南開大學
Nankai University

网络空间安全学院

网络安全技术实验报告

实验三：基于 MD5 算法的文件完整性校验程序

学院： 网安学院

年级： 2021 级

班级： 信息安全一班

学号： 2111408

姓名：周钰宸

手机号：13212168838

2024 年 5 月 11 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。

实验目的

本次实验的目的如下：

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。

实验要求

最终达到的要求为：

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。

本次实验，我按照上述实验要求，**在 Linux 系统中完整地复现了 MD5 算法，并且实现了通过命令行参数，利用复现的算法进行多种功能，包括生成 MD5 摘要和文件完整性校验等。最后还实现了两种策略来抵御 MD4 的字典攻击来增强 MD5 的安全性。具体工作详见实验内容部分。**

2 实验原理

2.1 MD5 算法原理

2.1.1 消息的填充与分割

图 2.1: MD5 填充策略

MD5 算法的第一步是准备输入消息以便处理。这一过程包括填充消息和分割消息，以满足算法处理的需要。

如图2.1所示，清晰地展示了 MD5 进行消息的填充和分割时候的基本原理。具体而言。

消息填充 首先，消息必须按照一定的规则填充，使其长度模 512 等于 448。填充的方式是，首先添加一个 '1'，然后添加若干个 '0'，直到消息长度达到 64 位少于下一个 512 的倍数。具体的填充公式可以表示为：

$$L + 1 + K = 448 \mod 512 \tag{1}$$

其中，L 是原始消息长度，K 是填充的 0 的数量。

在 MD5 算法的消息填充过程中，存在几种特殊情况需要详细说明。首先，考虑消息长度 L 模 512 的结果正好等于 448 位的情况。根据 MD5 的填充规则，消息必须至少填充 1 位。因此，当 $L \mod 512 = 448$ 时，算法将不仅填充至当前块的末尾，还将添加一个额外的 512 位块，完全由填充位构成。

另外，若 $L \mod 512$ 的结果位于 448 和 512（模 512 的结果为 0）之间，处理方式相似。算法将添加一个新的 512 位块，其中填充的位数为 $512 - (L \mod 512 - 448)$。

最后，如果 $L \mod 512$ 的结果小于 448，则仅需在当前块填充 $448 - (L \mod 512)$ 位。

$$\text{Padding Bits} = \begin{cases} 512 & \text{if } L \mod 512 = 448 \\ 512 - (L \mod 512 - 448) & \text{if } 448 < L \mod 512 < 512 \\ 448 - (L \mod 512) & \text{if } L \mod 512 < 448 \end{cases} \tag{2}$$

这些规则确保了每个消息块都能满足 MD5 算法处理的需求，同时保持了算法的一致性和完整性。

附加长度 在消息的末尾，我们还需要附加原始消息的长度（一个 64 位的整数），表示为 L。这样，整个消息的总长度就变为了 512 的倍数。

消息分割 填充并附加长度之后的消息被分割成若干个 512 位的块。每个块进一步分成 16 个 32 位的子块。这可以用下面的公式表达：

$$M = M_0, M_1, \ldots, M_{N-1} \tag{3}$$

其中 M_i 是第 i 个 32 位的子块，N 是 512 位块的总数。

这样处理后的消息块将被用于接下来的 MD5 压缩函数。

2.1.2 消息块的循环运算

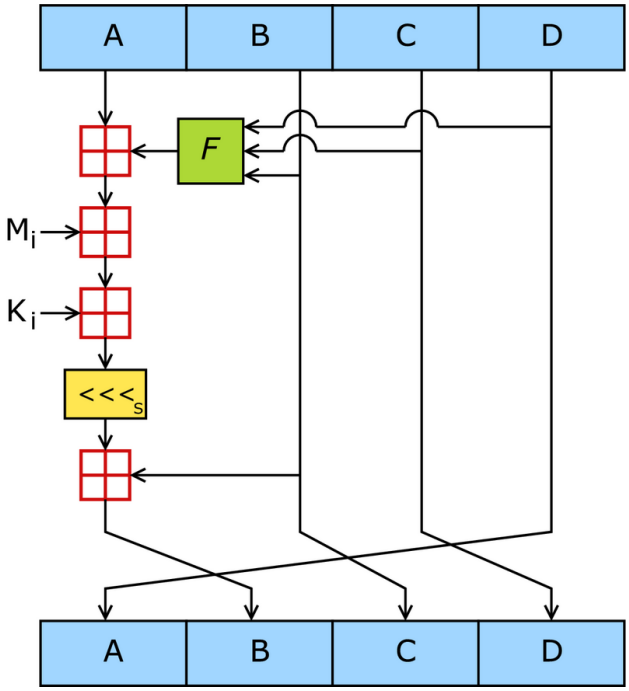

图 2.2: MD5 的消息块处理策略

MD5 算法中，**对每个 512 位的消息块进行循环运算是生成消息摘要的核心步骤**。这个过程涉及四轮循环运算，每轮包含 16 个相似的操作，使用不同的非线性函数和常数。

如图2.2所示，清晰地展示了 MD5 消息块进行循环运算时候的基本原理，具体而言。

四轮循环函数 MD5 使用以下四个非线性函数，每个函数对应一轮运算：

$$
\begin{aligned}
F(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\
G(X,Y,Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\
H(X,Y,Z) &= X \oplus Y \oplus Z \\
I(X,Y,Z) &= Y \oplus (X \vee \neg Z)
\end{aligned}
$$

其中 $\wedge$, $\vee$, $\neg$, 和 $\oplus$ 分别表示 AND, OR, NOT, 和 XOR 运算。

循环运算过程 每个 512 位的消息块通过以下步骤进行处理：

1. 初始化四个寄存器 A, B, C, D 为一个 128 位的链变量。
2. 对每个 512 位的消息块进行四轮循环处理。每一轮对 16 个子块使用上述函数进行 16 次操作。
3. 每次操作更新一个寄存器，如下所示：

$$A = B + ((A + F(B,C,D) + X[k] + T[i]) \lll s) \tag{4}$$

其中 $+$ 表示模 2^{32} 的加法，$\lll$ 表示循环左移，$X[k]$ 是消息的第 k 个 32 位字，$T[i]$ 是第 i 步使用的常数，s 是指定的位移量。

4. 每轮结束后，将 A, B, C, D 的结果与链变量的初始值相加。

2.1.3 摘要的生成

如图2.2所示，MD5 的摘要最后只需让已经经过了若干轮迭代的循环运算得到的初始向量，**即 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 比特的摘要即可**。

2.2 字典攻击与 MD5 变换算法

2.2.1 字典攻击的原理与实践

MD5 算法由于其设计特性，尽管不可逆，**但可以通过字典攻击被破解**。字典攻击涉及预先收集大量的明文与其对应的 MD5 散列值，并存储于数据库中。在需要破解特定 MD5 散列时，可以直接查询数据库找到对应的明文。这种方法的效率高，但依赖于已有的大规模散列集合。市场上有多种资源提供这样的 MD5 散列字典服务，覆盖了从几百万到数十亿的散列记录。

2.2.2 MD5 算法变换方法

为了提升 MD5 算法的安全性，可以采用几种变换策略来防御字典攻击。这些策略包括：

多次散列变换 对同一数据进行多次连续的 MD5 散列运算，每次散列的输出作为下一次的输入，从而生成一个不同于单次 MD5 运算的最终散列值。

分割散列变换 初始对数据进行一次 MD5 散列后，将结果散列值分割成若干部分，对每个部分再次进行 MD5 散列，然后合并这些散列结果，并可能再次进行散列，以生成最终的输出。

其他变换技术 包括添加额外的字符串（如盐值）、改变字符顺序或修改字符大小写等，都是用来增加 MD5 算法针对字典攻击的抵抗力。这些方法通过引入额外的不确定性来防止攻击者直接使用现成的 MD5 字典。

这些变换策略通过增加计算复杂度和引入不确定性来有效延缓或阻止字典攻击，从而提高了 MD5 算法的安全性。**本次实验就通过实现了前两种算法变换技术提高了 MD5 的安全性**。

2.3 Command Line Interface (CLI)

命令行接口（CLI）为用户提供了一种直接与程序交互的方式，通过在命令行中输入特定的参数和命令来控制程序的行为。在本实验中，我的程序被设计为可以接受多种命令行参数，以触发不同的功能。这种设计有以下几个显著的好处：

灵活性 使用命令行参数可以极大增强程序的灵活性。用户可以根据需要选择性地执行特定的操作，而无需通过图形用户界面的多个步骤。这一点对于自动化任务和脚本编写尤为重要。

易于自动化 命令行程序容易被脚本化和自动化。例如，在批处理操作或集成到其他软件工作流程中时，可以通过简单地编写脚本来控制程序的行为，而不需要人工干预。

资源效率 与图形用户界面（GUI）相比，CLI 通常需要更少的系统资源。这使得命令行工具特别适合于资源受限的环境，如服务器或嵌入式系统。

兼容性 命令行接口易于在不同的操作系统和环境中实现兼容性。只要操作系统支持基本的命令行操作，就能运行相同的程序，这在跨平台开发中非常有用。

通过实现一个功能丰富的命令行接口，本实验程序提供了一个强大而灵活的工具，适用于各种用户需求，能实现各种功能。从简单的日常任务到复杂的自动化处理。

3 实验内容

本次实验我依据实验目的，在 Linux 系统中完整地复现了 MD5 算法，并且实现了通过命令行参数，利用复现的算法进行多种功能，包括生成 MD5 摘要和文件完整性校验等。最后还实现了两种策略来抵御 MD4 的字典攻击来增强 MD5 的安全性，并提供了缺省的参数和默认文件提供程序交互的灵活性。具体而言：

实验内容

1. **平台与技术**:
   - 实验在 Linux 平台下进行。
   - 程序正确实现了 MD5 算法。
2. **功能**:
   - **生成 MD5 摘要**：程序能够为任意长度的字符串及任意大小的文件生成 MD5 摘要。
   - **验证文件完整性**：程序支持两种方式验证文件的完整性。
   - **额外功能**：程序还支持针对字典攻击的两种 MD5 算法变化策略，包括多次散列和分割散列变换。以及对参数的缺省选项，使用默认文件等选择。
3. **文件完整性验证方法**:
   - **手动输入 MD5 摘要比对**:
     - 用户手动输入一个已知的 MD5 摘要。
     - 程序计算当前文件的 MD5 摘要。
     - 对比两个摘要值，若相同，则确认文件完好；若不同，提示文件可能已被篡改或损坏。
   - **自动.md5 文件比对**:
     - 使用 Linux 系统工具 `md5sum` 生成目标文件的 MD5 摘要，并保存为后缀为`.md5` 的同名文件。
     - 程序读取并计算被测文件的 MD5 摘要。
     - 将程序计算的摘要与`.md5` 文件中的摘要进行比较。
     - 根据摘要是否一致，输出文件是否完整的结果。

3.1 规范习惯

由于本次实验我依旧在 Linux 平台进行编程。并依然严格要求自己，采用了非常规范的工程编程习惯。极为清晰的框架增强了我的代码的可读性和可维护性。这一点也会在具体实验步骤中展开描述。

```
/Lab03
├── bin/ Linux 平台可执行程序与测试文件
│   ├── MD5/ MD5 可执行程序
│   ├── test/ 默认待检验测试文件
│   └── MD5.md5/ 默认存储 MD5 值的.md5 文件
├── include/ C++ 项目头文件
│   ├── def.hpp/ 宏定义等全局定义
│   └── MD5.hpp/ MD5 类的成员以及函数声明
├── src/ C++ 项目实现文件
│   ├── MD5.cpp/ MD5 类的函数实现
│   └── main.cpp/ 主函数：实现 CML 交互以及文件完整性校验等功能
├── tmp/ 存放 MD5 正确性验证功能的.md5 临时文件的目录
└── README.md 提供对项目的介绍和一些操作的文档提示
```

