Лекція 13. Аутентифікація інформації. Алгоритми хешування та цифровий підпис.

Хэш-функции

Требования к хэш-функциям

Хэш-функцией называется односторонняя функция, предназначенная для получения профиля или "отпечатков пальцев" файла, сообщения или некоторого блока данных.

Хэш-код создается функцией H:

$h = H(M)$

Где M является сообщением произвольной длины и h является хэш-кодом фиксированной длины.

Рассмотрим *требования*, которым должна соответствовать хэш-функция для того, чтобы она могла использоваться в качестве аутентификатора сообщения. Рассмотрим очень простой пример хэш-функции. Затем проанализируем несколько подходов к построению хэш-функции.

Хэш-функция H, которая используется для аутентификации сообщений, должна обладать следующими *свойствами*:

1. Хэш-функция H должна применяться к блоку данных любой длины.
2. Хэш-функция H создает выход фиксированной длины.
3. $H(M)$ относительно легко (за полиномиальное время) вычисляется для любого значения M, а алгоритм вычисления должен быть практичным с точки зрения как аппаратной, так и программной реализации.
4. Для любого данного значения хэш-кода h вычислительно невозможно найти M такое, что $H(M) = h$. Такое свойство иногда называют **односторонностью**.
5. Для любого данного блока x вычислительно невозможно найти $y \neq x$, для которого $H(x) = H(y)$. Такое свойство иногда называют **слабой сопротивляемостью коллизиям**.
6. Вычислительно невозможно найти произвольную пару различных значений x и y, для которых $H(x) = H(y)$. Такое свойство иногда называют **сильной сопротивляемостью коллизиям**.

Первые три свойства описывают требования, обеспечивающие возможность практического применения функции хэширования для аутентификации сообщений.

Четвертое свойство определяет требование односторонности хэш-функции: легко создать хэш-код по данному сообщению, но невозможно восстановить сообщение по данному хэш-коду. Это свойство важно, если аутентификация с использованием хэш-функции включает секретное значение. Само секретное значение может не посылаться, тем не менее, если хэш-функция не является односторонней, противник может легко раскрыть секретное значение следующим образом. При перехвате передачи атакующий получает сообщение M и хэш-код $C = H(S_{AB} \| M)$. Если атакующий может инвертировать хэш-функцию, то, следовательно, он может получить $S_{AB} \| M = H^{-1}(C)$. Так как атакующий теперь знает и M и $S_{AB} \| M$, получить S_{AB} совсем просто.

Пятое свойство гарантирует, что невозможно найти другое сообщение, чье значение хэш-функции совпадало бы со значением хэш-функции данного сообщения. Это предотвращает подделку аутентификатора при использовании зашифрованного хэш-кода. В данном случае противник может читать сообщение и, следовательно, создать его хэш-код. Но так как противник не владеет секретным ключом, он не имеет возможности изменить сообщение так, чтобы получатель этого не обнаружил. Если данное свойство не выполняется, атакующий имеет возможность выполнить следующую последовательность действий: перехватить сообщение и его зашифрованный хэш-код, вычислить хэш-код сообщения, создать альтернативное сообщение с тем же самым хэш-кодом, заменить исходное сообщение на поддельное. Поскольку хэш-коды этих сообщений совпадают, получатель не обнаружит подмены.

Шестое свойство защищает против класса атак, построенных на парадоксе задачи о днях рождения.

Простые хэш-функции

Все хэш-функции выполняются следующим образом. Входное значение (сообщение, файл и т.п.) рассматривается как последовательность n-битных блоков. Входное значение обрабатывается последовательно блок за блоком, и создается n-битное значение хэш-кода.

Одним из простейших примеров хэш-функции является побитный XOR каждого блока:

$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$,

где

C_i – i-ый бит хэш-кода,

$1 \le i \le n$.

m – число n-битных блоков

входа.

b_{ij} – i-ый бит в j-ом блоке.

$\oplus$ – операция XOR.

В результате получается хэш-код длины n, известный как продольный контроль четности. Такая процедура достаточно эффективна при случайных сбоях для проверки целостности данных.

Часто при использовании подобного продольного избыточного контроля для каждого блока выполняется однобитный циклический сдвиг после вычисления хэш-кода. Это можно описать следующим образом.

- Установить n-битный хэш-код в ноль.
- Для каждого n-битного блока данных выполнить следующие операции:
  - сдвинуть циклически текущий хэш-код влево на один бит;
  - выполнить операцию XOR для очередного блока и хэш-кода.

Это даст эффект "случайности" входа и уничтожит любую регулярность, которая присутствует во входных значениях.

Хотя второй вариант считается более предпочтительным для обеспечения целостности данных и предохранения от случайных сбоев, он не может использоваться для обнаружения преднамеренных модификаций передаваемых сообщений. Зная сообщение, атакующий легко может создать новое сообщение, которое имеет тот же самый хэш-код. Для этого следует подготовить альтернативное сообщение и затем присоединить к нему подходящий n-битный блок, который вместе с новым сообщением сформирует желаемый хэш-код.

Хотя простого XOR или ротационного XOR (RXOR) недостаточно, если целостность обеспечивается только зашифрованным хэш-кодом, а само сообщение не шифруется, подобная простая функция может использоваться, когда все сообщение и присоединенный к нему хэш-код шифруются. Но и в этом случае следует помнить о том, что подобная хэш-функция не может проследить за тем, чтобы при передаче последовательность блоков не изменилась.

Первоначальный стандарт, предложенный NIST, использовал простой XOR, который применялся к 64-битным блокам сообщения, затем все сообщение шифровалось, используя режим сцепления блоков (CBC).

"Парадокс дня рождения"

Прежде чем рассматривать более сложные хэш-функции, необходимо проанализировать одну конкретную атаку на простые хэш-функции.

Так называемый "парадокс дня рождения" состоит в следующем. Предположим, количество выходных значений хэш-функции H равна n. Каким должно быть число k, чтобы для конкретного значения X и значений $Y_1,\ldots,Y_k$ вероятность того, что хотя бы для одного Y_i выполнялось равенство

$$H(X)=H(Y)$$

была бы больше 0.5.

Для одного Y вероятность того, что $H(X)=H(Y)$, равна $1/n$.

Соответственно, вероятность того, что $H(X)\neq H(Y)$, равна $1-1/n$.

Если создать k значений, то вероятность того, что ни для одного из них не будет совпадений, равна произведению вероятностей, соответствующих одному значению, т.е. $(1-1/n)^k$.

Следовательно, вероятность, по крайней мере, одного совпадения равна

$$1-(1-1/n)^k$$

По формуле бинома Ньютона

$$(1-a)^k=1-k\cdot a+\left(\frac{k\cdot(k-1)}{2!}\right)\cdot a^2-\left(\frac{k\cdot(k-1)\cdot(k-2)}{3!}\right)\cdot a^3\ldots\approx 1-k\cdot a \text{ (для малых } a)$$

$$1-(1-1/n)^k=1-(1-k/n)=k/n=0.5\Rightarrow k=n/2$$

Таким образом, мы выяснили, что для m-битового хэш-кода достаточно выбрать 2^{m-1} сообщений, чтобы вероятность совпадения хэш-кодов была больше 0.5.

Теперь рассмотрим следующую задачу: обозначим $P(n,k)$ вероятность того, что в множестве из k элементов, каждый из которых может принимать n значений, есть хотя бы два с одинаковыми значениями. Чему должно быть равно k, чтобы $P(n,k)$ была бы больше 0.5?

