С. БАРИЧЕВ, Р. СЕРОВ

ОСНОВЫ СОВРЕМЕННОЙ КРИПТОГРАФИИ

V 1.2

Москва, 2001

Предисловие к электронному изданию

Данный документ является доработанной версией издания «Основы современной криптографии», вышедшего в издательстве «Горячая линия – Телеком», в 2001 году. Саму книгу можно приобрести практически в любом книжном интернет-магазине (см. например, www.findbook.ru).

Целью электронного издания является популяризация теоретических основ современной криптографии, в первую очередь в образовательных заведениях. Документ может неограниченно копироваться. Для распечатки и копирования текста книги необходимо БЕСПЛАТНО получить пароль, направив на адрес bars@orc.ru запрос с темой «psw», а в теле указав цель копирования и распечатки (а также кол-во экземпляров). Авторы просят воздержаться от передачи полученных паролей другим лицам. Не допускается полное или частичное использование текста книги без указания источника.

Сергей Баричев, Роман Серов

СОДЕРЖАНИЕ

1. КРИПТОГРАФИЧЕСКИЕ СИСТЕМЫ

1.1. Основные понятия и определения

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.

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.

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 этой книге будут рассматриваться различные криптографические методы. Современная криптография включает в себя четыре крупных раздела:

1. Симметричные криптосистемы.
2. Криптосистемы с открытым ключом.
3. Системы электронной подписи.
4. 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.

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

Зашифрование – процесс преобразования открытых данных в зашифрованные при помощи шифра.

Вместо термина "открытые данные" часто употребляются термины *открытый текст* и *исходный текст*, а вместо "зашифрованные данные" – *шифрованный текст*.

Расшифрование – процесс, обратный шифрованию, т.е процесс преобразования зашифрованных данных в открытые при помощи шифра.

Под *шифрованием* понимается процесс зашифрования или расшифрования.

Криптографическая система, или *шифр* представляет собой семейство **T** обратимых преобразований открытого текста в шифрованный. Члены этого семейства индексируются или обозначаются символом k; параметр k обычно называется *ключом*. Преобразование T_k определяется соответствующим алгоритмом и значением ключа k.

Ключ – конкретное значение состояния некоторых параметров алгоритма криптографического преобразования, обеспечивающее выбор одного преобразования из семейства.

Пространство ключей **K** – это набор возможных значений ключа. Обычно ключ представляет собой последовательный ряд букв алфавита.

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

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

В *системах с открытым ключом* используются два ключа – *открытый* (*публичный*) и *закрытый* (*секретный*), которые математически связаны друг с другом. Информация зашифровывается 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 много частных показателей криптостойкости, среди которых:

- количество всех возможных ключей;
- количество операций, необходимое для вскрытия шифра;

- объем выборки исходного и/или шифрованного текста, необходимый для определения ключа;
- сложность решения математической задачи, лежащей в основе системы шифрования.

Однако следует понимать, что эффективность защиты информации криптографическими методами зависит не только от криптостойкости шифра, но и от множества других факторов, включая организационные.

1.2. 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);
- 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 должна быть равной длине 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.

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.

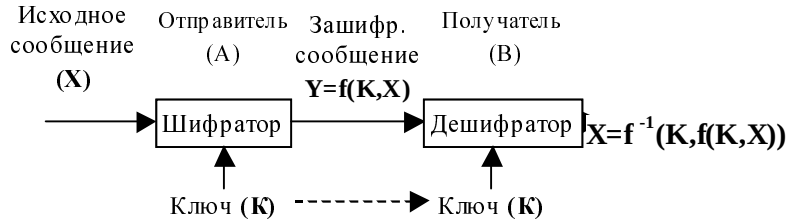


Рис 2.1. Схема симметричной криптосистемы.

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.

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 $x = (x_0, x_1, ..., x_{N-1})$, $y = (y_0, y_1, ..., y_{N-1})$. 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 π 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 π?

$(2^N - m)!$ 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 π является элементом, принадлежащим $SYM(Z_{2,N})$. Если известно, что π принадлежит небольшому подмножеству П из $SYM(Z_{2,N})$, то можно сделать более определенный вывод. Например, если

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

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

Криптографическое значение этого свойства должно быть очевидно: если исходный текст шифруется подстановкой π, 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 π 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 П[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 П[K] используется следующим образом. Пользователь i и пользователь j некоторым образом заключают соглашение относительно ключа k из K, выбирая, таким образом, элемент из П[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→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,
  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 $2^{64} \approx 1{,}8 \cdot 10^{19}$ 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 {А,..., 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 новом стандарте шифрования США *N* задано равным 128;
- достаточно большое пространство ключей для того, чтобы исключить возможность подбора ключа;
- сложные соотношения $\pi\{k,x\} : x \rightarrow y=\pi\{k,x\}$ 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. 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

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

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 ⊕ – 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.

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 выпол-

няет 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.

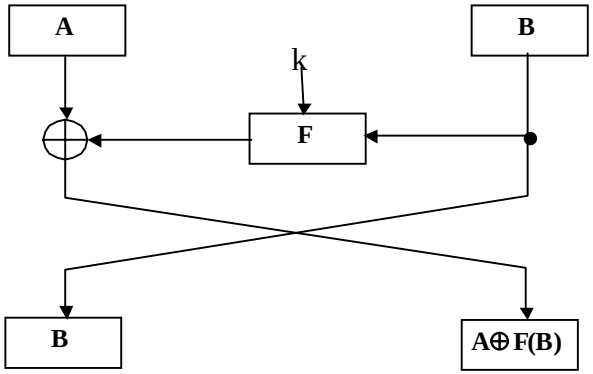


Рис. 2.2. Структура итерации сети Фейстела.

2.4. Алгоритмы блочного шифрования

2.4.1. Алгоритм DES и его модификации

Американский стандарт криптографического закрытия данных DES (Data Encryption Standard), принятый в 1978 г., 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:

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

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

58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,
62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,
57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7,

композиция $\theta \times \pi_{T_i}$, где θ – 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 π_{T_i}, $1 \le i \le 16$, описываются следующим образом:

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

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

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

Правый блок $\mathbf{X}'$ 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:

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

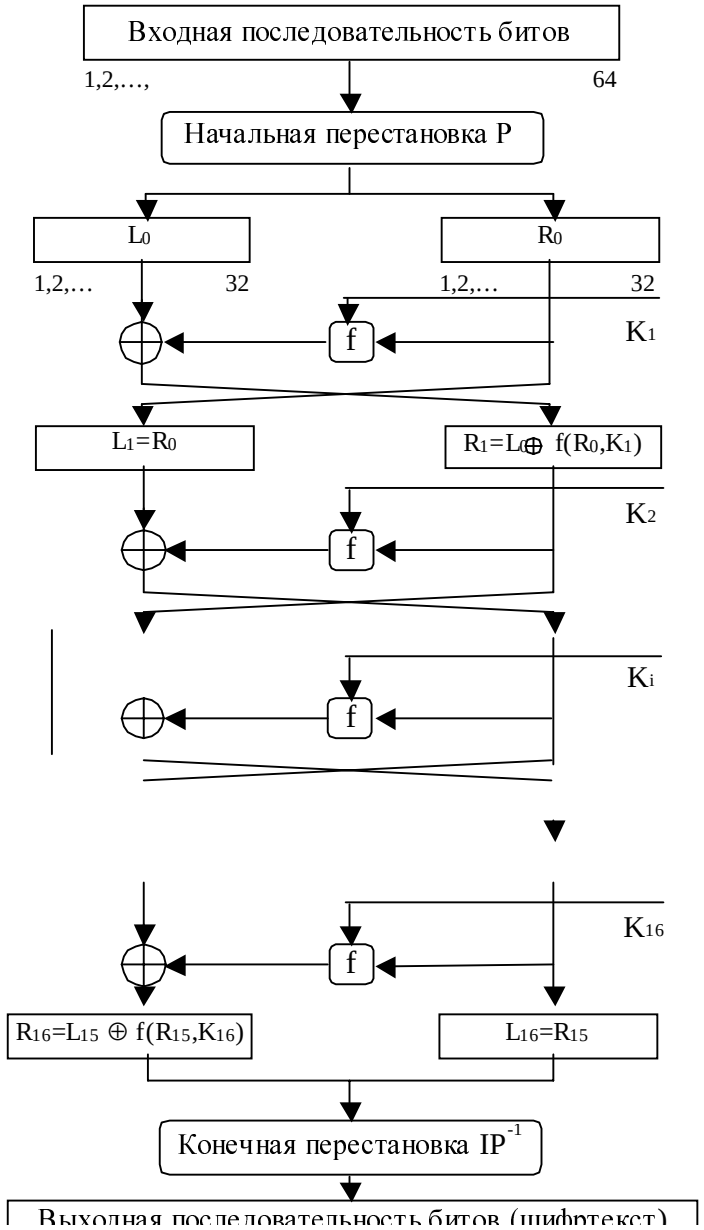


Рис 2.3. Схема алгоритма шифрования DES

Шаг 2. На i-циклической итерации 48 разрядов ключа

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

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 $\mathbf{Z}_{2,4}$ в $\mathbf{Z}_{2,4}$; 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 4×16 определяет подстановку на множестве $\mathbf{Z}_{2,4}$ 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-бокса):

16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25

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

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

Какие именно разряды ключа используются на i-циклической итерации, определяется по следующему алгоритму:

прежде всего 64 разряда ключа преобразуются в 56 путем выбрасывания каждого восьмого бита (который может использоваться для контроля целостности ключа);

производится начальная перестановка KP-1 56-разрядного ключа пользователя $\mathbf{k} = (k_{i,0}, k_{i,1} ..., k_{i,55})$:

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

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 KP-2. Эти разряды используются на первой итерации;

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

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

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
s	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

и вновь применяем KP-2 для получения очередной порции ключа.

Инверсией DES (обеспечивающей расшифрование зашифрованных посредством DES данных) является

$$DES = IP^{-1} \times \pi_{T_1} \times \theta \times K \times \theta \times \pi_{T_{16}} \times IP, \qquad (2.2)$$

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 EDE3 имеет длину $56 \times 3 = 168$ 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 $\mathrm{DES}^{-1}_{k_2}$, а не DES_{k_2}, является совместимость с DES: если выбрать $\mathbf{K}=k,k,k$, то $\mathrm{EDE3}_{\mathbf{K}} = \mathrm{DES}_{\mathrm{k}}$. 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 EDE3 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 eXtended), свободное от недостатков тройного DES.

DESX определяется как

$$DES_{k,k_1,k_2} = k_2 \oplus DES_k(k_1 \oplus \mathbf{x})$$

То есть, ключ DESX **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 $<x_i, DESX_{\mathbf{K}}(x_i)>$ в том случае, когда злоумышленник организует атаку на шифр по выбранному исходному тексту, получая множество пар $<P_j, DES_{\mathbf{K}}(P_j)>$.

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

$$DES\text{ - }PEP_{k,k_1,k_2} = k_2 + DES_k(k_1 + \mathbf{x})$$

где сложение определяется следующим образом: L.R + L'.R' = (L ◊ L').(R ◊ R'), |L|=|R|=|L'|=|R'|= 32, а ◊ 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.

2.4.2. Стандарт AES. Алгоритм Rijndael

В конце 1996 г. 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). Отбор проходил в два этапа, после первого среди претендентов осталось 15 кандидатов, после второго – 5. И вот, 2 октября 2000 года было принято окончательное решение. В качестве предлагаемого стандарта был выбран алгоритм Rijndael (произносится "Рейндал"). Этот алгоритм был разработан Винсентом Райманом (Vincent Rijman) и Йоан Дамен (Joan Daemen) и представляет собой алгоритм, не использующий сети Фейстела.

При описании алгоритма используется поле Галуа $GF(2^8)$, построенное как расширение поля $GF(2)$ по корням неприводимого многочлена $m(x) = x^8 + x^4 + x^3 + x + 1$. Данный многочлен выбран из соображений эффективности представления элементов поля. Элементарные операции, использующиеся в алгоритме, выполняются в указанном поле.

