网络安全技术

实验报告

学院：网安学院

年级：2020

班级：信安班

学号：2011395

姓名：魏伯繁

手机号：18611111192

2023 年 6 月 4 日

目录

1 实验目标

MD5 算法是目前最流行的一种信息摘要算法，在数字签名，加密与解密技术，以及文件完整性检测等领域中发挥着巨大的作用。熟悉 MD5 算法对开发网络应用程序，理解网络安全的概念具有十分重要的意义本章编程训练的目的如下：

1. 深入理解 MD5 算法的基本原理。
2. 掌握利用 MD5 算法生成数据摘要的所有计算过程。
3. 掌握 Linux 系统中检测文件完整性的基本方法。
4. 熟悉 Linux 系统中文件的基本操作方法

2 实验内容

本章编程训练的要求如下：

1. 准确地实现 MD5 算法的完整计算过程。
2. 对于任意长度的字符串能够生成 128 位 MD5 摘要。
3. 对于任意大小的文件能够生成 128 位 MD5 摘要。
4. 通过检查 MD5 摘要的正确性来检验原文件的完整性。

在 Linux 平台下编写应用程序，正确地实现 MD5 算法。要求程序不仅能够为任意长度的字符串生成 MD5 摘要，而且可以为任意大小的文件生成 MD5 摘要。同时，程序还可以利用 MD5 摘要验证文件的完整性。验证文件完整性分为两种方式：一种是在手动输入 MD5 摘要的条件下，计算出当前被测文件的 MD5 摘要，再将两者进行比对。若相同，则文件完好；否则，文件遭到破坏。另一种是先利用 Linux 系统工具 md5sum 为被测文件生成一个后缀为.md5 的同名文件，然后让程序计算出被测文件的 MD5 摘要，将其与.md5 文件中的 MD5 摘要进行比较，最后得出检测结果。

3 实验步骤

3.1 学习 MD5 基本原理

MD5（message-digest algorithm 5 的缩写）是一种信息摘要算法。信息摘要算法的研究由来已久，早在 1990 年 Ronald L. Rivest 就提出了与 MD5 属于同一系列的散列函数 MD4。经过大量的密码学分析与不断的攻击检测，Rivest 于 1991 年提出了它的改进算法 MD5。

目前 MD5 已经得到了广泛的应用，成为许多机构和组织的散列函数标准。作为散列函数，MD5 有两个重要的特性：第一、任意两组数据经过 MD5 运算后，生成相同摘要的概率微乎其微；第二、即便在算法与程序已知的情况下，也不能够从 MD5 摘要中获得原始的数据。

MD5 的典型应用就是为一段信息（message）生成信息摘要（message-digest），以防止被篡改。Linux 系统自带了计算和校验 MD5 摘要的工具程序—md5sum。用户可以在命令行终端直接运行该程序。md5sum 程序可以创建一个与原始文件名称相同，后缀为.md5 的文件，并将原始文件的 MD5 摘要保存在该文件中。

另外，md5sum 程序可以对文件的完整性进行检测。它通过重新计算原始文件的 MD5 摘要，将计算结果与同名的.md5 文件中的摘要进行对比，若相同，则文件完整；否则，原文件受到了破坏。如下所示，共包含了 2 次检验。第一次通过比较 nankai.md5 中的摘要验证文件 nankai.txt 是完整的。第二次修改了 nankai.txt 文件的内容（在标题后面增加了 written by sky），导致校验失败。

3.2 学习 MD5 算法

MD5 算法以 512 比特为单位对输入消息进行分组。每个分组是一个 512 比特的数据块。同时每个数据块又由 16 个 32 比特的子分组构成。摘要的计算过程就是以 512 比特数据块为单位进行的。

在 MD5 算法中，首先需要对输入消息进行填充，使其比特长度对 512 求余的结果等于 448。也就是说，消息的比特长度将被扩展至 N×512+448，即 N×64+56 个字节，其中 N 为正整数。

具体的填充方法如下：在消息的最后填充一位 1 和若干位 0，直到满足上面的条件时才停止用 0 对信息进行填充。然后在这个结果后面加上一个以 64 位二进制表示的填充前的消息长度。经过这两步的处理，现在消息比特长度 =N×512+448+64=(N+1) ×512。因为长度恰好是 512 的整数倍，所以在下一步中可以方便地对消息进行分组运算。