3.2 程序交互方式

3.2.1 程序运行格式

本次的可执行程序在 bin 子目录下的 MD5 文件（没有后缀，方便直接运行）。**运行时的命令参数如下：**

```
./MD5 [-Option] [file path of the file validated] [file path of the .md5 file]
                                                  [times]
```

即

```
./MD5 [选项] [被测文件路径] [.md5 文件路径] [计算次数]
```

其中各个参数含义为：

1. **[-Option]：选项参数**。可选的选项有 {-h，-t，-c，-v，-f，-fm}。
2. **[file path of the file validated]：被测文件路径**。指明用来计算 MD5 值 (-c) 或者进行文件完整性校验 (-f 或-fm) 时候的被测文件路径。
   - 在计算 MD5 值 (-c) 时，该路径不能省略。
   - 在进行手动文件完整性校验 (-v) 时，该路径可以省略，省略时采用 test 文件作为默认文件。
   - 在进行自动文件完整性校验 (-f) 时，要么与.md5 文件路径一起省略（省略时采用 test 文件作为默认文件），要么不能省略。
   - 在进行另一个参数选项的自动文件完整性校验 (-fm) 时，该路径可以省略，省略时采用 test 文件作为默认文件。

3. [**file path of the .md5 file**]：**.md5 文件路径**。指明根据被测文件内容生成的.md5 文件在文件系统中的路径。需要由用户放入到与 MD5 程序同级目录下，可以用户自动输入到.md5 文件中，也可以通过使用 Linux 系统工具 `md5sum` 生成。

   - 在进行文件完整性校验 (-f) 时，要么与被测文件路径一起省略（省略时采用 `MD5.md5` 文件作为默认文件），要么不能省略。
   - 在进行另一个参数选项的文件完整性校验 (-fm) 时，该路径不可以省略。

4. [**times**]：**进行 MD5 值计算时的计算次数**。为了抵抗 MD5 的字典攻击，通过在运行程序时候提供该命令行选项指明计算次数，可以进行 MD5 算法变换。

   - **times=1**：只进行一次运算便返回结果。
   - **times>1**：按照对应次数执行多次散列变换以增强安全性。
   - **times=-1**：执行分割散列变换以增强安全性。

   该参数在选项参数为-c 时不能省略。

上述命令行参数其中 `[Option]` 不能省略，其它选项均可以根据需要进行选择，以此实现多功能的交互性需求。

3.2.2 参数功能介绍

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。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。

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。

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] [计算次数]`。根据计算次数的不同会执行不同的安全性操作。可以执行多次散列变换和分割散列变化来提到 MD5 摘要的安全性。计算出的被测文件的 MD5 摘要并打印出来。被测文件必须与可执行文件 MD5 处于同一个目录中。

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]`。此处的参数 `[被测文件路径]` 选项可以进行省略，省略时采用 `test` 文件作为默认文件。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。同样的，被测文件必须与可执行文件 MD5 处于同一个目录中。

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。同样的，被测文件和.md5 必须与可执行文件 MD5 处于同一个目录中。此时可以省略.md5 文件路径，则采用 `MD5.md5` 文件作为默认.md5 文件。若两个文件路径均省略，则采用 `test` 文件作为默认被测文件，`MD5.md5` 文件作为默认.md5 文件。

6. **验证文件完整性方法三：**在控制台命令行输入./MD5 -fm [被测文件路径] [.md5 文件路径]，其实现的功能与-f 选项的功能一致。唯一不一样的地方在于通过此命令行选项，可以实现省略被测文件路径，省略时采用 test 文件作为默认文件，只提供.md5 文件路径。若两个文件路径均省略，则采用 test 文件作为默认被测文件，MD5.md5 文件作为默认.md5 文件。

4 实验步骤

接下来我会按照之前所述我的程序代码架构以此介绍我的各部分代码，将会结合具体代码解释其思路。

4.1 include

4.1.1 def.hpp

该头文件是其它头文件所包含的底层头文件，主要**通过宏定义的方式定义了一系列辅助程序功能的常量**。具体而言：

1. **#define DEFAULT_TMP_DIR "tmp"**
   默认的临时目录,用于保存程序校验正确性(-t 选项)过程中的文件。**文件格式命名为 RFC-{i}.md5，其中 i 为测试样例的号码**。

2. **#define DEFAULT_FILE_NAME "test"**
   缺省时默认的测试文件名，用于进行有效性验证。

3. **#define DEFAULT_MD5_FILE "MD5.md5"**
   缺省时默认的 MD5 哈希文件名，用于存储 MD5 哈希值。

4. **#define SUCCESS 0 与 #define FAILURE 1**
   程序成功与结束时返回的代码。这是由于**为了解决有可能文件打开失败带来的错误等进行处理。保证程序的正确性和安全性**。

5. **#define MANUAL 0 与 #define AUTO 1**
   这个是用于程序命令行交互时候文件完整性校验的两个方法的区分，MANUAL 表示手动验证，AUTO 表示自动验证。

6. **#define pHelp "-h", pTest "-t", pCompute "-c", pValidate "-v", pFile "-f", pFileMD5 "-fm"**
   对应程序命令行交互时候的可选选项，即 {-h, -t, -c, -v, -f, -fm}。

7. typedef int INT
   定义 INT 为整型数据类型。

具体代码如下：

```

// Defines the constants and types used in the MD5 validation algorithm.

```

```
#ifndef MD5_DEFS_H
#define MD5_DEFS_H

#define DEFAULT_TMP_DIR "tmp"      // Default temporary directory for saving files for correctne
#define DEFAULT_FILE_NAME "test"   // Default file name to be validated
#define DEFAULT_MD5_FILE "MD5.md5" // Default file name for MD5 hash
#define SUCCESS 0                  // Success return code
#define FAILURE 1                  // Failure return code
#define MANUAL 0                   // Manual validation approach
#define AUTO 1                     // Auto validation approach

/* Options for CommandLine Interaction */
#define pHelp "-h"     // Help option
#define pTest "-t"     // Test option
#define pCompute "-c"  // Compute option
#define pValidate "-v" // Manual validate option
#define pFile "-f"     // Automatic file validate option
#define pFileMD5 "-fm" // Automatic file validate option with default MD5 file

typedef int INT; // Define INT as an integer type

#endif // MD5_DEFS_H
```

4.1.2 MD5.hpp

接下来是 MD5.hpp 文件，通过 **MD5 类，集合了所有需要用到的常量，成员变量以及成员函数**。这样使用面向对象的形式进行封装，方便了函数的调用和变量的管理。

其中比较重要的常量与成员变量有：

- **uint32_t IV[4]**
  初始向量，实际上就是 `A`，`B`，`C`，`D`。这四个值不是随机的，而是根据 MD5 算法的标准规范设定的固定值，这些值是算法正确执行的基础。
- const uint32_t s_table[4][4]
  用于 roundHandler() 处理函数中的每轮循环位移操作的常量二维数组。
- const uint8_t padding[64]
  存放除了最后的填充长度以外的填充字节，第一个字节一定是 `0x80`，因为一定是 1 开头。
- uint8_t *messageDigest
  最终以字节形式存放的摘要数组。将 128 位分为了 16 份，每份是一个字节。
- **uint8_t *bufferBlock**
  用来对输入字节流进行缓冲的缓冲块。当填满了后会进行处理，用于 `update()` 函数。

- uint32_t *bitsCounter
  用于记录已经处理位数的常量数组。其中 `bitsCounter[0]` 表示低 32 位，`bitsCounter[1]` 表示高 32 位。同样用于 `update()` 函数。
- bool isFinished
  用于标记 MD5 算法摘要计算是否已经结束。以此来控制整体控制流程。

除此之外，还对一系列成员函数进行了声明，具体其功能和作用会在 `MD5.cpp` 中详细介绍。这里不再赘述，也可以通过下面代码的注释进行简单了解。

```
1
2  // Defines the MD5 class including constants and interfaces for MD5 algorithm.
3
4  #ifndef MD5_HPP
5  #define MD5_HPP
6  #include "def.hpp"
7  #include <cstdint>
8  #include <vector>
9  #include <cstring>
10 #include <filesystem>
11 #include <fstream>
12 #include <iostream>
13
14 class MD5
15 {
16 private:
17     /**
18      * Constants for MD5 algorithm
19      */
20     // Initial vectors for MD5 algorithm
21     uint32_t IV[4] = {0x67452301, 0xefcdab89,
22     0x98badcfe, 0x10325476};
23     // Shift values for each round
24     const uint32_t s_table[4][4] = {{7, 12, 17, 22}, {5, 9, 14, 20},
25     {4, 11, 16, 23}, {6, 10, 15, 21}};
26
27     // Padding values for each round
28     const uint8_t padding[64] = {0x80};
29     // The final message digest in Bytes form
30     uint8_t *messageDigest = new uint8_t[16];
31     // The buffer block for the input message
32     uint8_t *bufferBlock = new uint8_t[64];
33     // The counter for the number of bits in the input message
```

```
34      uint32_t *bitsCounter = new uint32_t[2];
35      // The flag to indicate whether the MD5 algorithm is finished
36      bool isFinished;
37
38      /**
39       * Preprocessing for MD5: padding the input message & decoding the input message
40       */
41       // Decode the MD5 message digest from BYTE to DWORD
42      void decode(const uint8_t *input, uint32_t *output, size_t length);
43      /**
44       * Basic operations commonly used in MD5 algorithm
45       */
46      uint32_t F(uint32_t x, uint32_t y, uint32_t z);
47      uint32_t G(uint32_t x, uint32_t y, uint32_t z);
48      uint32_t H(uint32_t x, uint32_t y, uint32_t z);
49      uint32_t I(uint32_t x, uint32_t y, uint32_t z);
50      uint32_t leftRotate(uint32_t x, uint32_t n);
51
52      /**
53       * Higher-level nonlinear operations used in MD5 algorithm to update initial vectors
54       */
55      void FF(uint32_t &a, uint32_t &b, uint32_t &c,
56      uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i);
57      void GG(uint32_t &a, uint32_t &b, uint32_t &c,
58      uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i);
59      void HH(uint32_t &a, uint32_t &b, uint32_t &c,
60      uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i);
61      void II(uint32_t &a, uint32_t &b, uint32_t &c,
62      uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i);
63
64      INT roundHandler(const uint8_t block[64]); // Handler for each round of MD5 algorithm
65
66      INT update(const void *input, size_t length);     // Bytes flow
67      INT update(const std::string &input_str);         // String flow
68      INT update(std::ifstream &input_file);            // File flow
69      INT update(const uint8_t *input, size_t length); // Update the status of the MD5 object
70
71      /**
72       * Postprocessing for MD5
73       */
74       // Encode the MD5 message digest from DWORD to BYTE
75      void encode(const uint32_t *input, uint8_t *output, size_t length);
```

```
76      // Convert the BYTE array to HEX string
77      std::string bytesToHexString(const uint8_t *input, size_t length);
78      // Finalization for MD5 algorithm, including padding and process the last block
79      void finalize();
80
81  public:
82      MD5();
83      ~MD5();
84
85      /**
86       * Computing Message Digest for the input string or file
87       */
88      INT compute(const void *input, size_t length); // Bytes flow
89      INT compute(const std::string &input_str);     // String flow
90      INT compute(std::ifstream &input_file);        // File flow
91
92      // Return the final message digest in Bytes form
93      const uint8_t *getMessageDigest();
94      // Return the final message digest in HEX string form
95      std::string toString();
96      // Reset the MD5 object especially for the inital vectors
97      void reset();
98  };
99
100 #endif // MD5_HPP
```

