# 第五章　基于 MD5 算法的文件完整性校验程序

## 5.1 本章训练目的与要求

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。

## 5.2 相关背景知识

### 5.2.1　MD5 简介

MD5（message-digest algorithm 5 的缩写）是一种信息摘要算法。信息摘要算法的研究由来已久，早在 1990 年 Ronald L. Rivest 就提出了与 MD5 属于同一系列的散列函数 MD4。经过大量的密码学分析与不断的攻击检测，Rivest 于 1991 年提出了它的改进算法 MD5。目前 MD5 已经得到了广泛的应用，成为许多机构和组织的散列函数标准。

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 的典型应用就是为一段信息（message）生成信息摘要（message-digest），以防止被篡改。Linux 系统自带了计算和校验 MD5 摘要的工具程序—md5sum。用户可以在命令行终端直接运行该程序。md5sum 程序可以创建一个与原始文件名称相同，后缀为.md5 的文件，并将原始文件的 MD5 摘要保存在该文件中。如下所示，对于文件 nankai.txt，利用 md5sum

程序计算出它的摘要（3c771aea5b7e191408ab6b0372ecbf0c）并保存在文件 nankai.md5 中。

```
[root@localhost MD5]# cd md5sum/
[root@localhost md5sum]# ls
nankai.txt
[root@localhost md5sum]# md5sum nankai.txt > nankai.md5
[root@localhost md5sum]# ls
nankai.md5    nankai.txt
[root@localhost md5sum]# vim nankai.md5
   1 3c771aea5b7e191408ab6b0372ecbf0c   nankai.txt
…
```

另外，md5sum 程序可以对文件的完整性进行检测。它通过重新计算原始文件的 MD5 摘要，将计算结果与同名的.md5 文件中的摘要进行对比，若相同，则文件完整；否则，原文件受到了破坏。如下所示，共包含了 2 次检验。第一次通过比较 nankai.md5 中的摘要验证文件 nankai.txt 是完整的。第二次修改了 nankai.txt 文件的内容（在标题后面增加了 written by sky），导致校验失败。

```
[root@localhost md5sum]# md5sum -c nankai.md5
nankai.txt: OK
[root@localhost md5sum]# vim nankai.txt
   1 About Nankai University       (written by sky)
   2
   3 A key multidisciplinary and research-oriented university directly under the jurisdiction of the Ministry of Education, Nankai University, located in Tianjin on the border of the sea of Bohai, is also the alma mater of our beloved late Premier Zhou Enlai.
…
[root@localhost md5sum]# md5sum -c nankai.md5
nankai.txt: FAILED
md5sum: WARNING: 1 of 1 computed checksum did NOT match
```

## 5.2.2 MD5 算法分析

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。

(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 种基本运算是指：

a. $X$：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。

(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。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。

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

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：

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

| | | | |
|---|---|---|---|
| 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：

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

| | | | |
|---|---|---|---|
| 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：

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

| | | | |
|---|---|---|---|
| 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：

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

| | | | |
|---|---|---|---|
| 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。

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

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。

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

## 5.3 实例编程练习

### 5.3.1 编程练习要求

在 Linux 平台下编写应用程序，正确地实现 MD5 算法。要求程序不仅能够为任意长度的字符串生成 MD5 摘要，而且可以为任意大小的文件生成 MD5 摘要。同时，程序还可以利用 MD5 摘要验证文件的完整性。验证文件完整性分为两种方式：一种是在手动输入 MD5 摘要的条件下，计算出当前被测文件的 MD5 摘要，再将两者进行比对。若相同，则文件完好；否则，文件遭到破坏。另一种是先利用 Linux 系统工具 md5sum 为被测文件生成一个后缀为.md5 的同名文件，然后让程序计算出被测文件的 MD5 摘要，将其与.md5 文件中的 MD5 摘要进行比较，最后得出检测结果。具体要求如下所示。