3.2.1 MD5 初始向量

MD5 算法中有四个 32 比特的初始向量。它们分别是：A=0x01234567，B=0x89abcdef，C=0xfedcba98，D=0x76543210。

3.2.2 五种基本运算

MD5 的五种基本运算分别是：

1. 与运算：对于两个二进制数的每一位，只有两位都是 1 的时候，结果的该位才是 1，否则为 0。
2. 或运算：对于两个二进制数的每一位，只要两位中的任意一位是 1，结果的该位就是 1，否则为 0。
3. 非运算：对于二进制数的每一位，0 变 1，1 变 0。
4. 异或运算：对于两个二进制数的每一位，只有两位不相同时，结果的该位才是 1，否则为 0。
5. 左移运算：将二进制数向左移动指定位数，右端补 0。

在 MD5 加密算法中，这五种运算都被广泛地使用。其中，左移运算用于处理消息分组的位移操作，其它四种运算则用于对数据进行处理和置换，以实现 MD5 的分组加密。

3.2.3 MD5 基本函数

MD5 算法中 4 个非线性基本函数分别用于 4 轮计算。它们分别是：

a. $F(x,y,z)=(x\wedge y)\vee(x\wedge z)$

b. $G(x,y,z)=(x\wedge z)\vee(y\wedge z)$

c. $H(x,y,z)=x\oplus y\oplus z$

d. $I(x,y,z)=y\oplus(x\vee z)$

图 3.1: MD5 基本函数

在设置好 4 个初始向量以后，就进入了 MD5 的循环过程。循环过程就是对每一个消息分组的计算过程。每一次循环都会对一个 512 比特消息块进行计算，因此循环的次数就是消息中 512 比特分组的数目，即上面的（N+1）。

MD5 循环体中包含了 4 轮计算（MD4 只有 3 轮），每一轮计算进行 16 次操作。每一次操作可概括如下：

a. 将向量 A、B、C、D 中的其中三个作一次非线性函数运算。

b. 将所得结果与剩下的第四个变量、一个 32 比特子分组 X[k] 和一个常数 T[i] 相加。

c. 将所得结果循环左移 s 位，并加上向量 A、B、C、D 其中之一；最后用该结果取代 A、B、C、D 其中之一的值。

在第一轮计算中，如果用表达式 $FF[abcd,k,s,i]$ 表示如下的计算过程，那么第一轮的 16 次操作可以表示为：

$$a=b+\left((a+I(b,c,d)+X[k]+T[i])<<<s\right)$$

FF[ABCD, 0, 7, 1]	FF [DABC, 1, 12, 2]	FF [CDAB, 2, 17, 3]	FF [BCDA, 3, 22, 4]
FF [ABCD, 4, 7, 5]	FF [DABC, 5, 12, 6]	FF [CDAB, 6, 17, 7]	FF [BCDA, 7, 22, 8]
FF [ABCD, 8, 7, 9]	FF [DABC, 9, 12, 10]	FF [CDAB, 10, 17, 11]	FF [BCDA, 11, 22, 12]
FF [ABCD, 12, 7, 13]	FF [DABC, 13, 12, 14]	FF [CDAB, 14, 17, 15]	FF [BCDA, 15, 22, 16]

在第二轮计算中，如果用表达式 $GG[abcd,k,s,i]$ 表示如下的计算过程，那么第二轮的 16 次操作可以表示为：

$$a=b+\left((a+G(b,c,d)+X[k]+T[i])<<<s\right)$$

GG [ABCD, 1, 5, 17]	GG [DABC, 6, 9, 18]	GG [CDAB, 11, 14, 19]	GG [BCDA, 0, 20, 20]
GG [ABCD, 5, 5, 21]	GG [DABC, 10, 9, 22]	GG [CDAB, 15, 14, 23]	GG [BCDA, 4, 20, 24]
GG [ABCD, 9, 5, 25]	GG [DABC, 14, 9, 26]	GG [CDAB, 3, 14, 27]	GG [BCDA, 8, 20, 28]
GG [ABCD, 13, 5, 29]	GG [DABC, 2, 9, 30]	GG [CDAB, 7, 14, 31]	GG [BCDA, 12, 20, 32]