4.2 src

接下来会介绍具体的实现文件，主要就是实现 MD5 算法的 `MD5.cpp` 和实现了命令行交互式摘要计算和文件完整性验证的 `main.cpp`。

4.2.1 MD5.cpp

首先这里介绍 `MD5.cpp` 中对 `MD5` 类中成员函数的实现。

1. **初始化与重置**：

   (1) **构造函数初始化**

   MD5 类的构造函数负责设置算法的初始状态。在此过程中，`isFinished` 标志首先被设置为 `false`，表示 MD5 的计算过程尚未完成。此外，`bufferBlock` 数组被清零，用于临时存储即将被处理的数据块。同时，`bitsCounter` 数组，负责追踪处理的数据总位数，也被初始化为零。这些步骤确保了算法从一个干净的状态开始，防止旧数据的残留影响新的计算。

```
1  MD5::MD5() : isFinished(false)
```

```
{
    memset(bufferBlock, 0, sizeof(bufferBlock));
    memset(bitsCounter, 0, sizeof(bitsCounter));
    memset(messageDigest, 0, sizeof(messageDigest));
}
```

(2) **状态重置 `reset()`**

`reset` 方法提供了一种方式，用于在不同的计算阶段重新初始化 MD5 对象的状态。这在连续使用同一 MD5 实例进行多次数据处理时极为重要。通过这个方法，`isFinished` 标志被重新设置为 `false`，内部状态寄存器 `IV` 被重新初始化为 MD5 算法规定的特定起始值，这四个初始向量分别是：0x67452301, 0xefcdab89, 0x98badcfe, 和 0x10325476。此外，`bufferBlock`, `bitsCounter`, 和 `messageDigest` 数组都会被清零，确保了从一个一致的、预定义的状态开始处理新的数据。

```
void MD5::reset()
{
    /**
     * Reset initial vectors for later usage
     */
    isFinished = false;
    IV[0] = 0x67452301;
    IV[1] = 0xefcdab89;
    IV[2] = 0x98badcfe;
    IV[3] = 0x10325476;
    memset(bufferBlock, 0, sizeof(bufferBlock));
    memset(bitsCounter, 0, sizeof(bitsCounter));
    memset(messageDigest, 0, sizeof(messageDigest));
    return;
}
```

2. **辅助函数**：

(1) **基础运算 F, G, H 与 I**

直接通过位运算实现，由此来提高算法的时间性能，这些都会进一步用于构建 `roundHandler()` 中的 `FF()`, `GG()`, `HH()` 和 `II()`。

```
// Basic operations commonly used in MD5 algorithm
uint32_t MD5::F(uint32_t x, uint32_t y, uint32_t z)
{
    return (x & y) | (~x & z);
}
uint32_t MD5::G(uint32_t x, uint32_t y, uint32_t z)
{
```

```
    return (x & z) | (y & ~z);
}
uint32_t MD5::H(uint32_t x, uint32_t y, uint32_t z)
{
    return x ^ y ^ z;
}
uint32_t MD5::I(uint32_t x, uint32_t y, uint32_t z)
{
    return y ^ (x | ~z);
}
```

(2) **循环位移 eftRotate(uint32_t x, uint32_t n)**

直接也是通过位运算实现,由此来提高算法的时间性能,这些都会进一步用于构建 roundHandler() 中的 FF(), GG(), HH() 和 II()。

```
uint32_t MD5::leftRotate(uint32_t x, uint32_t n)
{
    /**
     * Perforom left rotation on x by n bits
     * @param x: the input value
     * @param n: the number of bits to rotate
     */
    return (x << n) | (x >> (32 - n));
}
```

(3) **非线性处理函数 FF(), GG(), HH(), 和 II()**

MD5 算法中使用的非线性函数 FF(), GG(), HH(), 和 II() 起到了至关重要的作用。这些函数设计用于在算法的每一轮中处理数据块，通过特定的非线性数学运算确保哈希算法的复杂性和安全性。每个函数都采用不同的方法来混合输入数据的位，从而确保对输入数据的微小变化极其敏感，即使是一位的改变也会导致输出哈希值的显著不同。**这些辅助函数的设计关键在于它们能够破坏数据的任何规律性，从而提高算法的抗碰撞性和安全性**。

```
    // Higher-level operations used in MD5 algorithm
void MD5::FF(uint32_t &a, uint32_t &b, uint32_t &c,
uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i)
{
    /**
     * Nonlinear operations on inital vectors (a, b, c, d) using F
     * Based on hard-coded solution: a = b + ((a + I(b, c, d) + X[k] + T[i]) <<< s)
     * @param a, b, c, d: initial vector
     * @param M_k: the k-th block of message
     * @param s: shift value
     * @param i: the i-th round
```

```
12      */
13     a = b + (leftRotate(a + F(b, c, d) + M_k + T_i, s));
14     return;
15 }
16
17 void MD5::GG(uint32_t &a, uint32_t &b, uint32_t &c,
18 uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i)
19 {
20     a = b + (leftRotate(a + G(b, c, d) + M_k + T_i, s));
21     return;
22 }
23
24 void MD5::HH(uint32_t &a, uint32_t &b, uint32_t &c,
25 uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i)
26 {
27     a = b + (leftRotate(a + H(b, c, d) + M_k + T_i, s));
28     return;
29 }
30
31 void MD5::II(uint32_t &a, uint32_t &b, uint32_t &c,
32 uint32_t &d, uint32_t M_k, uint32_t s, uint32_t T_i)
33 {
34     a = b + (leftRotate(a + I(b, c, d) + M_k + T_i, s));
35     return;
36 }
```

(4) **DWORD 转换 BYTE 流** `encode(const uint32_t *input, uint8_t *output, size_t length)`

此函数负责将 MD5 算法内部使用的 32 位字（DWORD）数组转换为字节（BYTE）流。**这一转换对解决由于不同计算机架构造成的字节序问题至关重要**。此函数通过显式指定字节的顺序，保证 MD5 哈希在不同的硬件平台上能够被正确地计算和解释。

```
1  void MD5::encode(const uint32_t *input, uint8_t *output, size_t length)
2  {
3      /**
4       * Encode the MD5 message digest from DWORD to BYTE
5       * @param input: the input message digest in DWORD form
6       * @param output: the output message digest in BYTE formj
7       * @param length: the length of the input messjage digest
8       */
9      for (size_t i = 0, j = 0; j < length; i++, j += 4)
10     {
```

```
        output[j] = (uint8_t)(input[i] & 0xFF);
        output[j + 1] = (uint8_t)((input[i] >> 8) & 0xFF);
        output[j + 2] = (uint8_t)((input[i] >> 16) & 0xFF);
        output[j + 3] = (uint8_t)((input[i] >> 24) & 0xFF);
    }
}
```

(5) **BYTE 流转为 DWORD** `decode(const uint8_t *input, uint32_t *output, size_t length)`

与 encode 函数相反，decode 函数从字节流中重构 32 位字（DWORD）。此操作同样关键，因为它确保无论原始数据如何存储，内部处理总是在一个统一的字节序上进行。这允许 MD5 算法独立于平台的字节序，增强了其跨平台的兼容性。

```
void MD5::decode(const uint8_t *input, uint32_t *output, size_t length)
{
    /**
     * Decode the MD5 message digest from BYTE to DWORD
     * @param input: the input message digest in BYTE form
     * @param output: the output message digest in DWORD form
     * @param length: the length of the input message digest
     */
    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);
    return;
}
```

(6) **字节转换为字符串** `bytesToHexString(const uint8_t *input, size_t length)`

该函数用于摘要计算完成后的摘要显示，将以字节形式显示的摘要转换为十六进制字符形式。函数首先检查 ifFinished 是否为 true，否则返回空值。该函数会被上层的 toString() 函数调用。

```
std::string MD5::bytesToHexString(const uint8_t *input, size_t length)
{
    /**
     * Convert the BYTE array to HEX string
     * @param input: the input message digest in BYTE form
     * @param length: the length of the input message digest
     */
    // If the computation is not finished, return an empty string
    if (!isFinished)
        return "";
```

```
    std::string hexString;
    for (size_t i = 0; i < length; i++)
    {
        char hex[3];
        sprintf(hex, "%02x", input[i]);
        hexString += hex;
    }
    return hexString;
}
```

(7) **字符形式显示摘要 toString()**

该函数通过调用底层的 bytesToHexString(const uint8_t *input, size_t length) 来将以字节形式存储的摘要转换为十六进制字符形式输出。更加直观，并且方便文件完整性校验的实现。

```
std::string MD5::toString()
{
    /**
     * Convert the message digest to HEX string
     * Follow the usage of getMessgeDigest()
     */
    return bytesToHexString(messageDigest, 16);
}
```

3. **摘要计算**

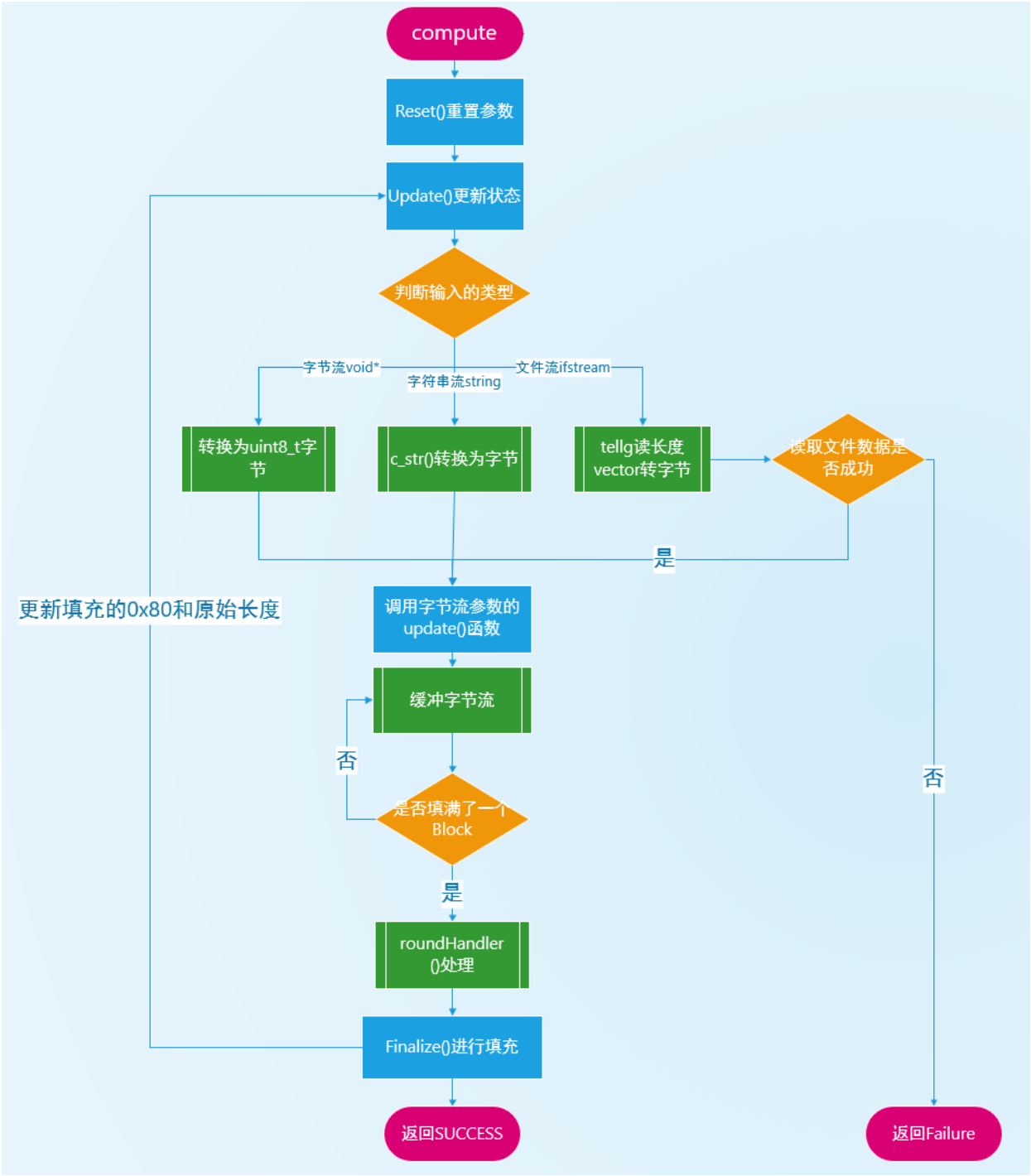

图 4.3: MD5 摘要计算实现流程图

如图4.3所示，该图清晰地展示了 **MD5 最核心的功能-摘要计算的整体流程。具体而言，主要是通过 `compute()`，`update()` 和 `roundHandler()` 和 `finalize()` 几个函数通过相互的彼此调用实现的**。

