Описание алгоритма MD5(RFC1321)

Алгоритм MD5 является алгоритмом вычисления "хэш-функции" (message digest) для различных целей - шифрование паролей, проверка целостности файлов, и т.д. На вход подается поток данных произвольной длины, а на выходе получаем хэш длиной 128 бит. Сила этого алгоритма заключается в том, что практически очень сложно, почти невозможно, найти две строки, дающие одинаковый хэш. Однако, при определенных условиях, возможно получение исходного текста. Но этот метод основан на определенном выборе начальных значений, поэтому не представляет практической ценности. Также MD5 алгоритм используется в приложениях криптографии и электронно-цифровых подписей для генерации ключа шифрования.

При разработке данного алгоритма принималось во внимание скорость работы на современных 32-разрядных системах, а также требование минимизации используемой памяти. По сравнению с MD4, MD5 более медленный (примерно на 15 процентов, как будет показано в конце статьи), чем MD4, но в то же время и более устойчивым, в силу особенностей построения. О сравнении скорости MD4 и MD5 будет отдельный параграф.

Введем следующие обозначения: под "словом" будет подразумеваться количество информации в 32 бита, а под "байтом" - 8 бит. Последовательность бит будем рассматривать как последовательность байт, старший байт идет первым, младший - последним. Аналогично представляется последовательность байт, как последовательность слов, только младший байт идет первым.

На вход алгоритма подается входной поток данных длиной N. N может быть произвольным целым неотрицательным числом. Это число может быть как кратным, так и нет, 8. Процесс вычисления MD5 суммы состоит из нескольких шагов. Рассмотрим их подробнее.

Шаг 1: выравнивание потока.

Процесс выравнивания заключается в дописывании в конец потока 1, а затем некоторого числа нулей. Нули добававляются до тех пор, пока длина всего потока не станет равной 512*N+448, т.е. равной 448 по модулю 512. Такое выравнивание происходит в любом случае, даже если длина потока уже удовлетворяет данному условию.

Шаг 2: добавление длины.

Затем в конец дописывается двоичное представление длины первоначального потока-всего 64 бита. Если же длина больше чем

2^64, то берутся младшие 64 бита. Это добавление представляет собой два «слова», младшее идет первым, за ним старшее.

После этого суммарная длина потока станет кратной 16 32-битным словам.

Дальнейшее вычисления основываются на представлении этого расширенного потока как массива слов длины N: A[0, ... N-1].

Шаг 3: инициализация MD буфера.

При вычислениях будет использоваться буфер из 4 слов - A, B, C, D, в котором хранятся результаты промежуточных вычислений. Начальные значения, находящиеся в этом буфере, следующие:

A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476

Шаг 4: обработка потока блоками по 16 слов.

Для дальнейших вычислений нам необходимо ввести следующие функции, зависящие от 3 параметров-«слов». Результатом работы этих функций будет также «слово».

F(x, y, z) = (x & y) | (~x & z)
G(x, y, z) = (x & z) | (y & ~z)
H(x, y, z) = x ^ y ^ z
I(x, y, z) = y ^ (x | ~z)

Здесь **«&»** обозначает побитовая операция «AND», **«|»** побитовая операция «XOR», **«^»** операция побитового «XOR», **«~»** операция побитового «NOT»

Таблицы истинности для этих функций:

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

x	y	z	G
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	z	H
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

x	y	z	I
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

На этом этапе нам также понадобится таблица констант T[1...64], заполненная с помощью следующей формулы: T[i] = [4294967296 * abs(sin(i))] , где [] есть операция взятия целой части.
Определим также операцию циклического сдвига слова X на Y: X<<<Y.

Дальнейшее описание алгоритма, его основная часть, будет написано на псевдокоде.

```
// разбиваем поток на блоки по 16 слов:
for i = 0 to N/16 - 1 do
{
    // i-й блок заносится в X
    for j = 0 to 15 do
        X[j] = M[i * 16 + j]

    // Сохраняем значения A, B, C, D
    AA = A
    BB = B
    CC = C
    DD = D

    // раунд 1
    // [abcd k s i] обозначает операцию
    //     a = b + ((a + F(b, c, d) + X[k] + T[i]) <<< s)
    // выполняется 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
    // [abcd k s i] обозначает операцию
    //     a = b + ((a + G(      b, c, d) + X[k] + T[i]) <<<
s)
    // выполняется 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
    // [abcd k s i] обозначает операцию
    //     a = b + ((a + H(b, c, d) + X[k] + T[i]) <<< s)
    // выполняется 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
    // [abcd k s i] обозначает операцию
    //     a = b + ((a + I(b, c, d) + X[k] + T[i]) <<< s)
    // выполняется 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]

    A += AA
    B += BB
    C += CC
    D += DD
}
```

Шаг 5: вывод MD5.

