网络安全技术

实
验
报
告

学　　院　网安学院

年　　级　21 级

班　　级　信息安全一班

学　　号　2113662

姓　　名　张丛

手机号　18086215842

2024 年 5 月 8 日

目录

一、实验目的

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：

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

本章编程训练的要求如下：

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

二、实验内容

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。

三、实验步骤及实验结果

（1）MD5 算法实现

MD5 算法分为 3 个部分：消息的填充与分割、消息块的循环运算、摘要的生成。

MD5 算法是程序的核心部分，通过 MD5 类来实现。

MD5 类定义如下：

```
class MD5
{
private:
    DWORD state[4];             //用于表示 4 个初始向量
    DWORD count[2];            //用于计数，count[0]表示低位，count[1]表示高位
    BYTE buffer_block[64];     //用于保存计算过程中按块划分后剩下的比特流
    BYTE digest[16];          //用于保存 128 比特长度的摘要
    bool is_finished;         //用于标志摘要计算过程是否结束
    static const BYTE padding[64];  //用于保存消息后面填充的数据块
    static const char hex[16];      //用于保存 16 进制的字符
    void Stop();
    void Transform(const BYTE block[64]);
    //将双字流转换为字节流
    void Encode(const DWORD *input, BYTE *output, size_t length);
    //将字节流转换为双字流
    void Decode(const BYTE *input, DWORD *output, size_t length);
    //将字节流按照十六进制字符串形式输出
```

```
18.      std::string BytesToHexString(const BYTE *input, size_t length);
19.
20.  public:
21.      MD5();
22.      MD5(const std::string &str);
23.      MD5(std::ifstream &in);
24.      void Update(std::ifstream &in);
25.      void Update(const BYTE* input, size_t length);
26.      const BYTE* GetDigest();
27.      std::string Tostring();
28.      void Reset();
29.  };
```

主要成员的功能：

`void Update(const BYTE* input, size_t length)`：对给定字符串进行 MD5 运算

`const BYTE* GetDigest()`：获取计算出的 MD5 摘要，返回一个指向摘要字节数组的指针。

`std::string Tostring()`：将 MD5 摘要转换为十六进制字符串形式并返回。

`void Reset()`：重置 MD5 算法的状态。

下面按照 MD5 算法的流程进行具体说明：

消息的填充与分割

在消息的最后填充一位 1 和若干位 0，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。

在实验代码中，通过 Update（）函数可以：

1. 更新 MD5 算法的状态，包括将输入数据填充到缓冲区中，并根据需要调用 MD5 算法的核心变换函数进行处理。
2. 计算输入数据的长度，并将其作为数据的一部分加入到 MD5 算法的状态中，以便最终计算出的摘要中包含输入数据的长度信息。
3. 同时也将输入数据分块处理，每个块的大小为 64 字节（512 比特）。

如下：

```
void MD5::Update(const BYTE* input,size_t length)
{
    DWORD i, index, partLen;
    is_finished = false;
    index = (DWORD)((count[0] >> 3) & 0x3f);
    if((count[0] += ((DWORD)length << 3)) < ((DWORD)length << 3)) {
        count[1]++;
 }
    count[1] += ((DWORD)length >> 29);
    partLen = 64 - index;
    if(length >= partLen){
        memcpy(&buffer_block[index], input, partLen);
        Transform(buffer_block);
        for (i = partLen; i + 63 < length; i += 64) {
            Transform(&input[i]);
 }
        index = 0;
    }
 else {
        i = 0;
    }
    memcpy(&buffer_block[index], &input[i], length-i);
}
```

对于文件流也有相应的处理：

```
void MD5::Update(std::ifstream &in) {
    if (!in)
        return;
    std::streamsize length;
    char buffer[BUFFER_SIZE];
    while (!in.eof()) {
        in.read(buffer, BUFFER_SIZE);
        length = in.gcount();
        if (length > 0)
            Update((const BYTE*)buffer, length);
    }
    in.close();
}
```

消息块的循环运算

MD5 算法包含 4 个初始向量，5 种基本运算，以及 4 个基本函数。

代码如下，其中 S 为循环左移的位数：

```
#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))
#define FF(a, b, c, d, x, s, ac) {(a) += F ((b), (c), (d)) + (x) + ac;(a) = ROTATE_LEFT ((a), (s));(a) += (b);}
#define GG(a, b, c, d, x, s, ac) {(a) += G ((b), (c), (d)) + (x) + ac;(a) = ROTATE_LEFT ((a), (s));(a) += (b);}
#define HH(a, b, c, d, x, s, ac) {(a) += H ((b), (c), (d)) + (x) + ac;(a) = ROTATE_LEFT ((a), (s));(a) += (b);}
#define II(a, b, c, d, x, s, ac) {(a) += I ((b), (c), (d)) + (x) + ac;(a) = ROTATE_LEFT ((a), (s));(a) += (b);}
```

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）。

循环体中包含了 4 轮计算，每一轮计算进行 16 次操作。每一次操作可概括如下：

- 将向量 A、B、C、D 中的其中三个作一次非线性函数运算。
- 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。
- 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。

