

# 实验四——HASH 函数 MD5

1611532 刘一静 信息安全

实验要求：

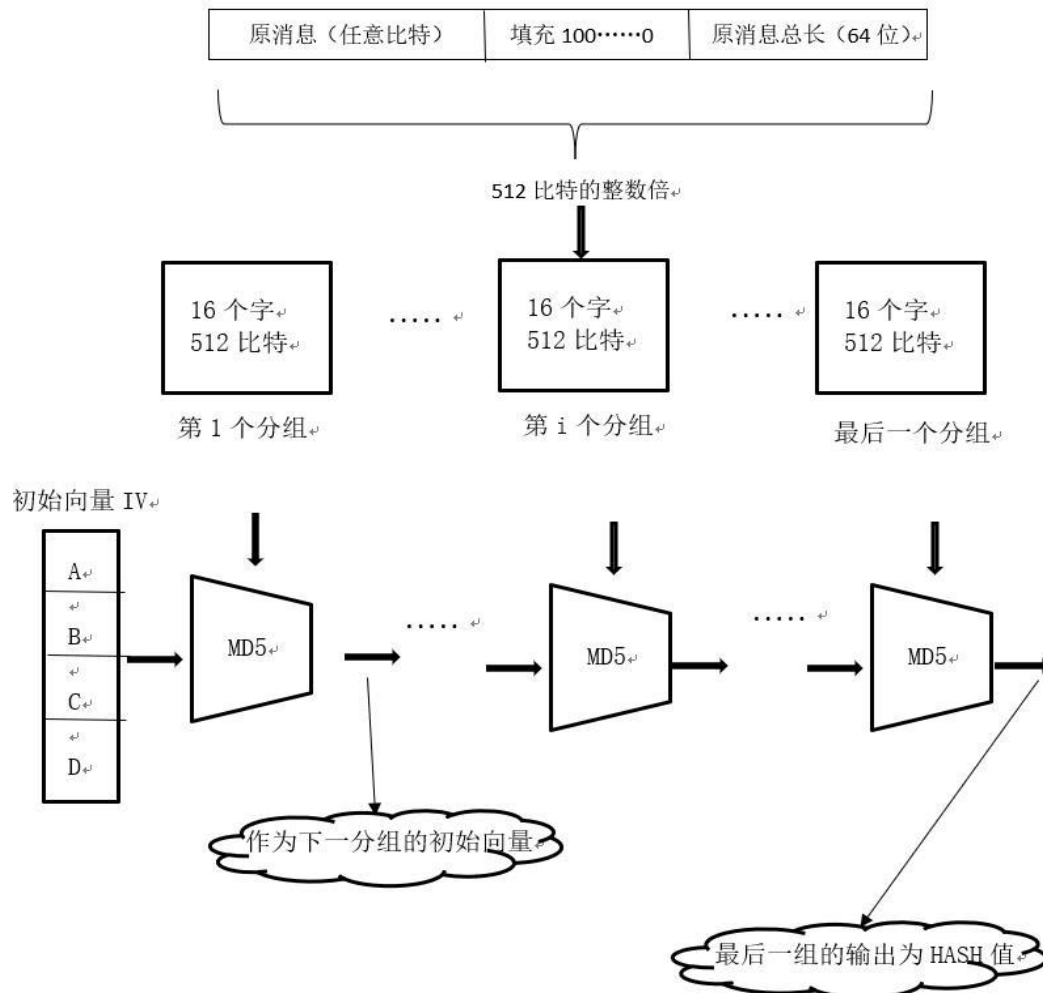
- 1、自己编写完整的 MD5 实现代码，并提交程序和程序流程图。
- 2、对编好的 MD5 算法，测试其雪崩效应，要求给出文本改变前和改变后的 Hash 值，并计算出改变的位数。写出 8 次测试的结果，并计算出平均改变的位数。

## 目录

一、MD5 算法.....	2
二、关键代码 .....	4
三、测试样例 .....	5
四、雪崩效应 .....	5

## 一、MD5 算法

MD5 是一种哈希函数，用来产生消息摘要，算法的输入可以是任意比特的明文消息，输出是固定的 128 比特哈希值。算法整体逻辑如下：



文字描述：

### 1. 消息填充

填充输入信息至  $N \times 512 + 448$  (即填充后的位数模 512 得 448)