接下来也会重点介绍该部分的思路和代码。

(1) **轮处理函数 `roundHandler(const uint8_t block[64])`**

在 MD5 的实现中，`roundHandler` 函数扮演了核心角色，负责处理每个 512 位的数据块。具

体而言：

- **负责功能：**roundHandler 函数通过执行四轮特定的非线性函数变换（FF，GG，HH，II），对数据块进行加密处理。这确保了即使是微小的数据变化也会在最终的哈希值中产生大的影响，这是密码学中的雪崩效应。增加哈希算法的复杂度和安全性。
- **安全性强化：**每一轮的处理都使用了不同的非线性函数（FF，GG，HH，II），这些函数设计用于打破数据的任何可能规律，从而增加哈希函数的复杂度和抗攻击能力。
- **临时数据保护：**通过在函数结束前清除所有临时数据，roundHandler 函数还帮助保护算法免受内存泄露或侧信道攻击的影响。

详细的代码如下：

```
INT MD5::roundHandler(const uint8_t block[64])
{
    /**
     * Round Handler for each round of MD5 algorithm
     * @param block: the input block of 64 bytes, 512 bits in total
     */

    // Store the temporary values of a, b, c, d
    uint32_t a = IV[0], b = IV[1], c = IV[2], d = IV[3];
    // Define the smaller block x which has 32Bytes for 16 ones
    uint32_t x[16];
    // Decode the uint8_t block for 64 -> uint32_t for 16
    decode(block, x, 64);

    /* Round 1 */
    FF(a, b, c, d, x[0], s_table[0][0], 0xd76aa478);  /* 1 */
    FF(d, a, b, c, x[1], s_table[0][1], 0xe8c7b756);  /* 2 */
    // blablabla

    /* Round 2 */
    GG(a, b, c, d, x[1], s_table[1][0], 0xf61e2562);  /* 17 */
    GG(d, a, b, c, x[6], s_table[1][1], 0xc040b340);  /* 18 */
    // blablabla

    /* Round 3 */
    HH(a, b, c, d, x[5], s_table[2][0], 0xfffa3942);  /* 33 */
    HH(d, a, b, c, x[8], s_table[2][1], 0x8771f681);  /* 34 */
    // blablabla

    /* Round 4 */
    II(a, b, c, d, x[0], s_table[3][0], 0xf4292244);  /* 49 */
    II(d, a, b, c, x[7], s_table[3][1], 0x432aff97);  /* 50 */
```

```
    // blablabla

    // Update the initial vectors
    IV[0] += a;
    IV[1] += b;
    IV[2] += c;
    IV[3] += d;

    // Clear the temporary values
    memset(x, 0, sizeof(x));
    return SUCCESS;
}
```

(2) **MD5 状态更新函数 update(const uint8_t *input, size_t length),const void *input, const std::string &input_str 和 std::ifstream &input_file**

update 函数能够处理包括字节流、字符串、文件流在内的多种形式的数据。它负责将不同形式的数据转换为统一的字节流，以便进行后续的加密处理。

- **更新内部计数器和缓冲区**：函数开始处理数据前，首先更新内部的位计数器，记录处理的总位数。接着计算缓冲区剩余空间，决定是立即处理当前数据还是等待更多数据。
- **数据分块处理**：MD5 将数据以 64 字节（512 位）的块进行处理。状态更新函数检查缓冲区数据是否足以形成完整块，足够则调用 roundHandler 处理，不足则保留等待。
- **支持多种数据接口**：update 函数通过提供多种数据接口（字节输入、字符串、文件流）增加了 MD5 算法的灵活性和应用范围，使其能够适应不同的使用场景。
- **安全和效率的保障**：通过内部缓冲区管理和数据块处理策略，update 函数保证了数据处理的安全性和算法的效率，同时确保数据的完整性和连续性。
- **面向未来的设计**：状态更新机制的设计考虑了处理网络流数据或实时数据流的潜在需求，展示了 MD5 算法的可扩展性和适应未来技术发展的能力。

详细的代码如下：

```

INT MD5::update(const uint8_t *input, size_t length)
{
    /**
     * Update the status of the MD5 message digest for the input Bytes flow
     of given length
     * @param input: bytes flow of the input
     * @param length: the given bytes length of the input bytes flow
     */
    uint32_t i, remainingIndex, remainingLength;
    // To get the index of the bytes that are not processed yet
    and was stored in the buffer
    remainingIndex = (uint8_t)((bitsCounter[0] >> 3) & 0x3F);
```

```
14
15      // To keep track of the bits of the message that has already been processed
16      if ((bitsCounter[0] += length << 3) < (length << 3))
17          bitsCounter[1]++;
18      bitsCounter[1] += uint32_t(length >> 29);
19
20      // To get the number of bytes that are not processed yet
21      remainingLength = 64 - remainingIndex;
22
23      /**
24       * Basic Security: Perform the basic MD5 algorithm
25       */
26      if (length >= remainingLength)
27      {
28          memcpy(&bufferBlock[remainingIndex], input, remainingLength);
29          roundHandler(bufferBlock);
30          for (i = remainingLength; i + 63 < length; i += 64)
31              roundHandler(&input[i]);
32          remainingIndex = 0;
33      }
34      else
35          i = 0;
36      memcpy(&bufferBlock[remainingIndex], &input[i], length - i);
37      return SUCCESS;
38  }
39
40  // Different forms of interfaces for MD5
41  INT MD5::update(const void *input, size_t length)
42  {
43      /**
44       * Update the status of the MD5 object in bytes flow
45       * @param input: bytes flow of the input
46       * @param length: the given length of the input bytes flow
47       */
48      // Call upon on the base interface
49      update((const uint8_t *)input, length);
50      return SUCCESS;
51  }
52
53  INT MD5::update(const std::string &input_str)
54  {
55      /**
```

```
56      * Update the status of the MD5 object in string flow
57      * @param input_str: the input string
58      */
59     // Call upon on the base interface
60     update(input_str.c_str(), input_str.length());
61     return SUCCESS;
62 }
63
64 INT MD5::update(std::ifstream &input_file)
65 {
66     /**
67      * Update the status of the MD5 object in file flow
68      * @param input_file: the input file stream
69      */
70     // Read the file content
71     input_file.seekg(0, std::ios::end);
72     std::streamsize size = input_file.tellg();
73     input_file.seekg(0, std::ios::beg);
74
75     std::vector<char> buffer(size);
76     if (input_file.read(buffer.data(), size))
77         // Call upon on the base interface
78         update((const uint8_t *)buffer.data(), size);
79     else
80         return FAILURE;
81     return SUCCESS;
82 }
83
```

(3) **填充和收尾处理 `finalize()`**

`finalize` 函数是 MD5 哈希过程的最后一步，负责正确填充消息并处理最后一个数据块。执行了至关重要的任务，确保了数据的正确填充和摘要的最终处理，确保所有数据块都符合 MD5 算法的要求，通过以下步骤完成操作：

- **检查是否已完成**：如果哈希计算已标记为完成，则不执行任何操作，防止重复填充或更改摘要。
- **计算填充长度**：根据当前缓冲区中未处理的数据量，计算所需的填充长度。如果未处理的数据少于 56 字节，则填充到 56 字节；如果超过，则填充至下一个数据块的末尾。
- **执行填充**：填充始于一个 0x80 字节，后跟足够数量的 0x00 字节，直到满足 MD5 算法的长度要求。
- **添加长度信息**：在填充后追加 64 位长度字段，该字段为原始消息的长度（位数），确保消息长度是 512 位的整数倍。
- **更新摘要和清理**：使用更新的缓冲区执行最终的摘要计算，然后清理内部状态，准备算

法结束或新的哈希计算。

- **状态标记**：将内部标志 isFinished 设置为真，标记摘要计算的完成。

详细的代码如下：

```
void MD5::finalize()
{
    /**
     * Finalization for MD5 algorithm, including padding
     and process the last block
     */
    uint8_t bits[8] = {0};
    uint32_t remainingIndex, paddingLength;
    // If the computation is finished, that means there is no need to

    padding and finalization
    if (isFinished)
        return;
    // Get the index of the bytes that are not processed yet
    and was stored in the buffer
    remainingIndex = (uint8_t)((bitsCounter[0] >> 3) & 0x3F);
    /**
     * Padding: 0x80 + 0x00 * (448 mod 512) + bitsLength
     *          First to compute the padding length
     *          If the remainingIndex < 56, paddingLength = 56 -
     ↪  remainingIndex
     *          Else simply add another block of padding,

     paddingLength = 120 - remainingIndex
     */
    paddingLength
    = (remainingIndex < 56) ? (56 - remainingIndex) : (120 -
    ↪  remainingIndex);
    // Store the length of the bits, for later filling
    encode(bitsCounter, bits, 8);
    // Perform the padding
    update(padding, paddingLength);
    // Perform the bits length
    update(bits, 8);

    // Store the final message digest
    encode(IV, messageDigest, 16);
    // Set the computation to be finished
```

```
37      isFinished = true;
38      // Clear the status
39      memset(bufferBlock, 0, sizeof(bufferBlock));
40      memset(bitsCounter, 0, sizeof(bitsCounter));
41      return;
42  }
```

(4) **摘要计算函数 compute(const void *input, size_t length),const std::string &input_string 和 std::ifstream &input_file**

compute 负责启动整个计算过程的关键接口。函数为用户提供了一个简便的接口来从不同类型的数据源计算 MD5 摘要。该函数的实现高度模块化，能够处理多种输入类型，并确保了处理流程的一致性和正确性。

- **通用处理流程**：对于所有输入类型，compute 函数首先调用 reset 方法以清除任何之前的状态或残留数据。这一步是必需的，以确保每次计算开始时 MD5 对象的状态是干净的。
- **更新摘要状态**：使用 update 方法处理实际的输入数据。这一步涉及将输入数据分块，并逐块应用 MD5 的核心算法。如果在更新过程中遇到任何错误（如输入数据不可读），则返回失败状态。
- **最终化处理**：调用 finalize 方法来完成摘要的计算。这包括填充任何剩余的数据块到适当的大小，并确保摘要反映了完整的输入数据。
- **成功或失败的返回**：如果整个计算流程成功完成，compute 返回 SUCCESS；如果有任何步骤失败，则返回 FAILURE。这样的设计使得调用者可以根据返回值判断处理结果，并据此进行相应的错误处理或后续操作。

