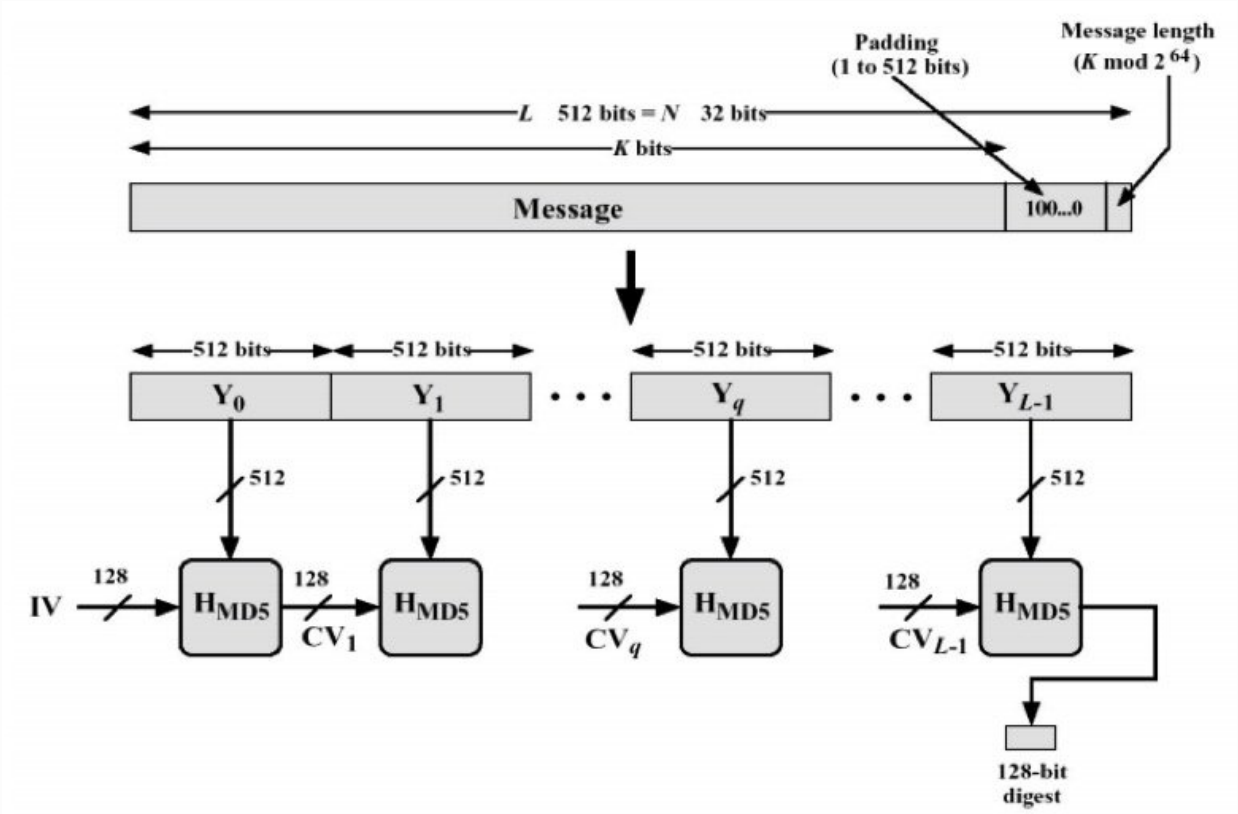
MD5 算法的程序设计和实现

MD5算法概述

MD5，全名Message Digest Algorithm 5，为计算机安全领域广泛使用的一种散列函数，用以提供消息的完整性保护。MD5是一种信息摘要算法，主要是通过特定的hash散列方法将文本信息转换成简短的信息摘要，压缩+加密+hash算法的结合体，是绝对不可逆的。MD5是输入不定长度信息，输出固定长度128-bits的算法。经过程序流程，生成四个32位数据，最后联合起来。

MD5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。MD5算法的过程分为四步：数据填充，设置初始值（标准向量），四轮循环运算，拼接结果。

