
南開大學
Nankai University

# 南　　开　　大　　学

## 网 络 空 间 安 全 学 院

**密码学 (1091) 第五次实验**

---

# Hash 函数 MD5

---

苗发生

年级：2020 级

专业：信息安全 & 法学双学位

指导教师：古力

2022 年 12 月 12 日

# 目录

# 一、 实验内容

## (一) 实验目的

通过实际编程了解 MD5 算法的过程，加深对 Hash 函数的认识

## (二) 实验环境

运行 Windows 操作系统的 PC 机，具有 VC 等语言编译环境

## (三) MD5 算法

### 1. MD5 算法介绍

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。

Hash，一般翻译做散列、杂凑，或音译为哈希，是把任意长度的输入通过散列算法变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来确定唯一的输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。MD5 就是一种散列函数，它不能称作加密算法，也不是压缩算法，因为不能够通过 MD5 值得到原来的输入。

MD5 与对称和非对称加密算法不同，对称和非对称算法是防止信息被窃取，而 MD5 算法的目标是用于证明原文的完整性。

MD5 算法框图如下：

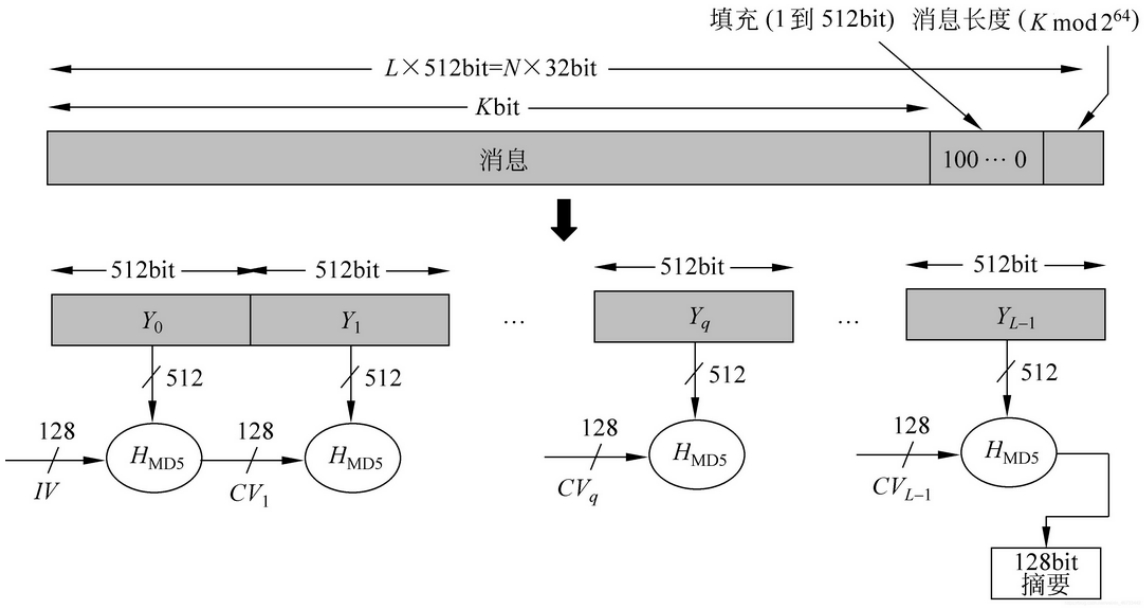


图 1: MD5 算法框图

### 2. MD5 的性质

1) 压缩性：任意长度的数据，算出的 MD5 值长度都是固定的 (相当于超损压缩)

2) 容易计算：从原数据计算出 MD5 值很容易

3) 抗修改性：对原数据进行任何改动，哪怕只修改 1 个字节，所得到的 MD5 值都有很大区别

4) 弱抗碰撞：已知原数据和其 MD5 值，想找到一个具有相同 MD5 值的数据（即伪造数据）是非常困难的

5) 强抗碰撞：想找到两个不同的数据，使它们具有相同的 MD5 值，是非常困难的

#### 3. MD5 算法作用

1) 防止数据被篡改（比如传输一个文件，并传输它的 md5 值，接收方可以对收到的文件进行 md5 计算后比较和发送过来的 md5 值是否一样，不一样说明不是同一个文件）

