网 络 安 全 技 术

实 验 报 告

学　　院：网络空间安全学院
年　　级：2021 级
专　　业：信息安全、法学双学位
学　　号：2113203
姓　　名：付政烨
手 机 号：15719340667
完成日期：2024 年 4 月 29 日

目　录

目录..... I
一、　实验目的..... 1
二、　实验内容..... 1
三、　实验步骤及实验结果..... 1
（一）　MD5 算法分析[1]..... 1
1. 消息的填充与分割 1
2. 消息块的循环运算 2
（二）　代码实现..... 4
1. MD5 类的实现 4
2. instruction 类的实现 10
3. main 函数实现..... 11
（三）　实验结果..... 12
1. Makefile 编写 12
2. 运行结果 13
四、　实验结论..... 15
参考文献..... 17

一、 实验目的

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。

二、 实验内容

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。

三、 实验步骤及实验结果

(一) MD5 算法分析[1]

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 $N \times 512 + 448$，即 $N \times 64 + 56$ 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，现在消息比特长度为：

$$N \times 512 + 448 + 64 = (N + 1) \times 512$$

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]。

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。

1. **初始向量**：MD5 算法中有四个 32 比特的初始向量。它们分别是：$A = 0x01234567, B = 0x89abcdef, C = 0xfedcba98, D = 0x76543210$。
2. **基本运算**：5 种基本运算详见表 3.1 。

表 3.1 逐比特逻辑运算符

运算符	描述
X	X 的逐比特逻辑“非”运算
$X \wedge Y$	X，Y 的逐比特逻辑“与”运算
$X \vee Y$	X，Y 的逐比特逻辑“或”运算
$X \oplus Y$	X，Y 的逐比特逻辑“异或”运算
$X \lll s$	X 循环左移 s 位

3. **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（详见表 3.2 ）。

表 3.2 逻辑函数定义

函数	定义
$F(x, y, z)$	$(x \wedge y) \vee (x \wedge z)$
$G(x, y, z)$	$(x \wedge z) \vee (y \wedge z)$
$H(x, y, z)$	$x \oplus y \oplus z$
$I(x, y, z)$	$y \oplus (x \wedge z)$

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。即。

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

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：

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

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：

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

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：

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

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：

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

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 $\lfloor 2^{32} \cdot |\sin(i)| \rfloor$ 的整数部分，其中 2^{32} 是 2 的 32 次方，$|\sin(i)|$ 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.

