случайным образом выбираем $a = 11$.

Итак, вычисляем значения $11^x \pmod{21}$, которые входят в функцию:

$$1/\sqrt{512}\left(|0, 1\rangle + |1, 11\rangle + |2, 16\rangle + |3, 8\rangle + |4, 4\rangle + |5, 2\rangle + |6, 1\rangle + |7, 11\rangle + \right.$$
$$+ |8, 16\rangle + |9, 8\rangle + |10, 4\rangle + |11, 2\rangle + |12, 1\rangle + |13, 11\rangle + |14, 16\rangle + |15, 8\rangle +$$
$$\left. + |16, 4\rangle + |17, 2\rangle + \ldots + |508, 4\rangle + |509, 2\rangle + |510, 1\rangle + |511, 11\rangle\right).$$

Как видите, это разложение всей последовательности, где $1/\sqrt{512} = 1/C$, а C — количество состояний $m = 2^9$.

Каждый кубит состоит из порядкового номера ($a^x \pmod{21}$). Возьмем следующий пример:

$$|1, 11\rangle \equiv 11^1 \pmod{21} = 11.$$

Шаг 3. Квантовое измерение

Если мы измерим только вторую часть последовательности — a^x, то функция деформируется в неконтролируемое состояние. Однако это измерение является сутью системы. Фактически из-за него вся последовательность деформируется в некоторое случайное базисное состояние (x_0) так, что применимо следующее:

$$\left|x_0, a^{x_0}\right\rangle.$$

Теперь вам должно быть понятнее, что мы можем заставить систему появиться в состоянии, показанном далее:

$$\left|x_0, f\left(x_0\right)\right\rangle.$$

Таким образом, после измерения состояние системы фиксируется (другими словами, появляется число) только в одном базисе (x_0), предположительно имеющем значение $x_0 = 2$.

Отбросив все остальные значения a^x, которые мы вычислили ранее, и отобрав только те, которые имеют значение, равное 2, мы извлекаем выделенные значения:

$$1/\sqrt{512}\left(|0, 1\rangle + |1, 11\rangle + |2, 16\rangle + |3, 8\rangle + |4, 4\rangle + |\mathbf{5}, \mathbf{2}\rangle + |6, 1\rangle + |7, 11\rangle + \right.$$
$$+|8, 16\rangle + |9, 8\rangle + |10, 4\rangle + |\mathbf{11}, \mathbf{2}\rangle + |12, 1\rangle + |13, 11\rangle + |14, 16\rangle + |15, 8\rangle +$$
$$\left. +|16, 4\rangle + |\mathbf{17}, \mathbf{2}\rangle + \ldots + |508, 4\rangle + |\mathbf{509}, \mathbf{2}\rangle + |510, 1\rangle + |511, 11\rangle\right).$$

Для упрощения обозначений отбросим вторую часть кубита (2) — она лишняя:

$$1/\sqrt{85}\left(|5\rangle + |11\rangle + |17\rangle + \ldots + |509\rangle\right).$$

Здесь 85 означает число выбранных m кубитов.

Теперь если мы проведем измерение, то найдем значение x:

$$11^x \equiv 2 \pmod{21}.$$

К сожалению, эта информация нам не пригодится. Но не нужно отчаиваться.

Шаг 4. Поиск подходящего варианта (r)

Напомню, что мы запрограммировали свой квантовый компьютер с определенной целью — найти выходное значение в системе, факторизующее (n). Чтобы достичь ее, мы должны найти такое значение (r), чтобы выполнялось условие:

$$a^r \equiv 1 \pmod{n}.$$

Обычно, когда мы имеем дело с функциями типа $a^x \pmod{n}$, мы знаем, что значения (x) имеют период (r), где $a^r = 1 \pmod{n}$.

Таким образом, вспомним предыдущую функцию, в которой появляются выбранные элементы:

$$|5\rangle + |11\rangle + |17\rangle + ... + |509\rangle.$$

В этом примере $r = 6$ является периодом функции.

Используя результат $r = 6$, мы можем проверить:

$$11^r \equiv 1 \pmod{21};$$
$$11^6 \equiv 2 \pmod{21}.$$

Период функции не всегда легко вычислить. Существует математический способ определения периода функции, использующий *квантовое преобразование Фурье* (quantum Fourier transform, QFT). Разберем его в следующем разделе.

Если вы хотите продолжить знакомство с алгоритмом Шора, можете пропустить следующий раздел и перейти непосредственно к последнему шагу алгоритма, вернувшись к QFT при втором прочтении.

QFT

Давайте узнаем немного больше о *преобразовании Фурье* (Fourier transform, FT) и квантовом преобразовании Фурье.

FT — это математическая функция, которую можно представить как разложение музыкального аккорда на амплитуды и частоты. Оно преобразует исходную функцию в новую, отражающую распределение частот в исходных данных. FT зависит от пространственной или временной частоты и называется *временной областью*.

Например, пусть функция f — это последовательность чисел с *периодической* структурой, представленной частотой повторения последовательности. Эти периоды можно выразить с помощью кривых, называемых *гармониками*.

На рис. 9.14 показан пример FT, где частота изображена в виде гармоник.

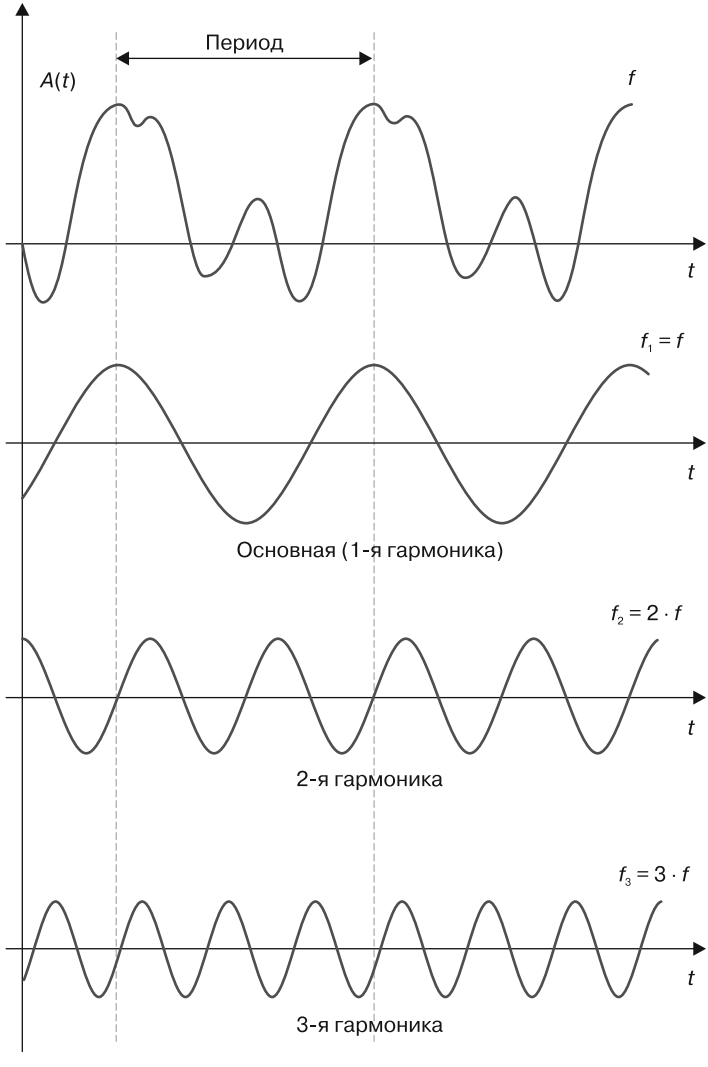


Рис. 9.14. Преобразование Фурье

Четыре графика демонстрируют функцию f и три различные гармоники. На каждом из них пунктирными линиями отмечен период. Кривые внутри каждого периода отражают последовательность чисел. Форма гармоник зависит от частоты повторения этой последовательности.

Числовой пример. Возьмем арифметическую последовательность чисел:

$$1, 3, 7, 2, 1, 3, 7, 2.$$

Видно, что блок из четырех чисел, 1, 3, 7, 2, повторяется два раза. Другими словами, мы можем разбить эту последовательность на две части:

$$1, 3, 7, 2 \mid 1, 3, 7, 2.$$

В данном случае символ | означает разделение последовательности на части.

Здесь эта последовательность имеет длину 8 и период 4. Длина, деленная на период, дает частоту *(f)*:

$$f = 8 / 4 = 2.$$

В данном случае $f = 2$ — это количество повторений последовательности.

Теперь мы можем сформулировать общее правило FT. Предположим, у нас есть последовательность, длина которой 2^m для целого числа m:

$$\left[a_0, a_1 \ldots a_{2^m - 1} \right].$$

Мы можем определить FT по формуле:

$$\mathrm{FT}(X) = 1/\sqrt{2^m} \sum_{c=0}^{2^m - 1} e^{\left(2\pi i c x / 2^m\right)} c.$$

Вторую часть формулы можно определить через:

$$j = e^{\left(2\pi i c x / 2^m\right)}.$$

Параметр x находится в диапазоне $0 \leqslant x < 2^m$.

С помощью формулы вычисления FT можно найти частоту повторения последовательности, используя в качестве входных данных определенные параметры. Если, к примеру, $cx = 1$ и $m = 3$, то получится следующий результат:

$$j = e^{(2\pi i / 8)}.$$

Подставляя различные значения x в уравнение с j, мы раскрываем формулу для вычисления FT(1):

$$\sqrt{8}\,\mathrm{FT}(1) = 1 + 3j + 7j^2 + 2j^3 + j^4 + 3j^5 + 7j^6 + 2j^7 = 0.$$

Поскольку все члены последовательности компенсируются, получаем следующее:

- $\mathrm{FT}(X_0)$ при $x = 0 \rightarrow e^0 = 1$, что является результатом $\mathrm{FT}(X_0) = 1$;
- $\mathrm{FT}(X_4)$ при $x = 4 \rightarrow e\pi i = -1$, что является результатом $\mathrm{FT}(X_4) = -1$.

При $j^4 = -1$ все члены компенсируются и мы получим $\mathrm{FT}(1) = 0$.

