Практическая работа № 14

Однонаправленные хеш-функции в криптографии

Цель: Изучение алгоритмов построения и использования однонаправленных хеш-функций.

14.1. Краткие теоретические сведения

Согласно Шнайдеру Б. однонаправленная функция *H*(*M)* применяется к сообщению произвольной длины *M* и возвращает значение фиксированной длины *h,* то есть

h = H(*M),* где *h* имеет длину ***m***.

Многие функции позволяют вычислять значение фиксированной длины по входным данным произвольной длины, но у рассматриваемых хэш-функций есть дополнительные свойства, делающие их однонаправленными:

- ✓ Зная *M,* легко вычислить *h.*
- ✓ Зная *H,* трудно определить *M,* для которого *H(M)=h.*
- ✓ Зная *M,* трудно определить другое сообщение *M' ,* для которого *H*(*M) = H(M').*

То есть смысл однонаправленных хэш-функций и состоит в обеспечении для *M* уникального идентификатора ("отпечатка пальца"). Если Алиса подписала *M* с помощью алгоритма цифровой подписи на базе *H*(*M),* а Боб может создать *M',* другое сообщение, отличное от *M,* для которого *H*(*M) = H(M'),* то Боб сможет утверждать, что Алиса подписала *M'.*

В ряде приложений однонаправленности недостаточно, необходимо выполнение другого требования, называемого **устойчивостью к коллизиям.** Его можно сформулировать так: трудно найти два случайных сообщения *M* и *M',* для которых *H*(*M) = H(M').*

Проанализируем протокол, впервые описанный Гидеоном Ювалом, который показывает, как, если предыдущее требование не выполняется, Алиса может использовать вскрытие методом дня рождения для обмана Боба. Он основан не на поиске другого сообщения *M,* для которого *H*(*M) = H(M'),* а на поиске двух случайных сообщений, *M* и *M',* для которых *H*(*M) = H(M')* *(метод «дней рождений» смотри в кн. Шнайдера раздел 7.4)*.

1. Алиса готовит две версии контракта: одну, выгодную для Боба, и другую, приводящую его к банкротству
2. Алиса вносит несколько незначительных изменений в каждый документ и вычисляет хэш-функции.

(Этими изменениями могут быть действия, подобные следующим: замена ПРОБЕЛА комбинацией ПРОБЕЛ-ЗАБОЙ-ПРОБЕЛ, вставка одного-двух пробелов перед возвратом каретки, и т.д. Делая или не делая по одному изменению в каждой из 32 строк, Алиса может легко получить 2^{32} различных документов.)

3. Алиса сравнивает хэш-значения для каждого изменения в каждом из двух документов, разыскивая пару, для которой эти значения совпадают. (Если выходом хэш-функции является всего лишь 64-разрядное значение, Алиса, как правило, сможет найти совпадающую пару сравнив 2^{32} версий каждого документа.) Она восстанавливает два документа, дающих одинаковое хэш-значение.
4. Алиса получает подписанную Бобом выгодную для него версию контракта, используя протокол, в котором он подписывает только хэш-значение.
5. Спустя некоторое время Алиса подменяет контракт, подписанный Бобом, другим, который он не подписывал. Теперь она может убедить арбитра в том, что Боб подписал другой контракт.

Это серьезная проблема. Одним из советов по ее избежанию является обязательное внесение мелких изменений в подписываемый документ.

При возможности успешного вскрытия этим методом, могут применяться и другие способы вскрытия. Например, противник может посылать системе автоматического контроля (может быть спутниковой) случайные строки сообщений со случайными строками подписей. На каком-то этапе подпись под одним из этих случайных сообщений окажется правильной. Враг не сможет узнать, к чему приведет эта команда, но, если его единственной целью является вмешательство в работу спутника, он своего добьется.

Длины значений однонаправленных хэш-функций

64-битовые хэш-функции слишком малы, чтобы противостоять вскрытию методом *«дней рождений»*. Более практичны однонаправленные хэш-функции, выдающие 128-битовые хэш-значения. При этом, чтобы найти два документа с одинаковыми хэш-значениями, для вскрытия методом *«дней рождений»* придется хэшировать 2^{64} случайных документов, что, впрочем, недостаточно, если нужна длительная безопасность. Поэтому ряд стандартов безопасного хэширования (Secure Hash Standard, SHS), использует 160-битовое хэш-значение. Это еще сильнее усложняет вскрытие методом *«дней рождений»*, для которого понадобится 2^{80} хэширований.

Для удлинения хэн-значений, выдаваемых конкретной хэш-функцией, был предложен следующий метод:

1. Для сообщения с помощью однонаправленных хэш-функций генерируется хэш-значение.
2. Хэш-значение конкатенируется с сообщением.
3. Для конкатенации вычисляется новое хэш-значения.
4. Создается хэш-значение большей длины, состоящее из объединения хэш-значения этапа (1) и хэш-значения этапа (3).
5. Этапы (1)-(4) повторяются нужное количество раз для обеспечения требуемой длины хэш-значения.

