УДК 004.021

Криптографические хэш-функции MD и их использование для цифровых подписей

Васильева Полина Александровна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Козич Виталий Геннадьевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Глаголев Владимир Александрович
Приамурский государственный университет имени Шолом-Алейхема
к.г.н., доцент кафедры информационных систем, математики и методик обучения

Аннотация
В данной статье были исследованы криптографические хэш-функции Message Digest, а также рассказано об их актуальности и применении в электронных цифровых подписях.
Ключевые слова: информационная безопасность, криптография, Message Digest, хэш-функция, электронная цифровая подпись.

MD cryptographic hash functions and their use for digital signatures

Vasilyeva Polina Alexandrovna
Sholom-Aleichem Priamursky State University
Student

Kozich Vitaliy Gennadievich
Sholom-Aleichem Priamursky State University
Student

Glagolev Vladimir Alexandrovich
Sholom-Aleichem Priamursky State University
candidate of geographical sciences, Associate Professor of the Department of Information System, Mathematics and teaching methods

Abstract
In this article cryptographic hash functions Message Digest have been investigated, as well as their relevance and application in electronic digital signatures.

Keywords: Information security, cryptography, Message Digest, hash function, digital signature.

Начнем с того, что цифровые подписи или, по-другому, электронные подписи достаточно актуальная на данный момент технология. Многие государственные и муниципальные организации в нашей стране пользуются ей. Электронная подпись предназначена для определения человека, подписавшего данный электронный документ, и является аналогом собственноручной подписи в случаях, предусмотренных законодательством. В отличие от собственноручной подписи, ЭЦП позволяет передавать подписанный документ по различным видам цифровой связи без пересылки физической копии документа и сохраняет при этом возможность проверки его подлинности. Это обусловлено тем, что электронная подпись связана с документом логически, а не посредством общего материального носителя. Если записать документ и его ЭЦП на два различных носителя, то не возникнет никаких сложностей с проверкой того, соответствует ли данная ЭЦП данному документу. Средства проверки могут установить подлинность подписи для электронного документа независимо от источника, из которого взята необходимая для проверки информация. Другим важным преимуществом является большая сложность ее подделки. На данный момент неизвестно ни одного случая подделывания ЭЦП, созданной в соответствии с действующими мировыми стандартами.

Проблемам использования указанных алгоритмов посвящены многочисленные публикации. А.А.Варфоломеев в своей статье подробно рассказал о схемах «анонимной цифровой подписи» и схемах цифровой подписи, обеспечивающих анонимность [1]. О хэш-алгоритмах пользователи опубликовали материалы на ресурсе «Хабрахабр» [2]. Об алгоритме Message Digest 2 написано на сайте «techopedia» [3]. На иностранном ресурсе «TechTarget» есть информация о Message Digest 4 [4]. В Национальной библиотеке им. Н. Э. Баумана подробно описан алгоритм Message Digest 5 [5]. Материал по хэш-функции MD6 представлен на интернет-ресурсе «Хабрахабр» [6].

Поскольку подписываемые документы как правило достаточно большого объёма, зачастую электронная подпись ставится не на сам документ, а на его хэш (битовая строка фиксированной длины). Для вычисления хэша используются специальные криптографические хэш-функции, которые выявляют изменения документа при проверке ЭЦП. В схеме подписи может быть использована такая надёжная хэш-функция как MD (Message Digest). Рассмотрим ее разновидности.

MD2 (Message Digest 2)

Эта криптографическая хэш-функция была разработана Рональдом Ривестом в 1989 году. На входе подается сообщение произвольный длины. Размер хэша соответствует 128 битам. На настоящее время алгоритм считается устаревшим. Раньше он использовался для инфраструктуры

открытых ключей, однако сейчас рекомендуется использовать MD5, речь о котором пойдет в статье позже.

