Cryptographic Hashing Functions - MD5

Farhad Ahmed Sagar

September 2016

1 Introduction

Cryptographic hash functions take data input (or massage) and generate a fixed size result (or digest). The result is called **checksome**. It is almost impossible to regenerate the input from the result of hash function. One thing to see here is that the hash functions are not encryption because you cannot decrypt the input from the output.

One of the most widely used Cryptographic hash Function is **MD5** or "message digest 5". MD5 creates a 128-bit message digest from the data input which is typically expressed in 32 digits hexadecimal number. MD5 hashes are unique for different inputs regardless of the size of the input. MD5 hashses looks like this.

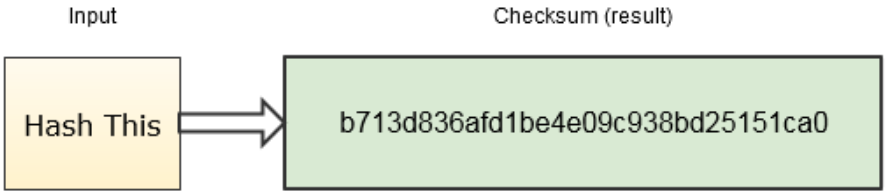

Figure 1: Hexadecimal representation of input by md5

It is widely used to make sure that the transferred file in a software has arrived safely. For example when you download a file from the Internet/Server it might be corrupted or there might be data loss due to connection loss, virus, hack attack or some other reason. One way to check if the downloaded file is same as you intended is by generating an MD5 hash on the server for the file and again for the downloaded file, if both of the hash matches then your file is perfect. It is also used in database to store passwords as hash instead of the original input.

2 History

MD5 or "message digest 5" algorithm was designed by professor Ronald Rivest. Rivest is a professor in MIT who also invented RSA, RC5 and the MD-message digest hashing functions. MD5 is a one way hashing function. So by definition it should fulfill two properties. One, it is one way which means one can create a hash value from a message but cannot recreate the message from the hash value. Two, it should be collision free that is two distinct messages cannot have the same hash value.

Rivest first designed MD2 for 8-bit machines in 1989. The original message is padded at first so that the total message is divisible by 16. Plus a 16-byte checksome is added to it to create a total 128-bit message digest or hash value. But collisions were for MD2 were found soon.

Rivest then developed MD4 for 32-bit machines in 1990. MD4 influenced a lot of cryptographic hash functions such as MD5, SHA-1. Same as MD2 collisions for MD4 were found soon enough. MD4 has been criticized even by Ronald Rivest because MD4 was designed to be fast which led to a lot of security risks.

MD5 was developed in 1991. MD5 is almost same as MD4 but with "safety belts". Its slower than MD4 but more secure. But over the years collisions were found in MD5. Den Boer and Bosselaers first found collision in MD5 in 1993. In March 2004 a project called **MD5CRK** was initiated to find collision in MD5 by using Birthday Attack.The project stopped as early as August 2004 after Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu showed and analytical attack that only takes one hour on an IBM p690 cluster. For security reseaons details of their method was not published.

3 Description

MD5 creates a 128bit message digest from data input. The output must be unique from other message digests.

Imagine a **b-bits** message to digest. To digest this message we need to follow 5 steps. Professor Rivest used the first two steps to prepare the input message for digestion by appending and padding its bits. In the third and fourth step he used a few helper functions which includes four word buffers and four auxiliary functions which are pre-initialized.

3.1 Append Padding Bits

The first step is to extend(padded) the b-bits message(input) so that the length of the message is equal to 448, modulo 512. In simpler word the message should be just 64-bits shy of being a multiple of 512 that is after padding the message+64 bit should be divisible by 512. (Message+64)/512 will have reminder 0. No matter the size of the message padding is always done. First a '1' bit is appended to the message and then a series of '0' bits.

