南開大學
Nankai University

网络空间安全学院

网络安全技术实验报告

基于 MD5 算法的文件完整性校验程序

姓名：杨鑫

学号：2011028

专业：信息安全

2023 年 5 月 14 日

目录

1 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，并完成基于 MD5 算法的文件完整性校验程序。实验环境 运行 Windows 操作系统的 PC 机，具有 VC 等语言编译环境。

2 实验原理

本次实验的流程图如下所示：

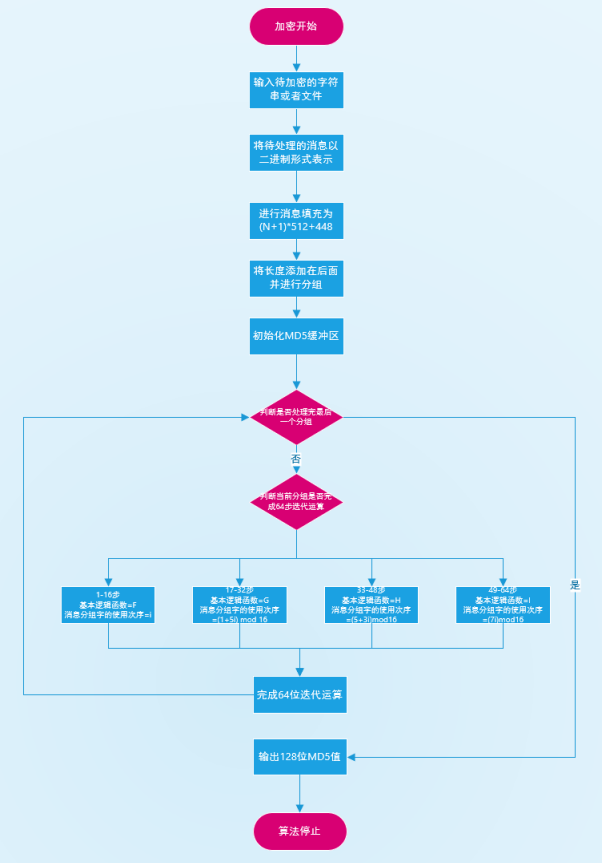

图 2.1: MD5 算法流程

Hash 函数是将任意长的数字串转换成一个较短的定长输出数字串的函数，输出的结果称为 Hash 值。Hash 函数具有快速性、单向性、无碰撞性等特点。Hash 函数可用于数字签名、消息的完整性检验、消息的来源认证检测等。现在常用的 Hash 算法有 MD5、SHA-1 等。下面从 MD5 入手来介绍 Hash 算法的实现机制。MD5 算法的主要步骤如下：**第一步**填充：如果输入信息的长度 (bit) 对 512 求余的结果不等于 448，就需要填充使得对 512 求余的结果等于 448。填充的方法是填充一个 1 和 n 个 0。填充完后，信息的长度就为 N*512+448(bit)；

第二步记录信息长度：用 64 位来存储填充前信息长度。这 64 位加在第一步结果的后面，这样信息长度就变为 N*512+448+64=(N+1)*512 位。

第三步装入标准的幻数(四个整数):标准的幻数(物理顺序)是(A=(01234567) 16 ,B=(89ABCDEF) 16 ,C=(FEDCBA98) 16 ,D=(76543210) 16)。如果在程序中定义应该是(A=0X67452301L,B=0XEFCDAB89L, C=0X98BADCFEL，D=0X10325476L)。

第四步四轮循环运算：循环的次数是分组的个数（N+1），将每一 512 字节细分为 16 个小组，每个小组 64 位 (8 个字节)，然后通过四个线性函数 F、G、H、I 定义的四种操作进行四轮运算，每轮循环后，将 A，B，C，D 分别加上 a，b，c，d，然后进入下一循环。MD5 的安全性：普遍认为 MD5 是很安全，因为暴力破解的时间是一般人无法接受的。实际上如果把用户的密码 MD5 处理后再存储到数据库，其实是很不安全的。因为用户的密码是比较短的，而且很多用户的密码都使用生日，手机号码，身份证号码，电话号码等等。或者使用常用的一些吉利的数字，或者某个英文单词。