И так, реализация алгоритма происходит таким образом. Расширение производится так: i байт, равных i, добавляется к сообщению, так чтобы его длина в байтах стала равной 0 по модулю 16. В конечном итоге, к сообщению добавляется от 1 до 16 байт. 16-байтная контрольная сумма сообщения добавляется к результату предыдущего действия. Для каждого 16-байтного блока дополненного сообщения 16 раз выполняются следующие действия по формуле на рисунке 1.

$$c = M[i * 16 + j],$$
$$C[j] = S[c \oplus L] \oplus C[j],$$
$$L = C[j]$$

Рисунок 1. Формула вычисления контрольной суммы

Здесь i — номер 16-байтного блока данных; j — номер текущего шага цикла; M[x] — x-й байт сообщения, то есть M[i*16+j] — это j-й байт текущего блока данных; c и L — временные переменные, L изначально содержит значение 0; C[j] — j-й байт массива контрольной суммы; перед вычислением контрольной суммы массив содержит нулевые байты; S[i] — i-й элемент 256-байтной матрицы из «случайно» переставленных цифр числа пи. Программная реализация показана на псевдокоде на рисунке 2.

```
/* Clear checksum. */
For i = 0 to 15 do:
   Set C[i] to 0.
end /* of loop on i */

Set L to 0.

/* Process each 16-word block. */
For i = 0 to N/16-1 do

   /* Checksum block i. */
   For j = 0 to 15 do
      Set c to M[i*16+j].
      Set C[j] to C[j] xor S[c xor L].
      Set L to C[j].
    end /* of loop on j */
 end /* of loop on i */
```

Рисунок 2. Вычисление контрольной суммы на псевдокоде

48-байтный буфер X используется для вычисления хэша. MD-буфера инициализируется нулями. Обработка сообщения блоками по 16 байт происходит на той же 256-байтной перестановочной матрице S, как и на

контрольной сумме. Каждый 16-байтный i-й блок дополненного сообщения, включая блок контрольной суммы, накладывается на буфер X следующим образом. Сначала Блок данных копируется в буфер (см. рис. 3).

$$X[16 + j] = M[i * 16 + j], \text{ j = 0...15,}$$
$$X[32 + j] = (X[16 + j] \oplus X[j]), \text{ j = 0...15.}$$

Рисунок 3. Формула копирования в буфер

Таким образом, если представить буфер X как 3 фрагмента по 16 байт, то после копирования блока данных второй из фрагментов буфера содержит обрабатываемый блок данных, а третий — результат применения побитовой операции «исключающее или» к текущему содержимому первого фрагмента и блока данных. Выполняется обработка буфера, которая состоит из 18 раундов преобразований, в рамках каждого из которых выполняется обновление каждого байта буфера по формуле на рисунке 4.

$$X[k] = (X[k] \oplus S[t]), t = X[k]$$

Рисунок 4. Формула обработки буфера

Здесь k = 0…47, t — временная переменная, которая изначально имеет нулевое значение, а после выполнения каждого j-го раунда (j = 0…17) t модифицируется по правилу: t=t+j mod 256. Программная реализация показана на псевдокоде на рисунке 5.

```
/* Process each 16-word block. */
 For i = 0 to N1/16-1 do

    /* Copy block i into X. */
    For j = 0 to 15 do
       Set X[16+j] to M[i*16+j].
       Set X[32+j] to (X[16+j] xor X[j]).
     end /* of loop on j */

    Set t to 0.

    /* Do 18 rounds. */
    For j = 0 to 17 do

       /* Round j. */
       For k = 0 to 47 do
          Set t and X[k] to (X[k] xor S[t]).
       end /* of loop on k */

       Set t to (t+j) modulo 256.
    end /* of loop on j */

 end /* of loop on i */
```

Рисунок 5. Обработка сообщений на псевдокоде

Хэш вычисляется как результат X[0…15]; в начале идет байт X[0], а в конце X[15].

MD4 (Message Digest 4)

Эта криптографическая хэш-функция была разработана профессором Рональдом Ривестом в 1990 году. Для произвольного входного сообщения функция генерирует 128-разрядное значение, называемое также дайджестом. Уровень ее безопасности был рассчитан на создание достаточно устойчивых систем ЭЦП. MD4 создавался как очень быстрый алгоритм хэширования, поэтому он оказался плох в плане криптостойкости.