Структура и алгоритмы MD-функций

Очевидно, не просто построить функцию, входные данные которой имеют произвольный размер, а тем более сделать ее однонаправленной. В реальности однонаправленные хэш-функции строятся на идее **функции сжатия.** Такая однонаправленная функция выдает хэш-значение длины n при заданных входных данных большей длины m . Входами функции сжатия являются блок сообщения и выход предыдущего блока текста. Выход представляет собой хэш-значение всех блоков, обработанных до этого момента. То есть, хэш-значение блока M_i, равно

$h_i = f(M_i , h_{i-1})$

Это хэш-значение вместе со следующим блоком сообщения становится следующим входом функции сжатия. Хэш-значением всего сообщения является хэш-значение последнего блока.

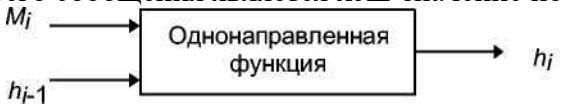


Рисунок 7.1 - Однонаправленная функция

Хэшируемый вход должен содержать бинарное представление длины всего сообщения. Таким образом преодолевается потенциальная проблема, вызванная тем, что сообщения различной длины могут давать одно и то же хэш-значение. Иногда такой метод называется **MD-усилением** (MD - Message Digest). Рассмотрим алгоритмы и свойства наиболее распространенных хеш-функций.

MD4 (Request for Comments: 1320 The MD4 Message-Digest Algorithm)

MD4 - это однонаправленная хэш-функция, изобретенная Роном Ривестом, алгоритм для входного сообщения выдает 128-битовое хэш-значение, которое называется дайджестом сообщения .

Цели, преследуемые при разработке алгоритма:

- ✓ *Безопасность.* Вычислительно невозможно найти два сообщения с одинаковым хэш-значением. Вскрытие грубой силой является самым эффективным.
- ✓ *Прямая безопасность.* Безопасность MD4 не основывается на каких-либо допущениях, например, предположении о трудности разложения на множители.
- ✓ *Скорость.* MD4 подходит для высокоскоростных программных реализаций. Она основана на простом наборе битовых манипуляций с 32-битовыми операндами.
- ✓ *Простота и компактность.* MD4 проста, насколько это возможно, и не содержит больших структур данных или сложных программных модулей.
- ✓ *Удачная архитектура.* MD4 оптимизирована для микропроцессорной архитектуры (особенно для микропроцессоров Intel), для более крупных и быстрых компьютеров можно выполнить любые необходимые изменения .

Описание алгоритма MD4

Пусть есть сообщение длиной **b** бит, для которого мы ищем MD. **b** может быть равно 0, оно не обязано быть кратно 8.

Представим его в виде: $m_0, m_1 \ldots m_{b-1}$

Расчет MD происходит за 5 шагов:

Шаг 1: Добавление байтов заполнителей:

Сообщение расширяется, так чтобы его длина (в битах) была 448 по модулю 512 – т.е. сообщению не хватает ровно 64 бит для кратности 512 битам. Добавление битов происходит в любом случае, даже если длина исходного сообщения изначально обладает этим свойством.

Добавление происходит следующим образом:

- первый добавленный бит – «1», остальные «0». Минимум 1 бит, максимум 512.

Шаг 2: Добавление длины:

64-битное значение b (длины исходного сообщения) добавляется к сообщению. Если же длина более 2^{64} (это маловероятно), то лишь младшие 64 бита длины используются.

Теперь сообщение имеет длину, кратную 512 битам (16 32-битных слова), его можно представить как слова: $M_0, M_1 \ldots M_{N-1}$, где N- кратно 16.

Шаг 3: Инициализация буфера MD

Буфер из четырех слов (A,B,C,D) , используется для расчета MD. Инициализируется:

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Шаг 4: Обработка сообщения блоками по 16 слов:

Сначала определим три вспомогательный функции, на вход получают 3 слова, результат – слово.

`F(X,Y,Z) = XY ∨ not(X) Z.`

`G(X,Y,Z) = XY ∨ XZ ∨ YZ.`

`H(X,Y,Z) = X xor Y xor Z,` где

`X ∨ Y` – побитовая операция `OR` над `X, Y.`

`X ⊕ Y` – побитовая операция исключающего ИЛИ,

`XY` – операция побитового умножения (AND) над `X` и `Y.`

Заметим, что в функции F каждый бит X действует как условие, если установлен берется соответсвующее значение бита из Y, иначе из Z (каждый бит F – Если X то Y, иначе Z).

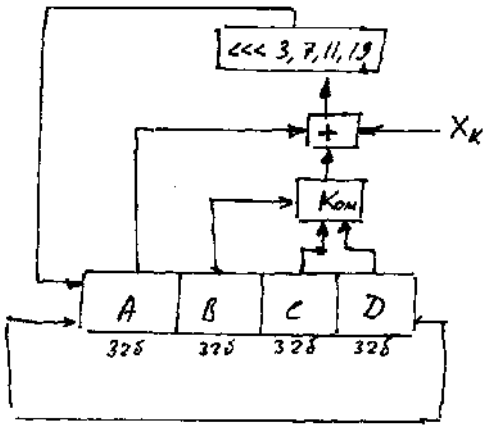
Функция G -есть функция побитового мажорирования (каждый бит G – Преобладающее значение X,Y,Z).

Функция H -функция побитовой операции XOR или функции четности (каждый бит H – xor, или контроль четности).

