网络安全技术

实
验
报
告

学　　院 网络空间安全学院

年　　级 2021

学　　号 2112060

姓　　名 孙蕗

2024 年 5 月 7 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：

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。

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。

二、实验原理

1. MD5

MD5 是一种信息摘要算法。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 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。

（1） 消息的填充与分割

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。

（2） 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

A) 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。

B) 基本运算

5 种基本运算是指：

a) $\overline{X}$：X 的逐比特逻辑非运算

b) $X \wedge Y$：X，Y 的逐比特逻辑与运算

c) $X \vee Y$：X，Y 的逐比特逻辑或运算

d) $X \oplus Y$：X，Y 的逐比特逻辑异或运算

e) $X <<< s$：X 循环左移 s 位

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

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）。

如下式所示，在一次循环开始时，首先要将初始向量 A、B、C、D 中的值保存到向量 A_0、B_0、C_0、D_0 中，然后再继续后面的操作。

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

MD5 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：

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：

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

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：

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

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：

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

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：

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

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。

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

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） 摘要的生成

如下式所示，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。

$$MD5(M)=A\ B\ C\ D$$

三、 实验步骤及实验结果

1. MD5 的设计与实现

程序可以分为两个主要部分：MD5 算法实现与文件完整性检验。其中前者为程序的核心部分，通过 MD5 类来实现。MD5 类可以为任意长度的消息生成 128 比特的 MD5 摘要。文件完整性检验包括读取被测文件，调用 MD5 类运算函数生成摘要，通过比较摘要判断文件完整性等工作。

（1） MD5 类的设计与实现

作为程序的核心部分，MD5 类负责摘要计算的全部过程，并对外提供各种接口。数组 state 表示 4 个初始向量。数组 count 是一个计数器，记录已经运算的比特数。buffer_block 是一个 64 字节的缓存块，保存消息被划分后不足 64 字节的数据。digest 用于保存生成的 MD5 摘要。Is_finished 标志 MD5 运算是否结束。

Update 函数将不同类型的输入划分为若干个 64 字节的分组，然后调用 Transform 函数进行 MD5 运算。Transform 函数对一个 512 比特消息分组进行 MD5 运算。Encode 函数和 Decode 函数实现消息分组在字节类型与双字类型之间的互相转换。Reset 函数用于重置初始向量。在对新消息进行 MD5 运算之前，需要把初始向量设为默认值。GetDigest 函数用于获得 MD5 摘要。BytesToHexString 函数将 MD5 摘要转换为 16 进制字符串形式。Tostring 函数将 MD5 摘要以 16 进制字符串形式输出。

```
class MD5
{
public:
    uint32_t state[4]; //用于表示 4 个初始向量
    uint32_t count[2]; //用于计数，count[0]表示低位，count[1]表示高位
    uint8_t buffer_block[64]; //用于保存计算过程中按块划分后剩下的比特流
    uint8_t 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 uint8_t 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
    MD5() { Reset(); }
    MD5(const string& str) { Reset(); }
    MD5(ifstream& in) { Reset(); }
    //对给定长度的字节流进行 MD5 运算
    void Update(const uint8_t* input, size_t length);
    //对给定长度的输入流进行 MD5 运算
    void Update(const void* input, size_t length);
    //对给定长度的字符串进行 MD5 运算
    void Update(const string& str);
    void Update(ifstream& in); //对文件中的内容进行 MD5 运算
    const uint8_t* GetDigest(); //将 MD5 摘要以字节流的形式输出
    string Tostring(); //将 MD5 摘要以字符串形式输出
    void Reset(); //重置初始变量
    void Stop(); //用于终止摘要计算过程，输出摘要
    void Transform(const uint8_t block[64]); //对消息分组进行 MD5 运算
    //将双字流转换为字节流
    void Encode(const uint32_t* input, uint8_t* output, size_t length);
    //将字节流转换为双字流
    void Decode(const uint8_t* input, uint32_t* output, size_t length);
    //将字节流按照十六进制字符串形式输出
    string BytesToHexString(const uint8_t* input, size_t length);
};
```