Алгоритм Rijndael представляет собой блочный шифр с переменной длиной блока и переменной длиной ключа. Длины блока и ключа могут быть выбраны независимо равными 128, 192 или 256 бит. Шифр является последовательностью итераций, выполняемых над некоторой промежуточной структурой, называемой *состоянием*. (Эта терминология заимствована из теории конечных автоматов.) Состояние может быть представлено в виде прямоугольного массива байтов. В массиве 4 строки, а число столбцов, обозначаемое как Nb, равно длине блока, деленной на 32. Ключ шифрования аналогичным образом представляется в виде прямоугольного байтового массива с 4 строками. Количество столбцов, обозначаемое Nk, равно длине ключа, деленной на 32. Входные и выходные значения алгоритма представляются в

виде одномерных байтовых массивов соответствующей длины. Состояние и ключевой массив заполняются из этих массивов вначале по столбцам, а затем по строкам. Количество итераций обозначается Nr зависит от Nb и Nk в соответствии со следующей таблицей:

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

Итерационное преобразование состоит из четырех различных преобразований. На C-подобном псевдокоде это выглядит так:

```
Round (State, RoundKey) {
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State, RoundKey);
}
```

Последняя итерация несколько отличается от всех остальных:

```
FinalRound (State, RoundKey) {
    ByteSub(State);
    ShiftRow(State);
    AddRoundKey(State, RoundKey);
}
```

Отдельные преобразования описываются ниже.

ByteSub

Это блок нелинейной обратимой байтовой замены (S-бокс), состоящий из двух операций:

1. Каждый байт заменяется на мультипликативный обратный к нему в поле $GF(2^8)$. Байт со значением '00'h отображается в себя.
2. Над каждым байтом выполняется аффинное преобразование в поле $GF(2)$, задаваемое следующим уравнением:

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

Это аффинное преобразование может быть описано в полиномиальном виде как $b(x) = (x^7 + x^6 + x^2 + x) + a(x)(x^7 + x^6 + x^5 + x^4 + 1) \bmod(x^8 + 1)$. Полином, на который производится умножение, выбран взаимно простым с модулем, так что умножение является обратимым.

Обратным к ByteSub будет преобразование, состоящее из обратного аффинного преобразования и взятия мультипликативного обратного в $GF(2^8)$.

ShiftRow

Это преобразование является циклическим сдвигом влево строк массива состояния на различную величину. Строка 0 не сдвигается, строка 1 сдвигается на C1 позиций, строка 2 – на C2 и строка 3 – на C3 позиций. Величины сдвига приведены в таблице:

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Обратным преобразованием будет циклический сдвиг строк массива вправо на то же количество позиций.

MixColumn

В этом преобразовании столбцы массива состояния рассматриваются как полиномы над полем $GF(2^8)$. Преобразование заключается в умножении столбца по модулю $x^4 + 1$ на фиксированный полином

$$c(x) = \text{'03h'}x^3 + \text{'01h'}x^2 + \text{'01h'}x + \text{'02h'}.$$

Этот полином является взаимно простым с $x^4 + 1$ и поэтому умножение обратимо. В матричной форме данное преобразование можно представить как

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

Обратное преобразование представляет собой умножение на полином, мультипликативно обратный к $c(x)$ по модулю $x^4 + 1$:

$$d(x) = \text{'0Bh'}x^3 + \text{'0Dh'}x^2 + \text{'09h'}x + \text{'0Eh'}.$$

AddRoundKey

Добавление ключа итерации осуществляется простым побитовым сложением по модулю 2 каждого байта массива состояния с соответствующим байтом массива ключа. Это преобразование является обратным самому себе.

Алгоритм обработки ключа.

Ключи итерации получаются из ключа шифрования с помощью *Алгоритма обработки ключа*, состоящего из двух компонентов – расширения ключа и выбора ключа итерации. Основные принципы его построения следующие:

- Общее число бит ключей итерации равно длине блока, умноженной на количество итераций плюс один. (Например, для блока 128 бит и 10 итераций потребуется 1408 бит ключей итерации).
- Ключ шифрования расширяется до *расширенного ключа.*
- Ключи итерации берутся из расширенного ключа следующим образом: первый ключ итерации состоит из первых Nb слов, второй – из следующих Nb слов и т.д.

Алгоритм расширения ключа

Расширенный ключ представляет собой линейный массив 4-байтовых слов и обозначается как W[Nb * (Nr + 1)]. Функция расширения ключа зависит от Nk. Существует две версии – для $Nk \le 6$ и для $Nk > 6$.

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)]) {
    for(i = 0; i < Nk; i++)
        W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++) {
        temp = W[i - 1];
        if (i % Nk == 0)
```

```
            temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
        W[i] = W[i - Nk] ^ temp;
    }
}
```

Здесь SubByte(W) – функция, возвращающая слово, в котором каждый байт является результатом применения блока замены шифра к байту, находящемуся на соответствующей позиции во входном слове. Функция RotByte(W) – циклический сдвиг байтов в слове, так что входное слово (*a, b, c, d*) преобразуется в слово (*b, c, d, a*).

Для Nk > 6 алгоритм выглядит так:

```
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)]) {
    for(i = 0; i < Nk; i++)
        W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++) {
        temp = W[i - 1];
        if (i % Nk == 0)
            temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
        else if (i % Nk == 4)
            temp = SubByte(temp);
        W[i] = W[i - Nk] ^ temp;
    }
}
```

Константы итерации Rcon не зависят от Nk и определяются как

$$\text{Rcon}[i] = (\text{RC}[i], '00', '00', '00'),$$

где RC[*i*] являются представлениями элементов поля $GF(2^8)$ со значениями x^{i-1}, т.е. RC[1] = 1 (т.е. '01'), и RC[*i*] = *x* (т.е. '02')·RC[*i* – 1].

Выбор ключа итерации

Ключ итерации с номером *i* задается словами из буфера расширенного ключа, начиная с W[Nb * *i*] и до W[Nb * (*i* + 1)].

Итак, процесс шифрования состоит из трех этапов:

- начального добавления подключа;
- Nr – 1 итераций;
- конечной итерации.

На псевдокоде это выглядит следующим образом:

```
Rijndael(State,CipherKey) {
```

```
    KeyExpansion(CipherKey,ExpandedKey) ;
    AddRoundKey(State,ExpandedKey);
    For( i=1 ; i<Nr ; i++ ) Round(State,ExpandedKey + Nb*i) ;
    FinalRound(State,ExpandedKey + Nb*Nr);
}.
```

2.4.3. Алгоритм RC6

В качестве одного из кандидатов фирмой RSA Data Security, Inc. был представлен алгоритм RC6 прошедший второй тур отбора. 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.

RC6 является полностью параметризованным алгоритмом шифрования. Конкретная версия RC6 обозначается как RC6–*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 RC6-*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 ⊕ *b* – побитовое "исключающее ИЛИ" *w*-битовых слов;

a × *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 RC6-*w*/*r*/*b* описывается следующим образом:

Вход: Исходный текст, записанный в 4 *w*-битовых входны регистрах *A*, *B*, *C*, *D*;
Число циклов шифрования *r*;
Ключевая таблица *S*[0; … 2*r* + 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 регистрах A, B, C, D;
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 RC6-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 and Q_{32} = 9E3779B9h – это константы, получаемые из двоичного представления e –2, где e – основание натуральных логарифмов, и ϕ – 1, где ϕ – золотое сечение, соответственно. Подобные же константы могут быть аналогичным образом получены и для RC6 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, предварительно загруженный в 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 {
    $A = S[i] = (S[i] + A + B) << 3$
    $B = L[j] = (L[j] + A + B) << (A + B)$
    $i = (i + 1) \bmod (2r + 4)$
    $j = (j + 1) \bmod c$
}

Структура шифра RC6 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.

2.4.4. 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 форми-

ровался 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.

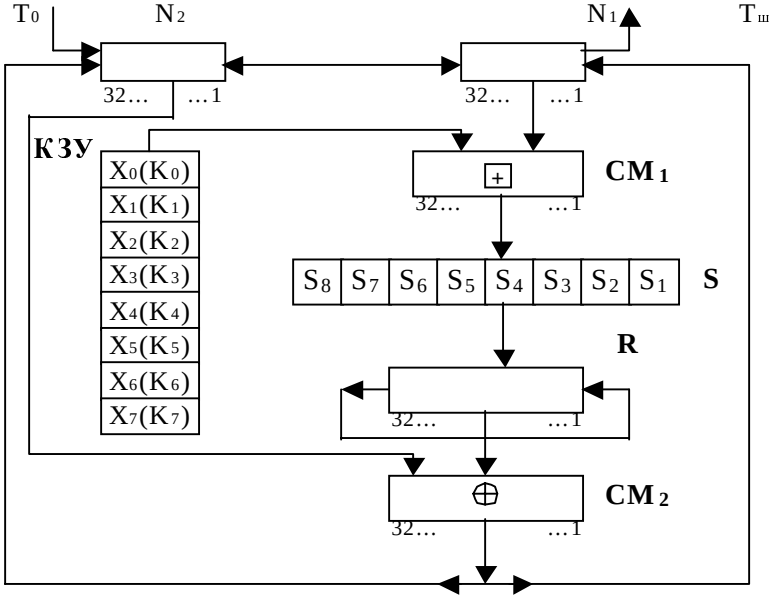


Рис 2.4. Алгоритм шифрования ГОСТ 28147-89.
Режим простой замены.

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

A ⊕ 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 $i = 1 \div 24$

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

для $i = 25 \div 31$

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

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 $T_{О}^{(i)}$ (i=1,2,...,m), где m определяется объемом шифруемых данных), зашифровываются в режиме гаммирования путем поразрядного сложения по модулю 2 с гаммой шифра $Г_{Ш}$, которая вырабатывается блоками по 64 бита, т.е.

$$Г_{Ш} = (Г_{Ш}^{(1)}, К\ , Г_{Ш}^{(m)}).$$

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

$$T_{Ш}^{(i)} = Г_{Ш}^{(i)} \oplus T_{О}^{(i)} = A(Y_{i-1}[+]C_2, Z_{i-1}\{+\}C_1) \oplus T_{О}^{(i)}).$$

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 = 01010104h и C_2 = 01010101h – 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:

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

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

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

2.4.5. Алгоритмы SAFER+, SAFER++

Алгоритм SAFER+ был предложен калифорнийской корпорацией Cylink совместно с Армянской академией наук как один из кандидатов на принятие в качестве нового стандарта шифрования AES и прошел первый тур отбора. Он является еще одним 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 $\text{K}_{2i\text{-}1}$ 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 $45^x \pmod{257}$ для байтов с номерами j = 1, 4, 5, 8, 9, 12, 13 и 16. При этом если x = 128, то $45^{128} \pmod{257} = 256$ 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.

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 M^{-1}, обратную к матрице шифрования.

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.

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

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

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

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:

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

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 и далее по циклу;
3. Выбранные байты складываются с соответствующими байтами слова смещения B_i по модулю 256. Результат сложения и является подключом K_i.

Алгоритм SAFER++ является дальнейшим развитием алгоритма SAFER+ и был представлен на европейский конкурс алгоритмов NESSIE. Этот алгоритм работает с блоками длиной 128 бит, но предусмотрен режим "обратной совместимости" для блоков 64 бита. Длина ключа может быть 128 или 256 бит.

Основная структура алгоритма осталась без изменений. Количество итераций равно 7 или 10 в зависимости от длины ключа.

Структура итераций в целом тоже осталась прежней. Итерация зашифрования состоит из слоя подмешивания подключа, слоя нелинейной замены, еще одного слоя подмешивания подключа и линейного обратимого преобразования. Основное отличие алгоритма заключается в структуре линейного преобразования, которое сделано более эффективным с вычислительной точки зрения. Рассмотрим его более подробно.

Вначале производится предварительная перестановка байтов в соответствии с перестановкой [9, 6, 3, 16, 1, 14, 11, 8, 5, 2, 15, 12, 13, 10, 7, 4]. Затем байты группируются по 4 и к каждой четверке применяется 4-точечное псевдопреобразование Адамара. После этого производится еще одна такая же перестановка, и еще раз применяется псевдопреобразование Адамара.

4-точечное псевдопреобразование Адамара задается невырожденной матрицей, имеющей обратную:

$$\mathbf{H}_4 = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \mathbf{H}_4^{-1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & -1 & 4 \end{bmatrix}.$$

Замечательная особенность этого преобразования заключается в том, что умножение на матрицы может быть реализовано путем всего лишь шести операций сложения (вычитания). Если входные байты обозначить как *a, b, c, d*, а выходные как *A, B, C, D*, то вычисления будут проводиться по формулам:

$$D = a + b + c + d \text{ (3 сложения)},$$
$$A = D + a \text{ (1 сложение)},$$
$$B = D + b \text{ (1 сложение)},$$
$$C = D + c \text{ (1 сложение)},$$

и обратно:

$$a = A - d \text{ (1 вычитание)},$$
$$b = B - d \text{ (1 вычитание)},$$
$$c = C - d \text{ (1 вычитание)},$$
$$d = D - a - b - c \text{ (3 вычитания)}.$$

Алгоритм расширения ключа претерпел незначительные изменения. Слова смещения $B_1, \ldots, B_{2r+1}$ вычисляются по следую-

щим формулам (слово B_1, как и в алгоритме SAFER+, не используется, можно считать его равным 0):

$$B_{i,j} = \begin{cases} 45^{(45^{17i+j} \bmod 257)} \bmod 257, i = 2\text{K}\,15, j = 1\text{K}\,16 \\ 45^{17i+j} \bmod 257, i = 16\text{K}\,21, j = 1\text{K}\,16 \end{cases}.$$

Процедура генерации подключей итераций проводится отдельно для подключей с четными и нечетными номерами. Для генерации подключей с нечетными номерами берутся первые 16 байт пользовательского ключа. Для генерации подключей с четными номерами берутся вновь первые 16 байт пользовательского ключа, если используется ключ 128 бит, либо оставшиеся 16 байт, если используется ключ 256 бит. Далее вычисляется байтовая контрольная сумма выбранных 16 байт, которая добавляется к ним справа. Затем из полученного расширенного ключа происходит выборка 16 байт подключа итерации с заданным номером. Выборка происходит следующим образом. Для получения подключа K_1 берутся 16 байт, начиная с 1, для подключа K_3 – 16 байт, начиная с 3, и т.д. по циклу. Выбранные байты складываются с байтами соответствующего слова смещения. После чего каждый байт расширенного ключа циклически сдвигается на 6 бит влево для нечетных подключей и на 3 бита для четных подключей. Процедура повторяется до получения всех требуемых подключей итерации.

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).

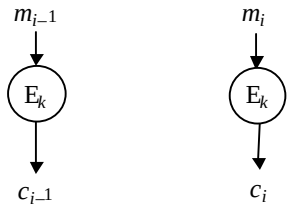
В режиме *электронной кодировочной книги* каждый блок исходного текста шифруется блочным шифром независимо от других (см. Рис. 2.5).

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.

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 2.6). Для начала процесса шифрования используется *синхропосылка* (или начальный вектор), которая передается в канал связи в открытом виде.

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.

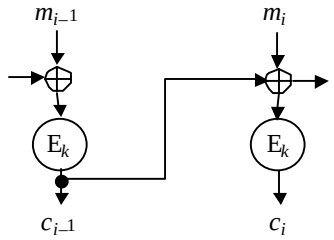


$c_i = E_k(m_i)$; $m_i = D_k(c_i)$

Рис. 2.5. Режим электронной кодировочной книги.

Скорость шифрования равна скорости работы блочного шифра, но простого способа распараллеливания процесса шифрова-

ния не существует, хотя расшифрование может проводиться параллельно.



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

Рис. 2.6. Режим сцепления блоков шифрованного текста.

Одной из потенциальных проблем режима CBC является возможность внесения контролируемых изменений в последующий расшифрованный блок исходного текста. Например, если злоумышленник изменит один бит в блоке, то весь блок будет расшифрован неверно, но в следующем блоке появится ошибка в соответствующей позиции. Есть ситуации, когда такое нежелательно. Для борьбы с этой угрозой исходный текст должен содержать определенную избыточность.

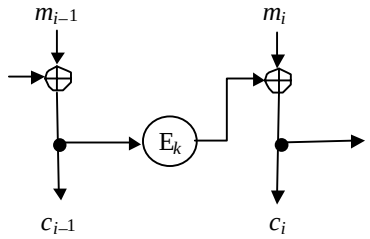
Известны модификации режима CBC. Рассмотрим некоторые из них. Режим *сцепления блоков шифрованного текста с распространением* (PCBC – Propagating CBC) отличается тем, что по модулю 2 складывается как предыдущий блок шифрованного, так и исходного текста:

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

Режим *сцепления блоков шифрованного текста с контрольной суммой* (CBCC –CBC with Checksum) отличается тем, что к последнему блоку исходного текста перед шифрованием прибавляется сумма по модулю два всех предыдущих блоков исходного текста. Это дает возможность проконтролировать целостность передаваемого текста с небольшими дополнительными накладными расходами.

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 2.7).



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

Рис. 2.7. Режим обратной связи по шифрованному тексту.

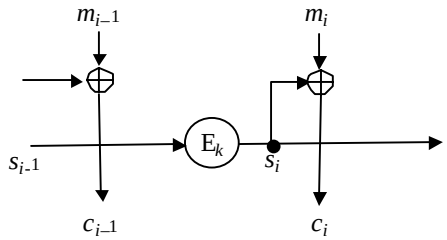
Стойкость режима CFB равна стойкости блочного шифра, лежащего в его основе и структура исходного текста скрывается за счет использования операции сложения по модулю 2. Манипулирование исходным текстом путем удаления блоков из начала или конца шифрованного текста становится невозможным. В режиме CFB если 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.

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. Существует модификация этого режима под названием

режим *обратной связи по выходу с нелинейной функцией* (OFBNLF – OFB with a NonLinear Function). В этом случае на каждом шаге меняется также и ключ шифрования:

$$c_i = E_{k_i}(m_i), k_i = E_k(k_{i-1});$$
$$p_i = D_{k_i}(c_i), k_i = E_k(k_{i-1}).$$



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

Рис. 2.8. Режим обратной связи по выходу.

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

В некоторых рассмотренных выше режимах шифрование блока зависит от шифрования предыдущего блока. Например, представим аппаратное устройство шифрования, работающее в режиме CBC. Даже если там будут три микросхемы, осуществляющие шифрование, только одна сможет работать в данный момент времени. Следующей микросхеме понадобится результат работы предыдущей. Решение заключается в *перемежении* нескольких потоков шифрования. Вместо одного потока в режиме CBC можно использовать четыре. Первый, пятый и далее каждый четвертый блоки будут шифроваться на одной микросхеме с одной синхропосылкой. Второй, шестой и далее каждый четвертый блок будут обрабатываться второй микросхемой со второй синхропосылкой и т. д. Таким образом, например, имея три микросхемы, осуществляющие шифрование со скоростью 33 Мбит/с, можно шифровать трафик в канале со скоростью 100 Мбит/с.

2.6. Потоковые шифры

2.6.1. Общие сведения о потоковых шифрах.

Потоковые шифры представляют собой разновидность гаммирования и преобразуют открытый текст в шифрованный последовательно по 1 биту. *Генератор ключевой последовательности*, иногда называемый *генератором бегущего ключа*, выдает последовательность бит k_1, k_2, …, k_i, … Эта ключевая последовательность складывается по модулю 2 с последовательностью бит исходного текста p_1, p_2, ..., p_i, … для получения шифрованного текста:

$$c_i = p_i \oplus k_i.$$