**1. 程序输入格式**

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。

**2. 程序的执行过程**

(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。

```
[root@localhost MD5]# ./MD5 -h
MD5: 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。

```
[root@localhost 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。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。

```
[root@localhost MD5]# ./MD5 -c nankai.txt
The MD5 value of file("nankai.txt") is
```

```
3c771aea5b7e191408ab6b0372ecbf0c
```

(4) 验证文件完整性方法一

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。

```
[root@localhost MD5]# ./MD5 -v nankai.txt
Please input the MD5 value of file("nankai.txt")...
abcdefghijklmnopqrstuvwxyz123456
The old MD5 value of file("nankai.txt") you have input is
abcdefghijklmnopqrstuvwxyz123456
The new MD5 value of file("nankai.txt") that has computed is
3c771aea5b7e191408ab6b0372ecbf0c
Match Error! The file has been modified!
[root@localhost MD5]# ./MD5 -v nankai.txt
Please input the MD5 value of file("nankai.txt")...
3c771aea5b7e191408ab6b0372ecbf0c
The old MD5 value of file("nankai.txt") you have input is
3c771aea5b7e191408ab6b0372ecbf0c
The new MD5 value of file("nankai.txt") that has computed is
3c771aea5b7e191408ab6b0372ecbf0c
OK! The file is integrated
```

(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。整个执行过成如下所示。

```
[root@localhost MD5]# ./MD5 -f nankai.txt nankai.md5
The old MD5 value of file("nankai.txt") in nankai.md5 is
3c771aea5b7e191408ab6b0372ecbf0c
The new MD5 value of file("nankai.txt") that has computed is
3c771aea5b7e191408ab6b0372ecbf0c
OK! The file is integrated
```

总之，本章编程训练有两大重点：一是掌握 MD5 算法，能够编程正确地实现该算法；二是理解在 Linux 环境下运用 MD5 算法检测文件完整性的过程，能够编程模拟这一过程。

### 5.3.2 编程训练设计与分析

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。它的定义如下：

```
class MD5
{
public:
    MD5();
    MD5(const string &str);
    MD5(ifstream &in);
    //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 BYTE* 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

private:
    //对给定长度的字节流进行 MD5 运算
    void Update(const BYTE* input,size_t length);
    void Stop();                                  //用于终止摘要计算过程，输出摘要
    void Transform(const BYTE 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 DWORD *input, BYTE *output, size_t length);
    //将字节流转换为双字流
    void Decode(const BYTE *input, DWORD *output, size_t length);
    //将字节流按照十六进制字符串形式输出
    string BytesToHexString(const BYTE *input, size_t length);

private:
    DWORD state[4];                 //用于表示 4 个初始向量
    DWORD count[2];                 //用于计数，count[0]表示低位，count[1]表示高位
    BYTE    buffer_block[64];       //用于保存计算过程中按块划分后剩下的比特流
    BYTE    digest[16];             //用于保存 128 比特长度的摘要
    bool    is_finished;            //用于标志摘要计算过程是否结束

    static const BYTE padding[64];  //用于保存消息后面填充的数据块
    static const char hex[16];      //用于保存 16 进制的字符
};
```

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。其中 Update 函数和 Transform 函数是 MD5 类的核心部分。下面将重点介绍这两个函数。

(1) Update 函数

MD5 类中有 4 个 Update 重载函数，其中 3 个公有函数是对外接口，在函数体中都调用了私有函数 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，再调用私有函数 Update。

由于 MD5 算法是以 64 字节（即 512 比特）为单位进行计算的，私有函数 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。私有函数 Update 的代码如下。

```
void MD5::Update(const BYTE* input,size_t length)
{
    DWORD i,index,partLen;

    //设置停止标识
    is_finished = false;
    //计算 buffer 已经存放的字节数
    index = (DWORD)((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] += ((DWORD)length << 3)) < ((DWORD)length << 3))   //判断是否进位
        count[1]++;
    count[1] += ((DWORD)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);
    }
```