填充内容为一个 1 和若干个 0，即第一个为 1 其他都是 0

填充完毕后，添加一个 64 位的字段，字段表示的是原字段长度，

这样就使得信息长度为  $N \times 512 + 448 + 64 = (N + 1) \times 512$ 。

## 2. 初始化变量(小端存储)

A = 0x67452301;

B = 0xefcdab89;

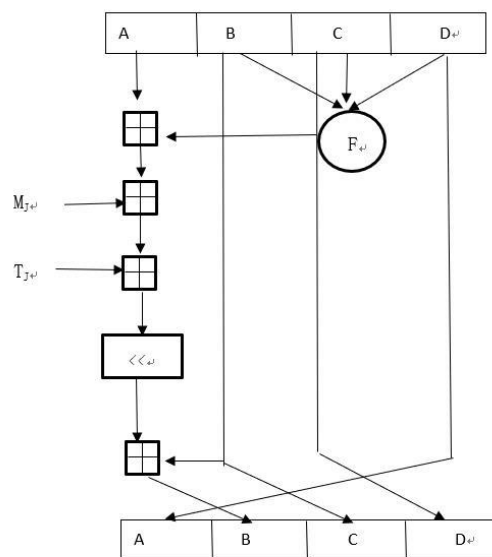
C = 0x98badcfe;

D = 0x10325476;

## 3. 核心运算

对每个分组进行 MD5 运算，输出的结果作为下一个分组输入的初始向量，最后一组的输出为计算得到的消息的哈希值。

每一分组的算法流程如下：



**文字描述：**每一个分组进行四轮计算，每轮循环都很相似。每轮进行 16 步操作。每步操作对 a、b、c 和 d 中的其中三个作一次非线性函数运算，然后将所得结果加上第四个变量，文本的一个子分组和一个常数。再将所得结果向左环移一个不定数，并加上 a、b、c 或 d 中之一。最后用该结果取代 a、b、c 或 d 中之一。表达式为：

$$b = b + (a + F(b, c, d) + Mj + Ti) \ll s)$$

## 注意：

1、每轮的非线性处理函数不同，分别为：

$$F(X, Y, Z) = (X \& Y) \mid ((\sim X) \& Z);$$

$$G(X, Y, Z) = (X \& Z) \mid (Y \& (\sim Z));$$

$$H(X, Y, Z) = X \wedge Y \wedge Z;$$

$$I(X, Y, Z) = Y \wedge (X \mid (\sim Z)).$$

2、分组中的每个字在每轮使用顺序也不同

第一轮为正常顺序；

第二轮第  $i$  步使用第  $(1 + 5 * i) \% 16$  个字；

第三轮第  $i$  步使用第  $(5 + 3 * i) \% 16$  个字；

第四轮第  $i$  步使用第  $(7 * i) \% 16$  个字。

## 二、关键代码

本次实验的代码量明显少于之前几次实验，并且代码结构很整齐。

主要使用的数据类型为 unsigned int 数组和 unsigned char 数组，unsigned char 是 8bit，1 字节；unsigned int 是 32bit，1 个字。

两个核心函数定义在 MD5 类的共有函数中：

```
1 string pad(const char * message); //填充
2 string fourround(unsigned int input[16]); //4轮处理
```

pad 函数得到 512 比特整数倍长度的一个数组，每次取 16 个元素，也就是 16 个字为一个分组，作为四轮处理函数的输入，最后一组调用 fourround 函数的返回值作为 pad 函数的返回值返回。这个值也就是所求的原始消息的哈希值。

代码如下：

```
1 string temp[100];
2 for (int i = 0; i < totalgroup; i++)
3 {
4     unsigned int input[16];
5     for (int j = 0; j < 16; j++)
6     {
7         input[j] = group[j+i*16];
8     }
9     temp[i] = fourround(input);
10 }
11 return temp[totalgroup - 1];
```