2) 防止直接显示明文（比如用在数据库存储密码，只需比较输入密码的 md5 值和数据库里的 md5 值是否相同，而不用传输明文；但数据库即使采用 md5 存储密码也是有可能出现安全漏洞的，比如计算常用密码的 md5 值，然后比较数据库中是否有同样的项）

3) 数字签名（因为难以由 md5 值得到原来的值，所以可以通过给出原来的值证明是否被篡改）

### （四） 实验内容和步骤

1) 算法分析:

请参照教材内容，分析 MD5 算法实现的每一步原理

2) 算法实现

利用 Visual C ++语言，自己编写 MD5 的实现代码，并检验代码实现的正确性。

3) 雪崩效应检验

尝试对一个长字符串进行 Hash 运算，并获得其运算结果。对该字符串进行轻微的改动，比如增加一个空格或标点，比较 Hash 结果值的改变位数。进行 8 次这样的测试。

### （五） 实验报告和要求

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

## 二、 实验具体流程

### （一） 算法分析

#### 1. 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，数据填充的流程图如下：

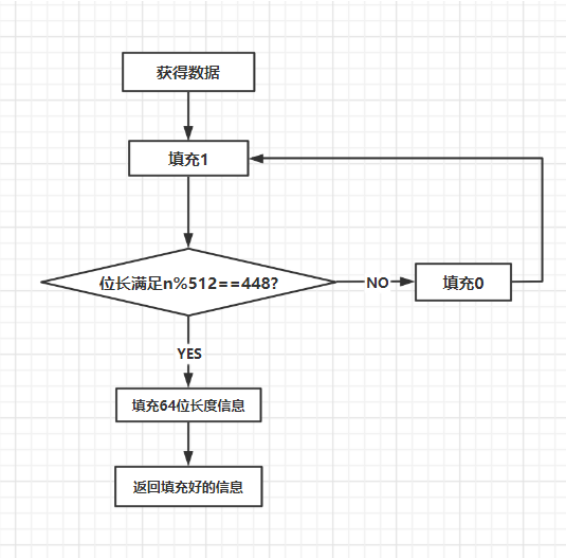


图 2: 数据填充流程图

需保证消息长度为 N*512+448

**2. 附加消息**

经过第一步整理完成后的数据的位数可以表示为 N*512+448，再向其后追加 64 位用来存储数据的长度，比如数据的长度为 16 字节，则用 10000 来填充后 64 位。这一步做完后，数据的位数将变成 (N+1)*512

**3. 以标准的幻数作为输入**

MD5 的实现需要每 512 个字节进行一次处理，后一次处理的输入为前一次处理的输出，因此，在循环处理开始之前，需要拿 4 个标准数作为输入，它们分别是：

1) A=0x67452301
2) B=0xefcdab89
3) C=0x98badcfe
4) D=0x10325476

**4. 进行 N 轮循环处理并输出最后的结果**

这一步重要的是每一轮的处理算法，每一轮处理也要循环 64 次，这 64 次循环被分为 4 组，每 16 次循环为一组，每组循环使用不同的逻辑处理函数，处理完成后，将输出作为输入进入下一轮循环

通过上面的标准 128bit 输入，参与每组 512bit 计算，得到一个新的 128 值，接着参与下一轮循环运算，最终得到一个 128 位的值, 具体运算如下：

这里用到 4 个逻辑函数 F,G,H,I，分别对应 4 轮运算，它们将参与运算。（4 轮 16 步）

1) 第一轮逻辑函数：$F(b,c,d) = (b\&c)|((\sim b)\&d)$ 参与第一轮的 16 步运算 (b,c,d 均为 32 位数)

2) 第二轮逻辑函数：$G(b,c,d) = (b\&d)|(c\&(\sim d))$ 参与第二轮的 16 步运算

3) 第三轮逻辑函数：$H(b,c,d) = b^c d$ 参与第三轮的 16 步运算

4) 第四轮逻辑函数：$I(b,c,d) = c \wedge (b|(\sim d))$ 参与第四轮的 16 步运算

5) 再引入一个移位函数 MOVE(X,n)，它将整型变量 X 左循环移 n 位，如变量 X 为 32 位，则 $MOVE(X,n) = (X << n)|(X >> (32-n))$