Сообщение в результате процесса вычисление хэша расширяется так. Бит, равный 1, добавляется к сообщению, а затем добавляются биты, равные 0, до тех пор, пока длина сообщения не станет равной 448 по модулю 512. В итоге, к сообщению добавляется как минимум 1 бит, и как максимум 512. 64-битное представление b добавляется к результату предыдущего действия. В маловероятном случае, когда b больше, чем 2^{64}, используются только 64 младших бита. Эти биты добавляются в виде двух 32-битных слов, и первым добавляется слово, содержащее младшие разряды. На этом этапе мы получаем сообщение длиной кратной 512 битам. Каждое 32-битное содержит четыре 8-битных слова, но следуют они в обратном порядке. Для вычисления хэша сообщения используется буфер, который состоит из 4 слов (32-битных регистров): (A,B,C,D). Как инициализируются регистры шестнадцатеричными числами показано на рисунке 6.

```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

Рисунок 6. Инициализация регистров

Далее определим 3 вспомогательные функции, каждая из которых получает на вход три 32-битных слова, и по ним вычисляет одно 32-битное (см. рис. 7).

$$F(X,Y,Z) = XY \vee \neg XZ$$
$$G(X,Y,Z) = XY \vee XZ \vee YZ$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$

Рисунок 7. Вспомогательные функции

На каждую битовую позицию F действует как условное выражение: если X, то Y; иначе Z. Функция F могла бы быть определена с использованием + вместо v, поскольку XY и ¬XZ не могут равняться 1 одновременно. G действует на каждую битовую позицию как функция максимального значения: если по крайней мере в двух словах из X,Y,Z соответствующие биты равны 1, то G выдаст 1 в этом бите, а иначе G выдаст бит, равный 0. Интересно отметить, что если биты X,Y, и Z статистически независимы, то биты F(X,Y,Z) и G(X,Y,Z) будут также статистически независимы. Функция H реализует побитовый xor, она обладает таким же свойством, как F и G. На рисунках 8-9 показан алгоритм хэширования на псевдокоде.

```
/* Обрабатываем каждый блок из 16-ти слов */
for i = 0 to N/16-1 do

  /* Заносим i-ый блок в переменную X */
  for j = 0 to 15 do
    set X[j] to M[i*16+j].
  end /* конец цикла по j */

  /* Сохраняем 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] означает следующую операцию:
       a = (a + F(b,c,d) + X[k]) <<< s. */
  /* Производим 16 следующих 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]
```

Рисунок 8. Обработка сообщений на псевдокоде

```
  /* Раунд 2 */
  /* Пусть [abcd k s] означает следующую операцию:
       a = (a + G(b,c,d) + X[k] + 5A827999) <<< s. */
  /* Производим 16 следующих 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]

  /* Раунд 3 */
  /* Пусть [abcd k s] означает следующую операцию:
       a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s. */
  /* Производим 16 следующих 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]

  /* Затем производим следующие операции сложения. (Увеличиваем значение в каждом регистре
     на величину, которую он имел перед началом итерации по i */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD

end /* конец цикла по i */
```

Рисунок 9. Обработка сообщений на псевдокоде

Результат выглядит как ABCD. То есть, выписываем 128 бит, начиная с младшего бита A, и заканчивая старшим битом D.

MD5 (*Message Digest 5*)

Данный 128-битный алгоритм был разработан профессором Рональдом Ривестом из Массачусетского технологического института в 1991 году и предназначен для создания дайджестов сообщения произвольной длины и последующей проверки их подлинности. Ранее считалось, что MD5 позволяет выдавать достаточно надёжный идентификатор для блока данных. На данный момент данная хэш-функция не рекомендуется к использованию, так как существуют способы нахождения коллизий. Надстройка «Keyed-Hashing for Message Authentication» позволяет хэшировать входное сообщение с некоторым ключом, т.е. аутентифицирует подпись.

На вход алгоритма поступает поток данных. Длина сообщения может быть абсолютно любой. Длина сообщения записывается в *L*. Это число целое и неотрицательное. После поступления данных идёт процесс подготовки вычисления потока. И так, сначала дописывают единичный бит в конец потока, а затем необходимое число нулевых бит. Далее входные данные (во всех случаях) выравниваются в новый размер L', сравнимый с 448 по модулю 512 (L'=512*N+48). При добавлении длины сообщения, в конец дописывают 64-битное представление длины данных до выравнивания. Сначала записываются младшие 4 байта, а после - старшие. Если длина превосходит 2^{64}-1, то дописываются только младшие биты. После этого длина потока становится кратной 512. Вычисления будут полагаться на представлении этого потока в виде массива слов по 512 бит. При инициализации буфера в ходе вычислений, инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами (см. рис 10).