（2） Update 函数

Update 函数将不同类型的输入划分为若干个 64 字节的分组，然后调用 Transform 函数进行 MD5 运算。

MD5 类中有 4 个 Update 重载函数， 4 个公有函数是对外接口，在函数体中都调用了函数 Update 开启 MD5 摘要计算过程。为了方便使用，3 个 Update 公有函数分别为字节流，字符串以及文件流提供了输入接口。虽然输入参数的类型不同，但是在这些接口函数中都会先将输入转化为标准字节流，再调用函数 Update。

由于 MD5 算法是以 64 字节（即 512 比特）为单位进行计算的，函数 Update 首先需要对长为 length 的字节流进行预处理，然后再调用 transform 函数对每一个 64 字节数据块进行计算。预处理并不是将字节流以 64 字节为单位简单地划分成若干数据块，而是需要考虑前一次运算后缓存中是否保存着未被计算的字节。如果有，新的字节必须接在这些字节的后面进行填充，直到填满一个 64 字节数据块后才可以按上述方法继续划分下去；否则，直接以 64 字节为单位进行划分。当划分到最后剩余的字节数不足 64 字节时，将剩余的字节保存在缓存中，等待下一个字节流将数据块填满 64 字节后一起计算。函数 Update 的代码如下。

Update 函数的流程可以分为下面 5 个步骤。

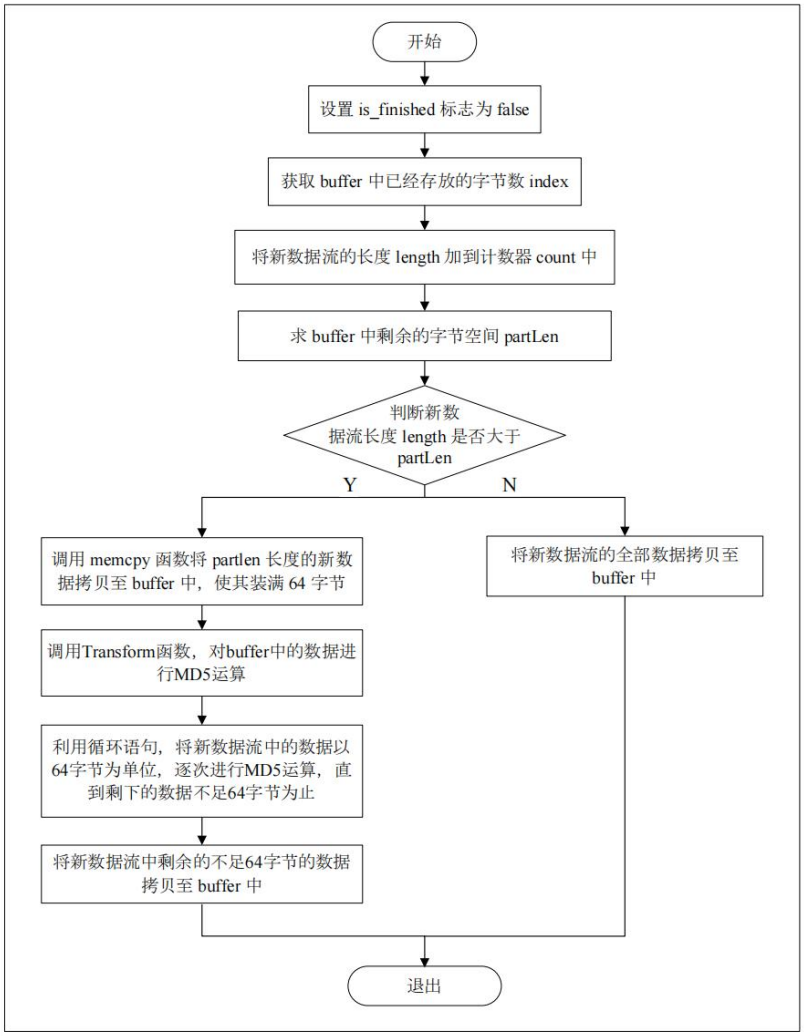
a) 将标志 is_finished 设为 false，表示 MD5 运算正在进行。