依据此，设计的主循环的流程图如下所示：

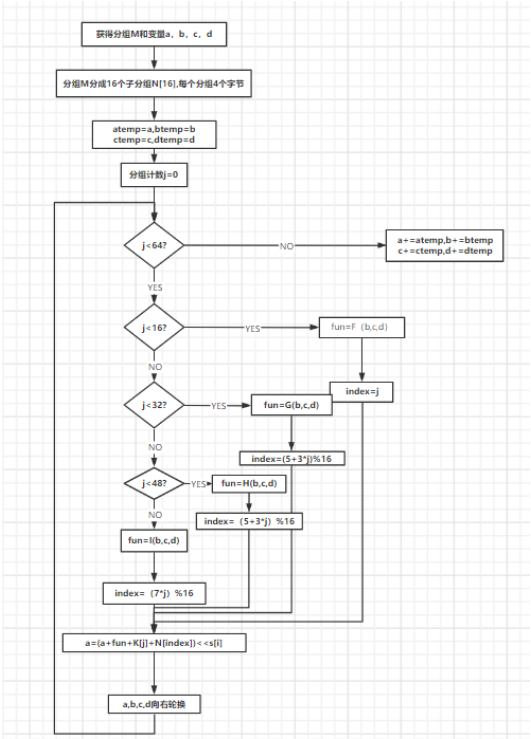


图 3: 主循环的流程图

## 5. 整体流程

依据上面的算法分析以及 MD5 加密的具体流程，做出算法的整体流程图，如下所示：

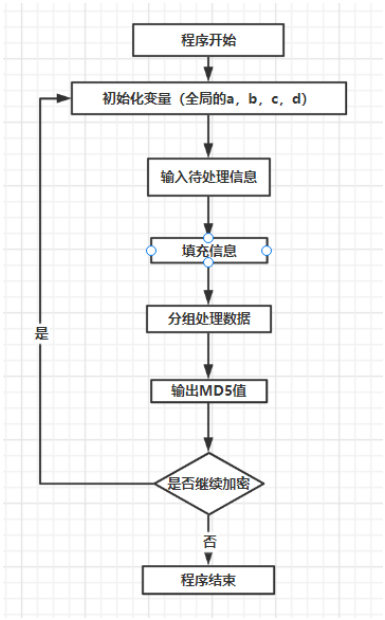


图 4: MD5 算法加密整体流程

## (二) MD5 加密

MD5 算法是单向散列算法的一种。单向散列算法也称为 HASH 算法，是一种将任意长度的信息压缩至某一固定长度（称之为消息摘要）的函数 (该压缩过程不可逆)。在 MD5 算法中，这个摘要是指将任意数据映射成一个 128 位长的摘要信息。并且其是不可逆的，即从摘要信息无法反向推演中原文，在演算过程中，原文的内容也是有丢失的。

### 1. MD5 类

为了使得代码具有可扩展性以及代码结构的美观，现将 MD5 相关参数和函数封装为 MD5 类，该类的实现如下:

MD5 类

```
class MD5 {
public:
        bool OVER;
        unsigned int Link[4];
        unsigned int Count[2];
        unsigned char Buffer[64];
        unsigned char Digest[16];
        static  unsigned char Padding[64];
        static  char hex[16];
        MD5(string& message);
        unsigned char* getDigest();
        string getstring();
        void Init(unsigned char* input, int len);
        void zip(unsigned char block[64]);
        void Encryption(unsigned int* input, unsigned char* output, int
            length);
```

```
};
```