详细的代码如下：

```
1   INT MD5::compute(const void *input, size_t length)
2   {
3       /**
4        * Computing the MD5 message digest for the input bytes flow
5        * @param input: bytes flow of the input
6        * @param length: the given length of the input bytes flow
7        */
8       // Reset all the parameters in the MD5 to clear out the status
9       reset();
10      if (update(input, length) == FAILURE)
11          return FAILURE;
12      finalize();
13      return SUCCESS;
14  }
15
16  INT MD5::compute(const std::string &input_string)
```

```
{
    /**
     * Computing the MD5 message digest for the input string
     * @param input_string: the input string
     */
    // Reset all the parameters in the MD5 to clear out the status
    reset();
    if (update(input_string) == FAILURE)
        return FAILURE;
    finalize();
    return SUCCESS;
}

INT MD5::compute(std::ifstream &input_file)
{
    /**
     * Computing the MD5 message digest for the input file
     * @param input_file: the input file stream
     */
    // Reset all the parameters in the MD5 to clear out the status
    reset();
    if (update(input_file) == FAILURE)
        return FAILURE;
    finalize();
    return SUCCESS;
}
```

4.2.2 main.cpp

在 main.cpp 中，主要通过一系列辅助函数接口，调用 MD5 的算法实现，**进而达到实验要求：命令行参数交互式多功能 MD5 应用程序，可以进行摘要计算和两种文件完整性校验。除此之外，我还实现了针对字典攻击的多次散列和分割三列变换，以及参数的缺省，以实现更好的 MD5 安全性和程序交互性**。

1. (1) **帮助信息函数 help(const char *program)**
   该函数比较简单，主要用于通过程序名字 program 来输出帮助信息，包括使用方式 Usage 和每个参数选项 Options 的详细解释。具体代码如下：

```
inline void help(const char *program)
{
    /**
     * Print the help information of the program
     * @param program the name of the program
```

```
6       * @return void
7       */
8      // Help Info
9      std::cerr << "------------------------------ MD5 Help Info
       ↪  ------------------------------" << std::endl;
10
11     std::string programPath(program);
12     size_t pos = programPath.find_last_of("/\\");
13     std::string programName = programPath.substr(pos + 1);
14
15     std::cerr << "Usage: ./" << programName << " [-Option] [file path of the
       ↪  file validated] [file path of the .md5 file] [times]" << std::endl;
16     std::cerr << "Options:" << std::endl;
17     std::cerr << "  [-h] --help" << std::endl;
18     std::cerr << "  [-t] --test MD5 application" << std::endl;
19     std::cerr << "  [-c] [file path of the file computed] [times]\n"
20               << "       --compute MD5 of the given file for [times]
                 ↪  times\n"
21               << "       --If [times] equals -1, it stands for divide and
                 ↪  merge method" << std::endl;
22     std::cerr << "  [-v] [file path of the file validated]\n"
23               << "       --validate the integrality of a given file by
                 ↪  manual input MD5 value" << std::endl;
24     std::cerr << "  [-f] [file path of the file validated] [file path of the
       ↪  .md5 file]\n"
25               << "       --validate the integrality of a given file by read
                 ↪  MD5 value from .md5 file\n"
26               << "       --[file path of the .md5 file] or both the file
                 ↪  path can be empty" << std::endl;
27     std::cerr << "  [-fm] [file path of the file validated] [file path of
       ↪  the .md5 file]\n"
28               << "       --validate the integrality of a given file by read
                 ↪  MD5 value from .md5 file\n"
29               << "       --[file path of the file validated] or both the
                 ↪  file path can be empty" << std::endl;
30
31     std::cerr
32         <<
           ↪  "--------------------------------------------------------------------------------
           ↪  << std::endl;
33     return;
```

```
}
```

(2) **测试正确性函数 `inline void test()`**

该函数同样比较简单，主要通过**给定的 MD5 官方文档的一系列测试样例进行测试并保存测试结果到 `tmp` 文件目录下**。

通过 `md5.compute()` 函数对每个样例进行摘要计算后，使用 `md5.toString()` 以十六进制形式进行输出。最后写入到 `tmp` 目录下的多个.md5 后缀文件中。对应关系就是代码中的样例顺序。详细的代码如下：

```
inline void test()
{
    /**
     * Test the MD5 application using offical test vectors to verify the correctness of the MD5 algorithm
     * @return void
     */

    std::cout << "------------------------------ Testing MD5 Correctness ------------------------------" << std::endl;
    std::cout << "[INFO] The following examples can be found in MD5 Offical Doc:RFC1321." << std::endl;
    std::vector<std::string> testStrings = {"", "a", "abc", "message digest", "abcdefghijklmnopqrstuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789", "12345678901234567890123456789012345678901234567890123456789012345678901234567890"};
    MD5 md5;
    for (int i = 0; i < testStrings.size(); i++)
    {
        std::string testString = testStrings[i];
        md5.compute(testString);
        std::string md5Value = md5.toString();
        std::cout << "MD5(\"" << testString << "\") = " << md5Value << std::endl;
        std::string md5FileName = "../tmp/RFC-" + std::to_string(i + 1) + ".md5";
        std::ofstream md5File(md5FileName);
        md5File << md5Value;
        md5File.close();
    }
```

```
28      std::cout <<
        ↪  "--------------------------------------------------------------------------------
        ↪  << std::endl;
29      return;
30  }
```

(3) **摘要计算函数 `computeMD5(const char *filePath, INT times = 1)`**

此函数设计用来计算由 `filePath` 指定的文件的 MD5 哈希。MD5 哈希值，并提供可选的安全增强功能以抵御针对 MD5 的字典攻击，根据参数 `times` 的不同，可以选择的方法有多次散列变换和分割散列变化。

- **错误处理**：函数首先检查是否能成功打开文件。如果打开失败，则报告错误并以 `FAILURE` 状态终止，确保不会对无法访问的文件进行进一步操作。
- **分割散列变换**：如果 `times` 设置为 -1，函数使用分割合并策略，包括：
  - 计算整个文件的 MD5 哈希。
  - 将哈希分成两部分。
  - 分别重新计算每部分的 MD5。
  - 将这两个新的哈希值合并成一个最终的 MD5 值。

  这种方法通过改变标准哈希程序，增加了安全性，使哈希结果更不可预测，更能抵抗某些攻击。
- **多次散列变化**：如果 `times` 大于一，哈希将被多次计算，每次都在上一次的哈希结果上进行。这种迭代哈希是增加安全性的另一种措施，它改变了哈希结果并需要更多努力来逆向工程。
- **输出**：不论使用哪种方法，最终计算出的 MD5 值都会显示出来，提供了操作结果的清晰反馈。

通过提供改变哈希计算过程的选项，`computeMD5` 函数增加了额外的安全层。这些选项有助于缓解简单哈希计算的风险，并根据安全需求提供适应性。

详细的代码如下：

```
1   inline INT computeMD5(const char *filePath, INT times = 1)
2   {
3       /**
4        * Compute the MD5 value of the given file
5        * @param filePath the file path of the file to be computed
6        * @param times In order to defend against the dictionary attack
7        *              The times for the MD5 algorithm to be computed
8        *              If times = -1, it stands for divide and merge method
9        * @return INT: 0 for SUCCESS, 1 for FAILURE
10       */
11      std::cout << "----------------------------- Compute MD5 for
        ↪  File------------------------------" << std::endl;
12      MD5 md5;
```

```
13      std::ifstream file(filePath, std::ios::binary);
14      std::string leftMD5, rightMD5, encryptedLeftMD5, encryptedRightMD5,
        ↪  finalMD5;
15      // To make sure if the file is opened successfully
16      if (!file)
17      {
18          std::cerr << "[ERROR] Cannot open the file: \"" << filePath << "\""
19          << std::endl;
20          std::cout <<
            ↪  "--------------------------------------------------------------------------
            ↪  << std::endl;
21          return FAILURE;
22      }
23      if (times == -1)
24      {
25          // Divide and Merge Method
26          std::cout << "[INFO] Using Divide and Merge Method to compute the
            ↪  MD5 value of file(\"" << filePath << "\")" << std::endl;
27          md5.compute(file);
28          // Divide the md5 value into two separate parts
29          leftMD5 = md5.toString().substr(0, 16);
30          rightMD5 = md5.toString().substr(16, 16);
31          // Compute the MD5 value of the two 16 bytes separately
32          md5.compute(leftMD5);
33          encryptedLeftMD5 = md5.toString();
34          md5.compute(rightMD5);
35          encryptedRightMD5 = md5.toString();
36          // Merge the two MD5 values
37          md5.compute(encryptedLeftMD5 + encryptedRightMD5);
38          finalMD5 = md5.toString();
39          std::cout << "The MD5 value of file(\"" << filePath << "\") is:" <<
            ↪  std::endl;
40          std::cout << finalMD5 << std::endl;
41          std::cout <<
            ↪  "--------------------------------------------------------------------------
            ↪  << std::endl;
42          return SUCCESS;
43      }
44      else
45      {
46          // Compute Multiple Times Method
47          std::cout << "[INFO] Computing the MD5 value of file(\"" << filePath
```

```
48          << "\") for " << times << " time(times)." << std::endl;
49          md5.compute(file);
50          times--;
51          while (times > 0)
52          {
53              // If it only computes once, directly output the MD5 value, it
                ↪  won't enter the loop
54              md5.compute(md5.toString());
55              times--;
56          }
57
58          std::cout << "The MD5 value of file(\"" << filePath << "\") is:" <<
            ↪  std::endl;
59          std::cout << md5.toString() << std::endl;
60          std::cout <<
            ↪  "--------------------------------------------------------------------------------
            ↪  << std::endl;
61          return SUCCESS;
62      }
63  }
```

(4) **文件完整性校验函数** `validate(const char *validatedFilePath = DEFAULT_FILE_NAME, const char *md5FilePath = DEFAULT_MD5_FILE, INT Approach = MANUAL)`: `validate` 函数是为了验证文件完整性而设计的关键功能，通过比较文件的 MD5 值来确定文件是否被篡改。**此函数支持两种校验方式：手动输入 MD5 值或自动从指定的 MD5 文件读取**。这使得函数能够灵活应对不同的使用场景。**同时也支持缺省的参数处理**。

- **手动验证方式**：用户需手动输入预期的 MD5 值，函数将此值与文件当前的 MD5 值进行比较。这种方式适用于环境中没有预存 MD5 值，或需要用户即时验证的场合。
- **自动验证方式**：自动从一个指定的.md5 文件中读取 MD5 值，然后将这个值与文件当前的 MD5 值进行比较。这种方法适用于批量处理或当预存了 MD5 值时。
- **错误处理**：验证过程中首先检查是否能成功打开目标文件及 MD5 文件。任何文件打开失败都将导致函数返回失败状态。
- **MD5 计算与比较**：使用 `MD5` 类的实例计算文件的 MD5 值，然后根据不同的验证方式进行比较。如果 MD5 值匹配，则确认文件未被篡改；如果不匹配，则认为文件可能已被修改。
- **缺省参数处理**：如果未提供 `validatedFilePath` 或 `md5FilePath` 参数，函数将使用默认值 `DEFAULT_FILE_NAME` 和 `DEFAULT_MD5_FILE`。这允许用户在不指定具体文件路径的情况下调用函数，简化了日常操作，特别是在开发和测试阶段。
- **结果输出**：无论验证结果如何，都会清晰地在控制台输出验证结果，为用户提供直接的反馈。

详细的代码如下：

```cpp
inline INT validate(const char *validatedFilePath = DEFAULT_FILE_NAME, const char *md5FilePath = DEFAULT_MD5_FILE, INT Approach = MANUAL)
{
    /**
     * Validate the integrality of the given file by manual or automatically input MD5 value
     * @param validatedFilePath the file path of the file to be validated
     * @param md5FilePath the file path of the .md5 file
     * @return INT: 0 for SUCCESS, 1 for FAILURE
     */
    MD5 md5;
    std::string computedMD5Value;
    std::string inputMD5Value;
    std::string readMD5Value;
    // Open the tested file
    std::ifstream validatedFile(validatedFilePath, std::ios::binary);
    // Open the .md5 file
    std::ifstream md5File(md5FilePath, std::ios::binary);
    switch (Approach)
    {
    case MANUAL:
        // Manual validation approach
        // Input the MD5 value of the file manually
        std::cout << "----------------------------- Validate File Integrality Manually------------------------------" << std::endl;
        std::cout << "Please input the MD5 value of the file(\"" << validatedFilePath << "\"):" << std::endl;
        std::cin >> inputMD5Value;

        // To make sure if the file is opened successfully
        if (!validatedFile)
        {
            std::cerr << "[ERROR] Cannot open the validated file: \"" << validatedFilePath << "\"" << std::endl;
            std::cout << "-------------------------------------------------------------------------------" << std::endl;
            return FAILURE;
        }