Для FT(0) числитель, являющийся результатом сложения, соответствует сумме всех членов последовательности:

$$1 + 3 + 7 + ... + 2 = 26.$$

Если продолжить вычисления, мы получим следующие результаты:

$$\mathrm{FT}(0) = 26/\sqrt{8};\ \mathrm{FT}(2) = (-12 + 2i)/\sqrt{8};$$
$$\mathrm{FT}(4) = 6/\sqrt{8};\ \mathrm{FT}(6) = (-12 - 2i)/\sqrt{8}.$$

Для всех остальных элементов FT(1) = FT(3) = FT(5) = FT(7) = 0.

Другими словами, FT подтверждает то, что мы вывели в начале этого примера: в последовательности 1, 3, 7, 2, 1, 3, 7, 2 пики функции являются ненулевым значением FT, кратным 2 (частоте).

Как мы вскоре увидим, этот результат критически важен в алгоритме Шора для определения периодов функции.

Однако мы хотим запрограммировать свой алгоритм в квантовом компьютере, поэтому нам нужна адаптированная для этих целей версия FT.

В таком случае значения частоты, которые будут использоваться для нахождения периода (r) в алгоритме Шора, вычисляются посредством QFT.

QFT определяется для базисного состояния $|X\rangle$ при ($0 \leqslant x < 2^m$):

$$\mathrm{QFT}(|X\rangle) = 1/\sqrt{2^m} \sum_{c=0}^{2^m-1} e^{(2\pi icx/2^m)} |c\rangle.$$

Представленные здесь формулы для вычисления FT и QFT очень похожи между собой, с той лишь разницей, что QFT используется в квантовой среде. Таким образом, параметр c из FT представлен в квантовом состоянии $|c\rangle$, и то же самое справедливо для $|X\rangle$.

Следовательно, в нашем примере частота вычисляется как:

$$\mathrm{QFT}\left(1/\sqrt{85}\left(|5\rangle + |11\rangle + |17\rangle + \ldots + |509\rangle\right)\right).$$

В результате получается следующая сумма:

$$1/\sqrt{85} \sum g(c)|c\rangle.$$

Для c это значение находится в диапазоне от 0 до 511:

$$g(c) = 1/\sqrt{512} \sum_{0 \leq x \leq 512} e^{(2\pi icx/512)}.$$

Это FT такой последовательности:

0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1... 0, 0, 0, 0, 0, 1, 0, 0.

Визуализируем частоту повторения последовательности с помощью символа разрыва:

$$0, 0, 0, 0, 0, 1, | 0, 0, 0, 0, 0, 0, 1, | \ldots 0, 0, 0, 0, 0, 0, 1, |, 0, 0.$$

Как видите, частота повторений составляет 512 / 6 ~ 85.

В этой синусоиде, задаваемой волнами, у которых есть высокие и низкие точки, имеются следующие пики (высшие точки):

$$c = 0, 85, 171, 256, 314, 427.$$

Ожидается, что 427 кратно искомой частоте:

$$427 \sim jf_0.$$

Таким образом, $r = 6$ — это период, вычисляемый как

$$427 / 512 \sim 5 / 6.$$

Важно! Я не стал показывать весь математический процесс, в результате которого получается $r = 6$, так как моей целью было продемонстрировать возможность применения QFT к этому алгоритму. Более подробную информацию о QFT можно найти на сессии проекта IBM Quantum Lab: https://qiskit.org/textbook/ch-algorithms/quantum-fourier-transform.html.

Теперь, когда вы знаете, как работает QFT и как использовать его в алгоритме Шора, выполним факторизацию с помощью квантового алгоритма.

Шаг 5. Факторизация (n)

На последнем шаге алгоритма Шора с помощью наибольшего общего делителя (НОД) вычисляется, возвращает ли (r) два множителя (n).

Как вы, вероятно, помните из средней школы, НОД — это наибольший положительный общий делитель двух чисел. Его можно использовать также для сокращения дроби до ее наименьших возможных делителей.

В программе Mathematica есть очень полезная функция сокращения, которая используется при делении на два целых числа.

В нашем случае, если $r = 6$ и $n = 21$, мы просто применим деление к этим двум числам:

- In [01]: = НОД [21, 6];
- Out [01]: = 3.

Здесь я попросил систему Mathematica рассчитать НОД [21, 6], то есть НОД для 21 и 6, и получил число 3.

Сократив дробь (разделив 21 и 6 на 3), мы получим 7 в качестве второго выхода:

- In [02]: = 21 / 6;
- Out [02]: = 7 / 2.

Следовательно:

$$21 = 7 \cdot 3.$$

У нас есть решение!

Заметки об алгоритме Шора

Питер Шор представил этот алгоритм в эпоху, когда квантовые компьютеры существовали лишь теоретически. Здесь я продемонстрировал выполнение факторизации на квантовом компьютере, однако на текущий момент не существует физически реализованных систем, способных выполнить этот алгоритм на практике.

Более того, алгоритм Шора не является детерминированным. Он дает вероятностную оценку для нахождения множителей, даже если после нескольких шагов (если первый найденный вариант отбрасывается) можно будет найти хороший вариант для факторизации (n).

В то же время реализация масштабируемого квантового компьютера, способного выполнять алгоритм Шора, ознаменует конец эпохи классической криптографии.

Пришло время ответить на первоначальный, но переформулированный вопрос о квантовых компьютерах: каковы будут последствия для криптографии, когда квантовые компьютеры достигнут достаточного числа кубитов для выполнения алгоритма Шора?

Обсудим эту тему в заключительном разделе главы, посвященном постквантовой криптографии (пост-Q-криптографии).

Пост-Q-криптография

Необходимо отметить, что пост-Q-криптография не имеет ничего общего с Q-криптографией, за исключением одной цели — устойчивости к квантовым атакам (возможно, это несбыточная мечта). Когда мы говорим о пост-Q-криптографии, то имеем в виду устойчивость классических алгоритмов к квантовым вычислениям.

Когда квантовые вычисления достигнут высокой мощности, выраженной в достаточном количестве кубитов, и будут реализованы на соответствующем оборудовании, многие алгоритмы, которые обсуждались в этой книге, будут взломаны. Однако сегодня квантовый компьютер занимает целую комнату и работает благодаря многокомпонентному оборудованию, большая часть которого требует низких температур для поддержания кубитов в состоянии запутанности. Как только размеры оборудования уменьшатся и вычислительная мощность возрастет, взлом большинства алгоритмов (RSA и Диффи — Хеллмана, протоколов Шнорра и большинства протоколов доказательства с нулевым разглашением — даже тех, которые используют эллиптические кривые, так как большинство реализаций уязвимы), вероятно, займет от 1 часа до нескольких дней.

Согласно документу Европейской комиссии по кибербезопасности, выпущенному в феврале 2021 года, некоторые категории алгоритмов могут сдержать ударную волну квантовых вычислений.

В этом исследовании комиссия так прокомментировала появление квантовых вычислений:

> *«Поэтому важно заблаговременно подготовить замену. Усугубляет ситуацию то, что любое зашифрованное сообщение, перехваченное сегодня, может быть расшифровано злоумышленником, имеющим доступ к мощному квантовому компьютеру, через 5, 10 или 20 лет. Другими словами, данные уязвимы перед так называемым ретроспективным дешифрованием».*

Далее перечислены категории алгоритмов, которые, как считается, устойчивы к квантовым вычислениям. Многие из них ранее были оценены NIST.

- *AES256*. Считается хорошим постквантовым вариантом (вы видели этот алгоритм в главе 3).
- *Хеш-функции*. Мы рассматривали их в главе 4.
- *Кодовая криптография*. В ней используется техника кодов с коррекцией ошибок, основанная на предложении *Макэлиса*.
- *Криптография на изогениях*. Этот вид криптографии основан на использовании отображений (изогений) между эллиптическими кривыми над конечными полями.
- *Криптография на решетках*. Представитель этого вида алгоритмов NTRU — вероятно, один из лучших алгоритмов, устойчивых к квантовым атакам.

Исходя из оценок, полученных в результате математического и криптоаналитического анализа, все эти категории алгоритмов устойчивы к квантовым вычислениям. Однако в сообществе распространяется скептицизм, большая часть которого часто нивелируется заявлениями о минимизации проблем.

Резюме

В этой главе вы подробно изучили Q-криптографию. Познакомившись с основными принципами Q-механики и фундаментальными основами этой научной области, глубоко погрузились в квантовое преобразование Фурье.

Мы исследовали потенциал квантовых вычислений и алгоритм Шора, способный подорвать основы классической криптографии при реализации квантового компьютера с достаточным количеством кубитов для выполнения факторизации.

Вы изучили, что такое Q-криптография, ее методы реализации и квантовые вычисления. Познакомились также с очень разрушительным алгоритмом, который можно реализовать в квантовой машине, — алгоритмом Шора.

Наконец, вы узнали, какие алгоритмы устойчивы к атакам квантового компьютера и почему.

Темы и концепции этой главы очень важны, потому что от них зависит будущее криптографии.

Теперь, рассмотрев основы Q-криптографии и основные современные криптографические алгоритмы, перейдем к главе 10, где обсудим еще один важный квантовый алгоритм — алгоритм поиска Гровера. Мы обратимся к основам квантового программирования.

10 Алгоритмы квантового поиска и квантовые вычисления

Сейчас мы рассмотрим очень интересный квантовый алгоритм, который демонстрирует, насколько мощным может быть *квантовый поиск* в неструктурированном наборе данных.

Алгоритм Гровера имеет множество потенциальных сфер применения, начиная с криптографии и заканчивая искусственным интеллектом. В криптографии будущего он, вероятно, сможет взломать симметричные ключи AES или SHA с помощью крупномасштабного квантового компьютера. Его основной минус в том, что он обеспечивает только квадратичное, а не экспоненциальное ускорение. Поэтому алгоритм Гровера будет актуален только в начале эпохи квантовых вычислений. Я уверен, что в будущем появится более эффективный алгоритм, который увеличит разрушительную силу квантового поиска и уничтожит классическую криптографию.

Обзор алгоритма Гровера

Влияние квантового поиска на классическую криптографию можно представить в контексте нахождения правильного ключа для расшифровки шифротекста, зашифрованного алгоритмом симметричного шифрования с огромным количеством ключей. Квантовый поиск способен найти нужный ключ не за годы или дни, а за минуты или секунды!

