MD5 and password protection

2014-10-23

YiWang ZHENG(12330423)

MD5 means Message Digest Algorithm – 5, is an algorithm to generate an unique identification string, just like our fingerprints identify who we are. So this algorithm is widely applied to help one checking whether the message he received is exactly the original message he wants. However it can also provide high security password protection with some extra random salt..

1. MD5 algorithm

   The algorithm of MD5 can be simply divided into four steps:

   - Bits filling

     Fill bits into the data until the number of bit is 512n+448, then append extra 64 bits which representing the bits number of the original data to the end.

   - Grouping

     Divide the data after bits filling into 512 bits every group

   - Calculate Link Variables

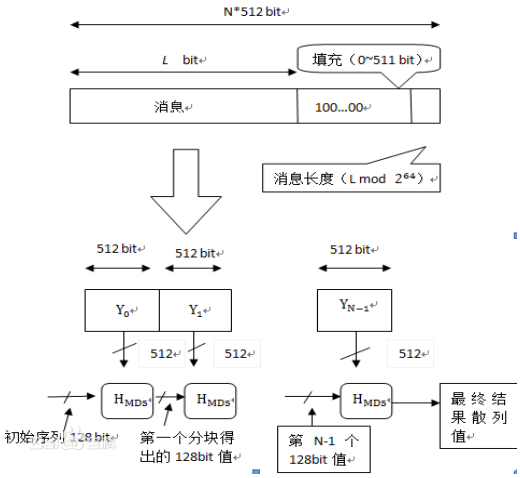
     Use the initial link variable A, B, C, D and the bits in the first group to generate new A, B, C, D;

     For every following group, use the new A, B, C, D generate by the previous loop to perform calculation and get new link variables for the new loop.

   - Link

     Link four link variables A, B, C, D to obtain the MD5 value of the input data.

Schematic diagram of MD5 algorithm



This is the pseudo-code to finish these four steps:

From [http://en.wikipedia.org/wiki/MD5]

```
//Note: All variables are unsigned 32 bit and wrap modulo 2^32 when
calculating
var int[64] s, K

//s specifies the per-round 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 }

//Use binary integer part of the sines of integers (Radians) as
constants:
for i from 0 to 63
    K[i] := floor(abs(sin(i + 1)) × (2 pow 32))
end for
//(Or just use the following table):
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 }

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

//Pre-processing: adding a single 1 bit
append "1" bit to message
/* Notice: the input bytes are considered as bits strings,
  where the first bit is the most significant bit of the byte.[46]


//Pre-processing: padding with zeros
append "0" bit until message length in bits ≡ 448 (mod 512)
append original length in bits mod (2 pow 64) to message


//Process the message in successive 512-bit chunks:
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 := a0
    var int B := b0
    var int C := c0
    var int D := d0
//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:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[16] := a0 append b0 append c0 append d0 //(Output is
in little-endian)

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

2. Password Protection with MD5

The security of password can be affected by many factors. Including the strength of the password created by user basically and the encryption algorithm for the password during authorizing and storage in the server end.

The problem is if we save user's password in somewhere with their clear text form, some one who has the higher level right may can access that file know other user's password, so we had better save password in cipher text so that even the invader can't figure out the exact password from it.

Encrypting with MD5 algorithm before writing password into local file can be a reliable solution since MD5 algorithm is almost irreversible. This solution has been applied to some operation system such as Linux and some Unix released version.

The implementation of applying MD5 algorithm to password protection :

- User sends password to server (First time)
- Server uses MD5 algorithm to encrypt the password and then save the MD5 value to database, or write into a certain file.
- User sends password to server next time(usually for login)
- Server use the data it received to generate a MD5 value and check whether it's the same as the local correct MD5 value.

3. Collision Problem

However, when different user register with the same password by accident, the hackers can find that the MD5 value stored in the database or written in a file of those user are the same, so by hashing a known password and comparing the hash value, they can know the users who using certain password. Even they still can not get the password of certain user, it do affects the security of password protection.

To solve this problem, means to generate different MD5 value for every password even the password are the same. A widely used solution is to add some extra condiments, which are usually called “salt”, at the end of(or some other certain place) the input data before performing MD5 algorithm. This method is also used by the Unix operation system, the hashes of salted passwords is stored in a password file - (/etc/passwd). This is some how like a new version of MD5 algorithm but it won't get the same value for the same input data.