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

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。

4. **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。

$$MD5(M) = ABCD$$

（二） 代码实现

1. MD5 类的实现

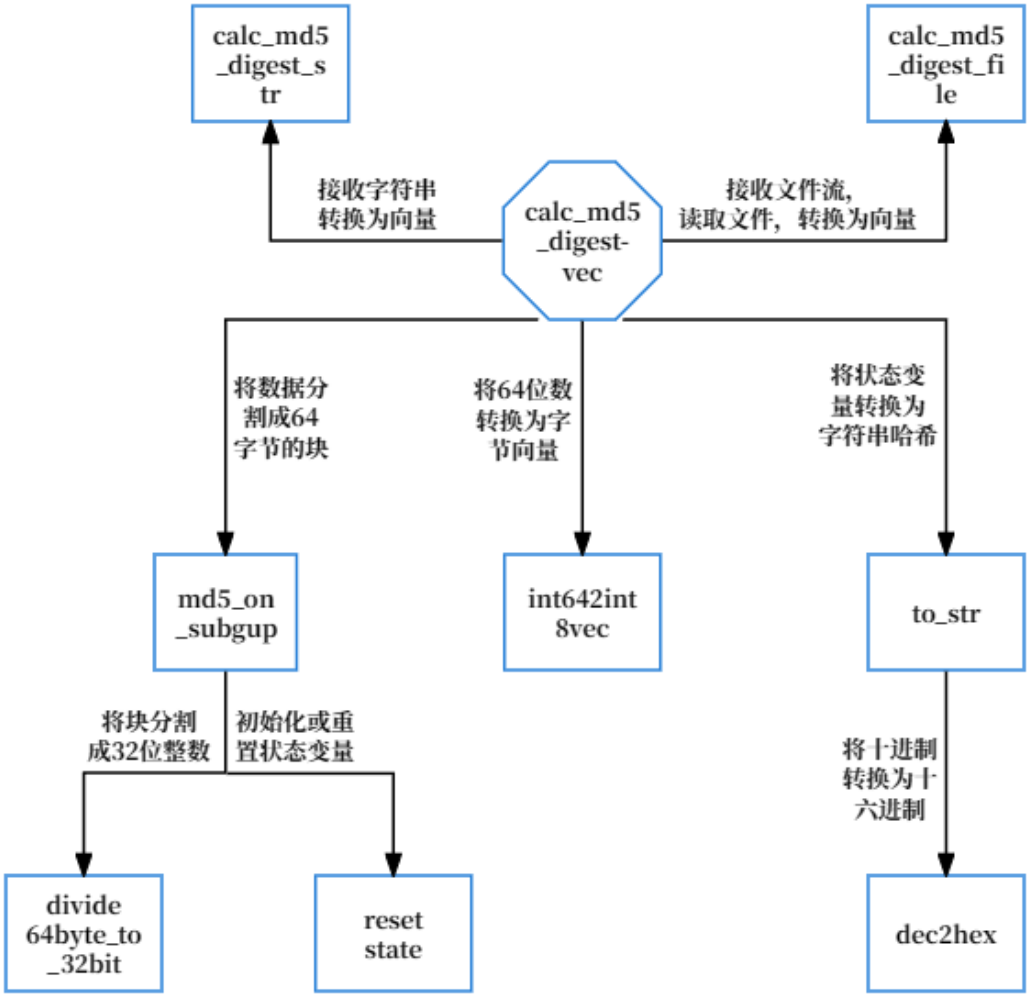

图 3.1 MD5 类函数关系图

图 3.1 展示了 MD5 类中各个函数的基础关系图。图中概括了 MD5 算法的主要函数及其相互关系。从顶部开始，流程分为三个主要分支，对应处理字符串、

文件流和字节向量的函数。这些函数最终都会调用核心的 calc_md5_digest_vec 函数，该函数负责执行 MD5 的填充处理，将数据分割成 64 字节的块，并对每个块进行进一步处理。图中突出了 md5_on_subgup 函数，它对 64 字节块执行 MD5 算法的四轮处理，以及 divide_64byte_to_32bit 函数，该函数将数据块分割成 32 位整数。在底层，reset_state 函数用于初始化算法状态，而 to_str 函数将最终的状态变量转换成字符串形式的哈希值。在转换过程中，dec2hex 函数负责将 32 位整数转换为 16 进制字符串。另外，int642int8vec 函数处理输入数据的实际长度，将 64 位整数转换为 8 个字节的向量（详见表 3.3 ）。

表 3.3 MD5 算法函数描述

函数名称	**描述**
calc_md5_digest (接受字符串)	接受一个字符串，将其转换为 uint8_t 向量，并调用 calc_md5_digest 进行处理。
calc_md5_digest (接受输入文件流)	接受一个输入文件流，读取文件内容到字符串，再将字符串转换为 uint8_t 向量，并调用 calc_md5_digest。
calc_md5_digest (接受 uint8_t 向量)	接受 uint8_t 向量作为输入，执行 MD5 的填充处理，然后将处理过的数据分割成 64 字节的块并对每个块调用 md5_on_subgup。
md5_on_subgup	对单个 64 字节块进行 MD5 的四轮处理。此函数内部首先通过调用 divide_64byte_to_32bit 将块转换成 32 位整数的向量。然后，根据 MD5 算法的规则更新内部状态。
divide_64byte_to_32bit	将 64 字节的块分割成 16 个 32 位的整数，用于 md5_on_subgup 中的处理。
reset_state()	初始化或重置 MD5 算法的四个状态变量，这是 calc_md5_digest 在开始处理新数据前所必需的。
to_str()	将 MD5 算法处理后的状态变量转换成一个字符串形式的哈希值，调用 dec2hex(uint32_t decimal) 将每个 32 位整数转为 16 进制字符串。
dec2hex	将单个 32 位整数转换为 16 进制字符串，供 to_str() 使用。
int642int8vec	将 64 位整数转换为 8 个字节的向量，用于 calc_md5_digest 中处理输入数据的真实长度。