Алгоритм Гровера может найти 128-битный симметричный ключ посредством перебора примерно за 264 итерации, 256-битный ключ — за 2128 итераций. Эти итерации требуют значительного количества кубитов, что в настоящее время возможно только в теории, но не на практике.

Алгоритм квантового поиска, представленный Ловом Гровером в 1996 году, и сам по себе важен для криптографии, но в первую очередь это прекрасный пример для введения в мир программирования квантовых алгоритмов.

Поиск в неструктурированном наборе данных похож на поиск правильного ключа расшифрования в наборе случайных ключей.

Однако такой неструктурированный поиск сопряжен с различными проблемами.

- *Оптимизация*. Поиск наилучшего маршрута между точками А и Б с посещением множества других точек (В, Г, Д, Е...). Например, именно такую задачу приходится решать службам доставок UPS и FedEx при оптимизации городских маршрутов.
- *Идентификация*. Алгоритмы классификации машинного обучения. Вкратце классификация — это форма *распознавания образов*. Например, алгоритм машинного обучения может распознавать и классифицировать электронные письма как спам или не спам.
- *Односторонние функции*. Вы встречали примеры этой проблемы на протяжении всей книги. К ним относятся хеш-функции, задачи дискретного логарифма и факторизации, решение которых трудно найти, но легко проверить. Например, цифровая подпись обеспечивает простоту проверки личности, однако сложная математическая задача в ее основе препятствует подделке. Хеш-функции, как и все остальные криптографические алгоритмы, состоят из ключей, поэтому, если есть возможность поиска ключей в базе данных, их также можно перебрать. Другим примером может быть нахождение дискретного логарифма внутри функции алгоритма Диффи —Хеллмана или факторизация больших простых чисел (алгоритм RSA из главы 3). Однако, на мой взгляд, с этими двумя категориями задач лучше справляется алгоритм Шора (см. главу 9). К этой категории относятся также решения судоку.

В целом алгоритм Гровера подходит для решения всех задач, связанных со случайным поиском и методом перебора. Однако для полного понимания алгоритма нам необходимо глубоко погрузиться в квантовые вычисления. Надеюсь, квантовое программирование вас увлечет.

В оригинальной статье Гровера, посвященной алгоритму квантового поиска, следующее вступление:

> *«Представьте себе телефонный справочник, содержащий N имен, расположенных в совершенно случайном порядке. Чтобы найти чей-то номер телефона с вероятностью 1/2, любой классический алгоритм (детерминированный или вероятностный) должен просмотреть как минимум N/2 имен. Квантово-механические системы могут находиться в состоянии суперпозиции и одновременно просматривать несколько имен. Правильно регулируя фазы различных операций, успешные вычисления усиливают друг друга, в то время как другие оказывают случайное влияние. В результате нужный номер телефона можно получить примерно за квадратный корень из N шагов».*

Телефонный справочник — не самый удачный пример, так как обычно он упорядочен по алфавиту, а не случайным образом. Однако, чтобы не сбиться с пути, указанного Гровером, я все же буду придерживаться этого примера.

В этом тексте есть несколько ключевых слов, которые помогут нам лучше понять проблему:

- «совершенно случайный порядок»;
- «вероятность»;
- «суперпозиция».

Прежде всего очевидно, что в этом примере мы работаем с набором данных, *состоящим из совершенно случайных элементов*. Наша задача — найти нужное имя в большом неупорядоченном списке. В данном случае подразумевается алфавитная неупорядоченность. Я не совсем согласен с примером Гровера, так как обычно имена в телефонном справочнике легко расставляются в алфавитном порядке с помощью классического компьютера. Возможно, в этом контексте лучше рассматривать большой набор ключей, в котором нужно найти тот, который расшифровывает шифротекст, или решение сложного уравнения, способного взломать алгоритм асимметричного шифрования.

Кроме того, во введении к статье Гровера основное внимание уделяется *вероятности* нахождения элемента в неструктурированной базе данных. Таким образом, мы пытаемся увеличить вероятность нахождения необходимого элемента с помощью квантовой системы. В классической системе есть 50%-ная вероятность найти число в ряде случайных чисел, исследуя первую половину содержимого, и такая же вероятность, если исследовать вторую половину содержимого. Очевидно, что при большом везении вы найдете нужный элемент с первой попытки, но вероятность остается 50 %. Мы увидим, что вероятность нахождения элемента увеличивается квадратично при использовании квантового алгоритма.

Во введении также говорится о том, что квантовые вычисления увеличивают вероятность нахождения элемента. Мы можем поместить систему в состояние *суперпозиции* и одновременно исследовать несколько элементов списка, который может состоять из имен, чисел или ключей — это не имеет значения.

Мы также поговорим о том, как достичь этого эффекта и что именно означает введение кубитов в состояние суперпозиции, о котором вы узнали в главе 9.

Правильно подобранные фазы различных операций значительно повышают вероятность нахождения элемента. Она приблизительно равна квадратному корню из N шагов, необходимых в классической системе. В математике можно встретить обозначение $O(N)$ («О большое»), которое определяет верхнюю

границу времени и памяти, используемых алгоритмом для выполнения задачи. Сокращение числа шагов в этом обозначении равно $O\left(\sqrt{N}\right)$.

В простом примере с выбором случайного значения от 1 до 100 в худшем случае классический подход предусматривает 99 попыток. Квантовый подход подразумевает квадратный корень из 100, или 10 *схем*. Далее мы рассмотрим, что такое схема и что значит запустить схему в квантовых вычислениях. Как видите, при использовании алгоритма Гровера мы получим квадратичное ускорение по сравнению с классическим поиском.

Вы вскоре увидите, как можно достичь этой фантастической цели. Однако прежде всего нужно углубиться в квантовые вычисления.

Элементы квантового программирования — квантовая информация и схемы

Если мы хотим начать программировать на квантовом компьютере, нам необходимо разбираться в квантовой информации, квантовых схемах и лежащей в их основе математике.

Однако эта книга не претендует на роль исчерпывающего руководства по квантовому программированию. Я хотел бы объяснить вам основы этой захватывающей области, потому что уверен, что она станет очень важной в будущем, и надеюсь, что вы увлечетесь этой темой.

Я постараюсь изложить материал как можно проще, хотя мы и имеем дело со сложным предметом. Чтобы лучше понять этот раздел, вам необходимо внимательно прочитать главу 2, в частности разделы, посвященные операциям булевой логики, и главу 9.

Сначала поговорим о классической информации и сравним ее с квантовой.

Классическая информация

Как мы видели ранее, в частности, когда говорили о логических и булевых вентилях в главе 2, классическая информация представляется битами 0 и 1. Здесь сосредоточимся на изучении того, что происходит в физической системе (как классической, так и квантовой), когда она хранит информацию. Мы будем называть хранимую информацию X, а ее состояние — S.

При анализе классической информации мы предполагаем, что в каждый момент времени X может находиться только в одном из конечного числа *классических состояний*. Позже рассмотрим кубиты и их квантовые состояния.

Когда мы говорим о классическом состоянии, мы имеем в виду конфигурацию системы, которую можно описать однозначно, без какой-либо неопределенности или ошибки.

На математическом уровне мы просто определяем множество классических состояний. Выбираем их так, чтобы они наилучшим образом описывали устройство, с которым мы работаем. Итак, мы обозначили конечное множество классических состояний S и будем измерять, в каком состоянии находится X.

Например, X может быть битом. В этом случае множество состояний содержит два элемента, которые математически записываются так:

- если X — бит, то его классическое состояние $S = \{0, 1\}$;
- если X — это переключатель на каком-либо устройстве в вашем доме, например на вентиляторе, то классическими состояниями могут быть значения «высоко», «средне», «низко» и «выключено»:

$$S = \{\text{высоко, средне, низко и выключено}\}.$$

Это лишь примеры классической информации, представляемой классической системой, находящейся в различных состояниях в зависимости от используемой системы и устройства. Эти классические состояния совершенно интуитивны, и нам остается только найти способ выразить их математически. С математической точки зрения классические состояния — это конечное множество, которое не может быть пустым. Таким образом, должно существовать хотя бы одно классическое состояние, в котором может находиться система, например «Вкл.» или «Выкл.». Для хранения информации, то есть представления битов {1, 0}, необходимо, чтобы существовало как минимум два классических состояния, в которых может находиться система.

Взгляните на рис. 10.1, где закрашенный круг представляет бит в нулевом состоянии.

Рис. 10.1. Бит, где $S = 0$

Здесь показан классический бит, который может находиться только в двух состояниях. В данном случае закрашенный круг обозначает, что состояние бита равно нулю, а незакрашенный — что альтернативным состоянием бита является единица.

В любой момент времени может существовать *неопределенность* относительно классического состояния системы, поэтому каждое возможное классическое состояние имеет связанную с ним *вероятность*.

Например, если X — это бит, то, возможно, он находится в состоянии 0 с вероятностью $3/4$ или в состоянии 1 с вероятностью $1/4$. Это *вероятностные состояния X*, которые математически можно представить следующим образом:

$$\Pr(X=0)=3/4;\ \Pr(X=1)=1/4.$$

Существует более лаконичный способ описания с помощью *вектора-столбца*:

$$(3/4) \leftarrow \text{вероятность } [0];$$
$$(1/4) \leftarrow \text{вероятность } [1].$$

Например, если мы подбросим монету несколько раз случайным образом, то примерно в 50 % случаев выпадет орел, а в остальных 50 % — решка. Следует иметь в виду, что добиться истинной случайности в классической системе очень сложно или даже невозможно. Однако мы увидим, что можно создать полностью случайный генератор кубитов с помощью квантового алгоритма, который реализуем позже. Здесь же, предполагая, что монета подбрасывается случайно (поскольку мы моделируем теоретическую классическую систему) и не имеет каких-либо нечестных модификаций, получим $1/2$ вероятности состояния 0 = орел и $1/2$ вероятности состояния 1 = решка.

Можно представить состояния случайности в виде линии перехода от 0 до 1 (рис. 10.2).

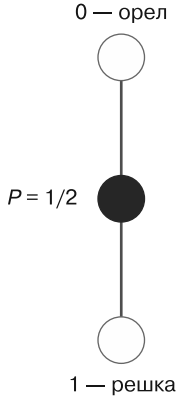


Рис. 10.2. Вероятность выпадения орла или решки при случайном подбрасывании монеты

Здесь представлены вероятности всех возможных состояний монеты до подбрасывания. После подбрасывания классическая монета примет одно определенное и детерминированное значение. Ожидается, что в половине случаев результатом будет орел, в половине — решка.