b) 将计数器 count 右移 3 位后再截取后 6 位，获得 buffer 中已经存放的字节数。

c) 更新计数器 count，将新数据流的长度加入计数器中。需要注意的是计数器 count 中保存的是比特数（等于字节数×8）。count[0]保存的是数值的低 32 位，count[1]保存的是高 32 位。

d) 求出 buffer 中的剩余字节数 partLen。

e) 判断新数据流的长度 length 是否大于 partLen，如果 length 大于 partLen，将 parLen 长度的新数据拷贝至 buffer 中，使其填满 64 字节，然后调用 Transform 函数对 buffer 中的数据块进行 MD5 运算。接着利用循环将新数据流中的数据以 64 字节为单位，逐次进行 MD5 运算，直到剩余数据不足 64 字节为止，最后将新数据流中不足 64 字节的数据拷贝至 buffer 中。如果 length 不大于 partLen，将新数据流的全部数据拷贝至 buffer 中即可。



```cpp
void MD5::Update(const uint8_t* input, size_t length)
{
    uint32_t i, index, partLen;
    //设置停止标识
    is_finished = false;
    //计算 buffer 已经存放的字节数
    index = (uint32_t)((count[0] >> 3) & 0x3f);
    //更新计数器 count，将新数据流的长度加上计数器原有的值
    if ((count[0] += ((uint32_t)length << 3)) < ((uint32_t)length << 3)) //判断是否进位
    {
        count[1]++;
    }
    count[1] += ((uint32_t)length >> 29);
    //求出 buffer 中剩余的长度
    partLen = 64 - index;
    //将数据块逐块进行 MD5 运算
    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;
    }
    //将不足 64 字节的数据复制到 buffer_block 中
    memcpy(&buffer_block[index], &input[i], length - i);
}
```

```cpp
//对给定长度的输入流进行 MD5 运算
void MD5::Update(const void* input, size_t length)
{
    // Reset();
    const uint8_t* input_bytes = static_cast<const uint8_t*>(input);
    Update(input_bytes, length);
}

//对给定长度的字符串进行 MD5 运算
void MD5::Update(const string& str)
{
    // Reset();
    Update(reinterpret_cast<const uint8_t*>(str.c_str()), str.length());
}

//对文件中的内容进行 MD5 运算
void MD5::Update(ifstream& in)
{
    // Reset();
    char buffer[1024]; // 缓冲区大小为 64 字节
    while (!in.eof()) // 循环读取文件直到文件末尾
    {
        in.read(buffer, sizeof(buffer)); // 从文件中读取数据到缓冲区
        size_t bytes_read = in.gcount(); // 获取实际读取的字节数
        Update(reinterpret_cast<const uint8_t*>(buffer), bytes_read); // 将缓冲区中的数据更新到 MD5 算法中
    }
}
```

（3） Transformer 函数

MD5 类中，Transform 函数负责对 64 字节数据块的进行 MD5 运算。在声明 Transform 函数之前，需要定义一些基本操作和基本运算。

A. Transform 基本操作与基本运算

a. 定义 MD5 算法的 4 个基本函数和 32 位双字循环左移操作

```
#define F(x, y, z) (((x) & (y)) | ((~x) & (z))) //F 函数
#define G(x, y, z) (((x) & (z)) | ((y) & (~z))) //G 函数
#define H(x, y, z) ((x) ^ (y) ^ (z)) //H 函数
#define I(x, y, z) ((y) ^ ((x) | (~z))) //I 函数
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))//循环左移
```

b. 定义四轮运算中的 FF、GG、HH、II 函数。

如下所示，其中 a、b、c、d 表示计算向量，x 表示一个 32 位的子块，s 表示循环左移的位数，ac 表示弧度

```
// 定义四轮运算中的FF、GG、HH、II函数
// a、b、c、d表示计算向量，x表示一个32位的子块，s表示循环左移的位数，ac表示弧度
#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); }
```

c. 定义 MD5 四轮迭代计算中向量 A、B、C、D 循环左移的位数

```
//压缩函数每轮每步中A分块循环左移的位数
const unsigned S[64] =
{
    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
};
```

d. 4294967296×abs(sin(i))的整数部分常数表

```
//常数表T
const unsigned T[64] =
{
    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
};
```