```

```
34          // Compute the MD5 value of the file
35          md5.reset();
36          md5.compute(validatedFile);
37          computedMD5Value = md5.toString();
38
39          // Compare the input MD5 value with the computed MD5 value
40          if (inputMD5Value == computedMD5Value)
41          {
42              std::cout << "[INFO] OK!The file(\"" << validatedFilePath <<
                ↪  "\") is integrality." << std::endl;
43              std::cout <<
                ↪  "--------------------------------------------------------------------------
                ↪  << std::endl;
44              return SUCCESS;
45          }
46          else
47          {
48              std::cout << "[INFO] Match Error! The file (\"" <<
                ↪  validatedFilePath << "\") has been modified!" << std::endl;
49              std::cout << "[INFO] It is not integrality." << std::endl;
50              std::cout <<
                ↪  "--------------------------------------------------------------------------
                ↪  << std::endl;
51              return FAILURE;
52          }
53          break;
54      case AUTO:
55          // Automatic validation approach
56          std::cout << "---------------------------- Validate File
            ↪  Integrality Automatically-----------------------------" <<
            ↪  std::endl;
57
58          // To make sure if the validated file is opened successfully
59          if (!validatedFile)
60          {
61              std::cerr << "[ERROR] Cannot open the validated file: \"" <<
                ↪  validatedFilePath << "\"" << std::endl;
62              std::cout <<
                ↪  "--------------------------------------------------------------------------
                ↪  << std::endl;
63              return FAILURE;
64          }
```

```
65          // To make sure if the .md5 file is opened successfully
66          if (!md5File)
67          {
68              std::cerr << "[ERROR] Cannot open the MD5 file: \"" <<
                ↪  md5FilePath << "\"" << std::endl;
69              std::cout <<
                ↪  "--------------------------------------------------------------------------
                ↪  << std::endl;
70              return FAILURE;
71          }
72
73          // Compute the MD5 value of the file
74
75          md5.reset();
76          md5.compute(validatedFile);
77          computedMD5Value = md5.toString();
78
79          // Read the MD5 value from the .md5 file
80          getline(md5File, readMD5Value);
81
82          // Compare the read MD5 value with the computed MD5 value
83          if (readMD5Value == computedMD5Value)
84          {
85              std::cout << "[INFO] OK!The file(\"" << validatedFilePath <<
                ↪  "\") is integrality." << std::endl;
86              std::cout <<
                ↪  "--------------------------------------------------------------------------
                ↪  << std::endl;
87              return SUCCESS;
88          }
89          else
90          {
91              std::cout << "[INFO] Match Error! The file (\"" <<
                ↪  validatedFilePath << "\") has been modified!" << std::endl;
92              std::cout << "[INFO] It is not integrality." << std::endl;
93              std::cout <<
                ↪  "--------------------------------------------------------------------------
                ↪  << std::endl;
94              return FAILURE;
95          }
96          break;
97      default:
```

```
98          // If the approach is not defined, print the help info to instruct
            ↪  the user
99          std::cout <<
            ↪  "-------------------------------------------------------------------------
            ↪  << std::endl;
100         return FAILURE;
101         break;
102     }
103 }
104
```

2. **主函数 main()**

在主函数 main() 中，主要做的就是通过 argc 确认参数数量，argv 获取参数具体值，由此解析参数。根据选项参数的取值在 {-h, -t, -c, -v, -f, -fm} 中的哪个来进行判断，进一步调用 help, test, computeMD5 和 validate 四个函数。

```
1  int main(int argc, char *argv[])
2  {
3      // Arguments Parsing
4      if (argc >= 2)
5      {
6      /**
7           * Normally parsing the arguments here
8           * (1) -h : help
9           * (2) -t : test MD5 application
10          * (3) -c : compute MD5 of the given file for [times] times
11          * (4) -v : validate the integrality of a given file by manual input
            ↪  MD5 value
12          * (5) -f : validate the integrality of a given file
13          by read MD5 value from .md5 file,
14          can leave the .md5 file path or both file path empty
15          * (6) -fm : validate the integrality of a given file
16          by read MD5 value from .md5 file,
17          can leave the tested file path or both file path empty
18     */
19          std::string optionArg = argv[1];
20          if (optionArg == pHelp)
21          {
22              // If the option is help, print the help information
23              help(argv[0]);
24          }
25          else if (optionArg == pTest)
```

```
26          {
27              // If the option is test, test the MD5 application
28              test();
29          }
30          else if (optionArg == pCompute)
31          {
32              if (argc == 4)
33              {
34                  // If the option is compute, compute the MD5 value of the given
                    ↪ file
35                  if (computeMD5(argv[2], std::stoi(argv[3])) == FAILURE)
36                      help(argv[0]);
37              }
38              else
39              {
40                  // If the option is compute,
41                  // But the number of arguments is not correct, print the help
                    ↪ info to instruct the user
42                  help(argv[0]);
43                  return 1;
44              }
45          }
46          else if (optionArg == pValidate)
47          {
48              if (argc == 3)
49              {
50                  if (validate(argv[2]) == FAILURE)
51                      help(argv[0]);
52              }
53              else
54              {
55                  std::cout << "[INFO] Using the default file to be validated:\""
                    ↪ << DEFAULT_FILE_NAME << "\"" << std::endl;
56                  if (validate(DEFAULT_FILE_NAME) == FAILURE)
57                      help(argv[0]);
58              }
59          }
60          else if (optionArg == pFile)
61          {
62              /**
63               * If the option is file, validate the integrality of the given
                  ↪ file by read MD5 value from .md5 file
```

```
64              * (1) -f [file path of the file validated] [file path of the .md5
                ↪  file]
65              * (2) -f [file path of the file validated]
66              * (3) -f
67              */
68              if (argc == 4)
69              {
70                  if (validate(argv[2], argv[3], AUTO) == FAILURE)
71                      help(argv[0]);
72              }
73              else if (argc == 3)
74              {
75                  std::cout << "[INFO] Using the default .md5 file to read
                    ↪  value:\"" << DEFAULT_MD5_FILE << std::endl;
76                  if (validate(argv[2], DEFAULT_MD5_FILE, AUTO) == FAILURE)
77                      help(argv[0]);
78              }
79              else if (argc == 2)
80              {
81                  std::cout << "[INFO] Using the default file to be validated:\""
                    ↪  << DEFAULT_FILE_NAME << "\" and the default MD5 file:\"" <<
                    ↪  DEFAULT_MD5_FILE << "\"" << std::endl;
82                  if (validate(DEFAULT_FILE_NAME, DEFAULT_MD5_FILE, AUTO) ==
                    ↪  FAILURE)
83                      help(argv[0]);
84              }
85              else
86              {
87                  // If the option is file,
88                  // But the number of arguments is > 4, print the help info to
                    ↪  instruct the user
89                  help(argv[0]);
90                  return 1;
91              }
92          }
93          else if (optionArg == pFileMD5)
94          {
95              /**
96               * If the option is fileMD5, validate the integrality of the given
                 ↪  file by read MD5 value from .md5 file
97               * (1) -fm [file path of the file validated] [file path of the .md5
                 ↪  file]
```

```
98              * (2) -fm [file path of the .md5 file]
99              * (3) -fm
100             */
101             if (argc == 4)
102             {
103                 if (validate(argv[2], argv[3], AUTO) == FAILURE)
104                     help(argv[0]);
105             }
106             else if (argc == 3)
107             {
108                 std::cout << "[INFO] Using the default file to be validated:\""
                    ↪  << DEFAULT_FILE_NAME << "\"" << std::endl;
109                 if (validate(DEFAULT_FILE_NAME, argv[2], AUTO) == FAILURE)
110                     help(argv[0]);
111             }
112             else if (argc == 2)
113             {
114                 std::cout << "[INFO] Using the default file to be validated:\""
                    ↪  << DEFAULT_FILE_NAME << "\" and the default MD5 file:\"" <<
                    ↪  DEFAULT_MD5_FILE << "\"" << std::endl;
115                 if (validate(DEFAULT_FILE_NAME, DEFAULT_MD5_FILE, AUTO) ==
                    ↪  FAILURE)
116                     help(argv[0]);
117             }
118             else
119             {
120                 // If the option is file,
121                 // But the number of arguments is > 4, print the help info to
                    ↪  instruct the user
122                 help(argv[0]);
123                 return 1;
124             }
125         }
126         else
127         {
128             // If the prasing option is not defined, print the help info to
                ↪  instruct the user
129             help(argv[0]);
130             return 1;
131         }
132     }
133     else
```

```
134     {
135         // If the number of arguments is less than 2,
136         print the help info to instruct the user
137         help(argv[0]);
138         return 1;
139     }
140     return 0;
141 }
```

5 实验结果

5.1 实验结果展示

最后在此对交互式 MD5 程序的全部功能进行展示，将会依次展示显示帮助信息（-h），测试 MD5 算法正确性（-t），计算指定文件 MD5 值（-c）以及两种变换算法，手动文件完整性校验（-v），自动完整性校验（-f 和-fm）。其中也会通过缺省参数的方式测试其交互功能。

5.1.1 显示帮助信息（-h）

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -h
------------------------------ MD5 Help Info ------------------------------
Usage: ./MD5 [-Option] [file path of the file validated] [file path of the .md5 file] [times]
Options:
  [-h] --help
  [-t] --test MD5 application
  [-c] [file path of the file computed] [times]
       --compute MD5 of the given file for [times] times
       --If [times] equals -1, it stands for divide and merge method
  [-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
       --[file path of the .md5 file] or both the file path can be empty
  [-fm] [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
       --[file path of the file validated] or both the file path can be empty
---------------------------------------------------------------------------
```

图 5.4: 显示帮助信息（-h）演示

如图5.4所示，可以看到通过在 MD5 程序后通过-h 选项，可以显示非常详细的帮助信息。包括程序使用的具体 Usage 格式和 Options 每个选项的含义解释。这样的帮助信息对于初次接触该程序的用户来说非常有用，可以快速了解程序的功能和使用方法。

5.1.2 测试 MD5 算法正确性（-t）

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -t
----------------------------- Testing MD5 Correctness -----------------------------
[INFO] The following examples can be found in MD5 Offical Doc:RFC1321.
MD5("") = d41d8cd98f00b204e9800998ecf8427e
MD5("a") = 0cc175b9c0f1b6a831c399e269772661
MD5("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") = d174ab98d277d9f5a5611c2c9f419d9f
MD5("12345678901234567890123456789012345678901234567890123456789012345678901234567890") = 57edf4a22be3c955ac49da2e2107b67a
-------------------------------------------------------------------------------
```

图 5.5: 测试 MD5 算法正确性（-t）演示

如图5.5所示，可以看到在 MD5 程序后通过-t 选项，可以看到提示“下面的样例来自 MD5 的官方文档”。并且下面依次列出来了**对于以下样例的测试结果：**