3 实验代码

本次 MD5 实验的主要代码如下：**define.h**

```
1
2  #ifndef DEFINE_DEFINE_H
3  #define DEFINE_DEFINE_H
4
5  #include <iostream>
6  #include <string>
7  #include <cstring>
8  #include <stdlib.h>
9  #include <fstream>
10 #include <sstream>
11 using namespace std;
12
13 //基本逻辑函数 使用 宏定义
14 #define F(b,c,d) (( b & c ) | (( ~b ) & ( d )))
15 #define G(b,c,d) (( b & d ) | ( c & ( ~d )))
16 #define H(b,c,d) ( b ^ c ^ d )
17 #define I(b,c,d) ( c ^ ( b | ( ~d )))
18
19 //x 循环左移 n 位 使用 宏定义
20 #define shift(x,n) (( x << n ) | ( x >> ( 32 - n )))
21
22 typedef unsigned int u_int;
23
24 //压缩函数每轮每步中 A 分块循环左移的位数
25 const unsigned s[64] =
26 {
27     7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,
28     5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,
29     4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,
30     6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21
31 };
```

```
32
33  //常数表 T
34  const unsigned T[64] =
35  {
36      0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
37      0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
38      0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
39      0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
40      0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
41      0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
42      0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
43      0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
44      0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
45      0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
46      0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
47      0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
48      0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
49      0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
50      0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
51      0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391
52  };
53
54
55  //整数转十六进制字符串 注意小端序
56  string int2hex(unsigned int Integer)
57  {
58      const string strHex = "0123456789abcdef";
59      unsigned x;
60
61      string temp;
62      string hexString = "";
63
64      for (int i = 0; i < 4; i++)
65      {
66          temp = "";
67          x = (Integer >> (i * 8)) & 0xff;
68
69          for (int j = 0; j < 2; j++)
70          {
71              temp.insert(0, 1, strHex[x % 16]);
72              x /= 16;
73          }
```

```
74          hexString += temp;
75      }
76      return hexString;
77  }
78
79  //MD5 处理函数
80  string md5(string message)
81  {
82      //定义 A、B、C、D 四个链接变量，小端序存储
83      unsigned int A = 0x67452301;
84      unsigned int B = 0xefcdab89;
85      unsigned int C = 0x98badcfe;
86      unsigned int D = 0x10325476;
87
88      //记录字符串的长度（字节 8 位）
89      int len = message.length();
90      //记录需要处理的分组数 以 512 位，64 个字节为一组
91      int num = ((len + 8) / 64) + 1;
92      u_int* messageByte = new u_int[num * 16];
93      memset(messageByte, 0, sizeof(u_int) * num * 16);
94
95      //填充字符串
96      for (int i = 0; i < len; i++) {
97          // 一个 unsigned int 对应 4 个字节，保存 4 个字符信息
98          messageByte[i / 4] |= message[i] << ((i % 4) * 8);
99      }
100     // 补充 1000...000
101     messageByte[len >> 2] |= 0x80 << ((len % 4) * 8);
102     // 填充原文长度
103     messageByte[num * 16 - 2] = (len << 3);
104
105     unsigned int a, b, c, d;
106
107     for (int i = 0; i < num; i++)
108     {
109         a = A;
110         b = B;
111         c = C;
112         d = D;
113         unsigned int g;
114         int k;
115
```

```
116             //经过 4 轮
117             for (int j = 0; j < 64; j++)
118             {
119                 if (j < 16)
120                 {
121                     g = F(b, c, d);
122                     k = j;
123                 }
124                 else if (j >= 16 && j < 32)
125                 {
126                     g = G(b, c, d);
127                     k = (1 + 5 * j) % 16;
128                 }
129                 else if (j >= 32 && j < 48)
130                 {
131                     g = H(b, c, d);
132                     k = (5 + 3 * j) % 16;
133                 }
134                 else if (j >= 48 && j < 64)
135                 {
136                     g = I(b, c, d);
137                     k = (7 * j) % 16;
138                 }
139 
140                 unsigned temp_d = d;
141                 d = c;
142                 c = b;
143                 b = b + shift(a + g + messageByte[i * 16 + k] + T[j], s[j]);
144                 a = temp_d;
145             }
146             A = a + A;
147             B = b + B;
148             C = c + C;
149             D = d + D;
150         }
151         return int2hex(A) + int2hex(B) + int2hex(C) + int2hex(D);
152     }
153 
154     #endif // DEFINE_DEFINE_H
```