На приемной стороне шифрованный текст складывается по модулю 2 с идентичной ключевой последовательностью для получения исходного текста:

$$c_i \oplus k_i = p_i \oplus k_i \oplus k_i = p_i.$$

Стойкость системы целиком зависит от внутренней структуры генератора ключевой последовательности. Если генератор выдает последовательность с небольшим периодом, то стойкость системы будет невелика. Напротив, если генератор будет выдавать бесконечную последовательность истинно случайных (не псевдослучайных!) бит, то мы получим одноразовый блокнот с идеальной стойкостью.

Реальная стойкость потоковых шифров лежит где-то посредине между стойкостью простой моноалфавитной подстановки и одноразового блокнота. Генератор ключевой последовательности выдает поток битов, который выглядит случайным, но в действительности является детерминированным и может быть в точности воспроизведен на приемной стороне. Чем больше генерируемый поток похож на случайный, тем больше усилий потребуется от криптоаналитика для взлома шифра.

Однако, если каждый раз при включении генератор будет выдавать одну и ту же последовательность, то взлом криптосистемы будет тривиальной задачей. Перехватив два шифрованных текста, злоумышленник может сложить их по модулю 2 и получить два исходных текста, сложенных также по модулю 2. Такую систему раскрыть очень просто. Если же в руках противника окажется

пара исходный текст - шифрованный текст, задача вообще становится тривиальной.

По этой причине все потоковые шифры предусматривают использование ключа. Выход генератора ключевой последовательности зависит от этого ключа. В этом случае простой криптоанализ будет невозможен.

Потоковые шифры наиболее пригодны для шифрования непрерывных потоков данных, например, в сетях передачи данных.

Структуру генератора ключевой последовательности можно представить в виде конечного автомата с памятью, состоящего из трех блоков: блока памяти, хранящего информацию о состоянии генератора, выходной функции, генерирующей бит ключевой последовательности в зависимости от состояния, и функции переходов, задающей новое состояние, в которое перейдет генератор на следующем шаге.

2.6.2. Самосинхронизирующиеся шифры

В 1946 году в США была запатентована базовая идея так называемых *самосинхронизирующихся потоковых шифров* (или *шифрования с автоключом* – CipherText Auto Key (CTAK)). Она заключается в том, что внутреннее состояние генератора является функцией фиксированного числа предшествующих битов шифрованного текста. Поскольку внутреннее состояние зависит только от n бит шифрованного текста, генератор на приемной стороне войдет в синхронизм с передающей стороной после получения n бит.

Реализация этого подхода выглядит следующим образом. Каждое сообщение предваряется случайным заголовком длиной n бит. Этот заголовок шифруется и передается в линию. На приемной стороне заголовок расшифровывается. Результат расшифрования будет неверным, но после обработки n бит заголовка оба генератора будут синхронизированы.

Недостатком системы является распространение ошибок. При искажении одного бита генератор на приемной стороне выдаст n неверных бит ключевой последовательности, пока ошибочный бит не будет вытолкнут из памяти, что приведет к ошибочному расшифрованию n бит исходного текста.

Кроме того, самосинхронизирующиеся шифры уязвимы для атак типа "воспроизведение". Злоумышленник записывает неко-

торое количество бит шифрованного текста. Затем, позднее, он подменяет биты трафика записанными – "воспроизводит" их. После некоторого количества "мусора", пока приемная сторона не синхронизируется, старый шифрованный текст будет расшифровываться нормально. У приемной стороны нет никаких средств определения того, что принимаемые данные не являются актуальными.

Самосинхронизирующиеся потоковые шифры могут быть реализованы в виде блочных шифров, используемых в режиме обратной связи по шифрованному тексту (см. 2.5). При этом за один раз может шифроваться произвольное число бит, меньшее либо равное длине блока. Проиллюстрируем это на примере шифрования по одному байту за цикл.

Блочный шифр работает над очередью размером, равным длине блока. Первоначально очередь заполняется синхропосылкой. Затем очередь шифруется и левые 8 бит складываются с первыми 8 битами исходного текста. Полученные 8 бит шифрованного текста передаются в линию, очередь сдвигается влево на 8 бит, левые биты отбрасываются, а правые заполняются 8 битами шифрованного текста, переданными в линию. Далее процедура повторяется. Число 8 взято только для примера. За один цикл работы блочного алгоритма может шифроваться и 1 бит, хотя это будет не слишком эффективно с точки зрения скорости работы схемы. В режиме CFB синхропосылка должна быть уникальна для каждого сообщения в течение срока действия ключа. Если это будет не так, злоумышленник сможет восстановить исходный текст.

В случае возникновения ошибки в шифрованном тексте на приемной стороне в общем случае возникнут ошибки при расшифровании текущего и последующих $\lfloor m/n \rfloor$ блоков, где m – размер блока, n – число бит, шифруемых за 1 цикл, т.е. пока ошибочный бит шифрованного текста не будет вытеснен из памяти.

2.6.3. Синхронные шифры

В этом случае выходные значения генератора не зависят от исходного или шифрованного текстов. Такие потоковые шифры носят название *синхронных*.

Основная сложность в данном подходе заключается в необходимости синхронизации генераторов ключа на передающей и приемной сторонах. Если в процессе передачи произошло выпадение или вставка хотя бы одного бита, то вся последовательность битов шифрованного текста после ошибочного бита не сможет быть расшифрована. Если такое произойдет, стороны должны провести повторную синхронизацию. При этом синхронизация должна быть проведена так, чтобы никакой отрезок ключевой последовательности не повторился, так что очевидное решение возвратиться к некоторому предыдущему состоянию генератора не подходит.

Положительным свойством синхронных потоковых шифров является отсутствие эффекта распространения ошибок. Один искаженный бит при передаче приведет к искажению только одного бита текста при расшифровании.

Синхронные шифры также защищают от вставок и выбрасываний отрезков шифрованного текста из потока. Такие операции приведут к нарушению синхронизации, что будет сразу же обнаружено на приемной стороне.

Однако, такие шифры уязвимы к изменению отдельных бит. Если злоумышленник знает исходный текст, то он сможет изменять биты в потоке шифрованного текста таким образом, что он будет расшифровываться так, как необходимо злоумышленнику.

Синхронный потоковый шифр может быть реализован в виде блочного шифра, работающего в режиме обратной связи по выходу (см. 2.5). Подобно описанному в п. 2.6.2 примеру, режим OFB может быть реализован с любым размером обратной связи, меньшим размера блока. Однако, данный способ не рекомендуется, так как при этом снижается период генератора до примерно $2^{m/2}$. При длине блока 64 это будет около 2^{32}, чего явно недостаточно.

В этом случае выходная функция очень часто выбирается простой, например, суммой по модулю 2 нескольких бит состояния или вообще одним битом состояния. Криптографическая стойкость обеспечивается функцией перехода к следующему состоянию, которая зависит от ключа. Иногда данный режим называют режимом *с внутренней обратной связью*, поскольку обратная связь является внутренней по отношению к алгоритму генерации ключевой последовательности.

Вариантом этого режима является схема, когда ключ задает начальное состояние генератора, после чего последний работает без дальнейшего вмешательства.

Еще одним способом построения потокового шифра является использование счетчика в качестве входного значения для блочного шифра. После каждого цикла шифрования блока значение счетчика увеличивается, чаще всего на единицу. Свойства данного режима в отношении распространения ошибок и синхронизации будут такими же, как и для режима OFB. В качестве счетчика может быть использован любой генератор псевдослучайных чисел, вне зависимости от его криптографической стойкости.

При использовании потокового шифра в режиме счетчика выбирается простая функция перехода и сложная, зависящая от ключа функция выхода. Функция перехода может быть простым счетчиком, увеличивающимся на единицу на каждом такте.

2.6.4. Примеры потоковых шифров

2.6.4.1. RC4

RC4 представляет собой потоковый шифр с переменной длиной ключа, разработанный в 1987 г. Роном Ривестом для компании RSA Data Security, Inc. В течение 7 лет этот шифр лицензировался компанией только на условиях неразглашения. Однако, в 1994 г. он был анонимно опубликован в Интернете и с тех пор стал доступен для независимого анализа.

Описывается шифр очень просто. Алгоритм работает в режиме OFB. Ключевая последовательность не зависит от исходного текста. Структура алгоритма включает блок замены размерностью 8×8: S_0, …, S_{255}. Блок замены представляет собой зависимую от ключа переменной длины перестановку чисел 0, ..., 255. Имеется два счетчика i и j, первоначально равные 0. Для генерирования псевдослучайного байта выполняются следующие действия:

$i = (i + 1) \bmod 256$
$j = (j + S_i) \bmod 256$
переставить S_i и S_j
$t = (S_i + S_j) \bmod 256$
$k = S_t$.

Затем байт k складывается по модулю 2 с байтом исходного текста для получения шифрованного.

Инициализация блока замены также проста. Вначале он заполняется линейно: $S_0 = 0$, $S_1 = 1$, ..., $S_{255} = 255$. Затем заполняется еще один 256-байтный массив ключом, при этом ключ может повторяться необходимое число раз для заполнения всего массива: k_0, ..., k_{255}. Счетчик j устанавливается в 0. После чего производятся следующие действия:

for $i = 0$ to 255
$j = (j + k_i + S_i) \bmod 256$
переставить S_i и S_j.

Шифрование по этому алгоритму примерно в 10 раз быстрее, чем шифрование DES при программной реализации.

Возможно обобщение алгоритма на большие длину слова и размер блока замены. Так, можно построить шифр с боком замены размерностью 16 × 16 (потребуется 128 Кбайт памяти) и длиной слова 16 бит. Этап инициализации будет значительно медленнее, необходим цикл до 65535, если мы хотим в точности следовать конструкции, но получившийся в результате алгоритм будет более быстрым.

2.6.4.2. SEAL

SEAL (Software Encryption ALgorithm) представляет собой приспособленный для программной реализации потоковый шифр, разработанный Филом Рогэвэем и Доном Копперсмитом из компании IBM. Алгоритм оптимизирован для 32-разрядных процессоров. Для эффективной работы ему требуются 8 32-разрядных регистров и кэш объемом несколько килобайт.

Одним из замечательных свойств этого шифра является то, что он не является потоковым шифром в традиционном смысле, а представляет собой семейство псевдослучайных функций. 160-битовый ключ k и 32-битовое значение n (индекс) шифр преобразует в L-битовую строку $k(n)$. L может принимать любое значение, меньшее 64 килобайт. Такой шифр мы будем обозначать SEAL(k, n, L). Предполагается, что если k выбирается случайно, то $k(n)$ будет вычислительно неотличима от случайной L-битовой функции от n.

На практике это дает еще одно преимущество. Большинство шифров генерирует битовые последовательности в одном направлении. Зная ключ k и позицию i, определить значение i-го бита ключевой последовательности можно только вычислив все биты до i-го один за другим. В случае семейства псевдослучайных функций мы получаем простой доступ к любому элементу ключевой последовательности. Это оказывается весьма полезным.

Предположим, что необходимо зашифровать содержимое жесткого диска компьютера по 512-байтным секторам. Тогда сектор с номером n будет шифроваться с помощью ключевой последовательности $k(n)$. При этом легко может быть обеспечен доступ к произвольному сектору диска.

Данный шифр также облегчает проблему синхронизации, свойственную традиционным потоковым шифрам. Сообщение с номером n будет шифроваться с ключевой последовательностью $k(n)$, и n будет передаваться вместе с сообщением. Получателю не нужно будет хранить состояние генератора ключевой последовательности и беспокоиться о потерянных сообщениях и их влиянии на процесс расшифрования.

Алгоритм SEAL предусматривает использование трех зависящих от ключа таблиц: R, S, и T. Эти таблицы заполняются на предварительном этапе при помощи алгоритма, основанного на SHA (см. 4.3.1), и зависят только от ключа.