Делаем следующее:

```
/* Обрабатываем каждый блок в 16 слов. */
    For (i = 0;  i<= N/16-1; i++)
    {   /* Копируем блок i в X. */
        For (j = 0; j<=15; j++)
        {  X[j] = M[i*16+j].
         } /* цикла j */
 /* Сохраняем значения A как AA, B как BB, C как CC, и D как DD. */
      AA = A
      BB = B
      CC = C
      DD = D
 /* Раунд 1. */
 /* Пусть [abcd k s] определяют операцию
```

```
            a = (a + F(b,c,d) + X[k]) <<< s. */
```
/* Сделать следующие 16 операций. */
```
    [ABCD  0  3]  [DABC  1  7]  [CDAB  2 11]  [BCDA  3 19]
    [ABCD  4  3]  [DABC  5  7]  [CDAB  6 11]  [BCDA  7 19]
    [ABCD  8  3]  [DABC  9  7]  [CDAB 10 11]  [BCDA 11 19]
    [ABCD 12  3]  [DABC 13  7]  [CDAB 14 11]  [BCDA 15 19]
```
/*Несложно заметить, что ABCD циклически сдвигаются влево, k идет по строкам от 0 до 15, a s циклически замкнуто на {3,7,11,19}.*/



/* Раунд 2. */

/* Пусть [abcd k s] определяют операцию
```
            a = (a + G(b,c,d) + X[k] + 5A827999) <<< s. */
```
/* Сделать следующие 16 операций. */
```
    [ABCD  0  3]  [DABC  4  5]  [CDAB  8  9]  [BCDA 12 13]
    [ABCD  1  3]  [DABC  5  5]  [CDAB  9  9]  [BCDA 13 13]
    [ABCD  2  3]  [DABC  6  5]  [CDAB 10  9]  [BCDA 14 13]
    [ABCD  3  3]  [DABC  7  5]  [CDAB 11  9]  [BCDA 15 13]
```
/*Несложно заметить, что ABCD циклически сдвигаются влево, k идет по столбцам от 0 до 15, a s циклически замкнуто на {3,5,9,13}.*/

/* Раунд 3. */

/* Пусть [abcd k s] определяют операцию
```
            a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s. */
```
/* Сделать следующие 16 операций. */
```
    [ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
    [ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
    [ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
    [ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]
```
/*Несложно заметить, что ABCD циклически сдвигаются влево, a s циклически замкнуто на {3,9,11,15}, k также подвержено закономерности*/

/* Изменяем ABCD – регистры для следующего блока */
```
        A = A + AA
        B = B + BB
        C = C + CC
        D = D + DD
    } /* цикла по i */
```
Примечание:

Величина 5A827999 - 32-битная константа, записанная начиная со старшей цифры. Представляет квадратный корень из 2 со старшей цифрой в начале.

6ED9EBA1 - представляет собой квадратный корень из 3 со старшей цифрой в начале.

Шаг 5: Вывод

MD – это A, B, C, D – начиная с младшего байта A и заканчивая старшим байтом D.

(128 бит)

После первого появления алгоритма криптоаналитиками был совершен ряд вскрытий отдельных этапов MD4. Хотя все эти вскрытия не были распространены на полный алгоритм, Ривест усилил свою разработку. В результате появился алгоритм MD5.

MD5 (Request for Comments: 1321 MD5)

MD5 - это улучшенная версия MD4. Хотя она сложнее MD4, их схемы похожи, и результатом MD5 также является 128-битовое хэш-значение.

Описание MD5

После некоторой первоначальной обработки MD5 обрабатывает входной текст 512-битовыми блоками, разбитыми на 16 32-битовых подблоков. Выходом алгоритма является набор из четырех 32-битовых блоков, которые объединяются в единое 128-битовое хэш-значение.

Во-первых, сообщение дополняется так, чтобы его длина была на 64 бита короче числа, кратного 512. Этим дополнением является 1, за которой вплоть до конца сообщения следует столько нулей, сколько нужно. Затем, к результату добавляется 64-битовое представление длины сообщения (истинной, до дополнения). Эти два действия служат для того, чтобы длина сообщения была кратна 512 битам (что требуется для оставшейся части алгоритма), и чтобы гарантировать, что разные сообщения не будут выглядеть одинаково после дополнения. Инициализируются четыре переменных:

A = 0x01234567
B = 0x89abcdef
C = 0xfedcba98
D = 0x76543210

Они называются **переменными сцепления.**

Теперь перейдем к основному циклу алгоритма. Этот цикл продолжается, пока не исчерпаются 512-битовые блоки сообщения.

Четыре переменных копируются в другие переменные*: A в a, B в b, C в c и D в d.*

Главный цикл состоит из четырех очень похожих этапов. На каждом этапе 16 раз используются различные операции. Каждая операция представляет собой нелинейную функцию над тремя переменными из набора *a, b, c* и *d.* Затем она добавляет этот результат к четвертой переменной, подблоку текста и константе. Далее результат циклически сдвигается вправо на переменное число битов и добавляет результат к одной из переменных *a, b, c* и *d.* Наконец результат заменяет одну из переменных *a,b, c* и *d.* Существуют четыре нелинейных функции, используемые по одной в каждой операции (для каждого этапа - другая функция).