Число различных способов выбора элементов таким образом, чтобы при этом не было дублей, равно

$$n\cdot(n-1)\cdot\ldots\cdot(n-k+1)=\frac{n!}{(n-k)!}$$

Всего возможных способов выбора элементов равно

n^k

Вероятность того, что дублей нет, равна

$$\frac{n!}{(n-k)!\cdot n^k}$$

Вероятность того, что есть дубли, соответственно равна

$$1-\frac{n!}{(n-k)!\cdot n^k}$$

$$P(n,k)=1-\frac{n!}{(n-k)!\cdot n^k}=1-\frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{n^k}=1-\left[\frac{n-1}{n}\cdot\frac{n-2}{n}\cdot\ldots\cdot\frac{n-k+1}{n}\right]=$$

$$=1-\left[\left(1-\frac{1}{n}\right)\cdot\left(1-\frac{2}{n}\right)\cdot\ldots\cdot\left(1-\frac{k-1}{n}\right)\right].$$

Известно, что

$(1-x)\le e^{-x}$ для всех $x\ge 0$.

Используя это неравенство, получаем:

$$P(n,k)>1-\left[\left(e^{-1/n}\right)\cdot\left(e^{-2/n}\right)\cdot\ldots\cdot\left(e^{-(k-1)/n}\right)\right]$$

$$P(n,k)>1-e^{-[(1/n)+(2/n)+\ldots+((k-1)/n)]}$$

$$P(n,k)>1-e^{-(k\cdot(k-1))/2\cdot n}$$

Для того, чтобы $P(n,k)$ была бы больше 0.5 требуется

$$1/2=1-e^{-(k\cdot(k-1))/2\cdot n}$$

$$2=e^{(k\cdot(k-1))/2\cdot n}$$

$$\ln(2)=\frac{k\cdot(k-1)}{2\cdot n}$$

Для больших k $k\cdot(k-1)\approx k^2$, тогда

$$k=\sqrt{2\cdot\ln(2)\cdot n}\approx 1.18\cdot\sqrt{n}\approx\sqrt{n}$$

Если хэш-код имеет длину m бит, т.е. принимает 2^m значений, то

$$k=\sqrt{2^m}=2^{m/2}$$

Подобный результат называется "парадоксом дня рождения", потому что в соответствии с приведенными выше рассуждениями для того, чтобы вероятность совпадения дней рождения у двух человек была больше 0.5, в группе должно быть всего 23 человека. Этот результат кажется удивительным, возможно, потому, что для каждого отдельного человека в группе вероятность того, что с его днем рождения совпадет день рождения кого-то другого в группе, достаточно мала. Но мы рассматриваем вероятность того, что для *какой-нибудь* пары людей в группе дни рождения совпадут. В группе из 23 человек имеется $\frac{23\cdot(23-1)}{2}=253$ различные пары, поэтому и вероятности такие высокие.

Вернемся к рассмотрению свойств хэш-функций. Предположим, что используется 64-битный хэш-код. Можно считать, что это вполне достаточная и, следовательно, безопасная длина для хэш-кода. Например, если зашифрованный хэш-код C передается с соответствующим незашифрованным сообщением M, то противнику необходимо будет найти M' такое, что

$$H(M)=H(M')$$

для того, чтобы подменить сообщение и обмануть получателя. В среднем противник должен перебрать 2^{63} сообщений для того, чтобы найти такое, у которого хэш-код равен перехваченному сообщению.

Тем не менее, возможны различного рода атаки, основанные на "парадоксе дня рождения". Возможна следующая стратегия:

1. Отправитель **A** готовится "подписать" сообщение с помощью присоединения к сообщению соответствующего m-битового значения MAC и шифрования этого значения MAC личным ключом **A** (см. рис. 8.5(в)).

2. Противник создает $2^{m/2}$ вариантов сообщения, которые все имеют, по сути, один смысл. Кроме того, противник подготавливает такое же количество сообщений, каждое из которых является вариантом поддельного сообщения и предназначено для замены настоящего.
3. Два набора сообщений сравниваются в поисках пары сообщений, имеющих одинаковый хэш-код. Вероятность успеха в соответствии с "парадоксом дня рождения" больше, чем 0.5. Если соответствующая пара не найдена, то создаются дополнительные исходные и поддельные сообщения до тех пор, пока не будет найдена пара.
4. Атакующий предлагает отправителю исходный вариант сообщения для подписи пользователю **A**. Эта подпись может быть затем присоединена к поддельному варианту для передачи получателю. Так как оба варианта имеют один и тот же хэш-код, будет создана одинаковая подпись. Противник будет уверен в успехе, даже не зная ключа шифрования.

Таким образом, если используется 64-битный хэш-код, то необходимая сложность вычислений составляет порядка 2^{32}.

В заключение отметим, что длина хэш-кода должна быть достаточно большой. Длина, равная 64 битам, в настоящее время не считается безопасной. Предпочтительнее, чтобы длина составляла порядка 100 битов.

Использование цепочки зашифрованных блоков

Существуют различные хэш-функции, основанные на создании цепочки зашифрованных блоков, но без использования секретного ключа. Одна из таких хэш-функций была предложена Рабином. Сообщение M разбивается на блоки фиксированной длины $M_1, M_2, \ldots, M_N$ и используется алгоритм симметричного шифрования, например DES, для вычисления хэш-кода G следующим образом:

$H_0 =$ начальное значение

$H_i = E_{Mi}\left[H_{i-1}\right]$

$G = H_N$

Это аналогично использованию шифрования в режиме CBC, но в данном случае секретного ключа нет. Как и в случае любой простой хэш-функции, этот алгоритм подвержен "атаке дня рождения", и если шифрующим алгоритмом является DES и создается только 64-битный хэш-код, то система считается достаточно уязвимой.

Могут осуществляться другие атаки типа "дня рождения", которые возможны даже в том случае, если противник имеет доступ только к одному сообщению и соответствующему ему зашифрованному хэш-коду и не может получить несколько пар сообщений и зашифрованных хэш-кодов. Возможен следующий сценарий: предположим, что противник перехватил сообщение с аутентификатором в виде зашифрованного хэш-кода, и известно, что незашифрованный хэш-код имеет длину m битов. Далее противник должен выполнить следующие действия:

- Используя описанный выше алгоритм, вычислить незашифрованный хэш-код G.
- Создать поддельное сообщение в виде $Q_1, Q_2, \ldots, Q_{N-2}$.
- Вычислить $H_i = E_{Qi}\left[H_{i-1}\right]$ для $1 \le i \le (N-2)$.
- Создать $2^{m/2}$ случайных блоков, для каждого такого блока X вычислить $E_X\left[H_{N-2}\right]$. Создать дополнительно $2^{m/2}$ случайных блоков Y и для каждого такого блока вычислить $D_Y\left[G\right]$, где D – дешифрующая функция, соответствующая E.
- Основываясь на "парадоксе дня рождения" можно сказать, что с высокой степенью вероятности будут существовать такие блоки X и Y, что $E_X\left[H_{N-2}\right] = D_Y\left[G\right]$.
- Создать сообщение $Q_1, Q_2, \ldots, Q_{N-2}, X, Y$. Это сообщение имеет хэш-код G и, следовательно, может быть использовано вместе с зашифрованным аутентификатором.

Эта форма атаки известна как атака "встреча посередине". В различных исследованиях предлагаются усовершенствованные варианты этого сценария, усиливающие преимущества, лежащего в его основе сцепления блоков. Например, Девис и Прайс описали следующий вариант:

$H_i = E_{Mi}\left[H_{i-1}\right] \oplus H_{i-1}$.