Заполнение таблиц можно описать с помощью функции $G_a(i)$, которая представляет собой функцию сжатия из алгоритма SHA. a – 160-битное значение, i – 32-битный индекс. Значение a используется для инициализации внутренних регистров A, B, C, D, и E в алгоритме SHA, а 512-битный блок для обработки представляет собой строку $i \| 0^{480}$. Выходное значение функции G также имеет длину 512 бит.

Построим теперь функцию Γ с выходным значением длиной 32 бита, переиндексировав функцию G: $\Gamma_a(i) = H^i_{i \bmod 5}$, где для $j = \lfloor i/5 \rfloor$ $H_0^{5j} \| H_1^{5j+1} \| H_2^{5j+2} \| H_3^{5j+3} \| H_4^{5j+4} = G_a(j)$. Теперь определим:

$$T[i] = \Gamma_a(i),\ 0 \le i < 512,$$
$$S[j] = \Gamma_a(0\text{x}1000 + j),\ 0 \le j < 256,$$
$$R[k] = \Gamma_a(0\text{x}2000 + k),\ 0 \le k < 256.$$

Собственно алгоритм генерации ключевой последовательности может быть описан на псевдокоде следующим образом:

function SEAL(a; n; L)
    y = ∅;
    for l = 0 to ∞ do
    Initialize(n; l; A; B; C; D; n_1; n_2; n_3; n_4);
    for i = 1 **to** 64 **do**
1   $P = A$ & 0x7fc; $B = B + T[P/4]$; $A = A >>> 9$; $B = B \oplus A$;
2   $Q = B$ & 0x7fc; $C = C \oplus T[Q/4]$; $B = B >>> 9$; $C = C + B$;
3   $P = (P + C)$ & 0x7fc; $D = D + T[P/4]$; $C = C >>> 9$; $D = D \oplus C$;
4   $Q = (Q + D)$ & 0x7fc; $A = A \oplus T[Q/4]$; $D = D >>> 9$; $A = A + D$;
5   $P = (P + A)$ & 0x7fc; $B = B \oplus T[P/4]$; $A = A >>> 9$;
6   $Q = (Q + B)$ & 0x7fc; $C = C + T[Q/4]$; $B = B >>> 9$;
7   $P = (P + C)$ & 0x7fc; $D = D \oplus T[P/4]$; $C = C >>> 9$;
8   $Q = (Q + D)$ & 0x7fc; $A = A + T[Q/4]$; $D = D >>> 9$;
9   $y = y \parallel B + S[4i - 4] \parallel C + S[4i - 3] \parallel D + S[4i - 2] \parallel A + S[4i - 1]$;
10 **if** $|y| \geq$ L **then** return $(y_0 y_1 \ldots y_{L-1})$;
11 **if** *odd*(i) **then** $(A; B; C; D) = (A + n_1; B + n_2; C \oplus n_1; D \oplus n_2)$
            else $(A; B; C; D) = (A + n_3; B + n_4; C \oplus n_3; D \oplus n_4)$;
    end;
    end;
end.

Алгоритм использует подпрограмму инициализации, которая, используя таблицы R и T, устанавливает начальные значения внутренних переменных и регистров.

procedure Initialize(n; l; A; B; C; D; n_1; n_2; n_3; n_4)
  $A = n \oplus R[4l]$;
  $B = (n >>> 8) \oplus R[4l + 1]$;
  $C = (n >>> 16) \oplus R[4l + 2]$;
  $D = (n >>> 24) \oplus R[4l + 3]$;
  for j = 1 **to** 2 **do**
    $P = A$ & 0x7fc; $B = B + T[P/4]$; $A = A >>> 9$;
    $P = B$ & 0x7fc; $C = C + T[P/4]$; $B = B >>> 9$;
    $P = C$ & 0x7fc; $D = D + T[P/4]$; $C = C >>> 9$;

$P = D$ & 0x7fc; $A = A + T[P/4]$; $D = D >>> 9$;
end;
$(n_1; n_2; n_3; n_4) = (D; B; A; C)$;
$P = A$ & 0x7fc; $B = B + T[P/4]$; $A = A >>> 9$;
$P = B$ & 0x7fc; $C = C + T[P/4]$; $B = B >>> 9$;
$P = C$ & 0x7fc; $D = D + T[P/4]$; $C = C >>> 9$;
$P = D$ & 0x7fc; $A = A + T[P/4]$; $D = D >>> 9$;
end.

Таблица T представляет собой блок замены размерностью 9×32. На каждом шаге 9 бит одного регистра (A, B, C или D) используются как указатель в таблице T. Значение, извлеченное из T, складывается (арифметически или порязрядно по модулю 2) со следующим регистром (вновь A, B, C или D). Затем первый регистр циклически сдвигается на 9 позиций. На некоторых шагах второй регистр дополнительно модифицируется путем сложения по модулю 2 со сдвинутым первым регистром. После 8 шагов каждый регистр маскируется сложением по модулю 2 с некоторым значением из S и выдается в ключевую последовательность. Итерация завершается сложением регистров A и C с зависимыми от n значениями n_1, n_2, n_3, n_4. Какое конкретно значение выбирается, зависит от четности номера итерации. Таким образом, основные идеи, лежащие в основе этого алгоритма, можно сформулировать следующим образом:

1. Использование большого секретного зависящего от ключа блока замены T;
2. Перемежение операций, не коммутирующих друг с другом (сложение и исключающее ИЛИ);
3. Использование внутреннего состояния, явно не проявляющегося в потоке данных (значения n_i, используемые в конце итерации для модификации регистров A и C);
4. Изменение шаговой функции в зависимости от номера шага и изменение итерационной функции в зависимости от номера итерации.
5. Использование известных и отработанных алгоритмов для заполнения таблиц.

Шифр SEAL требует пять элементарных машинных операции в пересчете на один байт текста при шифровании и расшифровании. Таким образом, он является одним из самых быстрых программно реализуемых алгоритмов.

2.6.4.3. WAKE

Алгоритм WAKE (от англ. Word Auto Key Encryption – шифрование слов с автоключом) был предложен Дэвидом Уилером. На выходе мы получаем последовательность 32-битовых слов, которые могут служить в качестве гаммы шифра. WAKE работает в режиме CFB – предыдущее слово шифрованного текста используется для генерации следующего слова ключевой последовательности.

В алгоритме используется специальный блок замены S из 256 32-битных слов. При этом старшие байты этих слов являются перестановкой чисел от 0 до 255, а остальные три младших байта выбираются случайными. Работа алгоритма описывается следующим образом:

Вначале инициализируется блок замены на основе ключа. Затем инициализируются четыре регистра A, B, C и D начальными значениями, также зависящими от ключа (возможно другого): a_0, b_0, c_0, d_0. Очередное слово ключевой последовательности получается по формуле:

$$k_i = d_i.$$

После этого изменяется значение регистров:

$$a_{i+1} = \mathrm{M}(a_i, d_i),$$
$$b_{i+1} = \mathrm{M}(b_i, a_{i+1}),$$
$$c_{i+1} = \mathrm{M}(c_i, b_{i+1}),$$
$$d_{i+1} = \mathrm{M}(d_i, c_{i+1}),$$

где

$$\mathrm{M}(x, y) = (x + y) >> 8 \oplus \mathrm{S}_{(x+y)\ \&\ 255},$$

здесь 8 младших бит суммы $x + y$ используются для входа в таблицу замены.

Хотя Уилер предложил способ генерации блока замены, может быть использован и другой алгоритм выбора перестановки и случайного заполнения. Данный шифр является достаточно быстрым, хотя и нестойким к атакам по выбранному исходному тексту.

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.

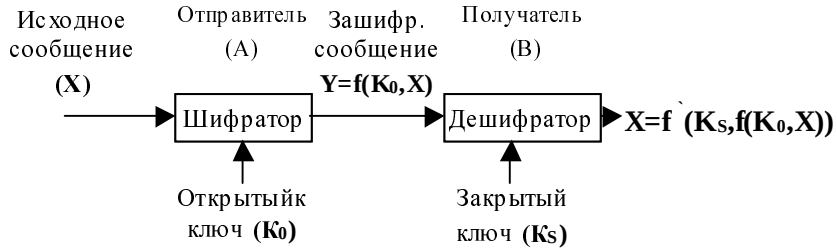


Рис 3.1. Схема асимметричной криптосистемы.

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 доказы-

вать 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 Σ множество всех конечных двоичных слов, а через Σ^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 $\mathbf{P} \subseteq \mathbf{NP}$. Является ли это включение строгим – одна из самых известных нерешенных задач математики. Большинство специалистов считают, что оно строгое (так называемая

гипотеза **P** ≠ **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 **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 **P** ≠ **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 **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 **P** ≠ **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 **P** ≠ **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 **P** ≠ **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.** Честная 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 Σ^n. Тогда для любого полинома p и всех достаточно больших n

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

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 (проблема Диффи-Хеллмана).

Допустим, задано большое простое число p и пусть g – примитивный элемент поля $GF(p)$. Тогда для любого a вычислить

$g^a \pmod p$ просто, а вычислить a по заданным $k = g^a \pmod p$ 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 значительное позитивное влияние на развитие криптографии.

3.3. Асимметричные системы шифрования

3.3.1. 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 А есть секретный ключ a и открытый ключ y, где $y = g^a \pmod p$. Предположим, что пользователь В желает послать сообщение m пользователю А. Сначала В 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". В посылает А пару (y_1, y_2).

После получения шифрованного текста пользователь А 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). Уравнение расшифрования в этом случае будет таким:

$$m = y_2/y_1{}^k \bmod p.$$

Однако, схема Эль-Гамаля не лишена определенных недостатков. Среди них можно указать следующие:

1. Отсутствие семантической стойкости. Если g – примитивный элемент $\boldsymbol{Z}_p$, то за полиномиальное время можно определить, является ли некоторое число x квадратичным вычетом, или нет. Это делается возведением в степень $x^{(p-1)/2} \bmod p$. Если результат равен 1, то x – квадратичный вычет по модулю p, если –1, то x – квадратичный невычет. Далее пассивный противник проверяет, являются ли g^k и g^v квадратичными вычетами. g^{kv} будет квадратичным вычетом тогда и только тогда, когда и g^k, и g^v будут квадратичными вычетами. Если это так, то $y_2 = m \cdot y^k \bmod p$ будет квадратичным вычетом, тогда и только тогда, когда само сообщение m будет квадратичным вычетом. То есть пассивный противник получает некоторую информацию об исходном тексте, имея лишь шифрованный текст и открытый ключ получателя.

2. Делимость шифра. Если дан шифрованный текст (y_1, y_2), то мы можем получить другой шифрованный текст, изменив только вторую часть сообщения. В самом деле, умножив y_2 на g^u $(u \neq 0)$, мы получим шифртекст для другого сообщения $m' = m \cdot g^u$.

Для защиты от подобных атак Шнорром и Якобссоном было предложено объединить схему шифрования Эль-Гамаля с цифро-

вой подписью Шнорра, что позволяет не только шифровать сообщение, но и аутентифицировать его.

Зашифрование осуществляется следующим образом. Автор сообщения выбирает случайные k, $s \in \mathbf{Z}_q$. далее от вычисляет $g^k \bmod p$, $my^k \bmod p$, $c = h(g^s, g^k, my^k)$ и $z = s + ck \bmod q$. Шифрованный текст представляет собой четверку (g^k, my^k, c, z).

Получив сообщение $(\bar{h}, \bar{f}, c, z)$, приемная сторона вначале верифицирует его, проверяя выполнение равенства $h(g^z \bar{h}^{-c}, \bar{h}, \bar{f}) = c$. В случае успешной верификации открытый текст получается по формуле $m = \bar{f} / \bar{h}^x \bmod p$. Расшифрование корректно, поскольку $\bar{h} = g^k$, $\bar{f} = my^k \bmod p$, так что $\bar{f} / \bar{h}^x = mg^{kx} g^{-kx} \bmod p = m$.

Заметим, что вторая половина шифрованного текста (c, z) зависит от исходного сообщения только через хэш-функцию h, которая статистически независима от m, и, следовательно, не содержит никакой информации об m.

Указанными авторами было предложено обобщение данной схемы на случай, когда пространство сообщений M представляет собой произвольную аддитивную группу, например $M = \mathbf{Z}_q^n$. Используются две статистически независимые хэш-функции $H : G^2 \times M \to \mathbf{Z}_q$ и $H_M : G \to M$. В базовой схеме шифрования $my^k \bmod p$ заменяется на $m + H_M(y^k) \in M$. Верификация шифрованного текста $(\bar{h}, \bar{f}, c, z)$ остается без изменений, а расшифрование будет проводиться по формуле $\bar{f} - H_M(\bar{h}^x)$.

3.3.2. Криптосистема, основанная на проблеме Диффи-Хеллмана

Данная система шифрования была представлена Мишелем Абдаллой, Михиром Беллэром и Филлипом Рогэвэем в рамках европейского проекта NESSIE (New European Schemes for Signatures, Integrity and Encryption). Эта система столь же эффективна, что и система Эль-Гамаля, но обладает дополнительными свойствами безопасности. Более того, стойкость системы может быть доказана в предположении о стойкости лежащих в ее основе криптографических примитивов.

Данная криптосистема реализуема на основе любой циклической группы, для которой может быть сформулирована проблема Диффи-Хеллмана, например, в $\{\boldsymbol{Z}_p^*\}$ или в группе точек на эллиптической кривой (см. 3.3.5).

Система строится из криптографических примитивов низкого уровня: групповой операции, симметричного шифра, функции хэширования (см. 4.3) и алгоритма вычисления кода аутентификации сообщения–имитовставки (MAC). Стойкость доказывается на основе предположения о сложности решения соответствующей проблемы Диффи-Хеллмана и предположения о стойкости входящих в схему симметричных примитивов.

Опишем криптографические примитивы, входящие в схему.

Циклическая группа $G = \{g\}$. Мы будем использовать мультипликативную запись групповой операции. Алгоритмы, реализующие эту операцию, будут работать с *представлениями* элементов группы в виде битовых строк фиксированной длины $gLen \in \boldsymbol{N}$. Способ кодирования $G \to \{0, 1\}^{gLen}$ не фиксируется и может выбираться, например, из соображений эффективности.