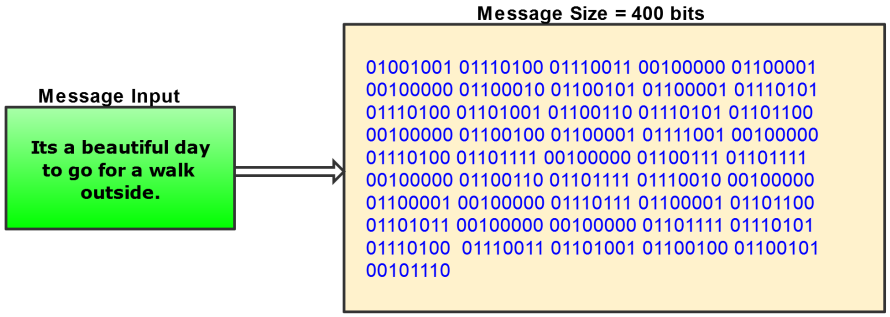

Figure 2: 400 bits original message

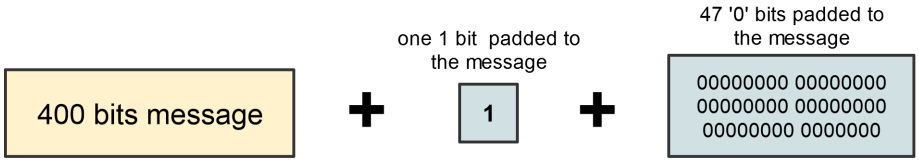

Figure 3: First one '1' bit is added and then '0' bits

For example if our message is 400 bits then we will add one '1' bit and 47 '0' bits which gives us 448 bits which is 64 bits shy of being divisible by 512. If our message is 1200 bits in size than we will add one '1' bit and 271 '0' bits which gives us 1472 bits. 1472+64 is divisible by 512. At least 1 bit and at most 512 bits are padded or extended to the original message.

3.2 Append Length

Now we take the original message and make a 64-bit representation of the original b-bit message. We append this to the result of the previous step. Now the message has a length that is exactly divisible by 512 or the message is multiple of 512. Which itself is divisible by 16.

At this point the message is divided into blocks of 512 bits each. Each 512 bits block is divided into 16 words of 32-bits each. We denote the words as M[0.....N-1] where N is a multiple of 16.

3.3 Initialize MD Buffer

MD5 uses a four word buffer each 32-bits long. We denote them by A,B,C,D. These are pre-initialized as:

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

3.4 Process Message in 16-word Blocks

We use two other helper functions. One, we use four auxiliary functions for MD5. Two, we use a table consisting 64-elements

3.4.1 Auxiliary Functions

Auxiliary functions take three inputs of 32-bits word and gives an output of 32-bit word. The auxiliary function apply logical and, or and xor to the inputs. Later on we will show how we use each of these 4 functions in 4 rounds (each round has 16 operations). The functions are:

$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$
$$H(B,C,D) = B \oplus C \oplus D$$
$$I(B,C,D) = C \oplus (B \vee \neg D)$$

3.4.2 The Table

The table consists of 64-elements where each element is computed using mathematical sin function:

$$abs(\sin(i+1)) \times 2^{32}$$

We denote the table as K[1.....64]. K[i] is the i-th element of the table. Elements of the table is pre calculated.

K[0.. 3]	0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee
K[4.. 7]	0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501
K[8..11]	0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be
K[12..15]	0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821
K[16..19]	0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa
K[20..23]	0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8
K[24..27]	0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed
K[28..31]	0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a
K[32..35]	0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c
K[36..39]	0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70
K[40..43]	0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05
K[44..47]	0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665
K[48..51]	0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039
K[52..55]	0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1
K[56..59]	0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1
K[60..63]	0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391

3.4.3 The Algorithm

Recall from step 2, where we divided the message into blocks of 512-bits and then each 512-bits block to 16 words of 32-bits. Those 16 word (each 32-bits) blocks are denoted as M[0....N-1]. Now for each word block we perform **4 rounds** where each round has **16 operations**. Each round follows basic pattern.

The operations of each round is denoted as [**abcd g s i**]. Here **i** is used to get the i-th value from the table k[i]. **i** also denotes the number of operation from 0 to 63 total 64 operations.

s denotes the number of bits to shift. here we pre define shift values for each 64 operations from array s[].

```
//Initializing s and K
var int[64] s, K

//calculating the values of K and inserting them into the table
//we can use the pre-calculated table K as well
for i from 0 to 63
    K[i] = floor(232 * abs(sin(i + 1)))
end for

//s specifies the per-calculated shift amounts
s[ 0..15] = {7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22}
s[16..31] = {5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20}
s[32..47] = {4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23 }
s[48..63] = {6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21 }
```

Next we initialize the four word buffer from step 3. Also we appedn the padding bits and append the length from step 1 and 2.

```
//Initialize word buffer:
var int a0 = 0x67452301 //A
var int b0 = 0xefcdab89 //B
var int c0 = 0x98badcfe //C
var int d0 = 0x10325476 //D

//Padding: adding a single 1 bit
append "1" bit to message
//Padding: padding with zeros
append "0" bit until message length in bits = 448 (mod 512)

//Append Length:
append original length in bits mod (2 pow 64) to message
```