Возможен другой вариант:

$H_i = E_{Hi-1}\left[M_i\right] \oplus M_i$.

Однако обе эти схемы также имеют уязвимости при различных атаках. В общем случае, можно показать, что некоторая форма "атаки дня рождения" имеет успех при любом хэш-алгоритме, включающем использование цепочки шифрованных блоков без применения секретного ключа.

Дальнейшие исследования были направлены на поиск других подходов к созданию функций хэширования.

Хэш-функция MD5

Рассмотрим алгоритм получения профиля сообщения MD5 (RFC 1321), разработанный Роном Райвестом из MIT.

Логика выполнения MD5

Алгоритм получает на входе сообщение произвольной длины и создает в качестве выхода профиль сообщения длиной 128 бит. Алгоритм состоит из следующих шагов:

Рис. 8.1. Логика выполнения MD5

Шаг 1: добавление недостающих битов

Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина $448 \bmod 512$). Это означает, что длина добавленного сообщения на 64 бита меньше, чем число, кратное 512. Добавление производится всегда, даже если сообщение имеет нужную длину. Например, если длина сообщения 448 битов, оно дополняется 512 битами до 960 битов. Таким образом, число добавляемых битов находится в диапазоне от 1 до 512.

Добавление состоит из единицы, за которой следует необходимое количество нулей.

Шаг 2: добавление значения длины

64-битное значение длины исходного (до добавления) сообщения в битах присоединяется к результату первого шага. Если первоначальная длина больше, чем 2^{64}, то используются только последние 64 бита. Таким образом, поле содержит длину исходного сообщения по модулю 2^{64}.

В результате первых двух шагов создается сообщение, длина которого кратна 512 битам. Это расширенное сообщение представляется как последовательность 512-битных блоков $Y_0, Y_1, \ldots, Y_{L-1}$, при этом общая длина расширенного сообщения равна $L \cdot 512$ битам. Таким образом, длина полученного расширенного сообщения кратна шестнадцати 32-битным словам.

Рис. 8.2. Структура расширенного сообщения

Шаг 3: инициализация MD-буфера

Используется 128-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер может быть представлен как четыре 32-битных регистра (A, B, C, D). Эти регистры инициализируются следующими шестнадцатеричными числами:

A = 01234567
B = 89ABCDEF
C = FEDCBA98
D = 76543210

Шаг 4: обработка последовательности 512-битных (16-словных) блоков

Основой алгоритма является функция сжатия – модуль, состоящий из четырех циклических обработок, обозначенный как H_{MD5}. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую f_F, f_G, f_H, f_I соответственно.

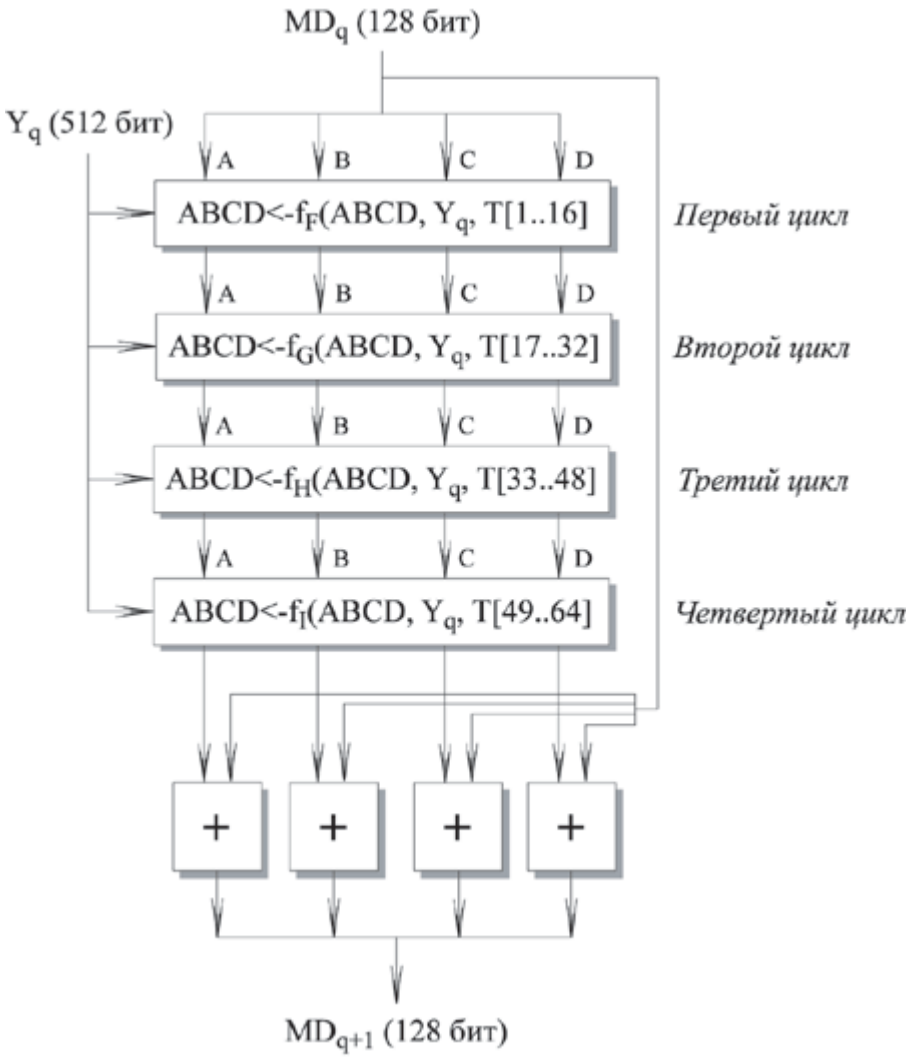

Рис. 8.3. Обработка очередного 512-битного блока

Каждый цикл принимает в качестве входа текущий 512-битный блок Y_q, обрабатывающийся в данный момент, и 128-битное значение буфера ABCD, которое является промежуточным значением профиля, и изменяет содержимое этого буфера. Каждый цикл также использует четвертую часть 64-элементной таблицы $T[1\ldots64]$, составленной из значений функции синуса. В таблице T i-ый элемент, обозначаемый $T[i]$, имеет значение, равное целой части числа $2^{32}\cdot|\sin(i)|$, где i задано в радианах. Так как $|\sin(i)|$ лежит в диапазоне между 0 и 1, каждый элемент T является целым, которое может быть представлено 32 битами. Таблица обеспечивает "случайный" набор 32-битных значений, которые должны ликвидировать любую регулярность во входных данных.

Для получения MD_{q+1} выход четырех циклов складывается по модулю 2^{32} с MD_q. Сложение выполняется независимо для каждого из четырех слов в буфере.

Шаг 5: выход

После обработки всех L 512-битных блоков выходом L-ой стадии является 128-битный профиль сообщения.