B. Transform 的实现

根据上面定义的公式，Transform 函数构造了 MD5 摘要的计算过程。Transform 函数的定义如下所示。它的执行流程分为以下 4 步。

a) 首先将初始向量 state 的数值赋给变量 a、b、c、d 中。

b) 调用 Decode 函数，将 64 字节的数据块划分为 16 个 32 比特大小的子分组。因为每一轮计算都是对 32 比特子分组进行操作，所以重新划分后可以方便后面的计算过程。

```
void MD5::Transform(const uint8_t block[64])
{
    uint32_t a = state[0], b = state[1], c = state[2], d = state[3], x[16];
    Decode(block, x, 64);
```

c) 依次调用函数 FF、GG、HH、II 展开 4 轮计算，其中每一轮计算包含 16 小步，每一步对一个 32 比特子分组进行运算。函数 FF、GG、HH、II 的前 4 个参数是变量 a、b、c、d 的不同排列，参数 X[k]表示对第 k 个子分组进行计算，S[i]表示某轮某步计算循环左移的位数，最后一个常数 T[i]表示 4294967296∫abs(sin(i))的整数部分。

```
// 第一轮
FF(a, b, c, d, x[0], S[0], T[0]);
FF(d, a, b, c, x[1], S[1], T[1]);
FF(c, d, a, b, x[2], S[2], T[2]);
FF(b, c, d, a, x[3], S[3], T[3]);
FF(a, b, c, d, x[4], S[4], T[4]);
FF(d, a, b, c, x[5], S[5], T[5]);
FF(c, d, a, b, x[6], S[6], T[6]);
FF(b, c, d, a, x[7], S[7], T[7]);
FF(a, b, c, d, x[8], S[8], T[8]);
FF(d, a, b, c, x[9], S[9], T[9]);
FF(c, d, a, b, x[10], S[10], T[10]);
FF(b, c, d, a, x[11], S[11], T[11]);
FF(a, b, c, d, x[12], S[12], T[12]);
FF(d, a, b, c, x[13], S[13], T[13]);
FF(c, d, a, b, x[14], S[14], T[14]);
FF(b, c, d, a, x[15], S[15], T[15]);

// 第2轮
GG(a, b, c, d, x[1], S[16], T[16]);
GG(d, a, b, c, x[6], S[17], T[17]);
GG(c, d, a, b, x[11], S[18], T[18]);
GG(b, c, d, a, x[0], S[19], T[19]);
GG(a, b, c, d, x[5], S[20], T[20]);
GG(d, a, b, c, x[10], S[21], T[21]);
GG(c, d, a, b, x[15], S[22], T[22]);
GG(b, c, d, a, x[4], S[23], T[23]);
GG(a, b, c, d, x[9], S[24], T[24]);
GG(d, a, b, c, x[14], S[25], T[25]);
GG(c, d, a, b, x[3], S[26], T[26]);
GG(b, c, d, a, x[8], S[27], T[27]);
GG(a, b, c, d, x[13], S[28], T[28]);
GG(d, a, b, c, x[2], S[29], T[29]);
GG(c, d, a, b, x[7], S[30], T[30]);
GG(b, c, d, a, x[12], S[31], T[31]);
```

```
// 第3轮
HH(a, b, c, d, x[5], S[32], T[32]);
HH(d, a, b, c, x[8], S[33], T[33]);
HH(c, d, a, b, x[11], S[34], T[34]);
HH(b, c, d, a, x[14], S[35], T[35]);
HH(a, b, c, d, x[1], S[36], T[36]);
HH(d, a, b, c, x[4], S[37], T[37]);
HH(c, d, a, b, x[7], S[38], T[38]);
HH(b, c, d, a, x[10], S[39], T[39]);
HH(a, b, c, d, x[13], S[40], T[40]);
HH(d, a, b, c, x[0], S[41], T[41]);
HH(c, d, a, b, x[3], S[42], T[42]);
HH(b, c, d, a, x[6], S[43], T[43]);
HH(a, b, c, d, x[9], S[44], T[44]);
HH(d, a, b, c, x[12], S[45], T[45]);
HH(c, d, a, b, x[15], S[46], T[46]);
HH(b, c, d, a, x[2], S[47], T[47]);

// 第4轮
II(a, b, c, d, x[0], S[48], T[48]);
II(d, a, b, c, x[7], S[49], T[49]);
II(c, d, a, b, x[14], S[50], T[50]);
II(b, c, d, a, x[5], S[51], T[51]);
II(a, b, c, d, x[12], S[52], T[52]);
II(d, a, b, c, x[3], S[53], T[53]);
II(c, d, a, b, x[10], S[54], T[54]);
II(b, c, d, a, x[1], S[55], T[55]);
II(a, b, c, d, x[8], S[56], T[56]);
II(d, a, b, c, x[15], S[57], T[57]);
II(c, d, a, b, x[6], S[58], T[58]);
II(b, c, d, a, x[13], S[59], T[59]);
II(a, b, c, d, x[4], S[60], T[60]);
II(d, a, b, c, x[11], S[61], T[61]);
II(c, d, a, b, x[2], S[62], T[62]);
II(b, c, d, a, x[9], S[63], T[63]);
```