Код аутентификации сообщения позволяет пользователям, обладающим общим секретным ключом, выработать битовую строку для аутентификации и проверки целостности данных. Пусть $Msg = \{0, 1\}^*$ - пространство сообщений, $mKey = \{0, 1\}^{mLen}$ – пространство ключей для вычисления MAC для некоторого $mLen \in \boldsymbol{N}$, $Tag = \{0, 1\}^{tLen}$ – включающее множество всех возможных значений MAC для некоторого $tLen \in \boldsymbol{N}$. В этих обозначениях код аутентификации сообщений представляет собой пару алгоритмов MAC = {MAC.gen, MAC.ver}. Алгоритм генерации MAC определяется как отображение

$$MAC.gen(k, x): mKey \times Msg \to Tag$$

и может быть вероятностным.

Алгоритм верификации MAC является отображением

$$MAC.ver(k, x, \tau): mKey \times Msg \times Tag \to \{0, 1\}$$

со свойством $MAC.ver(k, x, MAC.gen(k, x)) = 1$.

В качестве MAC можно использовать, например, блочный шифр с достаточной длиной блока и ключа в режиме сцепления блоков шифрованного текста.

Симметричный шифр позволяет пользователям, обладающим общим секретным ключом, обеспечить секретность. Пусть Msg, как и ранее, пространство сообщений, eKey = $\{0, 1\}^{eLen}$ – пространство ключей для некоторого $eLen \in \boldsymbol{N}$, Ctext = $\{0, 1\}^*$ – включающее множество всех возможных значений шифрованного текста и Coins = $\{0, 1\}^\infty$ - множество строк бесконечной длины. В этих обозначениях шифр представляет собой пару алгоритмов SYM = {SYM.enc, SYM.dec}. Алгоритм зашифрования определяется как отображение

$$\text{SYM.enc}(k, x, r)\text{: eKey} \times \text{Msg} \times \text{Coins} \to \text{Ctext}$$

Алгоритм расшифрования является отображением

$$\text{SYM.dec}(k, y)\text{: eKey} \times \text{Ctext} \to \text{Msg} \cup \{\text{BAD}\},$$

где значение BAD выдается, если шифртекст y не является результатом зашифрования никакого открытого текста.

Асимметричный шифр. Пусть Msg, Ctext, Coins определены как и ранее, PK $\subseteq \{0, 1\}^*$, SK $\subseteq \{0, 1\}^*$ - множества открытых и секретных ключей. Асимметричный шифр определяется как тройка алгоритмов ASYM = {ASYM.enc, ASYM.dec, ASYM.key}. Алгоритм зашифрования является отображением

$$\text{ASYM.enc}(pk, x, r)\text{: PK} \times \text{Msg} \times \text{Coins} \to \text{Ctext}$$

а расшифрования:

$$\text{ASYM.enc}(sk, y)\text{: SK} \times \text{Ctext} \to \text{Msg} \cup \{\text{BAD}\}$$

Алгоритм выработки ключа в качестве аргумента берет строку $r \in$ Coins и выдает пару ключей $(pk, sk) \in$ PK $\times$ SK. При этом должно выполняться следующее свойство:
$\forall\ (pk, sk)$: $\exists\ r' \in$ Coins: (pk, sk) = ASYM.key(r'), $\forall\ r \in$ Coins $\forall\ x \in$ Msg ASYM.dec(sk, ASYM.enc(pk, x, r)) = x.

Функция хэширования является отображением следующего вида:

$$\text{H: } \{0, 1)^{2gLen} \to \{0, 1)^{mLen + eLen}.$$

Теперь мы можем описать криптографические примитивы, непосредственно составляющие рассматриваемую криптографиче-

скую систему. Графически процесс зашифрования представлен на рис. 3.1.

Все ключевые пары в данном алгоритме выбираются так же, как и в криптосистеме Эль-Гамаля, т.е. пара $(pk, sk) \equiv (g^v, v)$ для некоторого случайного v. При отсылке сообщения выбирается некоторое случайное значение u и получателю отсылается g^u, что обеспечивает неявный обмен ключами по семе Диффи-Хеллмана. Таким образом, зашифрованное сообщение состоит из одноразового открытого ключа, текста, зашифрованного симметричным шифром, и кода аутентификации сообщения, выработанного с помощью алгоритма MAC.gen.

Процесс расшифрования и аутентификации графически представлен на рис. 3.2. Элементы принятого сообщения также выделены двойной рамкой.

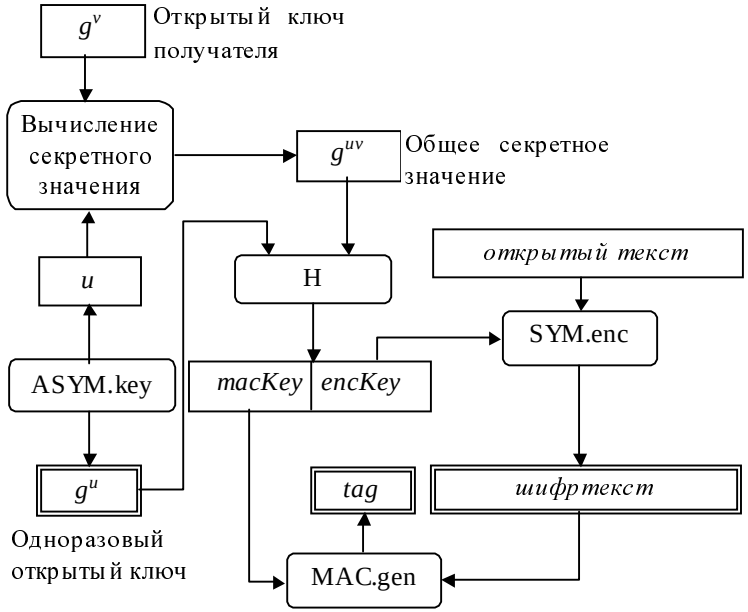


Рис. 3.2. Процесс зашифрования. Двойной рамкой выделены элементы зашифрованного сообщения.

Рассмотренная криптосистема является семантически стойкой и неделимой. В частности, неделимость обеспечивается тем, что значение g^u подается на вход функции хэширования. Если этого не сделать, то возможна атака, подобная описанной в п. 3.3.2 на

шифр Эль-Гамаля. Дело в том, что в некоторых группах (включая $\boldsymbol{Z}_p^*$) значения g^{uv} и g^v неоднозначно определяют значение g^u. Т.е. могут существовать $u \neq u'$, такие что $g^{uv} = g^{u'v}$. Пусть $l = (p - 1)/2$. Тогда $g^l = -1$. Если нам дано зашифрованное сообщение EM = g^u || encM || tag, то мы сможем вычислить новый шифртекст, EM' = $g^{u'}$ || encM ||tag, который с большой вероятностью будет соответствовать тому же самому открытому тексту. Пусть $g^{u'} = g^u \cdot g^l$. В этом случае $g^{u'v} = g^{uv} \cdot g^{lv} = g^{uv} \cdot (g^l)^v = g^{uv} \cdot (-1)^v$, и это равно g^{uv} тогда и только тогда, когда v – четное, т.е. в половине всех случаев. Таким образом, с вероятностью 1/2 вычисленный шифртекст будет являться результатом зашифрования того же самого исходного текста.

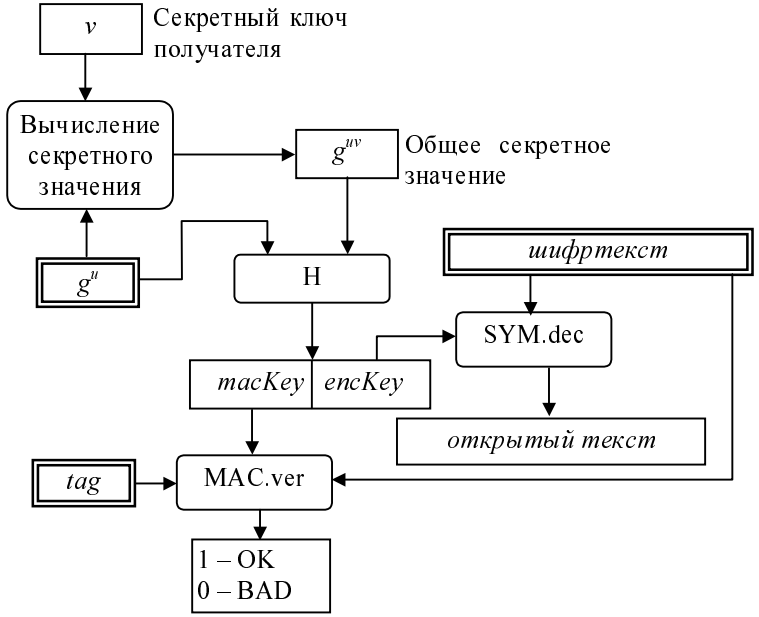


Рис. 3.3. Процесс расшифрования и аутентификации.

Эффективность предложенной схемы по существу та же, что и у шифра Эль-Гамаля, т.е. для зашифрования требуются две операции возведения в степень, а для расшифрования – одна. Тем самым для больших сообщений скорость шифрования будет определяться скоростью работы симметричного шифра и алгоритма вычисления кода аутентификации сообщения.

3.3.3. Криптосистема Ривеста-Шамира-Адлемана

Система Ривеста-Шамира-Адлемана (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

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

по основанию n_A:

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

Пользователь B зашифровывает текст при передаче его пользователю A, применяя к коэффициентам c_i отображение E_{e_A, n_A} :

$$E_{e_A, n_A} : c \longrightarrow c^{e_A} \pmod{n_A},$$

получая зашифрованное сообщение N'. В силу выбора чисел d_A и e_A, отображение E_{e_A, n_A} является взаимно однозначным, и обратным к нему будет отображение

$$E_{d_A, n_A} : c \longrightarrow c^{d_A} \pmod{n_A}$$

Пользователь A производит расшифрование полученного сообщения N', применяя E_{d_A, n_A} .

Для того чтобы найти отображение E_{d_A, n_A} , обратное по отношению к E_{e_A, n_A} , требуется знание множителей $n_A = p_A q_A$. Время выполнения наилучших из известных алгоритмов разло-

жения при $n > 10^{145}$ на сегодняшний день выходит за пределы современных технологических возможностей.

Существует вариант криптосистемы RSA, в которой вместо функции Эйлера используется функция Кармайкла λ, где $\lambda(n)$ – наименьшее целое t, такое что для любого целого x, взаимно простого с n, выполняется $x^t = 1 \bmod n$. Если n выбирается так, как описано выше, то $\lambda(n) = \text{НОК}(p-1, q-1)$

3.3.4. Криптосистемы Меркля-Хеллмана и Чора-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 **f**, так что i-й компонент **f** представляет собой место, занимаемое i-м объектом. Имеется рюкзак общим объемом K.

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

Важно отметить, что выбор **f** является критическим. Например, предположим, что **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 **f** и K вычислить **x** очень просто. Мы проверим, является ли K большим, чем последний элемент **f**, и если да, то мы делаем последний элемент **x** равным 1, вычитаем это значение из K и рекурсивно решаем меньшую проблему. Этот метод работает, поскольку когда K больше последнего элемента **f**, даже если мы выберем **x**=(1 1 1 … 1 0), то произведение **fx** все равно будет слишком маленьким, благодаря тому, что последовательность супервозрастающая. Таким образом, мы должны выбирать 1 в последней позиции **x**.

Ясно, что выбор **f** очень важен – в зависимости от **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 А получает свой открытый ключ следующим образом:

1. Выбирает супервозрастающую последовательность **f'** примерно из 100 элементов;
2. Выбирает случайное целое *m*, большее суммы элементов **f'**;
3. Выбирает другое случайное целое *w*, взаимно простое с *m*.
4. Теперь вычисляется **f''** умножением каждого компонента **f'** на *w* по модулю *m*;

$$\mathbf{f''} = \mathbf{f'}w \pmod{m}$$

5. И наконец, проводится случайная перестановка **P** элементов **f''** для получения открытого ключа **f**.

Теперь А раскрывает ключ **f** и держит в секрете **f'**, *m*, *w* и **P**.

Когда пользователь В хочет послать А сообщение (битовый вектор) **x**, он вычисляет $S = \mathbf{fx}$ и посылает это вычисленное *S*. Если данная система является стойкой, тогда для внешнего наблюдателя С вычисление **x** по *S* и публичному ключу **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 **f**.

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

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

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

Таким образом, А просто умножает *S* на мультипликативное обратное *w* по модулю *m*, а затем решает задачу рюкзака в случае

супервозрастающей последовательности **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 $\mathbf{f''} = \mathbf{f}$. Тогда для любого i

$$\mathbf{f}_i \equiv \mathbf{f'}_i w \bmod m$$

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

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

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 $\mathbf{k}$ и $\mathbf{f}$ 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:

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

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:

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

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 Чор 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

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

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 = **x*a**. В действительности, это не так уж и просто, поскольку сообщение должно быть длиной p бит и должно быть **x*1** = h, но Хор представил довольно прямолинейный метод преобразования неограниченной битовой строки в несколько блоков, каждый из которых имеет требуемую форму. A 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 **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.

3.3.5. Криптосистемы, основанные 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 полями Галуа произвольной характеристики и над полями Галуа характеристики 2.

***Определение* 3.2.** 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 ∞, 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 $\#E$. Теорема Хассе гласит, что $\#E = p + 1 - t$, где

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

$\#E$ называется *порядком* кривой E, а t – *следом* кривой E.

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) $\forall (x_1, y_1) \in E, y_1 \neq 0, (x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, где

$$x_3 = \lambda^2 - 2x_1,$$
$$y_3 = \lambda(x_1 - x_3) - y_1 \text{ и}$$
$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

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* 3.3.** 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

$$y^2 + xy = x^3 + ax^2 + b \qquad (3.2)$$

над полем $GF(2^m)$ вместе с дополнительной точкой ∞, 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) $\forall (x_1, y_1) \in E, x_1 \neq 0, (x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, где

$$x_3 = x_1^2 + \frac{b}{x_1^2},$$
$$y_3 = x_1^2 + (\lambda + 1)x_3 \text{ и}$$
$$\lambda = x_1 + \frac{y_1}{x_1}.$$

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

Группа точек на кривой имеет простую структуру, а именно она является абелевой группой ранга 1 или 2, т.е. изоморфна прямому произведению двух циклических групп $Z_{n_1} \times Z_{n_2}$, где n_1

и n_2 – целые числа, n_2 делит n_1 и n_2 делит $q - 1$, где q – порядок поля коэффициентов, при этом n_1 может быть равно 1. Индекс подгруппы Z_{n_1} в группе точек называют *кофактором* эллиптической кривой.

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.

4. ЭЛЕКТРОННЫЕ ЦИФРОВЫЕ ПОДПИСИ

4.1. Постановка задачи

Передача сообщения *отправителем* (пользователь А) *получателю* (пользователь В) 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 М (пользователь А – пользователю В) необходимо следующее:

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

Рассмотри эти пункты подробнее.

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

$$<\text{Идентификатор В, М}> \rightarrow \text{SIG}\{k_A, \text{М, идентификатор В}\}$$