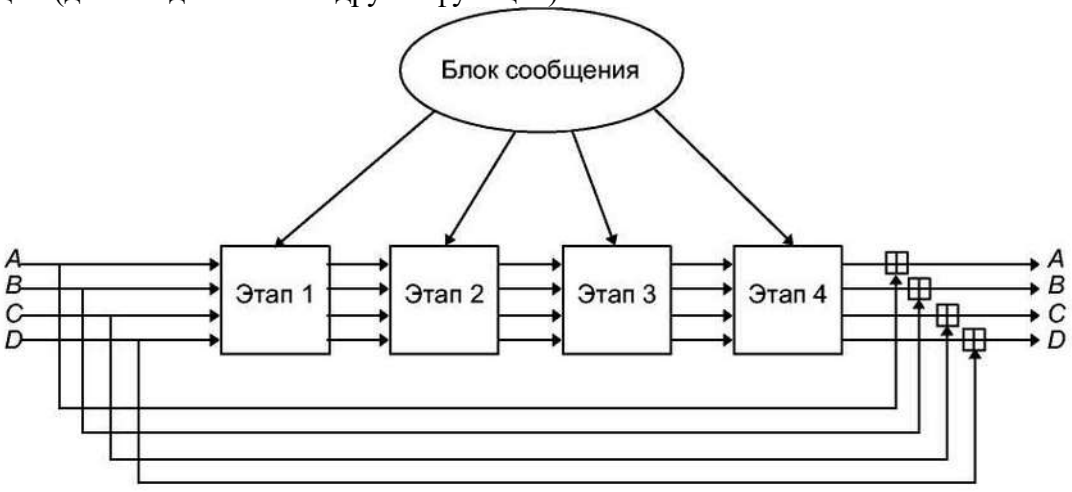


Рисунок 7.2 - Главный цикл MD5

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

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

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

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

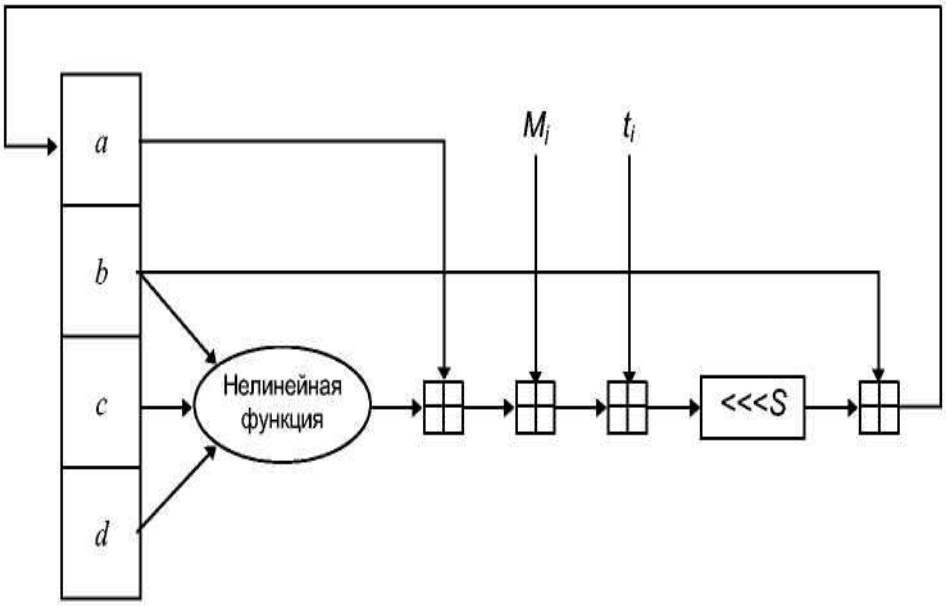


Рисунок 7.3 - Схема одной операции алгоритма MD5

Эти функции спроектированы так, чтобы, если соответствующие биты *X, Y* и Z независимы и несмещены, каждый бит результата также был бы независимым и несмещенным. Функция F - это побитовое условие: если *X,* то *Y,* иначе Z. Функция H - побитовая операция четности.

Если *Mj* обозначает j-ый подблок сообщения (от 0 до 15), а *<«s* обозначает циклический сдвиг влево на *s* битов, то используются следующие четыре операции:

FF(a,b,c,d,Mj,s,t$_{i}$) означает *a = b + ((a +* ¥*(b,c,d)* + M,- + *t$_{t}$) <«s)*

GG(a,b,c,dMj,s,t$_{i}$) означает *a = b + ((a + G(b,c,d) + Mj + t$_{t}$) <«s)*

HH(a,b,c,d,Mj,s,t$_{i}$) означает *a = b + ((a + H(b,c,d)* + M,- + *t$_{t}$) <«s)*

II{a,b,c,dMj,s,t$_{i}$) означает *a = b + ((a + I(b,c,d) + Mj + t$_{t}$) <«s)*

Четыре этапа алгоритма.

Шаг 1: Добавление байтов заполнителей:

Сообщение расширяется, так чтобы его длина (в битах) была 448 по модулю 512 – т.е. сообщению не хватает ровно 64 бит для кратности 512 битам. Добавление битов происходит в любом случае, даже если длина исходного сообщения изначально обладает этим свойством. Добавление происходит следующим образом:

- первый добавленный бит – «1», остальные «0». Минимум 1 бит, максимум 512.

Шаг 2: Добавление длины:

64 битное значение b (длины исходного сообщения) добавляется к сообщению. Если же длина более 2^64 (это маловероятно), то лишь младшие 64 бита длины используются.