d) 最后将变量 a、b、c、d 中的运算结果加到初始向量 state 上。

```
state[0] += a;
state[1] += b;
state[2] += c;
state[3] += d;

memset(x, 0, sizeof(x));
```

（4） Encode 函数与 Decode 函数

Encode 函数和 Decode 函数实现消息分组在字节类型与双字类型之间的互相转换。

```cpp
//将双字流转换为字节流
void MD5::Encode(const uint32_t* input, uint8_t* output, size_t length)
{
    for (size_t i = 0, j = 0; j < length; ++i, j += 4)
    {
        output[j] = input[i] & 0xff;
        output[j + 1] = (input[i] >> 8) & 0xff;
        output[j + 2] = (input[i] >> 16) & 0xff;
        output[j + 3] = (input[i] >> 24) & 0xff;
    }
}

//将字节流转换为双字流
void MD5::Decode(const uint8_t* input, uint32_t* output, size_t length)
{
    for (size_t i = 0, j = 0; j < length; ++i, j += 4)
    {
        output[i] = (uint32_t)input[j] |
            ((uint32_t)input[j + 1] << 8) |
            ((uint32_t)input[j + 2] << 16) |
            ((uint32_t)input[j + 3] << 24);
    }
}
```

（5） Reset 函数

Reset 函数用于重置初始向量。在对新消息进行 MD5 运算之前，需要把初始向量设为默认值。

```cpp
void MD5::Reset()
{
    count[0] = count[1] = 0;
    state[0] = 0x67452301;
    state[1] = 0xefcdab89;
    state[2] = 0x98badcfe;
    state[3] = 0x10325476;
    memset(buffer_block, 0, 64);
    memset(digest, 0, 16);
    is_finished = false;
}
```

（6） GetDigest 函数

GetDigest 函数用于获得 MD5 摘要。

```cpp
const uint8_t* MD5::GetDigest()
{
    if (!is_finished)
    {
        uint8_t bits[8];
        Encode(count, bits, 8);
        size_t index = count[0] / 8 % 64;
        size_t padLen = (index < 56) ? (56 - index) : (120 - index);
        Update(padding, padLen);
        Update(bits, 8);
        Encode(state, digest, 16);
        is_finished = true;
    }
    return digest;
}
```

（7） ByteToHexString 函数

BytesToHexString 函数将 MD5 摘要转换为 16 进制字符串形式。

```
//将字节流按照十六进制字符串形式输出
string MD5::BytesToHexString(const uint8_t* input, size_t length)
{
    stringstream ss;
    ss << std::hex << std::setfill('0');
    for (size_t i = 0; i < length; ++i)
    {
        ss << std::setw(2) << static_cast<unsigned>(input[i]);
    }
    return ss.str();
}
```

（8） Tostring 函数

Tostring 函数将 MD5 摘要以 16 进制字符串形式输出。

```
string MD5::Tostring()
{
    const uint8_t* digest = GetDigest();
    return BytesToHexString(digest, 16);
}
```