在第三轮计算中，如果用表达式 HH[abcd, k, s, i]表示如下的计算过程，那么第三轮的 16 次操作可以表示为：

$$a = b + \left(\left(a + H(b,c,d) + X[k] + T[i] \right) <<< s \right)$$

HH [ABCD, 5, 4, 33]	HH [DABC, 8, 11, 34]	HH [CDAB, 11, 16, 35]	HH [BCDA, 14, 23, 36]
HH [ABCD, 1, 4, 37]	HH [DABC, 4, 11, 38]	HH [CDAB, 7, 16, 39]	HH [BCDA, 10, 23, 40]
HH [ABCD, 13, 4, 41]	HH [DABC, 0, 11, 42]	HH [CDAB, 3, 16, 43]	HH [BCDA, 6, 23, 44]
HH [ABCD, 9, 4, 45]	HH [DABC, 12, 11, 46]	HH [CDAB, 15, 16, 47]	HH [BCDA, 2, 23, 48]

在第四轮计算中，如果用表达式 II[abcd, k, s, i]表示如下的计算过程，那么第四轮的 16 次操作可以表示为：

$$a = b + \left(\left(a + I(b,c,d) + X[k] + T[i] \right) <<< s \right)$$

II [ABCD, 0, 6, 49]	II [DABC, 7, 10, 50]	II [CDAB, 14, 15, 51]	II [BCDA, 5, 21, 52]
II [ABCD, 12, 6, 53]	II [DABC, 3, 10, 54]	II [CDAB, 10, 15, 55]	II [BCDA, 1, 21, 56]

II [ABCD, 8, 6, 57]	II [DABC, 15, 10, 58]	II [CDAB, 6, 15, 59]	II [BCDA, 13, 21, 60]
II [ABCD, 4, 6, 61]	II [DABC, 11, 10, 62]	II [CDAB, 2, 15, 63]	II [BCDA, 9, 21, 64]

在上面式子中，X[k]表示一个 512 比特消息块中的第 k 个 32 比特大小的子分组。也就是说，一个 512 比特数据块是由 16 个 32 比特的子分组构成的 。常数 T[i]表示 4294967296×abs(sin(i))的整数部分。4294967296 是 2 的 32 次方，abs(sin(i))是对 i 的正弦取绝对值，其中 i 以弧度为单位。

如下式所示，在 4 轮计算结束后，将向量 A、B、C、D 中的计算结果分别与向量 A_0、B_0、C_0、D_0 相加，最后将结果重新赋给向量 A、B、C、D。

$$A = A_0 + A \quad B = B_0 + B \quad C = C_0 + C \quad D = D_0 + D$$

至此，对一个 512 比特消息块的运算过程已经介绍完毕。MD5 算法将通过不断地循环，计算所有的消息块，直到处理完最后一块消息分组为止。

3.3 生成摘要

最后将 4 个 32 比特向量 A、B、C、D 按照从低字节到高字节的顺序拼接成 128 比特的摘要。

3.4 代码编写

首先对需要生成的常量进行初始化

```
1  #define A 0x67452301
2  #define B 0xefcdab89
3  #define C 0x98badcfe
4  #define D 0x10325476
5
```

```
const char str16[] = "0123456789abcdef";

const unsigned int T[] = {
    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 };

const unsigned int s[] = { 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};
```