结构和模块



数据结构

采用32位无符号整数作为存储单元，常量的整数部分是正弦(弧度)* 2 ^ 32；r指定每个轮班的数量；定义Left Rotatate 函数为 (x) << (c) | (x) >> (32 - (c))。

```
// Constants are the integer part of the sines of
integers (in radians) * 2^32.
const uint32_t k[64] = {
0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee ,
0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 ,
0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be ,
0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 ,
0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa ,
0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 ,
0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed ,
0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a ,
0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c ,
```

```
0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 ,
0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 ,
0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 ,
0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 ,
0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 ,
0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 ,
0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 };

// r specifies the per-round shift amounts
const uint32_t r[] = {7, 12, 17, 22, 7, 12, 17,
22, 7, 12, 17, 22, 7, 12, 17, 22, 5,  9, 14, 20,
5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 4,
11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11,
16, 23, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15,
21, 6, 10, 15, 21};

// leftrotate function definition
# define LEFTROTATE(x, c) (((x) << (c)) | ((x) >>
(32 - (c))))
```

数据填充

对消息进行数据填充，使消息的长度对512取模得448，设消息长度为X，即满足X mod 512=448。根据此公式得出需要填充的数据长度。填充方法：在消息后面进行填充，填充第一位为1，其余为0。填充完后，信息的长度就是512*N+448。之后，用剩余的位置（512-448=64位）记录原文的真正长度，把长度的二进制值补在最后。这样处理后的信息长度就是512*(N+1)。

```
// Fill Data
// append "1" bit to message
// append "0" bits until message length in bits ≡
448 (mod 512)
```

```
// append length mod (2^64) to message

for (new_len = initial_len + 1; new_len % (512/8)
!= 448/8; new_len++);

msg = (uint8_t*)malloc(new_len + 8);
memcpy(msg, initial_msg, initial_len);
msg[initial_len] = 0x80; // append the "1" bit;
most significant bit is "first"
for (offset = initial_len + 1; offset < new_len;
offset++)
msg[offset] = 0; // append "0" bits

// append the len in bits at the end of the
buffer.
to_bytes(initial_len*8, msg + new_len);
// initial_len>>29 == initial_len*8>>32, but
avoids overflow.
to_bytes(initial_len>>29, msg + new_len + 4);
```

设置初始值

MD5的哈希结果长度为128位，按每32位分成一组共4组。这4组结果是由4个初始值h_0～h_3经过不断演变得到。MD5的官方实现中，h_0～h_3的初始值如下（16进制）：

- h_0 = 0x67452301
- h_1 = 0xefcdab89
- h_2 = 0x98badcfe
- h_3 = 0x10325476

h_0	**01**	**23**	**45**	**67**
h_1	**89**	**AB**	**CD**	**EF**

h_2	FE	DC	BA	98
h_3	76	54	32	10

```
// Initialize variables - simple count in
nibbles:
h0 = 0x67452301;
h1 = 0xefcdab89;
h2 = 0x98badcfe;
h3 = 0x10325476;
```

循环运算

MD5所用到的函数有四种线性函数(&是与,| 是或,~是非,^是异或)。如果X、Y和Z的对应位是独立和均匀的，那么结果的每一位也应是独立和均匀的。利用上面的四种操作，生成四个重要的计算函数。首先我们声明中间变量a、b、c、d。这个循环的循环次数为512位分组的个数。每次循环执行64不计算，上述4个函数每个16次，具体如下：

```
// These vars will contain the hash
uint32_t h0, h1, h2, h3;

// Message (to prepare)
uint8_t *msg = NULL;
size_t new_len, offset;
uint32_t w[16];
uint32_t a, b, c, d, i, f, g, temp;

// Process the message in successive 512-bit
chunks:
// ForEach 512-bit chunk of message:
for(offset=0; offset<new_len; offset += (512/8))
{
```

```
    // break chunk into sixteen 32-bit words
w[j], 0 ≤ j ≤ 15
    for (i = 0; i < 16; i++)
        w[i] = to_int32(msg + offset + i*4);

    // Initialize hash value for this chunk:
    a = h0;
    b = h1;
    c = h2;
    d = h3;

    // Main loop:
    for(i = 0; i<64; i++) {
        if (i < 16) {
            f = (b & c) | ((~b) & d);
            g = i;
        } else if (i < 32) {
            f = (d & b) | ((~d) & c);
            g = (5*i + 1) % 16;
        } else if (i < 48) {
            f = b ^ c ^ d;
         g = (3*i + 5) % 16;
        } else {
            f = c ^ (b | (~d));
            g = (7*i) % 16;
        }
        temp = d;
        d = c;
        c = b;
        b = b + LEFTROTATE((a + f + k[i] + w[g]),
r[i]);
        a = temp;
    }

    // Add this chunk's hash to result so far:
    h0 += a;
```

```
    h1 += b;
    h2 += c;
    h3 += d;
}

// cleanup
free(msg);

//var char digest[16] := h0 append h1 append h2
append h3
to_bytes(h0, digest);
to_bytes(h1, digest + 4);
to_bytes(h2, digest + 8);
to_bytes(h3, digest + 12);
}
```