```
A = 01 23 45 67; // 67452301h
B = 89 AB CD EF; // EFCDAB89h
C = FE DC BA 98; // 98BADCFEh
D = 76 54 32 10. // 10325476h
```

Рисунок 10. Инициализация буфера

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD по-другому называется инициализирующим вектором. Для вычислений понадобятся функции из 4х раундом (см. рис. 11).

1-й раунд: $\mathbf{FunF}(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$,
2-й раунд: $\mathbf{FunG}(X,Y,Z) = (X \wedge Z) \vee (\neg Z \wedge Y)$,
3-й раунд: $\mathbf{FunH}(X,Y,Z) = X \oplus Y \oplus Z$,
4-й раунд: $\mathbf{FunI}(X,Y,Z) = Y \oplus (\neg Z \vee X)$,

Рисунок 11. Функции от трех параметров

Далее элемент *n* заносится в блок данных из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками. Четыре этапа данной процедуры показаны на рисунках 12-15.

```
/* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[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]
```

Рисунок 12. Этап 1

```
/* [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
[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]
```

Рисунок 13. Этап 2

```
/* [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
[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]
```

Рисунок 14. Этап 3

```
/* [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
[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]
```

Рисунок 15. Этап 4

MD6 (Message Digest 6)

Данный алгоритм переменной разрядности был разработан тем же профессором, что и MD5. Предназначен для создания дайджестов сообщений произвольной длины и предлагается на смену менее совершенному MD5. Используется для проверки целостности и подлинности опубликованных сообщений, путём сравнения дайджеста сообщения с опубликованным. Также широко используется для генерации ключей фиксированной длины в алгоритмах шифрования на основе заданной ключевой строки. MD6 предполагается доказуемо устойчивой к дифференциальному криптоанализу (с помощью которого была взломана MD5).

В хэш-функции использованы достаточно оригинальные идеи. За один раз обрабатывается 512 байт, затрудняя проведение ряда атак и облегчая процесс распараллеливание. При этом применяются древовидные структуры.

Нелинейность функции основывается на использовании ряда простейших операций: XOR, а также сложения и сдвига. Количество раундов составляет: r=40+(d/4), что очень велико.

Заключение

Проведя обзор на различные функции MD можно сделать выводы:

1. MD2 относительно медленный алгоритм хэширования по сравнению с остальными известными алгоритмами.
2. MD4 использует три цикла из 16 шагов каждый, в то время как MD5 использует четыре цикла из 16 шагов каждый.
3. MD5 использует четыре элементарные логические функции, по одной на каждом цикле, по сравнению с тремя в MD4, по одной на каждом цикле.
4. MD6 достаточно устойчив к криптоанализу по сравнению с MD5. Зная MD6, невозможно восстановить входное сообщение, так как разным сообщениям может соответствовать один MD6.

В результате работы был сделан обзор на существующие алгоритмы хэширования MD. Кратко были изложены их плюсы и минусы, а также сравнения друг с другом. Было рассказано об их применении в электронных цифровых подписях. Данное исследование будет полезно для изучения дисциплин, связанных с информационной безопасностью.

Библиографический список

1. Варфоломеев А.А. О схемах «анонимной цифровой подписи» и схемах цифровой подписи, обеспечивающих анонимность // Вопросы кибербезопасности. 2015. №1(9). С. 44-48.
2. Хэш-алгоритмы // Хабрахабр URL: https://habrahabr.ru/post/93226/ (дата обращения: 20.11.2017).
3. Message Digest 2 (MD2) // techopedia URL: https://www.techopedia.com/definition/31699/message-digest-2-md2 (дата обращения: 20.11.2017).
4. MD4 // TechTarget URL: http://searchsecurity.techtarget.com/definition/MD4 (дата обращения: 20.11.2017).
5. MD5 (Message Digest 5) // Национальная библиотека им. Н. Э. Баумана URL: http://ru.bmstu.wiki/MD5_(Message_Digest_5) (дата обращения: 20.11.2017).
6. Новая хэш-функция MD6 // Хабрахабр URL: https://habrahabr.ru/post/45849/ (дата обращения: 20.11.2017).