**БАРИЧЕВ С.Г., ГОНЧАРОВ В.В., СЕРОВ Р.Е.**

# ОСНОВЫ СОВРЕМЕННОЙ КРИПТОГРАФИИ

Москва
Горячая линия – Телеком
2001

СОДЕРЖАНИЕ

# ПРЕДИСЛОВИЕ К ЭЛЕКТРОННОМУ ИЗДАНИЮ

Настоящая версия книги подготовлена авторами на основе печатного издания:

**БАРИЧЕВ С.Г. и др. «Основы современной криптографии». – М.: «Горячая линия – Телеком», 2001 – 120 с.**

и предназначена для свободного распространения. Любая часть книги может быть использована в любом виде с обязательным указанием источника. Не допускается внесение изменений в полную электронную версию.

Отличие электронной версии от печатной состоит в том, что в электронной версии отсутствуют две главы, посвященные проблемам управления криптографическими ключами и имитозащиты информации, а также приложения о практических аспектах использования криптографических алгоритмов. Кроме того, в печатной версии устранен ряд неточностей, имеющих место в версии электронной.

Печатную версию стоимостью 50 р. (без учета почтовых расходов) можно заказать по почте наложенным платежом, выслав на адрес bars@orc.ru заказ в произвольной форме, указав название книги, количество экземпляров и почтовый адрес. По этому же адресу можно заказать и оптовую поставку указанной книги (от 20 экз.)

# Глава 1
# КРИПТОГРАФИЧЕСКИЕ СИСТЕМЫ

Проблема защиты информации путем ее преобразования, исключающего ее прочтение посторонним лицом, волновала человеческий ум с давних времен. История криптографии - ровесница истории человеческого языка. Более того, первоначально письменность сама по себе была криптографической системой, так как в древних обществах ею владели только избранные. Священные книги древнего Египта, древней Индии тому примеры.

С широким распространением письменности криптография стала формироваться как самостоятельная наука. Первые криптосистемы встречаются уже в начале нашей эры. Так, Цезарь в своей переписке использовал уже более-менее систематический шифр, получивший его имя.

Бурное развитие криптографические системы получили в годы первой и второй мировых войн. Появление вычислительных средств в послевоенные годы ускорило разработку и совершенствование криптографических методов. Вообще история критографии крайне увлекательна, и достойна отдельного рассмотрения, как например в [1].

Почему проблема использования криптографических методов в информационных системах (ИС) стала в настоящий момент особо актуальна?

С одной стороны, расширилось использование компьютерных сетей, в частности, глобальной сети Интернет, по которым передаются большие объемы информации государственного, военного, коммерческого и частного характера, не допускающего возможность доступа к ней посторонних лиц.

С другой стороны, появление новых мощных компьютеров, технологий сетевых и нейронных вычислений сделало возможным дискредитацию криптографических систем, еще недавно считавшихся практически нераскрываемыми.

Все это постоянно подталкивает исследователей на создание новых криптосистем и тщательный анализ уже существующих.

Проблемой защиты информации путем ее преобразования занимается криптология (κρυπτος - тайный, λογος - наука (слово) (греч.)). Криптология разделяется на два направления – *криптографию* и *криптоанализ*. Цели этих направлений прямо противоположны.

*Криптография* занимается поиском и исследованием методов преобразования информации с целью скрытия ее содержания.

Сфера интересов *криптоанализа* - исследование возможности расшифровывания информации без знания ключей.

В книге рассмотрены различные криптографические методы. Современная криптография разделяет их на четыре крупных класса.

1. Симметричные криптосистемы.
2. Криптосистемы с открытым ключом.
3. Системы электронной цифровой подписи (ЭЦП).
4. Системы управление ключами.

Основные направления использования криптографических методов – передача конфиденциальной информации по каналам связи (например, электронная почта), установление подлинности передаваемых сообщений, хранение информации (документов, баз данных) на носителях в зашифрованном виде.

Итак, криптография дает возможность преобразовать информацию таким образом, что ее прочтение (восстановление) возможно только при знании ключа.

В качестве информации, подлежащей шифрованию и расшифрованию, будут рассматриваться *тексты*, построенные на некотором *алфавите*. Под этими терминами понимается следующее.

А*лфавит* - конечное множество используемых для кодирования информации знаков.

*Текст* - упорядоченный набор из элементов алфавита.

В качестве примеров алфавитов, используемых в современных ИС можно привести следующие:

- алфавит $\boldsymbol{Z}_{33}$ – 32 буквы русского алфавита (исключая "ё") и пробел;
- алфавит $\boldsymbol{Z}_{256}$ – символы, входящие в стандартные коды *ASCII* и КОИ-8;
- двоичный алфавит - $\boldsymbol{Z}_2 = \{0,1\}$;
- восьмеричный или шестнадцатеричный алфавит.

*Шифрование* – процесс преобразования исходного текста, который носит также название *открытого текста*, в *шифрованный текст*.

*Расшифрование* – процесс, обратный шифрованию. На основе ключа шифрованный текст преобразуется в исходный.

*Криптографическая система* представляет собой семейство $\boldsymbol{T}$ преобразований открытого текста. Члены этого семейства индексируются, или обозначаются символом $k$; параметр $k$ обычно называется *ключом*. Преобразование $T_k$ определяется соответствующим алгоритмом и значением ключа $k$.

*Ключ* – информация, необходимая для беспрепятственного шифрования и расшифрования текстов.

Пространство ключей $\boldsymbol{K}$ – это набор возможных значений ключа. Обычно ключ представляет собой последовательный ряд букв алфавита.

Криптосистемы подразделяются на *симметричные* и *асимметричные* (или *с открытым ключом*).

В *симметричных криптосистемах* для шифрования, и для расшифрования используется один и тот же ключ.

В *системах с открытым ключом* используются два ключа - *открытый* и *закрытый* (*секретный*), которые математически связаны друг с другом. Информация шифруется с помощью открытого ключа, который доступен всем желающим, а расшифровывается с помощью закрытого ключа, известного только получателю сообщения.

Термины *распределение ключей* и *управление ключами* относятся к процессам системы обработки информации, содержанием которых является выработка и распределение ключей между пользователями.

*Электронной цифровой подписью* называется присоединяемое к тексту его криптографическое преобразование, которое позволяет при получении текста другим пользователем проверить авторство и подлинность сообщения.

*Криптостойкостью* называется характеристика шифра, определяющая его стойкость к расшифрованию без знания ключа (т.е. криптоанализу). Имеется несколько показателей криптостойкости, среди которых:

- количество всех возможных ключей;
- среднее время, необходимое для успешной криптоаналитической атаки того или иного вида.

Эффективность шифрования с целью защиты информации зависит от сохранения тайны ключа и криптостойкости шифра.

## *Требования к криптографическим системам*

Процесс криптографического закрытия данных может осуществляться как программно, так и аппаратно. Аппаратная реализация отличается существенно большей стоимостью, однако ей присущи и преимущества: высокая производительность, простота, защищенность и т.д. Программная реализация более практична, допускает известную гибкость в использовании.

Для современных криптографических систем защиты информации сформулированы следующие общепринятые требования:

- зашифрованное сообщение должно поддаваться чтению только при наличии ключа;
- число операций, необходимых для определения использованного ключа шифрования по фрагменту шифрованного сообщения и соответствующего ему открытого текста, должно быть не меньше общего числа возможных ключей;
- число операций, необходимых для расшифровывания информации путем перебора всевозможных ключей должно иметь строгую нижнюю оценку и выходить за пределы возможностей современных компьютеров (с учетом возможности использования сетевых вычислений), или требовать неоприемлемо высоких затрат на эти вычисления;
- знание алгоритма шифрования не должно влиять на надежность защиты;
- незначительное изменение ключа должно приводить к существенному изменению вида зашифрованного сообщения даже при шифровании одного и того же исходного текста;
- незначительное изменение исходного текста должно приводить к существенному изменению вида зашифрованного сообщения даже при использовании одного и того же ключа;
- структурные элементы алгоритма шифрования должны быть неизменными;
- дополнительные биты, вводимые в сообщение в процессе шифрования, должен быть полностью и надежно скрыты в шифрованном тексте;
- длина шифрованного текста не должна превосходить длину исходного текста;
- не должно быть простых и легко устанавливаемых зависимостей между ключами, последовательно используемыми в процессе шифрования;
- любой ключ из множества возможных должен обеспечивать надежную защиту информации;
- алгоритм должен допускать как программную, так и аппаратную реализацию, при этом изменение длины ключа не должно вести к качественному ухудшению алгоритма шифрования.

# Глава 2

# СИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ

## 2.1. Основные классы симметричных криптосистем

Под симметричными криптографическими системами понимаются такие криптосистемы, в которых для шифрования и расшифрования используется один и тот же ключ. Для пользователей это означает, что прежде, чем начать использовать систему, необходимо получить общий секретный ключ так, чтобы исключить к нему доступ потенциального злоумышленника. Все многообразие симметричных криптосистем основывается на следующих базовых классах.

*Моно- и многоалфавитные подстановки.*

Моноалфавитные подстановки – это наиболее простой вид преобразований, заключающийся в замене символов исходного текста на другие (того же алфавита) по более или менее сложному правилу. В случае моноалфавитных подстановок каждый символ исходного текста преобразуется в символ шифрованного текста по одному и тому же закону. При многоалфавитной подстановке закон преобразования меняется от символа к символу. Для обеспечения высокой криптостойкости требуется использование больших ключей. К этому классу относится так называемая криптосистема с одноразовым ключом, обладающая абсолютной теоретической стойкостью, но, к сожалению, неудобная для практического применения.

*Перестановки.*

Также несложный метод криптографического преобразования, заключающийся в перестановке местами символов исходного текста по некоторому правилу. Шифры перестановок в настоящее время не используются в чистом виде, так как их криптостойкость недостаточна.

*Блочные шифры.*

Представляют собой семейство обратимых преобразований блоков (частей фиксированной длины) исходного текста. Фактически блочный шифр – это система подстановки блоков. В настоящее время блочные шифры наиболее распространены на практике. Российский и американский стандарты шифрования относятся именно к этому классу шифров.

*Гаммирование*.

Представляет собой преобразование исходного текста, при котором символы исходного текста складываются (по модулю, равному мощности алфавита) с символами псевдослучайной последовательности, вырабатываемой по некоторому правилу. Собственно говоря, гаммирование нельзя целиком выделить в отдельный класс криптографических преобразований, так как эта псевдослучайная последовательность может вырабатываться, например, с помощью блочного шифра. В случае, если последовательность является истинно случайной (например, снятой с физического датчика) и каждый ее фрагмент используется только один раз, мы получаем криптосистему с одноразовым ключом.

Количество известных на сегодня симметричных криптосистем веьсма велико, многие из которых были разработаны сотни лет назад. В дальнейшем мы остановимся только на тех, которые используются сегодня на практике, а это в основном так называемые блочные шифры. Такие же криптосистемы как шифры Цезаря, Плейфера, Хилла, Вернама и другие полезны для изучения только с методической точки зрения, их описания можно найти например в [2].

## 2.2. Общие сведения о блочных шифрах

Под *N-разрядным блоком* будем понимать последовательность из нулей и единиц длины $N$:

$$x = (x_0 , x_1 , ..., x_{N-1}) \in Z_{2,N};$$

$x$ в $Z_{2,N}$ можно интерпретировать как вектор и как двоичное представление целого числа

$$\|x\| = \sum_{i=0}^{N-1} x_i 2^{N-i-1} .$$

Например, если $N = 4$, то

(0,0,0,0)→0 (0,0,0,1)→1 (0,0,1,0)→2 (0,0,1,1)→3
(0,1,0,0)→4 (0,1,0,1)→5 (0,1,1,0)→6 (0,1,1,1)→7
(1,0,0,0)→8 (1,0,0,1)→9 (1,0,1,0)→10 (1,0,1,1)→11
(1,1,0,0)→12 (1,1,0,1)→13 (1,1,1,0)→14 (1,1,1,1)→15.

*Блочным шифром* будем называть элемент $\pi \in SYM(Z_{2,N})$:

$$\pi: x \to y = \pi(x),$$

где $x = (x_0, x_1, ..., x_{N-1})$, $x = (y_0, y_1, ..., y_{N-1})$. Хотя блочные шифры являются частными случаями подстановок (только на алфавитах очень большой мощности), их следует рассматривать особо, поскольку, во-первых, большинство симметричных шифров, используемых в системах передачи информации, являются

блочными и, во-вторых, блочные шифры удобнее всего описывать в алгоритмическом виде, а не как обычные подстановки.

Предположим, что

$$\pi(x_i) = y_i \ , \ 0 \le i < m,$$

для некоторого $\pi \in SYM(Z_{2,N})$, исходного текста $X = \{x_i: x_i \in Z_{2,N}\}$ и шифрованного текста $Y = \{y_i\}$. Что можно сказать о $\pi(x)$, если $x \notin \{x_i\}$? Поскольку $\pi$ является перестановкой на $Z_{2,N}$ , то $\{y_i\}$ различны и $\pi(x) \notin \{y_i\}$ при $x \notin \{x_i\}$. Что же еще можно сказать о $\pi$?

$(2^N - m)!$ из $(2^N)!$ перестановок в $SYM(Z_{2,N})$ удовлетворяет уравнению

$$\pi(x_i) = y_i \ , \ 0 \le i < m,$$

Дальнейшая спецификация $\pi(x)$ при отсутствии дополнительной информации не представляется возможной. Это определяется в основном тем обстоятельством, что $\pi$ является элементом, принадлежащим $SYM(Z_{2,N})$. Если известно, что $\pi$ принадлежит небольшому подмножеству П из $SYM(Z_{2,N})$, то можно сделать более определенный вывод. Например, если

$$\Pi = \{\pi_j : 0 \le j < 2\}, \ \pi(i) = (i+j) \ (mod \ 2^N), \ 0 \le i < 2 \ ,$$

то значение $\pi(x)$ при заданном значении $x$ однозначно определяет $\pi$. В этом случае $X$ является подмножеством подстановок Цезаря на $Z_{2,N}$.

Криптографическое значение этого свойства должно быть очевидно: если исходный текст шифруется подстановкой $\pi$, выбранной из полной симметрической группы, то злоумышленник, изучающий соответствие между подмножествами исходного и шифрованного текстов

$$x_i \leftrightarrow y_i \ , \ 0 \le i < m,$$

не в состоянии на основе этой информации определить исходный текст, соответствующий $y \notin \{y_i\}$.

Если для шифрования исходного текста используется подсистема $\pi$ из $\Pi \in SYM(Z_{2,N})$, то получающуюся в результате систему подстановок П будем называть системой *блочных шифров* или *системой блочных подстановок*. Блочный шифр представляет собой частный случай моноалфавитной подстановки с алфавитом $Z_{2^N} = Z_{2,N}$ . Если информация исходного текста не может быть представлена $N$-разрядными блоками, как в случае стандартного алфавитно-цифрового текста, то первое, что нужно сделать, это перекодировать исходный текст именно в этот формат. Перекодирование можно осуществить несколькими способами и с практической точки зрения неважно, какой из способов был выбран.

В установках обработки информации блочные шифры будут использоваться многими пользователями. *Ключевой системой блочных шифров* является подмножество $\Pi[K]$ симметрической группы $SYM(Z_{2,N})$

$$\Pi[K] = \{\pi\{k\}: k \in K\},$$

индексируемое по параметру $k \in K$; $k$ является ключом, а $K$ - пространством ключей. При этом не требуется, чтобы различные ключи соответствовали различным подстановкам $Z_{2,N}$.

Ключевая система блочных шифров $\Pi[K]$ используется следующим образом. Пользователь $i$ и пользователь $j$ некоторым образом заключают соглашение относительно ключа $k$ из $K$, выбирая, таким образом, элемент из $\Pi[K]$ и передавая текст, зашифрованный с использованием выбранной подстановки. Запись

$$y = \pi\{k, x\}$$

будем использовать для обозначения $N$-разрядного блока шифрованного текста, который получен в результате шифрования $N$-разрядного блока исходного текста $x$ с использованием подстановки $\pi\{k\}$, соответствующей ключу $k$. Положим, что злоумышленнику

- известно пространство ключей $K$;
- известен алгоритм определения подстановки $\pi\{k\}$ по значению ключа $k$;
- неизвестно, какой именно ключ $k$ выбрал пользователь.

Какими возможностями располагает злоумышленник? Он может:

- получить ключ вследствие небрежности пользователя $i$ или пользователя j;
- перехватить (путем перехвата телефонных и компьютерных сообщений) шифрованный текст $y$, передаваемый пользователем $i$ пользователю j, и производить пробы на все возможные ключи из $K$ до получения читаемого сообщения исходного текста;
- получить соответствующие исходный и шифрованный тексты ($x \rightarrow y$) и воспользоваться методом пробы на ключ;
- получить соответствующие исходный и шифрованный тексты и исследовать соотношение исходного текста $x$ и шифрованного текста $y$ для определения ключа $k$;
- организовать каталог $N$-разрядных блоков с записью частот их появления в исходном или шифрованном тексте. Каталог дает возможность производить поиск наиболее вероятных слов, используя, например, следующую информацию:

1) листинг на языке ассемблера характеризуется сильно выраженным структурированным форматом, или
2) цифровое представление графической и звуковой информации имеет ограниченный набор знаков.

Предположим, что $N = 64$ и каждый элемент $SYM(Z_{2,N})$ может быть использован как подстановка, так что $K = SYM(Z_{2,N})$.

Тогда:

существует $2^{64}$ 64-разрядных блоков; злоумышленник не может поддерживать каталог с $2^{64} = 1{,}8x10^{19}$ строками;

проба на ключ при числе ключей, равном $(2^{64})!$, практически невозможна; соответствие исходного и шифрованного текстов для некоторых $N$-разрядных блоков $\pi\{k,x_i\} = y_i$ , $0 \le i < m$, не дает злоумышленнику информации относительно значения $\pi\{k, x\}$ для $x \notin \{x_i\}$.

Системы шифрования с блочными шифрами, алфавитом $Z_{2,64}$ и пространством ключей $K = SYM(Z_{2,64})$ являются неделимыми в том смысле, что поддержание каталога частот появления букв для 64-разрядных блоков или проба на ключ при числе ключей $2^{64}$ выходит за пределы возможностей злоумышленника. Следует сравнить эту проблему с той задачей, с которой сталкивается злоумышленник в процессе криптоанализа текста, зашифрованного подстановкой Цезаря с алфавитом $\{A,..., Я,\}$; для определения ключа подстановки Цезаря требуется лишь $log_2 32 = 5$ бит, в то время как для пространства ключей $K = SYM(Z_{2,64})$ требуется $2^{64}$ битов.

К сожалению, разработчик и злоумышленник находятся в одинаковом положении: разработчик не может создать систему, в которой были бы реализованы все $2^{64}!$ подстановок $SYM(Z_{2,64})$, а злоумышленник не может испытать такое число ключей. Остается согласиться с тем, что не каждый элемент из $SYM(Z_{2,64})$ будет использован в качестве подстановки.

Таким образом, требования к хорошему блочному шифру формулируются следующим образом. Необходимы:

- достаточно большое $N$ (64 или более) для того, чтобы затруднить составление и поддержание каталога. В требованиях к новому стандарту шифрования ;
- достаточно большое пространство ключей для того, чтобы исключить возможность подбора ключа;
- сложные соотношения $\pi\{k,x\} : x \to y=\pi\{k,x\}$ между исходным и шифрованным текстами с тем, чтобы аналитические и (или) статистические методы определения исходного текста и (или) ключа на основе соответствия исходного и шифрованного текстов были бы по возможности нереализуемы.

## 2.3. Генерирование блочных шифров

Одним из наиболее распространенных способов задания блочных шифров является использование так называемых сетей Фейстела (по имени исследователя, работавшего в свое время в *IBM*, и одного из авторов стандарта *DES*). Сеть Фейстела представляет собой общий метод преобразования произвольной функции (обычно называемой $F$-функцией) в перестановку на множестве блоков. Эта конструкция была изобретена Хорстом Фейстелом и была использована в большом количестве шифров, включая *DES* и ГОСТ 28147-89. $F$-функция, представляющая собой основной строительный блок сети Фейстела, всегда выбирается нелинейной и практически во всех случаях необратимой.

Формально $F$-функцию можно представить в виде отображения

$$F: Z_{2,N/2} \times Z_{2,k} \to Z_{2,N/2},$$

где $N$ – длина преобразуемого блока текста (должна быть четной), $k$ – длина используемого блока ключевой информации.

Пусть теперь $X$ – блок текста, представим его в виде двух подблоков одинаковой длины $X = \{A, B\}$. Тогда одна итерация (или раунд) сети Фейстела определяется как

$$X_{i+1} = B_i \| (F(B_i, k_i) \oplus A_i)$$

где $X_i = \{A_i, B_i\}$, $\|$ – операция конкатенации, а $\oplus$ – побитовое исключающее ИЛИ. Структура итерации сети Фейстела представлена на рис. 2.1. Сеть Фейстела состоит из некоторого фиксированного числа итераций, определяемого соображениями стойкости разрабатываемого шифра, при этом на последней итерации перестановка местами половин блока текста не производится, т.к. это не влияет на стойкость шифра.

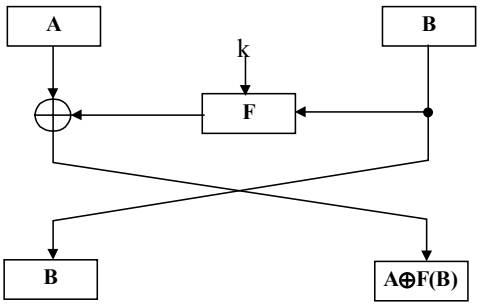



Рис . 2.1. Структура итерации сети Фейстела .