如图 5-1 所示，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。

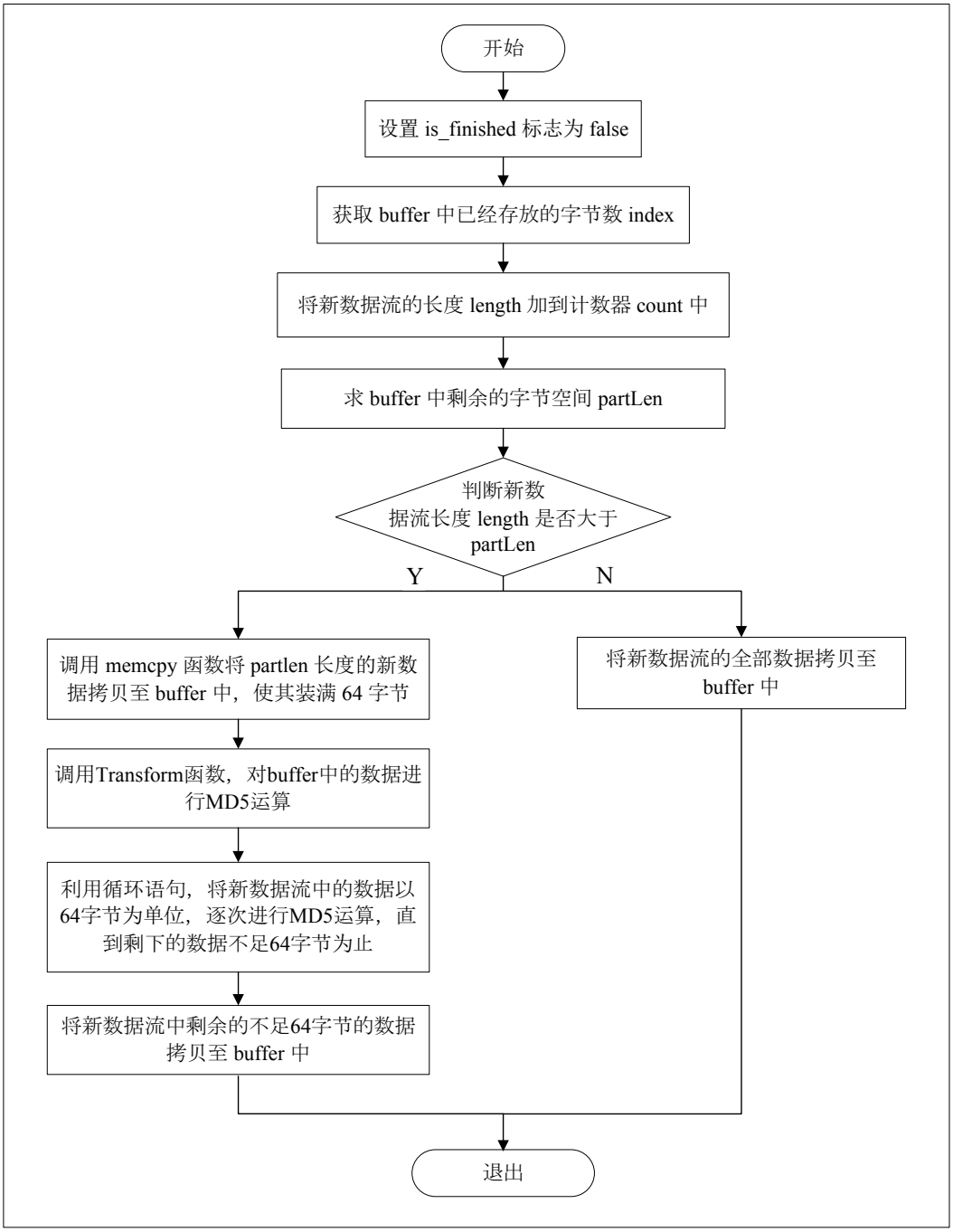


图 5-1 Update 函数流程图

(2) Transform 函数

在 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。下面将给出这些操作和运算的定义。

首先定义在 MD5 四轮迭代计算中向量 A、B、C、D、循环左移的位数。例如，#define S11 7 表示第一轮计算中式子[ABCD, 0, 7, 1]中循环左移的位数。如果需要查阅所有公式关于循环左移位数的定义，请参考具体代码。