该类中包含实现 MD5 加密所需的所有函数和必要参数

### 2. 初始化

我们通过将初始化封装为 Init () 函数进行初始化工作，我们设定了一个 count[2] 数组，他负责记录当前字符串 [位数]，然后我们获取当前已有的字节数（count[0]»3 再模 64），然后我们用 count[0]+len«3，即已有位数加上新增加的位数，然后我们判断是否有溢出，若有，则将高位 count[1]+1，接着我们让 count[1] 获取高位的位数，即先让 len 右移 32 得到高位，再左移 3 位得到位数，即 count[1]+=len»29。然后我们将新加入字符长度与待填充长度进行比较，在第一次调用此函数时，等同于我们将输入的字符串字节数与 64 字节数即 512 比特进行比较，若大于等于，则说明我们可以先将数据按 64 字节分组，先对这些 64 字节的组执行压缩函数，然后再对最后一组不足 64 字节的数据执行后续填充操作等。具体 C++ 实现如下：

初始化

```
void MD5::Init(unsigned char* input, int len) {
        unsigned int i, nowlength, waitlength;
        OVER = false;
        nowlength = (unsigned int)((Count[0] >> 3) & 0x3f);
        Count[0] += (unsigned int)len << 3;
        if ((Count[0]) < ((unsigned int)len << 3)) {
                Count[1] += 1;
        }
        Count[1] += ((unsigned int)len >> 29);
        waitlength = 64 - nowlength;
        if (len >= waitlength) {
                memcpy(&Buffer[nowlength], input, waitlength);
                zip(Buffer);
                for (i = waitlength; i + 63 < len; i += 64) {
                        zip(&input[i]);
                }
                nowlength = 0;
        }
        else {
                i = 0;
        }
        memcpy(&Buffer[nowlength], &input[i], len - i);
}
```

### 3. MD5 压缩

压缩函数 $H_{MD5}$ 中有四轮处理过程，每轮又对缓冲区 ABCD 进行 16 步迭代运算，在每一步的迭代中，有四个关键的逻辑函数，实现如下：

逻辑函数

```
unsigned int Left_shift(unsigned int& num, unsigned int& n)
{
```

```
        return (num << n) | (num >> (32 - n));
}
void Round_One(unsigned int& a, unsigned int& b, unsigned int& c, unsigned
    int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
        a += F(b, c, d) + x + ac;
        a = Left_shift(a, s);
        a += b;
}
void Round_Two(unsigned int& a, unsigned int& b, unsigned int& c, unsigned
    int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
        a += G(b, c, d) + x + ac;
        a = Left_shift(a, s);
        a += b;
}
void Round_Three(unsigned int& a, unsigned int& b, unsigned int& c, unsigned
    int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
        a += H(b, c, d) + x + ac;
        a = Left_shift(a, s);
        a += b;
}
void Round_Four(unsigned int& a, unsigned int& b, unsigned int& c, unsigned
    int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
        a += I(b, c, d) + x + ac;
        a = Left_shift(a, s);
        a += b;
}
```

基于此逻辑函数实现的 MD5 压缩算法如下:

MD5 压缩

```
void MD5::zip(unsigned char block[64]) {
        unsigned int a = Link[0], b = Link[1], c = Link[2], d = Link[3], x
            [16];
        for (int i = 0, j = 0; j < 64; ++i, j += 4) {
                x[i] = ((unsigned int)block[j]) | (((unsigned int)block[j +
                    1]) << 8) |
                        (((unsigned int)block[j + 2]) << 16) | (((unsigned
                            int)block[j + 3]) << 24);
        }
        Round_One(a, b, c, d, x[0], s[0][0], 0xd76aa478);
    ......................

        Round_Two(a, b, c, d, x[1], s[1][0], 0xf61e2562);
    ......................

```

```
        Round_Three(a, b, c, d, x[5], s[2][0], 0xfffa3942);
    ......................
        Round_Four(a, b, c, d, x[0], s[3][0], 0xf4292244);
    ......................
        Link[0] += a;
        Link[1] += b;
        Link[2] += c;
        Link[3] += d;
}
```