## 注意：大端小端问题

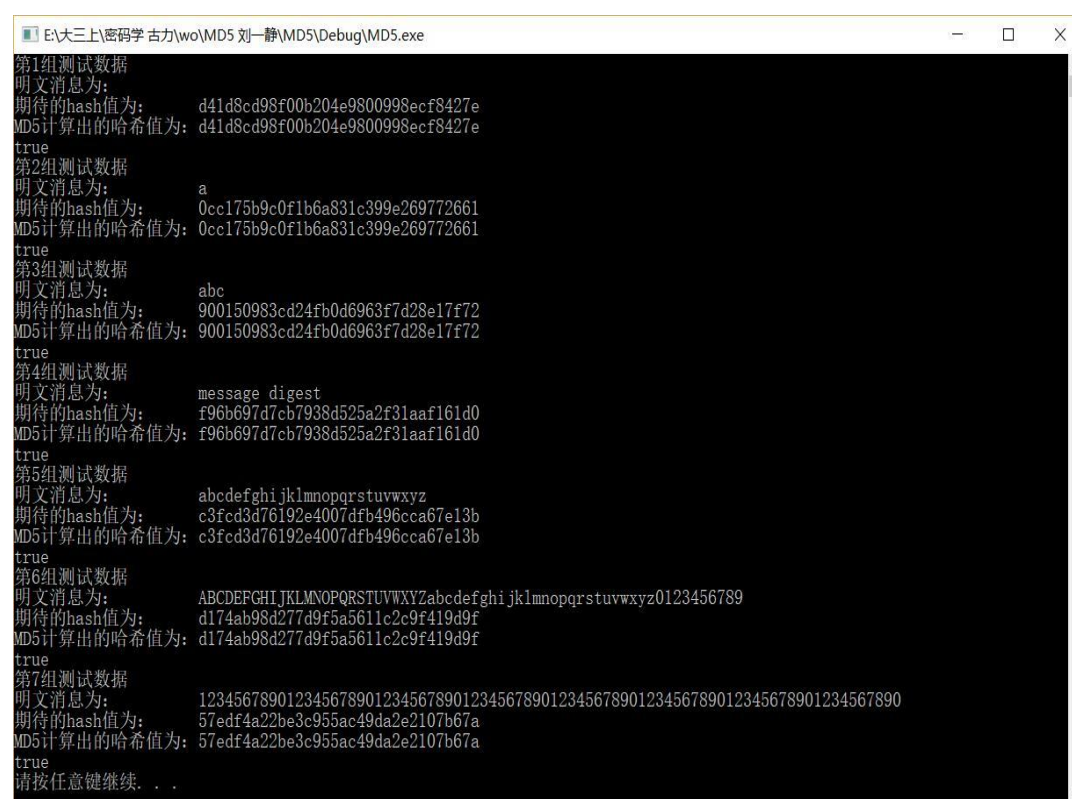
小端：就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

大端：就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

fourround 的返回值需要将 A, B, C, D 从小端编址转化为大端编址，再进行拼接，返回。

## 三、测试样例

使用文档中给出的测试样例进行测试，结果如下：



```
E:\大三上\密码学 古力\wo\MD5 刘一静\MD5\Debug\MD5.exe
第1组测试数据
明文消息为:
期待的hash值为: d41d8cd98f00b204e9800998ecf8427e
MD5计算出的哈希值为: d41d8cd98f00b204e9800998ecf8427e
true
第2组测试数据
明文消息为: a
期待的hash值为: 0cc175b9c0f1b6a831c399e269772661
MD5计算出的哈希值为: 0cc175b9c0f1b6a831c399e269772661
true
第3组测试数据
明文消息为: abc
期待的hash值为: 900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: 900150983cd24fb0d6963f7d28e17f72
true
第4组测试数据
明文消息为: message digest
期待的hash值为: f96b697d7cb7938d525a2f31aaf161d0
MD5计算出的哈希值为: f96b697d7cb7938d525a2f31aaf161d0
true
第5组测试数据
明文消息为: abcdefghijklmnopqrstuvwxyz
期待的hash值为: c3fcd3d76192e4007dfb496cca67e13b
MD5计算出的哈希值为: c3fcd3d76192e4007dfb496cca67e13b
true
第6组测试数据
明文消息为: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
期待的hash值为: d174ab98d277d9f5a5611c2c9f419d9f
MD5计算出的哈希值为: d174ab98d277d9f5a5611c2c9f419d9f
true
第7组测试数据
明文消息为: 1234567890123456789012345678901234567890123456789012345678901234567890
期待的hash值为: 57edf4a22be3c955ac49da2e2107b67a
MD5计算出的哈希值为: 57edf4a22be3c955ac49da2e2107b67a
true
请按任意键继续. . .
```