MD5.h 提供了一个基本的 MD5 算法实现，并且支持对不同输入进行摘要计算。其包括四个基本逻辑运算函数（F、G、H、I），以及对应的循环移位操作。类 MD5 包含了计算 MD5 摘要的方法，可以对字符串或文件进行处理。在内部，算法使用了一个状态数组来存储中间结果，并提供了一系列辅助函数来支持 MD5 算法的计算过程。

md5.h 摘要

```
#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 LS(x, n) (((x) << (n)) | ((x) >> (32-(n)))) // 左旋转函数
#define FF(a, b, c, d, x, s, ac) { (a) += F ((b), (c), (d)) + (x) + ac; (a) = LS ((a), (s)); (a) += (b); }
#define GG(a, b, c, d, x, s, ac) { (a) += G ((b), (c), (d)) + (x) + ac; (a) = LS ((a), (s)); (a) += (b); }
#define HH(a, b, c, d, x, s, ac) { (a) += H ((b), (c), (d)) + (x) + ac; (a) = LS ((a), (s)); (a) += (b); }
#define II(a, b, c, d, x, s, ac) { (a) += I ((b), (c), (d)) + (x) + ac; (a) = LS ((a), (s)); (a) += (b); }
#define T(i) 4294967296 * abs(sin(i)) // 用于生成T表的函数，该表在MD5算法中用于加速
class MD5 {
public:
    void calc_md5_digest(const string &str); // 计算字符串的MD5摘要
    void calc_md5_digest(ifstream &in); // 计算文件的MD5摘要
    string to_str(); // 将MD5算法的结果转换为字符串形式
private:
    void reset_state(); // 初始化MD5算法的内部状态
    void calc_md5_digest(vector<uint8_t> input); // 对输入数据进行MD5计算
    void md5_on_subgup(const vector<uint8_t> block); // 对512位的消息块进行MD5处理
    vector<uint32_t> divide_64byte_to_32bit(const vector<uint8_t>input); // 将64字节数据划分为16个32位组
```

```
    string dec2hex(uint32_t decimal); // 将32位的十进制数转换为十六进制字符串
    vector<uint8_t> int642int8vec(uint64_t num); // 将64位整数转换为8位整数向量
    uint32_t state[4]; // 存储MD5算法中的四个缓冲区状态（A、B、C、D）
};
```

（1）calc_md5_digest 方法

MD5 类包含三个重载的 calc_md5_digest 方法，用于处理不同类型的输入：字符串、文件和字节向量[3]。

- **字符串输入**：将字符串转换为字节向量后，调用处理字节向量的方法。
- **文件输入**：读取整个文件内容为字符串，再将其转换为字节向量，之后调用处理字节向量的方法。
- **字节向量输入**：这是主要的处理逻辑，包括添加填充，确保长度符合 MD5 算法要求，并对每个 64 字节的分组进行处理。

字节向量输入

```
void MD5::calc_md5_digest(vector<uint8_t> input) {
    vector<uint8_t> true_length = int642int8vec(input.size() * 8);
    vector<uint8_t> tmp_padding(64, (uint8_t)0);
    tmp_padding[0] = (uint8_t)128;
    if (input.size() * 8 % 512 == 448)
        input.insert(input.end(), tmp_padding.begin(), tmp_padding.end());
    else {
        int index = 0;
        while(input.size() * 8 % 512 != 448)
            input.push_back(tmp_padding[index++]);
    }
    input.insert(input.end(), true_length.begin(), true_length.end());

    int md5_subgroup = input.size() / 64;
    for (int i = 0; i < md5_subgroup; ++i) {
        vector<uint8_t> md5_input;
        md5_input.insert(md5_input.end(), input.begin() + i * 64, input.begin() + (i +
```

```
            1) * 64);
18          md5_on_subgup(md5_input);
19      }
20  }
```

calc_md5_digest 对输入向量进行处理，确保其长度满足 MD5 算法的要求：

1. **长度的字节表示**：首先，计算输入数据的长度（单位：比特），并通过 int642int8vec 函数将长度的 64 位整数表示转换为字节向量。
2. **初始化填充向量**：创建一个 64 字节的填充向量，其中第一个字节设置为 128（二进制的 10000000），表示消息的结束，并在必要时用来填充数据。
3. **条件填充**：
   - 如果输入数据的长度已经满足 $input.size() * 8\%512 == 448$，即在添加长度信息后能够整除 512，直接添加整个填充向量。
   - 如果不满足，通过循环添加填充字节，直到数据长度达到合适的大小（$448 mod 512$）。
4. **添加长度向量**：将表示数据原始长度的字节向量追加到输入数据的末尾。
5. **分组处理**：将填充和长度处理后的数据分割成 64 字节的块，并对每一块调用 md5_on_subgup 函数进行进一步的 MD5 处理。