Данная структура шифров обладает рядом достоинств, а именно:

- процедуры шифрования и расшифрования совпадают, с тем исключением, что ключевая информация при расшифровании используется в обратном порядке;
- для построения устройств шифрования можно использовать те же блоки в цепях шифрования и расшифрования.

Недостатком является то, что на каждой итерации изменяется только половина блока обрабатываемого текста, что приводит к необходимости увеличивать число итераций для достижения требуемой стойкости.

В отношении выбора *F*-функции каких-то четких стандартов не существует, однако, как правило, эта функция представляет собой последовательность зависящих от ключа нелинейных замен, перемешивающих перестановок и сдвигов.

Другим подходом к построению блочных шифров является использование обратимых зависящих от ключа преобразований. В этом случае на каждой итерации изменяется весь блок и, соответственно, общее количество итераций может быть сокращено. Каждая итерация представляет собой последовательность преобразований (так называемых "слоев"), каждое из которых выполняет свою функцию. Обычно используются слой нелинейной обратимой замены, слой линейного перемешивания и один или два слоя подмешивания ключа. К недостаткам данного подхода можно отнести то, что для процедур шифрования и расшифрования в общем случае нельзя использовать одни и те же блоки, что увеличивает аппаратные и/или программные затраты на реализацию.

## 2.4. Алгоритмы блочного шифрования

### *Алгоритм DES и его модификации*

Американский стандарт криптографического закрытия данных *DES* (*Data Encryption Standard*) является типичным представителем семейства *блочных шифров*. Этот шифр допускает эффективную аппаратную и программную реализацию, причем возможно достижение скоростей шифрования до нескольких мегабайт в секунду. Шифр *DES* представляет собой результат 33 отображений:

$$DES = IP^{-1} \times \pi_{T_{16}} \times \theta \times \ldots \times \theta \times \pi_{T_1} \times IP, \qquad (2.1)$$

где *IP* (*Initial Permutation* – исходная перестановка) представляет собой проволочную коммутацию с инверсией $IP^{-1}$;

композиция $\theta \times \pi_{T_i}$, где $\theta$ – перестановка местами правой и левой половин блока данных, представляет собой одну итерацию Фейстела. Отметим, что в последнем цикле шифрования по алгоритму *DES* перестановка местами половин блока не производится.

Подстановки $\pi_{T_i}$, $1 \le i \le 16$, описываются следующим образом:

**Шаг 1.** На *i*-м цикле входной блок $\boldsymbol{x}^i$ длиной 64 символа

$$\boldsymbol{x}^i = (x_{i,0}, x_{i,1} \ldots, x_{i,63})$$

делится на два блока по 32 символа $\boldsymbol{X} = (x_{i,0}, x_{i,1} \ldots, x_{i,31})$ и $\boldsymbol{X}' = (x'_{i,0}, x'_{i,1} \ldots, x'_{i,31})$.

Правый блок $\boldsymbol{X}'$ разбивается на восемь блоков по четыре символа:

| | | | |
|---|---|---|---|
| $x'_{i,0}$ | $x'_{i,1}$ | $x'_{i,2}$ | $x'_{i,3}$ |
| $x'_{i,4}$ | $x'_{i,5}$ | $x'_{i,6}$ | $x'_{i,7}$ |
| $x'_{i,8}$ | $x'_{i,9}$ | $x'_{i,10}$ | $x'_{i,11}$ |
| $x'_{i,12}$ | $x'_{i,13}$ | $x'_{i,14}$ | $x'_{i,15}$ |
| $x'_{i,16}$ | $x'_{i,17}$ | $x'_{i,18}$ | $x'_{i,19}$ |
| $x'_{i,20}$ | $x'_{i,21}$ | $x'_{i,22}$ | $x'_{i,23}$ |
| $x'_{i,24}$ | $x'_{i,25}$ | $x'_{i,26}$ | $x'_{i,29}$ |
| $x'_{i,28}$ | $x'_{i,29}$ | $x'_{i,30}$ | $x'_{i,31}$ |

Эти восемь блоков путем копирования крайних элементов преобразуются в восемь блоков из шести символов:

| | | | | | |
|---|---|---|---|---|---|
| $x_{i,31}$ | $x_{i,0}$ | $x_{i,1}$ | $x_{i,2}$ | $x_{i,3}$ | $x_{i,4}$ |
| $x_{i,3}$ | $x_{i,4}$ | $x_{i,5}$ | $x_{i,6}$ | $x_{i,7}$ | $x_{i,8}$ |
| $x_{i,7}$ | $x_{i,8}$ | $x_{i,9}$ | $x_{i,10}$ | $x_{i,11}$ | $x_{i,12}$ |
| $x_{i,11}$ | $x_{i,12}$ | $x_{i,13}$ | $x_{i,14}$ | $x_{i,15}$ | $x_{i,16}$ |
| $x_{i,15}$ | $x_{i,16}$ | $x_{i,17}$ | $x_{i,18}$ | $x_{i,19}$ | $x_{i,20}$ |
| $x_{i,19}$ | $x_{i,20}$ | $x_{i,21}$ | $x_{i,22}$ | $x_{i,23}$ | $x_{i,24}$ |
| $x_{i,23}$ | $x_{i,24}$ | $x_{i,25}$ | $x_{i,26}$ | $x_{i,27}$ | $x_{i,28}$ |
| $x_{i,27}$ | $x_{i,28}$ | $x_{i,29}$ | $x_{i,30}$ | $x_{i,31}$ | $x_{i,0}$ |

**Шаг 2.** На $i$-циклической итерации 48 разрядов ключа

$$(k_{i,0}, k_{i,1}, \dots, k_{i,47}).$$

поразрядно суммируются (по модулю 2) с полученными выше 48 разрядами данных.

**Шаг 3.** $j$-й блок из шести символов ($0 \le j < 8$) подается на вход блока подстановки ($S$-бокс) $S[j]$ имеет шестиразрядный вход и четырехразрядный выход и представляет собой четыре преобразования из $\boldsymbol{Z}_{2,4}$ в $\boldsymbol{Z}_{2,4}$; два крайних разряда входного блока служат для выборки одного из этих преобразований. Каждая из восьми подстановок $S[0]$, $S[1]$,..., $S[7]$ осуществляется с использованием четырех строк и 16 столбцов матрицы с элементами $\{0,1,...,15\}$. Каждый из массивов размерностью $4\times16$ определяет подстановку на множестве $\boldsymbol{Z}_{2,4}$ следующим образом. Если входом является блок из шести символов

$$(z_0, z_1, z_2, z_3, z_4, z_5),$$

то две крайние позиции ($z_0$, $z_5$) интерпретируются как двоичное представление целых чисел из набора $\{0,1,2,3\}$.

Эти целые определяют номер строки (от 0 до 3). Оставшиеся четыре символа ($z_1$, $z_2$, $z_3$, $z_4$) интерпретируются как двоичное представление целых чисел из набора $\{0,1,...,15\}$ и служат для определения столбца в массиве (от 0 до 15). Таким образом, входной блок (0,0,1,0,1,1) соответствует строке 1 и столбцу 5.

**Шаг 4.** 32 разряда, составляющие выход $S$-бокса, подаются на вход блока проволочной коммутации ($P$-бокс).

**Шаг 5.** Компоненты правого входного 32-разрядного блока $\boldsymbol{X}'$, преобразованного в $T(\boldsymbol{X}')$, поразрядно суммируются по модулю 2 с компонентами левого входного 32-разрядного блока $\boldsymbol{X}$.

На каждой итерации используется 48-разрядный ключ ($k_{i,0}$, $k_{i,1}$ ..., $k_{i,47}$). Поскольку входным ключом *DES* является 56-разрядный блок $\boldsymbol{k} = = (k_{i,0}, k_{i,1} \dots, k_{i,55})$, то каждый его разряд используется многократно.

Какой именно из ключей используется на $i$-циклической итерации, определяется списком ключей. Для описания списка ключей введены дополнительные элементы.

Ключ определяется по следующему алгоритму:

производится начальная перестановка $PC$-1 ключа 56-разрядного ключа пользователя $\boldsymbol{k} = (k_{i,0}, k_{i,1} \dots, k_{i,55})$. Получаемый в результате 56-разрядный блок рассматривается как два 28-разрядных блока: левый – $C_0$ и правый – $D_0$;

производится левый циклический сдвиг блоков $C_0$ и $D_0$ $s[1]$ раз для получения блоков $C_1$ и $D_1$;

из сцепления блоков ($C_1$, $D_1$) выбираются 48 разрядов с помощью перестановки $PC$-2. Эти разряды используются на первой итерации;

используемые на $i$-й циклической итерации разряды ключа определяются методом индукции. Для получения блоков $C_i$ и $D_i$ производим левый циклический сдвиг $s[i]$ раз блоков $C_{i-1}$ и $D_{i-1}$.

Инверсией *DES* (обеспечивающей расшифрование зашифрованных посредством *DES* данных) является

$$DES^{-1} = IP^{-1} \times \pi_{T1} \times \pi^* \times \dots \times \pi^* \times \pi_{T16} \times IP, \qquad (2.2)$$

Расшифрование зашифрованного посредством *DES* текста осуществляется с использованием тех же блоков благодаря обратимости преобразования.

Таков общий алгоритм *DES*. Попробуем проанализировать его эффективность.

Поскольку длина блоков исходного текста равна 64, поддержка каталогов частот использования блоков является для злоумышленника задачей, выходящей за пределы современных технических возможностей.

Однако, данный алгоритм, являясь первым опытом стандарта шифрования имеет ряд недостатков. За время, прошедшее после создания *DES*, компьютерная техника развилась настолько быстро, что оказалось возможным осуществлять исчерпывающий перебор ключей и тем самым раскрывать шифр. Стоимость этой атаки постоянно снижается. В 1998 г. была построена машина стоимостью около 100000 долларов, способная по данной паре <исходный текст, шифрованный текст> восстановить ключ за среднее время в 3 суток. Таким образом, *DES*, при его использовании стандартным образом, уже стал далеко не оптимальным выбором для удовлетворения требованиям скрытности данных.

Было выдвинуто большое количество предложений по усовершенствованию *DES*, которые отчасти компенсируют указанные недостатки. Мы рассмотрим два из них.

Наиболее широко известным предложением по усилению *DES* является так называемый «тройной *DES*», одна из версий которого определяется формулой

$$\mathrm{EDE3}_{k_1 k_2 k_3}(\mathbf{x}) = \mathrm{DES}_{k_3}(\mathrm{DES}^{-1}_{k_2}(\mathrm{DES}_{k_1}(\mathbf{x}))).$$

То есть, ключ для *EDE*3 имеет длину 56 × 3 = 168 бит, и шифрование 64-битового блока осуществляется шифрованием с одним подключом, расшифрованием с другим и затем шифрованием с третьим. (Причина, по которой вторым шагом является $\mathrm{DES}_{k_2}^{-1}$, а не $\mathrm{DES}_{k_2}$, является совместимость с *DES*: если выбрать $\boldsymbol{K}=k,k,k$, то $EDE3_{\boldsymbol{K}} = DES_k$. Причина использования *DES* три раза вместо двух заключается в существовании атаки «встреча в середине» на двойной *DES*.)

Проблема с тройным *DES* состоит в том, что он гораздо медленнее, чем сам *DES* – его скорость составляет ровно одну треть исходной. При использовании *EDE*3 в режиме сцепления блоков это замедление скажется как на аппаратном, так и на программном (даже если попытаться компенсировать его дополнительной аппаратной частью) уровнях. Во многих случаях такое падение производительности неприемлемо.

В 1984 г. Рон Ривест предложил расширение *DES*, называемое *DESX* (*DES* e*Xtended*), свободное от недостатков тройного *DES*.

*DESX* определяется как

$$\mathrm{DES}_{k,k_1,k_2} = k_2 \oplus DES_k(k_1 \oplus \boldsymbol{x})$$

То есть, ключ *DESX* $\boldsymbol{K} = k,k_1,k_2$ состоит из 54+64+64=184 бит и включает три различных подключа: ключ «*DES*» $k$, предварительный «зашумляющий» ключ $k_1$ и завершающий «зашумляющий» ключ $k_2$.

Для шифрования блока сообщения мы складываем его поразрядно по модулю 2 с $k_1$, шифруем его алгоритмом *DES* с ключом $k$ и вновь поразрядно складываем его по модулю 2 с $k_2$. Таким образом, затраты *DESX* на шифрование блока всего на две операции сложения по модулю 2 больше, чем затраты исходного алгоритма.

В отношении *DESX* замечательно то, что эти две операции «исключающее ИЛИ» делают шифр гораздо менее уязвимым по отношению к перебору ключей. Укажем, что *DESX* затрудняет получение даже одной пары $<x_i,\ DESX_{\boldsymbol{K}}(x_i)>$ в том случае, когда злоумышленник организует атаку на шифр по выбранному исходному тексту, получая множество пар $<P_j,\ DES_{\boldsymbol{K}}(P_j)>$.

*DESX* предназначался для *увеличения* защищенности *DES* против перебора ключей и *сохранения* его стойкости против других возможных атак. Но *DESX* в действительности также увеличивает стойкость против дифференциального и линейного криптоанализа, увеличивая требуемое количество проб с выбранным исходным текстом до величины, превышающей $2^{60}$. Дальнейшее увеличение стойкости против этих атак может быть достигнуто заменой в *DESX* операции «исключающее ИЛИ» на сложение, как это было сделано в

$$\text{DES - PEP}_{k,k_1,k_2} = k_2 + DES_k(k_1 + \boldsymbol{x})$$

где сложение определяется следующим образом: $L.R + L'.R' = (L \lozenge L').(R \lozenge R')$, $|L|=|R|=|L'|=|R'|= 32$, а $\lozenge$ обозначает сложение по модулю $2^{32}$.

Сказанное не означает, что невозможно построить машину, раскрывающую *DESX* за приемлемое время. Но оно подразумевает, что такая машина должна использовать какую-либо радикально новую идею. Это не может быть машина, реализующая перебор ключей в общепринятом смысле.

Таким образом, практически во всех отношениях *DESX* оказывается лучше *DES*. Этот алгоритм прост, совместим с *DES*, эффективно реализуем аппаратно, может использовать существующее аппаратное обеспечение *DES* и в его отношении было доказано, что он увеличивает стойкость к атакам, основанным на переборе ключей.

## *Алгоритм RC6*

Национальным институтом стандартов США (*NIST*) был объявлен конкурс на создание нового общенационального стандарта шифрования, который должен прийти на замену *DES*. Разрабатываемому стандарту было присвоено рабочее наименование *AES* (*Advanced Encryption Standard*). В качестве одного из кандидатов фирмой *RSA Data Security, Inc*. был представлен алгоритм *RC*6. В нем предусматривается использование четырех рабочих регистров, а также введена операция целочисленного умножения, позволяющая существенно увеличить возмущения, вносимые каждым циклом шифрования, что приводит к увеличению стойкости и/или возможности сократить число циклов.

*RC*6 является полностью параметризованным алгоритмом шифрования. Конкретная версия *RC*6 обозначается как *RC*6–*w*/*r*/*b*, где $w$ обозначает длину слова в битах, $r$ – ненулевое количество итерационных циклов шифрования, а $b$ – длину ключа в байтах. Во всех вариантах *RC*6-*w*/*r*/*b* работает с четырьмя $w$-битовыми словами, используя шесть базовых операций, обозначаемых следующим образом:

$a + b$ – целочисленное сложение по модулю $2^w$;

$a - b$ – целочисленное вычитание по модулю $2^w$;

$a \oplus b$ – побитовое "исключающее ИЛИ" $w$-битовых слов;

$a \times b$ – целочисленное умножение по модулю $2^w$;

$a << b$ – циклический сдвиг $w$-битового слова влево на величину, заданную $\log_2 w$ младшими битами $b$;

$a >> b$ – циклический сдвиг $w$-битового слова вправо на величину, заданную $\log_2 w$ младшими битами $b$;

Шифрование при помощи *RC*6-*w*/*r*/*b* описывается следующим образом:

Вход: Исходный текст, записанный в 4 $w$-битовых входных регистр
Число циклов шифрования $r$;
Ключевая таблица $S[0; \ldots 2r + 3]$ $w$-битовых слов.

Выход: Шифрованный текст в регистрах $A$, $B$, $C$, $D$.

Процедура:

$B = B + S[0]$
$D = D + S[1]$
*for* $i = 1$ *to* $r$ *do* {
$t = (B \times (2B + 1)) <<< \log_2 w$
$u = (D \times (2D + 1)) <<< \log_2 w$
$A = ((A \oplus t) <<< u) + S[2i]$
$C = ((C \oplus u) <<< t) + S[2i + 1]$
$(A; B; C; D) = (B; C; D; A)$
}
$A = A + S[2r + 2]$
$C = C + S[2r + 3]$

Расшифрование в этих обозначениях выглядит очень похоже:

Вход: Шифрованный текст, записанный в 4 $w$-битовых входных регистр
Число циклов шифрования $r$;
Ключевая таблица $S[0; \ldots 2r + 3]$ $w$-битовых слов.

Выход: Исходный текст в регистрах $A$, $B$, $C$, $D$.

Процедура:

$C = C - S[2r + 3]$
$A = A - S[2r + 2]$
*for* $i = r$ *downto* 1 *do* {
$(A; B; C; D) = (D; A; B; C)$
$u = (D \times (2D + 1)) <<< \log_2 w$
$t = (B \times (2B + 1)) <<< \log_2 w$
$C = ((C - S[2i + 1]) >>> t) \oplus$ u
$A = ((A - S[2i]) >>> u) \oplus t$
}
$D = D - S[1]$
$B = B - S[0]$

Алгоритм вычисления ключей для *RC*6-$w$/$r$/$b$ выглядит следующим образом:

Пользователь задает ключ длиной $b$ байтов. Достаточное число ненулевых байтов дописываются в конец, чтобы получилось целое число слов. Затем эти байты записываются начиная с младшего в массив из $c$ слов, т.е. первый байт ключа записывается в $L[0]$, и т.д., а $L[c - 1]$ при необходимости дополняется со стороны старших разрядов нулевыми байтами. В результате работы алгоритма генерации ключей будет вычислено $2r + 4$ слов, которые будут записаны в массиве $S[0; \ldots; 2r + 3]$.

Константы $P_{32} = B7E15163h$ a*nd* $Q_{32} = 9E3779B9h$ – это константы, , получаемые из двоичного представления $e$ –2, где $e$ – основание натуральных логарифмов, и $\phi$ – 1, где $\phi$ – золотое сечение, соответственно. Подобные же константы могут быть аналогичным образом получены и для *RC*6 с другим размером слова. Выбор констант является в некотором роде произвольным, и поэтому можно использовать и другие константы, получая при этом "частные" версии алгоритма.

Вход: Определенный пользователем $b$-байтовый ключ, предварите
массив $L[0;\ldots c - 1]$;
Число циклов шифрования $r$.

Выход: Ключевая таблица $S[0;\ldots 2r + 4]$ из $w$-битовых слов.

Процедура:

$S[0] = P_w$
*for* $i = 1$ *to* $2r + 3$ *do*
$S[i] = S[i - 1] + Q_w$
$A = B = i = j = 0$
$v = 3 \times max\{c, 2r + 4\}$
*for* $s = 1$ *to* $v$ *do* {
$A = S[i] = (S[i] + A + B) <<< 3$
$B = L[j] = (L[j] + A + B) <<< (A + B)$
$i = (i + 1)$ *mod* $(2r + 4)$
$j = (j + 1)$ *mod* $c$
}

Структура шифра *RC*6 является обобщением сети Фейстела. Блок текста разбивается не на 2, а на 4 подблока, и на каждой итерации изменяются 2 подблока из четырех. При этом в конце итерации шифрования производится циклический сдвиг подблоков влево (при расшифровании, соответственно, вправо). Однако, такое обобщение привело к тому, что было утеряно свойство инвариантности блоков шифрования и расшифрования, хотя это и не является определяющим в оценке данного алгоритма.

## *Российский стандарт шифрования ГОСТ 28147-89*

В Российской Федерации установлен единый стандарт криптографического преобразования текста для информационных систем. Он рекомендован к использованию для защиты любых данных, представленных в виде двоичного кода, хотя не исключаются и другие методы шифрования. Данный стандарт формировался с учетом мирового опыта и, в частности, были приняты во внимание недостатки и нереализованные возможности алгоритма *DES*, поэтому использование стандарта ГОСТ предпочтительнее.

Данный алгоритм также построен с использованием сети Фейстела.

Введем ассоциативную операцию конкатенации, используя для нее мультипликативную запись. Кроме того будем использовать следующие операции сложения:

$A \oplus B$ – побитовое сложение по модулю 2;

$A$ [+] $B$ – сложение по модулю $2^{32}$;

$A$ {+} $B$ – сложение по модулю $2^{32}$–1;.

Алгоритм криптографического преобразования предусматривает несколько режимов работы. Во всех режимах используется ключ $W$ длиной 256 бит, представляемый в виде восьми 32-разрядных чисел $X(i)$.

$$W = X(7)X(6)X(5)X(4)X(3)X(2)X(1)X(0)$$

Для расшифрования используется тот же ключ, но процесс расшифрования является инверсным по отношению к исходному.

Базовым режимом работы алгоритма является *режим простой замены.*

Пусть открытые блоки разбиты на блоки по 64 бит в каждом, которые обозначим как $T_O$.

