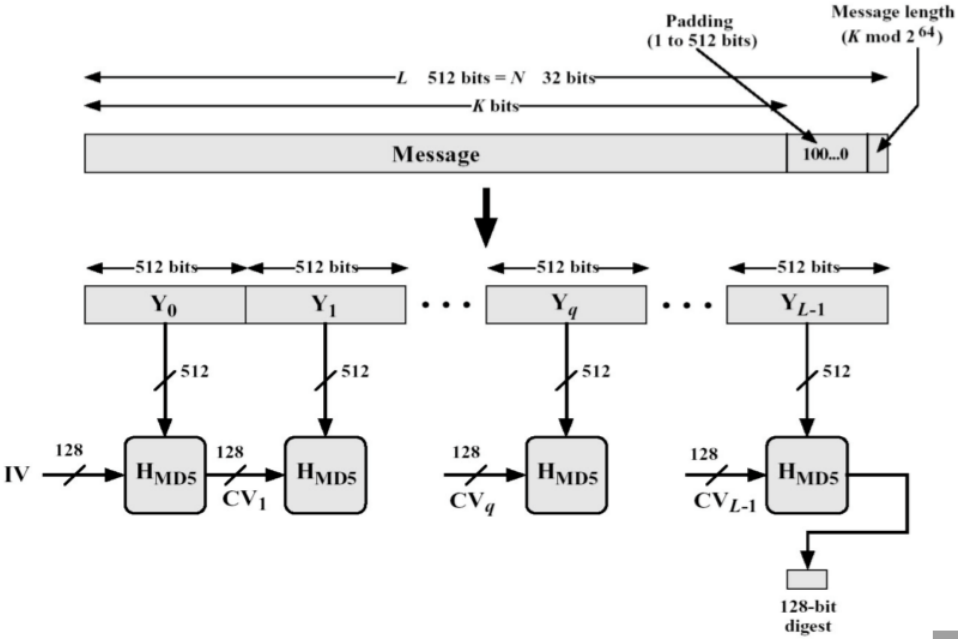
Assignment2

15331416 赵寒旭 数字媒体技术

1. Design an executable MD5 program in C/C++ with some short samples.

1.1 MD5 算法概述

MD5 信息摘要算法把输入的任意长度的信息计算生成 128 位长度输出结果。



(1) 填充

在原始消息数据尾部填充标识 100…0，填充后的消息位数 L≡448 (mod 512)。至少要填充 1 个位，所以标识长度 1~512 位。

再向上述填充好的消息尾部附加原始消息的位数的低 64 位，最后得到一个长度 L 是 512 位整数倍的消息。

即数据扩展至 K*512+448 位，最后 64 位表示数据的原始长度 b（添加填充位前的长度），若遇到 b 大于 2^64（极少），只取低 64 位。

(2) 分块

把填充后的消息结果分割为 L 个 512 位的分组，结果表示为 N 个 32 位长的字。（N=L*16）

(3) 初始化 MD 缓冲区

初始化一个 128 位的 MD 缓冲区，也表示为 4 个 32 位长度的寄存器(A,B,C,D)。在 MD 缓冲区进行迭代，最后一步的 128 位输出即为算法结果。

ABCD 分别是 32 位寄存器，初始化用十六进制数，采用小端存储。

注意：此处小端意义：

一个字 32 位，一个字节 8 位，4 个字节一组转化，在字节单位上，此处小端将低位字节排放在低地址端，高位字节排放在高地址端，在字节内部，并不做转换，

保持原顺序存储。

A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476

Word A	01	23	45	67
Word B	89	AB	CD	EF
Word C	FE	DC	BA	98
Word D	76	54	32	10

(4) 处理 16 个 32 位字的数据

以 512 位（16*32）消息分组为单位，每一分组经过 4 个循环的压缩算法。

1) 定义 4 个辅助函数，每个函数输入三个 32 位长的字，输出一个 32 位长的字。

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

4 轮循环中使用的生成函数（轮函数）g 是一个 32 位非线性逻辑函数，
在相应各轮的定义如右图。

轮次	Function g	$g(b, c, d)$
1	$F(b,c,d)$	$(b\wedge c)\vee(\neg b\wedge d)$
2	$G(b,c,d)$	$(b\wedge d)\vee(c\wedge\neg d)$
3	$H(b,c,d)$	$b\oplus c\oplus d$
4	$I(b,c,d)$	$c\oplus(b\vee\neg d)$