Это всего лишь графическое представление вероятности бита (0, 1) в классическом состоянии. Как видите, классическое состояние очень линейно и предсказуемо.

Мы также можем выполнять манипуляции с классической информацией и использовать булеву логику (как вы видели в главе 2), чтобы изменить состояние бита или выполнить над ним операции.

Чтобы еще лучше понять это представление (которое подведет нас к квантовому представлению), рассмотрим уже известный нам логический вентиль НЕ.

С помощью булева оператора НЕ можно переместить классический бит из состояния 0 в состояние 1 (рис. 10.3).

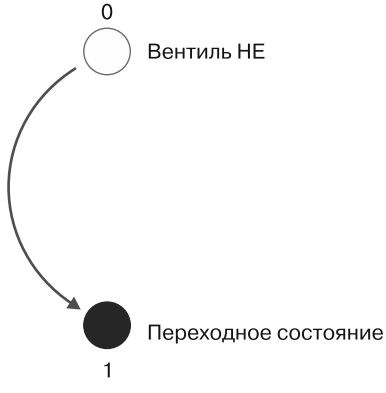


Рис. 10.3. Операция НЕ над классическими битами

На рисунке показано изменение состояния классического бита с 0 на 1 с помощью вентиля НЕ.

Давайте посмотрим, как все это функционирует в квантовой системе, а также исследуем ее особенности.

Квантовая информация, вентили и схемы

Подобно тому как бит является базовым элементом информации в классическом компьютере, кубит — это базовый элемент информации в квантовом компьютере.

Основное различие между классической и квантовой информацией заключается в том, что квантовую систему можно перевести в состояние *суперпозиции*, как вы

уже видели в этой главе. Теперь я хочу на практике объяснить, что это значит и как этого достичь, манипулируя кубитами на квантовом компьютере. Другими словами, хочу познакомить вас с миром квантового программирования.

Для этого мне необходимо ввести понятия *квантовых вентилей* и *квантовых схем*.

Прежде всего квантовый вентиль аналогичен классическому булеву вентилю, но следует помнить, что в квантовых вычислениях мы работаем в трехмерном сферическом пространстве (сфере Блоха), а не на плоскости. Это указывает на фундаментальное отличие от классических вычислений: квантовые частицы (кубиты) могут находиться в более чем двух состояниях.

Как мы уже видели, квантовые состояния обычно представляются *кетами* в системе обозначений бра и кет. На основании того, как выглядит сфера Блоха, можно сказать, что состояние кубита геометрически представляется любой точкой на поверхности сферы (рис. 10.4).

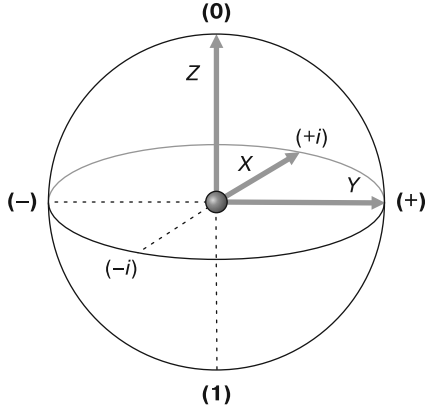


Рис. 10.4. Состояние кубита, представленное на сфере Блоха

Рассмотрим квантовые схемы, которые лежат в основе программирования квантовых алгоритмов и выполнения квантовых вычислений.

Вентили Паули (X, Z)

Вентиль Паули состоит из трех различных вентилей, соответствующих вращению кубита вокруг осей X, Y и Z. Если глубже проанализируем сферу, геометрически представляющую все возможные состояния кубита, то увидим, что $|0\rangle$ и $|1\rangle$ лежат на оси Z, ортогональной по отношению к осям X и Y.

Предположим, наш кубит находится в начальном состоянии $|0\rangle$ на оси Z (рис. 10.5). Если мы применим квантовый оператор НЕ, также называемый *вентилем Паули-X*, то состояние кубита изменится так же, как и при применении классического вентиля НЕ. Состояние кубита $|0\rangle$ преобразуется в $|1\rangle$. И наоборот, если начальное состояние кубита $|1\rangle$, то вентиль Паули-X преобразует его в состояние $|0\rangle$. При начальном состоянии $|0\rangle$ кубит будет повернут на 180° по часовой стрелке вокруг оси X.

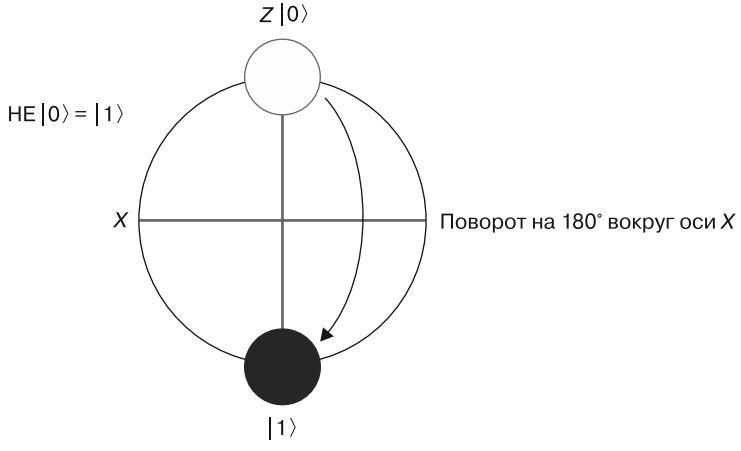


Рис. 10.5. Вентиль Паули-X выполняет поворот на 180° вокруг оси X

Как уже отмечалось, вентиль Паули-X является квантовым эквивалентом вентиля НЕ в булевой логике относительно стандартного базиса $|0\rangle$, $|1\rangle$. При повороте оси Z этот вентиль выполняет поворот на 180° вокруг оси X сферы.

Вентиль Паули-X обозначается следующим символом (рис. 10.6).

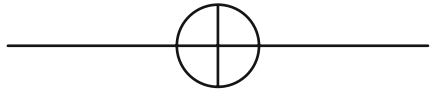


Рис. 10.6. Обозначение вентиля Паули-X

Математически его можно представить в виде матрицы:

$$X = \text{НЕ}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Трехмерное представление вращения вентиля Паули-X выглядит так (рис. 10.7).

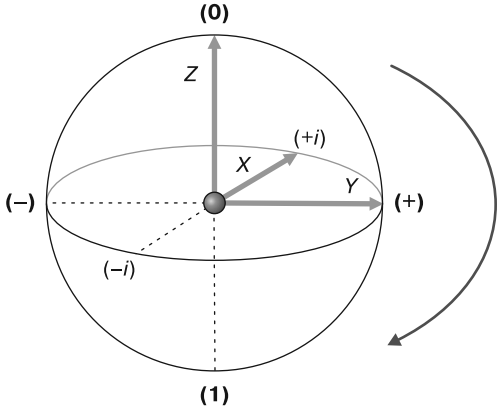


Рис. 10.7. Вращение вентиля Паули-X вокруг оси *X*. Он преобразует состояние 0 в состояние 1

Если вращение осуществляется вокруг оси *Z*, то мы имеем дело с *вентилем Паули-Z*, обозначаемым следующим символом (рис. 10.8).

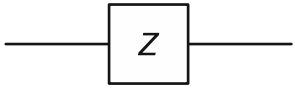


Рис. 10.8. Обозначение вентиля Паули-Z

Можно записать вентиль Паули-Z в виде матрицы:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Вентили Паули-Z сохраняют базисное состояние $|0\rangle$ без изменений, а $|1\rangle$ преобразуют в $|-1\rangle$. Благодаря этому их иногда называют *фазовыми переключателями*. Вентили Паули-Z очень важны для реализации оракула в алгоритме Гровера. *Оракулы*, о которых вы подробнее узнаете далее в этой главе, полезны для увеличения вероятности нахождения искомого элемента.

Лучше понять эту операцию поможет геометрическое изображение вентиля Паули-Z. На рис. 10.9 показано трехмерное представление вращения вокруг осей *Z* и *X*, в результате чего состояние $|-\rangle$ меняется на $|+\rangle$, в то время как состояния $|0\rangle$ и $|1\rangle$ остаются неизменными.

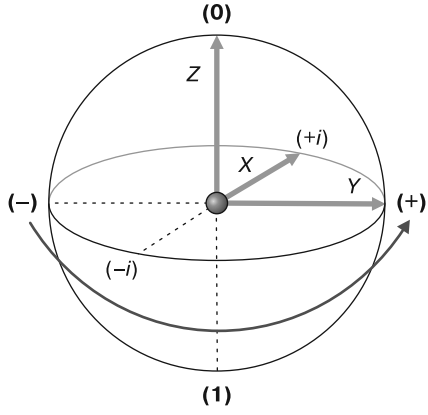


Рис. 10.9. Вращение на 180° вентилем Паули-Z: переход из состояния $|-\rangle$ в состояние $|+\rangle$

Вентиль идентичности

Существует еще один квантовый вентиль — *вентиль идентичности*, противоположный вентилю Паули-X. Это простая матрица, определяющая кубит в его начальном состоянии:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Вентиль идентичности имеет следующее обозначение (рис. 10.10).

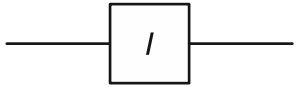


Рис. 10.10. Обозначение вентиля идентичности

Важно! Как вы могли заметить, вентили на схемах изображаются вместе с соединительными линиями, проходящими слева направо. Эти линии передают информацию и соединяют вентили, выполняющие операции над передаваемыми по ним данными. В качестве аналогии можно представить электрическую цепь, где переключатели соединены проводами. В квантовых вычислениях схемы соединяют логические квантовые вентили.

Вентили Адамара

Очень важным квантовым вентилем, необходимым для реализации суперпозиции и *квантовых генераторов случайных чисел* (КГСЧ), является вентиль Адамара.

Вентиль Адамара создает равномерную суперпозицию в системе. Его важность обусловлена часто встречающейся необходимостью рандомизировать начальное состояние перед выполнением вычислений. Вентиль Адамара делает это предсказуемым образом.