Очередная последовательность бит $T_O$ разделяется на две последовательности $B(0)$ и $A(0)$ по 32 бита (левый и правый блоки). Далее выполняется итеративный процесс шифрования, описываемый следующими формулами:

для $i$ = 1÷24

$$\begin{cases} A(i) = RK(A(i-1)[+]X_{(i-1)(\mathrm{mod}\ 8)}) \oplus B(i-1) \\ B(i) = A(i-1) \end{cases};$$

для $i$ = 25÷31

$$\begin{cases} A(i) = RK(A(i-1)[+]X_{(32-i)}) \oplus B(i-1) \\ B(i) = A(i-1) \end{cases};$$

и для $i$ = 32

$$\begin{cases} A(32) = A(31) \\ B(32) = RK(A(31)[+]X_0) \oplus B(31) \end{cases}$$

Здесь $i$ обозначает номер итерации. Заметим, что подобно *DES*, на последнем цикле перестановка половин блока не производится.

Функция шифрования включает две операции над 32-разрядным аргументом – $K(\cdot)$ и $R(\cdot)$.

Первая операция является подстановкой. Блок подстановки $K$ состоит из 8 узлов замены $K(1)...K(8)$ с памятью по 64 бита каждый. Поступающий на блок подстановки 32-разрядный вектор разбивается на 8 последовательно идущих 4-разрядных вектора, каждый из который преобразуется в 4-разрядный вектор соответствующим узлом замены, представляющим из себя таблицу из 16 целых чисел в диапазоне 0...15. Входной вектор определяет адрес строки в таблице, число из которой является выходным вектором. Затем полученные 4-разрядные векторы вновь последовательно объединяются в 32-разрядный выходной.

Вторая операция – циклический сдвиг 32-разрядного вектора, полученного в результате подстановки $K$ на 11 шагов влево.

64-разрядный блок зашифрованных данных $T_ш$ представляется в виде

$$T_ш = A(32)B(32).$$

Остальные блоки открытых данных в режиме простой замены зашифровываются аналогично.

Следует учитывать, что данный режим шифрования рекомендуется использовать только для шифрования ключевой информации. Для шифрования данных следует использовать два других режима.

Второй режим шифрования называется *режимом гаммирования*.

Открытые данные, разбитые на 64-разрядные блоки $T_O^{(i)}$ ($i$=1,2,...,$m$), где $m$ определяется объемом шифруемых данных), зашифровываются в режиме гаммирования путем поразрядного сложения по модулю 2 с гаммой шифра $\Gamma_Ш$, которая вырабатывается блоками по 64 бита, т.е.

$$\Gamma_Ш = (\Gamma_Ш^{(1)}, \ldots, \Gamma_Ш^{(m)}).$$

Уравнение шифрования данных в режиме гаммирования может быть представлено в следующем виде:

$$T_Ш^{(i)} = \Gamma_Ш^{(i)} \oplus T_O^{(i)} = A(Y_{i-1}[+]C_2, Z_{i-1}\{+\}C_1) \oplus T_O^{(i)}).$$

В этом уравнении $T_Ш^{(i)}$ обозначает 64-разрядный блок зашифрованного текста, A – функцию шифрования в режиме простой замены (аргументами этой функции являются два 32-разрядных числа). $C_1$ =

01010104$h$ и $C_2$ = 01010101$h$ – константы, заданные в ГОСТ 28147-89. Величины $Y_i$ и $Z_i$ определяются итерационно по мере формирования гаммы следующим образом:

$(Y_0, Z_0) = A(S)$, где $S$ – 64-разрядная двоичная последовательность;
$(Y_i, Z_i) = (Y_{i-1} [+] C_2, Z_{i-1} \{+\} C_1)$, $i = 1, 2, ..., m$.

64-разрядная последовательность $S$, называемая синхропосылкой, не является секретным элементом шифра, но ее наличие необходимо как на передающей стороне, так и на приемной.

*Режим гаммирования с обратной связью* очень похож на режим гаммирования. Как и в последнем, разбитые на 64-разрядные блоки $T_{\text{О}}^{(i)}$ открытые данные зашифровываются путем поразрядного сложения по модулю 2 с гаммой шифра $\Gamma_{\text{Ш}}$, которая вырабатывается блоками по 64 бита:

$$\Gamma_{\text{Ш}} = (\Gamma_{\text{Ш}}^{(1)}, \ldots, \Gamma_{\text{Ш}}^{(m)}).$$

Уравнения шифрования данных в режиме гаммирования с обратной связью выглядят следующим образом:

$$T_{\text{Ш}}^{(1)} = A(S) \oplus T_{\text{О}}^{(1)} = \Gamma_{\text{Ш}}^{(1)} \oplus T_{\text{О}}^{(1)},$$
$$T_{\text{Ш}}^{(i)} = A(T_{\text{Ш}}^{(i-1)}) \oplus T_{\text{О}}^{(i)} = \Gamma_{\text{Ш}}^{(i)} \oplus T_{\text{О}}^{(i)}.$$

## *Алгоритм* SAF*ER*+

Этот алгоритм был предложен калифорнийской корпорацией *Cylink* как один из кандидатов на принятие в качестве нового стандарта шифрования *AES*. Он является примером шифра, не использующего структуру сети Фейстела. Шифр работает с блоками длиной 128 бит и с ключами длиной 128, 192 или 256 бит, в соответствии с требованиями *NIST* к новому стандарту. Процедуры шифрования и расшифрования представляют собой последовательность итераций, число которых зависит от длины ключа и равно 8, 12 и 16 соответственно. При шифровании после (а при расшифровании – перед) всех итераций производится еще одно подмешивание подключа.

Каждая итерация состоит из четырех слоев – подмешивания первого подключа, нелинейной обратимой замены, подмешивания второго подключа и линейного перемешивания. При этом используются только байтовые операции, что делает этот шифр особенно привлекательным для реализации на микропроцессорах малой разрядности.

Слои подмешивания первого и второго подключа имеют сходную структуру. На $i$-й итерации используются два подключа $K_{2i-1}$ и $K_{2i}$ длиной по 128 бит. При добавлении первого подключа байты 1, 4, 5, 8, 9, 12, 13 и 16 текстового блока складываются с соответствующими байтами подключа поразрядно по модулю 2, а байты 2, 3, 6, 7, 10, 11, 14 и 15 складываются с байтами подключа по модулю 256. Второй подключ в конце итерации добавляется аналогично, только те байты, которые складывались поразрядно, теперь складываются по модулю 256, и наоборот.

Слой нелинейной замены устроен следующим образом. Значение $x$ байта $j$ преобразуется в $45^x$ (*mod* 257) для байтов с номерами $j$ = 1, 4, 5, 8, 9, 12, 13 и 16. При этом если $x$ = 128, то $45^{128}$ (*mod* 257) = 256 представляется нулем. Значения байтов с номерами j = 2, 3, 6, 7, 10, 11, 14 и 15 преобразуются в $\log_{45}(x)$, при этом если $x$ = 0, то $\log_{45}(0)$ представляется числом 128. Нетрудно заметить, что эти операции являются обратимыми (в действительности, они обратны друг другу). Производить вычисления экспонент и логарифмов при шифровании и расшифровании не обязательно – можно заранее вычислить таблицы замены (всего для их хранения потребуется 512 байтов) и использовать их при работе.

Линейное перемешивание представляет собой умножение текстового блока справа на специальную невырожденную матрицу $M$. При этом все операции выполняются побайтно по модулю 256.

$$\mathbf{M} = \begin{bmatrix}
2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 & 4 & 2 & 4 & 2 & 1 & 1 & 4 & 4 \\
1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 2 & 2 \\
1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 2 & 1 & 1 \\
1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 \\
4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 1 & 1 & 1 & 1 & 2 & 2 \\
2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 1 & 2 & 2 & 4 & 4 & 1 & 1 \\
1 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\
2 & 1 & 16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 4 & 2 & 4 & 2 \\
2 & 1 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 \\
4 & 2 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 \\
2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 \\
4 & 2 & 2 & 2 & 1 & 1 & 4 & 4 & 1 & 1 & 4 & 2 & 2 & 1 & 16 & 8 \\
4 & 2 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 8 & 4 \\
16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 \\
8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2
\end{bmatrix}$$

Подмешивание подключа $K_{2r+1}$ производится так же, как и подмешивание первого подключа в каждой итерации.

При расшифровании вначале подмешивается подключ $K_{2r+1}$, при этом операция сложения по модулю 256 заменяется на вычитание. Затем выполняются итерации расшифрования.

*i*-я итерация расшифрования выполняет преобразование, обратное к *r*-*i*+1-й итерации шифрования (*r* – число итераций) и также содержит 4 слоя. Вначале текстовый блок умножается на матрицу, обратную к матрице шифрования

$$\mathbf{M}^{-1} = \begin{bmatrix}
2 & -2 & 1 & -2 & 1 & -1 & 4 & -8 & 2 & -4 & 1 & -1 & 1 & -2 & 1 & -1 \\
-4 & 4 & -2 & 4 & -2 & 2 & -8 & 16 & -2 & 4 & -1 & 1 & -1 & 2 & -1 & 1 \\
1 & -2 & 1 & -1 & 2 & -4 & 1 & -1 & 1 & -1 & 1 & -2 & 2 & -2 & 4 & -8 \\
-2 & 4 & -2 & 2 & -2 & 4 & -1 & 1 & -1 & 1 & -1 & 2 & -4 & 4 & -8 & 16 \\
1 & -1 & 2 & -4 & 1 & -1 & 1 & -2 & 1 & -2 & 1 & -1 & 4 & -8 & 2 & -2 \\
-1 & 1 & -2 & 4 & -1 & 1 & -1 & 2 & -2 & 4 & -2 & 2 & -8 & 16 & -4 & 4 \\
2 & -4 & 1 & -1 & 1 & -2 & 1 & -1 & 2 & -2 & 4 & -8 & 1 & -1 & 1 & -2 \\
-2 & 4 & -1 & 1 & -1 & 2 & -1 & 1 & -4 & 4 & -8 & 16 & -2 & 2 & -2 & 4 \\
1 & -1 & 1 & -2 & 1 & -1 & 2 & -4 & 4 & -8 & 2 & -2 & 1 & -2 & 1 & -1 \\
-1 & 1 & -1 & 2 & -1 & 1 & -2 & 4 & -8 & 16 & -4 & 4 & -2 & 4 & -2 & 2 \\
1 & -2 & 1 & -1 & 4 & -8 & 2 & -2 & 1 & -1 & 1 & -2 & 1 & -1 & 2 & -4 \\
-1 & 2 & -1 & 1 & -8 & 16 & -4 & 4 & -2 & 2 & -2 & 4 & -1 & 1 & -2 & 4 \\
4 & -8 & 2 & -2 & 1 & -2 & 1 & -1 & 1 & -2 & 1 & -1 & 2 & -4 & 1 & -1 \\
-8 & 16 & -4 & 4 & -2 & 4 & -2 & 2 & -1 & 2 & -1 & 1 & -2 & 4 & -1 & 1 \\
1 & -1 & 4 & -8 & 2 & -2 & 1 & -2 & 1 & -1 & 2 & -4 & 1 & -1 & 1 & -2 \\
-2 & 2 & -8 & 16 & -4 & 4 & -2 & 4 & -1 & 1 & -2 & 4 & -1 & 1 & -1 & 2
\end{bmatrix}$$

Затем подмешивается подключ $K_{2r-2i+2}$, так же, как и второй подключ в итерации шифрования, только сложение с ключом по модулю 256 заменяется вычитанием.

После этого производится обратная нелинейная замена, т.е. те байты, которые при шифровании возводились в степень, логарифмируются, и наоборот.

И, наконец, подмешивается подключ $K_{2r-2i+1}$ (с учетом тех же замечаний, что относились к подмешиванию подключа $K_{2r-2i+2}$).

Для завершения описания алгоритма осталось указать, как из ключа пользователя получаются подключи для итераций.

При обработке ключа пользователя применяются так называемые *слова смещения* $B_2$, $B_3$, ..., $B_{33}$ длиной по 16 байт, которые вычисляются по формулам:

$$B_{i,j} = \begin{cases} 45^{(45^{17i+j} \bmod 257)} \bmod 257, i = 2 \ldots 17 \\ 45^{17i+j} \bmod 257, i = 18 \ldots 33 \end{cases},$$

где $B_{i,j}$ – $j$-й байт $i$-го слова смещения (j = 1…16). При этом значение $B_{i,j}$=256 представляется нулем. Слова смещения являются константами и могут быть вычислены заранее.

Для длины ключа в 128 бит используются только слова $B_2$, … $B_{17}$, для ключа в 192 бита используются слова смещения $B_2$,…,$B_{25}$, а для ключа в 256 бит используются все слова.

Генерация подключей осуществляется по следующему алгоритму:

В качестве первого подключа используются первые шестнадцать байтов пользовательского ключа. Далее пользовательский ключ записывается в ключевой регистр размером на один байт больше длины ключа. После этого все байты ключа суммируются поразрядно по модулю 2 и результат записывается в дополнительный байт регистра. После чего для получения требуемых подключей повторяется итеративная процедура, заключающаяся в следующем:

1. Содержимое каждого байта в регистре циклически сдвигается влево на 3 позиции;
2. Производится выборка 16 байт из регистра. При этом для получения подключа $K_i$ выбираются идущие подряд байты регистра, начиная с $i$-го (если достигнут конец регистра, то оставшиеся байты выбираются с его начала);
3. Выбранные байты складываются с соответствующими байтами слова смещения $B_i$ по модулю 256. Результат сложения и является подключом $K_i$.

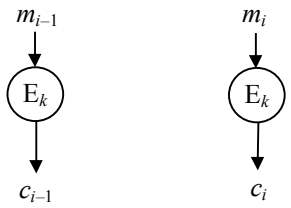
## 2.5. Режимы применения блочных шифров

Для шифрования исходного текста произвольной длины блочные шифры могут быть использованы в нескольких режимах. Мы рассмотрим четыре режима применения блочных шифров, наиболее часто встречающиеся в системах криптографической защиты информации, а именно режимы *электронной кодировочной книги* (*ECB* – Electronic Code *Book*), *сцепления блоков шифрованного текста* (*CBC – Cipher Block Chaining*), *обратной связи по шифрованному тексту* (*CFB – Cipher Feedback*) и *обратной связи по выходу* (*OFB – Output Feedback*).

В режиме *электронной кодировочной книги* каждый блок исходного текста шифруется блочным шифром независимо от других (см. Рис. 2.2).

Стойкость режима *ECB* равна стойкости самого шифра. Однако, структура исходного текста при этом не скрывается. Каждый одинаковый блок исходного текста приводит к появлению одинакового блока шифрованного текста. Исходным текстом можно легко манипулировать путем удаления, повторения или перестановки блоков. Скорость шифрования равна скорости блочного шифра.

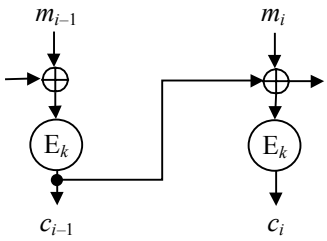
Режим *ECB* допускает простое распараллеливание для увеличения скорости шифрования. К несчастью, никакая обработка невозможна до поступления блока (за исключением генерации ключей). Заметим, что режим *ECB* соответствует режиму простой замены ГОСТ.




$$c_i = E_k(m_i);\ m_i = D_k(c_i)$$

Рис. 2.2. Режим простой замены

В режиме *сцепления блоков шифрованного текста* (*CBC*) каждый блок исходного текста складывается поразрядно по модулю 2 с предыдущим блоком шифрованного текста, а затем шифруется (см. рис. 2.3). Для начала процесса шифрования используется *синхропосылка* (или начальный вектор), которая передается в канал связи в открытом виде.




$$c_i = E_k(m_i \oplus c_{i-1});\ m_i = D_k(c_i) \oplus c_{i-1}$$

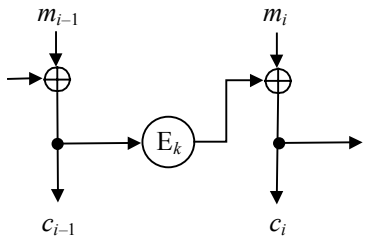
Рис. 2.3. Режим сцепления блоков шифрованного текста.

Стойкость режима *CBC* равна стойкости блочного шифра, лежащего в его основе. Кроме того, структура исходного текста скрывается за счет сложения предыдущего блока шифрованного текста с очередным

блоком открытого текста. Стойкость шифрованного текста увеличивается, поскольку становится невозможной прямая манипуляция исходным текстом, кроме как путем удаления блоков из начала или конца шифрованного текста.

Скорость шифрования равна скорости работы блочного шифра, но простого способа распараллеливания процесса шифрования не существует, хотя расшифрование может проводиться параллельно.

В режиме *обратной связи по шифрованному тексту* (*CFB*) предыдущий блок шифрованного текста шифруется еще раз, и для получения очередного блока шифрованного текста результат складывается поразрядно по модулю 2 с блоком исходного текста. Для начала процесса шифрования также используется начальный вектор (см. рис. 2.4).



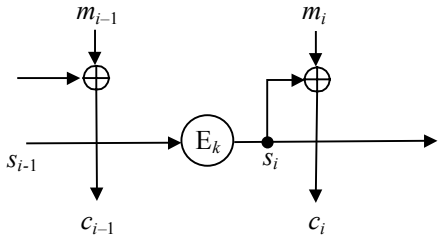

$$c_i = m_i \oplus E_k(c_{i-1}); \; m_i = E_k(c_{i-1}) \oplus m_i$$

Рис. 2.4. Режим обратной связи по шифрованному тексту.

Стойкость режима *CFB* равна стойкости блочного шифра, лежащего в его основе и структура исходного текста скрывается за счет использования операции сложения по модулю 2. Манипулирование исходным текстом путем удаления блоков из начала или конца шифрованного текста становится невозможным. В режиме *CFB* если два блока шифрованного текста идентичны, то результаты их шифрования на следующем шаге также будут идентичны, что создает возможность утечки информации об исходном тексте.

Скорость шифрования равна скорости работы блочного шифра и простого способа распараллеливания процесса шифрования также не существует. Этот режим в точности соответствует режиму гаммирования с обратной связью алгоритма ГОСТ 28147-89.

Режим *обратной связи по выходу* (*OFB*) подобен режиму *CFB* за исключением того, что величины, складываемые по модулю 2 с блоками исходного текста, генерируются независимо от исходного или шифрованного текста. Для начала процесса шифрования также используется начальный вектор. Режим *OFB* обладает преимуществом перед режимом *CFB* в том смысле, что любые битовые ошибки, возникшие в процессе передачи, не влияют на расшифрование последующих блоков. Однако, возможна простая манипуляция исходным текстом путем изменения шифрованного текста.




$$c_i = m_i \oplus s_i; \; m_i = c_i \oplus s_i; \; s_i = E_k(s_{i-1})$$

Рис. 2.5. Режим обратной связи по выходу.

Хотя в этом случае простого способа распараллеливания процесса шифрования также не существует, время можно сэкономить, выработав ключевую последовательность заранее.

# Глава 3
# АСИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ

## 3.1. Общие положения

Еще одним обширным классом криптографических систем являются так называемые асимметричные или двухключевые системы. Эти системы характеризуются тем, что для шифрования и для расшифрования используются разные ключи, связанные между собой некоторой зависимостью. При этом данная зависимость такова, что установить один ключ, зная другой, с вычислительной точки зрения очень трудно.

Один из ключей (например, ключ шифрования) может быть сделан общедоступным, и в этом случае проблема получения общего секретного ключа для связи отпадает. Если сделать общедоступным ключ расшифрования, то на базе полученной системы можно построить систему аутентификации передаваемых сообщений. Поскольку в большинстве случаев один ключ из пары делается общедоступным, такие системы получили также название криптосистем с открытым ключом.

Криптосистема с открытым ключом определяется тремя алгоритмами: генерации ключей, шифрования и расшифрования. Алгоритм генерации ключей открыт, всякий может подать ему на вход случайную строку $r$ надлежащей длины и получить пару ключей ($k_1$, $k_2$). Один из ключей (например, $k_1$) публикуется, он называется открытым, а второй, называемый секретным, хранится в тайне. Алгоритмы шифрования $E_{k_1}$ и расшифрования $D_{k_2}$ таковы, что для любого открытого текста $m$ $D_{k_2}(E_{k_1}(m)) = m$.

Рассмотрим теперь гипотетическую атаку злоумышленника на эту систему. Противнику известен открытый ключ $k_1$, но неизвестен соответствующий секретный ключ $k_2$. Противник перехватил криптограмму $d$ и пытается найти сообщение $m$, где $d = E_{k_1}(m)$. Поскольку алгоритм шифрования открыт, противник может просто последовательно перебрать все возможные сообщения длины $n$, вычислить для каждого такого сообщения $m_i$ криптограмму $d_i = E_{k_1}(m_i)$ и сравнить $d_i$ с $d$. То сообщение, для которого $d_i = d$ и будет искомым открытым текстом. Если повезет, то открытый текст будет найден достаточно быстро. В худшем же случае перебор будет выполнен за время порядка $2^n T(n)$, где $T(n)$ – время, требуемое для шифрования сообщения длины $n$. Если сообщения имеют длину порядка 1000 битов, то такой перебор неосуществим на практике ни на каких самых мощных компьютерах.