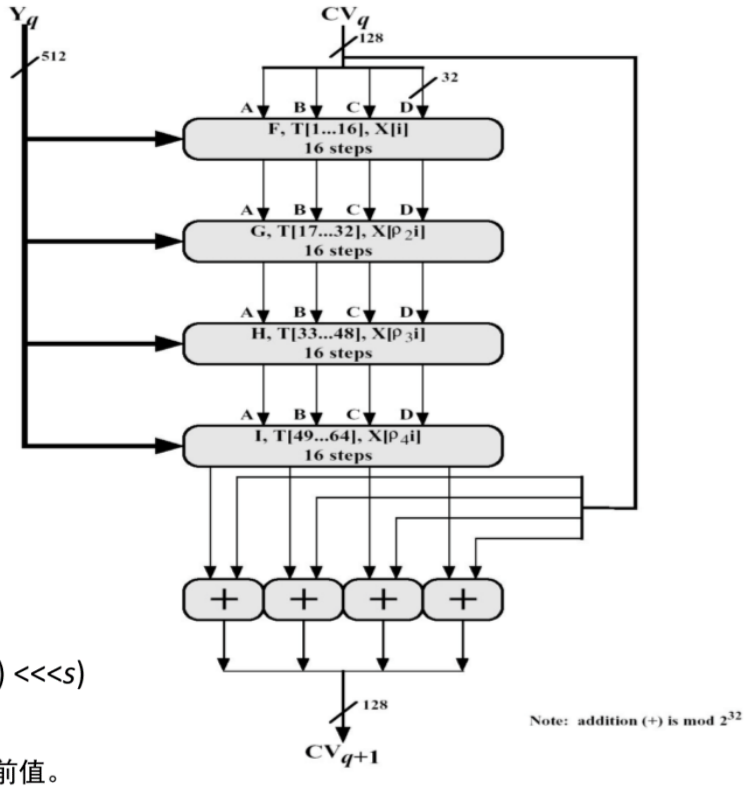
2) MD5 压缩函数：循环迭代运算

MD5 第 q 分组的 4 轮循环逻辑（压缩函数）如右图所示。

H_{MD5} 从 CV 输入 128 位，从 消息分组输入 512 位，完成 4 轮循环后，输出 128 位， 用于下一轮输入的 CV 值。

每轮循环分别固定不同的 生成函数 F, G, H, I，结合 指定的 T 表元素 T[] 和消 息分组的不同部分 X[] 做 16 次运算，生成下一轮循环的输入。

总共有 64 次迭代运算。



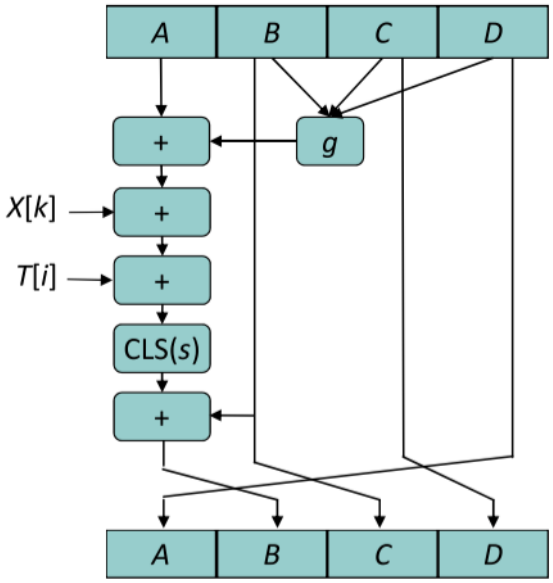
3) 每轮循环中的一步运算逻辑

$a \leftarrow b + ((a + g(b,c,d) + X[k] + T[i]) <<< s)$

说明：

- a, b, c, d : MD 缓冲区 (A, B, C, D) 的当前值。
- g : 轮函数 (F, G, H, I 中的一个)。
- $<<<s$: 将32位输入循环左移 (CLS) s 位。
- $X[k]$: 当前处理消息分组的第 k 个32位字，即 $M_{q\times16+k}$。
- $T[i]$: T 表的第 i 个元素，32位字。
- + : 模 2^{32} 加法。