数据处理

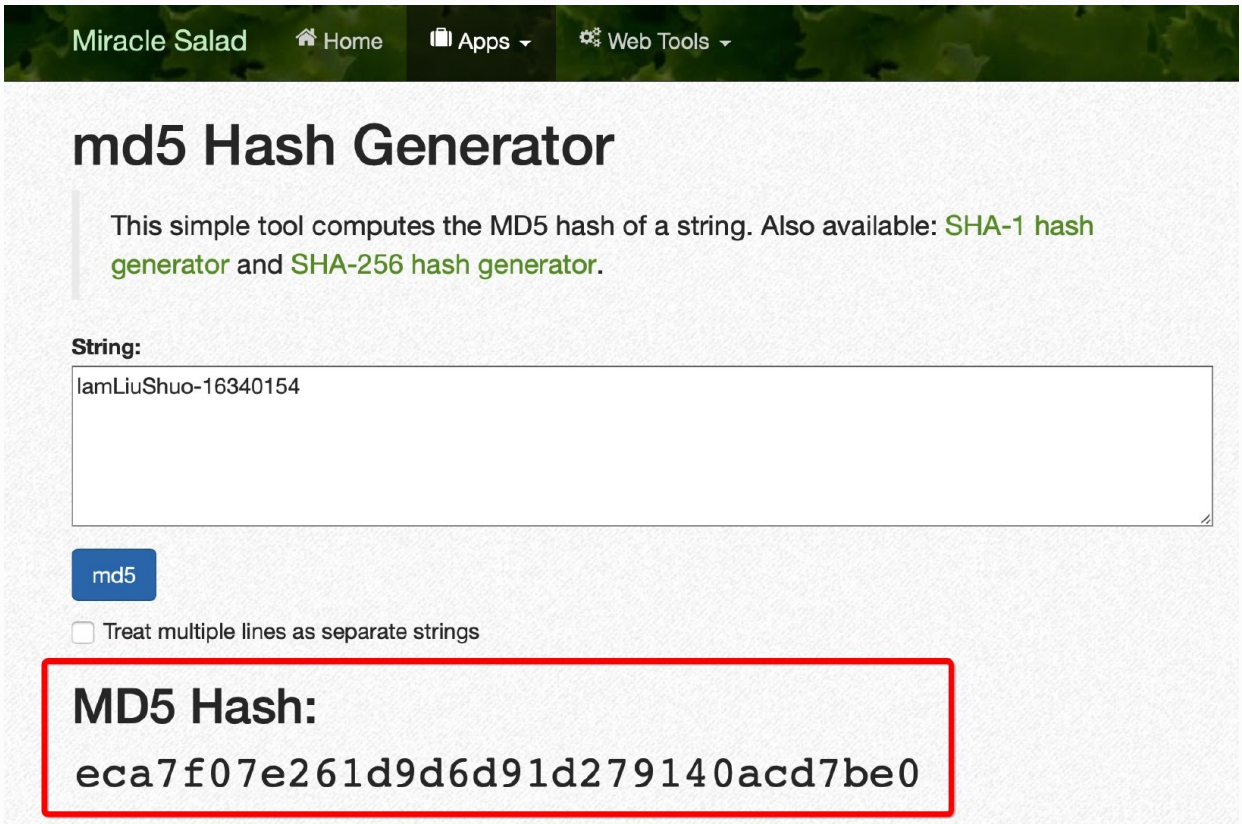
处理完所有的512位的分组后，得到一组新的$h_0 \sim h_3$的值，将这些值按$h_0 \sim h_3$的顺序级联，就得到了想要的MD5散列值。输出依然要考虑内存存储的大小端问题。

```
// benchmark
for (i = 0; i < 1000000; i++) {
    md5((uint8_t*)msg, len, result);
}
```

结果

编译运行MD5.c，得到可执行文件，输入字符串“*IamLiuShuo-16340154*”，并在miraclesalad尝试对比结果，发现一致，实验成功！

```
[NinodeMacBook-Pro:desktop nino$ gcc MD5.c
[NinodeMacBook-Pro:desktop nino$ ./a.out IamLiuShuo-16340154
eca7f07e261d9d6d91d279140acd7be0
```



参考

- MD5算法原理与实现
- MD5加密算法原理及实现
- MD5值算法原理