对应的函数是 Transform(),根据指导书对 4 轮计算的具体说明，代码如下：

```
1.  void MD5::Transform(const BYTE block[64]) {
2.      DWORD a = state[0], b = state[1], c = state[2], d = state[3], x[16];
3.      Decode(block, x, 64);
4.      /* 第1 轮 */
5.      FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
6.   FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
7.   FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
8.   FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
9.   FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
10.  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
11.  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
12.  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
13.  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
14.  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
15.  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
16.  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
17.  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
18.  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
19.  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
20.  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */
```

```
21.
22.     /* 第 2 轮 */
23.     GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
24.     GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
25.     GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
26.     GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
27.     GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
28.     GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
29.     GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
30.     GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
31.     GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
32.     GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
33.     GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
34.     GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
35.     GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
36.     GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
37.     GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
38.     GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */
39.
40.     /* 第 3 轮 */
41.     HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
42.     HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
43.     HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
44.     HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
45.     HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
46.     HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
47.     HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
48.     HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
49.     HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
50.     HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
51.     HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
52.     HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
53.     HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
54.     HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
55.     HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
56.
57.     /* 第 4 轮 */
58.     II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
59.     II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
60.     II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
61.     II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
62.     II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
63.     II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
64.     II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
```

```
    II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
    II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
    II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
    II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
    II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
    II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
    II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
    II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
    II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */
        state[0] += a;
        state[1] += b;
        state[2] += c;
        state[3] += d;
}
```

摘要的生成

如下：

```
void MD5::Stop() {
    // 保存当前状态和计数
    BYTE bits[8];
    DWORD oldState[4];
    DWORD oldCount[2];
    memcpy(oldState, state, 16);
    memcpy(oldCount, count, 8);

    // 编码总位数到一个8字节数据块中
    Encode(count, bits, 8);
    // 计算填充长度
    DWORD index = (DWORD)((count[0] >> 3) & 0x3f);
    DWORD padLen = (index < 56) ? (56 - index) : (120 - index);
    // 填充数据
    Update(padding, padLen);
    // 填充总位数
    Update(bits, 8);
    // 计算最终的MD5摘要
    Encode(state, digest, 16);

    // 恢复状态和计数
    memcpy(state, oldState, 16);
```

```
        memcpy(count, oldCount, 8);
    }
```

（2）文件完整性检验

在 main 函数中，程序通过区分参数 argv[1]的不同值来启动不同的工作流程。

如果 argv[1]等于“-h”，表示显示帮助信息：

```
    void print_h(int argc, char *argv[]) {
        if (2 != argc) {
            std::cout << "参数错误." << std::endl;
            return;
        }
        std::cout << "MD5: usage:\n" << "\t" << "[-h] --help information " << std::endl;
        std::cout << "\t" << "[-t] --test MD5 application" << std::endl;
        std::cout << "\t" << "[-c] [file path of the file computed]" << std::endl;
        std::cout << "\t" << "\t" <<"--compute MD5 of the given file" << std::endl;
        std::cout << "\t" << "[-v] [file path of the file validated]" << std::endl;
        std::cout << "\t" << "\t" <<"--validate the integrality of a given file by manual input MD5 value" << std::endl;
        std::cout << "\t" << "[-f] [file path of the file validated] [file path of the .md5 file]" << std::endl;
        std::cout << "\t" << "\t" <<"--validate the integrality of a given file by read MD5 value from .md5 file" << std::endl;
    }
```

如果 argv[1]等于“-t”，表示显示测试信息：

```
    void print_t(int argc, char *argv[]) {
        if (2 != argc) {
            std::cout << "参数错误." << std::endl;
            return;
        }
        std::string test[] = {"", "a", "abc", "message digest", "abcdefghijklmnopqrstuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789", "12345678901234567890123456789012345678901234567890123456789012345678901234567890"};
        MD5 md5;
        for (int i = 0; i < 7; ++i) {
```

```
            md5.Update((const BYTE*)test[i].c_str(), test[i].length());
            std::cout << "MD5(\"" + test[i] + "\") = " << md5.Tostring()<< std::endl;
        }
    }
```

如果 argv[1]等于“-c”，表示计算被测文件的 MD5 摘要；

通过调用 MD5 类的 Update()函数计算摘要：

```
    void print_c(int argc, char *argv[]) {
        if (3 != argc) {
            std::cout << "参数错误." << std::endl;
            return;
        }
        std::string filePath = argv[2];
        std::ifstream fileStream(filePath);
        MD5 md5;
        md5.Update(fileStream);
        std::cout << "The MD5 value of file(\"" << filePath << "\") is " << md5.Tostring()
    << std::endl;
    }
```