然后按照实验指导书的要求对 MD5 类进行初始化，并定义其中的与或非异或以及位移函数，并通过对这些函数的拼接定义 F G H I 四个基本函数。之后定义一些需要在后续重复使用的循环压缩、编码函数、字符串填充以及输出函数

```
class MD5 {
private:
    unsigned int tempA, tempB, tempC, tempD, strlength;
public:
MD5() {
    tempA = A;
```

```
7       tempB = B;
8       tempC = C;
9       tempD = D;
10      strlength = 0;
11  }
12  // F 函数
13  unsigned int F(unsigned int b, unsigned int c, unsigned int d) {
14      return (b & c) | ((~b) & d);
15  }
16  // G 函数
17  unsigned int G(unsigned int b, unsigned int c, unsigned int d) {
18      return (b & d) | (c & (~d));
19  }
20  // H 函数
21  unsigned int H(unsigned int b, unsigned int c, unsigned int d) {
22      return b ^ c ^ d;
23  }
24  // I 函数
25  unsigned int I(unsigned int b, unsigned int c, unsigned int d) {
26      return c ^ (b | (~d));
27  }
28  // 移位操作函数
29  unsigned int shift(unsigned int a, unsigned int n) {
30      return (a << n) | (a >> (32 - n));
31  }
32  // 编码函数
33  string encode(string src) {
34      vector<unsigned int> rec = padding(src);
35      for(unsigned int i = 0; i < strlength/16; i++) {
36          unsigned int num[16];
37          for(int j = 0; j < 16; j++) {
38              num[j] = rec[i*16+j];
39          }
40          iterateFunc(num, 16);
41      }
```

```
42      return format(tempA) + format(tempB) + format(tempC) + format(tempD);
43  }
44  // 循环压缩
45  void iterateFunc(unsigned int* X, int size = 16) {
46      unsigned int a = tempA,
47                   b = tempB,
48                   c = tempC,
49                   d = tempD,
50                   rec = 0,
51                   g, k;
52      for(int i = 0; i < 64; i++) {
53          if(i < 16) {
54              // F 迭代
55              g = F(b, c, d);
56              k = i;
57          }
58          else if(i < 32) {
59              // G 迭代
60              g = G(b, c, d);
61              k = (1 + 5*i) % 16;
62          }
63          else if(i < 48) {
64              // H 迭代
65              g = H(b, c, d);
66              k = (5 + 3*i) % 16;
67          }
68          else {
69              // I 迭代
70              g = I(b, c, d);
71              k = (7*i) % 16;
72          }
73          rec = d;
74          d = c;
75          c = b;
76          b = b + shift(a + g + X[k] + T[i], s[i]);
```

```
77              a = rec;
78          }
79          tempA += a;
80          tempB += b;
81          tempC += c;
82          tempD += d;
83      }
84      // 填充字符串
85      vector<unsigned int> padding(string src) {
86          // 以 512 位,64 个字节为一组
87          unsigned int num = ((src.length() + 8) / 64) + 1;
88          vector<unsigned int> rec(num*16);
89          strlength = num*16;
90          for(unsigned int i = 0; i < src.length(); i++){
91              // 一个 unsigned int 对应 4 个字节, 保存 4 个字符信息
92              rec[i>>2] |= (int)(src[i]) << ((i % 4) * 8);
93          }
94          // 补充 1000...000
95          rec[src.length() >> 2] |= (0x80 << ((src.length() % 4)*8));
96          // 填充原文长度
97          rec[rec.size()-2] = (src.length() << 3);
98          return rec;
99      }
100     // 整理输出
101     string format(unsigned int num) {
102         string res = "";
103         unsigned int base = 1 << 8;
104         for(int i = 0; i < 4; i++) {
105             string tmp = "";
106             unsigned int b = (num >> (i * 8)) % base & 0xff;
107             for(int j = 0; j < 2; j++) {
108                 tmp = str16[b%16] + tmp;
109                 b /= 16;
110             }
111             res += tmp;
```

```
112     }
113     return res;
114 }
115
```