Рассмотрим более детально логику каждого из четырех циклов выполнения одного 512-битного блока. Каждый цикл состоит из 16 шагов, оперирующих с буфером ABCD. Каждый шаг можно представить в виде:

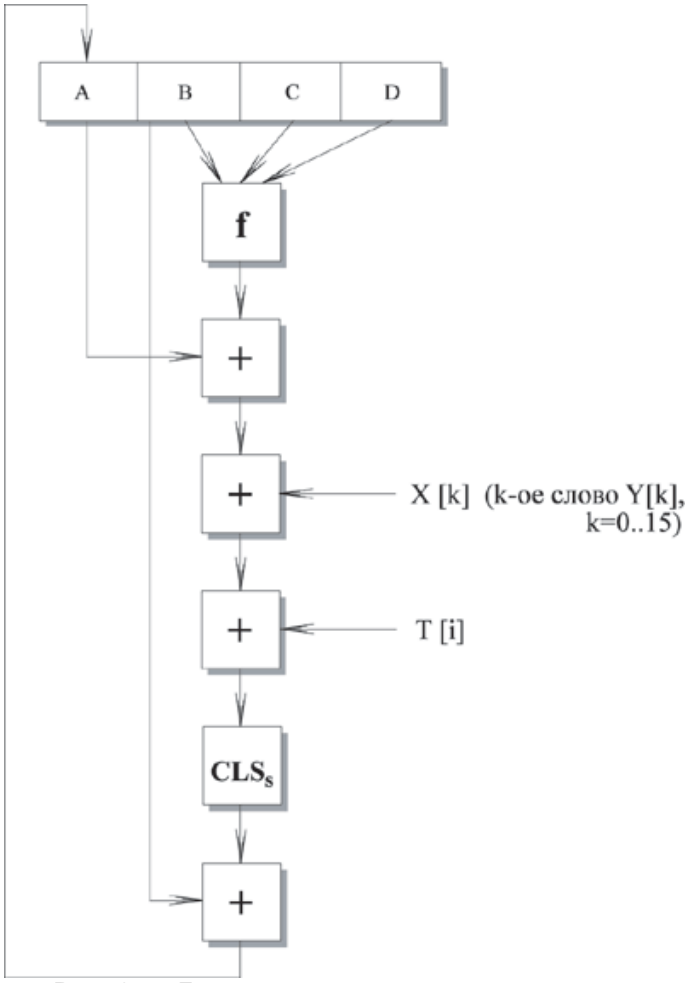

Рис. 8.4. Логика выполнения отдельного шага

$$A \leftarrow B + CLS_S\left(A + f\left(B,C,D\right) + X\left[k\right] + T\left[i\right]\right)$$

где

A, B, C, D – четыре слова буфера; после выполнения каждого отдельного шага происходит циклический сдвиг влево на одно слово.

f – одна из элементарных функций f_F, f_G, f_H, f_I.

CLS_S – циклический сдвиг влево на S битов 32-битного аргумента.

$X\left[k\right]$ – k-ое 32-битное слово в q-ом 512-битовом блоке сообщения, т.е. $M\left[q \cdot 16 + k\right]$.

$T\left[i\right]$ – i-ое 32-битное слово в матрице T.

$+$ – сложение по модулю 2^{32}.

На каждом из четырех циклов алгоритма используется одна из четырех элементарных логических функций. Каждая элементарная функция получает три 32-битных слова на входе и на выходе создает одно 32-битное слово. Каждая функция является множеством побитовых логических операций, т.е. n-ый бит выхода является функцией от n-ых битов трех входных значений. Элементарные функции следующие:

$$f_F = \left(B \cap C\right) \cup \left(\overline{B} \cap D\right)$$
$$f_G = \left(B \cap D\right) \cup \left(C \cap \overline{D}\right)$$
$$f_H = B \oplus C \oplus D$$
$$f_I = C \oplus \left(B \cup \overline{D}\right)$$

Тут логічні операції AND, OR, NOT, XOR представлені символами $\cap$, $\cup$, $\overline{\ }$, $\oplus$ відповідно.

Массив из 32-битных слов $X\left[0\ldots15\right]$ содержит значение 512-битного входного блока, который обрабатывается в настоящий момент. В рамках раунда каждое из 16 слов $X\left[0\ldots15\right]$ применяется ровно один раз, для одного шага, а порядок, в котором эти слова используются, меняется от раунда к раунду.

Каждый из 64 элементов T длиной в 32-битовое слово используется только один раз, для одного шага одного раунда. Обратите внимание и на то, что в результате выполнения одного шага раунда обновляется только 4 байта буфера ABCD. Следовательно, на протяжении раунда каждый байт буфера обновляется четыре раза и еще один раз в конце, когда получается конечное выходное значение блока. Наконец заметим, что в каждом раунде используются разные последовательности четырех циклических сдвигов влево с различными величинами сдвига. Целью всей этой сложности является обеспечение практической невозможности отыскать коллизии (пары 512-битовых блоков, порождающих одинаковый вывод).

Можно суммировать алгоритм MD5 следующим образом:

$$MD_0 = IV,$$

$$MD_{q+1} = MD_q + f_I\left[Y_q, f_H\left[Y_q, f_G\left[Y_q, f_F\left[Y_q, MD_q\right]\right]\right]\right],$$

$$MD = MD_L,$$

где

IV – начальное значение буфера ABCD, определенное на шаге 3,

Y_q – q-ый 512-битный блок сообщения,

L – число блоков в сообщении (включая поля дополнения и длины),

MD – окончательное значение профиля сообщения.

Стойкость алгоритма MD5

Алгоритм MD5 имеет следующее свойство: каждый бит хэш-кода является функцией от каждого бита входа. Сложное многократное использование элементарных функций f_F, f_G, f_H, f_I обеспечивает то, что результат хорошо перемешан; то есть маловероятно, чтобы два сообщения, выбранные случайно, даже если они имеют явно похожие закономерности, имели одинаковый хэш-код. Считается, что MD5 является наиболее сильной хэш-функцией для 128-битного хэш-кода, то есть трудность нахождения двух сообщений, имеющих одинаковый профиль, имеет порядок 2^{64} операций. В то время, как трудность нахождения сообщения с данным профилем имеет порядок 2^{128} операций.

Несколько результатов попыток атаковать MD5, тем не менее, заслуживают внимания.

1. Показано, что используя дифференциальный криптоанализ, можно за разумное время найти два сообщения, которые создают один и тот же профиль при использовании только одного цикла MD5. Подобный результат можно продемонстрировать для каждого из четырех циклов. Однако обобщить эту атаку на полный алгоритм MD5 из четырех циклов пока не удалось.
2. Существует способ выбора блока сообщения и двух соответствующих ему промежуточных значений профиля, которые создают одно и то же выходное значение. Это означает, что выполнение MD5 над таким отдельным блоком из 512 бит приведет к одинаковому выходу для двух различных входных значений в буфере ABCD. Такую ситуацию называют *псевдоколлизией*. Пока способа расширения данного подхода для успешной атаки на MD5 не существует.
3. Заслуживает наибольшего внимания техника, позволяющая генерировать коллизию для функции сжатия MD5, т.е. атака строится на рассмотрении действия MD5 на отдельном 512-битовом блоке и нахождении другого блока, дающего такое же 128-битовое выходное значение. На данный момент не известно способа обобщить этот подход на случай полного сообщения и начального значения (IV) алгоритма MD5. Тем не менее, этот подход может быть весьма близким к успеху.

Таким образом, с точки зрения криптоанализа MD5 должен теперь считаться уязвимым. Кроме того, если речь идет об атаках с перебором всех вариантов, MD5 сегодня уязвим в отношении атак, использующих парадокс задачи о днях рождения, требующих в данном случае усилий порядка 2^{64}. В результате необходимо заменить популярный алгоритм MD5 функцией хэширования, имеющей хэш-коды большей длины и более устойчивой в отношении известных методов криптоанализа. Двумя реальными на сегодняшний день кандидатами представляются алгоритмы SHA-1 и RIPEMD-160.