同时，calc_md5_digest 将输入数据分成多个 64 字节的组并进行处理，首先根据 MD5 算法要求，添加必要的填充，使数据长度满足特定条件（总长度应该在完成所有处理后为 448 mod 512）。在数据末尾添加表示原始数据长度的 64 位长整数。将调整后的数据分成多个 64 字节的块，每个块都单独进行 MD5 的处理。这通过 md5_on_subgup 方法实现，该方法处理每个 64 字节的块。

（2）md5_on_subgup 方法

md5_on_subgup 方法是 MD5 算法中的核心部分，负责对一个 64 字节的数据块进行四轮详细的运算处理。方法开始时，首先利用 divide_64byte_to_32bit 函数将数据块分解为 16 个 32 位的整数。这些整数随后通过四轮不同的处理函数（FF, GG, HH, II）进行处理，每轮包含 16 次操作。

第一轮使用 FF 函数，侧重于添加和混合块中的数据，通过特定的非线性运

算和位移来更新状态变量 a, b, c, d。第二轮使用 GG 函数，修改非线性函数，进一步增加数据之间的依赖性。第三轮使用 HH 函数，改变非线性函数形式，增强数据位的混淆，提高对攻击的抵抗力。最后一轮采用 II 函数，通过复杂的逻辑运算最大化扰乱处理结果，确保输出结果的随机性和预测难度[4]。

md5_on_subgup 函数实现

```
void MD5::md5_on_subgup(const vector<uint8_t> block) {
    uint32_t a = state [0], b = state [1], c = state [2], d = state [3];
    vector<uint32_t> x = divide_64byte_to_32bit (block);
    FF(a, b, c, d, x [0],  7,  T(1));
     ...
    GG (a, b, c, d, x [1],  5,  T(17));
     ...
    HH (a, b, c, d, x [5],  4,  T(33));
     ...
     II (a, b, c, d, x [0],  6,  T(49));
     ...
     state [0] += a;
     state [1] += b;
     state [2] += c;
     state [3] += d;
}
```

每步操作都将结果加回到初始状态变量中，并最终将这些变量的结果累加到全局状态数组中，为处理下一个数据块做好准备。这种方法的设计确保了 MD5 算法的不可逆性和抗碰撞性，使其能够有效地用于数字签名和数据完整性验证。

（3）其他辅助函数

1. **to_str**：将处理后的状态数组（MD5 的最终散列值）转换为 16 进制字符串输出。
2. **divide_64byte_to_32bit**：将 64 字节的数据块转换为 32 位整数数组。
3. **dec2hex**：将 32 位整数转换为 16 进制字符串。
4. **int642int8vec**：将 64 位整数转换为 8 字节的向量。

5. **reset_state**：重置 MD5 计算的初始状态，这是算法开始前的重要步骤。（详见源码）

2. instruction 类的实现

上述代码是一个 MD5 散列值计算程序的实现，它提供了一系列的命令行工具来处理和验证文件的 MD5 散列值。程序包含几个主要功能：

1. **帮助信息（help_info 函数）**：输出程序的使用说明，解释不同的命令行参数如何使用，包括查看帮助、测试 MD5 算法、计算文件的 MD5 散列值、验证文件的 MD5 散列值等。
2. **MD5 测试（md5_test 函数）**：执行 MD5 算法的自我测试，对一系列预定义的字符串计算 MD5 散列值并打印结果。这个功能验证算法的实现是否正确。
3. **计算文件 MD5（calc_md5 函数）**：根据用户指定的文件路径，计算并打印该文件的 MD5 散列值，并将结果保存到一个文件中。
4. **用户输入 MD5 验证（input_md5_recalc 函数）**：允许用户输入一个预期的 MD5 散列值，然后程序会重新计算指定文件的 MD5 散列值并与用户输入的值进行比较，验证文件的完整性。
5. **读取 MD5 文件并重新计算比较（read_md5_recalc 函数）**：从一个指定的 MD5 文件读取旧的散列值，重新计算指定文件的 MD5 散列值，并将新旧散列值进行比较，检查文件是否被修改。

instruction.h 部分摘要

```
1 void md5_test(int argc, char *argv[]);
2 void help_info(int argc, char *argv[]);
3 void calc_md5(int argc, char *argv[]);
4 void input_md5_recalc(int argc, char *argv[]);
5 void read_md5_recalc(int argc, char *argv[]);
```