Мы рассмотрели лишь один из возможных способов атаки на криптосистему и простейший алгоритм поиска открытого текста, называемый обычно алгоритмом полного перебора. Используется также и другое название: «метод грубой силы». Другой простейший алгоритм поиска открытого текста – угадывание. Этот очевидный алгоритм требует небольших вычислений, но срабатывает с пренебрежимо малой вероятностью (при больших длинах текстов). На самом деле противник может пытаться атаковать криптосистему различными способами и использовать различные, более изощренные алгоритмы поиска открытого текста.

Кроме того, злоумышленник может попытаться восстановить секретный ключ, используя знания (в общем случае несекретные) о математической зависимости между открытым и секретным ключами. Естественно считать криптосистему стойкой, если любой такой алгоритм требует практически неосуществимого объема вычислений или срабатывает с пренебрежимо малой вероятностью. (При этом противник может использовать не только детерминированные, но и вероятностные алгоритмы.) Это и есть теоретико-сложностный подход к определению стойкости. Для его реализации в отношении того или иного типа криптографических систем необходимо выполнить следующее:

1) дать формальное определение системы данного типа;
2) дать формальное определение стойкости системы;
3) доказать стойкость конкретной конструкции системы данного типа.

Здесь сразу же возникает ряд проблем.

Во-первых, для применения теоретико-сложностного подхода необходимо построить математическую модель криптографической системы, зависящую от некоторого параметра, называемого параметром безопасности, который может принимать сколь угодно большие значения (обычно для простоты предполагается, что параметр безопасности может пробегать весь натуральный ряд).

Во-вторых, определение стойкости криптографической системы зависит от той задачи, которая стоит перед противником, и от того, какая информация о схеме ему доступна. Поэтому стойкость систем приходится определять и исследовать отдельно для каждого предположения о противнике.

В-третьих, необходимо уточнить, какой объем вычислений можно считать «практически неосуществимым». Из сказанного выше следует, что эта величина не может быть просто константой, она должна быть представлена функцией от растущего параметра безопасности. В соответствии с тезисом Эдмондса алгоритм считается эффективным, если время его выполнения ограничено некоторым полиномом от длины входного слова (в нашем случае – от параметра безопасности). В противном случае говорят, что вычисления по данному алгоритму практически неосуществимы. Заметим также, что сами криптографические системы должны быть эффективными, т. е. все вычисления, предписанные той или иной схемой, должны выполняться за полиномиальное время.

В-четвертых, необходимо определить, какую вероятность можно считать пренебрежимо малой. В криптографии принято считать таковой любую вероятность, которая для любого полинома $p$ и для всех достаточно больших $n$ не превосходит $1/p(n)$, где $n$ – параметр безопасности.

Итак, при наличии всех указанных выше определений, проблема обоснования стойкости криптографической системы свелась к доказательству отсутствия полиномиального алгоритма, который решает задачу, стоящую перед противником. Но здесь возникает еще одно и весьма серьезное препятствие: современное состояние теории сложности вычислений не позволяет доказывать сверхполиномиальные нижние оценки сложности для конкретных задач рассматриваемого класса. Из этого следует, что на данный момент стойкость криптографических систем может быть установлена лишь с привлечением каких-либо недоказанных предположений. Поэтому основное направление исследований состоит в поиске наиболее слабых достаточных условий (в идеале – необходимых и достаточных) для существования стойких систем каждого из типов.

В основном, рассматриваются предположения двух типов – общие (или теоретико-сложностные) и теоретико-числовые, т. е. предположения о сложности конкретных теоретико-числовых задач. Все эти предположения в литературе обычно называются криптографическими.

Рассмотрим одно из таких предположений.

Обозначим через $\Sigma$ множество всех конечных двоичных слов, а через $\Sigma^n$ – множество всех двоичных слов длины $n$. Подмножества $L \in \Sigma$ в теории сложности принято называть языками. Говорят, что машина Тьюринга $M$ работает за полиномиальное время (или просто, что она полиномиальна), если существует полином $p$ такой, что на любом входном слове длины $n$ машина $M$ останавливается после выполнения не более, чем $p(n)$ операций. Машина Тьюринга $M$ распознает (другой термин – принимает) язык $L$, если на всяком входном слове $x \in L$ машина $M$ останавливается в принимающем состоянии, а на всяком слове $x \notin L$ – в отвергающем. Класс **P** – это класс всех языков, распознаваемых машинами Тьюринга, работающими за полиномиальное время. Функция $f$: $\Sigma \to \Sigma$ вычислима за полиномиальное время, если существует полиномиальная машина Тьюринга такая, что если на вход ей подано слово $x \in \Sigma$, то в момент останова на ленте будет записано значение $f(x)$. Язык $L$ принадлежит классу ***NP***, если существуют предикат $P(x,y)$: $\Sigma\times\Sigma \to \{0,1\}$, вычислимый за полиномиальное время, и полином $p$ такие, что $L = \{x| \exists\, y\ P(x,y)\& \ |y| \le p(|x|)\}$.

То есть язык $L$ принадлежит ***NP***, если для всякого слова из $L$ длины $n$ можно угадать некоторую строку полиномиальной от $n$ длины и затем с помощью предиката $P$ убедиться в правильности догадки. Ясно, что **P** $\subseteq$ ***NP***. Является ли это включение строгим – одна из самых известных нерешенных задач математики. Большинство специалистов считают, что оно строгое (так называемая гипотеза $\boldsymbol{P \neq NP}$). В классе ***NP*** выделен подкласс максимально сложных языков, называемых ***NP***-полными: любой ***NP***-полный язык распознаваем за полиномиальное время тогда и только тогда, когда $\boldsymbol{P = NP}$.

Нам еще потребуется понятие вероятностной машины Тьюринга. В обычных машинах Тьюринга (их называют детерминированными) новое состояние, в которое машина переходит на очередном шаге, полностью определяется текущим состоянием и тем символом, который обозревает головка на ленте.

В вероятностных машинах новое состояние может зависеть еще и от случайной величины, которая принимает значения 0 и 1 с вероятностью 1/2 каждое. Можно считать, что вероятностная машина Тьюринга имеет дополнительную случайную ленту, на которой записана бесконечная двоичная случайная строка. Случайная лента может читаться только в одном направлении и переход в новое состояние может зависеть от символа, обозреваемого на этой ленте.

Рассмотрим теперь следующий естественный вопрос: не является ли гипотеза $\boldsymbol{P \neq NP}$ необходимым и достаточным условием для существования стойких криптографических схем?

В самом деле, необходимость во многих случаях почти очевидна. Вернемся к рассмотренному выше примеру. Определим следующий язык:

$$L = \{(k_1,d,i)| \exists \text{ сообщение } m\colon\ d = E_{k_1}(m) \text{ и } m_i = 1\}.$$

Ясно, что $L \in$ ***NP***: можно просто угадать открытый текст $m$ и проверить за полиномиальное время, что $d = E_{k_1}(m)$ и $i$-й бит $m$ равен 1. Если да, то входное слово $(k_1,d,i)$ принимается, в противном случае – отвергается.

В предположении $\boldsymbol{P = NP}$ существует детерминированный полиномиальный алгоритм, распознающий язык $L$. Зная $k_1$ и $d$, с помощью этого алгоритма можно последовательно, по биту, вычислить открытый текст $m$. Тем самым доказано, что криптосистема нестойкая.

Что же касается вопроса о достаточности предположения $\boldsymbol{P \neq NP}$, то здесь напрашивается следующий подход: выбрать какую-либо ***NP***-полную задачу и построить на ее основе криптографическую схему, задача взлома которой (т. е. задача, стоящая перед противником) была бы ***NP***-полной. Такие попытки предпринимались в начале 80-х годов, в особенности в отношении криптосистем с открытым ключом, но к успеху не привели.

Результатом всех этих попыток стало осознание следующего факта: даже если $\boldsymbol{P \neq NP}$, то любая ***NP***-полная задача может оказаться трудной лишь на некоторой бесконечной последовательности входных слов. Иными словами, в определение класса ***NP*** заложена мера сложности «в худшем случае». Для стойкости же криптографической схемы необходимо, чтобы задача противника была сложной «почти всюду». Таким образом, стало ясно, что для криптографической стойкости необходимо существенно более сильное предположение, чем $\boldsymbol{P \neq NP}$. А именно, предположение о существовании односторонних функций.

## 3.2. Односторонние функции и функции-ловушки

Центральным понятием в теории асимметричных криптографических систем является понятие односторонней функции.

Неформально под *односторонней функцией* понимается эффективно вычислимая функция, для обращения которой (т.е. для поиска хотя бы одного значения аргумента по заданному значению функции) не существует эффективных алгоритмов. Заметим, что обратная функция может и не существовать.

Под функцией мы будем понимать семейство отображений $\{f_n\}$, где $f_n$: $\Sigma^n \to \Sigma^m$, $m = m(n)$. Для простоты предположим, что $n$ пробегает натуральный ряд, а отображения $f_n$ определены всюду. Функция $f$ называется *честной*, если $\exists$ полином $q(x)$, такой что $\forall\ n\ q(m(n)) \geq n$.

Формально понятие односторонней функции описывается следующим образом:

*Определение* 3.1. Четная функция $f$ называется односторонней, если

1. Существует полиномиальный алгоритм, который для всякого $x$ вычисляет $f(x)$;
2. Для любой полиномиальной вероятностной машины Тьюринга $A$ выполнено следующее. Пусть строка $x$ выбрана случайным образом из множества $\Sigma^n$. Тогда для любого полинома $p$ и всех достаточно больших $n$

$$P\{f(A(f(x))) = f(x)\} < 1/p(n).$$

Второе условие качественно означает следующее. Любая полиномиальная вероятностная машина Тьюринга $A$ может по данному $y$ найти $x$ из уравнения $f(x) = y$ лишь с пренебрежимо малой вероятностью.

Заметим, что требование честности опустить нельзя. Поскольку длина входного слова $f(x)$ машины $A$ равна $m$, ей может не хватить полиномиального от $m$ времени просто на выписывание строки $x$.

Существование односторонних функций является необходимым условием стойкости многих криптосистем. Вернемся к примеру, приведенному в п. 3.1. Рассмотрим функцию $f$, такую, что $f(r) = k_1$. Она вычислима с помощью алгоритма $G$ за полиномиальное время. Покажем, что если $f$ – не односторонняя функция, то криптосистема нестойкая. Предположим, что существует полиномиальный вероятностный алгоритм $A$, обращающий $f$ с вероятностью по крайней мере $1/p(n)$ для некоторого полинома $p$. Злоумышленник может подать на вход алгоритма значение ключа $k_1$ и получить с указанной вероятностью некоторое значение $r'$ из прообраза. Далее злоумышленник подает $r'$ на вход алгоритма $G$ и получает пару ключей ($k_1$, $k_2$'). Хотя $k_2$' не обязательно совпадает с $k_2$, по определению криптосистемы $D_{k_2'}(E_{k_1}(m)) = m$ для любого открытого текста $m$. Поскольку $k_2$' найден с вероятностью по крайней мере $1/p(n)$, схема нестойкая.

*Функцией-ловушкой* называется односторонняя функция, для которой обратную функцию вычислить просто, если имеется некоторая дополнительная информация, и сложно, если такая информация отсутствует.

В качестве задач, приводящих к односторонним функциям, можно привести следующие.

1. Разложение числа на простые сомножители.

Вычислить произведение двух простых чисел очень просто. Однако, для решения обратной задачи – разложения заданного числа на простые сомножители, эффективного алгоритма в настоящее время не существует.

2. Дискретное логарифмирование в конечном простом поле.

Допустим, задано большое простое число $p$ и пусть $g$ – примитивный элемент поля $GF(p)$. Тогда для любого $a$ вычислить $g^a(mod\ p)$ просто, а вычислить $a$ по заданным $k = g^a(mod\ p)$ и $p$ оказывается затруднительным.

Криптосистемы с открытым ключом основываются на односторонних функциях-ловушках. При этом открытый ключ определяет конкретную реализацию функции, а секретный ключ дает информацию о ловушке. Любой, знающий ловушку, может легко вычислять функцию в обоих направлениях, но тот, у кого такая информация отсутствует, может производить вычисления только в одном направлении. Прямое направление используется для шифрования и для верификации цифровых подписей, а обратное – для расшифрования и выработки цифровой подписи.

Во всех криптосистемах с открытым ключом чем больше длина ключа, тем выше различие между усилиями, необходимыми для вычисления функции в прямом и обратном направлениях (для того, кто не обладает информацией о ловушке).

Все практические криптосистемы с открытым ключом основываются на функциях, считающихся односторонними, но это свойство не было доказано в отношении ни одной из них. Это означает, что теоретически возможно создание алгоритма, позволяющего легко вычислять обратную функцию без знания информации о ловушке. В этом случае, криптосистема, основанная на этой функции, станет бесполезной. С другой стороны, теоретические исследования могут привести к доказательству существования конкретной нижней границы сложности обращения некоторой функции, и это доказательство будет существенным событием, которое окажет существенное позитивное влияние на развитие криптографии.

## 3.3. Асимметричные системы шифрования

### *Криптосистема Эль-Гамаля*

Система Эль-Гамаля – это криптосистема с открытым ключом, основанная на проблеме логарифма. Система включает как алгоритм шифрования, так и алгоритм цифровой подписи.

Множество параметров системы включает простое число $p$ и целое число $g$, степени которого по модулю $p$ порождают большое число элементов $Z_p$. У пользователя $A$ есть секретный ключ $a$ и открытый ключ $y$, где $y = g^a \pmod p$. Предположим, что пользователь $B$ желает послать сообщение $m$ пользователю $A$. Сначала $B$ выбирает случайное число $k$, меньшее $p$. Затем он вычисляет

$$y_1 = g^k \pmod p \text{ и } y_2 = m \oplus (y^k \pmod p),$$

где $\oplus$ обозначает побитовое "исключающее ИЛИ". $B$ посылает $A$ пару $(y_1, y_2)$.

После получения шифрованного текста пользователь $A$ вычисляет

$m = (y_1^a \bmod p) \oplus y_2$.

Известен вариант этой схемы, когда операция $\oplus$ заменяется на умножение по модулю $p$. Это удобнее в том смысле, что в первом случае текст (или значение хэш-функции) необходимо разбивать на блоки той же длины, что и число $y^k \pmod p$. Во втором случае этого не требуется и можно обрабатывать блоки текста заранее заданной фиксированной длины (меньшей, чем длина числа $p$).

### *Криптосистема Ривеста-Шамира-Эйделмана*

Система Ривеста-Шамира-Эйделмана (*Rivest*, *Shamir*, *Adleman – RSA*) представляет собой криптосистему, стойкость которой основана на сложности решения задачи разложения числа на простые сомножители. Кратко алгоритм можно описать следующим образом:

Пользователь $A$ выбирает пару различных простых чисел $p_A$ и $q_A$ , вычисляет $n_A = p_A q_A$ и выбирает число $d_A$, такое что НОД($d_A$, $\varphi(n_A)$) = 1, где $\varphi(n)$ – функция Эйлера (количество чисел, меньших $n$ и взаимно простых с $n$. Если $n = pq$, где $p$ и $q$ – простые числа, то $\varphi(n) = (p - 1)(q - 1)$). Затем он вычисляет величину $e_A$, такую, что $d_A \cdot e_A = 1 \pmod{\varphi(n_A)}$, и размещает в общедоступной справочной таблице пару $(e_A, n_A)$, являющуюся открытым ключом пользователя $A$.

Теперь пользователь $B$, желая передать сообщение пользователю $A$, представляет исходный текст

$$\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1}), \boldsymbol{x} \in Z_n , 0 \le i < n,$$

по основанию $n_A$:

$$N = c_0 + c_1 n_A + \ldots.$$

Пользователь В зашифровывает текст при передаче его пользователю А, применяя к коэффициентам $c_i$ отображение $\mathrm{E}_{e_\mathrm{A}, n_\mathrm{A}}$ :

$$\mathrm{E}_{e_\mathrm{A}, n_\mathrm{A}} : c \longrightarrow c^{e_\mathrm{A}} (\mathrm{mod}\, n_\mathrm{A}) ,$$

получая зашифрованное сообщение $N'$. В силу выбора чисел $d_A$ и $e_A$, отображение $\mathrm{E}_{e_\mathrm{A}, n_\mathrm{A}}$ является взаимно однозначным, и обратным к нему будет отображение

$$\mathrm{E}_{d_\mathrm{A}, n_\mathrm{A}} : c \longrightarrow c^{d_\mathrm{A}} (\mathrm{mod}\, n_\mathrm{A})$$

Пользователь А производит расшифрование полученного сообщения $N'$, применяя $\mathrm{E}_{d_\mathrm{A}, n_\mathrm{A}}$ .

Для того чтобы найти отображение $\mathrm{E}_{d_\mathrm{A}, n_\mathrm{A}}$ , обратное по отношению к $\mathrm{E}_{e_\mathrm{A}, n_\mathrm{A}}$ , требуется знание множителей $n_A = p_A q_A$. Время выполнения наилучших из известных алгоритмов разложения при $n > 10^{145}$ на сегодняшний день выходит за пределы современных технологических возможностей.

### *Криптосистемы Меркля-Хеллмана и Хора-Ривеста*

Криптосистемы Меркля-Хеллмана и Хора-Ривеста основаны на использовании односторонней функции, известной под названием "задача укладки рюкзака.

Пусть имеется $n$ объектов, так что можно составить $n$-компонентный вектор $\boldsymbol{f}$, так что $i$-й компонент $\boldsymbol{f}$ представляет собой место, занимаемое $i$-м объектом. Имеется рюкзак общим объемом $K$.

Теперь задачу укладки рюкзака может быть сформулирована следующим образом: нам даны $\boldsymbol{f}$ и $K$, и требуется найти битовый вектор $\boldsymbol{x}$, такой что $\boldsymbol{fx}=K$. Доказано, что не существует эффективного алгоритма вычисления $\boldsymbol{x}$ по $\boldsymbol{f}$ и $K$ в общем случае. Таким образом, мы можем использовать вектор $\boldsymbol{f}$ для шифрования $n$-битового сообщения $\boldsymbol{x}$ путем вычисления произведения $K = \boldsymbol{fx}$.

Важно отметить, что выбор $\boldsymbol{f}$ является критическим. Например, предположим, что $\boldsymbol{f}$ выбирается в виде супервозрастающей последовательности. В этой ситуации для любого $i$

$$f_i > \sum_{j=1}^{i-1} f_j \ .$$

В этом случае при данных $\boldsymbol{f}$ и $K$ вычислить $\boldsymbol{x}$ очень просто. Мы проверим, является ли $K$ большим, чем последний элемент $\boldsymbol{f}$, и если да, то мы делаем последний элемент $\boldsymbol{x}$ равным 1, вычитаем это значение из $K$ и рекурсивно решаем меньшую проблему. Этот метод работает, поскольку когда $K$ больше последнего элемента $\boldsymbol{f}$, даже если мы выберем $\boldsymbol{x}$=(1 1 1 … 1 0), то произведение $\boldsymbol{fx}$ все равно будет слишком маленьким, благодаря тому, что последовательность супервозрастающая. Таким образом, мы должны выбирать 1 в последней позиции $\boldsymbol{x}$.

Ясно, что выбор $\boldsymbol{f}$ очень важен – в зависимости от $\boldsymbol{f}$ мы можем получить, а можем и не получить одностороннюю функцию. Однако, именно существование этого простого случая позволяет нам создать функцию-ловушку, которую мы можем использовать для построения криптосистемы с открытым ключом.

Пользователь $A$ получает свой открытый ключ следующим образом:

1. Выбирает супервозрастающую последовательность $\boldsymbol{f}'$ примерно из 100 элементов;
2. Выбирает случайное целое $m$, большее суммы элементов $\boldsymbol{f}'$;
3. Выбирает другое случайное целое $w$, взаимно простое с $m$.
4. Теперь вычисляется $\boldsymbol{f}''$ умножением каждого компонента $\boldsymbol{f}'$ на $w$ по модулю $m$;
$$\boldsymbol{f}'' = \boldsymbol{f}' w \ (mod\ m)$$
5. И наконец, проводится случайная перестановка $\boldsymbol{P}$ элементов $\boldsymbol{f}''$ для получения открытого ключа $\boldsymbol{f}$.

Теперь А раскрывает ключ $\boldsymbol{f}$ и держит в секрете $\boldsymbol{f}'$, $m$, $w$ и $\boldsymbol{P}$.

Когда пользователь В хочет послать А сообщение (битовый вектор) $\boldsymbol{x}$, он вычисляет $S = \boldsymbol{fx}$ и посылает это вычисленное $S$. Если данная система является стойкой, тогда для внешнего наблюдателя С вычисление $\boldsymbol{x}$ по $S$ и публичному ключу $\boldsymbol{f}$ будет эквивалентно решению задачи рюкзака в общем случае. Допустим, что предположение о стойкости верно. В этом случае, хотя С не может расшифровать сообщение, А может это сделать, применяя секретные значения, которые она использовала при вычислении $\boldsymbol{f}$.

Пользователь А может вычислить $S'=\boldsymbol{f}'\boldsymbol{x}$, так что она сможет решить задачу рюкзака в случае супервозрастающей последовательности. Вычисление $S'$ производится следующим образом.

$$S' = \mathbf{f}'\mathbf{x} = \sum_i \mathbf{f}'_i \, \mathbf{x}_i \bmod m = w^{-1} \sum_i w\mathbf{f}'\mathbf{x} \bmod m =$$

$$= w^{-1} \sum_i \mathbf{f}_i \mathbf{x}_i \bmod m = w^{-1} S \bmod m$$

Таким образом, А просто умножает $S$ на мультипликативное обратное $w$ по модулю $m$, а затем решает задачу рюкзака в случае супервозрастающей последовательности $\boldsymbol{f}'$, и теперь она сможет прочитать сообщение.