2. Коды аутентичности сообщений – MAC

Код аутентичности сообщения MAC, называемый также криптографической контрольной суммой, генерируется функцией C вида

$$MAC = C_K(M),$$

где M обозначает сообщение переменной длины, K – секретный ключ, известный только отправителю и получателю, а $C_K(M)$ – аутентификатор фиксированной длины. Значение MAC присоединяется к сообщению в пункте отправления в то время, когда известно или предполагается, что

данное сообщение корректно. Получатель удостоверяется в правильности сообщения путем повторного вычисления значения MAC.

2.1. Необходимые свойства кодов аутентичности сообщений

Когда в целях обеспечения конфиденциальности методами традиционного шифрования или шифрования с открытым ключом шифруется все сообщение, степень защищенности всей схемы в целом зависит от длины ключа. Если исключить возможность ущербности алгоритма, то противнику ничего не остается, как только выполнять анализ с перебором всех возможных ключей. В среднем такой анализ потребует $2^{(k-1)}$ попыток в случае k-битового ключа. Так, например, для анализа при наличии только шифрованного текста противник, имея шифрованный текст C, должен будет осуществлять поиск $P_i = D_{Ki}(C)$ для всех возможных значений ключа K_i, пока не будет найдено значение P_i, по форме соответствующее приемлемому открытому тексту.

Что касается кодов аутентичности сообщений, то здесь ситуация совершенно другая. Вообще, функция вычисления MAC отображает множество значений в одно. Область определения функции складывается из сообщений произвольной длины, тогда как область значений задается всеми возможными значениями MAC и всеми возможными ключами. Если используются n-битовые значения MAC, то таких значений всего 2^n, тогда как число возможных сообщений равно N, где $N >> 2^n$. Кроме того, имеется 2^k возможностей для k-битового ключа.

Используя метод простого перебора, как должен действовать противник, чтобы найти ключ? Если конфиденциальность не предполагается, то противник имеет доступ к открытым сообщениям и соответствующим им значениям MAC. Предположим, что $k > n$, т.е. длина ключа больше длины MAC. Тогда, имея известные M_1 и MAC_1, а также зная, что $MAC_1 = C_K(M_1)$, криптоаналитик может вычислить $MAC_i = C_{Ki}(M_1)$ для всех возможных значений ключа K_i. При этом по крайней мере для одного из ключей будет получено совпадение $MAC_i = MAC_1$. Обратите внимание на то, что противник получит для значений MAC 2^k вариантов, тогда как для них имеется только $2^n < 2^k$ различных значений. Таким образом, подходящее значение MAC будет соответствовать целому ряду ключей и противник не сможет узнать, какой из этих ключей является правильным. В среднем совпадение будет иметь место для $2^k / 2^n = 2^{(k-n)}$ ключей. Поэтому противнику придется итерировать свои усилия.

■ Раунд 1

- Даны M_1, $MAC_1 = C_K(M_1)$
- Вычисляется $MAC_i = C_{Ki}(M_1)$ для всех 2^k вариантов ключей
- Число совпадений равно $\approx 2^{(k-n)}$

■ Раунд 2

- Даны M_2, $MAC_2 = C_K(M_2)$
- Вычисляется $MAC_i = C_{Ki}(M_2)$ для оставшихся $2^{(k-n)}$ вариантов ключей
- Число совпадений равно $\approx 2^{(k-2\cdot n)}$

и т.д. В среднем потребуется a раундов вычисления, если $k = a \cdot n$. Например, если используется 80-битовый ключ и длина значения MAC равна 32 битам, то в первом раунде будет получено около 2^{48} возможных ключей. Во втором раунде это число сократится до 2^{16}. Третий раунд должен дать только один ключ, который и будет, по идее, тем ключом, который использовал отправитель.

Если длина ключа меньше или равна длине значения MAC, то, скорее всего, уже в первом раунде будет получено единственное совпадение. Возможно и то, что совпадений окажется более одного, и в таком случае противнику придется выполнить тот же тест с новой парой (сообщение, MAC).

Таким образом, попытка перебора всех вариантов для поиска ключа аутентификации потребует не меньше, а вероятно, и больше усилий, чем попытка поиска ключа шифрования той же длины.

Функция вычисления MAC должна иметь следующие свойства.

1. Если противник получит в свое распоряжение M и $C_K(M)$, для него должно быть практически невозможно вычислить сообщение M', для которого $C_K(M') = C_K(M)$.

2. Значения $C_K(M)$ должны быть равномерно распределенными в том смысле, что для случайно выбранных сообщений M и M' вероятность того, что $C_K(M') = C_K(M)$, должна быть равна 2^{-n}, где n равно числу битов значения MAC.

3. Пусть M' получается в результате некоторой известной трансформации M, т.е. $M' = f(M)$. Например, f может означать изменение на противоположный одного или нескольких заданных битов. В

таком случае вероятность того, что $C_K(M') = C_K(M)$, должна быть равна 2^{-n}.

Первое из этих требований соответствует примеру, в котором противник имеет возможность создать новое сообщение, соответствующее заданному значению MAC, даже если он не знает или не желает искать ключ. Требование (2) имеет своей целью воспрепятствовать анализу с перебором всех вариантов при наличии избранного открытого текста. Иначе говоря, если противник не знает K, но имеет доступ к функции вычисления MAC и может представить ей сообщения для получения соответствующих значений MAC, то он может проверять различные сообщения до тех пор, пока не обнаружит сообщения, соответствующего данному значению MAC. Если функция вычисления MAC имеет однородное распределение, то метод перебора должен потребовать для этого в среднем $2^{(n-1)}$ попыток.

Последнее из требований гарантирует, что алгоритм аутентификации не будет слабее для одних частей или битов сообщения, чем для других. Если бы это было не так, то противник, который получит M и $C_K(M)$, может попытаться изменить M в сторону известного "слабого места", где можно быстрее обнаружить сообщение, соответствующее имеющемуся значению MAC.

2.2. Коды аутентичности сообщений на основе DES

Одна из наиболее широко используемых функций вычисления значений MAC, называемая алгоритмом аутентификации данных (Data Authentication Algorithm), опирается на использование DES. Описание этого алгоритма существует и как публикация FIPS (PUB FIPS 113), и в виде стандарта ANSI (X9.17).

Алгоритм можно определить как шифрование DES в режиме сцепления блоков (CBC) с нулевым вектором инициализации. Данные (например, сообщение, запись, файл или программа), которым требуется аутентификация, представляются в виде последовательности 64-битовых блоков $D_1, D_2, \ldots, D_N$. При необходимости, конечный блок дополняется справа нулями, чтобы образовался полный 64-битовый блок. Используя алгоритм E шифрования DES и секретный ключ K, код аутентичности данных (DAC – Data Authentication Code) можно вычислить следующим образом (рис. 8.6):

$$O_1 = E_K(D_1),$$
$$O_2 = E_K(D_2 \oplus O_1),$$
$$O_3 = E_K(D_3 \oplus O_2),$$
$$\ldots$$
$$O_N = E_K(D_N \oplus O_{N-1}).$$

Значение DAC представляет собой либо весь блок O_N, либо крайние слева M битов этого блока, где $16 \le M \le 64$.

Оказывается, что этот алгоритм отвечает всем сформулированным выше требованиям.