### 4. MD5 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。依据此实现的 C++ 代码如下：

填充

```
unsigned char* MD5::getDigest() {
        if (!OVER) {
                OVER = true;
                unsigned char Bits[8];
                unsigned int oldState[4];
                unsigned int oldCount[2];
                int nowlength, waitlength;
                memcpy(oldState, Link, 16);
                memcpy(oldCount, Count, 8);
                Encryption(Count, Bits, 8);
                nowlength = (unsigned int)((Count[0] >> 3) & 0x3f);
                waitlength = (nowlength < 56) ? (56 - nowlength) : (120 -
                    nowlength);
                Init(Padding, waitlength);
                nowlength = (unsigned int)((Count[0] >> 3) & 0x3f);
                memcpy(&Buffer[nowlength], Bits, 8);
                zip(Buffer);
                Encryption(Link, Digest, 16);
                memcpy(Link, oldState, 16);
                memcpy(Count, oldCount, 8);
        }
        return Digest;
}
```

### 5. MD5 加密

在上面实现的逻辑函数，压缩函数，填充函数的基础上，通过调用相关函数实现 MD5 加密，C++ 代码实现如下：

MD5 加密

```
void MD5::Encryption(unsigned int* Input_txt, unsigned char* Result, int Len)
    {
        for (int i = 0, j = 0; j < Len; ++i, j += 4) {
                Result[j] = (unsigned char)(Input_txt[i] & 0xff);
                Result[j + 1] = (unsigned char)((Input_txt[i] >> 8) & 0xff);
                Result[j + 2] = (unsigned char)((Input_txt[i] >> 16) & 0xff);
                Result[j + 3] = (unsigned char)((Input_txt[i] >> 24) & 0xff);
        }
}
```

### 6. 打印结果

为了使得输出为 16 进制，且每两个 16 进制数为一组，再加前缀 0X，故需要对输出的结果进行格式化控制，C++ 代码实现如下：

格式化输出函数

```
void Print(string& txt) {
        string md5 = MD5(txt).getstring();
        cout << "MD5: " << endl;
        //cout << "0x";
        for (int i = 0; i < 32; i++) {
                if (i % 2 == 0 && i != 0&&i!=16)
                        cout << "  ";
                if (i % 2 == 0) {
                        cout << "0x";
                }
                cout << md5[i];
                if (i == 15)
                        cout << endl;
        }
        cout << endl;
        return;
}
```

### 7. 测试

通过利用提前给定的样例进行测试，本次 MD5 加密程序在加密完成后可以选择是否继续进行加密，如果继续进行加密，则输入 1，否则输入 0，根据提示进行操作即可，测试结果如下：