Next we divide the message into 512-bit blocks and each of those blocks to 16 words of 32-bits.

```
//Process the message in successive 512-bit chunks
//Break each chunk into 16 words
for each 512-bit chunk of message
    break chunk into sixteen 32-bit words M[j], 0 <= j <= 15
    //Initialize hash value for this chunk:
    var int A := AA
    var int B := BB
    var int C := BB
    var int D := DD
```

Next, we move on to the main loop where each of the 64 operations from 4 rounds take place.
From the denoted operation **[abcd g s i], g** is used to get the word from our 32-bit word block M[j], $0 \leq j \leq 15$. We can pre calculate **g**. For each of the four rounds we can get the value of **g** based on **i**.
For round 1 $\mathbf{g} = i$.
For round 2 $\mathbf{g} = (5 \times i + 1) mod 16$.
For round 3 $\mathbf{g} = (3 \times i + 1) mod 16$.
For round 4 $\mathbf{g} = (7 \times i) mod 16$.
Each of the rounds use one auxiliary function for each of its operation. Round 1 uses function F, round 2 uses function G, round 3 uses function H and round 4 uses function I.

```
//continuing from the previous loop
for each 512-bit chunk of message
    .
    .
    .
```

```
    //Main loop:
    for i from 0 to 63
        if 0 <= i <= 15 then
            F = (B and C) or ((not B) and D)
            g = i
        else if 16 <= i <= 31
            F = (D and B) or ((not D) and C)
            g = (5*i + 1) mod 16
        else if 32 <= i <= 47
            F = B xor C xor D
            g = (3*i + 5) mod 16
        else if 48 <= i <= 63
            F = C xor (B or (not D))
            g = (7*i) mod 16
        dTemp = D
        D = C
        C = B
        B = B + leftrotate((A + F + K[i] + M[g]), s[i])
        A = dTemp
    end for

    //Add this chunk's hash to result so far:
    AA = AA + A
    BB = BB + B
    CC = CC + C
    DD = DD + D
end for

//leftrotate function
leftrotate (x, c)
    return (x << c) binary or (x >> (32-c));
```

After all operations the buffer AA,BB,CC,DD contains MD5 digest of our input message.

```
var char digest[16] := AA append BB append CC append DD
```

3.5 Output

After all the operations and all 4 previous steps are done. The buffer AA, BB, CC, DD contains MD5 digest of our input message.

4 Disadvantages

Even after being widely used one of the major flaws of MD5 is that its not collision free as first thought of. As we discussed in section 2 (History) of this paper collision for MD5 has been found multiple times. Collision means

having the same output for two distinct inputs.

MD5 has 128-bits output. Which means if we consider all the outputs possible by different inputs we can have 2^{128} distinct outputs. So we can definitely have collision for two distinct inputs as we can have a lot more than 2^{128} inputs.

5 Applications

One of the most important usage of MD5 is to check if transferred data between server has arrived safely. To check the integrity of files transferred between server we can generate MD5 hash on both side of the server and check if they match. If the hash value matches than file transfer was successful.

Another usage of MD5 is one-way password encryption. In any website database if we need to store password it is important that we don't store the password string as it is inside the database. Instead we use MD5 or any other hashing function to generate a hash value of the original password input. Usually developers design their programs in a way that while registering for a website the user password will be stored as hash value inside the database. And while logging in the password will be hashed again and matched with the hash value of the in the database, if they matches than user can get access. That way if any hacker can break into the database he can only see the hash value of the password which will prevent him from logging into a user's account.

6 Conclusion

MD5 is a one-way hash function that creates a hash value from an input. It provides a way to secure any messages and an affective way to check the integrity of any data passed between servers. Although collision has been found for MD5 it is still one of the most widely used hash function in the world.

References

[1] by Ronald L. Rivest, "The MD4 Message Digest Algorithm" Internet RFC 1320 (April 1992).

[2] by Xie Tao, Fanbao Liu, and Dengguo Feng, "Fast Collision Attack on MD5." (2013).

[3] Wikipedia, MD5, Available at: https://en.wikipedia.org/wiki/MD5. (2015)

[4] Search Security, MD5v Definition, Available at: http://searchsecurity.techtarget.com/definition/MD5. (2005)

[5] Iusmentis, The MD5 cryptographic hash function, Available at: http://www.iusmentis.com/technology/hashfunctions/md5/. (2005)

[6] by Srikanth Ramesh, What is MD5 Hash and How to Use it, Available at:http://www.gohacking.com/what-is-md5-hash/. (2009)