最后，根据实验指导书的定义，我们还需要再用户给出不同的命令行参数时进行不同的处理规则，比如输出帮助信息，输出示例，计算指定文件的 MD5 值，给定一个 MD5 值以及一个文件判定文件是否被修改，MD5 值的给定方式可以是命令行输入也可以从.MD5 文件中进行读取。

```
1  int main(int argc,char**argv) {
2  cout<<argc<<endl;
3  if(argc<2){
4
5  }
6  for(int i=1;i<argc;i++){
7    string arg = string(argv[i]);
8    if(arg=="-h"){
9       cout<<"[-h] --help information"<<endl;
10      cout<<"[-t] --test MD5 application"<<endl;
11      cout<<"[-c] [file path of the file computed]"<<endl;
12      cout<<"      --compute MD5 of the given file"<<endl;
13      cout<<"[-v] [file path of the file validated]"<<endl;
14      cout<<"     --validate the integrality of a given file by manual input MD5 value"<<endl;
15      cout<<"[-f] [file path of the file validated] [file path of the .md5 file]"<<endl;
16      cout<<"      --validate the integrality of a given file by read MD5 value from .md5 file"
17   }
18   if(arg=="-t"){
19      cout<<"MD5 result of 'nankai' is ";
20      cout<<getMD5("a")<<endl;
21      cout<<"MD5 result of test.txt which contain 'nankai' is ";
22      cout<<getMD5(readFile("test.txt"))<<endl;
23   }
24   if(arg=="-c"){
25      string path = (argv[++i]);
26      cout<<"The MD5 value of file "<<path<<" is "<<getMD5(readFile(path))<<endl;
```

```
27      }
28      if(arg=="-v"){
29          string path = (argv[++i]);
30          cout<<"Please input the MD5 value of file"<<endl;
31          string old_md5;
32          cin>>old_md5;
33          cout<<"The old MD5 value of file "<<path<<" you have input is "<<old_md5<<endl;
34          cout<<"The old MD5 value of file "<<path<<" that has computed is "<<getMD5(readFile(path))
35          string new_md5= getMD5(readFile(path));
36          if(new_md5==old_md5){
37              cout<<"OK! The file is integrated"<<endl;
38          }else{
39              cout<<"Match Error! The file has been modified!"<<endl;
40          }
41      }
42      if(arg=="-f"){
43          string path = (argv[++i]);
44          string path_ = (argv[++i]);
45          cout<<"The old MD5 value of file "<<path<< " in "<<path_<<" is "<<readMD5(path_)<<endl;
46          cout<<"The new MD5 value of file "<<path<< " that has computed is "<<getMD5(readFile(path
47          if(readMD5(path_)==getMD5(readFile(path))){
48              cout<<"OK! The file is integrated"<<endl;
49          }else{
50              cout<<"Match Error! The file has been modified!"<<endl;
51          }
52      }
53  }
54
```

3.5 实现效果

3.5.1 h 参数实验

通过指定参数 h 确实观察到命令行输出提示信息

```
(sf) wbf@wbfubuntu:~/network_security/md5$ ./md5 -h
2
[-h] --help information
[-t] --test MD5 application
[-c] [file path of the file computed]
     --compute MD5 of the given file
[-v] [file path of the file validated]
     --validate the integrality of a given file by manual input MD5 value
[-f] [file path of the file validated] [file path of the .md5 file]
     --validate the integrality of a given file by read MD5 value from .md5 file
(sf) wbf@wbfubuntu:~/network_security/md5$
```

3.6 t 参数实验

通过指定参数 t 确实观察到命令行输出测试样例的 md5 信息，与实验手册的对比发现程序编写正确。

```
(sf) wbf@wbfubuntu:~/network_security/md5$ ./md5 -t
2
MD5 result of 'nankai' is 0cc175b9c0f1b6a831c399e269772661
MD5 result of test.txt which contain 'nankai' is 387c82f575a3c4ece394d7f168067c13
(sf) wbf@wbfubuntu:~/network_security/md5$
```

3.6.1 c 参数实验

通过指定参数 c 后跟随想要检查的目标文件的相对路径，发现命令行成功输出指定文件的 md5 值

```
MD5 result of test.txt which contain 'nankai' is 387c82f575a3c4ece394d7f168067c13
(sf) wbf@wbfubuntu:~/network_security/md5$ ./md5 -c test.txt
3
The MD5 value of file test.txt is 387c82f575a3c4ece394d7f168067c13
(sf) wbf@wbfubuntu:~/network_security/md5$
```

3.6.2 v 参数实验

通过指定参数 v 后跟随想要检查的目标文件的相对路径，并根据命令行提示输入原来的 md5 值，程序会根据文件重新计算 md5 来判断二者是否相同

```
Match Error! The file has been modified!
(sf) wbf@wbfubuntu:~/network_security/md5$ ./md5 -v test.txt
3
Please input the MD5 value of file
387c82f575a3c4ece394d7f168067c13
The old MD5 value of file test.txt you have input is 387c82f575a3c4ece394d7f168067c13
The old MD5 value of file test.txt that has computed is 387c82f575a3c4ece394d7f168067c13
OK! The file is integrated
(sf) wbf@wbfubuntu:~/network_security/md5$
```

```
(sf) wbf@wbfubuntu:~/network_security/md5$ ./md5 -v test.txt
3
Please input the MD5 value of file
387c82f575a3c4ece394d7f168067c12
The old MD5 value of file test.txt you have input is 387c82f575a3c4ece394d7f168067c12
The old MD5 value of file test.txt that has computed is 387c82f575a3c4ece394d7f168067c13
Match Error! The file has been modified!
(sf) wbf@wbfubuntu:~/network_security/md5$
```