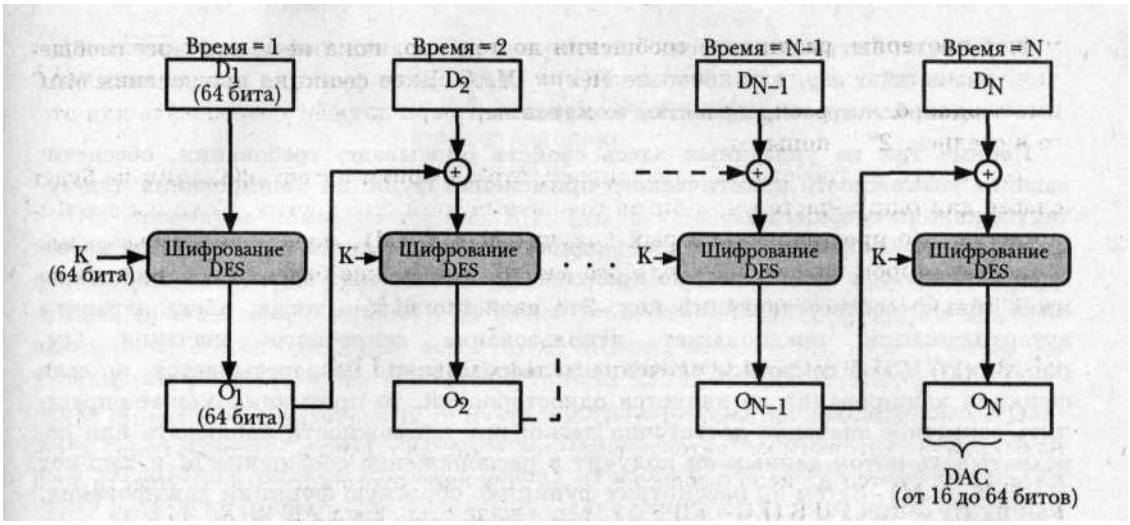

Рис. 8.6. Алгоритм аутентификации данных (PUB FIPS 113)

2.3. HMAC

Мы рассмотрели пример алгоритма вычисления кода аутентичности сообщения (MAC), основанного на использовании симметричного блочного шифра. Традиционно применяемый этим алгоритмом подход был наиболее общим подходом к созданию MAC. В последние годы возрос интерес к методу вычисления MAC на основе криптографических хэш-кодов. Объясняется это следующими обстоятельствами.

- Криптографические функции хэширования, например MD5 и SHA-1, в виде программной реализации обычно выполняются быстрее, чем симметричные блочные шифры, например DES.
- Программные библиотеки для криптографических функций хэширования широко доступны.
- Для криптографических функций хэширования нет ограничений на экспорт из США или других стран, тогда как для симметричных блочных шифров они предусмотрены, даже в случае использования шифра для вычисления значений MAC.

Такой алгоритм носит название HMAC, и он описан в RFC 2104.

В алгоритме HMAC хэш-функция представляет собой "черный ящик". Это, во-первых, позволяет использовать существующие реализации хэш-функций, а во-вторых, обеспечивает легкую замену существующей хэш-функции на новую.

Введем следующие обозначения:

H – встроенная хэш-функция.

b – длина блока используемой хэш-функции.

n – длина хэш-кода.

K – секретный ключ. К этому ключу слева добавляют нули, чтобы получить b-битовый ключ K^+.

Вводится два вспомогательных значения:

$ipad$ – значение '00110110', повторенное b/8 раз.

$opad$ – значение '01011010', повторенное b/8 раз.

Далее HMAC вычисляется следующим образом:

$$HMAC_K = H\left[\left(K^+ \oplus opad\right) \| H\left(K^+ \oplus ipad\right) \| M\right].$$

Подробно описать алгоритм можно следующим образом.

1. К значению K слева добавляются нули, чтобы получить b-битовую строку K^+.
2. Значение K^+ связывается операцией XOR с $ipad$, в результате чего получается b-битовый блок S_i.
3. К S_i присоединяется M.
4. К потоку, полученному на шаге 3, применяется функция H.
5. Значение K^+ связывается операцией XOR с $opad$, в результате чего получается b-битовый блок S_o.
6. Результат хэширования, полученный на шаге 4, присоединяется к S_o.
7. К потоку, полученному на шаге 6, применяется функция H, и результат подается на выход.

Обратите внимание на то, что связывание с $ipad$ (шаг 3) означает переключение половины битов K. Точно так же и связывание с $opad$ означает переключение половины битов K, но для другого набора битов. В действительности в результате пропускания S_i и S_o через функцию сжатия алгоритма хэширования из K получается два ключа, сгенерированных псевдослучайным образом.

3. Цифровая подпись

3.1. Требования к цифровой подписи

Аутентификация защищает двух участников, которые обмениваются сообщениями, от воздействия некоторой третьей стороны. Однако простая аутентификация не защищает участников друг от друга, тогда как и между ними тоже могут возникать определенные формы споров.

Например, предположим, что Джон посылает Мери аутентифицированное сообщение, используя одну из схем, показанных на рис. 8.4 (см. Лекцию 11). Рассмотрим возможные конфликты, которые могут при этом возникнуть:

- Мери может подделать сообщение и утверждать, что оно пришло от Джона. Мери достаточно просто создать сообщение и присоединить аутентификационный код, используя ключ, который разделяют Джон и Мери.
- Джон может отрицать, что он посылал сообщение Мери. Так как Мери может подделать сообщение, у нее нет способа доказать, что Джон действительно посылал его.

В ситуации, когда обе стороны не доверяют друг другу, необходимо нечто большее, чем аутентификация на основе общего секрета. Возможным решением подобной проблемы является использование цифровой подписи. Цифровая подпись должна обладать следующими свойствами:

1. Должна быть возможность проверить автора, дату и время создания подписи.
2. Должна быть возможность установить достоверность содержимого сообщения на время создания подписи.
3. Подпись должна быть проверяема третьей стороной для разрешения споров.

Таким образом, функция цифровой подписи включает, в частности, и функцию аутентификации.

На основании этих свойств можно сформулировать следующие требования к цифровой подписи:

1. Подпись должна быть битовым образцом, который зависит от подписываемого сообщения.
2. Подпись должна использовать некоторую уникальную информацию отправителя для предотвращения подделки или отказа.
3. Создавать цифровую подпись должно быть относительно легко.
4. Должно быть вычислительно невозможно подделать цифровую подпись как созданием нового сообщения для существующей цифровой подписи, так и созданием ложной цифровой подписи для некоторого сообщения.
5. Цифровая подпись должна быть достаточно компактной и не занимать много памяти.

Защищенная хэш-функция, встроенная в соответствующую схему, подобную показанным на рис. 8.5(в) или 8.5(г) (см. Лекцию 11), удовлетворяет этим требованиям.

Существует несколько подходов к использованию функции цифровой подписи. Все они могут быть разделены на две категории: прямые и арбитражные.

3.2. Прямая и арбитражная цифровые подписи

При использовании прямой цифровой подписи взаимодействуют только сами участники, т.е. отправитель и получатель. Предполагается, что получатель знает открытый ключ отправителя. Цифровая подпись может быть создана шифрованием всего сообщения (рис. 8.1(в)) или его хэш-кода закрытым ключом отправителя (рис. 8.5(в)).