При получении ключа мы начинаем с супервозрастающей последовательности и затем скрываем имеющуюся в ней структуру. Другими словами, построенная последовательность выглядит так, что в ней нет никакой структуры. Но система может быть сделана еще более стойкой (или, по крайней мере, так будет казаться) путем повторения этого процесса скрытия структуры. Если мы выберем другие $m$ и $w$, тогда мы сможем построить новый открытый ключ, который не может быть построен с использованием единственной пары ($m$, $w$). Мы можем повторять это снова и снова, и система выглядит все более и более стойкой с каждой итерацией.

В 1982 г Эди Шамир открыл атаку на криптосистему, использующую одну итерацию. Это оказалось началом падения систем, основанных на задаче рюкзака.

Допустим, что перестановка не применяется, так что $\boldsymbol{f}'' = \boldsymbol{f}$. Тогда для любого $i$

$$\boldsymbol{f}_i \equiv \boldsymbol{f}'_i w \ mod\ m$$

По определению модульной конгруэнтности должен существовать вектор $\boldsymbol{k}$, такой что для любого $i$

$$u\boldsymbol{f}_i - m\boldsymbol{k}_i = \boldsymbol{f}'_i$$

где $u$ – это мультипликативное обратное к $w$ по модулю $m$ (напомним, что мы выбирали $m$ и $w$ взаимно простыми, так что это обратное существует). После этого в результате деления получаем:

$$\frac{u}{m} - \frac{\mathbf{k}_i}{\mathbf{f}_i} = \frac{\mathbf{f}'_i}{\mathbf{f}_i m}$$

Поскольку $m$ очень велико, выражение справа будет очень маленьким, поэтому покомпонентное частное $\boldsymbol{k}$ и $\boldsymbol{f}$ близко к $u/m$. Подставляя вместо $i$ 1 и вычитая из первоначального уравнения, получим:

$$\left| \frac{\mathbf{k}_i}{\mathbf{f}_i} - \frac{\mathbf{k}_1}{\mathbf{f}_1} \right| = \left| \frac{\mathbf{f}'_i}{m\mathbf{f}_i} - \frac{\mathbf{f}'_1}{m\mathbf{f}_1} \right|$$

Поскольку обе величины справа положительны и вычитаемое очень мало, мы можем записать:

$$\left|\frac{\mathbf{k}_i}{\mathbf{f}_i}-\frac{\mathbf{k}_1}{\mathbf{f}_1}\right|<\frac{\mathbf{f'}_i}{m\mathbf{f}_i}$$

Также заметим, что поскольку ***f'*** супервозрастающая, каждый элемент должен быть меньше половины следующего, поэтому для любого *i* имеем:

$$\boldsymbol{f'}_i < m \cdot 2^{i-n}$$

Далее мы можем записать:

$$\left|\frac{\mathbf{k}_i}{\mathbf{f}_i}-\frac{\mathbf{k}_1}{\mathbf{f}_1}\right|<\frac{2^{i-n}}{\mathbf{f}_i}$$

После несложных преобразований получаем:

$$|\boldsymbol{k}_i \cdot \boldsymbol{f}_1 - \boldsymbol{k}_1 \cdot \boldsymbol{f}_i| < \boldsymbol{f}_1 \cdot 2^{i-n}$$

Оказывается, что поскольку ***f*** открыт, всего лишь несколько этих неравенств (три или четыре) однозначно определяют ***k***. Эти неравенства относятся к области целочисленного программирования, поэтому ***k*** можно быстро найти, например, с помощью алгоритма Ленстры. А если мы знаем ***k***, то мы можем легко раскрыть систему.

Допустим, что мы выполним перестановку ***f*** до опубликования, т.е. ***P*** не является идентичной. Поскольку нам нужны только первые 3 или 4 элемента ***k***, мы можем просто перебрать все варианты, количество которых определяется третьей или четвертой степенью размерности ***k***.

В дальнейшем были разработаны методы вскрытия систем, использующих несколько итераций, и в настоящее время любая система, использующая модульное умножение для скрытия легко разрешимой задачи рюкзака, может быть эффективно раскрыта. Однако, рассмотренный метод не является единственным способом применения задачи рюкзака в криптографии.

В 1986 г. Бен-Цион Хор предложил криптосистему, на сегодняшний день единственную, не использующую модульное умножение для скрытия простой задачи укладки рюкзака. Это также единственная система, основанная на задаче укладки рюкзака, которая не раскрыта.

Во-первых, отметим, что любая супервозрастающая последовательность должна расти экспоненциально, поскольку минимальная супервозрастающая последовательность – это степени двойки. Во-вторых, отметим, что причина, по которой используются супервозрастающие последовательности заключается в том, что любая *h*-элементная сумма из нее уникальна.

Другими словами, если мы представим нашу последовательность в виде вектора f, функция скалярного произведения f на битовый вектор x будет однозначна и поэтому может быть обращена. Но оказывается возможным построить последовательность, растущую только полиномиально, но сохраняющую свойство единственности *h*-элементных сумм. Конструкция такой последовательности была опубликована в 1962 г.

Пусть $GF(p)$ – поле целых чисел по модулю простого числа $p$, и $GF(p^h)$ – расширение степени $h$ основного поля. Также пусть **1** – вектор, все элементы которого равны 1.

С формальной точки зрения мы строим последовательность длины $p$, такую что для любого $i$ от 0 до $p-1$

$$1 \le a_i \le p^h - 1$$

и для каждых различных ***x***, ***y***, таких что ***x**1** = ***y**1** = ***h***, ***x**a*** и ***y**a*** также должны быть различны. Мы можем представлять векторы ***x*** и ***y*** как битовые (т.е. содержащие только 0 и 1).

Далее построение проводится довольно просто. Во-первых, выберем $t$ – алгебраический элемент степени $h$ над $GF(p)$, т.е. минимальный многочлен с коэффициентами из $GF(p)$, корнем которого является $t$, имеет степень $h$. Далее выберем $g$ – мультипликативный генератор (примитивный элемент) поля $GF(p^h)$, т.е. для каждого элемента $x$ из $GF(p^h)$ (кроме нуля) существует некоторое $i$, такое, что $g$ в степени $i$ будет равно $x$.

Теперь рассмотрим аддитивный сдвиг $GF(p)$, т.е. множество

$$t + GF(p) = \{t + i \mid 0 \le i \le p - 1 \} \subset GF(p^h)$$

Пусть каждый элемент вектора **a** будет логарифмом по основанию $g$ соответствующего элемента из $t+GF(p)$:

$$a_i = \log_g(t+i)$$

Мы должны проверить, что **a**, определенная подобным образом, удовлетворяет заданным свойствам. Определенно, каждый элемент в **a** будет лежать в заданном диапазоне, поскольку $g$ порождает $GF(p,h)$. Теперь пусть у нас есть различные ***x*** и ***y***, такие что ***x**1** = ***y**1** = $h$, но ***x**a*** = ***y**a***. Тогда, возводя $g$ в степень ***x**a*** и ***y**a***, получим:

$$g^{\sum_{i=0}^{p-1} x_i a_i} = g^{\sum_{i=0}^{p-1} y_i a_i}$$

Поэтому мы также можем записать

$$\prod_{i=0}^{p-1} (g^{a_i})^{x_i} = \prod_{i=0}^{p-1} (g^{a_i})^{y_i}$$

и далее

$$\prod_{i=0}^{p-1}(t+i)^{x_i} = \prod_{i=0}^{p-1}(t+i)^{y_i} \ .$$

Теперь заметим, что произведение в обеих частях неравенства представляет собой приведенный многочлен от $t$ степени $h$. Иными словами, если бы мы вычислили оба этих произведения и заменили значение $t$ формальным параметром, например, $z$, тогда старшим членом на каждой стороне был бы $x$ в степени $h$ с коэффициентом 1. Мы знаем, что если мы подставим значение $t$ вместо $z$, то значения этих двух полиномов будут равны. Поэтому вычтем один из другого, старшие члены сократятся, и если мы подставим $t$, то получим 0. Мы получили полином степени $h$–1, корнем которого является $t$. Но это противоречит тому, что мы выбрали $t$ алгебраическим элементом степени $h$. Таким образом, доказательство закончено и построение корректно.

Хор разработал метод использования данного построения в качестве основы криптосистемы. Кратко он заключается в следующем. Мы выбираем $p$ и $h$ достаточно маленькими, чтобы мы могли вычислять дискретные логарифмы в $GF(p^h)$. Хор рекомендует $p$ около 200, а $h$ около 25. Затем мы выбираем $t$ и $g$ как указано выше. Для каждого из них будет много вариантов, и мы можем просто произвести случайный выбор (В действительности, будет так много пар $<t,g>$, что очень большое количество пользователей могут использовать одинаковые $p$ и $h$, и вероятность того, что два пользователя выберут одинаковые ключи, будет пренебрежимо мала.). Затем мы следуем конструкции Боуза-Чоула. Мы вычисляем логарифмы по основанию $g$ от $t+i$ для каждого $i$, это даст нам **a**. Наконец, мы выбираем случайную перестановку **a**, которая и будет нашим ключом. Мы публикуем результат перестановки **a** вместе с $p$ и $h$. Величины $t$, $g$ и использованная перестановка остаются в секрете.

Чтобы послать сообщение $A$, $B$ просто берет свое сообщение и вычисляет $S = \boldsymbol{x}^*\boldsymbol{a}$. В действительности, это не так уж и просто, поскольку сообщение должно быть длиной $p$ бит и должно быть $\boldsymbol{x}^*\mathbf{1} = h$, но Хор представил довольно прямолинейный метод преобразования неограниченной битовой строки в несколько блоков, каждый из которых имеет требуемую форму. А получает $S$. Он возводит $g$ в степень $S$ и выражает результат в виде полинома от $t$ степени $h$ с коэффициентами из $GF(p)$. Далее он вычисляет $h$ корней этого полинома, затем применяет обратную подстановку и получает индексы элементов в $\boldsymbol{x}$, содержащих единицы.

Интересно отметить, что если кто-либо откроет эффективный метод вычисления дискретных логарифмов, то такой алгоритм не только не поможет вскрыть эту систему, но и облегчит генерацию ключей, так как при этом мы должны вычислять дискретные логарифмы.

До настоящего времени не было опубликовано ни одного эффективного метода вскрытия этой системы при знании только открытого ключа.

## *Криптосистема, основанная на эллиптических кривых*

Рассмотренная выше криптосистема Эль-Гамаля основана на том, что проблема логарифмирования в конечном простом поле является сложной с вычислительной точки зрения. Однако, конечные поля являются не единственными алгебраическими структурами, в которых может быть поставлена задача вычисления дискретного логарифма. В 1985 году Коблиц и Миллер независимо друг от друга предложили использовать для построения криптосистем алгебраические структуры, определенные на множестве точек на эллиптических кривых. Мы рассмотрим случаи определения эллиптических кривых над простыми конечными полями произвольной характеристики и над полями Галуа характеристики 2.

***Определение* 3.1.** Пусть $p > 3$ – простое число. Пусть $a$, $b \in GF(p)$ такие, что $4a^2 + 27b^2 \neq 0$. Эллиптической кривой $E$ над полем $GF(p)$ называется множество решений $(x, y)$ уравнения

$$y^2 = x^3 + ax + b \qquad (3.1)$$

над полем $GF(p)$ вместе с дополнительной точкой $\infty$, называемой *точкой в бесконечности*.

Представление эллиптической кривой в виде уравнения (3.1) носит название *эллиптической кривой в форме Вейерштрасса.*

Обозначим количество точек на эллиптической кривой $E$ через $\#E$. Верхняя и нижняя границы для $\#E$ определяются теоремой Хассе:

$$p + 1 - 2\sqrt{p} \le \#E \le p + 1 + 2\sqrt{p} \ .$$

Зададим бинарную операцию на $E$ (в аддитивной записи) следующими правилами:

(i) $\infty + \infty = \infty$;

(ii) $\forall\ (x, y) \in E, (x, y) + \infty = (x, y)$;

(iii) $\forall\ (x, y) \in E, (x, y) + (x, -y) = \infty$;

(iv) $\forall\ (x_1, y_1) \in E, (x_2, y_2) \in E, x_1 \neq x_2, (x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, где

$$x_3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \text{ и}$$
$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

(v) $\forall\ (x_1, y_1) \in E, y_1 \neq 0, (x_1, y_1) + (x_1, y_1) = (x_2, y_2)$, где

$$x_2 = \lambda^2 - 2x_1,$$
$$y_2 = \lambda(x_1 - x_3) - y_1 \text{ и}$$
$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

Множество точек эллиптической кривой $E$ с заданной таким образом операцией образует абелеву группу.

Если $\#E = p + 1$, то кривая $E$ называется *суперсингулярной*.

Эллиптическая не являющаяся суперсингулярной кривая $E$ над полем $GF(2^m)$ характеристики 2 задается следующим образом.

***Определение* 3.2.** Пусть $m > 3$ – целое число. Пусть $a, b \in GF(2^m)$, $b \neq 0$. Эллиптической кривой $E$ над полем $GF(2^m)$ называется множество решений $(x, y)$ уравнения

$$y^2 + xy = x^3 + ax + b \tag{3.2}$$

над полем $GF(2^m)$ вместе с дополнительной точкой $\infty$, называемой *точкой в бесконечности*.

Количество точек на кривой $E$ также определяется теоремой Хассе:

$$q + 1 - 2\sqrt{q} \leq \#E \leq q + 1 + 2\sqrt{q},$$

где $q = 2^m$. Более того, $\#E$ четно.

Операция сложения на $E$ в этом случае задается следующими правилами:

(i) $\infty + \infty = \infty$;

(ii) $\forall (x, y) \in E, (x, y) + \infty = (x, y)$;

(iii) $\forall (x, y) \in E, (x, y) + (x, x + y) = \infty$;

(iv) $\forall (x_1, y_1) \in E, (x_2, y_2) \in E, x_1 \neq x_2, (x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, где

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a,$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \text{ и}$$
$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}.$$

(v) $\forall (x_1, y_1) \in E, x_1 \neq 0, (x_1, y_1) + (x_1, y_1) = (x_2, y_2)$, где

$$x_2 = \lambda^2 + \lambda + a,$$
$$y_2 = x_1^2 + (\lambda + 1)x_3 \text{ и}$$
$$\lambda = x_1 + \frac{y_1}{x_1}.$$

В этом случае множество точек эллиптической кривой $E$ с заданной таким образом операцией также образует абелеву группу.

Пользуясь операцией сложения точек на кривой, можно естественным образом определить операцию умножения точки $P \in E$ на произвольное целое число $n$:

$$nP = P + P + \ldots + P,$$

где операция сложения выполняется $n$ раз.

Теперь построим одностороннюю функцию, на основе которой можно будет создать криптографическую систему.

Пусть $E$ – эллиптическая кривая, $P \in E$ – точка на этой кривой. Выберем целое число $n < \#E$. Тогда в качестве прямой функции выберем произведение $nP$. Для его вычисления по оптимальному алгоритму потребуется не более $2 \cdot \log_2 n$ операций сложения. Обратную задачу сформулируем следующим образом: по заданным эллиптической кривой $E$, точке $P \in E$ и произведению $nP$ найти $n$. В настоящее время все известные алгоритмы решения этой задачи требуют экспоненциального времени.

Теперь мы можем описать криптографический протокол, аналогичный известному протоколу Диффи-Хеллмана. Для установления защищенной связи два пользователя $A$ и $B$ совместно выбирают эллиптическую кривую $E$ и точку $P$ на ней. Затем каждый из пользователей выбирает свое секретное целое число, соответственно $a$ и $b$. Пользователь $A$ вычисляет произведение $aP$, а пользователь $B$ – $bP$. Далее они обмениваются вычисленными значениями. При этом параметры самой кривой, координаты точки на ней и значения произведений являются открытыми и могут передаваться по незащищенным каналам связи. Затем пользователь $A$ умножает полученное значение на $a$, а пользователь $B$ умножает полученное им значение на $b$. В силу свойств операции умножения на число $a \cdot bP = = b \cdot aP$. Таким образом, оба пользователя получат общее секретное значение (координаты точки $abP$), которое они могут использовать для получения ключа шифрования. Отметим, что злоумышленнику для восстановления ключа потребуется решить сложную с вычислительной точки зрения задачу определения $a$ и $b$ по известным $E$, $P$, $aP$ и $bP$.

# Глава 4
# ЭЛЕКТРОННЫЕ ЦИФРОВЫЕ ПОДПИСИ

## 4.1. Постановка задачи

Передача сообщения *отправителем* (пользователь $A$) *получателю* (пользователь $B$) предполагает передачу данных, побуждающую пользователей к определенным действиям. Передача данных может представлять собой передачу фондов между банками, продажу акций или облигаций на автоматизированным рынке, а также передачу приказов (сигналов) по каналам электросвязи. Участники нуждаются в защите от множества злонамеренных действий, к которым относятся:

*отказ* – отправитель впоследствии отказывается от переданного сообщения;

*фальсификация* – получатель подделывает сообщение;

*изменение* – получатель вносит изменения в сообщение;

*маскировка* – пользователь маскируется под другого.

Для верификации (подтверждения) сообщения $M$ (пользователь $A$ – пользователю $B$) необходимо следующее:

- Отправитель (пользователь $A$) должен внести в $M$ подпись, содержащую дополнительную информацию, зависящую от $M$ и, в общем случае, от получателя сообщения и известной только отправителю закрытой информации $k_A$.
- Необходимо, чтобы правильную подпись $M$: $SIG\{k_A, M,$ идентификатор $B\}$ в сообщении для пользователя $B$ нельзя было составить без $k_A$.
- Для предупреждения повторного использования устаревших сообщений процедура составления подписи зависеть от времени.
- Пользователь $B$ должен иметь возможность удостовериться, $SIG\{k_A, M,$ идентификатор $B\}$ – есть правильная подпись $M$ пользователем $A$.

Рассмотрим эти пункты подробнее.

1. *Подпись сообщения* – определенный способ шифрования $M$ путем криптографического преобразования. Закрываемым элементом $k_A$ в преобразовании

$$<\text{Идентификатор } B, M> \rightarrow SIG\{k_A, M, \text{идентификатор } B\}$$

является ключ криптопреобразования.

Во всех практических криптографических системах $k_A$ принадлежит конечному множеству ключей $K$. Исчерпывающая проверка всех ключей, задаваемых соответствующими парами

$$<M, \text{идентификатор } B> \leftrightarrow SIG\{k_A, M, \text{идентификатор } B\}.$$

в общем должна привести к определению ключа $k_A$ злоумышленником. Если множество $K$ достаточно велико и ключ $k$ определен методом случайного выбора, то полная проверка ключей невозможна. Говоря, что составить правильную подпись без ключа невозможно, мы имеем в виду, что определение $SIG\{k_A, M,$ идентификатор $B\}$ без $k_A$ с вычислительной точки зрения эквивалентно поиску ключа.

2. Доступ к аппаратуре, программам и файлам системы обработки информации обычно контролируется *паролями*. Подпись – это вид пароля, зависящий от отправителя, получателя информации и содержания передаваемого сообщения.

3. Подпись должна меняться от сообщения к сообщению для предупреждения ее повторного использования с целью проверки нового сообщения. Цифровая подпись отличается от рукописной, которая обычно не зависит от времени составления и данных. Цифровая и рукописная подписи идентичны в том смысле, что они характерны только для данного владельца.

4. Хотя получатель информации не может составить правильную подпись, он должен уметь удостоверять ее подлинность. В обычных коммерческих сделках, таких, например, как продажа недвижимой собственности, эту функцию зачастую выполняет третье, независимое доверенное лицо (нотариус).

Установление подлинности подписи – это процесс, посредством которого каждая сторона устанавливает подлинность другой. Обязательным условием этого процесса является сохранение тайны. Во многих случаях нам приходится удостоверять свою личность, например, подписью или водительскими правами при получении денег по чеку либо фотографией в паспорте при пересечении границы. Для того чтобы в системе обработки данных получатель мог установить подлинность отправителя, необходимо выполнение следующих условий.

- Отправитель (пользователь $A$) должен обеспечить получателя (пользователя $B$) удостоверяющей информацией $AUTH\{k_A, M,$ идентификатор $B\}$, зависящей от секретной информации $k_A$, известной только пользователю $A$.
- Необходимо, чтобы удостоверяющую информацию $AUTH\{k_A,$ идентификатор $B\}$ от пользователя $A$ пользователю $B$ можно было дать только при наличии ключа $k_A$.
- Пользователь $B$ должен располагать процедурой проверки того, что $AUTH\{k_A,$ идентификатор $B\}$ действительно подтверждает личность пользователя $A$.
- Для предупреждения использования предыдущей проверенной на достоверность информации процесс установления подлинности должен иметь некоторую зависимость от времени.

Отметим, что установление подлинности и верификация передаваемого сообщения имеют сходные элементы: цифровая подпись является удостоверением подлинности информации с добавлением требования о ее зависимости от содержания передаваемого сообщения.

## 4.2. Алгоритмы электронной цифровой подписи

### *Цифровые подписи, основанные на асимметричных криптосистемах*

Для формирования системы ЭЦП можно использовать криптографическую систему Ривеста-Шамира-Эйделмана.