如果 argv[1]等于“-v”，表示根据手工输入的 MD5 摘要验证文件完整性；

对 MD5 类生成的摘要和手动输入的摘要进行对比，输出结果：

```
    void print_v(int argc,char *argv[]) {
        if (3 != argc) {
            std::cout << "参数错误." << std::endl;
            return;
        }
        std::string filePath = argv[2];
        std::cout << "Please input the MD5 value of file(\"" << filePath << "\")..." << st
    d::endl;
        std::string inputMD5;
        std::cin >> inputMD5;
        std::cout << "The old MD5 value of file(\"" << filePath << "\") you have input is"
    << std::endl << inputMD5 << std::endl;
        std::ifstream fileStream(filePath);
        MD5 md5;
        md5.Update(fileStream);
        std::string genMD5 = md5.Tostring();
```

```
15.      std::cout << "The new MD5 value of file(\"" << filePath << "\") that has computed is" << std::endl << genMD5 << std::endl;
16.      if (!genMD5.compare(inputMD5)) {
17.          std::cout << "OK! The file is integrated" << std::endl;
18.      }
19.      else {
20.          std::cout << "Match Error! The file has been modified!" << std::endl;
21.      }
22.  }
```

如果 argv[1]等于“-f”，表示根据.md5 文件中的摘要验证文件完整性。

对 MD5 类生成的摘要和 linux 程序生成的摘要进行比较，输出结果：

```
1.   void print_f(int argc, char *argv[]) {
2.      if (4 != argc) {
3.          std::cout << "参数错误." << std::endl;
4.          return;
5.      }
6.      std::string filePath = argv[2];
7.      std::string md5Path = argv[3];
8.      std::ifstream md5Stream(md5Path);
9.      std::string oldMD5Str((std::istreambuf_iterator<char>(md5Stream)), std::istreambuf_iterator<char>());
10.     oldMD5Str = (std::string)strtok(const_cast<char*>(oldMD5Str.c_str())," ");
11.     std::cout << "The old MD5 value of file(\"" << filePath << "\") in " << md5Path << " is " << std::endl << oldMD5Str << std::endl;
12.     std::ifstream fileStream(filePath);
13.     MD5 md5;
14.     md5.Update(fileStream);
15.     std::string genMD5 = md5.Tostring();
16.     std::cout << "The new MD5 value of file(\"" << filePath << "\") that has computed is" << std::endl << genMD5 << std::endl;
17.     if (!genMD5.compare(oldMD5Str)) {
18.         std::cout << "OK! The file is integrated" << std::endl;
19.     }
20.     else {
21.         std::cout << "Match Error! The file has been modified!" << std::endl;
22.     }
23.
```

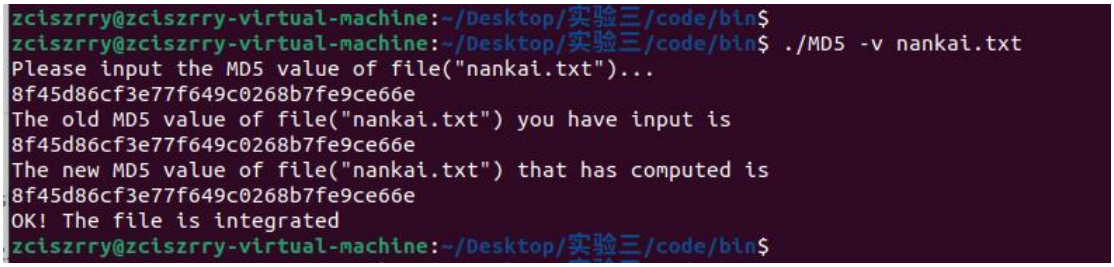
（3）实验结果

帮助信息：

测试信息：

生成 MD5 摘要：

手动输入验证摘要：



```
zciszrry@zciszrry-virtual-machine:~/Desktop/实验三/code/bin$
zciszrry@zciszrry-virtual-machine:~/Desktop/实验三/code/bin$ ./MD5 -v nankai.txt
Please input the MD5 value of file("nankai.txt")...
1433223
The old MD5 value of file("nankai.txt") you have input is
1433223
The new MD5 value of file("nankai.txt") that has computed is
8f45d86cf3e77f649c0268b7fe9ce66e
Match Error! The file has been modified!
zciszrry@zciszrry-virtual-machine:~/Desktop/实验三/code/bin$
```

程序生成验证：

```
zciszrry@zciszrry-virtual-machine:~/Desktop/实验三/code/bin$ ./MD5 -f nankai.txt nankai.md5
The old MD5 value of file("nankai.txt") in nankai.md5 is
8f45d86cf3e77f649c0268b7fe9ce66e
The new MD5 value of file("nankai.txt") that has computed is
8f45d86cf3e77f649c0268b7fe9ce66e
OK! The file is integrated
zciszrry@zciszrry-virtual-machine:~/Desktop/实验三/code/bin$
```

四、实验遇到的问题及其解决方法

MD5 算法的核心变换函数的流程较为复杂，需要仔细阅读分析进行理解。

linux 中对文件和文件路径的操作需要进行学习。

五、实验结论

学习了 MD5 算法的实现流程，亲自进行了 MD5 算法的编程。

了解了 MD5 的历史由来和现实的广泛应用，在 Linux 口令保护和 MD5 加密方法等方面也学习到了更多的知识。