Конфиденциальность может быть обеспечена дальнейшим шифрованием всего сообщения вместе с подписью открытым ключом получателя (асимметричное шифрование) или разделяемым секретным ключом (симметричное шифрование) (см. рис. 8.1(г) и 8.5(г)). Заметим, что обычно функция подписи выполняется первой, и только после этого выполняется функция конфиденциальности. В случае возникновения спора некая третья сторона должна просмотреть сообщение и его подпись. Если функция подписи выполняется над зашифрованным сообщением, то для разрешения споров придется хранить сообщение как в незашифрованном виде (для практического использования), так и в зашифрованном (для проверки подписи). Либо в этом случае необходимо хранить ключ симметричного шифрования, для того чтобы можно было проверить подпись исходного сообщения. Если цифровая подпись выполняется над незашифрованным сообщением, получатель может хранить только сообщение в незашифрованном виде и соответствующую подпись к нему.

Все прямые схемы, рассматриваемые далее, имеют общее слабое место. Действенность схемы зависит от безопасности закрытого ключа отправителя. Если отправитель впоследствии не захочет признать факт отправки сообщения, он может утверждать, что закрытый ключ был потерян или украден, и в результате кто-то подделал его подпись. Можно применить административное управление, обеспечивающее безопасность закрытых ключей, для того чтобы, по крайней мере, хоть в какой-то степени ослабить эти угрозы. Один из возможных способов состоит в требовании в каждую подпись сообщения включать отметку времени (дату и время) и сообщать о скомпрометированных ключах в специальный центр.

Другая угроза состоит в том, что закрытый ключ может быть действительно украден у X в момент времени T. Нарушитель может затем послать сообщение, подписанное подписью X и помеченное временной меткой, которая меньше или равна T.

Проблемы, связанные с прямой цифровой подписью, могут быть частично решены с помощью арбитра. Существуют различные схемы с применением арбитражной подписи. В общем виде арбитражная подпись выполняется следующим образом. Каждое подписанное сообщение от отправителя X к получателю Y первым делом поступает к арбитру A, который проверяет подпись для данного сообщения. После этого сообщение датируется и посылается к Y с указанием того, что оно было проверено арбитром. Присутствие A решает проблему схем прямой цифровой подписи, при которых X может отказаться от сообщения.

Арбитр играет исключительно важную роль в подобного рода схемах, и все участники должны ему доверять.

Рассмотрим некоторые возможные технологии арбитражной цифровой подписи.

3.2.1. Симметричное шифрование, арбитр видит сообщение:

$(1)\quad X \to A:\ \ M \,\|\, E_{Kxa}\left[ID_X \,\|\, H(M)\right]$

Предполагается, что отправитель X и арбитр A разделяют секретный ключ K_{XA} и что A и Y разделяют секретный ключ K_{AY}. X создает сообщение M и вычисляет его хэш-значение $H(M)$.

Затем X передает сообщение и подпись A. Подпись состоит из идентификатора X и хэш-значения, все зашифровано с использованием ключа K_{XA}. A дешифрует подпись и проверяет хэш-значение.

$$(2)\quad A \rightarrow Y:\ E_{Kay}\left[ID_X \parallel M \parallel E_{Kxa}\left[ID_X \parallel H(M)\right] \parallel T\right]$$

Затем A передает сообщение к Y, шифруя его K_{AY}. Сообщение включает ID_X, первоначальное сообщение от X, подпись и отметку времени. Y может дешифровать его для получения сообщения и подписи. Отметка времени информирует Y о том, что данное сообщение не устарело и не является повтором. Y может сохранить M и подпись к нему. В случае спора Y, который утверждает, что получил сообщение M от X, посылает следующее сообщение к A:

$$E_{Kay}\left[ID_X \parallel M \parallel E_{Kxa}\left[ID_X \parallel H(M)\right]\right]$$

Арбитр использует K_{AY} для получения ID_X, M и подписи, а затем, используя K_{XA}, может дешифровать подпись и проверить хэш-код. По этой схеме Y не может прямо проверить подпись X; подпись используется исключительно для разрешения споров. Y считает сообщение от X аутентифицированным, потому что оно прошло через A. В данном сценарии обе стороны должны иметь высокую степень доверия к A:

1. X должен доверять A в том, что тот не будет раскрывать K_{XA} и создавать фальшивые подписи в форме $E_{Kxa}\left[ID_X \parallel H(M)\right]$.
2. Y должен доверять A в том, что он будет посылать $E_{Kay}\left[ID_X \parallel M \parallel E_{Kxa}\left[ID_X \parallel H(M)\right] \parallel T\right]$ только в том случае, если хэш-значение является корректным и подпись действительно была создана X.
3. Обе стороны должны доверять A в решении спорных вопросов.

3.2.2. Симметричное шифрование, арбитр не видит сообщение:

Если арбитр не является такой доверенной стороной, то X должен добиться того, чтобы никто не мог подделать его подпись, а Y должен добиться того, чтобы X не мог отвергнуть свою подпись.

Предыдущий сценарий также предполагает, что A имеет возможность читать сообщения от X к Y и что возможно любое подсматривание. Рассмотрим сценарий, который, как и прежде, использует арбитраж, но при этом еще обеспечивает конфиденциальность. В таком случае также предполагается, что X и Y разделяют секретный ключ K_{XY}.

$$(1)\quad X \rightarrow A:\ ID_X \parallel E_{Kxy}\left[M\right] \parallel E_{Kxa}\left[ID_X \parallel H\left(E_{Kxy}\left[M\right]\right)\right]$$

X передает A свой идентификатор, сообщение, зашифрованное K_{XY}, и подпись. Подпись состоит из идентификатора и хэш-значения зашифрованного сообщения, которые зашифрованы с использованием ключа K_{XA}. A дешифрует подпись и проверяет хэш-значение. В данном случае A работает только с зашифрованной версией сообщения, что предотвращает его чтение.

$$(2)\quad A \rightarrow Y:\ E_{Kay}\left[ID_X \parallel E_{Kxy}\left[M\right] \parallel E_{Kxa}\left[ID_X \parallel H\left(E_{Kxy}\left[M\right]\right)\right] \parallel T\right]$$

A передает Y все, что он получил от X плюс отметку времени, все шифруя с использованием ключа K_{AY}.

Хотя арбитр и не может прочитать сообщение, он в состоянии предотвратить подделку любого из участников, X или Y. Остается проблема, как и в первом сценарии, что арбитр может сговориться с отправителем, отрицающим подписанное сообщение, или с получателем, для подделки подписи отправителя.

3.2.3. Шифрование открытым ключом, арбитр не видит сообщение:

Все обсуждаемые проблемы могут быть решены с помощью схемы открытого ключа.

$$(1)\quad X \rightarrow A:\ ID_X \parallel E_{KRx}\left[ID_X \parallel E_{KUy}\left(E_{KRx}\left[M\right]\right)\right]$$

В этом случае X осуществляет двойное шифрование сообщения M, сначала своим закрытым ключом KR_X, а затем открытым ключом Y (KU_Y). Получается подписанная секретная версия сообщения. Теперь это подписанное сообщение вместе с идентификатором X шифруется KR_X и вместе с ID_X посылается A. Внутреннее, дважды зашифрованное, сообщение недоступно арбитру (и всем, исключая Y). Однако A может дешифровать внешнюю шифрацию, чтобы убедиться, что сообщение пришло от X (так как только X имеет KR_X). Проверка дает гарантию, что пара закрытый/открытый ключ законна, и тем самым верифицирует сообщение.

$$(2)\quad A \rightarrow Y:\ E_{KRa}\left[ID_X \parallel E_{KUy}\left(E_{KRx}\left[M\right]\right) \parallel T\right]$$

Затем A передает сообщение Y, шифруя его KR_A. Сообщение включает ID_X, дважды зашифрованное сообщение и отметку времени.