Окончательный результат, находящийся в буфере A-D, и есть почти готовый хэш. Выводя «слова» из этого буфера в обратном порядке, мы получим готовый хэш. Т.е. md5hash=DCBA.

Теперь о свойствах алгоритма в целом.
Для начала краткое описание отличий от MD4.

Различия между MD4 и MD5

Следующий список представляет собой различия между MD4 и MD5:

1. MD5 имеет на один раунд больше - 4 против 3 у MD4.
2. Что бы уменьшить влияние входного текста была введена уникальная константа для каждого раунда-T[i].
3. Во втором раунде заменили функцию g с (XY v XZ v YZ) на (XZ v Y not(Z)), для того, чтобы сделать g менее симметричной.
4. На каждом шаге используется значение, полученное на предыдущем шаге. Это дает более быстрое изменение результата при изменении входных данных.
5. Для этих же целей количество сдвигов различается от раунда к раунду, и выбрано так, что бы еще более увеличить этот эффект.
6. Изменен порядок, в котором обрабатываются слова в раундах 2 и 3, для того чтобы сделать их менее похожими друг на друга.

Скорость работы и производительность

Для того, что бы показать разницу в скорости MD4 и MD5, был проведен следующий эксперимент. Результаты занесены в таблицу. Суть эксперимента заключается в том, что бы замерить время, затраченное на построение 10000 хэшей от сообщения размером 10000 байт.
Вычисления проводились на Intel Pentium III 750 МГц. В качестве двух реализаций были выбраны реализация из пакета OpenSSL и основанная на RFC1321.

	MD4		MD5	
RFC	2.574 сек	37940 кБ/сек	2.614 сек	37359 кБ/сек
OpenSSL	0.891 сек	109603 кБ/сек	1.152 сек	84771 кБ/сек

Результаты исследования показывают, что:
- В реализации RFC MD5 медленнее MD4 на 1.55%..4.92%
- В реализации MD5 медленнее MD4 на 29.29%..39.82%

Но, тем не менее, в настоящее время MD5 используется гораздо шире, чем MD4. Это связано в первую очередь с повышенной надежностью первого.

НАДЕЖНОСТЬ

В 1996 году появилась статья, которая позволила некоторым авторам считать данный алгоритм взломанным. Основная идея такая: если бы была возможность задать произвольные начальные значения буфера MD5 (**0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476**), то тогда можно было бы подобрать два сообщения, которые не различаются, кроме, быть может, в нескольких разрядах, таких, что для них может быть построен один и тот же дайджест. Математически это выражается так: MD5(IV, M1) = MD5(IV, M2),
где IV = Initial Values = начальные значения, M1 и M2 два разных сообщения.

Автор данного сообщения, Hans Dobbertin, нашел, что если в качестве начальных значений буфера использовать A = 0x12AC2375, B = 0x3B341042, C = 0x5F62B97C, D = 0x4BA763ED, и задать содержимое блока данных для преобразования следующим образом:
X0 = 0xAA1DDABE, X1 = 0xD97ABFF5, X2 = 0xBBF0E1C1,

X3 = 0x32774244, X4 = 0x1006363E, X5 = 0x7218209D,
X6 = 0xE01C136D, X7 = 0x9DA64D0E, X8 = 0x98A1FB19,
X9 = 0x1FAE44B0, X10 = 0x236BB992, X11 = 0x6B7A779B,
X12 = 0x1326ED65, X13 = 0xD93E0972, X14 = 0xD458C868,
X15 = 0x6B72746A.
Тогда второе сообщение строится из первого по такой формуле:

$$X'_i = \begin{cases} X_i, i < 16, i \neq 14 \\ Xi + 2^9, i = 14 \end{cases}$$

Тогда MD5(IV, X) = MD5(IV, X') =
BF90E670752AF92B9CE4E3E1B12CF8DE.

СТОЙКОСТЬ К НАХОЖДЕНИЮ КОЛЛИЗИЙ

Нахождению коллизий препятствует быстрый обвал выходной функции. К примеру, рассмотрим два хэша от двух сообщений, различающихся в одном бите:
MD5(abc) = **900150983CD24FB0D6963F7D28E17F72**
MD5(acc) = **1673448EE7064C989D02579C534F6B66.**

Отсюда хорошо видно, что выходные значения отличаются значительно. Полный перебор же - занятие бессмысленное, так как при имеющихся скоростях порядка 100 Мб/сек, полный перебор занял бы около 10^{62} лет, и при этом потребовал бы 2^{230} Гб места. Так что MD5 по праву считается надежным алгоритмом.

Список литературы:

[1] RFC1321: The MD5 Message-Digest Algorithm
[2] Статья об MD5: http://www.argc-argv.relc.com/1_2003/MD5_article.pdf