Пользователь *A* вырабатывает цифровую подпись предназначенного для пользователя *B* сообщения *M* с помощью следующего преобразования

$$SIG(M) = \mathrm{E}_{e_B,n_\mathrm{B}}(\mathrm{E}_{d_\mathrm{A},n_\mathrm{A}}(M)).$$

При этом он использует:

- свое секретное преобразование $\mathrm{E}_{d_\mathrm{A},n_\mathrm{A}}$;
- открытое преобразование $\mathrm{E}_{e_B,n_\mathrm{B}}$ пользователя *B*.

Затем он передает пользователю *B* пару <*M*, *SIG*(*M*)>.

Пользователь *B* может верифицировать это подписанное сообщение сначала при помощи своего секретного преобразования $\mathrm{E}_{d_B,n_\mathrm{B}}$ с целью получения

$$\mathrm{E}_{d_\mathrm{A},n_\mathrm{A}}(M) = \mathrm{E}_{d_B,n_\mathrm{B}}(SIG(M)) = \mathrm{E}_{d_B,n_\mathrm{B}}(\mathrm{E}_{e_B,n_\mathrm{B}}(\mathrm{E}_{d_\mathrm{A},n_\mathrm{A}}(M))).$$

и затем открытого $\mathrm{E}_{e_\mathrm{A},n_\mathrm{A}}$ пользователя *A* для получения сообщения *M*:

$$M = \mathrm{E}_{e_\mathrm{A},n_\mathrm{A}}(\mathrm{E}_{d_\mathrm{A},n_\mathrm{A}}(M)).$$

Затем пользователь *B* производит сравнение полученного сообщения *M* с тем, которое он получил в результате проверки цифровой подписи, и принимает решение о подлинности/подложности полученного сообщения.

В рассмотренном примере проверить подлинность ЭЦП может только пользователь *B*. Если же требуется обеспечение возможности верификации ЭЦП произвольным пользователем (например, при циркулярной рассылке документа), то алгоритм выработки ЭЦП упрощается, и подпись вырабатывается по формуле

$$SIG(M) = \mathrm{E}_{d_\mathrm{A},n_\mathrm{A}}(M),$$

а пользователи осуществляют верификацию с использованием открытого преобразования отправителя (пользователя *A*):

$$M = \mathrm{E}_{e_\mathrm{A},n_\mathrm{A}}(SIG(M)) = \mathrm{E}_{e_\mathrm{A},n_\mathrm{A}}(\mathrm{E}_{d_\mathrm{A},n_\mathrm{A}}(M)).$$

Вместо криптосистемы *RSA* для подписи сообщений можно использовать и любую другую асимметричную криптосистему.

Недостатком подобного подхода является то, что производительность асимметричной криптосистемы может оказаться недостаточной для удовлетворения предъявляемым требованиям. Возможным решением является применение специальной эффективно вычислимой функции, называемой *хэш-функцией* или *функцией хэширования.* Входом этой функции является сообщение, а выходом – слово фиксированной длины, много меньшей, чем длина исходного сообщения.

ЭЦП вырабатывается по той же схеме, но при этом используется не само сообщение, а значение хэш-функции от него. Это существенным образом ускорит выработку и верификацию ЭЦП. Требования, предъявляемые к функциям хэширования, а также примеры хэш-функций рассмотрены в п. 4.3.

Очень часто бывает желательно, чтобы электронная цифровая подпись была разной, даже если дважды подписывается одно и то же сообщение. Для этого в процесс выработки ЭЦП необходимо внести элемент "случайности". Способ сделать это был предложен Эль-Гамалем, аналогично тому, как это делается в системе шифрования, носящей его имя.

Выбирается большое простое число $p$ и целое число $g$, являющееся примитивным элементом в $Z_p$. Эти числа публикуются. Затем выбирается секретное число $x$ и вычисляется открытый ключ для проверки подписи $y = g^x \pmod p$.

Далее для подписи сообщения *M* вычисляется его хэш-функция $m = h(M)$. Выбирается случайное целое $k$: $1 < k < (p - 1)$, взаимно простое с $p - 1$, и вычисляется $r = g^k \pmod p$. После этого с помощью расширенного алгоритма Евклида решается относительно $s$ уравнение $m = xr + ks \pmod{p - 1}$. Подпись образует пара чисел $(r, s)$. После выработки подписи значение $k$ уничтожается.

Получатель подписанного сообщения вычисляет хэш-функцию сообщения $m = h(M)$ и проверяет выполнение равенства $y^r r^s \pmod p = = g^m$. Корректность этого уравнения очевидна:

$$y^r r^s = g^{x\cdot r}\cdot g^{k\cdot s} = g^{x\cdot r + k\cdot s} = g^m \ (mod\ p).$$

Еще одна подобная схема была предложена Шнорром. Как обычно, $p$ – большое простое число, $q$ – простой делитель ($p - 1$), $g$ – элемент порядка $q$ в $Z_p$, $k$ – случайное число, $x$ и $y = g^x \ (mod\ p)$ – секретный и открытый ключи соответственно. Уравнения выработки подписи выглядят следующим образом:

$$r = g^k \ (mod\ p);\ e = h(m, r);\ s = k + xe \ (mod\ q).$$

Подписью является пара ($r$, $s$). На приемной стороне вычисляется значение хэш-функции $e = h(m, r)$ и проверяется выполнение равенства $r = g^s y^{-e} (mod\ p)$, при этом действия с показателями степени производятся по модулю $q$.

## *Стандарт цифровой подписи* DSS

В США принят стандарт на выработку и верификацию цифровой подписи, называемый *DSS* (*Digit*al *Signatur*e *Standard*). Согласно этому стандарту, электронная цифровая подпись вырабатывается по следующей схеме:

1. Предварительный этап.

Выбираются числа *p, q и g*, такие, что $p$ – простое число длины $l$, где $l$ кратно 64 и $512 \le l \le 1024$; $q$ – простой делитель числа $p - 1$ длиной 160 бит; $g$ – элемент порядка $q$ в $Z_p$. Эти три числа являются открытыми данными.

Выбирается секретный ключ $x$, $1 \le x < q$, и вычисляется открытый ключ для проверки подписи $y = g^x (mod\ p)$.

2. Выработка электронной цифровой подписи.

Вычисляется значение хэш-функции от сообщения $h(m)$. При этом используется алгоритм безопасного хэширования *SHA* (*Secur*e *Hashing A*l*gorithm*), на который ссылается стандарт. Значение хэш-функции $h(m)$ имеет длину 160 бит.

Далее подписывающий выбирает случайное значение $k$, $1 \le k < q$, вычисляет $k^{-1} (mod\ q)$, и вырабатывает пару значений:

$$r = g^k \ (mod\ p)(mod\ q);$$
$$s = k^{-1} \ (h(m) + xr) \ (mod\ q)$$

Эта пара значений ($r$, $s$) и является электронной подписью под сообщением $M$. После выработки цифровой подписи значение $k$ уничтожается.

3. Верификация электронной цифровой подписи.

Пусть было принято сообщение $m_1$. Тогда уравнение проверки выглядит следующим образом:

$$r \equiv g^{h(m_1)\cdot s^{-1}} \cdot y^{r\cdot s^{-1}} (\mathrm{mod}\, p)(\mathrm{mod}\, q).$$

В самом деле:

$$g^{h(m)\cdot s^{-1}} \cdot y^{r\cdot s^{-1}} (\mathrm{mod}\, p)(\mathrm{mod}\, q) = g^{h(m)\cdot s^{-1}} \cdot g^{x\cdot r\cdot s^{-1}} (\mathrm{mod}\, p)(\mathrm{mod}\, q) =$$
$$= g^{s^{-1}(h(m)+xr)} (\mathrm{mod}\, p)(\mathrm{mod}\, q) = g^{(k^{-1}(h(m)+x\cdot r)^{-1}\cdot(h(m)+xr)} (\mathrm{mod}\, p)(\mathrm{mod}\, q) =$$
$$= g^{(k^{-1})^{-1}\cdot(h(m)+xr)^{-1}\cdot(h(m)+xr)} (\mathrm{mod}\, p)(\mathrm{mod}\, q) = g^k (\mathrm{mod}\, p)(\mathrm{mod}\, q) \equiv r.$$

## *Стандарт цифровой подписи ГОСТ Р 34.10-94*

Российский стандарт ЭЦП разрабатывался позже американского, поэтому параметры этого алгоритма выбраны с учетом возросших возможностей потенциального противника по вскрытию криптосистем. В частности, увеличена длина значения хэш-функции, что снижает вероятность столкновений, и, соответственно, порядок элемента-генератора, что делает более сложным решение задачи дискретного логарифма для восстановления секретного ключа. При описании алгоритма будут использоваться следующие обозначения:

$B^*$ – множество всех конечных слов в алфавите $B = \{0,1\}$.

$|A|$ – длина слова A

$V_k(2)$ – множество всех двоичных слов длины $k$.

$A||B$ – конкатенация слов A и $B$, также обозначается как AB.

$A^k$ – конкатенация $k$ экземпляров слова A.

$<N>_k$ – слово длины $k$, содержащее запись $N (mod\ 2^k)$, где $N$ – неотрицательное целое.

$\oplus$ – побитовое сложение слов по модулю 2.

[+] – сложение по правилу $A$ [+] $B = <A+B>_k$ ($k = |A| = |B|$).

$m$ – передаваемое сообщение.

$m_1$ – полученное сообщение

$h$ – хэш-функция, отображающая последовательность $m$ в слово $h(m) \in V_{256}(2)$.

$p$ – простое число, $2^{509} < p < 2^{512}$, либо $2^{1020} < p < 2^{1024}$.

$q$ – простое число, $2^{254} < q < 2^{256}$ и $q$ является делителем для $(p - 1)$.

$a$ – целое число, $1 < a < p - 1$, при этом $a^q (mod\ p) = 1$.

$k$ – целое число, $0 < k <$ q.

$x$ – секретный ключ пользователя для формирования подписи, $0 < x < q$.

$y$ – открытый ключ для проверки подписи, $y = a^x (mod\ p)$.

Система ЭЦП включает в себя процедуры выработки и проверки подписи под данным сообщением.

Цифровая подпись, состоящая из двух целых чисел, вычисляется с помощью определенного набора правил, изложенных в стандарте.

Числа $p$, $q$ и $a$, являющиеся параметрами системы, не являются секретными. Конкретный набор их значений может быть общим для группы пользователей. Целое число $k$, которое генерируется в процедуре подписи сообщения, должно быть секретным и должно быть уничтожено сразу после выработки подписи. Число $k$ снимается с физического датчика случайных чисел или вырабатывается псевдослучайным методом с использованием секретных параметров.

Процедура выработки подписи включает в себя следующие шаги:

1. Вычислить $h(m)$ – значение хэш-функции $h$ от сообщения $m$. Если $h(m)(mod\ q) = 0$, то присвоить $h(m)$ значение $0^{255}1$.
2. Выработать целое число $k$, $0 < k < q$.
3. Вычислить два значения: $r' = a^k (mod\ p)$ и $r = r'\ (mod\ q)$. Если $r = 0$, то перейти к шагу 2 и выработать другое значение числа $k$.
4. С использованием секретного ключа $x$ пользователя вычислить значение $s = (xr + kh(m))(mod\ q)$. Если $s = 0$, то перейти к шагу 2, в противном случае закончить работу алгоритма.

Заметим, что сообщение, дающее нулевое значение хэш-функции, не подписывается. В противном случае уравнение подписи упростилось бы до $s = xr\ (mod$ q) и злоумышленник легко мог бы вычислить секретный ключ $x$.

Проверка цифровой подписи возможна при наличии у получателя открытого ключа отправителя, пославшего сообщение.

Уравнение проверки будет следующим:

$$r \equiv (a^{s \cdot h(m_1)^{-1}} \cdot y^{-r \cdot h(m_1)^{-1}} (\mathrm{mod}\ p))(\mathrm{mod}\ q).$$

В самом деле,

$$\begin{aligned}(a^{s \cdot h(m)^{-1}} \cdot y^{-r \cdot h(m)^{-1}} (\mathrm{mod}\ p))(\mathrm{mod}\ q) &= (a^{s \cdot h(m)^{-1}} \cdot a^{-r \cdot x \cdot h(m)^{-1}} (\mathrm{mod}\ p))(\mathrm{mod}\ q) = \\ &= a^{h(m)^{-1} \cdot (s - x \cdot r)} (\mathrm{mod}\ p)(\mathrm{mod}\ q) = a^{h(m)^{-1}(x \cdot r + k \cdot h(m) - x \cdot r)} (\mathrm{mod}\ p)(\mathrm{mod}\ q) = \\ &= a^{k \cdot h(m)^{-1} \cdot h(m)} (\mathrm{mod}\ p)(\mathrm{mod}\ q) = a^k (\mathrm{mod}\ p)(\mathrm{mod}\ q) \equiv r.\end{aligned}$$

Вычисления по уравнению проверки реализуются следующим образом:

1. Проверить условия: $0 < s < q$ и $0 < r < q$. Если хотя бы одно из этих условий не выполнено, то подпись считается недействительной.
2. Вычислить $h(m_1)$ – значение хэш-функции $h$ от полученного сообщения $m_1$.
   Если $h(m_1)(mod\ q) = 0$, присвоить $h(m_1)$ значение $0^{255}1$.
3. Вычислить значение $v = (h(m_1))^{q-2} (mod\ q)$, что является не чем иным, как мультипликативным обратным к $h(m_1)$ $(mod\ q)$. Вообще говоря, алгоритм проверки можно несколько ускорить, если вычислять $h(m_1)^{-1} (mod\ q)$ с помощью расширенного алгоритма Евклида, а не путем возведения в степень.
4. Вычислить значения:
   $z_1 = sv (mod\ q)$ и
   $z_2 = (q - r)v (mod\ q)$
5. Вычислить значение
   $u = (a^{z_1} y^{z_2} (\mathrm{mod}\ p))(\mathrm{mod}\ q)$
6. Проверить условие
   $r = u$

При совпадении значений $r$ и $u$ получатель принимает решение о том, что полученное сообщение подписано данным отправителем и в процессе передачи не нарушена целостность сообщения, т.е. $m_1 = m$. В противном случае подпись считается недействительной.

## *Цифровые подписи, основанные на симметричных криптосистемах*

На первый взгляд, сама эта идея может показаться абсурдом. Действительно, общеизвестно, что так называемая «современная», она же двухключевая криптография возникла и стала быстро развиваться в последние десятилетия именно потому, что ряд новых криптографических протоколов типа протокола цифровой подписи не удалось эффективно реализовать на базе традиционных криптографических алгоритмов, широко известных и хорошо изученных к тому времени. Тем не менее, это возможно. И первыми, кто обратил на это внимание, были родоначальники криптографии с открытым ключом У. Диффи и М. Хеллман, опубликовавшие описание подхода, позволяющего выполнять процедуру цифровой подписи

одного бита с помощью блочного шифра. Прежде чем изложить эту идею, сделаем несколько замечаний о сути и реализациях цифровой подписи.

Стойкость какой-либо схемы подписи (т.е. выполнение требований, описанных в п. 4.1.) доказывается обычно установлением равносильности соответствующей задачи вскрытия схемы какой-либо другой, о которой известно, что она вычислительно неразрешима. Практически все современные алгоритмы ЭЦП основаны на так называемых «сложных математических задачах» типа факторизации больших чисел или логарифмирования в дискретных полях.

Однако, доказательство невозможности эффективного вычислительного решения этих задач отсутствует, и нет никаких гарантий, что они не будут решены в ближайшем будущем, а соответствующие схемы взломаны – как это произошло с «ранцевой» схемой цифровой подписи. Более того, с бурным прогрессом средств вычислительных техники «границы надежности» методов отодвигаются в область все больших размеров блока.

Всего пару десятилетий назад, на заре криптографии с открытым ключом считалось, что для реализации схемы подписи *RSA* достаточно даже 128-битовых чисел. Сейчас эта граница отодвинута до 1024-битовых чисел – практически на порядок, – и это далеко еще не предел. Это приводит к необходимости переписывать реализующие схему программы, и зачастую перепроектировать аппаратуру.

Ничего подобного не наблюдается в области классических блочных шифров, если не считать изначально ущербного и непонятного решения комитета по стандартам США ограничить размер ключа алгоритма *DES* 56-ю битами, тогда как еще во время обсуждения алгоритма предлагалось использовать ключ большего размера. Схемы подписи, основанные на классических блочных шифрах, свободны от указанных недостатков:

- во-первых, их стойкость к попыткам взлома вытекает из стойкости использованного блочного шифра, поскольку классические методы шифрования изучены гораздо больше, а их надежность обоснована намного лучше, чем надежность асимметричных криптографических систем;
- во-вторых, даже если стойкость использованного в схеме подписи шифра окажется недостаточной в свете прогресса вычислительной техники, его легко можно будет заменить на другой, более устойчивый, с тем же размером блока данных и ключа, без необходимости менять основные характеристики всей схемы – это потребует только минимальной модификации программного обеспечения;

Итак, вернемся к схеме Диффи и Хеллмана подписи одного бита сообщения с помощью алгоритма, базирующегося на любом классическом блочном шифре. Предположим, в нашем распоряжении есть алгоритм зашифрования $E_K$, оперирующий блоками данных $X$ размера $n$ и использующий ключ размером $n_K$: $|X| = n$, $|K| = n_K$. Структура ключевой информации в схеме следующая: секретный ключ подписи $k_S$ *выбирается* как произвольная (случайная) пара ключей $k_0$, $k_1$ используемого блочного шифра:

$$k_S = (k_0, k_1);$$

Таким образом, размер ключа подписи равен удвоенному размеру ключа использованного блочного шифра:

$$|K_S| = 2|K| = 2n_K.$$

Ключ проверки представляет собой результат шифрования двух блоков текста $X_0$ и $X_1$ с ключами $k_0$ и $k_1$ соответственно:

$$k_V = (C_0, C_1) = (E_{k_0}(X_0), E_{k_1}(X_1))$$

где являющиеся параметром схемы блоки данных несекретны и известны проверяющей подпись стороне. Таким образом, размер ключа проверки подписи равен удвоенному размеру блока использованного блочного шифра:

$$|k_V| = 2|X| = 2n.$$

Алгоритм *Sig* выработки цифровой подписи для бита $t$ ($t \in \{0,1\}$) заключается просто в выборе соответствующей половины из пары, составляющей секретный ключ подписи:

$$s = S(t) = k_t.$$

Алгоритм *Ver* проверки подписи состоит в проверке уравнения $E_{k_t}(X_t) = C_t$, которое, очевидно, должно выполняться для нашего $t$. Получателю известны все используемые при этом величины.

Таким образом, функция проверки подписи будет следующей:

$$V(t, s, K_C) = \begin{cases} 1, & E_S(X_t) = C_t \\ 0, & E_S(X_t) \neq C_t \end{cases}.$$

Покажем, что данная схема работоспособна, для чего проверим выполнение необходимых свойств схемы цифровой подписи:

1. Невозможность подписать бит $t$, если неизвестен ключ подписи. Действительно, для выполнения этого злоумышленнику потребовалось бы решить уравнение $E_S(X_t) = C_t$ относительно $s$, что эквивалентно

определению ключа для известных блоков шифрованного и соответствующего ему открытого текста, что вычислительно невозможно в силу использования стойкого шифра.

2. Невозможность подобрать другое значение бита $t$, которое подходило бы под заданную подпись, очевидна: число возможных значений бита всего два и вероятность выполнения двух следующих условий одновременно пренебрежимо мала в просто в силу использования криптостойкого алгоритма:

$$\boldsymbol{E}_S(\boldsymbol{X}_0) = \boldsymbol{C}_0,$$
$$\boldsymbol{E}_S(\boldsymbol{X}_1) = \boldsymbol{C}_1.$$

Таким образом, предложенная Диффи и Хеллманом схема цифровой подписи на основе классического блочного шифра обладает такой же стойкостью, что и лежащий в ее основе блочный шифр, и при этом весьма проста. Однако, у нее есть два существенных недостатка.

Первый недостаток заключается в том, что данная схема позволяет подписать лишь один бит информации. В блоке большего размера придется отдельно подписывать каждый бит, поэтому даже с учетом хэширования сообщения все компоненты подписи – секретный ключ, проверочная комбинация и собственно подпись получаются довольно большими по размеру и более чем на два порядка превосходят размер подписываемого блока. Предположим, что в схеме используется криптографический алгоритм $E_K$ с размером блока и ключа, соответственно $n$ и $n_K$. Предположим также, что используется функция хэширования с размером выходного блока $n_H$. Тогда размеры основных рабочих блоков будут следующими:

размер ключа подписи: $n_{kS} = 2n_H \cdot n_K$.

размер ключа проверки подписи: $n_C = 2n_H n$.

размер подписи: $n_S = n_H \cdot n_K$.

Если, например, в качестве основы в данной схеме будет использован шифр ГОСТ 28147–89 с размером блока $n = 64$ бита и размером ключа $n_K = 256$ бит, и для выработки хэш–блоков будет использован тот же самый шифр в режиме выработки имитовставки, что даст размер хэш–блока $n_H = 64$ то размеры рабочих блоков будут следующими:

размер ключа подписи: $n_{kS} = 2n_H \cdot n_K = 2 \cdot 64 \cdot 256$ бит= 4096 байт;

размер ключа проверки подписи: $n_C = 2n_H n = 2 \cdot 64 \cdot 64$ бит = 1024 байта.

размер подписи: $n_S = n_H \cdot n_K = 64 \cdot 256$ бит = 2048 байт.