这些功能使用户能够通过命令行界面方便地进行 MD5 散列值的计算和验证，适用于需要确保文件未被篡改或验证数据完整性的场景（详见源码）。

3. main 函数实现

main 实现本次实验命令行程序，该程序根据用户提供的命令执行相应的操作。代码首先定义了一个哈希表，将用户输入的命令与对应的操作函数关联起来。然后，根据用户输入的命令，在哈希表中查找相应的操作函数，并执行该函数。如果用户未提供命令或提供了无效的命令，则输出错误信息。

main 函数实现

```
int main(int argc, char *argv[]) {
    typedef void(*OpFunction)(int, char*[]);
    unordered_map<string, OpFunction> opcode_table = {
        {"-t", md5_test},
        {"-h", help_info},
        {"-c", calc_md5},
        {"-v", input_md5_recalc},
        {"-f", read_md5_recalc}
    };
    if (argc < 2) {
        return -1;
    }
    string op = argv[1];
    auto it = opcode_table.find(op);
    if (it != opcode_table.end()) {
        OpFunction func = it->second;
        func(argc, argv);
    }

    return 0;
}
```

（三） 实验结果

下面的代码实现了一个简单的 Makefile，用于编译 md5 目标程序。它定义了编译器和编译选项，使用变量 OUTDIR 指定了输出目录为 bin，并且指定了目标文件名为 md5。然后将这些目标文件链接在一起，生成最终的可执行文件。此外，还定义了一个 clean 命令用于删除生成的目标文件和可执行文件。

1. Makefile 编写

main 函数实现

```
CC := g++
CFLAGS := −std=c++11
OUTDIR := bin

TARGET := $(OUTDIR)/md5
SOURCES := $(wildcard ∗.cpp)
OBJECTS := $(SOURCES:%.cpp=$(OUTDIR)/%.o)

$( shell  mkdir −p $(OUTDIR))

all :  $(TARGET)

$(TARGET): $(OBJECTS)
    $(CC) $(OBJECTS) −o $@


$(OUTDIR)/%.o: %.cpp
    $(CC) $(CFLAGS) −c $< −o $@

.PHONY: clean

clean :
    −rm −rf $(OUTDIR)
```

```
fu@fu-virtual-machine:~/桌面/lab3$ make clean
rm -rf bin
fu@fu-virtual-machine:~/桌面/lab3$ make
g++ -std=c++11 -c instruction.cpp -o bin/instruction.o
g++ -std=c++11 -c main.cpp -o bin/main.o
g++ -std=c++11 -c md5.cpp -o bin/md5.o
g++ bin/instruction.o bin/main.o bin/md5.o -o bin/md5
```

图 3.2 MakeFile 运行示例

2. 运行结果

（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。

```
fu@fu-virtual-machine:~/桌面/lab3$ cd bin
fu@fu-virtual-machine:~/桌面/lab3/bin$ ./md5 -h
+-------------------------------------------------+
|            程序的使用方法如下:                  |
+-------------------------------------------------+
| 请输入./md5 -[]来进行相应的操作                 |
|   [-h] --获取程序用法                           |
|        --输入./md5 -h即可                       |
|   [-t] --测试MD5算法                            |
|        --输入./md5 -t即可                       |
|   [-c] --计算被测试文件的MD5摘要并打印          |
|       用法为: ./md5 -c filepath                 |
|   [-v] --用户输入测试文件的MD5摘要，然后重新计算|
|         测试文件的MD5摘要，并逐位比较两个摘要。 |
|         用法为: ./md5 -v filepath               |
|   [-f] --程序读取输入的.md5文件的摘要，并重新计算|
|         测试.txt文件的md5，最后将两者进行逐位比较|
|       用法为: ./md5 -f .txt filepath .md5 filepath|
+-------------------------------------------------+
```

图 3.3 ./MD5 –h

（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。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。因此我们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。