Вентиль Адамара выполняет поворот на 90° вокруг оси *X*. Он преобразует кубит из состояния $|0\rangle$ в состояние $|+\rangle$. Взгляните на рис. 10.11.

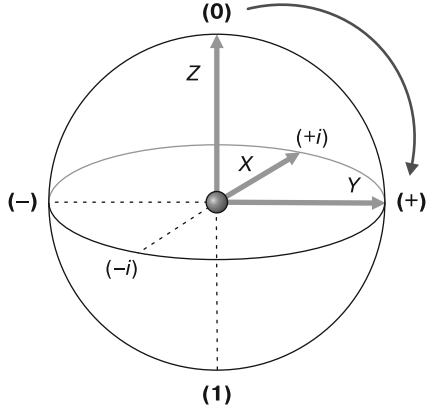


Рис. 10.11. Вентиль Адамара на сфере Блоха

Для обозначения вентиля Адамара используется следующий символ (рис. 10.12).

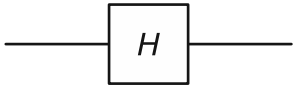


Рис. 10.12. Символ вентиля Адамара

Вентиль Адамара включает в себя все единицы, кроме последнего бита, –1, и выражается матрицей:

$$H = 1/\sqrt{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Важно! $1 / \sqrt{2}$ обеспечивает то, что сумма вероятности всех возможных исходов равна 1.

Как уже говорилось, одним из возможных вариантов применения вентиля Адамара является реализация КГСЧ. Фактически вентиль Адамара ставит все кубиты в равномерную суперпозицию, что эквивалентно случайной возможности принимать любое состояние одновременно.

Одной из проблем криптографии является генерация случайных чисел. Может показаться, что генерировать случайные числа просто, но это не так, особенно в классическом физическом мире. Рассмотрим, например, подбрасывание монеты. Принято считать, что вероятности выпадения орла и решки составляют 50 %. Это не так. Если один и тот же человек многократно подбрасывает монету, он может сознательно влиять на результат, нарушая равномерность распределения.

В квантовой механике ситуация принципиально иная. Используя вентиль Адамара, мы можем реализовать квантовый алгоритм, генерирующий истинно случайные числа, хотя это уже тема для отдельного обсуждения.

Теперь пришло время углубиться в алгоритм Гровера и проанализировать его сложную структуру.

Глубокое погружение в алгоритм Гровера

Учитывая сложность темы и значимость этого алгоритма для квантовых вычислений, я сначала поясню некоторые обозначения и термины, используемые в этой главе, а также расскажу о подходе, который поможет формализовать наши математические выражения.

Если у нас есть функция f, отображающая двоичные строки некоторой длины N в алфавит двоичного кода, приведенный на рис. 10.13, то мы имеем дело с *моделью запроса*, осуществляющей поиск в неструктурированном наборе данных. Наша задача заключается в том, чтобы найти *решение* функции f (входные данные задачи) в неструктурированном наборе данных. Аналогией может служить поиск решения сложных уравнений, способного взломать алгоритм симметричного или асимметричного шифрования, или в более общем смысле — подбор комбинации, открывающей замок банковской ячейки. Именно так применяется этот алгоритм в криптографии.

Важно отметить, что решения может и не быть. Это происходит, когда искомое имя или число отсутствует в наборе данных, и тогда мы просто констатируем, что решения задачи не существует. Тем не менее для объяснения принципа работы рассмотрим *задачу поиска с единственным* решением. Это поможет мне лучше объяснить такую задачу.

Для начала давайте рассмотрим двоичное кодирование английского алфавита (см. рис. 10.13).

Двоичный код букв алфавита

1	A	**00001**	14	N	**01110**
2	B	**00010**	15	O	**01111**
3	C	**00011**	16	P	**10000**
4	D	**00100**	17	Q	**10001**
5	E	**00101**	18	R	**10010**
6	F	**00110**	19	S	**10011**
7	G	**00111**	20	T	**10100**
8	H	**01000**	21	U	**10101**
9	I	**01001**	22	V	**10110**
10	J	**01010**	23	W	**10111**
11	K	**01011**	24	X	**11000**
12	L	**01100**	25	Y	**11001**
13	M	**01101**	26	Z	**11010**

Рис. 10.13. Английский алфавит в двоичном коде

Общее количество входных строк для функции f составляет N, выраженное как 2 в степени n:

$$N = 2^n.$$

Таким образом, N представляет собой квадрат исходной длины входной строки n для функции f.

Вспомним пример, приведенный Гровером в его статье. Повторю, мне он не очень нравится, поскольку поиск в списке имен обычно выполняется по алфавиту. Тем не менее мы рассмотрим его, чтобы избежать путаницы.

Предположим, у нас есть справочник, состоящий из $N = 2^7$ записей, или из 128 имен с телефонными номерами. Слева указана позиция каждого имени в справочнике, за которой идут сами имена и номера телефонов:

[1] Джон: 111 222 3333;

[2] Фил: 222 333 4444;

[3] Виктор: 999 111 0000;

...

[128] Элейн: 333 444 5555.

Вы, вероятно, согласитесь, что поиск номера телефона Элейн (последняя позиция) аналогичен поиску в неупорядоченном массиве чисел, соответствующих каждому имени и расположенных случайным образом (не по алфавиту), как показано на рис. 10.14.

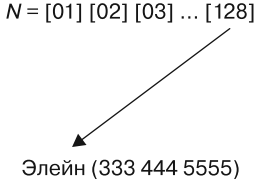


Рис. 10.14. Случайное положение записи Элейн в базе данных

Это происходит потому, что нам неизвестен номер Элейн в списке.

Классический способ найти решение — последовательно перебрать все N элементов до обнаружения имени Элейн на последней позиции [128]. В худшем случае придется перебрать все записи слева направо. Таким образом, если разделить справочник на две части, то вероятность найти Элейн в первой части составит 50 %, во второй — тоже 50 %:

$$50\ \% - - - \{[1, 2... 64]\} \quad \{[65, 55... 128]\} - - - 50\ \%.$$

Помните, что мы знаем нужный элемент (выходное значение — Элейн), но не имеем представления о порядке входных значений и их положения в списке.

Теперь мы можем применить алгоритм Гровера к задаче поиска и посмотреть, как работает квантовая система и насколько она может ускорить поиск.

Алгоритм Гровера можно разделить на три основных этапа.

1. *Инициализация n* кубитов в равной суперпозиции. В последующих разделах мы увидим, как достичь суперпозиции и какие вентили использовать в квантовой схеме. Сейчас мы ничего не знаем о функции f, поэтому нужно начать с равной вероятности найти решение.
2. *Итерация*. Здесь t раз применяется *оператор Гровера G* (позже я расскажу, как определить количество итераций t, а пока предположим, что t — целое число, выбранное произвольно). В нем и заключается суть алгоритма. Далее мы рассмотрим, как реализовать оператор Гровера, который также можно назвать оракулом.
3. *Квантовое измерение* и получение варианта решения. Если мы правильно выберем t, то вероятность нахождения решения будет высока.

Проанализируем эти этапы, и вся картина станет гораздо яснее.

Псевдокод для выполнения алгоритма Гровера

Я хочу показать вам псевдокод для алгоритма Гровера, разделенный на три основных этапа из предыдущего раздела.

1. Подготовка квантового списка Q, инициализированного в состоянии $|0\rangle$, и *применение вентиля Адамара H* для перевода кубитов в состояние суперпозиции.
2. *Применение оператора Гровера G t* раз до достижения максимальной вероятности успешного поиска.
3. *Измерение нового состояния* кубитов и нахождение решения.

Далее представлена общая схема алгоритма Гровера (рис. 10.15). Не беспокойтесь, если пока не все в ней понятно. После моего объяснения вы сможете самостоятельно реализовать этот алгоритм с помощью симулятора Qiskit от IBM.

В квантовых схемах провода представляют кубиты, а вентили — единицы операций и измерений. Схема — это последовательность операций, упорядоченная слева направо.

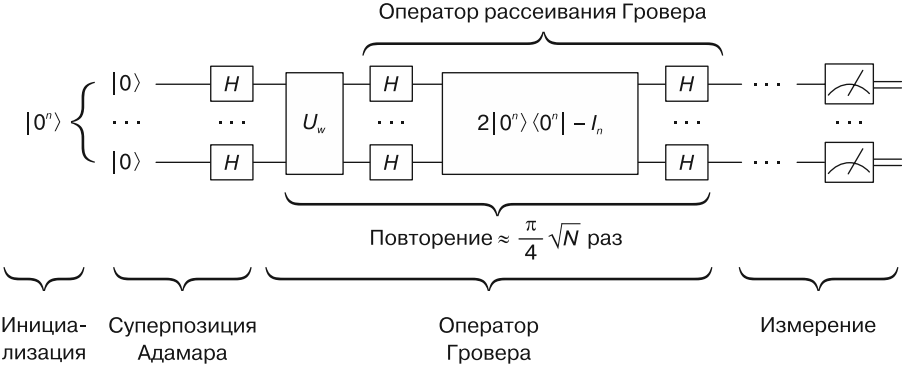


Рис. 10.15. Схема алгоритма Гровера, включающая суперпозицию Адамара, оператор Гровера и измерение

Поочередно рассмотрим этапы этого алгоритма квантового поиска.

Этап 1. Подготовка списка кубитов в суперпозиции

Первое, что мы должны сделать, — подсчитать количество записей в списке и подготовить достаточно кубитов для обработки всех N элементов (искомых имен).

Мы генерируем состояние суперпозиции с помощью вентиля Адамара. Для начального состояния $|0\rangle$ он работает следующим образом.

Поскольку предполагается, что количество элементов составляет [128], мы подготавливаем квантовый список $N = 128$ и переводим все 128 кубитов в состояние суперпозиции.

Рассмотрим состояние суперпозиции подробнее.

Вспомним мысленный эксперимент из главы 9 с шариками, которые ведут себя одновременно как частицы и как волны. С практической точки зрения лучшее определение суперпозиции было дано в Массачусетском технологическом институте — это *способность одновременно находиться в нескольких состояниях*.

Чтобы перевести кубиты в состояние суперпозиции, исследователи манипулируют ими с помощью прецизионных лазеров или микроволновых лучей. По сути, происходит манипуляция поляризацией фотонов и спином электрона, результатом которой является суперпозиция всех состояний частицы.