2. 程序输入格式及文件完整性检验执行过程

程序为命令行程序，可执行文件名为 MD5.exe，命令行格式如下：

./MD5 [选项] [被测文件路径] [.md5 文件路径]

其中[选项]是程序为用户提供的各种功能。在本程序中[选项]包括{-h,-t,-c,-v,-f}5 个基本功能。[被测文件路径]为应用程序指明被测文件在文件系统中的路径。[.md5 文件路径]为应用程序指明由被测文件生成的.md5 文件在文件系统中的路径。其中前两项为必选项，后两项可以根据功能进行选择。

（1） 打印帮助信息

在控制台命令行中输入 ./MD5 -h，打印程序的帮助信息。如下所示，帮助信息详细地说明了程序的选项和执行参数。用户可以通过查询帮助信息充分了解程序的功能

```
int main(int argc, char* argv[])
{ //argc=num of arguments, argv[] store arguments.
    unordered_map<string, void(*)(int, char* [])>
        opcode_table = { {"-t", print_test_info},
                         {"-h", print_help_info},
                         {"-c", print_calc_md5},
                         {"-v", print_input_md5_recalc},
                         {"-f", print_read_md5_recalc} };
    if (argc < 2)
    {
        cout << "error argument, argc = " << argc << endl;
        return -1;
    }
    string op = argv[1];
    if (opcode_table.find(op) != opcode_table.end())
    {
        opcode_table[op](argc, argv);
    }
    return 0;
}
```

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

```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ g++ *.cpp -o md5
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ ./md5 -h
usage:  [-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
```

（2） 打印测试信息

在控制台命令行中输入 ./MD5 －t，打印程序的测试信息。如下所示，测试信息是指本程序对特定字符串输入所生成的 MD5 摘要。所谓特定的字符串是指在 MD5 算法官方文档（RFC1321）中给出的字符串。同时，该文档也给出了这些特定字符串的 MD5 摘要。因此只需要将本程序的计算结果与文档中的正确摘要进行比较，就可以验证程序的正确性。

```cpp
void print_test_info(int argc, char* argv[])
{
    if (2 != argc)
    {
        cout << "error arguments" << endl;
    }
    vector<string> str = { "", "a", "abc", "message digest", "abcdefghijklmnopqrstuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789", "12345678901234567890123456
    MD5 md5;
    for (int i = 0; i < str.size(); ++i)
    {
        MD5 md5;
        // 更新 MD5 计算器
        md5.Update(str[i]);
        // 获取 MD5 摘要
        const uint8_t* digest = md5.GetDigest();
        // 将 MD5 摘要转换为十六进制字符串
        string md5_hex_string = md5.BytesToHexString(digest, 16);
        cout << "MD5(\"" + str[i] + "\") = " << md5_hex_string << endl;
    }
}
```

```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ ./md5 -t
MD5("") = d41d8cd98f00b204e9800998ecf8427e
MD5("a") = 0cc175b9c0f1b6a831c399e269772661
MD5("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") = d174ab98d277d9f5a5611c2c9f419d9f
MD5("12345678901234567890123456789012345678901234567890123456789012345678901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

（3） 为指定文件生成 MD5 摘要

在控制台命令行中输入 ./MD5 －c［被测文件路径］，计算出的被测文件的 MD5 摘要并打印出来。

```cpp
void print_calc_md5(int argc, char* argv[])
{
    if (argc != 3)
        cout << "error arguments" << endl;
    string file_path = argv[2];
    ifstream file_stream(file_path);
    MD5 md5;
    // 更新 MD5 计算器
    md5.Update(file_stream);
    // 获取 MD5 摘要
    const uint8_t* digest = md5.GetDigest();
    // 将 MD5 摘要转换为十六进制字符串
    string md5_hex_string = md5.BytesToHexString(digest, 16);
    cout << "The MD5 value of file(\"" << file_path << "\") is " << md5_hex_string << endl;
}
```

如下所示，被测文件 nankai.txt 与可执行文件 MD5 处于同一个目录中。

可以看到这里 nankai.txt 文件的摘要值为 e45e3c3c8ed674eefdbc41399fd7fec9。