- MD5(””) = d41d8cd98f00b204e9800998ecf8427e
- MD5(”a”) = 0cc175b9c0f1b6a831c399e269772661
- MD5(”abc”) = 900150983cd24fb0d6963f7d28e17f72
- MD5(”message digest”) = f96b697d7cb7938d525a2f31aaf161d0
- MD5(”abcdefghijklmnopqrstuvwxyz”) = c3fcd3d76192e4007dfb496cca67e13b
- MD5(”ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789”) = d174ab98d277d9f5a5611c2c9f419d9f
- MD5(”12345678901234567890123456789012345678901234567890123456789012345678901234567890”) = 57edf4a22be3c955ac49da2e2107b67a

以上结果均与**实验报告中与官方文档的测试 MD5 结果相同，答案完全正确。证明了我们实现的 MD5 算法的正确性与可靠性**。

5.1.3 计算指定文件 MD5 值（-c）

图 5.6: 测试文件展示

首先如图5.6所示，此时与我们的可执行程序 MD5 同一目录下有两个文件：**test 文件，实际上内容是本次实验的实验手册，没有.pdf 后缀。另一个是.md5 文件**。后面将会使用该手册文件进行 MD5 哈希值摘要的计算。

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ md5sum test
44d099b6e6afee22adee248ab6d60bb8  test
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -c test 1
------------------------------ Compute MD5 for File------------------------------
[INFO] Computing the MD5 value of file("test") for 1 time(times).
The MD5 value of file("test") is:
44d099b6e6afee22adee248ab6d60bb8
--------------------------------------------------------------------------------
```

图 5.7: 单次计算摘要结果与 md5sum 结果对比

如图5.7所示，可以看到通过命令`./MD5 -c test 1`指定对和 MD5 可执行文件下的 test 文件进行一次 MD5 摘要计算的结果为 **44d099b6e6afee22adee248ab6d60bb8。和使用 Linux 的 `md5sum test` 的结果完全一致，进一步证明了我们的 MD5 算法的正确性与可靠性**。

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -c test 5
------------------------------ Compute MD5 for File------------------------------
[INFO] Computing the MD5 value of file("test") for 5 time(times).
The MD5 value of file("test") is:
1bfd1490d57a3e4b74ba577fcc881bed
--------------------------------------------------------------------------------
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -c test -1
------------------------------ Compute MD5 for File------------------------------
[INFO] Using Divide and Merge Method to compute the MD5 value of file("test")
The MD5 value of file("test") is:
093f6366b7e9aa486e4e7998ae702995
--------------------------------------------------------------------------------
```

图 5.8: 多次散列变换和分割散列变换

如图5.8所示，可以看到：

- **多次散列变换：** 通过命令`./MD5 -c test 5`对和 MD5 可执行文件下的 test 文件进行 5 次 MD5 摘要计算，计算结果为 1bfd1490d57a3e4b74ba577fcc881bed。

  进行多次散列（在此例中为五次）可以有效地增加破解的难度。每次散列都基于前一次的输出，这样连续的变换可以帮助防御简单的暴力破解和彩虹表攻击。连续散列的结果与单次散列相比，更不容易通过预先计算的散列值库来反向求解原始数据，因为攻击者需要针对每一次的散列结果都进行破解尝试。

- **分割散列变换：** 通过命令`./MD5 -c test -1`对和 MD5 可执行文件下的 test 文件进行了分割散列变换，计算结果为 093f6366b7e9aa486e4e7998ae702995。

  分割散列方法通过将数据分成多个部分单独计算 MD5，然后再对这些结果进行一次最终的 MD5 计算，以此来提高安全性。这种方法的优点在于它改变了攻击者预期的数据结构，使得攻击更加困难。分割处理后的数据不容易与已知的任何预计算彩虹表等匹配的字典攻击，因此提高了抗攻击的能力。

这些增强的 MD5 变换方法提供了额外的安全层次，**使得我实现的 MD5 散列程序可以根据用户的不同需要，针对性地选择散列算法，更适用于需要更高安全性的现代场景。证明了班次实验实现的的 MD5 程序的安全性，鲁棒性和灵活性较强。**

5.1.4 手动文件完整性校验（-v）

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -v test
---------------------------- Validate File Integrality Manually----------------------------
Please input the MD5 value of the file("test"):
44d099b6e6afee22adee248ab6d60bb8
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -v test
---------------------------- Validate File Integrality Manually----------------------------
Please input the MD5 value of the file("test"):
1bfd1490d57a3e4b74ba577fcc881bed
[INFO] Match Error! The file ("test") has been modified!
[INFO] It is not integrality.
--------------------------------------------------------------------------------
------------------------------ MD5 Help Info ------------------------------
Usage: ./MD5 [-Option] [file path of the file validated] [file path of the .md5 file] [times]
Options:
  [-h] --help
  [-t] --test MD5 application
  [-c] [file path of the file computed] [times]
       --compute MD5 of the given file for [times] times
       --If [times] equals -1, it stands for divide and merge method
  [-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
       --[file path of the .md5 file] or both the file path can be empty
  [-fm] [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
       --[file path of the file validated] or both the file path can be empty
--------------------------------------------------------------------------------
```

图 5.9: 手动文件完整性校验（-v）演示

如图5.9所示，可以看到通过命令`./MD5 -v test`。会启动手动文件完整性校验功能，此时如果输入正确的 `test` 文件的 MD5 摘要值即 44d099b6e6afee22adee248ab6d60bb8，则也会显示文件完整的验证结果。若输入的不是正确的 `test` 文件的 MD5 摘要值，则会显示 `test` 文件并不完整，已被修改的提示信息来说明验证结果。**这进一步证明了 MD5 算法实现的正确性与可靠性**。

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -v
[INFO] Using the default file to be validated:"test"
---------------------------- Validate File Integrality Manually----------------------------
Please input the MD5 value of the file("test"):
44d099b6e6afee22adee248ab6d60bb8
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
```

图 5.10: 缺省测试文件路径的手动文件完整性校验（-v）演示

如图5.10所示，若通过命令`./MD5 -v` 缺省不给出文件路径，会自动使用和 MD5 程序可执行文件同目录下的 `test` 文件进行验证。

5.1.5 自动完整性校验（-f）

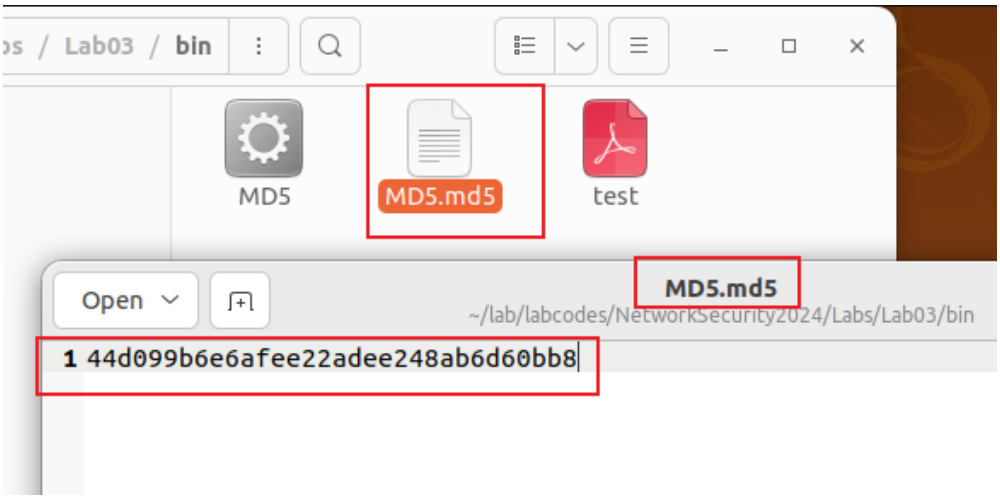

图 5.11: .md5 文件展示

如图5.11所示，仍然使用和 MD5 可执行程序同目录下的 `test` 测试文件和 `MD5.md5` 摘要存储文件。其中可以看到 `MD5.md5` 摘要存储文件中的内容为 44d099b6e6afee22adee248ab6d60bb8。就是正确的 `test` 文件的 MD5 摘要数值。

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -f test MD5.md5
---------------------------- Validate File Integrality Automatically----------------------------
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -f test
[INFO] Using the default .md5 file to read value:"MD5.md5
---------------------------- Validate File Integrality Automatically----------------------------
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -f
[INFO] Using the default file to be validated:"test" and the default MD5 file:"MD5.md5"
---------------------------- Validate File Integrality Automatically----------------------------
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
```

图 5.12: 自动完整性校验（-f）演示

接下来分别输入三组不同的命令。如图5.12所示，可以看到：

- 不缺省文件路径：若通过命令`./MD5 -f test MD5.md5`，可以看到通过读取 `MD5.md5` 中的 MD5 摘要值和程序自动计算的结果对比，是完全相同的，说明了测试文件是完整的。这和我们的预料结果一样。进一步证明了程序的正确性与可靠性。
- 缺省.md5 文件路径：若通过命令`./MD5 -f test`，可以看到通过缺省.md5 文件路径。会直接使用 `MD5.md5` 作为默认.md5 文件读取 MD5 摘要值。同样结果也是文件完整。
- 同时缺省 test 和.md5 文件路径：若通过命令`./MD5 -f`，可以看到通过同时缺省测试文件和.md5 文件路径。会直接使用 `test` 作为默认测试文件，`MD5.md5` 作为默认.md5 文件。同样结果也是文件完整。

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -f MD5.md5 test
---------------------------- Validate File Integrality Automatically-----------------------------
[INFO] Match Error! The file ("MD5.md5") has been modified!
[INFO] It is not integrality.
--------------------------------------------------------------------------------
------------------------------ MD5 Help Info ------------------------------
Usage: ./MD5 [-Option] [file path of the file validated] [file path of the .md5 file] [times]
Options:
  [-h] --help
  [-t] --test MD5 application
  [-c] [file path of the file computed] [times]
       --compute MD5 of the given file for [times] times
       --If [times] equals -1, it stands for divide and merge method
  [-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
       --[file path of the .md5 file] or both the file path can be empty
  [-fm] [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
       --[file path of the file validated] or both the file path can be empty
--------------------------------------------------------------------------------
```

图 5.13: 自动完整性校验（-f）（故意错误版）演示

如图5.13所示，可以看到通过命令./MD5 -f MD5.md5 test，即故意换掉被测文件和.md5 文件路径，将二者交换。改为让 MD5.md5 作为被测文件，test 作为读取 MD5 摘要值的文件。可以看到此时输出了文件不完整的结果，这和我们预料的一样。进一步证明了程序的正确性和可靠性。