Второй недостаток данной схемы, быть может, менее заметен, но столь же серьезен. Дело в том, что пара ключей выработки подписи и проверки подписи могут быть использованы только один раз. Действительно, выполнение процедуры подписи бита сообщения приводит к раскрытию половины секретного ключа, после чего он уже не является полностью секретным и не может быть использован повторно. Поэтому для каждого подписываемого сообщения необходим свой комплект ключей подписи и проверки. Это практически исключает возможность использования рассмотренной схемы Диффи–Хеллмана в первоначально предложенном варианте в реальных системах ЭЦП.

Однако, несколько лет назад Березин и Дорошкевич предложили модификацию схемы Диффи–Хеллмана, фактически устраняющую ее недостатки.

Центральным в этом подходе является алгоритм «односторонней криптографической прокрутки», который в некотором роде может служить аналогом операции возведения в степень. Как обычно, предположим, что в нашем распоряжении имеется криптографический алгоритм $E_K$ с размером блока данных и ключа соответственно $n$ и $n_K$ бит, причем $n \le n_K$.

Пусть в нашем распоряжении также имеется некоторая функция отображения $\boldsymbol{n}$–битовых блоков данных в $\boldsymbol{n_K}$–битовые $Y = P_{n \to n_K}(X)$, $|X| = n$, $|Y| = n_K$. Определим рекурсивную функцию $\boldsymbol{R_k}$ «односторонней прокрутки» блока данных $T$ размером $n$ бит $k$ раз ($k \ge 0$) при помощи следующей формулы:

$$\boldsymbol{R_k(T)} = \begin{cases} \boldsymbol{T}, \boldsymbol{k} = 0, \\ \boldsymbol{E}_{\boldsymbol{P_{n \to n_K}(R_{k-1}(T))}}(\boldsymbol{X}), \boldsymbol{k} > 0, \end{cases}$$

где $X$ – произвольный несекретный $n$-битовый блок данных, являющийся параметром процедуры прокрутки.

По своей идее функция односторонней прокрутки чрезвычайно проста, надо всего лишь нужное количество раз ($k$) выполнить следующие действия: расширить $n$-битовый блок данных $T$ до размера ключа использованного алгоритма шифрования ($n_K$), на полученном расширенном блоке как на ключе зашифровать блок данных $X$, результат зашифрования занести на место исходного блока данных ($T$). По определению операция $R_k(T)$ обладает двумя важными для нас свойствами:

1. Аддитивность и коммутативность по числу прокручиваний:

$$R_{k+k'}(T) = R_{k'}(R_k(T)) = R_k(R_{k'}(T)).$$

2. Односторонность или необратимость прокрутки: если известно только некоторое значение функции $R_k(T)$, то вычислительно невозможно найти значение $R_{k'}(T)$ для любого $k' < k$ – если бы это было возможно, в нашем распоряжении был бы способ определить ключ шифрования по известному входному и выходному блоку алгоритма $E_K$. что противоречит предположению о стойкости шифра.

Теперь покажем, как указанную операцию можно использовать для подписи блока $T$, состоящего из $n_T$ битов.

Секретный ключ подписи $k_S$ выбирается как произвольная пара блоков $k_0$, $k_1$, имеющих размер блока данных используемого блочного шифра, т.е. размер ключа выработки подписи равен удвоенному размеру блока данных использованного блочного шифра: $|k_S| = 2n$;

Ключ проверки подписи вычисляется как пара блоков, имеющих размер блоков данных использованного алгоритма по следующим формулам:

$$k_C = (C_0,C_1) = (R_2{}^{n_T-1}(K_0), R_2{}^{n_T-1}(K_1)).$$

В этих вычислениях также используются несекретные блоки данных $X_0$ и $X_1$, являющиеся параметрами функции «односторонней прокрутки», они обязательно должны быть различными. Таким образом, размер ключа проверки подписи также равен удвоенному размеру блока данных использованного блочного шифра: $|k_C| = 2n$.

Вычисление и проверка ЭЦП будут выглядеть следующим образом:

Алгоритм $Sig_{n_T}$ выработки цифровой подписи для $n_T$-битового блока $T$ заключается в выполнении «односторонней прокрутки» обеих половин ключа подписи $T$ и $2^{n_T}-1-T$ раз соответственно:

$$s = Sig_{n_T}(T) = (s_0,s_1) = R_T(k_0), R_{2^{nT}-1-T}(k_1)) .$$

Алгоритм $Ver_{n_T}$ проверки подписи состоит в проверке истинности соотношений $R_{2^{nT}-1-T}(s_0) = C_0, R_T(s_1) = C_1$, которые, очевидно, должны выполняться для подлинного блока данных $\boldsymbol{T}$:

$$R_2{}^{n_T-1-T}(s_0) = R_2{}^{n_T-1-T}(R_T(k_0)) = R_2{}^{n_T-1-T+T}(k_0) = R_2{}^{n_T-1}(k_0) = C_0,$$

$$R_T(s_1) = R_T(R_2{}^{n_T-1-T}(k_1)) = R_{T+2}{}^{n_T-1-T}(k_1) = R_2{}^{n_T-1}(k_1) = C_1.$$

Таким образом, функция проверки подписи будет следующей:

$$\boldsymbol{V(T,s,K_C)} = \begin{cases} 1, \boldsymbol{R_{2^{nT}-1-T}(s_0) = C_0} \ \ \& \ \ \boldsymbol{R_T(s_1) = C_1}, \\ 0, \boldsymbol{R_{2^{nT}-1-T}(s_0) \neq C_0} \ \ | \ \ \boldsymbol{R_T(s_1) \neq C_1}. \end{cases}$$

Покажем, что для данной схемы выполняются необходимые условия работоспособности схемы подписи:

Предположим, что в распоряжении злоумышленника есть $n_T$-битовый блок $T$, его подпись $s = (s_0,s_1)$, и ключ проверки $k_C = (C_0,C_1)$. Пользуясь этой информацией, злоумышленник пытается найти правильную подпись $s' = (s'_0,s'_1)$ для другого $n_T$-битового блока $T'$. Для этого ему надо решить следующие уравнения относительно $s'_0$ и $s'_1$:

$$R_2{}^{n_T-1-T'}(s'_0) = C_0,$$
$$R_{T'}(s'_1) = C_1.$$

В распоряжении злоумышленника есть блок данных $T$ с подписью $s = (s_0,s_1)$, что позволяет ему вычислить одно из значений $s'_0,s'_1$, даже не владея ключом подписи:

(a) если $T < T'$, то $s'_0 = R_{T'}(k_0) = R_{T'-T}(R_T(k_0)) = R_{T'-T}(s_0)$,

(b) если $T > T'$, то $s'_1 = R_2{}^{n_T-1-T'}(k_1) = R_{T-T'}(R_2{}^{n_T-1-T}(k_1)) = R_{T-T'}(s_1)$.

Однако для нахождения второй половины подписи ($\boldsymbol{s'_1}$ и $\boldsymbol{s'_0}$ в случаях (a) и (b) соответственно) ему необходимо выполнить прокрутку в обратную сторону, т.е. найти $R_k(X)$, располагая только значением для большего $k$, что является вычислительно невозможным. Таким образом, злоумышленник не может подделать подпись под сообщением, если не располагает секретным ключом подписи.

Второе требование также выполняется: вероятность подобрать блок данных $T'$, отличный от блока $T$, но обладающий такой же цифровой подписью, чрезвычайно мала и может не приниматься во внимание. Действительно, пусть цифровая подпись блоков $T$ и $T'$ совпадает. Тогда подписи обоих блоков будут равны соответственно:

$$s = S_{n_T}(T) = (s_0,s_1) = (R_T(k_0),\ R_2{}^{n_T-1-T}(k_1)),$$

$$s' = S_{n_T}(T') = (s'_0,s'_1) = (R_{T'}(k_0),\ R_2{}^{n_T-1-T'}(k_1)),$$

но $s=s'$, следовательно:

$$R_T(k_0) = R_{T'}(k_0) \text{ и } R_2{}^{n_T-1-T}(k_1) = R_2{}^{n_T-1-T'}(k_1).$$

Положим для определенности $T \leq T'$, тогда справедливо следующее:

$$R_{T'-T}(k^*_0) = k^*_0,\ R_{T'-T}(k^*_1) = k^*_1, \text{ где } k^*_0 = R_T(k_0),\ k^*_1 = R_2{}^{n_T-1-T'}(k_1)$$

Последнее условие означает, что прокручивание двух различных блоков данных одно и то же число раз оставляет их значения неизменными. Вероятность такого события чрезвычайно мала и может не приниматься во внимание.

Таким образом рассмотренная модификация схемы Диффи–Хеллмана делает возможным подпись не одного бита, а целой битовой группы. Это позволяет в несколько раз уменьшить размер подписи и ключей подписи/проверки данной схемы. Однако надо понимать, что увеличение размера подписываемых битовых групп приводит к экспоненциальному росту объема необходимых вычислений и начиная с некоторого значения делает работу схемы также неэффективной. Граница «разумного размера» подписываемой группы находится где-то около десяти бит, и блоки большего размера все равно необходимо подписывать «по частям».

Теперь найдем размеры ключей и подписи, а также объем необходимых для реализации схемы вычислений. Пусть размер хэш–блока и блока используемого шифра одинаковы и равны $n$, а размер подписываемых битовых групп равен $n_T$. Предположим также, что если последняя группа содержит меньшее число битов, обрабатывается она все равно как полная $n_T$-битовая группа. Тогда размеры ключей подписи/проверки и самой подписи совпадают и равны следующей величине:

$$|K_S| = |K_C| = |s| = 2n\left\lceil \frac{n}{n_T} \right\rceil \approx 2\frac{n^2}{n_T} \text{ бит},$$

где $\lceil x \rceil$ обозначает округление числа $x$ до ближайшего целого в сторону возрастания. Число операций шифрования $E_K(X)$, требуемое для реализации процедур схемы, определяются нижеследующими соотношениями:

при выработке ключевой информации оно равно:

$$W_K = 2 \cdot (2^{n_T} - 1)\left\lceil \frac{n}{n_T} \right\rceil \approx \frac{2^{n_T+1} n}{n_T},$$

при выработке и проверке подписи оно вдвое меньше:

$$W_S = W_C = (2^{n_T} - 1)\left\lceil \frac{n}{n_T} \right\rceil \approx \frac{2^{n_T} n}{n_T}.$$

Размер ключа подписи и проверки подписи можно дополнительно уменьшить следующими приемами:

1. Нет необходимости хранить ключи подписи отдельных битовых групп, их можно динамически вырабатывать в нужный момент времени с помощью генератора криптостойкой гаммы. Ключом подписи в этом случае будет являться обычный ключ использованного в схеме подписи блочного шифра. Например, если схема подписи будет построена на алгоритме ГОСТ 28147-89, то размер ключа подписи будет равен 256 битам.
2. Аналогично, нет необходимости хранить массив ключей проверки подписи отдельных битовых групп блока, достаточно хранить его значение хэш-функции этого массива. При этом алгоритм выработки ключа подписи и алгоритм проверки подписи будут дополнены еще одним шагом – вычислением хэш-функции массива проверочных комбинаций отдельных битовых групп.

Таким образом, проблема размера ключей и подписи решена, однако, второй недостаток схемы – одноразовость ключей – не преодолен, поскольку это невозможно в рамках подхода Диффи–Хеллмана.

Для практического использования такой схемы, рассчитанной на подпись $N$ сообщений, отправителю необходимо хранить $N$ ключей подписи, а получателю – $N$ ключей проверки, что достаточно неудобно. Эта проблема может быть решена в точности так же, как была решена проблема ключей для множественных битовых групп – генерацией ключей подписи для всех $N$ сообщений из одного мастер-ключа и свертывание всех проверочных комбинаций в одну контрольную комбинацию с помощью алгоритма вычисления хэш-функции.

Такой подход решил бы проблему размера хранимых ключей, но привел бы к необходимости вместе подписью каждого сообщения высылать недостающие $N$–1 проверочных комбинаций, необходимых для вычисления хэш-функции массива всех контрольных комбинаций отдельных сообщений. Ясно, что такой вариант не обладает преимуществами по сравнению с исходным.

Упомянутыми выше авторами был предложен механизм, позволяющий значительно снизить остроту проблемы. Его основная идея – вычислять контрольную комбинацию (ключ проверки подписи) не как хэш-функцию от линейного массива проверочных комбинаций всех сообщений, а попарно – с помощью бинарного дерева. На каждом уровне проверочная комбинация вычисляется как хэш-функция от конкатенации двух проверочных комбинаций младшего уровня. Чем выше уровень комбинации, тем больше отдельных ключей проверки "учитывается" в ней.

Предположим, что наша схема рассчитана на $2^L$ сообщений. Обозначим через $C_i^{(l)}$ $i$-тую комбинацию $l$-того уровня. Если нумерацию комбинаций и уровней начинать с нуля, то справедливо следующее условие: $0 \le i < 2^{L-l}$, а $i$-ая проверочная комбинация $l$-того уровня рассчитана на $2^l$ сообщений с номерами от $i \cdot 2^l$ до $(i+1) \cdot 2^l - 1$ включительно. Число комбинаций нижнего, нулевого уровня равно $2^L$, а самого верхнего, $L$-того уровня – одна, она и является контрольной комбинацией всех $2^L$ сообщений, на которые рассчитана схема.

На каждом уровне, начиная с первого, проверочные комбинации рассчитываются по следующей формуле:

$$C_i^{(l+1)} = H(C_{2i}^{(l)} \parallel C_{2i+1}^{(l)}),$$

где через $A \parallel B$ обозначен результат конкатенации двух блоков данных $A$ и $B$, а через $H(X)$ – процедура вычисления хэш-функции блока данных $X$.

При использовании указанного подхода вместе с подписью сообщения необходимо передать не $N$–1, как в исходном варианте, а только $\log_2 N$ контрольных комбинаций. Передаваться должны комбинации, соответствующие смежным ветвям дерева на пути от конечной вершины, соответствующей номеру использованной подписи, к корню.

Пример организации проверочных комбинаций в виде двоичного дерева в схеме на восемь сообщений приведена на рисунке 4.1. Так, при передаче сообщения № 5 (контрольная комбинация выделена рамкой) вместе с его подписью должны быть переданы контрольная комбинация сообщения № 4 ($C_4^{(0)}$), общая для сообщений №№ 6–7 ($C_3^{(1)}$) и общая для сообщений №№ 0–3 ($C_0^{(2)}$), все они выделены на рисунке другим фоном.

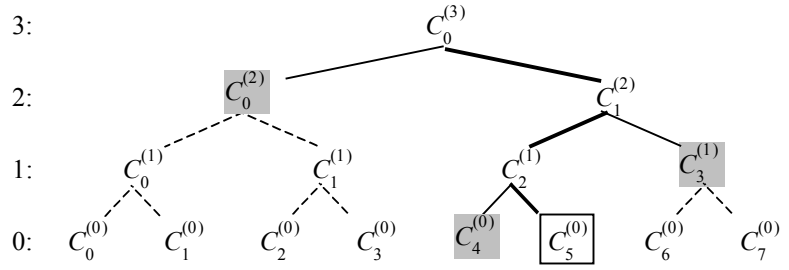



Рис. 4.1. Двоичное дерево для схемы ЭЦП на 8 сообщений

При проверке подписи значение $C_5^{(0)}$ будет вычислено из сообщения и его подписи, а итоговая контрольная комбинация, подлежащая сравнению с эталонной, по следующей формуле:

$$C = C_0^{(3)} = H(C_0^{(2)} \parallel H(H(C_4^{(0)} \parallel C_5^{(0)}) \parallel C_3^{(1)})).$$

Необходимость отправлять вместе с подписью сообщения дополнительную информацию, нужную для проверки подписи, на самом деле не очень обременительна. Действительно, в системе на $1024=2^{10}$ подписей вместе с сообщением и его подписью необходимо дополнительно передавать 10 контрольных комбинаций, а в системе на $1048576 = 2^{20}$ подписей – всего 20 комбинаций. Однако, при большом числе подписей, на которые рассчитана система, возникает другая проблема – хранение дополнительных комбинаций, если они рассчитаны предварительно, или их выработка в момент формирования подписи.

Дополнительные контрольные комбинации, которые передаются вместе с подписью и используются при ее проверке, вырабатываются при формировании ключа проверки по ключу подписи и могут храниться в системе и использоваться в момент формирования подписи, либо вычисляться заново в этот момент.

Первый подход предполагает затраты дисковой памяти, так как необходимо хранить $2^{L+1}$–2 значений хэш-функции всех уровней, а второй требует большого объема вычислений в момент формирования подписи. Можно использовать и компромиссный подход – хранить все хэш-комбинации начиная с некоторого уровня $l^*$, а комбинации меньшего уровня вычислять при формировании подписи.

В рассмотренной выше схеме подписи на 8 сообщений можно хранить все 14 контрольных комбинаций, используемых при проверки (всего их 15, но самая верхняя не используется), тогда при проверке подписи их не надо будет вычислять заново. Можно хранить 6 комбинаций начиная с уровня 1 ($C_0^{(1)}$, $C_1^{(1)}$, $C_2^{(1)}$, $C_3^{(1)}$, $C_0^{(2)}$, $C_1^{(2)}$), тогда при проверке подписи сообщения № 5 необходимо будет заново вычислить комбинацию $C_4^{(0)}$, а остальные ($C_0^{(2)}$,$C_3^{(1)}$) взять из таблицы, и т.д.. Указанный подход позволяет достичь компромисса между быстродействием и требованиям к занимаемому количеству дискового пространства.

Отметим, что отказ от хранения комбинаций одного уровня приводит к экономии памяти и росту вычислительных затрат примерно вдвое, то есть зависимость носит экспоненциальный характер.

## 4.3. Функции хэширования

Функция хэширования $H$ представляет собой отображение, на вход которого подается сообщение переменной длины $M$, а выходом является строка фиксированной длины $H(M)$. В общем случае $H(M)$ будет гораздо меньшим, чем $M$, например, $H(M)$ может быть 128 или 256 бит, тогда как $M$ может быть размером в мегабайт или более.

Функция хэширования может служить для обнаружения модификации сообщения. То есть, она может служить в качестве криптографической контрольной суммы (также называемой кодом обнаружения изменений или кодом аутентификации сообщения).

Теоретически возможно, что два различных сообщения могут быть сжаты в одну и ту же свертку (так называемая ситуация "столкновения"). Поэтому для обеспечения стойкости функции хэширования необходимо предусмотреть способ избегать столкновений. Полностью столкновений избежать нельзя, поскольку в общем случае количество возможных сообщений превышает количество возможных выходных значений функции хэширования. Однако, вероятность столкновения должна быть низкой.

Для того, чтобы функция хэширования могла должным образом быть использована в процессе аутентификации, функция хэширования $H$ должна обладать следующими свойствами:

1. $H$ может быть применена к аргументу любого размера;
2. Выходное значение $H$ имеет фиксированный размер;
3. $H(x)$ достаточно просто вычислить для любого $x$. Скорость вычисления хэш-функции должна быть такой, чтобы скорость выработки и проверки ЭЦП при использовании хэш-функции была значительно больше, чем при использовании самого сообщения;
4. Для любого $y$ с вычислительной точки зрения невозможно найти $x$, такое что $H(x)=y$.
5. Для любого фиксированного $x$ с вычислительной точки зрения невозможно найти $x' \neq x$, такое что $H(x')=H(x)$.

Свойство 5 гарантирует, что не может быть найдено другое сообщение, дающее ту же свертку. Это предотвращает подделку и также позволяет использовать $H$ в качестве криптографической контрольной суммы для проверки целостности.

Свойство 4 эквивалентно тому, что $H$ является односторонней функцией. Стойкость систем с открытыми ключами зависит от того, что открытое криптопреобразование является односторонней функцией-ловушкой. Напротив, функции хэширования являются односторонними функциями, не имеющими ловушек.

## *Функция хэширования* SHA

Алгоритм безопасного хэширования *SHA* (*Secure Hash Algorithm*) принят в качестве стандарта США в 1992 г. и предназначен для использования совместно с алгоритмом цифровой подписи, определенным в стандарте *DSS*. При вводе сообщения $M$ алгоритм вырабатывает 160-битовое выходное сообщение, называемое сверткой (*Message Digest*), которая и используется при выработке ЭЦП.

Рассмотрим работу алгоритма подробнее.

Прежде всего исходное сообщение дополняется так, чтобы его длина стала кратной 512 битам. При этом сообщение дополняется даже тогда, когда его длина уже кратна указанной. Процесс происходит следующим образом: добавляется единица, затем столько нулей, сколько необходимо для получения сообщения, длина которого на 64 бита меньше, чем кратная 512, и затем добавляется 64-битовое представление длины исходного сообщения.

Далее инициализируются пять 32-битовых переменных следующими шестнадцатеричными константами:

$A$ = 67452301

$B$ = *EFCDAB*89

$C$ = 98*BADCFE*

$D$ = 10325476

$E$ = *C*3*D*2*E*1*F*0,

Далее эти пять переменных копируются в новые переменные a, b, c, $d$ и e соответственно.