每轮循环中的一步运算逻辑：



各轮迭代中 X[k]之间的关系：

- 第1轮迭代：$X[j]$, $j = 1..16$.
  顺序使用 X[0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15]
- 第2轮迭代：$X[\rho_2(j)]$, $\rho_2(j) = (1 + 5j) \bmod 16$, $j = 1..16$.
  顺序使用 X[1, 6,11, 0, 5,10,15, 4, 9,14, 3, 8,13, 2, 7,12]
- 第3轮迭代：$X[\rho_3(j)]$, $\rho_3(j) = (5 + 3j) \bmod 16$, $j = 1..16$.
  顺序使用 X[5, 8,11,14, 1, 4, 7,10,13, 0, 3, 6, 9,12,15, 2]
- 第4轮迭代：$X[\rho_4(j)]$, $\rho_4(j) = 7j \bmod 16$, $j = 1..16$.
  顺序使用 X[0, 7,14, 5,12, 3,10, 1, 8,15, 6,13, 4,11, 2, 9]

各次迭代运算采用的 T 值

T[33..36] = { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
T[37..40] = { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
T[41..44] = { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
T[45..48] = { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
T[49..52] = { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
T[53..56] = { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
T[57..60] = { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
T[61..64] = { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

各次迭代运算采用的左循环移位的 s 值

s[1..16] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }
s[17..32] = { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }
s[33..48] = { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }
s[49..64] = { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }

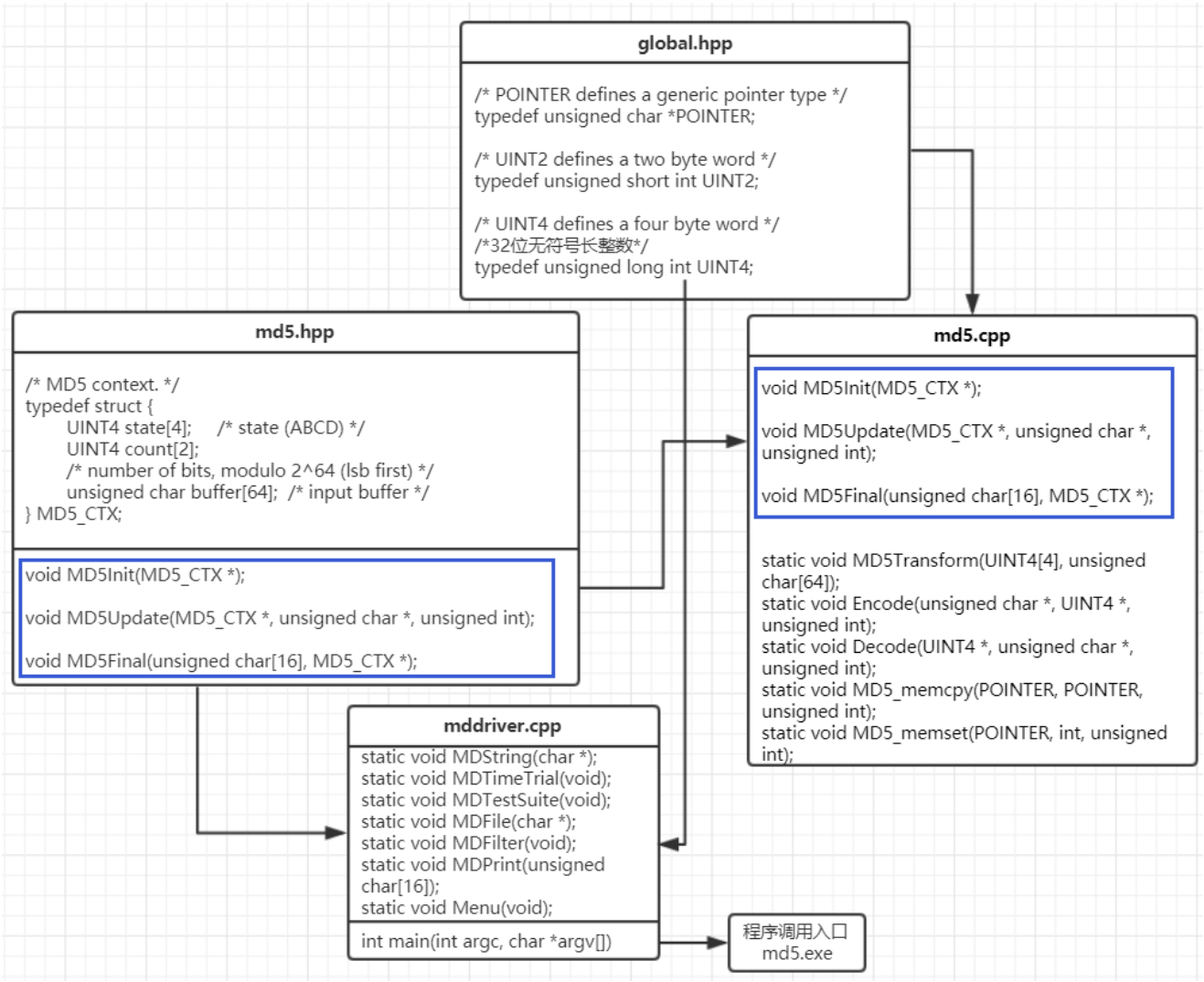
(5) 输出结果

最终结果形式为：A，B，C，D，低位字节 A 开始，高位字节 D 结束。

1.2 C++算法实现

此部分主要参考了 https://www.rfc-editor.org/pdfrfc/rfc1321.txt.pdf 资料
网址：ietf.org　MD5(1992,RFC1321)
为方便显示代码块，将部分源代码转为 markdown 高亮显示截图。

(1) 程序文件结构



(2) 具体实现过程

1) global.hpp　全局头文件

```
/* POINTER defines a generic pointer type */
typedef unsigned char *POINTER;

/* UINT2 defines a two byte word */
typedef unsigned short int UINT2;

/* UINT4 defines a four byte word */
/*32位无符号长整数*/
typedef unsigned long int UINT4;
```

为复杂的声明定义在本程序中全局通用的较为简单的别名。
用无符号长整型变量存储寄存器中保存的值。

2) md5.hpp

```
/* MD5 context. */
typedef struct {
    UINT4 state[4];                                   /* state (ABCD) */
    UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
    unsigned char buffer[64];                         /* input buffer */
} MD5_CTX;

/* MD5 initialization. */
void MD5Init(MD5_CTX *);

/* MD5 block update operation. */
void MD5Update(MD5_CTX *, unsigned char *, unsigned int);

/* MD5 finalization. */
void MD5Final(unsigned char[16], MD5_CTX *);
```

state[4]对应 4 个 32 位寄存器 A,B,C,D
count[2]是形成分组的累计位数（一边分组压缩，一遍累计长度），共 64 位
buffer 输入缓冲区

3) md5.cpp
主要功能：实现 md5.hpp 中定义的三个函数
a) MD5Init:：MD5 初始化
寄存器（A，B，C，D）置 16 进制初值作为初始向量，采用小端存储。
count 置 0。

```
void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int inputLen)
```

```
    context->count[0] = context->count[1] = 0;
    /* Load magic initialization constants.*/
    context->state[0] = 0x67452301;
    context->state[1] = 0xefcdab89;
    context->state[2] = 0x98badcfe;
    context->state[3] = 0x10325476;
```

b) MD5Update：MD5 的主要计算过程
传入输入缓冲区和输入数据长度，进行 MD5 操作，更新 context 的值

```
void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int inputLen)
```

c) MD5Final：结束 MD5 操作，得到 MD5 输出结果，context 归零。

```
void MD5Final(unsigned char digest[16], MD5_CTX *context)
```

d) MD5Transform
MD5 基本变换，用于在 MD5Update 时改变寄存器的值。

```
static void MD5Transform(UINT4 state[4], unsigned char block[64])
```

e) Encode
存储的无符号长整型数转为 4 个字节，以位处理对象数据

```
static void Encode(unsigned char *output, UINT4 *input, unsigned int len)
```

f) Decode
4 个字节转一个无符号长整型数

```
static void Decode(UINT4 *output, unsigned char *input, unsigned int len)
```

g) MD5_memcpy
按 len 长度把 input 中的值逐位赋值给 output，用于 MD5Update 中 context 的更新等操作。

```
static void MD5_memcpy(POINTER output, POINTER input, unsigned int len)
```

h) MD5_memset
把 output 前 len 位置为 value 值。
用于在 MD5Final 中把 cntext 值置 0。

```
static void MD5_memset(POINTER output, int value, unsigned int len)
```