Теперь сообщение имеет длину, кратную 512 битам (16 32 битных слова), его можно представить как слова:

$M_0, M_1 \ldots M_{N-1}$, где N- кратно 16

Шаг 3: Инициализация буфера MD:

Буфер из четырех слов (A,B,C,D), используется для расчета MD. Инициализируется:

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Шаг 4: Обработка сообщения блоками по 16 слов:

Сначала определим четыре вспомогательных функции, на вход получают 3 слова, результат – слово.

F(X,Y,Z) = XY $\vee$ not(X) Z

G(X,Y,Z) = XZ $\vee$ Y not(Z)

H(X,Y,Z) = X xor Y xor Z

I(X,Y,Z) = Y xor (X ∨ not(Z))
Каждый бит F – Если X то Y, иначе Z
Каждый бит G – Преобладающее значение X,Y,Z
Каждый бит H – xor, или контроль четности
Этот шаг использует специальную таблицу T[1..64], заполненную из значений синуса. T[i]=Целая часть 4294967296 * abs(sin(i)), где i в радианах.
Делаем следующее:
Обрабатываем каждый блок в 16 слов. */

```
For (i = 0;  i<= N/16-1; i++
 { /* Копируем блок i в X. */
  For (j = 0; j<=15; j++)
   { X[j] = M[i*16+j].
  } /* цикла j */
  /* Сохраняем значения A как AA, B как BB, C как CC, и D как DD. */
  AA = A
  BB = B
  CC = C
  DD = D
  /* Раунд 1. */
/* Пусть [abcd k s i] определяют операцию
     a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
  /* Делаем 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]

   /* Раунд 2. */
  /* Пусть [abcd k s i] определяют операцию
     a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
  /* Делаем 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]
   /* Раунд 3. */
  /* Пусть [abcd k s t] определяют операцию
     a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
  /* Делаем 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]

  /* Раунд 4. */
  /* Пусть [abcd k s t] определяют операцию
     a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
  /* Делаем 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]
```

/* Изменяем ABCD – регистры для следующего блока */

A = A + AA

B = B + BB

C = C + CC

D = D + DD

} /* цикла по i */

Шаг 5: Вывод

MD – это A,B,C,D – начиная с младшего байта A и заканчивая старшим байтом D (128 бит).

Безопасность MD5

Рон Ривест привел следующие улучшения MD5 в сравнении с MD4:

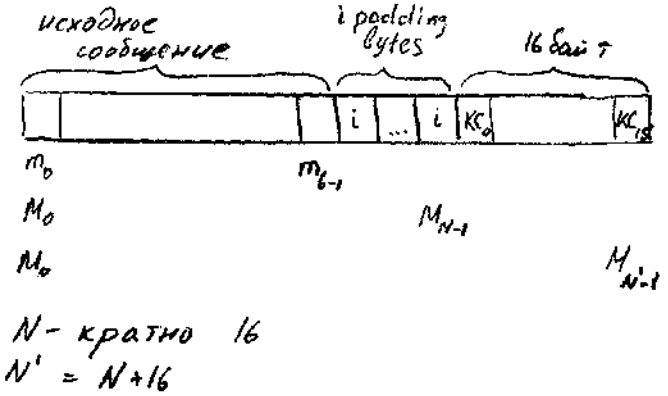
1. Добавился еще один этап.
2. Теперь в каждом действии используется уникальная прибавляемая константа.
1. Функция G на этапе 2 была изменена, чтобы сделать G менее симметричной.
3. Теперь каждое действие добавляется к результату предыдущего этапа. Это обеспечивает более быстрый лавинный эффект.
4. Изменился порядок, в котором использовались подблоки сообщения на этапах 2 и 3, чтобы сделать шаблоны менее похожими.
5. Значения циклического сдвига влево на каждом этапе были приближенно оптимизированы для ускорения лавинного эффекта. Четыре сдвига, используемые на каждом этапе, отличаются от значений, используемых на других этапах.

MD2 (Request for Comments: 1319 MD2)

MD2 - это другая 128-битовая однонаправленная хэш-функция, разработанная Роном Ривестом. Она вместе с MD5 используется для цифровой подписи в протоколах PEM. Безопасность MD2 опирается на случайную перестановку байтов. Эта перестановка фиксирована и зависит от цифр числа ***pi***. Идентификаторы *So, S1, S2, , S255* являются перестановкой.

Чтобы выполнить хэширование сообщения *M* необходимо*:*

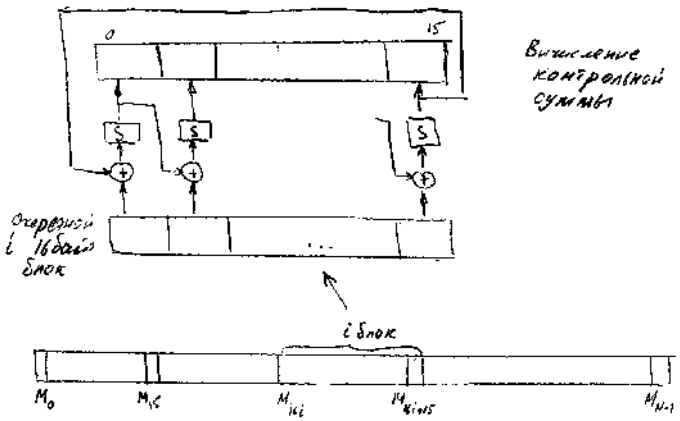
1. Дополнить сообщение *i* байтами, значение *i* должно быть таким, чтобы длина полученного сообщения была кратна 16 байтам.