Главный цикл может быть довольно просто описан на псевдокоде следующим образом:

*for* ($t$ = 0; $t$ < 80; $t$++){

    *temp* = (a <<< 5)+$f_t$(b,c,$d$) + e + $W_t$ + $K_t$;

    e = $d$; $d$ = c; c = b <<< 30; b = a; a = *temp*;

},

где

<<< - операция циклического сдвига влево;

$K_t$ – шестнадцатеричные константы, определяемые по следующим формулам:

$$\mathbf{K}_t = \begin{cases} \mathbf{5A827999}, & t = 0..19 \\ \mathbf{6ED9EBA1}, & t = 20..39 \\ \mathbf{8F1BBCDC}, & t = 40..59 \\ \mathbf{CA62C1D6}, & t = 60..79 \end{cases},$$

функции $f_t(x, y, z)$ задаются следующими выражениями:

$$\mathbf{f(x,y,z)_t} = \begin{cases} \mathbf{X} \wedge \mathbf{Y} \vee \neg\mathbf{X} \wedge \mathbf{Z}, & \mathbf{t} = 0..19 \\ \mathbf{X} \oplus \mathbf{Y} \oplus \mathbf{Z}, & \mathbf{t} = 20..39, 60..79 \\ \mathbf{X} \wedge \mathbf{Y} \vee \mathbf{X} \wedge \mathbf{Z} \vee \mathbf{Y} \wedge \mathbf{Z}, & \mathbf{t} = 40..59 \end{cases},$$

значения $W_t$ получаются из 32-битовых подблоков 512-битового блока расширенного сообщения по следующему правилу:

$$W_t = \begin{cases} M_t, & t = 0..19 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1, & t = 16..79 \end{cases}.$$

После окончания главного цикла значения a, b, c, *d* и e складываются с содержимым *A*, *B*, *C*, *D* и *E* соответственно и осуществляется переход к обработке следующего 512-битового блока расширенного сообщения. Выходное значение хэш-функции является конкатенацией значений *A*, *B*, *C*, *D* и *E*.

## *Функция хэширования ГОСТ Р 34.11-94*

При описании функции хэширования будут использоваться те же обозначения, что использовались при описании алгоритма выработки цифровой подписи согласно ГОСТ Р 34.10, и, кроме того, пусть

*M* – последовательность двоичных символов, подлежащих хэшированию.

*h* – хэш-функция, отображающая последовательность *M* в слово $h(M) \in V_{256}(2)$.

$E_K(A)$ – результат шифрования слова A на ключе *K* с использованием алгоритма шифрования по ГОСТ 28147 в режиме простой замены.

*H* – стартовый вектор хэширования.

**Общие положения**

Под хэш-функцией *h* понимается отображение

$$h: B^* \to V_{256}(2).$$

Для определения хэш-функции необходимы:

- алгоритм вычисления шаговой функции хэширования $\kappa$, где

$$\kappa: V_{256}(2) \times V_{256}(2) \to V_{256}(2);$$

- описание итеративной процедуры вычисления значения хэш-функции *h*.

Алгоритм вычисления шаговой функции хэширования состоит из трех частей:

- генерации четырех 256-битных ключей;
- шифрующего преобразования – шифрования 64-битных подслов слова *H* на ключах $K_i$ ($i$ = 1, 2, 3, 4) с использованием алгоритма ГОСТ 28147 в режиме простой замены;
- перемешивающего преобразования результата шифрования.

**Генерация ключей.**

Рассмотрим $X = (b_{256}, b_{255}, \ldots, b_1) \in V_{256}(2)$.

Пусть $X = x_4 \| x_3 \| x_2 \| x_1 = \eta_{16} \| \eta_{15} \| \ldots \| \eta_1 = \xi_{32} \| \xi_{31} \| \ldots \| \xi_1$,

где $x_i \in V_{64}(2)$, $i = 1..4$; $\eta_j \in V_{16}(2)$, $j = 1..16$; $\xi_k \in V_8(2)$, $k = 1..32$.

Обозначим $A(X) = (x_1 \oplus x_2) \| x_4 \| x_3 \| x_2$.

Задается преобразование $P$: $V_{256}(2) \to V_{256}(2)$ слова $\xi_{32} \| \ldots \| \xi_1$ в слово $\xi_{\varphi(32)} \| \xi_{(\varphi)31} \| \ldots \| \xi_{\varphi(1)}$, где $\varphi(i+1+4(k-1)) = 8i + k$, $i = 0..3$, $k = 1..8$.

Для генерации ключей необходимо использовать следующие исходные данные:

- слова $H, M \in V_{256}(2)$;
- константы: слова $C_i$ ($i$ = 2,3,4), имеющие значения $C_2 = C_4 = 0^{256}$ и $C_3 = 1^8 0^8 1^{16} 0^{24} 1^{16} 0^8 (0^8 1^8)^2 1^8 0^8 (0^8 1^8)^4 (1^8 0^8)^4$.

При вычислении ключей реализуется следующий алгоритм:

1. Присвоить значения
   $i = 1$, $U = H$, $V = M$.
2. Выполнить вычисление
   $W = U \oplus V$, $K_1 = P(W)$.
3. Присвоить $i = i + 1$.
4. Проверить условие $i = 5$. При положительном исходе перейти к шагу 7. При отрицательном – перейти к шагу 5.
5. Выполнить вычисление
   $U = A(U) \oplus C_i$, $V = A(A(V))$, $W = U \oplus V$, $K_i = P(W)$;
6. Перейти к шагу 3
7. Конец работы алгоритма.

**Шифрующее преобразование**

На данном этапе осуществляется шифрование 64-битных подслов слова *H* на ключах $K_i$ ($i$ = 1, 2, 3, 4).

Для шифрующего преобразования необходимо использовать следующие исходные данные:

$H = h_4 \| h_3 \| h_2 \| h_1$, $h_i \in V_{64}(2)$, $i = 1..4$ и набор ключей $K_1$, $K_2$, $K_3$, $K_4$.

После выполнения шифрования получают слова

$$s_i = E_{K_i}(h_i), \text{ где } i = 1, 2, 3, 4,$$

т.е. в результате получается вектор

$$S = s_4 \| s_3 \| s_2 \| s_1.$$

**Перемешивающее преобразование**

На данном этапе осуществляется перемешивание полученной последовательности с применением регистра сдвига.

Исходными данными являются слова $H$, $M \in V_{256}(2)$ и слово $S \in V_{256}(2)$.

Пусть отображение

$$\psi: V_{256}(2) \to V_{256}(2)$$

преобразует слово

$$\eta_{16}\|\ldots\|\eta_1,\ \eta_i \in V_{16}(2),\ i = 1..16$$

в слово

$$\eta_1 \oplus \eta_2 \oplus \eta_3 \oplus \eta_4 \oplus \eta_{13} \oplus \eta_{16}\|\eta_{16}\|\ldots\|\eta_2.$$

Тогда в качестве значения шаговой функции хэширования принимается слово

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

где $\psi^i$ – $i$-я степень преобразования $\psi$.

**Процедура вычисления хэш-функции**

Исходными данными для процедуры вычисления значения функции $h$ является подлежащая хэшированию последовательность $M \in B^*$. Параметром является стартовый вектор хэширования $H$ – произвольное фиксированное слово из $V_{256}(2)$.

Процедура вычисления функции $h$ на каждой итерации использует следующие величины:

$M \in B^*$ – часть последовательности $M$, не прошедшая процедуры хэширования на предыдущих итерациях;

$H \in V_{256}(2)$ – текущее значение хэш-функции;

$\Sigma \in V_{256}(2)$ – текущее значение контрольной суммы;

$L \in V_{256}(2)$ – текущее значение длины обработанной на предыдущих итерациях части последовательности $M$.

Алгоритм вычисления функции $h$ включает в себя следующие три этапа:

**Этап 1**

Присвоить начальные значения текущих величин

$M := M$; $H := H$; $\Sigma := 0^{256}$; $L := 0^{256}$

**Этап 2**

Проверить условие $|M|>256$. Если да, то перейти к этапу 3. В противном случае выполнить последовательность вычислений:

$L := <L+|M|>_{256}$; $M' := 0^{256-|M|}\|M$; $\Sigma := \Sigma\ [+]\ M'$

$H := \kappa(M',H)$; $H := \kappa(L,H)$; $H := \kappa(\Sigma,H)$;

Конец работы алгоритма. $H$ содержит значение хэш-функции.

**Этап 3**

Вычислить подслово $M_S \in V_{256}(2)$ слова $M$ ($M=M_P\|M_S$). Далее выполнить последовательность вычислений:

$H := \kappa(M_S,H)$; $L := <L+256>_{256}$; $\Sigma := \Sigma\ [+]\ M_S$; $M := M_P$

Перейти к этапу 2.

## *Функция хэширования* MD*5*

Предположим, что нам дано сообщение длиной $b$ бит, где $b$ – произвольное неотрицательное целое число, и пусть биты сообщения записаны в следующем порядке:

$$m_0\ m_1\ \ldots\ m_{(b-1)}.$$

Для вычисления свертки сообщения выполняются следующие пять шагов.

**Шаг 1. Добавление битов заполнения.**

Сообщение дополняется (расширяется) так, что его длина (в битах) становится сравнимой с 448 по модулю 512. Расширение выполняется всегда, даже если длина сообщения уже сравнима с 448 по модулю 512

Расширение выполняется следующим образом: к сообщению добавляется один бит, равный 1, а оставшиеся биты заполняются нулевыми значениями. Таким образом, количество добавленных битов может лежать в диапазоне от 1 до 512 включительно.

**Шаг 2. Добавление длины.**

64-битное представление длины сообщения $b$ (до добавления битов расширения) дописывается к результату предыдущего шага. В том маловероятном случае, когда длина сообщения превысит $2^{64}$, используются только младшие 64 бита представления. Эти биты добавляются в виде двух 32-разрядных слов, при этом младшее слово дописывается первым. После этой операции длина сообщения в точности кратна 512 битам и, аналогично, кратна 16 (32-разрядным) словам. Обозначим $M[0..N\text{-}1]$ слова полученного сообщения.

**Шаг 3. Инициализация буфера свертки.**

Буфер из четырех слов ($A$, $B$, $C$, $D$) используется для вычисления свертки сообщения. Эти регистры инициализируются следующими шестнадцатеричными значениями:

*A* = 01234567,
*B* = 89abc*def*,
*C* = *fed*cba98,
*D* = 76543210.

**Шаг 4. Обработка сообщения блоками по 16 слов.**

Определим сначала 4 вспомогательные функции, аргументом и результатом каждой из которых являются 32-битовые слова.

$F(X,Y,Z) = XY \vee \neg(X)\, Z$

$G(X,Y,Z) = XZ \vee Y\, \neg(Z)$

$H(X,Y,Z) = X \oplus Y \oplus Z$

$I(X,Y,Z) = Y \oplus (X \vee \neg(Z))$.

На этом шаге используется таблица из 64 слов *T*[1…64], построенная на основе функции синуса. Пусть *T*[*i*] обозначает *i*–й элемент таблицы, который равен целой части от 4294967296 ×abs(*sin*(*i*)), где *i* выражено в радианах.

Выполняются следующие шаги.

/* Обработать каждый 16-словный блок. */
*for i* = 0 *to N*/16 – 1 *do*
/* Копирование *i*-го блока в *X*. */
*For j* = 0 *to* 15 *do*
*X*[*j*] = *M*[*i**16 + *j*].
/* Сохранение *A* в *AA*, *B* в *BB*, *C* в *CC*, и *D* в *DD*. */
*AA* = *A*
*BB* = *B*
*CC* = *C*
*DD* = *D*
/* Этап 1. */
/* Пусть [*abcd k s i*] обозначает операцию
*a* = *b* + ((*a* + *F*(*b*,*c*,*d*) + *X*[*k*] + *T*[*i*]) << *s*). */
/* Выполните следующие 16 операций */
[*ABCD* 0 7 1] [*DABC* 1 12 2] [*CDAB* 2 17 3] [*BCDA* 3 22 4]
[*ABCD* 4 7 5] [*DABC* 5 12 6] [*CDAB* 6 17 7] [*BCDA* 7 22 8]
[*ABCD* 8 7 9] [*DABC* 9 12 10] [*CDAB* 10 17 11] [*BCDA* 11 22 12]
[*ABCD* 12 7 13] [*DABC* 13 12 14] [*CDAB* 14 17 15] [*BCDA* 15 22 16]
/* Шаг 2. */
/* Пусть [*abcd k s i*] обозначает операцию
*a* = *b* + ((*a* + *G*(*b*,*c*,*d*) + *X*[*k*] + *T*[*i*]) << *s*). */
/* Выполните следующие 16 операций */
[*ABCD* 1 5 17] [*DABC* 6 9 18] [*CDAB* 11 14 19] [*BCDA* 0 20 20]
[*ABCD* 5 5 21] [*DABC* 10 9 22] [*CDAB* 15 14 23] [*BCDA* 4 20 24]
[*ABCD* 9 5 25] [*DABC* 14 9 26] [*CDAB* 3 14 27] [*BCDA* 8 20 28]
[*ABCD* 13 5 29] [*DABC* 2 9 30] [*CDAB* 7 14 31] [*BCDA* 12 20 32]
a = b + ((a + *H*(b,c,*d*) + *X*[*k*] + *T*[*i*]) <<< *s*). */
/* Шаг 2. */
/* Пусть [*abcd k s i*] обозначает операцию
*a* = *b* + ((*a* + *H*(*b*,*c*,*d*) + *X*[*k*] + *T*[*i*]) << *s*). */
/* Выполните следующие 16 операций */
[*ABCD* 5 4 33] [*DABC* 8 11 34] [*CDAB* 11 16 35] [*BCDA* 14 23 36]
[*ABCD* 1 4 37] [*DABC* 4 11 38] [*CDAB* 7 16 39] [*BCDA* 10 23 40]
[*ABCD* 13 4 41] [*DABC* 0 11 42] [*CDAB* 3 16 43] [*BCDA* 6 23 44]
[*ABCD* 9 4 45] [*DABC* 12 11 46] [*CDAB* 15 16 47] [*BCDA* 2 23 48]
/* Шаг 4. */
/* Пусть [*abcd k s i*] обозначает операцию
*a* = *b* + ((*a* + *I*(*b*,*c*,*d*) + *X*[*k*] + *T*[*i*]) << *s*). */
/* Выполните следующие 16 операций */
[*ABCD* 0 6 49] [*DABC* 7 10 50] [*CDAB* 14 15 51] [*BCDA* 5 21 52]
[*ABCD* 12 6 53] [*DABC* 3 10 54] [*CDAB* 10 15 55] [*BCDA* 1 21 56]
[*ABCD* 8 6 57] [*DABC* 15 10 58] [*CDAB* 6 15 59] [*BCDA* 13 21 60]
[*ABCD* 4 6 61] [*DABC* 11 10 62] [*CDAB* 2 15 63] [*BCDA* 9 21 64]
/* Затем выполните следующие операции сложения */
*A* = *A* + *AA*
*B* = *B* + *BB*
*C* = *C* + *CC*
*D* = *D* + *DD*
e*nd* /* цикла по *i* */

**Шаг 5. Выход.**

Свертка сообщения содержится в регистрах *A*, *B*, *C*, *D*. Т.е., мы начинаем с младшего байта *A* и заканчиваем старшим байтом *D*.

Этим завершается описание алгоритма *MD5*.

# Приложение
# ПРОГРАММНЫЙ ПАКЕТ PGP

Пакет PGP (Pretty Good Privacy) без сомнений является на сегодня самым распространенным программным продуктом, позволяющим использовать современные надежные криптографические алгоритмы для защиты информации в персональных компьютерах.

К основным преимуществам данного пакета, выделяющим его среди других аналогичных продуктов следует отнести следующие:

1. **Открытость**. Исходный код всех версий программ PGP доступен в открытом виде. Любой эксперт может убедиться в том, что в программе эффективно реализованы криптоалгоритмы. Так как сам способ реализации известных алгоритмов был доступен специалистам, то открытость повлекла за собой и другое преимущество - эффективность программного кода.

2. **Стойкость**. Для реализации основных функций использованы лучшие (по крайней мере на начало 90-х) из известных алгоритмов, при этом допуская использование достаточно большой длины ключа для надежной защиты данных

3. **Бесплатность**. Готовые базовые продукты PGP (равно как и исходные тексты программ) доступны в Интернете в частности на официальном сайте PGP Inc. (www.pgpi.org).

4. **Поддержка как централизованной** (через серверы ключей) **так и децентрализованной** (через «сеть доверия») **модели** распределения открытых ключей.

4. **Удобство программного интерфейса**. PGP изначально создавалась как продукт для широкого круга пользователей, поэтому освоение основных приемов работы отнимает всего несколько часов. Первоначально использовался интерфейс командной строки, 5-я и особенно 6-я версия обладают всеми преимуществами интерфейса Windows.

История PGP начинается в 1991 г., когда программист Филипп Циммерман (Philip Zimmerman) на основе публично известных алгоритмов шифрования написал программу для защиты файлов и сообщений от несанкционированного прочтения. К тому времени вокруг криптографических продуктов для гражданских целей в США складывалась неоднозначная ситуация, с одной стороны они стали достоянием общественности, с другой - правительственные организации стремились внести ряд ограничений. Так в 1991 году появился законопроект S.266 («Билль о прочтении зашифрованной корреспонденции»), действовали ограничение на экспорт криптографических продуктов, снятые фактически только недавно, затем в 1994 году появился на свет законопроект "О цифровой телефонии".

Но истинным апофеозом стал проект Clipper, инициированный в 1993 году агентством национальной безопасности США (АНБ), согласно которому организации и частные пользователи должны были бы сдавать на депонирование используемые ключи. Это давало бы возможность спецслужбам получить доступ к любой интересующей их информации. Правда, из-за технологической сложности, дороговизны и общественного осуждения проект был заморожен.

В таких условиях программа PGP как своеобразное выражение технологического протеста не могла не появиться. За что Циммерман был подвергнут преследованиям, конкретно ему пытались инкриминировать экспорт криптографических алгоритмов (программа быстро распространилась за пределы США через Интернет). В итоге обвинение было снято, в 1996 году была образована компания Pretty Good Privacy, Inc. Знаменитый продукт был экспортирован официальным хотя и курьезным способом - исходный текст программы был опубликован в виде книги затем вывезен из США, отсканирован и скомпилирован в виде программы.

Первая наиболее популярная версия PGP - это вторая (2.x), "классическая" PGP с строчным интерфейсом. Она выполняла ряд базовых функций, которые перешли и в более поздние версии:

- генерацию пары из закрытого/открытого ключа;
- шифрование файла с помощью открытого ключа любого пользователя PGP (в том числе своего);
- расшифровку файла с помощью своего закрытого ключа;
- наложение цифровой подписи с помощью своего закрытого ключа на файл (аутентификация файла) или на открытый ключ другого пользователя (сертификация ключа);
- проверку (верификацию) своей подписи или подписи другого пользователя с помощью его открытого ключа.

В настоящий момент распространена 6-я версия (для Windows) PGP.

Рис. 1. Утилита PGPkeys из пакета PGP 6.0.2i

В основу шифрования в PGP положен гибридный алгоритм, соединяющий достоинства асимметричного и симметричного шифрования. Первое как известно позволяет избежать обмена ключевой информацией, второе требует меньших вычислительных ресурсов.

Для шифрования сообщения (файла) генерируется случайный ключ, который используется для симметричного криптоалгоритма. Сам ключ шифруется с помощью открытого ключа того, кому предназначается закрытая информация.

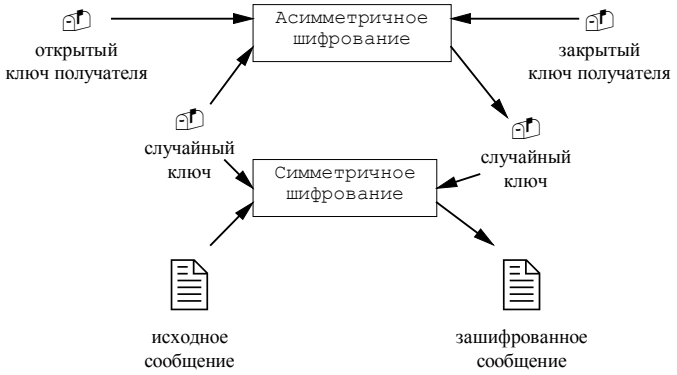



Рис. 2. Гибридный криптоалгоритм, используемый в PGP

Для затруднения критоанализа и выравнивания статистических характеристик шифруемой информации перед закрытием она подвергается сжатию по изветсному алгоритму ZIP (Jean-Loup Gailly, Mark Adler, Richard B. Wales). Дли шифрования используется несколько алгоритмов, некоторые из которых выбирает сам пользователь.

Асимметричные криптографические алгоритмы

| Алгоритм | Длина ключа, Кбит | Примечания |
|---|---|---|
| RSA | До 1 | Используется как для шифрова формирования подписи (дайджест MD |
| DHE (Алгоритм Диффи-Эль-Гамаля) | До 4 | Используется для шифрования |
| DSS | До 2 | Используется для формирования подп 160 бит) |

Симметричные криптографические алгоритмы

| Алгоритм | Режим использования | Длина блока, бит | Длина ключа, бит | Примечания |
|---|---|---|---|---|
| CAST | CFB | 64 | 128 | Используется с ключами DH |
| IDEA | CFB | 64 | 128 | Используется с ключами RS |
| тройной DES | CFB | 64 | 168 | Используется с ключами DH |

С декабря 1997 года Pretty Good Privacy, Inc входит в состав гигантской компании Network Associates (образованной в результате слияния McAfee Assosiates и Network General). Однако с начала 2000 года PGP снова существует как самостоятельная компания.

Литература:


1. Жельников В.А. Криптография от папируса до компьютера. М., BF, 1997.
2. Романец Ю.Ф., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. М., Радио и связь, 1999 г.
3. Введение в криптографию. Под общей редакцией Ященко В.В. М., МЦНМО-ЧеРо, 1998.