其次定义 MD5 算法的四个基本函数，代码如下。

```
#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
```

然后定义 32 位双字的循环左移操作。如下所示，其中 x 表示一个 32 位双字，n 表示循环左移的位数。

```
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))
```

最后定义四轮计算中的 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。

```
#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);
}
```

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。

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，Sij 表示第 i 轮第 j 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。

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

```
void MD5::Transform(const BYTE block[64])
```

```
{
    DWORD a = state[0], b = state[1], c = state[2], d = state[3], x[16];

    Decode(block, x, 64);

    /* 第 1 轮 */
    FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
    FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
     ……
    /* 第 2 轮 */
    GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
    GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
     ……
    /* 第 3 轮 */
    HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
    HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
     ……
    /* 第 4 轮 */
    II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
    II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
     ……

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

**2. 文件完整性检验的设计与实现**

文件完整性检验是在 main 函数中实现的。应用程序为用户提供了多个选项，不但可以在命令行下计算文件的 MD5 摘要，验证文件的完整性，还可以显示程序的帮助信息和 MD5 算法的测试信息。

在 main 函数中，程序通过区分参数 argv[1]的不同值来启动不同的工作流程。如果 argv[1]等于“-h”，表示显示帮助信息；如果 argv[1]等于“-t”，表示显示测试信息；如果 argv[1]等于“-c”，表示计算被测文件的 MD5 摘要；如果 argv[1]等于“-v”，表示根据手工输入的 MD5 摘要验证文件完整性；如果 argv[1]等于“-h”，表示根据.md5 文件中的摘要验证文件完整性。

帮助信息可以协助用户快速地了解命令行输入格式。测试信息可以让用户验证 MD5 算法的正确性。用于测试的消息都是 MD5 算法官方文档（RFC1321）中给出的例子，如果计算的结果相同，则充分说明程序的 MD5 运算过程是正确无误的。

程序提供了两种验证文件完整性的方式：一种是让用户手工输入被测文件的 MD5 摘要，然后调用 MD5 类的运算函数重新计算被测文件的 MD5 摘要，最后将两个摘要逐位进行比

较，进而验证文件的完整性。另一种是从与被测文件对应的.md5 文件中读取 MD5 摘要，然后调用 MD5 类的运算函数重新计算被测文件的 MD5 摘要，最后将两个摘要逐位进行比较，进而验证文件的完整性。

手工输入验证的代码如下。它分为 6 个步骤。

(1) 首先比较参数 argv[1]，判断是否通过手工输入进行验证。若是，则继续下面步骤；否则，退出。

(2) 检测被测文件的路径是否存在，若存在，则继续下面步骤；否则，退出。

(3) 输入被测文件的 MD5 摘要并保存在数组 InputMD5 中。

(4) 打开被测文件，读取被测文件内容，并调用 Update 函数重新计算被测文件的 MD5 摘要。

(5) 调用 Tostring 函数将 MD5 摘要表示成 16 进制字符串形式。

(6) 最后调用 strcmp 函数判断两个摘要是否相同，若相同，则说明被测文件是完整的；否则，说明文件受到了破坏。