2. Добавить к сообщению 16 байтов контрольной суммы.

При ее вычислении используется 256-байтовую "random" permutation, созданную из цифр числа π. Обозначим S[i] - i-тый элемент таблицы

```
L:= C0=…  =C15:=0              { Обнуляем контрольную сумму}
For (i = 0; i<= N/16-1; i++)   { обрабатываем каждый 16-байтовый блок}
  {   For (j = 0; j<=15; j++)        {Обрабатываем блок i}
      { Cj:= S[Mi*16+j ⊕ L].
        L:= Cj
      }
  }
```



Получаем 16-байтовую checksum C_0 ...C_{15} , добавляемую к сообщению.
Получим сообщение $M^*_0 \ldots M^*_{N'-1}$ с добавленной контрольной суммой, где N'=N + 16.

3. Проинициализировать 48-байтовый блок: $X_o, X_1, X_2, \ldots, X_{41}$. Заполнить первые 16 байтов X нулями, во вторые 16 байтов X скопировать первые 16 байтов сообщения, а третьи 16 байтов X должны быть равны XOR первых и вторых 16 байтов X.
4. Вот как выглядит функция сжатия:

$t = 0$
For ($j = 0$; $j <= 17$; j++)
  For ($k = 0$; $k <= 47$; k++)
    $t = X_k \ XOR \ S_t$,
$X_k = t$
$t = (t + j) \bmod 256$

5. Скопировать во вторые 16 байтов X вторые 16 байтов сообщения, а третьи 16 байтов X должны быть равны XOR первых и вторых 16 байтов X. Выполнить этап (4). Повторять этапы (3) и (4) по очереди для каждых 16 байтов сообщения.
6. Выходом являются первые 16 байтов X.

Хотя в MD2 пока не было найдено слабых мест, она работает медленнее большинства других предлагаемых хэш-функций.

Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA)

Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA), необходимый для обеспечения безопасности Алгоритма цифровой подписи (Digital Signature Algorithm, DSA). Для любого входного сообщения длиной меньше 2^{64} битов SHA выдает 160-битовый результат, называемый кратким содержанием сообщения. Далее, краткое содержание сообщения становится входом DSA, который вычисляет подпись для сообщения. Подписывание краткого содержания вместо всего сообщения часто повышает эффективность процесса, так как краткое содержание сообщения намного меньше, чем само сообщение. То же краткое содержание сообщения должно быть получено тем, кто проверяет подпись, если принятая им версия сообщения используется в качестве входа SHA. SHA называется безопасным, так как он разработан так, чтобы было вычислительно невозможно найти сообщение, соответствующее данному краткому содержанию сообщения или найти два различных сообщения с одинаковым кратким содержанием сообщения . Любые изменения, произошедшие при п ередаче сообщения, с очень высокой вероятностью приведут к изменению краткого содержания сообщения, и подпись не пройдет проверку. Принципы, лежащие в основе SHA, аналогичны использованным профессором Рональдом Л. Ривестом при проектировании алгоритма краткого содержания сообщения MD4. SHA разработан по образцу упомянутого алгоритма.

SHA выдает 160-битовое хэш-значение, более длинное, чем у MD5.

Описание SHA

Во-первых, сообщение дополняется, чтобы его длина была кратной 512 битам. Используется то же дополнение, что и в MD5: сначала добавляется 1, а затем нули так, чтобы длина полученного

сообщения была на 64 бита меньше числа, кратного 512, а затем добавляется 64-битовое представление длины оригинального сообщения.

Инициализируются пять 32-битовых переменных (в MD5 используется четыре переменных, но рассматриваемый алгоритм должен выдавать 160-битовое хэш-значение):

A = 0x67452301
B = 0xefcdab89
C = 0x10325476
D = 0x10325476
E = 0xc3d2elf0

Затем начинается главный цикл алгоритма. Он обрабатывает сообщение 512-битовыми блоками и продолжается, пока не исчерпаются все блоки сообщения.

Сначала пять переменных копируются в другие переменные*: A в a, B в b, C в c, D в d и E в e.*

Главный цикл состоит из четырех этапов по 20 операций в каждом (в MD5 четыре этапа по 16 операций в каждом). Каждая операция представляет собой нелинейную функцию над тремя из *a, b, c, d* и *e,* а затем выполняет сдвиг и сложение аналогично MD5. В SHA используется следующий набор нелинейных функций:

$f_t(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$, для t=0 до 19 $f_t(X,Y,Z) = X \oplus Y \oplus Z$, для t=20 до 39

$f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$, для t=40 до 59 $f_t(X,Y,Z) = X \oplus Y \oplus Z$, для t=60 до 79 в алгоритме используются следующие четыре константы:

K_t = 0x5a827999, для t=0 до 19
K_t = 0x6ed9ebal , для t=20 до 39
K_t = 0x8flbbcdc, для t=40 до 59
K_t = 0xca62cld6, для t=60 до 79

(Если интересно, как получены эти числа, то :0x5a827999 = $2^{1/2}/4$, 0x6ed9ebal = $3^{1/2}/4$, 0x8flbbcdc = $5^{1/2}/4$, 0xca62cld6 = $10^{1/2}/4$; все умножено на 2^{32})

Блок сообщения превращается из 16 32-битовых слов (M_o по M_{15}) в 80 32-битовых слов *(W_0* no *W_{79})* с помощью следующего алгоритма:

$W_t = M_t$, для t = 0 по 15
$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$, для t= 16 по 79

Если *t* - это номер операции (от 1 до 80), *W,* представляет собой *t-ый* подблок расширенного сообщения, а <<< - это циклический сдвиг влево на *s* битов, то главный цикл выглядит следующим образом:

```
FOR (t = 0; t<= 79; 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
}
```

На рисунке 7.4 показана одна операция. Сдвиг переменных выполняет ту же функцию, которую в MD5 выполняет использование в различных местах различных переменных.