Процедура перевода кубита в состояние суперпозиции аналогична той, что вы видели в алгоритме Шора в главе 9. Здесь проанализируем ее более подробно. Имея 128 кубитов, инициализированных нулевым состоянием, обозначим их числом $N = 128$ и математически представим как:

$$N = 2^7 = 128 = |0, 000...0\rangle.$$

В предыдущем разделе сказано, что с помощью вентиля Адамара мы получаем суперпозицию.

Как и в алгоритме Шора, выполняем преобразование к каждому отдельному биту (n) внутри обозначений бра и кета до n-го бита, чтобы получить *квантовое состояние суперпозиции*. Таким образом, получаем последовательность вида:

$$|u\rangle = 1/\sqrt{2^n}\left(|0000\ldots0\rangle + |1000\ldots1\rangle + \ldots + |1111\ldots1\rangle\right),$$

где $|u\rangle$ — состояние суперпозиции списка Q. Будем использовать его для обозначения равномерного состояния суперпозиции.

Поскольку мы ничего не знаем о функции f, то задаем одинаковую вероятность для всех состояний с помощью вентилей Адамара.

Где и как можно использовать вентиль Адамара для создания суперпозиции?

Как я говорил в предыдущем разделе, вентиль Адамара переводит все кубиты в состояние равномерной вероятности. На рис. 10.16 показана симуляция для четырех кубитов в Qiskit. В правой части схемы изображена одинаковая вероятность для всех четырех кубитов.

Рис. 10.16. Вентиль Адамара в Qiskit

Четыре кубита приведены здесь в качестве примера. Их должно быть 128, но для простого примера такое количество чрезмерно.

Вспоминая математические обозначения вентиля Адамара, приведу матричное выражение:

$$H = 1/\sqrt{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Вентиль Адамара переводит все кубиты в суперпозицию. В данном случае четыре кубита в равномерной суперпозиции имеют одинаковую вероятность 6,25 %. Фактически 6,25 % · 16 *(все возможные состояния четырех кубитов)* = 100 %. Это отражено на графике симулятора Qiskit от IBM (рис. 10.17).

Рис. 10.17. График, показывающий все 16 базисных состояний, каждое из которых имеет равную вероятность 6,25 %

На рис. 10.18 показана одинаковая суперпозиция кубитов на сфере Блоха, представленной в Qiskit.

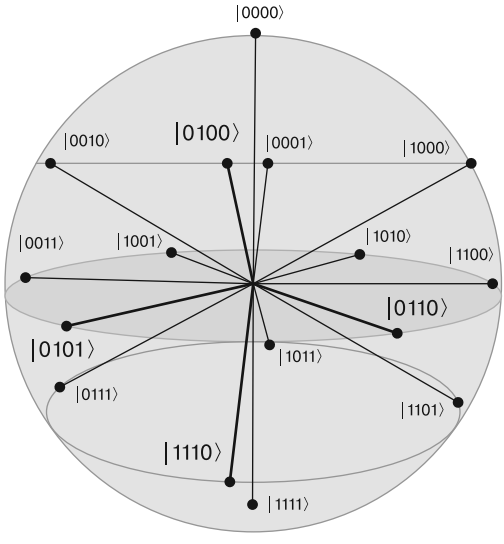


Рис. 10.18. Визуализация всех 16 состояний четырех кубитов в суперпозиции с одинаковыми вероятностями

Теперь, когда мы подготовили список Q и перевели квантовую систему в состояние суперпозиции, приступим к следующему шагу — реализации оракула.

Этап 2. Итерации оператора Гровера G

Это самая сложная часть алгоритма, и я понимаю, что она может вызвать некоторые трудности. Это связано с тем, что нам придется работать с понятиями и терминами, которые в большинстве своем противоречат интуиции.

После перевода наших 128 кубитов в состояние суперпозиции все состояния кубитов имеют одинаковую вероятность быть выбранными. Таким образом, мы находимся в исходной точке — поиск пока не продвинулся.

Возвращаясь к примеру со справочником из 128 имен, напомню, что мы уже знаем цель запроса — Элейн. Теперь необходимо *максимизировать вероятность ее обнаружения*.

Для этого нужно применить специальную технику, называемую *усилением амплитуды*. В геометрии под этим понимается итеративный процесс, позволяющий максимально приблизиться к целевому квантовому состоянию (решению).

Итак, нам требуется вычислить элемент t — количество применений оператора Гровера G, о котором я расскажу в следующем разделе. Однако перед этим нужно погрузиться в отдельные компоненты алгоритма — запросную модель вычислений и оракул.

Запросная модель вычислений

Мы подходим к проблеме квантовых вычислений. Возникает логичный вопрос: зачем использовать квантовый компьютер вместо классического?

Ответ заключается в том, что для определенных задач квантовые компьютеры обеспечивают экспоненциальное ускорение. В случае квантового поиска алгоритм Гровера позволяет достичь такого ускорения. Особый интерес для нас представляет время выполнения вычислений.

Мы будем изучать *модель вычисления запросов*. Она отличается от классического представления о вычислениях, где есть *вход* → *выход*.

Эта модель включает входящие запросы и выход, то есть ответ на задачу, выраженный в виде значений ИСТИНА или ЛОЖЬ, ДА или НЕТ и т. п., что относится скорее к булевой логике, чем к математике. Мы называем секретный вычислительный процесс оракулом, и сейчас увидим, как он работает.

Взгляните на изображение вычислительной модели (рис. 10.19).

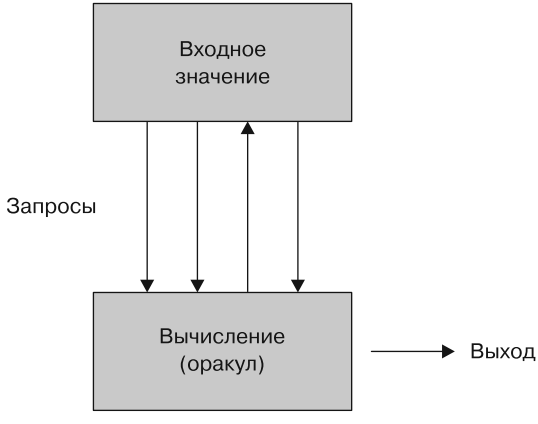


Рис. 10.19. Модель вычисления запроса

Оракулом называется модель, которая на любой заданный вход возвращает один выход: *ДА — НЕТ* или *ИСТИНА — ЛОЖЬ*.

Довольно странно, что такой метод может существовать, потому что он несколько странный и контринтуитивный (я уверен, вы согласитесь со мной). Действи-

тельно, если мы добавим вход (в виде запроса) в такой механизм вычислений, как оракул, его способность давать ответ, вероятно, вызовет вопросы[1].

В Древней Греции при храме Аполлона в Дельфах существовал *Дельфийский оракул*, при котором жили жрицы — пифии. Когда их спрашивали о чем-то, они давали ответы, но не раскрывали всего, что знали. Аналогично черный ящик — это система, которая обрабатывает входные данные и выдает результат, не раскрывая ничего о процессе его определения.

В качестве мысленного эксперимента для упрощения понимания оракула мы можем рассмотреть следующую задачу.

Свяжем каждое имя из нашего списка Q (Джон, Фил, Элейн и т. д.) с какой-либо геометрической фигурой: кругом, шестиугольником, треугольником и т. д. Задача теперь — не найти имя, а определить связанную с ним фигуру. Например, если мы ассоциируем Элейн с прямоугольным треугольником, Джона — с кругом, а Фила — с квадратом, нам нужно будет найти правильный вариант среди множества различных фигур: круга, прямоугольника, квадрата и шестиугольника.

Важным условием здесь является то, что оракул «знает» решение (точно так же, как мы знаем, что результат — это Элейн) и создает фильтр, пропускающий только прямоугольный треугольник. Таким образом, на запрос «найти Элейн» оракул применит соответствующий фильтр и найдет Элейн. На рис. 10.20 изображен фильтр оракула.

Рис. 10.20. Фильтрация оракула по признаку «прямоугольный треугольник»

Однако эта модель все еще имеет одну проблему: скорость нахождения решения оракулом нельзя назвать высокой. Если Элейн, как и прямоугольный треугольник, находится на последнем месте списка, то квантовые вычисления займут столько же ресурсов, сколько и классические. Тем не менее оракул полезен для

[1] В быту оракулом иногда называют гадательные игрушки вроде магического шара, отвечающие на произвольные вопросы случайным образом.

реализации квантового алгоритма, так как он способен правильно настроить систему в соответствии с нашей целью.

Если выражаться более конкретно и математически корректно, входные данные представлены функцией *f*, которую мы можем вычислить. В остальном же мы не знаем и не понимаем, как она работает.

Рассмотрим пример задачи запроса — *задачу поиска с единственным решением*. Вы увидите, что она гораздо проще, чем то, что мы только что рассмотрели.

Задача поиска с единственным решением и вероятность усиления амплитуды

Аналогично булевой схеме функция ИЛИ возвращает определенный результат в ответ на конкретные входные значения функции. К счастью, эту задачу относительно легко понять интуитивно.

В нашем случае есть функция *f*, принимающая *n* кубитов в качестве входных данных.

Выходные данные выглядят следующим образом:

- 1, если функция *f* включает строку кубитов, при которой $f(x) = 1$;
- 0, если такой строки не существует.

Теперь мы переходим к задаче поиска единственного решения, которая связана с проблемой ИЛИ, но не тождественна ей.

Шаг 1. Обязательство относительно входных данных

Здесь мы имеем обязательство относительно входных данных. Входные данные, не удовлетворяющие этому обязательству, считаются незначащими. Другими словами, мы ищем единственную строку (z), которая удовлетворяет условиям поиска.

- *Вход* — $f: \sum n \rightarrow \sum$, где $\sum n$ обозначает двоичный алфавит длиной *n*, а функция *f* отображает двоичные строки длиной *n* в двоичный алфавит $\sum$.
- *Обязательство* — существует ровно одна строка z, принадлежащая $\sum n$, для которой $f(z) = 1$ при $f(x) = 0$ для всех строк $x \neq z$.
- *Выход* — единственная строка z.