图 3.4 ./MD5 –t

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

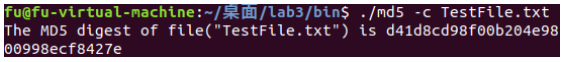

图 3.5 ./MD5 –c

（4）验证文件完整性

在控制台命令行中输入：

```
/MD5 –c [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。整个执行过程如下所示。

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。整个执行过成如下所示。

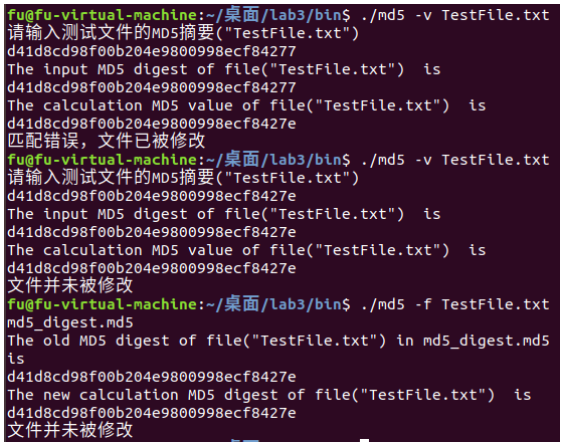

图 3.6 验证文件完整性

（5）利用系统工具 md5sum 为测试文件生成 MD5 摘要

在控制台命令行输入：

```
1 md5sum TestFile.txt
```

在本次实验中，除了亲自实现 MD5 算法外，我还使用了 Linux 系统中的标准命令行工具 md5sum 生成文件的 MD5 摘要。通过将手动实现的算法输出与 md5sum 工具的结果进行比较，两者的摘要值完全一致，从而验证了自行编写程序的准确性和可靠性。

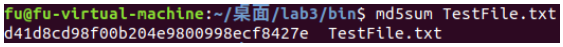

图 3.7 系统工具 md5sum 生成的 MD5 摘要

四、 实验结论

在这次关于 MD5 算法的实验中，我不仅深入了解了 MD5 算法的原理和具体实现，而且亲自动手编写了代码，通过实践提升了自己的编程技能和对信息安全算法的理解。

首先，实验让我对 MD5 算法有了全面的认识。MD5（Message Digest Algorithm 5）是一种广泛使用的加密散列算法，生成固定长度的 128 位散列值，主要用于确保信息传输完整无误。通过这次实验，我详细了解了 MD5 的工作流程，包括填充、分组、计算和输出等步骤。在实验中，我手动实现了这一过程，这让我对算法的每个环节都有了切实的理解。编程实践是这次实验的核心部分。通过

使用 C++ 编程语言实现 MD5 算法，我不仅复习和应用了之前学习的数据结构和算法知识，而且还学习到了如何在实际编程中处理位运算和循环逻辑。在编码过程中，我遇到了诸如数据类型转换、内存管理等问题，这些问题的解决增强了我解决实际问题的能力。此外，这次实验也让我认识到了算法实现的复杂性和实用性。MD5 虽然因其安全性问题在某些安全要求极高的场合逐渐被其他更安全的算法（如 SHA-256）替代，但在许多不那么敏感的应用中仍然十分有用。通过这次实验，我了解到无论是在学术还是在工业应用中，理论知识与实际应用之间往往存在差距，而通过实践可以有效地桥接这一差距[5]。

实验中的困难和挑战也让我学到了如何查找资料和解决问题。在编写代码的过程中，我经常需要查阅在线文档、阅读相关书籍和论文以解决编程中遇到的具体问题。这种能力在任何技术领域都是非常宝贵的。这次 MD5 算法的实验不仅增强了我的编程能力，也加深了我对信息安全领域的理解和兴趣。通过亲自实现和测试一个实用的加密算法，我更加确信自己未来想要在计算机科学和网络安全领域寻求发展。这次实验经验将为我后续的学习和研究奠定坚实的基础。

参考文献

[1] 吴功宜. 网络安全高级软件编程技术[M]. 网络安全高级软件编程技术, 2010

[2] 吴功宜. 计算机网络高级教程[J]. 计算机教育, 2008(1): 1

[3] 张裔智, 赵毅, 汤小斌. Md5 算法研究[J]. 计算机科学, 2008, 35(7): 3

[4] Boer B D, Bosselaers A. Collisions for the compression function of md5[C]. Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology. 1994

[5] Update A, Update S. The status of md5 after a recent attack[J]. 1996