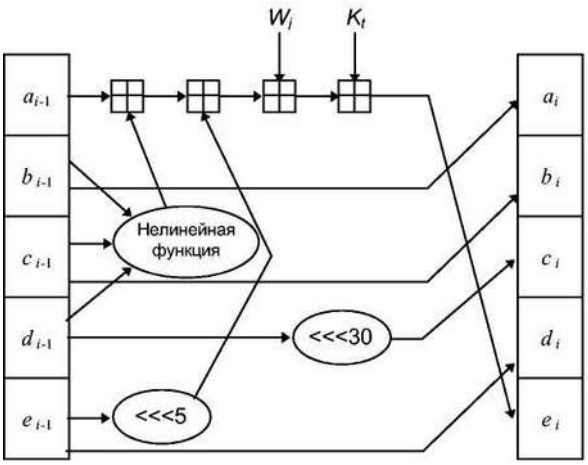


Рисунок 7.4 - Одна операция SHA

После всего этого *a, b, c, d* и *e* добавляются *к A, B, C D* и *E,* соответственно, и алгоритм продолжается для следующего блока данных. Окончательным результатом служит объединение *A, B, C D* и *E.*

Безопасность SHA

SHA очень похожа на MD4, но выдает 160-битовое хэш-значение. Главным изменением является введение расширяющего преобразования и добавление выхода предыдущего шага в следующий с целью получения более быстрого лавинного эффекта.

Российский алгоритм хэширования ГОСТ Р34.11–94

ГОСТ Р34.11–94. Российский алгоритм. Длина свертки – 256 бит (очень удобно для формирования по паролю ключа для ГОСТ 28147–89).

Создатели ***ГОСТ Р34.11–94*** использовали метод последовательного хэширования использующий хэш-функцию с фиксированным размером входа (см. рис. 1.), т. е. функцию сжатия с коэффициентом 7.5.

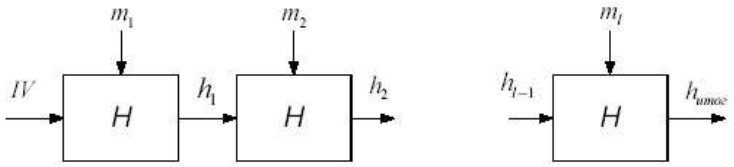


Рис. 7.5. Метод последовательного хэширования

Если необходимо хэшировать сообщение $m = (m_1, m_2, \ldots, m_l)$, то хэширование выполняется следующим образом:

$$h_0 \leftarrow IV,$$
$$h_i \leftarrow H(m_i, h_{i-1}), \quad для\, i = 1, 2, \ldots, l$$
$$h_{итог} \leftarrow h_l.$$

Здесь H_i – функция сжатия, а h_i – переменная сцепления.

Если последний блок содержит меньше чем **n** бит, то он «набивается» до достижения длины **n**. В отличие от стандартных предпосылок, предполагающих, что сообщение предварительно уже было разбито на блоки и произведена «набивка» последнего блока (это соответствует форматированию входного сообщения априори) до начала хэширования, в ГОСТ Р34.11–94 процедура хэширования ожидает конца сообщения (форматирование входного сообщения апостериори). «Набивка» производится следующим образом: последний блок сдвигается вправо, а затем освободившиеся разряды заполняются нулями до достижения длины в 256 бит. Алгоритм хэширования по ГОСТ Р34.11–94 можно классифицировать как устойчивый к столкновениям (**n** = 256, следовательно атака по парадоксу дней рождения потребует приблизительно операций хэширования) код, а также выявляющий модификации (*Collision Resistant Hash Function, CRHF*). Хэш-функцию по ГОСТ Р34.11 – 94 можно легко преобразовать в код аутентификации сообщения любым известным методом (например, HMAC, методом секретного префикса, суффикса, оболочки и т. д.).

Однако разработчики предусмотрели дополнительные меры защиты, для чего параллельно рассчитываются:

контрольная сумма, представляющая собой сумму всех блоков сообщения (последний блок суммируется уже «набитым») по правилу $A + B \bmod 2^k$, где $k = |A| = |B|$, а $|A|$ и $|B|$ битовые длины слов A и B (далее на рисунках и в тексте эту операцию будем обозначать значком Ä);

длина хэшируемого сообщения в битах, приводимая по модулю 256 ($\bmod 2^{256}$).

Рассчитанные значения в финальной функции сжатия используются для вычисления итогового хэша (см. рис. 7.6).