является ключ криптопреобразования.

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

<M, идентификатор B> ↔ SIG{k_A, M, 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

4.2.1. Цифровые подписи, основанные на асимметричных криптосистемах

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

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

$$SIG(M) = E_{e_B,n_B}(E_{d_A,n_A}(M)).$$

При этом он использует:

- свое секретное преобразование E_{d_A,n_A};
- открытое преобразование E_{e_B,n_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 E_{d_B,n_B} с целью получения

$$E_{d_A,n_A}(M) = E_{d_B,n_B}(SIG(M)) = E_{d_B,n_B}(E_{e_B,n_B}(E_{d_A,n_A}(M))).$$

и затем открытого E_{e_A,n_A} пользователя А для получения сообщения М:

$$M = E_{e_A,n_A}(E_{d_A,n_A}(M)).$$

Затем пользователь В производит сравнение полученного сообщения М 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 В. Если же требуется обеспечение возможности верификации ЭЦП произвольным пользователем (например, при циркулярной рассылке документа), то алгоритм выработки ЭЦП упрощается, и подпись вырабатывается по формуле

$$SIG(M) = E_{d_A,n_A}(M),$$

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

$$M = E_{e_A,n_A}(SIG(M)) = E_{e_A,n_A}(E_{d_A,n_A}(M)).$$

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:

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

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 $y = g^x \pmod p$ – секретный и открытый ключи соответственно. Уравнения выработки подписи выглядят следующим образом:

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

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

Другой вариант подписи Шнорра выглядит так. Для подписи сообщения m автор выбирает случайное $k \in \mathbf{Z}_q$, вычисляет $g^k \bmod p$, $e = h(g^k, m)$ и $z = k + xe \bmod q$. Подписью является тройка (m, e, z). Проверка подписи заключается в проверке равенства $h(g^z y^{-e}, m) = e$. В самом деле, $g^z y^{-e} = g^{k + xe} g^{-xe} = g^k$.

4.2.2. Стандарт цифровой подписи DSS

Новая редакция стандарта на выработку и верификацию цифровой подписи DSS (Digital Signature Standard) принята в США 7 января 2000 г. (FIPS PUB 186-2) Согласно этому стандарту,

электронная цифровая подпись может вырабатывается по одному из трех алгоритмов: DSA (Digital Signature Algorithm), основанному на проблеме логарифма в конечном поле, ANSI X9.31 (RSA DSA) или ANSI X9.63 (EC DSA – алгоритм выработки подписи, основанной на проблеме логарифма в группе точек эллиптической кривой над конечным полем).

Опишем алгоритм DSA.

1. Предварительный этап – выбор параметров.

Выбираются числа p, q *и* g, такие, что p – простое число, $2^{l-1} < p < 2^l$, где l кратно 64 и $512 \le l \le 1024$; q – простой делитель числа $p - 1$ длиной 160 бит ($2^{159} < q < 2^{160}$); g – элемент порядка q в Z_p. g выбирается в виде $g = h^{(p-1)/q}$, где $1 < h < p - 1$ и $h^{(p-1)/q} > 1$. Эти три числа являются открытыми данными и могут быть общими для группы пользователей.

Выбирается секретный ключ x, $0 < x < q$, и вычисляется открытый ключ для проверки подписи $y = g^x \pmod p$.

2. Выработка электронной цифровой подписи.

Вычисляется значение хэш-функции от сообщения $h(m)$. При этом используется алгоритм безопасного хэширования SHA-1 (Secure Hashing Algorithm), на который ссылается стандарт (FIPS PUB 180-1). Значение хэш-функции $h(m)$ имеет длину 160 бит.

Далее подписывающий выбирает случайное или псевдослучайное значение k, $0 < k < q$, вычисляет $k^{-1} \pmod q$, и вырабатывает пару значений:

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

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:

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

В самом деле:

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

Алгоритм выработки ЭЦП, основанный на эллиптических кривых, может быть описан следующим образом:

1. Выбор параметров.

Стандарт определяет поля, над которыми задаются эллиптические кривые. Это простые поля Галуа и поля Галуа характеристики 2. Выбор полей в стандарте сделан исходя из требования повышения вычислительной эффективности машинных операций умножения в поле. Для этого в качестве простых модулей выбраны так называемые обобщенные числа Мерсенна (Табл. 4.1).

Стандарт фиксирует кривые, которые должны использоваться в алгоритмах, и примерные базовые точки на этих кривых. Пользователь может либо воспользоваться приведенными в стандарте базовыми точками, либо сгенерировать свои, если, например, ему понадобится обеспечить криптографическое разделение сетей ЭВМ. В частности, кривые P-192 … P-521 представляют собой эллиптические кривые простого порядка r вида $y^2 = x^3 - 3x + b$ над полем $GF(p)$.

Таблица 4.1.

Кривая	p
P-192	$2^{192} - 2^{64} - 1$
P-224	$2^{224} - 2^{96} + 1$
P-256	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
P-384	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
P-521	$2^{521} - 1$

Для задания кривых над полями характеристики 2 выбраны порождающие многочлены, указанные в табл. 4.2. При этом возможно использовать представление полей как в полиномиальном, так и в нормальном базисах.

Таблица 4.2.

Кривая	Порождающий многочлен $p(t)$	Тип нормального базиса
K-163, B-163	$t^{163} + t^7 + t^6 + t^3 + 1$	4
K-233, B-233	$t^{233} + t^{74} + 1$	2

K-283, B-283	$t^{283} + t^{12} + t^7 + t^5 + 1$	6
K-409, B-409	$t^{409} + t^{87} + 1$	4
K-571, B-571	$t^{571} + t^{10} + t^5 + t^2 + 1$	10

Коэффициенты b заданы для каждого размера поля. Например, кривая P-521 задается коэффициентом b = 051 953eb961 8e1c9a1f 929a21a0 b68540ee a2da725b 99b315f3 b8b48991 8ef109e1 56193951 ec7e937b 1652c0bd 3bb1bf07 3573df88 3d2c34f1 ef451fd4 6b503f00 (в шестнадцатеричном виде) и имеет порядок r = 6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449 (в десятичной записи). Поскольку определенные в стандарте кривые имеют простой порядок, группы точек на них являются циклическими (кофактор этих кривых равен 1) и порядок базовой точки n в точности равен r.

Для каждого из полей $GF(2^m)$ в стандарте указаны по 2 кривых: псевдослучайная вида $y^2 + xy = x^3 + x^2 + b$, имеющая кофактор 2, и специальная кривая Коблица, или аномальная двоичная кривая вида $y^2 + xy = x^3 + ax^2 + 1$, где $a = 0$ или 1. Кофактор кривой Коблица равен 2, если $a = 1$ и 4, если $a = 0$. Например, для поля $GF(2^{163})$ псевдослучайная кривая задается коэффициентом b = 2 0a601907 b8c953ca 1481eb10 512f7874 4a3205fd, а кривая Коблица коэффициентом $a = 1$.

Генерация ключевой пары. Пользователь выбирает в качестве секретного ключа целое число d, $0 < d < n$, где n – порядок базовой точки G на эллиптической кривой. Далее он вычисляет $Q = dG$ и публикует Q в качестве открытого ключа.

2. Выработка ЭЦП.

Для того, чтобы подписать сообщение m, пользователь:

1. выбирает случайное число k, $0 < k < n - 1$;
2. вычисляет $kG = (x_1, y_1)$, $r = x_1 \bmod n$. Если $r = 0$, то перейти к шагу 1;
3. вычисляет $k^{-1} \bmod n$;
4. вычисляет $e = \mathrm{SHA}(m)$. Выбор варианта функции хэширования осуществляется в зависимости от используемого поля.
5. вычисляет $s = k^{-1}(e + dr) \bmod n$. Если $s = 0$, то перейти к шагу 1;
6. Подписью сообщения m является пара (r, s).

3. Верификация ЭЦП.

Для проверки подписи получатель сообщения выполняет следующие действия:

1. проверяет, что r и s лежат на интервале $(0, n)$.
2. вычисляет $e = \mathrm{SHA}(m)$;
3. вычисляет $w = s^{-1} \bmod n$;
4. вычисляет $u_1 = ew \bmod n$ и $u_2 = rw \bmod n$;
5. вычисляет $X = u_1G + u_2Q$. Если $X = \infty$, то подпись отвергается, иначе, вычислить $v = x_1 \bmod n$, где $X = (x_1, y_1)$;
6. Принять подпись, если и только если $v = r$.

Корректность схемы доказывается несложно. Если сообщение m действительно подписано отправителем, то $s = k^{-1}(e + dr) \bmod n$. Откуда

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2d \pmod{n}.$$

Таким образом, $u_1G + u_2Q = (u_1 + u_2d)G = kG$, и поэтому $v = r$, как и требуется.

4.2.3. Стандарт цифровой подписи ГОСТ Р 34.10-94

Российский стандарт ЭЦП разрабатывался позже первоначального варианта 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 $a^q \pmod p = 1$.

k – целое число, $0 < k <$ q.

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

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

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 $h(m) \pmod q = 0$, то присвоить $h(m)$ значение $0^{255}1$.
2. Выработать целое число k, $0 < k < q$.
3. Вычислить два значения: $r' = a^k \pmod p$ и $r = r' \pmod q$. Если $r = 0$, то перейти к шагу 2 и выработать другое значение числа k.
4. С использованием секретного ключа x пользователя вычислить значение $s = (xr + kh(m)) \pmod q$. 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 $s = xr \pmod q$ 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:

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

В самом деле,

$$(a^{s \cdot h(m)^{-1}} \cdot y^{-r \cdot h(m)^{-1}} \pmod p) \pmod q = (a^{s \cdot h(m)^{-1}} \cdot a^{-r \cdot x \cdot h(m)^{-1}} \pmod p) \pmod q =$$
$$= a^{h(m)^{-1} \cdot (s - x \cdot r)} \pmod p \pmod q = a^{h(m)^{-1}(x \cdot r + k \cdot h(m) - x \cdot r)} \pmod p \pmod q =$$
$$= a^{k \cdot h(m)^{-1} \cdot h(m)} \pmod p \pmod q = a^k \pmod p \pmod q \equiv r.$$

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 $h(m_1) \pmod q = 0$, присвоить $h(m_1)$ значение $0^{255}1$.
3. Вычислить значение $v = (h(m_1))^{q-2} \pmod q$, что является не чем иным, как мультипликативным обратным к $h(m_1) \pmod q$. Вообще говоря, алгоритм проверки можно несколько ускорить, если вычислять $h(m_1)^{-1} \pmod q$ с помощью расширенного алгоритма Евклида, а не путем возведения в степень.
4. Вычислить значения:
   $z_1 = sv \pmod q$ и
   $z_2 = (q - r)v \pmod q$
5. Вычислить значение
   $u = (a^{z_1} y^{z_2} \pmod p) \pmod q$
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 сообще-

ния, т.е. $m_1 = m$. В противном случае подпись считается недействительной.

4.2.4. Алгоритм цифровой подписи, основанный на эллиптических кривых

Берется эллиптическая кривая E в форме Вейерштрасса над простым полем. Задается величина $J(E)$, называемая *инвариантом* эллиптической кривой:

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

Коэффициенты a и b кривой E определяются по известному инварианту следующим образом:

$$\begin{cases} a \equiv 3k \pmod{p}, \\ b \equiv 2k \pmod{p}, \text{ где } k \equiv \dfrac{J(E)}{1728 - J(E)} \pmod{p}, \quad J(E) \neq 0, 1728 \end{cases}.$$

Точку Q будем называть точкой кратности k, $k \in \boldsymbol{Z}$, если для некоторой точки P выполнено равенство $Q = kP$.

Параметрами схемы ЭЦП являются следующие значения:

p – модуль эллиптической кривой, простое число, $p > 2^{255}$;

эллиптическая кривая, задаваемая инвариантом $J(E)$ или коэффициентами a и b;

целое число m - порядок группы точек эллиптической кривой E;

простое число q - порядок циклической подгруппы группы точек эллиптической кривой E, для которого выполнены следующие условия:

$$\begin{cases} m = nq, \ \ n \in Z, \quad n \geq 1 \\ 2^{255} < q < 2^{256} \end{cases}.$$

базовая точка $P \neq \infty$ на кривой, имеющая порядок q, т. е. удовлетворяющая равенству $qP = \infty$. Координаты этой точки обозначим через (x_p, y_p);

хэш-функция, отображающая сообщения произвольной длины в множество двоичных векторов длины 256. Хэш-функция устанавливается стандартом Р 34.11-94.

Каждый пользователь схемы ЭЦП должен обладать личной ключевой парой:

секретный ключ пользователя – целое число d, $0 < d < q$;

открытый ключ пользователя – точка Q с координатами (x_q, y_q), удовлетворяющая равенству $dP = Q$.

Параметры ЭЦП должны удовлетворять следующим условиям:

$p^t \neq 1 \pmod q$, для всех целых t=1,2, … B, где B удовлетворяет неравенству $B \geq 11$;

$m \neq p$;

$J(E) = 0, 1728$.

ЭЦП под сообщением M вырабатывается по следующему алгоритму:

1. Вычислить хэш-функцию сообщения $M : \bar{h} = h(M)$.

2. Вычислить целое число α, двоичным представлением которого является вектор $\bar{h}$, и определить

$$e \equiv \alpha \pmod q .$$

3. Вычислить случайное целое число k, удовлетворяющее неравенствам

$$0 < k < q;$$

4. Вычислить точку эллиптической кривой $C=kP$ и положить

$$r \equiv x_c \pmod q,$$

где x_c – x-координата точки C. Если $r = 0$, то вернуться на шаг 3.