Эта схема имеет ряд преимуществ по сравнению с предыдущими двумя схемами. Во-первых, никакая информация не разделяется участниками до начала соединения, предотвращая договор об обмане. Во-вторых, некорректные данные не могут быть посланы, даже если KR_X скомпрометирован, при условии, что не скомпрометирован KR_A. В заключение, содержимое сообщения от X к Y неизвестно ни A, ни кому бы то ни было еще.

3.3. Стандарт цифровой подписи DSS

Национальный институт стандартов и технологии США (NIST) разработал федеральный стандарт цифровой подписи DSS. Для создания цифровой подписи используется алгоритм DSA (Digital Signature Algorithm). В качестве хэш-алгоритма стандарт предусматривает использование алгоритма SHA-1 (Secure Hash Algorithm). DSS первоначально был предложен в 1991 году и пересмотрен в 1993 году в ответ на публикации, касающиеся безопасности его схемы. В 1996 году в него были внесены незначительные изменения.

3.3.1. Подход DSS

DSS использует алгоритм, который разрабатывался для использования только в качестве цифровой подписи. В отличие от RSA, его нельзя использовать для шифрования или обмена ключами. Тем не менее, это технология открытого ключа.

Рассмотрим отличия подхода, используемого в DSS для создания цифровых подписей, от применения таких алгоритмов как RSA.

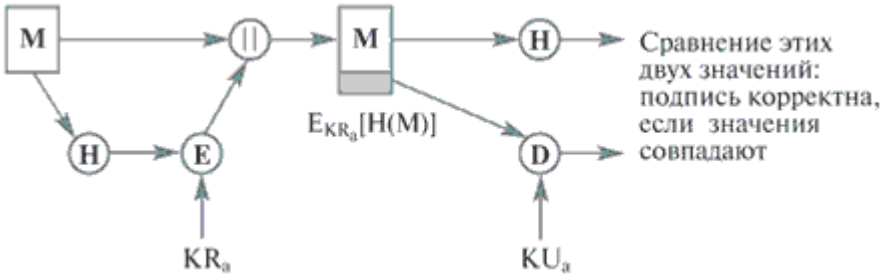

Рис. 10.1. Создание и проверка подписи с помощью алгоритма RSA

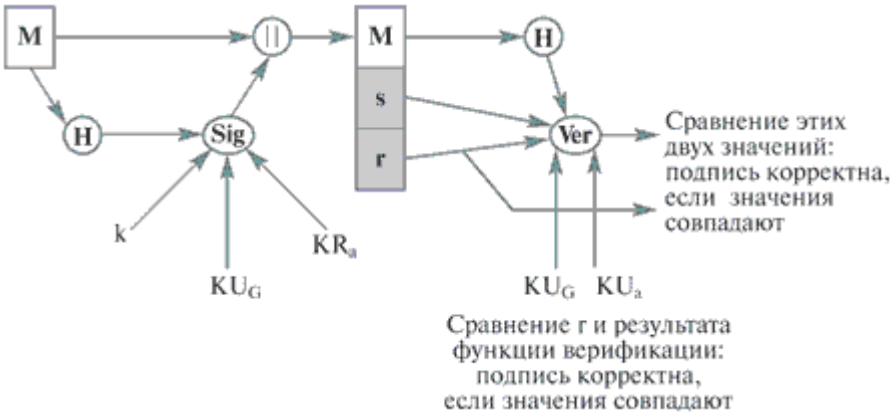

Рис. 10.2. Создание и проверка подписи с помощью стандарта DSS

В подходе RSA подписываемое сообщение подается на вход сильной хэш-функции, которая создает хэш-код фиксированной длины. Для создания подписи этот хэш-код шифруется с использованием закрытого ключа отправителя. Затем сообщение и подпись пересылаются получателю. Получатель вычисляет хэш-код сообщения и проверяет подпись, используя открытый ключ отправителя. Если вычисленный хэш-код равен дешифрованной подписи, то считается, что подпись корректна.

Подход DSS также использует сильную хэш-функцию. Хэш-код является входом функции подписи вместе со случайным числом k, созданным для этой конкретной подписи. Функция подписи также зависит от закрытого ключа отправителя KR_A и множества параметров, известных всем участникам. Можно считать, что это множество состоит из глобального открытого ключа KU_G. Результатом является подпись, состоящая из двух компонент, обозначенных как s и r.

Для проверки подписи получатель также создает хэш-код полученного сообщения. Этот хэш-код вместе с подписью является входом в функцию верификации. Функция верификации зависит от

глобального открытого ключа KU_G и от открытого ключа отправителя KU_A. Выходом функции верификации является значение, которое должно равняться компоненте r подписи, если подпись корректна. Функция подписи такова, что только отправитель, знающий закрытый ключ, может создать корректную подпись.

Теперь рассмотрим детали алгоритма, используемого в DSS.

3.3.2. Алгоритм цифровой подписи.

DSS основан на трудности вычисления дискретных логарифмов и базируется на схеме, первоначально представленной ElGamal и Schnorr.

Общие компоненты группы пользователей.

Существует три параметра, которые являются открытыми и могут быть общими для большой группы пользователей.

Выбирается 160-битное простое число q, т.е. $2^{159} < q < 2^{160}$.

Затем выбирается такое простое число p длиной между 512 и 1024 битами, чтобы q было делителем $(p-1)$.

Наконец, выбирается число g вида $h^{(p-1)/q} \bmod p$, где h является целым между 1 и $(p-1)$ с тем ограничением, что g должно быть больше, чем 1.

Зная эти числа, каждый пользователь выбирает закрытый ключ и создает открытый ключ.

Личный ключ отправителя.

Закрытый ключ x должен быть числом между 1 и $(p-1)$ и должен быть выбран случайно или псевдослучайно.

x – случайное или псевдослучайное целое, $0 < x < q$.

Открытый ключ отправителя.

Открытый ключ вычисляется из закрытого ключа по формуле $y = g^x \bmod p$. Вычислить y по известному x довольно просто. Однако, имея открытый ключ y, вычислительно невозможно определить x, который является дискретным логарифмом у по основанию g.

$$y = g^x \bmod p$$

Случайное число, уникальное для каждой подписи.

k – случайное или псевдослучайное целое, $0 < k < q$, уникальное для каждого выполнения подписи.

Подписывание.

Для создания подписи отправитель вычисляет две величины, r и s, которые являются функцией от компонент открытого ключа (p, q, g), закрытого ключа пользователя (x), хэш-кода сообщения $H(M)$ и целого k, уникального для каждой подписи.

$$r = (g^x \bmod p) \bmod q$$

$$s = [k^{-1} \cdot (H(M) + x \cdot r)] \bmod q$$

Подпись = (r, s)

Проверка подписи.

Получатель выполняет проверку подписи с использованием следующих формул. Он создает величину v, которая является функцией от компонент общего открытого ключа, открытого ключа отправителя и хэш-кода полученного сообщения. Если эта величина равна компоненте r в подписи, то подпись считается действительной.

$$w = s^{-1} \bmod q$$

$$u_1 = [H(M) \cdot w] \bmod q$$

$$u_2 = r \cdot w \bmod q$$

$$v = [(g^{u1} \cdot y^{u2}) \bmod p] \bmod q$$

подпись корректна, если $v = r$.

Обратите внимание на то, что проверка в конце осуществляется со значением r, которое не зависит от сообщения вообще. Значение r является функцией k и трех компонентов глобального открытого ключа.

При трудности вычисления дискретных логарифмов для противника оказывается нереальным с точки зрения вычислений найти k по известному r или найти x по известному s.