5.1.6 自动完整新校验（-fm）

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -fm test MD5.md5
---------------------------- Validate File Integrality Automatically-----------------------------
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -fm MD5.md5
[INFO] Using the default file to be validated:"test"
---------------------------- Validate File Integrality Automatically-----------------------------
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -fm
[INFO] Using the default file to be validated:"test" and the default MD5 file:"MD5.md5"
---------------------------- Validate File Integrality Automatically-----------------------------
[INFO] OK!The file("test") is integrality.
--------------------------------------------------------------------------------
```

图 5.14: 自动完整新校验（-fm）演示

如图5.14所示，和自动完整新校验（-f）功能类似，还是通过缺省不同的参数都达到了我们预期的效果。不过唯一不同的点是第二组命令是./MD5 -f MD5.md5 缺省的是被测文件路径，此时直接使用 test 作为测试文件。

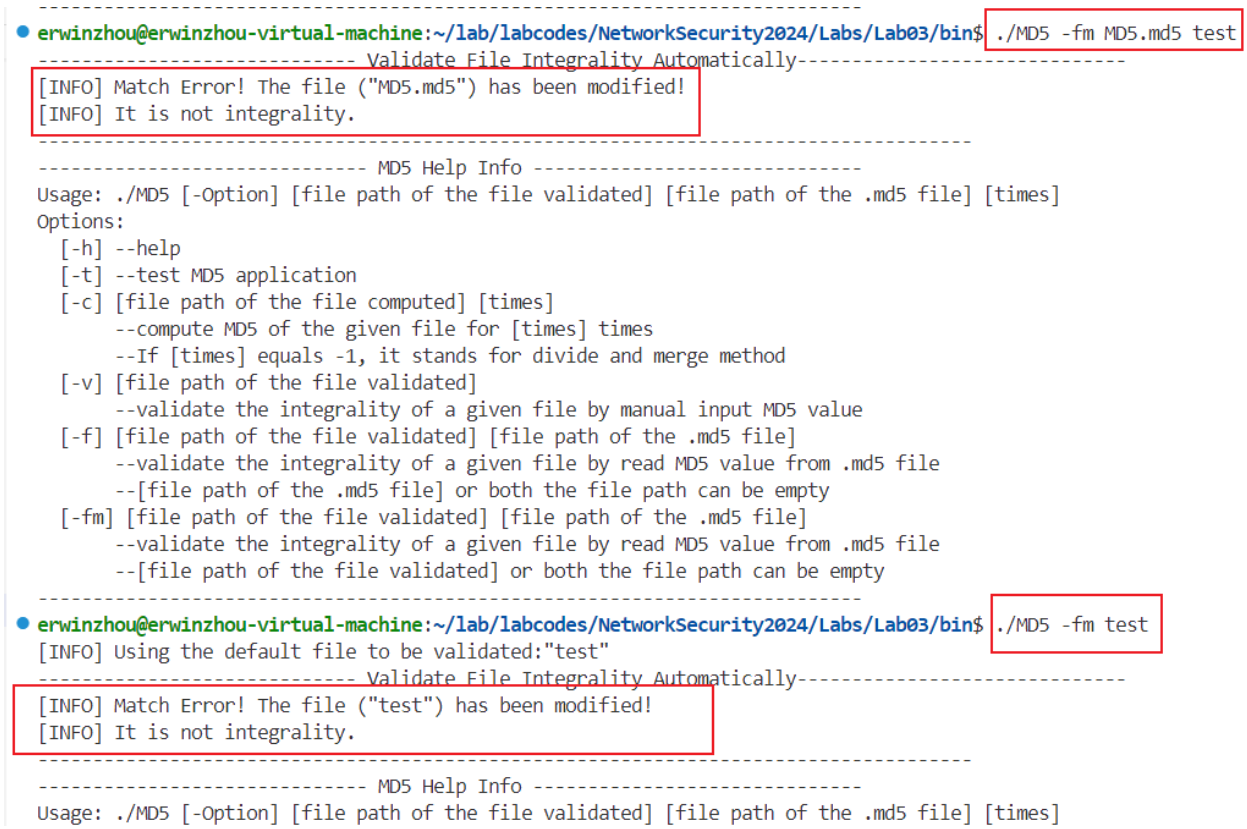

图 5.15: 自动完整新校验（-fm）（故意错误版）演示

如图5.15所示，也是故意交换了被测文件和存储 MD5 值的文件，也是出现了文件不完整的预期结果。除此之外，还尝试了只提供 `test` 文件路径。此时也出现了文件不完整的验证结果。这是因为`-fm`选项只允许缺省被测文件，若只提供了一个文件路径，会默认视作存储 MD5 值的文件。因此相当于将 `test` 同时作为了被测文件和存储 MD5 值的文件，自然结果会不完整。

到此所有结果与功能演示完毕，可以看到，实验完成度非常高，整体来说较为成功。

6 实验遇到的问题及其解决方法

6.1 MD5

6.1.1 MD5 的字节序问题

- **遇到的问题：**在小端字节序的系统中，低位字节位于内存的起始地址；而在大端字节序的系统中，高位字节首先被存储。

  MD5 算法在设计时考虑到了跨平台的兼容性，**特别规定了在处理字节数据时必须采用小端字节序**。字节序问题源于不同计算机架构对字节的存储方式不同。例如，x86 架构采用小端模式存储，而某些配置下的 PowerPC 或 ARM 架构使用大端模式。如果直接在一个采用大端字节序的系统上进行 MD5 计算，而不对数据进行适当处理，将会得到错误的哈希结果。

- **解决方法：**为了解决由于字节序差异而可能出现的问题，**本实验中的 MD5 实现特别引入了 `encode` 和 `decode` 函数**。这两个函数在 MD5 算法处理数据前后进行字节序的转换，确保了无论在何种字节序的系统上执行，MD5 哈希的计算都能保持一致。

- **字节序转换的实现**：
  - `encode` 函数：在数据处理完毕后将内部使用的 DWORD（32 位无符号整数）格式转换为 BYTE（8 位无符号整数）格式，确保按照小端方式输出每个字的低位字节在前。
  - `decode` 函数：在数据处理前将输入的 BYTE 数据转换为内部处理所需的 DWORD 格式，同样确保按照小端方式将低位字节排在前。

  具体代码详见实验步骤部分。**这样的处理不仅符合 MD5 的规范要求，也消除了不同平台间的字节序差异，从而确保了哈希值的一致性和准确性**。

6.1.2 MD5 的字典攻击

- **遇到的问题：MD5 算法天生容易受到字典攻击**。字典攻击涉及预先收集大量的明文与其对应的 MD5 散列值，并存储于数据库中。在需要破解特定 MD5 散列时，可以直接查询数据库找到对应的明文。
- **解决方法：**本次实验通过在命令行交互程序的计算 MD5 摘要部分通过提供 `[times]` 选项，具体代码详见实验步骤部分。让用户能够根据不同的安全需求，选择**多次散列变换和分割散列变换两种变换算法，由此提高 MD5 安全性以及实现的程序的可靠性和灵活性**。

6.2 Command Line Interface

6.2.1 校验文件是否存在

- **遇到的问题：**在我最开始实现交互式命令行程序时候，直接使用 `filePath` 对应的 `std::ifstream` 流，导致有时候出现文件即使不存在也能计算出结果的情况，相当诡异。实际上在读取文件时，可能会面临多种问题，如文件不存在、权限不足或文件损坏等。
- **解决方法：**通过两个 `def.hpp` 中的宏常量 `SUCCESS` 和 `FAILURE` 记录程序可能成功或者失败的运行状态。然后在每次进行 `std::ifstream` 的打开之前，都先检查其状态以检测文件是否存在。不存在则返回 `FAILURE`，即使停止程序，并输出错误信息和帮助信息提示用户。

示例代码如下：

```
1     // To make sure if the file is opened successfully
2     if (!validatedFile)
3     {
4         std::cerr << "[ERROR] Cannot open the validated file: \"" <<
          ↪  validatedFilePath << "\"" << std::endl;
5         std::cout <<
          ↪  "--------------------------------------------------------------------------------
          ↪  << std::endl;
6         return FAILURE;
7     }
```

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/NetworkSecurity2024/Labs/Lab03/bin$ ./MD5 -fm 我不存在啦啦啦
[INFO] Using the default file to be validated:"test"
---------------------------- Validate File Integrality Automatically-----------------------------
[ERROR] Cannot open the MD5 file: "我不存在啦啦啦"
--------------------------------------------------------------------------------
------------------------------ MD5 Help Info ------------------------------
Usage: ./MD5 [-Option] [file path of the file validated] [file path of the .md5 file] [times]
Options:
  [-h] --help
  [-t] --test MD5 application
  [-c] [file path of the file computed] [times]
       --compute MD5 of the given file for [times] times
       --If [times] equals -1, it stands for divide and merge method
  [-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
       --[file path of the .md5 file] or both the file path can be empty
  [-fm] [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
       --[file path of the file validated] or both the file path can be empty
--------------------------------------------------------------------------------
```

图 6.16: 打开不存在文件报错示例

6.2.2 缺省参数

- **遇到的问题**：实验手册中的命令行参数交互比较单独，不允许有参数的省略，缺少了程序的灵活性，实际上大部分可交互性的命令行程序都提供了缺省参数的默认选项。
- **解决方法**：通过首先在 `def.hpp` 中的宏常量 `DEFAULT_FILE_NAME` 和 `DEFAULT_MD5_FILE` 指定使用 `test` 作为默认测试文件，`MD5.md5` 作为默认.md5 文件。

  然后首先在 `validate` 函数中指定默认形参。

```
1  inline INT validate(const char *validatedFilePath = DEFAULT_FILE_NAME,
   ↪  const char *md5FilePath = DEFAULT_MD5_FILE, INT Approach = MANUAL)
2  /* blablabla */
```

  另外在 `main` 函数中也通过检查函数参数个数的方式进行默认路径的选择和设置。

```
1  else if (optionArg == pFile)
2  {
3      /**
4       * If the option is file, validate the integrality of the given
       ↪  file by read MD5 value from .md5 file
5       * (1) -f [file path of the file validated] [file path of the .md5
       ↪  file]
6       * (2) -f [file path of the file validated]
7       * (3) -f
8       */
9      if (argc == 4)
```

```
10              {
11                  if (validate(argv[2], argv[3], AUTO) == FAILURE)
12                      help(argv[0]);
13              }
14              else if (argc == 3)
15              {
16                  std::cout << "[INFO] Using the default .md5 file to read
                    ↪  value:\"" << DEFAULT_MD5_FILE << std::endl;
17                  if (validate(argv[2], DEFAULT_MD5_FILE, AUTO) == FAILURE)
18                      help(argv[0]);
19              }
20              else if (argc == 2)
21              {
22                  std::cout << "[INFO] Using the default file to be validated:\""
                    ↪  << DEFAULT_FILE_NAME << "\" and the default MD5 file:\"" <<
                    ↪  DEFAULT_MD5_FILE << "\"" << std::endl;
23                  if (validate(DEFAULT_FILE_NAME, DEFAULT_MD5_FILE, AUTO) ==
                    ↪  FAILURE)
24                      help(argv[0]);
25              }
26              else
27              {
28                  // If the option is file,
29                  // But the number of arguments is > 4, print the help info to
                    ↪  instruct the user
30                  help(argv[0]);
31                  return 1;
32              }
33          }
34          /* blablabla */
```

7 实验结论

本次实验是本学期网络安全实验课的第三次课程作业，**在 Linux 平台上实现了 MD5 算法多功能命令行交互程序。除此之外，我还进一步实现了对字典攻击进行抵御的两种变换算法以及缺省参数的处理以达到更高的安全性与交互性**。总的来说我收获颇丰，具体而言：

1. 本次实验我继续维持着规范的编程架构，**通过清晰的架构，让我本次代码具有极高的可读性和可维护性，我也会保持这个习惯，用于我今后的其它项目中**。并且我继续利用了 Visual Studio Code 在 Linux 平台下使用 Google Test 进行 Debug 的能力，提升了我解决问题的能力。

2. 本次实验我通过对 MD5 算法原理进行复习后，将全部流程进行了实现。**在这期间还遇到了栈溢出等问题，但我凭借耐心一一解决。有的自主创新，并没有完全按照实验参考书的代码和解决思**

路。这**不仅让我对 MD5 算法有了更加清晰的认识，并且增强了我的信息安全实践能力**。

3. 本次实验在完成了基础的编程要求后，**我对自己提出了更高的要求：提高程序的交互性和 MD5 的安全性**。因此我提供了缺省的默认路径选择，额外的参数提供以实现更好的交互；实现了两种 MD5 变换算法以满足不同的安全性需求。**这都让我的编程实践能力更加贴近现实，符合工程需要，更加全面合理**。

4. 本次实验后我认真思考与总结，总结了许多实验解决问题的思路，我希望我可以在未来对本次实验工作进一步完善。

作为本学期的第三次实验，让我对 MD5 信息摘要算法实现以及命令行交互式程序都有了深刻的认识。我希望经过本学期的多次实验，能够**将网络安全的理论知识学透，更重要的是将每次实验做好，更好地应用于实践！争取发挥更多的创造性，成为一名有趣的信安人**。

参考文献