```
if (!strcmp(pValidate,argv[1]))                    //判断是否通过手工输入进行验证
{       ……
    if (argv[2] == NULL)                           //判断是否输入了被测文件路径
      ……
      //手动输入了被测文件的 MD5 摘要
    cout<<"Please input the MD5 value of file(\""<<argv[2]<<"\")..."<<endl;
      cin>>InputMD5;
    InputMD5[32] = '\0';
      ……
      //打开被测文件
    pFilePath = argv[2];
    ifstream File_2(pFilePath);
      ……
      //读取文件内容并计算 MD5 摘要
    MD5 md5_obj3;
    md5_obj3.Reset();
    md5_obj3.Update(File_2);
      ……
      //比较摘要，进行验证
    str = md5_obj3.Tostring();
    const char* pResult = str.c_str();
    if (strcmp(pResult,InputMD5))
      ……
    else
      ……
}
```

通过.md5 文件进行验证的代码如下。它与手工输入验证类似，可分为 6 个步骤：

(1) 首先比较参数 argv[1]，判断是否通过.md5 文件进行验证。若是，则继续下面步骤；否则，退出。

(2) 检测被测文件的路径和.md5 文件的路径是否存在，若存在，则继续下面步骤；否则，退出。

(3) 打开.md5 文件，读取文件中的记录，调用 strtok 函数获得被测文件的 MD5 摘要。

(4) 打开被测文件，读取被测文件内容，并调用 Update 函数重新计算被测文件的 MD5 摘要。

(5) 调用 Tostring 函数将 MD5 摘要表示成 16 进制字符串形式。

(6) 最后调用 strcmp 函数判断两个摘要是否相同，若相同，则说明被测文件是完整的；否则，说明文件受到了破坏。

```
if (!strcmp(pFile,argv[1]))        //判断是否通过.md5 文件进行验证
{      ……
       //判断是否输入了被测文件和.md5 文件路径
      if (argv[2] == NULL || argv[3] == NULL)
       ……
       //打开.MD5 文件
      pFilePath = argv[3];
      ifstream File_3(pFilePath);
       ……
      //读取.MD5 文件中的一行记录
      File_3.getline(Record,50);

      //以空格为标记,获得.MD5 文件中的 MD5 值与对应文件名
      pMD5 = strtok(Record,pS);
      pFileName = strtok(NULL,pS);
       ……
      //打开被测文件
      pFilePath = argv[2];
      ifstream File_4(pFilePath);
       ……
       //读取文件内容并计算 MD5 摘要
      MD5 md5_obj4;
      md5_obj4.Reset();
      md5_obj4.Update(File_4);
       ……
      str = md5_obj4.Tostring();
      const char* pResult2 = str.c_str();
       //比较摘要，进行验证
      if (strcmp(pResult2,pMD5))
          ……
      else
```

```
        ……
}
```

## 5.4 扩展与提高

### 5.4.1 MD5 算法与 Linux 口令保护

除了验证文件完整性以外，MD5 算法在其它方面还有着广泛的应用。众所周知，Linux 系统的用户口令需要经过加密才能保存在配置文件中，而 MD5 算法则是加密用户口令的最常用算法。

最早，Linux 用户口令加密后保存在用户配置文件/etc/passwd 中。文件中的每一行对应一个用户，并用冒号（:）划分为 7 个字段。用户口令加密后就放在每一行的第二个字段里。一般情况下，/etc/passwd 文件允许所有用户读取，但只允许 root 用户写入。因此，恶意用户可以轻松地读取加密后的口令字符串，然后使用字典攻击等手段获得其它用户的口令。鉴于上述原因，后期的 Linux 版本专门设置一个用户影子口令文件/etc/shadow 用于保存用户口令，并且只允许 root 用户读取该文件。这样普通用户就无法读取加密后的口令文件，从而进一步提高了口令的安全性。

在/etc/passwd 文件中，每一行的第二个字段表示是否有加密口令。若为 x，表示该账户设置了口令，否则，表示没有设置口令。以下面的/etc/passwd 文件为例，第一行记录表明 root 用户设置了口令。而真正的口令加密字符串则保存在/etc/shadow 文件中。

```
1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
…
```

与/etc/passwd 文件类似，/etc/shadow 文件的每一行对应一个用户，用冒号（:）划分了九个字段。用户加密后的口令就保存在第二个字段中。如下所示，root 用户的口令加密字符串是“$1$Q.DJ5Zss$P.WvrDK/ogM9w/KNlrEAR0”。该字符串是利用 MD5 算法生成的 128 位摘要。因为只有 root 用户才有权查看，所以有力地保障了 Linux 口令的安全性。