Теперь, основываясь на этом обязательстве, разделим множество всех битовых строк на два подмножества — A_0 и A_1:

- A_1 содержит решение задачи поиска;
- A_0 содержит все не-решения.

Нас особенно интересуют два квантовых состояния:

- $|A_0\rangle$ — *равномерная суперпозиция* всех не-решений;
- $|A_1\rangle$ — *равномерная суперпозиция* единственного решения.

Как мы знаем, при применении квантового вентиля обычно меняется состояние кубита. Иными словами, мы управляем спином частиц или поляризацией фотонов, участвующих в этом процессе, чтобы выполнить отражение или поворот угла на сфере Блоха. Поскольку это изменение состояния достигается с помощью таких квантовых вентилей, как вентили Адамара или Паули-Z, мы уделим особое внимание их влиянию на состояние для демонстрации механизма усиления амплитуды вероятности получения решения. В частности, проследим, как поворачивается угол на каждом шаге алгоритма Гровера и как он меняется в процессе итераций.

Разобрав эти аспекты, мы готовы перейти к следующему этапу алгоритма — оператору *G*.

Шаг 2. Оператор Гровера G

В этом разделе я покажу геометрическую интерпретацию действия оператора *G*, применяемого *t* раз для получения решения. Основное внимание будет уделено математике, связанной с нахождением угла θ (тета) и его *отражений (вращений)*, которые обеспечивают *увеличение вероятности* нахождения решения.

Прежде всего рассмотрим геометрическое представление действия двух подшагов применения оператора *G*:

- *подшаг 1* — оракул;
- *подшаг 2* — увеличение вероятности.

Действие *G*, как я уже объяснял в математических терминах, заключается в следующем:

- в отображении выходных данных (оракула) (см. рис. 10.20);
- увеличении вероятности для получения решения (рассеивании) (рис. 10.21 и 10.22).

На рис. 10.21 показано действие применения знака минус к целевому состоянию $|11\rangle$ на примере четырех возможных состояний (оракула).

На рис. 10.22 изображено действие усиления амплитуды, которое для четырех состояний, $|00\rangle$, $|01\rangle$, $|10\rangle$ и $|11\rangle$, повышает вероятность получения решения $|11\rangle$ до 1, то есть до 100 %.

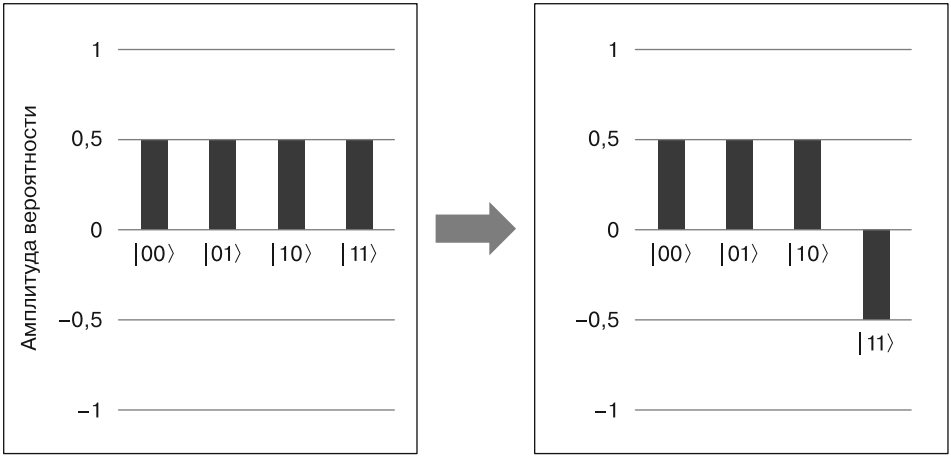


Рис. 10.21. Амплитуда вероятности 1

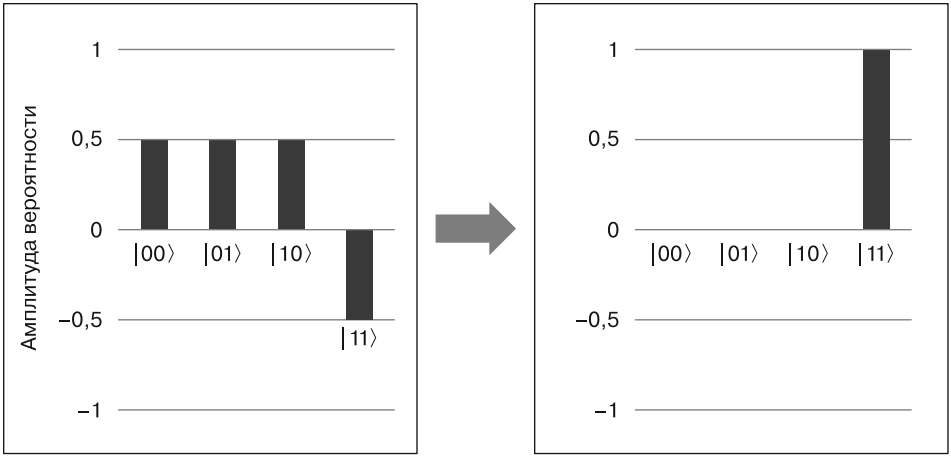


Рис. 10.22. Амплитуда вероятности 2

Теперь нужно определить угол θ и его выражение.

Действие G — это поворот на заданный угол θ, представленный матрицей. Угол задается следующей формулой:

$$\theta = \sin^{-1}\left(\sqrt{|A_1|}\,/\,N\right).$$

Поскольку вращения происходят в двумерном подпространстве, я последовательно покажу геометрическое действие каждой итерации G при поиске нужной строки z, то есть номера телефона Элейн.

Помните, что с помощью обозначения $|u\rangle$ мы описываем состояние суперпозиции списка Q. Таким образом, после инициализации и перед любой итерацией G мы будем иметь начальное состояние кубитов:

$$|u\rangle = \cos(\theta|A_0\rangle) + \sin(\theta|A_1\rangle).$$

На первой итерации G происходит поворот на 2θ. Общий угол будет равен 3θ:

$$G|u\rangle = \cos(3\theta|A_0\rangle) + \sin(3\theta|A_1\rangle).$$

На второй итерации $G_2|u\rangle$ мы выполняем еще один поворот на 2θ. Общий угол будет равен:

$$G_2|u\rangle = \cos(5\theta|A_0\rangle) + \sin(5\theta|A_1\rangle).$$

После t итераций угол будет равен:

$$G_t|u\rangle = \cos((2t\theta + 1)|A_0\rangle) + \sin((2t\theta + 1)|A_1\rangle).$$

По сути, мы применяем поворот на 2θ при каждой итерации G.

Следующая диаграмма показывает отражение угла θ относительно начального состояния в геометрическом представлении (рис. 10.23).

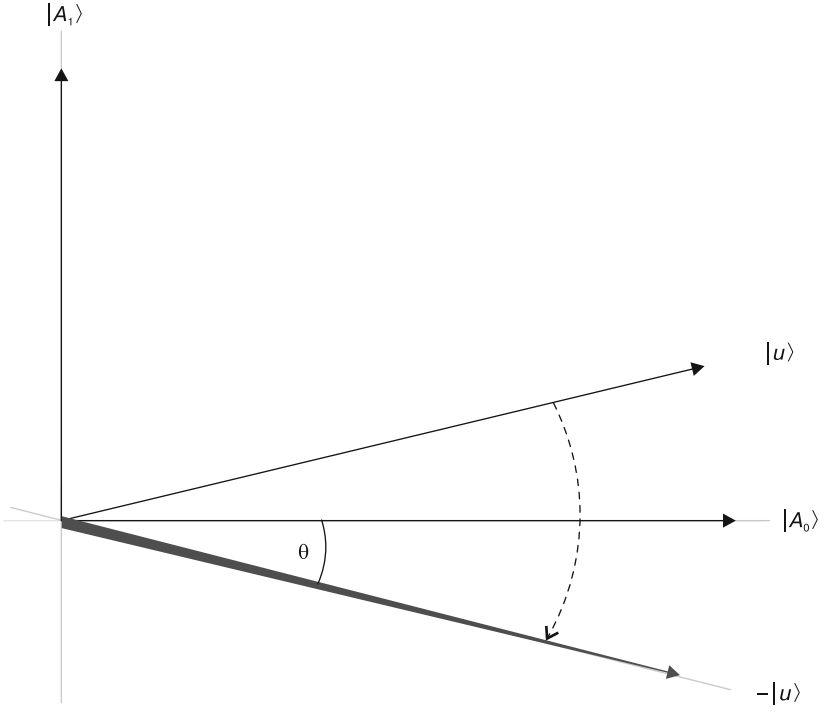


Рис. 10.23. Геометрическое представление отражения угла θ

Поскольку коэффициент при $|A_1\rangle$ равен $\sin(2t\theta + 1)$, вероятность (p) получения решения $|A_1\rangle$ составит:

$$p = \sin^2((2t + 1)\theta).$$

Мы должны максимально увеличить вероятность, в идеале доведя ее до 1. Таким образом, при вращении вектора состояния с помощью итераций G мы рассматриваем $|A_1\rangle$ в качестве целевого состояния. Более того, важно минимизировать значение t, так как оно представляет собой количество запросов, необходимое для достижения цели.

Можно представить количество запросов t как число геометрических фигур, которые должны пройти через фильтр (оракул) на рис. 10.20 для нахождения решения z.

Поэтому, чтобы приблизить эту вероятность к 1 и минимизировать t, имеет смысл стремиться к углу вектора состояния, равному:

$$\left((2t+1)\theta\right) = \pi/2 \leftrightarrow t = \pi/4\theta - 1/2,$$

где $\pi/2$ — это наименьший угол, который максимизирует p и обеспечивает наилучшую вероятность нахождения целевого состояния $|A_1\rangle$.

Следует отметить, что угол не обязательно точно равен $\pi/2$, потому что t должно быть целым числом, а $\pi/2$ таковым не является. Поэтому лучше всего приблизительно выбрать целое число, близкое к $t = \pi/4\theta - 1/2$.

Поскольку t должно быть неотрицательным целым числом, его оптимальным значением будет:

$$t = \pi/4\theta.$$

Это целое положительное число, которое минимизирует количество итераций G для достижения максимальной вероятности успеха.