5. Вычислить значение

$$s \equiv (rd + ke) \pmod q.$$

Если $s = 0$, то вернуться на шаг 3.

6. Вычислить двоичные вектора, соответствующие числам r и s. Определить цифровую подпись $\zeta = (\bar{r} \| \bar{s})$ как конкатенацию двух двоичных векторов.

Для проверки подписи ζ под полученным сообщением M необходимо выполнить следующие действия.

1. По полученной подписи ζ вычислить целые числа r и s. Если выполнены неравенства $0 < r < q$, $0 < s < q$, то перейти к следующему шагу. В противном случае подпись не верна.

2. Вычислить хэш-функцию полученного сообщения $M : \bar{h} = h(M)$.

3. Вычислить целое число α, двоичным представлением которого является вектор $\bar{h}$ и определить $e \equiv \alpha \pmod{q}$.

4. Вычислить значение $v \equiv e^{-1} \pmod{q}$.

5. Вычислить значения

$$z_1 \equiv sv \pmod{q}, \quad z_2 \equiv -rv \pmod{q}.$$

6. Вычислить точку эллиптической кривой $C = z_1 P + z_2 Q$ и определить

$$R \equiv x_c \pmod{q},$$

где x_c – x-координата точки C.

7. Если выполнено равенство $R = r$, то подпись принимается, в противном случае подпись не верна.

4.2.5. 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:

$$Ver(t, s, k_V) = \begin{cases} 1, E_s(X_t) = C_t \\ 0, E_s(X_t) \neq C_t \end{cases}.$$

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:

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

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:

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

Алгоритм Ver_{n_T} проверки подписи состоит в проверке истинности соотношений $R_{2^{n_T}-1-T}(s_0) = C_0, R_T(s_1) = C_1$, 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:

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

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 $T \le T'$, 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:

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

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:

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

при выработке и проверке подписи оно вдвое меньше:

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

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:

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

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

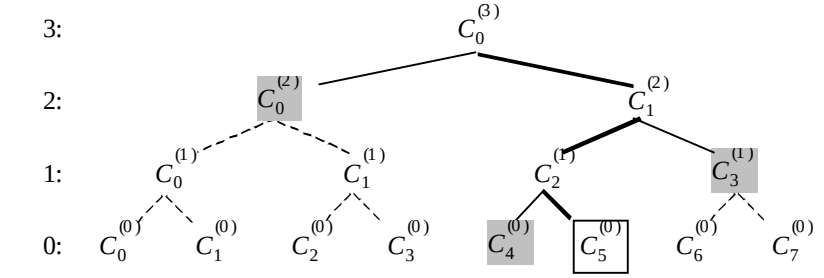


Рис. 4.1. Двоичное дерево для схемы ЭЦП на 8 сообщений

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.

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:

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

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 (MDC – Manipulation Detection Code) или проверкой целостности сообщения (MIC – Message Integrity Check)).

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.

Классическая хэш-функция является открытым преобразованием. В специальном случае, когда она зависит от ключа, результат ее вычисления носит название *кода аутентификации сообщения* (MAC – Message Authentication Code).

4.3.1. 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 = EFCDAB89
C = 98BADCFE
D = 10325476
E = C3D2E1F0,

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.

4.3.2. Функции хэширования SHA-256, SHA-512 и SHA-384

Стойкость функции хэширования к поиску столкновений примерно равна $2^{n/2}$, где *n* – длина выходного значения функции. В связи с разработкой в США нового стандарта шифрования с длиной ключа 128, 192 и 256 бит, потребовалось создать "сопровождающие" алгоритмы, обеспечивающие такой же уровень стойкости. В этом пункте описаны алгоритмы вычисления функций хэширования с длиной выходного значения 256, 512 и 384 бита, которые предполагается принять в качестве нового стандарта США.

Рассмотрим вначале алгоритм SHA-256. Его описание можно разбить на две части – описание функции сжатия и алгоритма обработки сообщения. Функция сжатия представляет собой по

сути алгоритм блочного шифрования с размером блока 256 бит промежуточного значения функции хэширования с использованием очередного текстового блока в качестве ключа. Помимо обычных обозначений, при описании функции мы будем использовать следующие: R^n – сдвиг слова вправо на n позиций и S^n – циклический сдвиг слова вправо n позиций. Размер слова равен 32 битам. Сложение производится по модулю 2^{32}. Стартовый вектор хэширования $H^{(0)}$ представляет собой набор из 8 32-разрядных слов, получаемых взятием дробной части квадратных корней первых 8 простых чисел:

$H^{(0)}$ = {6a09e667, bb67ae85, 3c6ef372, a54ff53a, 510e527f, 9b05688c, 1f83d9ab, 5be0cd19}.

Далее вычисление происходит по следующей схеме.

1. Предварительная обработка. Хэшируемое сообщение вначале дополняется битовой строкой так, что его длина становится кратной 512 битам. Способ дополнения аналогичен использованному в алгоритме SHA-1: добавляется 1, затем столько нулей, чтобы длина стала на 64 меньше кратной 512 и затем добавляется 64-битовое представление длины исходного сообщения.

2. Сообщение разбивается на блоки по 512 бит $M^{(1)}$, $M^{(2)}$, …, $M^{(N)}$.

3. Основной цикл:

for i = 1 to N { //N = количество блоков в дополненном сообщении

// Инициализировать регистры a, b, c, d, e, f, g, h $(i-1)$-м
// промежуточным значением хэш-функции

$a = H_1^{(i-1)}; b = H_2^{(i-1)}; c = H_3^{(i-1)}; d = H_4^{(i-1)};$

$e = H_5^{(i-1)}; f = H_6^{(i-1)}; g = H_7^{(i-1)}; h = H_8^{(i-1)};$

Применить функцию сжатия SHA-256 к регистрам a, b; ..., h.

for j = 0 to 63 {

Вычислить $Ch(e, f, g)$, $Maj(a, b, c)$, $\Sigma_0(a)$, $\Sigma_1(e)$, and W_j

// Определения см. ниже

$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_j + W_j$

$T_2 = \Sigma_0(a) + Maj(a, b, c)$

$h = g; g = f; f = e; e = d + T_1$

$d = c$; $c = b$; $b = a$; $a = T_1 + T_2$

}

// вычислить i-е промежуточное значение $H^{(i)}$.

$H_1^{(i)} = a + H_1^{(i-1)}; H_2^{(i)} = b + H_2^{(i-1)}; \text{K}; H_8^{(i)} = h + H_8^{(i-1)}$

}

$H^{(N)} = (H_1^{(N)}, H_2^{(N)}, H_3^{(N)}, H_4^{(N)}, H_5^{(N)}, H_6^{(N)}, H_7^{(N)}, H_8^{(N)})$ и будет искомым значением хэш-функции сообщения M.

В SHA-256 используются шесть логических функций, аргументы и значения которых – 32-битовые слова.

$Ch(x; y; z) = (x \wedge y) \oplus (\neg \text{x} \wedge \text{z})$

$Maj(x, y, z) = (\text{x} \wedge \text{y}) \oplus (\text{x} \wedge \text{z}) \oplus (\text{y} \wedge \text{z})$

$\Sigma_0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$

$\Sigma_1(\text{x}) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x)$

$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$

$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$

Блоки расширенного сообщения $W_0, \ldots, W_{63}$ вычисляются следующим образом:

$W_j = M_j^{(i)}, j = 0,...,15;$

for j = 16 to 63 {

$W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$

}

Слова-константы $K_0, \ldots, K_{63}$ берутся как первые 32 бита дробных частей кубических корней первых 64 простых чисел, и в 16-ричном виде представлены здесь:

428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5 d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174 e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da 983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967 27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85 a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070 19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3 748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2

Функция SHA-512 подобна по своей структуре SHA-256, но работает с размером слова 64 бита. Вначале текст дополняется

так, чтобы его длина была кратна 1024. Процедура дополнения аналогична: добавляется 1, затем столько нулей, что длина текста станет на 128 меньше, нежели кратная 1024, а затем 128-битовое представление длины исходного текста.

Стартовый вектор хэширования также задается аналогично: берутся первые 64 бита дробных частей квадратных корней первых 8 простых чисел:

H(0) = {6a09e667f3bcc908, bb67ae8584caa73b, 3c6ef372fe94f82b, a54ff53a5f1d36f1, 510e527fade682d1, 9b05688c2b3e6c1f, 1f83d9abfb41bd6b, 5be0cd19137e2179}.

Далее исходный текст разбивается на блоки по 1024 бита $M^{(1)}$, $M^{(2)}$, …, $M^{(N)}$. После чего блоки обрабатываются последовательно.

Основной цикл вычислений выглядит точно так же, как и в случае SHA-256, только вычисляемые функции и производимые операции определены на 64-битовых словах, а не на 32-битовых (в частности, сложение выполнятся по модулю 2^{64}). Функция сжатия отличается только количеством итераций в цикле:

for j = 0 to 79 {

Вычислить $Ch(e, f, g)$, $Maj(a, b, c)$, $\Sigma_0(a)$, $\Sigma_1(e)$, and W_j

$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_j + W_j$

$T_2 = \Sigma_0(a) + Maj(a, b, c)$

$h = g$; $g = f$; $f = e$; $e = d + T_1$

$d = c$; $c = b$; $b = a$; $a = T_1 + T_2$

}

Искомым значением хэш-функции исходного сообщения M будет $H^{(N)} = (H_1^{(N)}, H_2^{(N)}, H_3^{(N)}, H_4^{(N)}, H_5^{(N)}, H_6^{(N)}, H_7^{(N)}, H_8^{(N)})$.

Отличия есть только в определении некоторых используемых логических функций, а именно:

$\Sigma_0(x) = S^{28}(x) \oplus S^{34}(x) \oplus S^{39}(x)$

$\Sigma_1(x) = S^{14}(x) \oplus S^{18}(x) \oplus S^{41}(x)$

$\sigma_0(x) = S^{1}(x) \oplus S^{8}(x) \oplus R^{7}(x)$

$\sigma_1(x) = S^{19}(x) \oplus S^{61}(x) \oplus R^{6}(x)$

Блоки расширенного сообщения W_0, …, W_{79} вычисляются аналогично SHA-256:

$W_j = M_j^{(i)}$, $j = 0,...,15$;

for j = 16 to 79 {

$W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$

}

Слова-константы K_0, …, K_{79} берутся как первые 64 бита дробных частей кубических корней первых 80 простых чисел, и в 16-ричном виде представлены здесь:

428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240ca1cc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273eceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817.

Функция SHA-384 определяется точно так же, как и функция SHA-512 с тем исключением, что в качестве стартового вектора хэширования берутся первые 64 бита квадратных корней простых чисел с девятого по шестнадцатое:

H(0) = {cbbb9d5dc1059ed8, 629a292a367cd507, 9159015a3070dd17, 152fecd8f70e5939, 67332667ffc00b31, 8eb44a8768581511, db0c2e0d64f98fa7, 47b5481dbefa4fa4}.

Далее выход функции обрезается до 384 левых бит и эти биты берутся в качестве значения функции хэширования SHA-384.

4.3.3. 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 κ, 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 ψ^i – *i*-я степень преобразования ψ.

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.

4.3.4. Функция хэширования MD5

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-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 = 89abcdef,
C = fedcba98,
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 ∨ ¬(X) Z
G(X,Y,Z) = XZ ∨ Y ¬(Z)
H(X,Y,Z) = X ⊕ Y ⊕ Z
I(X,Y,Z) = Y ⊕ (X ∨ ¬(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
end /* 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.

5. УПРАВЛЕНИЕ КРИПТОГРАФИЧЕСКИМИ КЛЮЧАМИ

отсутствует в электронной версии

6. ИМИТОЗАЩИТА ИНФОРМАЦИИ В АСУ

отсутствует в электронной версии

7. ВОПРОСЫ РЕАЛИЗАЦИИ КРИПТОГРАФИЧЕСКИХ СИСТЕМ

отсутствует в электронной версии

ЛИТЕРАТУРА

1. Баричев С.Г., Гончаров В.В., Серов Р.Е. Основы современной криптографии. М., "Горячая линия – Телеком", 2001.
2. Варфоломеев А.А., Жуков А.Е., Пудовкина М.А. Поточные криптосистемы. Основные свойства и методы анализа стойкости. М., ПАИМС, 2000.
3. Введение в криптографию. Под общ. ред. Ященко В.В. М., МЦНМО-ЧеРо, 1998.
4. Герасименко В.А. Защита информации в автоматизированных системах обработки данных., кн. 1, 2. М., Энергоатомиздат, 1994.
5. ГОСТ 28147-89.
6. ГОСТ Р 34.10-94.
7. ГОСТ Р 34.10-94.
8. Конхейм А. Г. Основы криптографии. М., Радио и связь, 1987.
9. Мафтик С. Механизмы защиты в сетях ЭВМ. М., Мир, 1993.
10. Мельников В.В. Защита информации в компьютерных системах. М., Финансы и статистика, 1997.
11. Молдовян А.А., Молдовян Н.А., Советов Б.Я. Криптография. СПб., "Лань", 2000.
12. Молдовян Н.А. Скоростные блочные шифры. СПб, Издательство СПбГУ, 1998.
13. Нечаев В.И. Элементы криптографии (Основы теории защиты информации). М., Высшая школа, 1999.
14. Основы криптозащиты АСУ. Под ред. Б.П. Козлова. М., МО, 1996.
15. Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. М., Радио и связь, 1999.
16. Ухлинов А.М. Управление безопасностью информации в автоматизированных системах. М. МИФИ, 1996.