3.6.3 f 参数实验

通过指定参数 f 后跟随想要检查的目标文件的相对路径，并跟随保存有指定文件 md5 值的.md5 后缀的文件进行比对。

```
(sf) wbf@wbfubuntu:~/network_security/md5$ ./md5 -f test.txt test.md5
4
The old MD5 value of file test.txt in test.md5 is 387c82f575a3c4ece394d7f168067c13
The new MD5 value of file test.txt that has computed is 387c82f575a3c4ece394d7f168067c13
OK! The file is integrated
(sf) wbf@wbfubuntu:~/network_security/md5$
```

3.7 后续探索

3.7.1 MD5 算法与 Linux 口令保护

除了验证文件完整性以外，MD5 算法在其它方面还有着广泛的应用。众所周知，Linux 系统的用户口令需要经过加密才能保存在配置文件中，而 MD5 算法则是加密用户口令的最常用算法。最早，Linux 用户口令加密后保存在用户配置文件/etc/passwd 中。文件中的每一行对应一个用户，并用冒号(:) 划分为 7 个字段。用户口令加密后就放在每一行的第二个字段里。

一般情况下，/etc/passwd 文件允许所有用户读取，但只允许 root 用户写入。因此，恶意用户可以轻松地读取加密后的口令字符串，然后使用字典攻击等手段获得其它用户的口令。

鉴于上述原因，后期的 Linux 版本专门设置一个用户影子口令文件/etc/shadow 用于保存用户口令，并且只允许 root 用户读取该文件。这样普通用户就无法读取加密后的口令文件，从而进一步提高了口令的安全性。在/etc/passwd 文件中，每一行的第二个字段表示是否有加密口令。若为 x，表示该账户设置了口令，否则，表示没有设置口令。以下面的/etc/passwd 文件为例，第一行记录表明 root 用户设置了口令。而真正的口令加密字符串则保存在/etc/shadow 文件中。

3.7.2 Linux 系统 GRUB 的 MD5 加密方法

在使用 Linux 过程中，用户经常会忘记自己之前设定的口令。鉴于这种情况，GRUB 为用户提供了一条恢复口令的捷径。GRUB 是一个引导装入器，负责装入内核并引导 Linux 系统。类似于在计算机上安装两个 windows 时出现的选单管理器 OS Loader，GRUB 可以让我们选择使用哪一个操作系统。与 Linux 之前的引导装入器 LILO 相比，GRUB 具有功能强大，引导过程灵活，安全性高等优点。

3.7.3 字典攻击与 MD5 变换算法

因为 MD5 算法的运算过程不可逆，所以对任意一个 MD5 散列逆向计算出明文的破解机是不存在的。目前，破解 MD5 的最有效方法之一就是字典攻击。所谓字典攻击，就是指事先收集大量明文和对应的 MD5 散列，并把它们成对地保存在数据库中，然后在数据库中寻找与当前 MD5 散列对应的明文进行破解。当然如果数据库中没有相应的记录，破解工作也就无法进行。

如果攻击者拥有数据量巨大的密码字典，并且建立了许多 MD5 原文/散列对照数据库，那么就能快速地找到常用明文字符串的 MD5 散列。因此字典攻击已经成为一种破解 MD5 散列的高效途径。然而，上述字典所使用的都是常规的 MD5 算法，即原文–>MD5–> 密文。如果我们对 MD5 算法进行变换，就可以使这些 MD5 字典无所作为。

对 MD5 进行变换的方法很多，最容易理解的就是对同一原文进行多次 MD5 运算。在下面的代码中，自定义了一个函数 md5_more，它有两个参数 data 和 times，前者表示需要加密的明文，后者表

示重复加密的次数。可以看到，在函数体中，调用了 times 次 md5 函数，因此将明文加密了 times 遍。这种变换也可以用递归的方法来实现，此处不再赘述。

```
function md5_more(data, times)
{   //循环使用 MD5
    for (i = 0; i < times; i++) {
    data = md5(data);
    }
    return data;
}
```

另一种变换的主要思想是：首先经过一次 MD5 运算，得到一个由 32 个字符（用 16 进制表示）组成的散列字符串，然后将该字符串分割为若干子串，再对每个子串进行 MD5 运算，最后将每个子串的 MD5 散列合并成一个字符串作为 MD5 算法的输入并计算出一个最终的散列。