Есть еще одно важное замечание, которое мы должны принять во внимание. Магическое число итераций $t = \pi/4\theta$ справедливо только для задачи поиска с единственным решением. Другими словами, мы знаем, что решение существует и оно единственное. Поэтому мы говорим именно о поиске единственного решения.

Для поиска мы имеем $s = |A_1\rangle = 1$, поэтому угол θ определяется следующим уравнением:

$$\theta = \sin^{-1}\left(\sqrt{|A_1\rangle}/N\right) \approx \sqrt{1/N}.$$

Поскольку угол θ равен обратному значению синуса квадратного корня из $|A_1\rangle / N$, чем меньше углы, тем больше функция синуса напоминает вентиль идентичности.

Подставив $\theta = \sqrt{1/N}$ в уравнение $t = \pi/4\theta$, получаем:

$$t \approx \pi/4\theta \cdot \sqrt{N}.$$

Итак, теперь у нас есть определенное число, связанное с N, чтобы приблизительно вычислить t. И это хорошо, потому что, как я уже говорил, алгоритм работает примерно как $O\left(\sqrt{N}\right)$.

Посмотрим на примере, что происходит, когда мы повторяем поворот на угол θ для получения решения.

Числовой пример. У нас есть $N = 128$ строк, соответствующих 128 именам в неструктурированном виде:

$$N = 2^7 = 128.$$

Чтобы понять, как работает алгоритм, достаточно определить угол θ, который можно вычислить с помощью следующего математического выражения:

$$\theta = \sqrt{1/N} = \sqrt{1/128} = 0{,}0885.$$

Если подставить это значение в формулу для t, получим:

$$t \approx \pi/\left(4/\theta\right) \approx 8.$$

Итак, нам нужно сделать восемь запросов для получения решения. Давайте искать решение!

На рис. 10.24 показано начальное состояние суперпозиции $|u\rangle$. Помните, что $|A_1\rangle$ — это целевое решение, а $|A_0\rangle$ представляет собой равномерную суперпозицию всех строк.

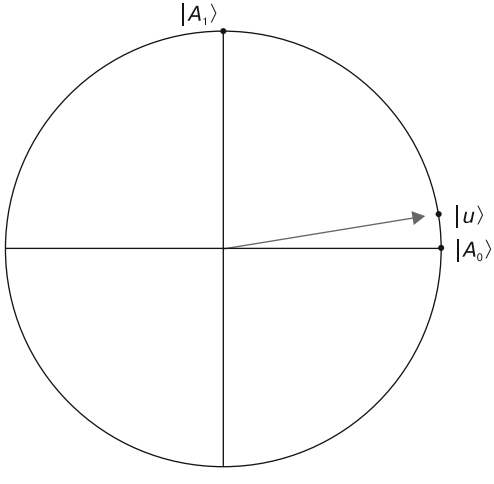


Рис. 10.24. Состояние суперпозиции $|u\rangle$

Как видите, $|A_0\rangle$ и $|u\rangle$ близки, потому что $|A_0\rangle$ — это почти суперпозиция всех строк. Не хватает только одной, которая и является единственным решением.

Применяя G восемь раз, мы непрерывно поворачиваем состояние в сторону $|A_1\rangle$. Наконец, на восьмой итерации достигаем состояния, наиболее близкого к $|A_1\rangle$. Эти восемь итераций на 128 строках являются наилучшей оптимизацией, достигаемой алгоритмом Гровера (рис. 10.25).

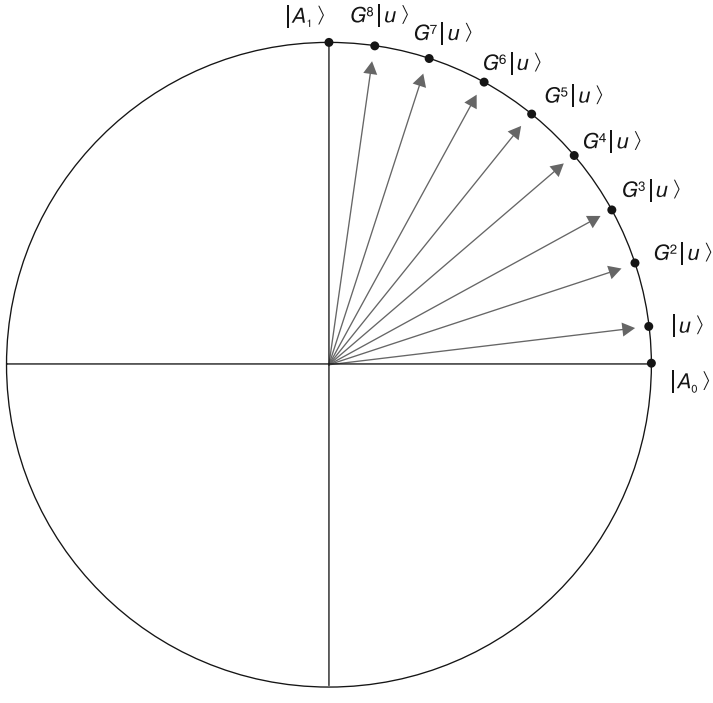


Рис. 10.25. Восемь итераций G для достижения решения

На восьмой итерации мы достигаем точки, наиболее близкой к целевому состоянию $|A_1\rangle$.

Теперь посмотрим, что произойдет при квантовом измерении.

Шаг 3. Квантовое измерение

Если мы остановимся и проведем измерение в этой точке, то получим решение $|A_1\rangle$. В этом случае вероятность успеха составляет 99,5 %. Если же продолжить итерации, то возникнет обратный эффект, отдаляющий от решения.

В таблице на рис. 10.26 показаны вероятности для малых степеней двойки. Напомню, что $128 = 2^7$.

Вероятности нахождения единственного решения

N	*p*(*N*, 1)	*N*	*p*(*N*, 1)
2	50 %	128	99,56199 %
4	100 %	256	99,99470 %
8	94,53125 %	512	99,94480 %
16	96,13190 %	1024	99,94612 %
32	99,91823 %	2048	99,99968 %
64	99,65857 %	4096	99,99453 %

Рис. 10.26. Вероятность успеха при различных значениях *N*

Мы видим, что при $N = 2$ вероятность равна 50 %, что не является удовлетворительным вариантом. При $N = 4$ вероятность равна 100 %. В этом случае мы получаем решение всего за один запрос.

Как видно из таблицы, для диапазона от 1000 до 4000 строк квантовое измерение дает вероятность, очень близкую к 100 % (> 99,9 %).

Резюме

В этой главе мы проанализировали квантовые вычисления и алгоритмы квантового поиска. Познакомившись с основными принципами квантового программирования и квантовыми вентилями, вы узнали, как построить алгоритм Гровера для выполнения квантового поиска.

Тем не менее алгоритм Гровера еще не улучшил процесс экспоненциального поиска в неструктурированной базе данных. Он всего лишь важная отправная точка для создания новых эффективных квантовых алгоритмов. Как вы видели в этой главе, такая модель поиска пока неосуществима из-за огромного числа кубитов, необходимых для получения решения. В перспективе применение квантового поиска может оказаться разрушительным для классических алгоритмов. Проблема поиска в неструктурированной базе данных аналогична перебору ключей в симметричной криптографии. Поэтому, если в будущем поднимется квантовая волна, она, вероятно, смоет многие классические алгоритмы.

Теперь, когда мы дошли до конца книги, примите мои поздравления: вы прошли сложный путь через концепции, трудные доказательства и объяснения алгоритмов. Надеюсь, я помог сделать криптографию понятнее. Продолжайте учиться! Приглашаю вас посетить сайт QuantumLab, в работе которого я принимаю участие (www.quantumlab.science), и присоединиться к нашему сообществу энтузиастов квантовой криптографии.

Read IT Club

Комьюнити рецензентов
и переводчиков ИТ-литературы

Миссия участников клуба — обеспечить высокое качество профессиональной переводной литературы в России. «Книжные дебагеры» проверяют корректность терминологии и подписей на схемах и иллюстрациях, чтобы сделать книги более понятными русскоязычному читателю. Стать участником Read IT Club может любой ИТ-специалист, готовый поделиться опытом с сообществом.

присоединиться к нам

Харрисон Ферроне

ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ UNITY. ПОПУЛЯРНЫЕ ШАБЛОНЫ И ЛУЧШИЕ ПРАКТИКИ СОЗДАНИЯ ИГР НА UNITY И C#

Хотите научиться писать в Unity понятный и простой в сопровождении код для игр? Тогда вы обратились по адресу! Изучение популярных паттернов проектирования Unity позволит использовать весь их потенциал при создании захватывающих проектов. Изучая практические примеры, вы освоите такие порождающие паттерны, как Прототип, помогающий эффективно создавать врагов, и поведенческие паттерны, например Наблюдатель, для реализации реактивной игровой механики. По мере чтения вы также научитесь замечать негативные последствия плохих архитектурных решений и справляться с ними с помощью простых, но эффективных методов.

Когда вы прочтете эту книгу, ваши приемы разработки игр в Unity изменятся. Вы не только станете более опытным разработчиком, но и научитесь использовать в работе передовые паттерны проектирования.

SPRINT book

Бадр Насслахсен

SPRING SECURITY

4-е издание

Опытные хакеры постоянно охотятся за уязвимыми приложениями, поэтому никогда не переставайте беспокоиться о безопасности. Задача становится особенно сложной, если приходится работать с унаследованным кодом, новыми технологиями и сторонними фреймворками. Научитесь защищать Java-приложения с помощью Spring Security — проверенного и гибко настраиваемого фреймворка для аутентификации и управления доступом.

Книга начинается с объяснения того, как реализовать различные механизмы аутентификации и ограничить доступ к вашему приложению. Затем вы узнаете, как совместить Spring Security с популярными веб-фреймворками. Также рассматриваются защита от фиксации сессий, ограничение числа одновременных подключений и использование механизмов управления сессиями для решения административных задач. Четвертое издание книги, обновленное в соответствии с Java 17/21 и Spring Security 6, рассматривает продвинутые сценарии защиты RESTful веб-сервисов и микросервисов. В нем подробно разбираются проблемы, связанные с аутентификацией без сохранения состояния, и дается и краткий пошаговый подход к их решению.

К концу книги вы сможете без труда связать Spring Security 6 с нативными образами GraalVM, что значительно повысит производительность и безопасность приложений.

SPRINT book