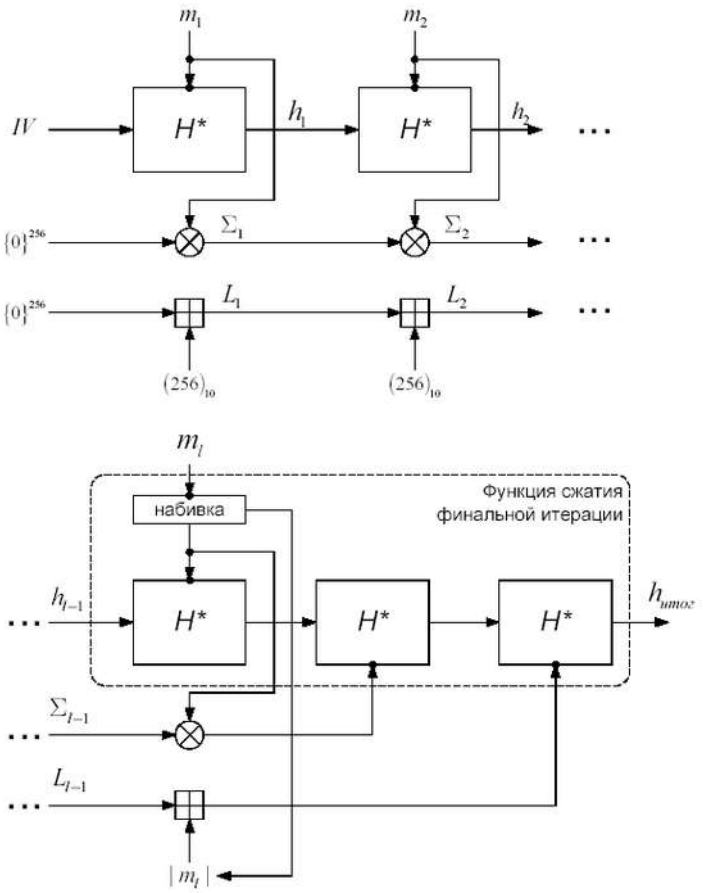


Рис. 7.6. Общая схема функции хэширования по ГОСТ Р34.11– 94

Замечание 1 («набивка»). Указывать в передаваемом сообщении, сколько было добавлено нулей к последнему блоку, не требуется, т. к. длина сообщения участвует в хэшировании.

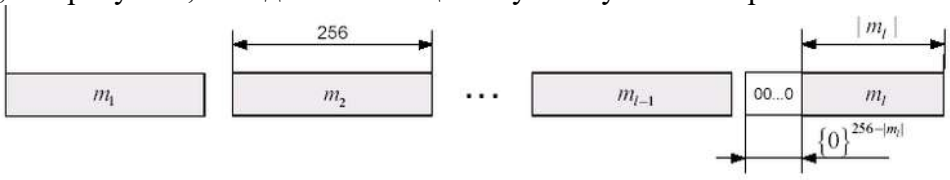


Рис. 7.7. «Набивка» сообщения

Замечание 2. Согласно ГОСТ Р34.11–94, начальный вектор IV – произвольное фиксированное слово длиной 256 бит ($IV \in \{01\}^{256}$). В таком случае, если он априорно не известен верифицирующему целостность сообщения, то он должен передаваться вместе с сообщением с гарантией целостности. Для сообщений небольших объемов задачу противнику можно усложнить, если вектор IV выбирать из небольшого множества допустимых величин (но при этом увеличивается вероятность угадывания хэш-величины противником). Также он может задаваться в рамках организации, домена как константа (обычно, как в тестовом примере).

14.2. Задание к практической работе

1. Изучить алгоритм и разработать программу реализации алгоритма хеш-функции в соответствии с вариантом:
   - алгоритм ***MD2*** - <номер по журналу > MOD 5 = 0;
   - алгоритм ***MD4*** - <номер по журналу > MOD 5 = 1;
   - алгоритм ***MD5*** - <номер по журналу > MOD 5 = 2;
   - алгоритм ***SHA*** - <номер по журналу > MOD 5 = 3;
   - алгоритм ***ГОСТ Р34.11–94*** - <номер по журналу > MOD 5 = 4**.**
2. Проанализировать алгоритм с точки зрения его безопасности.

Контрольные вопросы к практической работе:

1. Какова кратность длины исходного сообщения?
2. Какова разрядность вычисленного хеш-значения?
3. Какие логические операции используются в алгоритме?
4. Для чего используется метод МД-усиления?
5. Для чего используются однонаправленные хеш-функции?

СПИСОК ИСТОЧНИКОВ

1. В.Столлингс, Криптография и защита сетей. Принципы и практика, Издательский дом «Вильямс», Москва, С-Петербург, Киев, 2001 г., 669 стр.
2. Соколов А.В., Шаньгин В.Ф., Защита информации в распределенных корпоративных сетях и системах, ДМК, Москва, 2002 г.
3. Саломаа А. Криптография с открытым ключем, М.:Мир, 1995 г., 320 стр.
4. Шнайдер Б., «Прикладная криптография. Протоколы, алгоритмы и исходные тексты на языке С»
5. Баричев С., «Криптография без секретов»
6. Дж. Брассард «Современная криптология», 1988 г.
7. Шеннон К. «Теория связи в секретных системах. Примеры секретных систем»
8. Исагулиев К. П., Справочник по криптологии, 2004 г, 238 стр.