MD5 Algorithm

By Daniel Ravina, March 25th, 2015

Introduction

MD5, (aka. 'Message-Digest Algorithm 5'), the most commonly used cryptographic hash function, was invented in 1991, by Ronald Rivest at MIT. The idea behind cryptographic hashing is to take an arbitrary block of data and regardless of the type and size, return a fixed-size “hash”. An MD5 hash has 128-bit value, which is typically represented as a 32-character hexadecimal number.

Some examples:

```
md5 ("Hello World")         == "b10a8db164e0754105b7a99be72e3fe5" // true
md5 ("COMP 3761 is Great!") == "5a198786dba6db7102d30df8f93f43e1" // true
md5 ("")                    == "d41d8cd98f00b204e9800998ecf8427e" // true
```

This shows that all input streams yield hashes of the same length. Further more - changing just one character yields sweeping changes in the entire value of the hash:

```
md5 ("Hello World") == "b10a8db164e0754105b7a99be72e3fe5" // true
md5 ("Hello WorlD") == "64db00194eb82c028d6a75f576f9202f" // true
```

The MD5 is a "one-way" operation. There is no possible way to trace back the original input. Some websites claim to *'decrypt'* the hashes, but in reality they are just searching the value in a giant database (~43.5 billion records) to find the match.

Algorithm

MD5 algorithm consists of 5 steps:

1. Appending Padding Bits
2. Appending Length
3. Initializing MD Buffer
4. Processing Message in 512-bit Blocks
5. Output

Pre-steps

`Let b = the length (in bits) of the input string`

STEP 1 - Appending Padding Bits

The message is extended so that its length (in bits) is close to 448, modulo 512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended continuously so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

STEP 2 - Appending Length

A 64-bit representation of `b` is appended to the result of the previous step. At this point the resulting message has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. `Let M[0 ... N-1]` denote the words of the resulting message, where N is a multiple of 16.

STEP 3 - Initializing MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

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

STEP 4 - Processing Message in 512-bit (16-Word Blocks) Blocks

first define four utility functions that each take as input three 32-bit words and produce as output one 32-bit word.

```
F (X,Y,Z) = (X AND Y) OR (NOT(X) AND Z)
G (X,Y,Z) = (X AND Z) OR (Y AND NOT(Z))
H (X,Y,Z) = X XOR Y XOR Z
I (X,Y,Z) = (Y XOR (X OR NOT(Z))
```

This step uses a 64-element table `T[1 ... 64]` constructed from the sine function.

`Let T[i]` denote the i-th element of the table, which is equal to the integer part of `4294967296` times `abs(sin(i))`, where `i` is in radians.

Now do the following:

```
// Process each 16-word block.
For i = 0 to N/16-1 do
    // Copy block i into X.
    For j = 0 to 15 do
      Set X[j] to M[i*16+j].
    end
    // Save A as AA, B as BB, C as CC, and D as DD.
    AA = A
    BB = B
    CC = C
    DD = D

    // ROUND 1
    // Let [abcd k s i] denote the operation
    // a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s).
    // Do the following 16 operations:
    [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]

    // ROUND 2
    // Let [abcd k s i] denote the operation
    // a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)
    // Do the following 16 operations:
    [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]

  // ROUND 3
  // Let [abcd k s t] denote the operation
  // a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)
  // Do the following 16 operations:
  [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]

  // ROUND 4
  // Let [abcd k s t] denote the operation
  // a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)
  // Do the following 16 operations:
  [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]

  // Then perform the following additions. (That is increment each
  // of the four registers by the value it had before this block
  // was started.)
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD

end // of loop on i
```

STEP 5 - Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

Applications

MD5 is mainley used for data and file verification. When a file is downloaded over a network, comparing it's md5 hash with the original hash can guarantee that the file is not corrupted.

MD5 was also used to securely store passwords, However, in 1996, a flaw was found in the design of MD5 and it was no longer recommended to use it for situations where security is the prime concern.