```
1 root:$1$Q.DJ5Zss$P.WvrDK/ogM9w/KNlrEAR0:13997:0:99999:7:::
2 bin:*:13948:0:99999:7:::
……
```

### 5.4.2 Linux 系统 GRUB 的 MD5 加密方法

在使用 Linux 过程中，用户经常会忘记自己之前设定的口令。鉴于这种情况，GRUB 为用户提供了一条恢复口令的捷径。GRUB 是一个引导装入器，负责装入内核并引导 Linux 系统。类似于在计算机上安装两个 windows 时出现的选单管理器 OS Loader，GRUB 可以让我们选择使用哪一个操作系统。与 Linux 之前的引导装入器 LILO 相比，GRUB 具有功能强大，引导过程灵活，安全性高等优点。

下面介绍如何利用引导工具 GRUB 恢复口令：

(1) 首先启动计算机，在 GRUB 界面时，选择启动的 Linux 选项，然后按 e 键。

(2) 选择以 kernel 开头的一行，按 e 键进入编辑模式。在此行的末尾按空格键后输入 single，如下所示。最后按回车键退出编辑。

```
kernel /boot/vmlinuz-2.4.18-14 single ro root=LABEL=/    single
```

(3) 回到 GRUB 界面后，按 b 键来引导进入单用户模式。

(4) 进入单用户模式后，利用 password 命令设置新密码。

GRUB 虽然提供了一种恢复口令的方法，但是也带来了一个破解口令的漏洞。恶意用户完全可以利用上述方法获得 Linux 系统的用户口令。为了阻止未授权的用户登录启动引导程序，需要对 GRUB 设置密码。给 GRUB 设置密码一般有两种方式，一种是设置明文密码，一种是用 MD5 设置加密密码。

### 1. GRUB 设置明文密码

密码设置过程如下：在/etc 目录下打开配置文件 grub.conf，在“timeout = …”一行下面插入语句“password=＊＊＊＊”。其中＊代表明文密码。然后在“title...”一行下面插入语句“lock”。保存退出之后重启计算机。再次进入 GRUB 界面时，系统将提示需要按“p”键输入 GRUB 密码。

### 2. GRUB 设置 MD5 加密密码

将密码以明文形式保存在配置文件中是一种不安全的方法。如果利用 MD5 算法对 GRUB 密码进行加密，然后再将加密字符串保存在配置文件中，就可以进一步提高密码的安全性。MD5 加密密码的设置过程如下。

(1) 首先生成 GRUB 的加密密码。在命令行输入 grub，进入 GRUB 界面。然后输入 md5crypt(或 password --md5)，再输入你的密码，就产生一个 md5 加密字符串，如下所示。复制下该字符后，输入“quit”退出。

```
grub> md5crypt
Password: ******
Encrypted: $1$mLpvT$y8iEtGvO7/IBRJdJhiD831
grub> quit
```

(2) 如下所示，在/etc 目录下打开配置文件 grub.conf，在“timeout = …”一行下面插入语句“password --md5 加密字符串”。然后在语句“title...”语句之后插入语句“lock”。保存退出之后重启计算机。

```
[root@localhost ~]# cd /etc
[root@localhost etc]# vim grub.conf
 ……
10 default=0
11 timeout=10
12 password --md5 $1$mLpvT$y8iEtGvO7/IBRJdJhiD831
```