计算出来的 hash 值与期待的正确 hash 值均相同，MD5 实现正确。

## 四、雪崩效应

以“abc”字符串为例，每组数据改变 abc 对应 16 进制对应 2 进制中的一位。

将输出的 16 进制哈希值转化为 2 进制，比较预期与计算出的 128 比特哈希值改变的位数，输出结果如下：

```
第1组测试数据
明文消息为:      ibc
期待的hash值为:  900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: 63c3ae7301647113d04e201e02f67b18
false
期待哈希值的二进制:10010000000000010101000010011000001111001101001001001111101100001101011010010110001111110111110100101
000111000010111111101110010
计算哈希值的二进制:01100011110000111010111001110011000000010110010001110001000100111101000001001110001000000001111000000
010111101100111101100011000
改变的位数为: 68

第2组测试数据
明文消息为:      aba
期待的hash值为:  900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: 79af87723dc295f95bdb277a61189a2a
false
期待哈希值的二进制:10010000000000010101000010011000001111001101001001001111101100001101011010010110001111110111110100101
000111000010111111101110010
计算哈希值的二进制:011110011101011111000011101110010001111011100001010010101111110010101101111011011001001110111101001100
001000110001001101000101010
改变的位数为: 61

第3组测试数据
明文消息为:      abg
期待的hash值为:  900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: 894852696ee75656ba33c03041b1fa7f
false
期待哈希值的二进制:10010000000000010101000010011000001111001101001001001111101100001101011010010110001111110111110100101
000111000010111111101110010
计算哈希值的二进制:10001001010010000101001001101001011011101110011101010110010101101011101000110011110000000011000001000
001101100011111101001111111
改变的位数为: 59

第4组测试数据
明文消息为:      abk
期待的hash值为:  900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: 15ef0ee43032ec645b40f84193c045b5
false
期待哈希值的二进制:10010000000000010101000010011000001111001101001001001111101100001101011010010110001111110111110100101
000111000010111111101110010
计算哈希值的二进制:00010101111011110000111011100100001100000011001011101100011001000101101101000000111110000100000110010
0111110000000100010110110101
改变的位数为: 67
```

```
第6组测试数据
明文消息为:      abC
期待的hash值为:  900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: 36cf1fa7e384dfd385f07a50ffdf0418
false
期待哈希值的二进制:10010000000000010101000010011000001111001101001001001111101100001101011010010110001111110111110100101
000111000010111111101110010
计算哈希值的二进制:0011011011001111000111111010011111100011100001001101111110100111000010111110000011110100101000011111
111110111110000010000011000
改变的位数为: 73

第7组测试数据
明文消息为:      aBc
期待的hash值为:  900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: dbbbbe4975e026e04a687871f296a2b2
false
期待哈希值的二进制:10010000000000010101000010011000001111001101001001001111101100001101011010010110001111110111110100101
000111000010111111101110010
计算哈希值的二进制:11011011101110111110010010010111010111100000001001101110000001001001101000011110000111000111110
010100101101010001010110010
改变的位数为: 67

第8组测试数据
明文消息为:      Abc
期待的hash值为:  900150983cd24fb0d6963f7d28e17f72
MD5计算出的哈希值为: 35593b7ce5020eae3ca68fd5b6f3e031
false
期待哈希值的二进制:10010000000000010101000010011000001111001101001001001111101100001101011010010110001111110111110100101
000111000010111111101110010
计算哈希值的二进制:00110101010110010011101101111100111001010000001000001110101011100011110010100110100011111101010110110
110111100111110000000110001
改变的位数为: 59
```

统计出平均改变的位数为: 65 位。