除了上述两种变换方法之外，还有附加字符串干涉，字符串次序干涉，大小写变换干涉等多种方法。可以说，MD5 变换算法以增加计算开销为代价，有效地抑制了字典攻击，提高了算法的安全性。但是，一旦攻击者获得了某个应用程序指定的 MD5 变换算法，那么该变换算法就再也没有任何秘密可言，唯一有益的是消耗了攻击者更多的计算资源。

4 实验遇到的问题及其解决方法

4.1 摘要的表示

在一开始我是想使用 char 或者 string 来表示摘要的，但是在网上查阅资料后最终使用了 unsigned int 来表示摘要，具体原因如下：在 MD5 计算中，摘要是由 32 位无符号整数的 4 个字（即 128 位）组成的。这样做的原因是因为：

1. 无符号整数可以保证摘要的数据类型是正整数，避免了符号位的影响。
2. 无符号整数的取值范围是 [0, 232-1]，满足 MD5 算法中取模运算的要求。
3. 无符号整数的运算符合二进制数的规律，方便了 MD5 算法中的位运算和逻辑运算。
4. 无符号整数的内存占用量和速度都比较合适，提高了 MD5 算法的效率。

在实现 MD5 算法时，为了保证计算结果的正确性，摘要必须严格按照 32 位无符号整数来表示和处理。这样可以保证计算过程中避免了数据类型转换带来的精度问题。同时，使用无符号整数还可以使得 MD5 算法的实现更加简洁、高效和稳定。

4.2 字节序问题

在使用 C++ 编写 MD5 程序时，可能会出现与字节序有关的问题。MD5 算法在存储、传输数据时需要按特定的字节序，通常为 Big-Endian。如果程序的字节序不一致，将导致数据读取和处理错误。我通过使用如下办法来解决字节序的问题

1. 利用 C++ 标准库提供的字节序转换函数。例如，htonl() 函数可以实现从主机字节序到网络字节序

的转换，而 ntohl() 函数则可以实现从网络字节序到主机字节序的转换。

2. 使用位运算和移位操作来处理字节序。例如，可以将 16 进制位数右移 8 位，然后在进行二进制或运算，将结果写入摘要的相应字节位置。

3. 让程序根据系统自动决定字节序。这可以通过判断系统的大小端模式并进行相应的处理来实现，适用于不需要考虑程序的移植性的情况。

总之，在编写 C++ 程序时，需要特别注意字节序问题，尽可能采用标准库提供的字节序转换函数或者使用位运算和移位操作，来保证程序的正确性和稳定性。

5 实验结论

本次实验要求通过 C++ 语言准确地实现 MD5 算法的完整计算过程，并能对任意长度的字符串或任意大小的文件，生成 128 位 MD5 摘要，并通过检验 MD5 摘要的正确性来检验原文件的完整性。

在实验过程中，我首先学习了 MD5 算法的原理和实现方法，在此基础上编写了 C++ 程序实现了 MD5 算法的完整计算过程。实现过程中，需要注意处理字节序问题，确保按照大端模式进行计算。

接着，我测试了程序对字符串和文件的 MD5 摘要计算功能，均能够准确生成 128 位 MD5 摘要。同时，由于 MD5 算法具有强抗碰撞性和不可逆性，因此通过检验生成的 MD5 摘要的正确性，能够较为可靠地检验原文件的完整性。

通过本次实验，我不仅掌握了 MD5 算法的原理和实现方法，还了解了 C++ 语言在实现算法中的重要作用。同时，我也深刻认识到数据完整性在计算机领域中的重要性，以及如何通过 MD5 摘要等技术保护数据的完整性和安全性。

在实验中，我还发现了一些问题和不足之处。首先，对于极长的字符串或文件，程序可能会出现内存不足的问题，因此需要进行适当的优化和处理。其次，在实现文件读取和写入功能时，需要考虑数据流的稳定性和可靠性问题，以避免数据丢失或损坏。

总之，本次实验让我深入了解了 MD5 算法的实现过程和作用，在此基础上进一步提高了程序设计和编程能力。同时，实验中也对我日后开展信息安全或数据完整性相关工作具有重要的参考和指导价值。