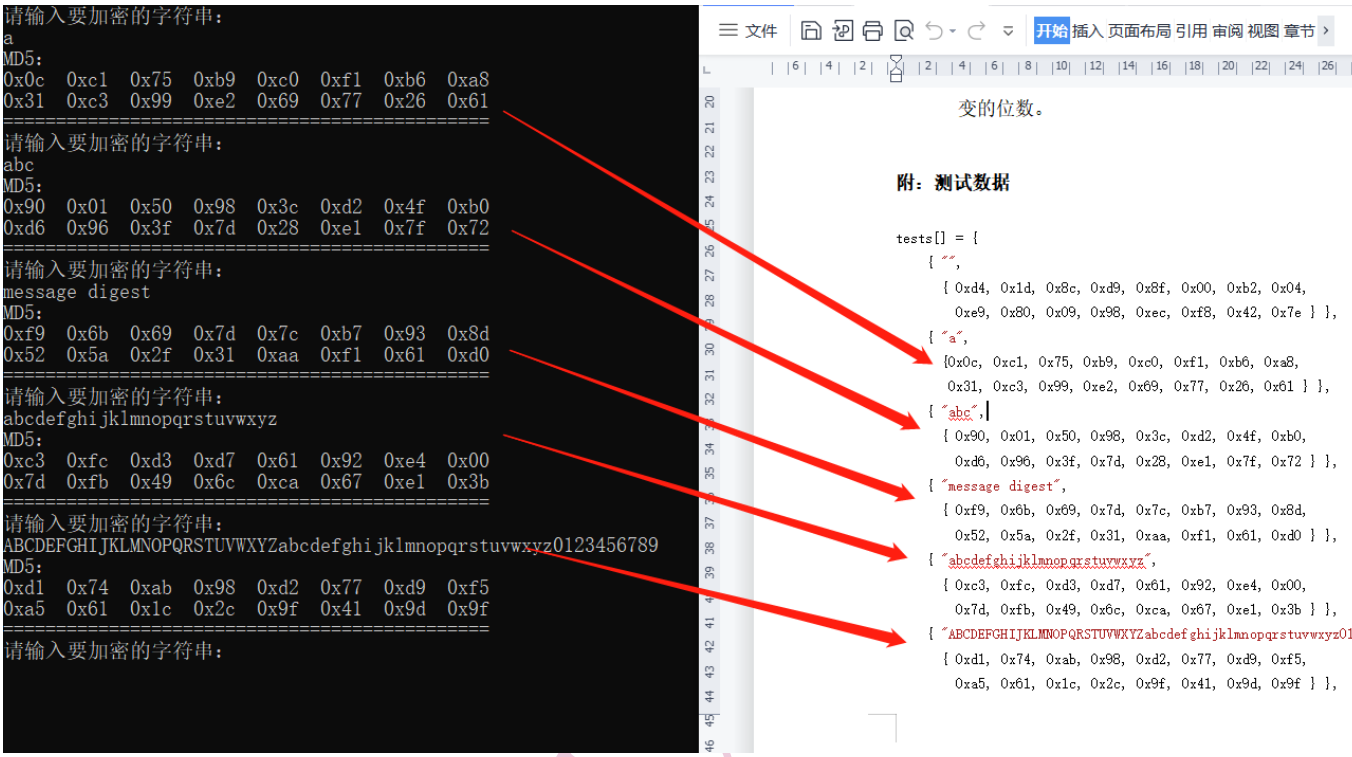


图 5: MD5 算法测试

对结果进行对比分析，MD5 加密成功！

## (三) 雪崩效应检测

检测 MD5 算法的雪崩效应，只需要利用 for 循环每次更改明文的一个 bit，之后分别进行 MD5 加密，之后对得到的结果进行统计 Bit 的改变数，因此，检测 MD5 的雪崩效应主要有两部分：

1) 逐比特更改明文串，此部分利用 for 循环来实现

2) 将更改后的明文进行加密后，统计得到的 MD5 值和原始 MD5 值相差的比特个数

### 1. 逐比特更改明文并进行 MD5 加密

由于 2 进制只有 0 和 1，故我们只需要判断明文中对应字符的 ASCII 值为奇数还是偶数，若为偶数则 +1，若为奇数则-1，进而实现比特翻转，具体 C++ 代码实现如下：

逐比特更改待加密明文

```
for (int i = 0; i < Times; i++) {
        cout << "进行第" << i + 1 << "次雪崩测试" << endl;
        cout << "原始消息: "<< Initialtxt <<endl;
        NewChangetxt[i] = Initialtxt;
        if (NewChangetxt[i][i] % 2 == 1)
                NewChangetxt[i][i] -= 1;
        else
                NewChangetxt[i][i] += 1;
        cout << "新消息" << i + 1 << ": " << NewChangetxt[i] << endl;
```

```
                cout << "原始MD5: " << endl;
                Print(Initialtxt);
                NewChangeMD5[i] = MD5(NewChangetxt[i]).getstring();
                cout << "新MD5" << i + 1 << ": "<<endl;
                Print(NewChangetxt[i]);
                MD5Change[i] = Compaer_bit_count(NewChangeMD5[i], InitialMD5)
                    ;
                cout << "摘要相差二进制位数: " << MD5Change[i] << endl; cout
                    << endl << endl;
        }
        cout << "摘要位数改变情况: " << endl;
```

### 2. 统计更改明文后的雪崩效应

由于每次更改明文后均需要进行统计 Bit 更改的个数,故我们将其封装为 Compare_Bit_Count 函数，具体实现如下:

统计 MD5 值更改的位数

```
int Compaer_Bit_Count(string NewSumry, string OrigiSumry)
{
        int Difference = 0;
        for (int part = 0; part < 8; part++)
        {
                char* END_one;
                char* End_two;
                string NewSumry_temp = NewSumry.substr(0 + 4 * part, 4);
                long NewMD5_Int = static_cast<long>(strtol(NewSumry_temp.
                    c_str(), &END_one, 16));
                string OrigiSumry_temp = OrigiSumry.substr(0 + 4 * part, 4);
                long OrigiMD5_Int = static_cast<long>(strtol(OrigiSumry_temp.
                    c_str(), &End_two, 16));
                for (int round = 0; round < 16; round++)
                {
                        if (NewMD5_Int % 2 != OrigiMD5_Int % 2)
                        {
                                Difference++;
                        }
                        NewMD5_Int /= 2;
                        OrigiMD5_Int /= 2;
                }
        }
        return Difference;
}
```

### 3. 测试雪崩效应

我们的雪崩效应检测了 48 次，每次进行逐比特翻转明文后统计摘要更改的位数。由于部分 ASCII 值对应的字符不可打印，即逐比特反转后的字符串不存在，故我们选择”ABCDEFGHI-

JKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw“字符串进行测试。

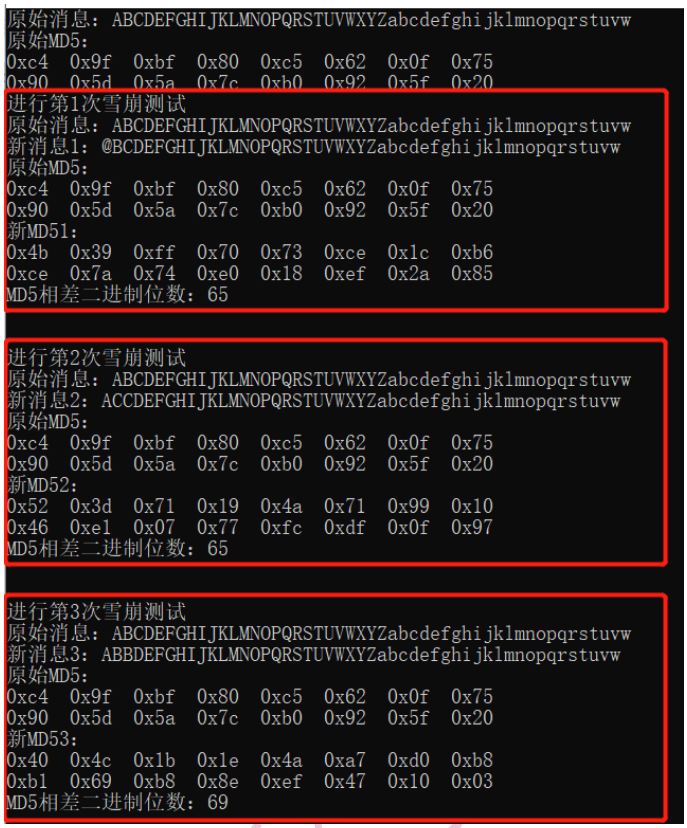


图 6: 检测雪崩效应

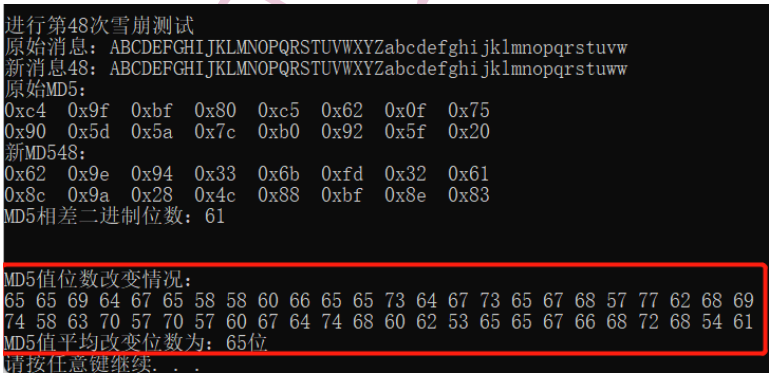


图 7: 统计雪崩效应平均每次更改位数

分析结果可知：每次更改一个 Bit，重复 48 次，MD5 加密结果平均每次变化 65 位，检测到雪崩效应，实验成功！