main.cpp

```cpp
#include<iostream>
#include<string>
#include"define.h"

using namespace std;

void usage() {
    cerr << "YangMD5 [ -s string ] [ -f file ] [ -c md5 file ] [ -b file1 file2 ]"
        << " -s string" << endl
        << "            the input string " << endl
        << " -f file" << endl
        << "            the input filepath" << endl
        << " -c md5 file" << endl
        << "            input the md5 and file to check the file" << endl
        << " -b file1 file2" << endl
        << "            input the file1 and file2 to check the same" << endl;
}

int main(int argc, char** argv) {
    for (size_t i = 1; i < argc; i++) {
        if (string(argv[i]) == "-s") {
            if (argc > 3) {
                usage();
                exit(1);
            }
            string tmps = string(argv[i + 1]);
            cout<<"String MD5: " << md5(tmps) << endl;
        }
    else if (string(argv[i]) == "-f") {
        if (argc > 3) {
            usage();
            exit(1);
        }
        string filepath = string(argv[i + 1]);
        ifstream ifile(filepath.data());
        ostringstream buf;
        char ch;
        while (buf && ifile.get(ch))
            buf.put(ch);
        string input = buf.str();
        cout << "File MD5: " << md5(input) << endl;
```

```
42      }
43      else if (string(argv[i]) == "-c") {
44          if (argc > 4) {
45              usage();
46              exit(1);
47          }
48          string imd5 = string(argv[i + 1]);
49          string filepath = string(argv[i + 2]);
50          ifstream ifile(filepath.data());
51          ostringstream buf;
52          char ch;
53          while (buf && ifile.get(ch))
54              buf.put(ch);
55          string input = buf.str();
56          string fmd5 = md5(input);
57          string flag = strcmp(fmd5.c_str(), imd5.c_str()) ? "false" : "true";
58          cout << "Input MD5: " << imd5 << endl
59              << "File  MD5: " << fmd5 << endl
60              << flag << endl;
61          }
62      else if (string(argv[i]) == "-b") {
63          if (argc > 4) {
64              usage();
65              exit(1);
66          }
67          string filepath1 = string(argv[i + 1]);
68          string filepath2 = string(argv[i + 2]);
69          ifstream ifile1(filepath1.data());
70          ostringstream buf1;
71          char ch;
72          while (buf1 && ifile1.get(ch))
73              buf1.put(ch);
74          string input1 = buf1.str();
75          string f1md5 = md5(input1);
76          ifstream ifile2(filepath2.data());
77          ostringstream buf2;
78          while (buf2 && ifile2.get(ch))
79              buf2.put(ch);
80          string input2 = buf2.str();
81          string f2md5 = md5(input2);
82          string flag = strcmp(f1md5.c_str(), f2md5.c_str()) ? "false" : "true";
83          cout << "File1 MD5: " << f1md5 << endl
```

```
84                 << "File2 MD5: " << f2md5 << endl
85                 << flag << endl;
86             }
87         else if(string(argv[i]) == "-h"){
88             usage();
89             exit(1);
90             }
91         }
92         return 0;
93     }
```

4 实验结果

加密消息的结果与实际的字符串的 Hash：

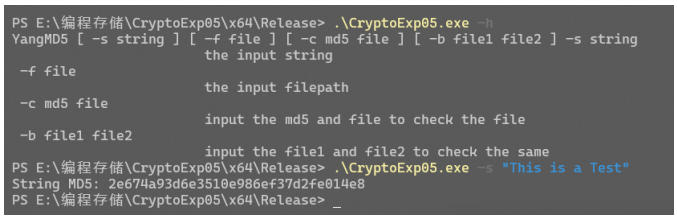

图 4.2: 加密程序加密字符串

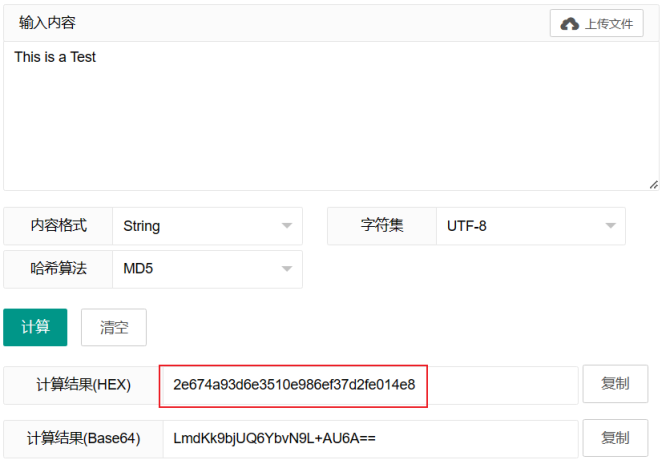

图 4.3: 实际字符串 Hash

可以发现，两者一摸一样证明程序的正确性。

加密文件的结果和文件的 HASH：

图 4.4: 加密得到文件 Hash

可以发现通过提供 Hash 值判断文件并没有被损坏，验证了文件的完整性。

同时这里还提供判断两个文件是否相同的命令，通过使用-b 命令可以判断两个文件的 MD5 值是否相等：

图 4.5: 判断两个文件的 Hash 是否相等

至此实验完成。

5 实验总结

在基于 MD5 算法实现文件完整性校验的实验中，通过计算文件的 MD5 值来验证文件的完整性。MD5 算法是一种常见的哈希函数，可以将任意长度的消息压缩成一个 128 位的哈希值。在文件完整性校验中，将使用 MD5 算法计算文件的哈希值，并将其与原始文件的哈希值进行比较，以确保文件未被篡改或损坏。

实验中，首先需要了解 MD5 算法的原理和计算过程。MD5 算法基于位运算、逻辑运算和模运算等数学原理，通过多轮计算生成 128 位的哈希值。在实现文件完整性校验时，需要使用专门的 MD5 计算工具或编程语言库来计算文件的 MD5 值。

接下来，需要对比计算得到的 MD5 值与原始文件的 MD5 值是否一致。如果一致，则说明文件未被篡改或损坏；如果不一致，则说明文件已经被篡改或损坏。在实际应用中，我们可以将原始文件的 MD5 值存储在一个独立的文件中，以便于后续比较。

总体来说，基于 MD5 算法实现文件完整性校验是一种简单而有效的方法。通过计算文件的 MD5 值，可以快速地验证文件的完整性，从而确保文件的安全性和可靠性。在实际应用中，需要注意保护原始文件和 MD5 值文件的安全性，以避免被恶意篡改或删除。