```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ ./md5 -c nankai.txt
The MD5 value of file("nankai.txt") is e45e3c3c8ed674eefdbc41399fd7fec9
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$
```

（4） 验证文件完整性方法一

在控制台命令行中输入 ./MD5 -v [被测文件路径]，程序会先让用户输入被测文件的 MD5 摘要，然后重新计算被测文件的 MD5 摘要，最后将两个摘要逐位比较。若一致，则说明文件是完整的，否则，说明文件遭到破坏。

```cpp
void print_input_md5_recalc(int argc, char* argv[])
{
    if (3 != argc)
    {
        cout << "error arguments" << endl;
    }
    string file_path = argv[2];
    cout << "Please input the MD5 value of file(\"" << file_path << "\")..." << endl;
    string input_md5;
    cin >> input_md5;

    cout << "The old MD5 value of file(\"" << file_path << "\") you have input is" << endl << input_md5 << endl;
    ifstream file_stream(file_path);
    MD5 md5;
    md5.Update(file_stream);
    // 获取 MD5 摘要
    const uint8_t* digest = md5.GetDigest();
    // 将 MD5 摘要转换为十六进制字符串
    string res_md5 = md5.BytesToHexString(digest, 16);
    cout << "The new MD5 value of file(\"" << file_path << "\") that has computed is" << endl << res_md5 << endl;

    if (!res_md5.compare(input_md5))
    {
        cout << "OK! The file is integrated" << endl;
    }
    else
    {
        cout << "Match Error! The file has been modified!" << endl;
    }
}
```

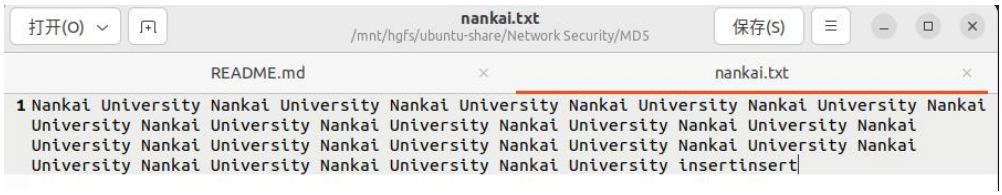
整个执行过程如下所示：

对文件进行完整性校验，未对 nankai.txt 做修改。可以看到两个摘要是一

致的，说明文件是完整的

```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ ./md5 -v nankai.txt
Please input the MD5 value of file("nankai.txt")...
e45e3c3c8ed674eefdbc41399fd7fec9
The old MD5 value of file("nankai.txt") you have input is
e45e3c3c8ed674eefdbc41399fd7fec9
The new MD5 value of file("nankai.txt") that has computed is
e45e3c3c8ed674eefdbc41399fd7fec9
OK! The file is integrated
```

下面对 nankai.txt 文件进行修改，如下面在原始 nankai.txt 文件中添加内容。可以看到两个摘要是不一致的，文件遭到破坏。




```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ ./md5 -v nankai.txt
Please input the MD5 value of file("nankai.txt")...
e45e3c3c8ed674eefdbc41399fd7fec9
The old MD5 value of file("nankai.txt") you have input is
e45e3c3c8ed674eefdbc41399fd7fec9
The new MD5 value of file("nankai.txt") that has computed is
95878813146c7d807ca4cfbae209969c
Match Error! The file has been modified!
```

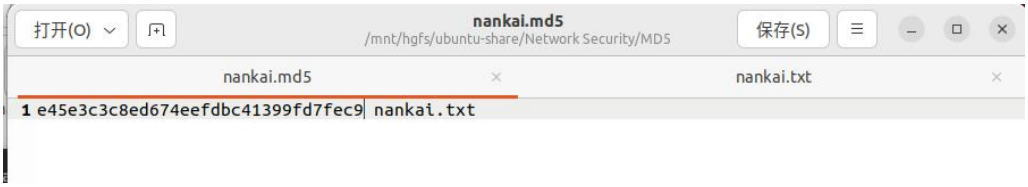
（5） 验证文件完整性方法二

在控制台命令行输入 ./MD5 -f [被测文件路径] [.md5 文件路径]，程序会自动读取.md5 文件中的摘要，然后重新计算出被测文件的 MD5 摘要，最后将两者逐位比较。若一致，则说明文件是完整的，否则，说明文件遭到破坏。