```
13 splashimage=(hd0,0)/grub/splash.xpm.gz
14 hiddenmenu
15 title Fedora (2.6.23.1-42.fc8)
16 lock
17         root (hd0,0)
18         kernel /vmlinuz-2.6.23.1-42.fc8 ro root=/dev/VolGroup00/LogVol00 rhgb quiet
19         initrd /initrd-2.6.23.1-42.fc8.img
```

(3) 如图 5-2 所示，再次进入 GRUB 界面后，系统提示需要按“p”键输入 GRUB 密码才能引导启动。

经过上面的设置，未授权用户就不能轻易地引导系统，从而获得用户口令了。由此可见，MD5 算法在保护用户口令，维护 Linux 系统安全中扮演着重要的角色。


图 5-2 GRUB 密码保护

### 5.4.3 字典攻击与 MD5 变换算法

因为 MD5 算法的运算过程不可逆，所以对任意一个 MD5 散列逆向计算出明文的破解机是不存在的。目前，破解 MD5 的最有效方法之一就是字典攻击。所谓字典攻击，就是指事先收集大量明文和对应的 MD5 散列，并把它们成对地保存在数据库中，然后在数据库中寻找与当前 MD5 散列对应的明文进行破解。当然如果数据库中没有相应的记录，破解工作也就无法进行。

目前有许多收集 MD5 字典的网站。下面列举了 3 个代表性的网站，供读者查阅。

(1) http://md5.rednoize.com 网站采用搜索引擎的形式，支持明文字符串与 MD5 散列间的双向转换。目前拥有 1,963,442 条记录。

(2) http://www.neeao.com/md5/ 这是一个中文网站，目前拥有近 17 亿条记录。

(3) http://www.xmd5.org/index_cn.htm 这是一个多语言网站，目前拥有 2000 万条记录。

如果攻击者拥有数据量巨大的密码字典，并且建立了许多 MD5 原文/散列对照数据库，那么就能快速地找到常用明文字符串的 MD5 散列。因此字典攻击已经成为一种破解 MD5 散列的高效途径。然而，上述字典所使用的都是常规的 MD5 算法，即原文-->MD5-->密文。如果我们对 MD5 算法进行变换，就可以使这些 MD5 字典无所作为。

对 MD5 进行变换的方法很多，最容易理解的就是对同一原文进行多次 MD5 运算。在下面的代码中，自定义了一个函数 md5_more，它有两个参数 data 和 times，前者表示需要加密的明文，后者表示重复加密的次数。可以看到，在函数体中，调用了 times 次 md5 函数，因此将明文加密了 times 遍。这种变换也可以用递归的方法来实现，此处不再赘述。

```
function md5_more(data, times)
{   //循环使用 MD5
    for (i = 0; i < times; i++) {
    data = md5(data);
    }
    return data;
}
```

另一种变换的主要思想是：首先经过一次 MD5 运算，得到一个由 32 个字符（用 16 进制表示）组成的散列字符串，然后将该字符串分割为若干子串，再对每个子串进行 MD5 运算，最后将每个子串的 MD5 散列合并成一个字符串作为 MD5 算法的输入并计算出一个最终的散列。

在下面的代码中，函数 divide 首先对明文进行一次 MD5 运算，然后将得到的散列分为左、右两个字符串，各包含 16 个字符。分别对两个子串进行 MD5 运算后，再将计算结果合成一个 64 字节的字符串。最后用 md5 算法将该字符串再计算一次，得到最终的结果。

```
//把密文分割成两段，每段 16 个字符
function md5_divide(data)
{
    //先把明文加密成长度为 32 字符的密文
    data = md5(data);
    //把密码分割成两段
    left = substr(data, 0, 16);
    right = substr(data, 16, 16);
    //分别加密后再合并
    data = md5(left).md5(right);
    //最后把长字符串再加密一次，成为 32 字符散列
    return md5($data);
}
```

除了上述两种变换方法之外，还有附加字符串干涉，字符串次序干涉，大小写变换干涉等多种方法。可以说，MD5 变换算法以增加计算开销为代价，有效地抑制了字典攻击，提高了算法的安全性。但是，一旦攻击者获得了某个应用程序指定的 MD5 变换算法，那么该变换算法就再也没有任何秘密可言，唯一有益的是消耗了攻击者更多的计算资源。