```
void print_read_md5_recalc(int argc, char* argv[])
{
    if (4 != argc)
    {
        cout << "error arguments" << endl;
    }
    string file_path = argv[2];
    string md5_path = argv[3];

    ifstream md5_stream(md5_path);
    string old_md5_digest;
    md5_stream >> old_md5_digest;
    cout << "The old MD5 value of file(\"" << file_path << "\") in " << md5_path << " is " << endl << old_md5_digest << endl;

    ifstream file_stream(file_path);
    MD5 md5;
    md5.Update(file_stream);
    const uint8_t* digest = md5.GetDigest();
    string res_md5 = md5.BytesToHexString(digest, 16);

    cout << "The new MD5 value of file(\"" << file_path << "\") that has computed is" << endl << res_md5 << endl;
    if (!res_md5.compare(old_md5_digest))
    {
        cout << "OK! The file is integrated!" << endl;
    }
    else
    {
        cout << "Match Error! The file has been modified!" << endl;
    }
}
```

整个执行过成如下所示：

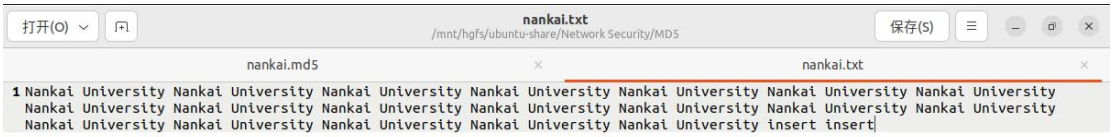
下图是 nankai.md5 文件的内容



进行两个摘要的比较，可以看到两个摘要是一致的，文件是完整的

```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ ./md5 -f nankai.txt nankai.md5
The old MD5 value of file("nankai.txt") in nankai.md5 is
e45e3c3c8ed674eefdbc41399fd7fec9
The new MD5 value of file("nankai.txt") that has computed is
e45e3c3c8ed674eefdbc41399fd7fec9
OK! The file is integrated!
```

下面对 nankai.txt 文件进行修改，如下面在原始 nankai.txt 文件中添加内容。可以看到两个摘要是不一致的，文件遭到破坏。




```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Network Security/MD5$ ./md5 -f nankai.txt nankai.md5
The old MD5 value of file("nankai.txt") in nankai.md5 is
e45e3c3c8ed674eefdbc41399fd7fec9
The new MD5 value of file("nankai.txt") that has computed is
7194b77869f0895133017f8aa920c05e
Match Error! The file has been modified!
```

四、 实验结论

本次实验着重深入学习了 MD5 算法的原理与实现，并通过编程实践进行了实验。本次实验不仅实现了 MD5 算法的核心部分，还学习了在 Linux 系统中检测文件完整性的方法。MD5 算法是一种广泛应用于信息摘要领域的算法，通过对输入消息进行填充、分割，然后对每个消息分组进行循环运算，最终生成 128 位的摘要。掌握了 MD5 算法的消息填充、分组循环运算和摘要生成过程。

通过实验编程，实现了 MD5 算法的核心部分。这包括设计 MD5 类，能够处理任意长度的消息并生成对应的 MD5 摘要。在编程过程中，需要掌握 MD5 算法中的基本操作和基本运算，以及四轮迭代计算中的基本函数和常数表。

通过编程实践，加深了对 MD5 算法的理解，并提高了编程能力和对 Linux

系统命令行操作的熟练程度。在实验过程中，不仅要理解 MD5 算法的原理，还需要将其转化为实际可运行的程序，这对综合能力提升有着积极的影响。MD5 算法在信息安全领域具有重要意义，本次实验使我对网络安全的重要性有了更深刻的认识。通过对文件完整性的检验，我意识到了 MD5 算法在确保数据完整性方面的重要作用。