4) mddriver.cpp

主要函数声明如下：

```
1.  static void MDString(char *);
2.  static void MDTimeTrial(void);
3.  static void MDTestSuite(void);
4.  static void MDFile(char *);
5.  static void MDFilter(void);
6.  static void MDPrint(unsigned char[16]);
7.  static void Menu(void);
```

a) MDString

将一个字符串 string 经过 MD5 操作得到摘要 digest 的值，打印结果

```
static void MDString(char *string)
```

调用 md5.hpp 中声明的函数，得到 digest 结果

```
7.      MDInit(&context);
8.      MDUpdate(&context, (unsigned char *)string, len);
9.      MDFinal(digest, &context);
```

b) MDTimeTrial

运行时间测量。

计算对测试长度数据做 MD5 需要的时间。

```
1.  #define TEST_BLOCK_LEN 1000
2.  #define TEST_BLOCK_COUNT 1000
```

c) MDTestSuite

固定样本测试函数

```
1.     MDString("");
2.     MDString("a");
3.     MDString("abc");
4.     MDString("message digest");
5.     MDString("abcdefghijklmnopqrstuvwxyz");
6.     MDString("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
7.     MDString("12345678901234567890123456789012345678901234567890123456789012345678
901234567890");
```

d) MDFile

对指定文件进行 MD5 操作得到其 digest 值，打印结果

```
static void MDFile(char *filename)
```

e) MDFilter

计算标准输入的 digest，打印结果

```
    MDInit(&context);

    if (fgets((char *)buffer, 128, stdin) != NULL) {
        len = strlen((char *)buffer);
        buffer[len - 1] = '\0'; // 尾部换行符去除
        len--;
    }
    else {
        return;
    }


    MDUpdate(&context, buffer, len);
    MDFinal(digest, &context);
```

f) MDPrint

以 16 进制打印 digest

```
printf("%02x", digest[i]);
```

g) Menu

在命令行显示功能菜单

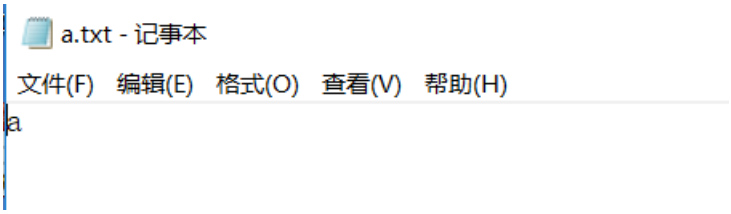
```
/* Main driver.

Arguments (may be any combination):
-s string - digests string
-t        - runs time trial
-x        - runs test script
filename  - digests file
(none)    - digests standard input
-h        - print the menu
*/
```

1.3 程序运行截图

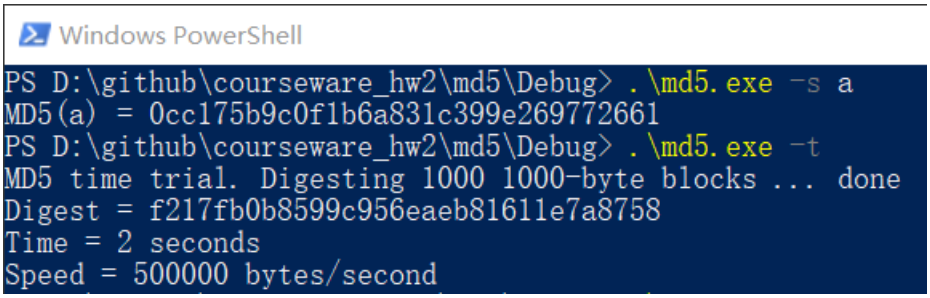
按照 menu 功能顺序依次测试

注：文件 a.txt 事先放入 Debug 文件夹，内容为 a。



命令行字符串输入进行 MD5 转换：-s a

时间测试：-t

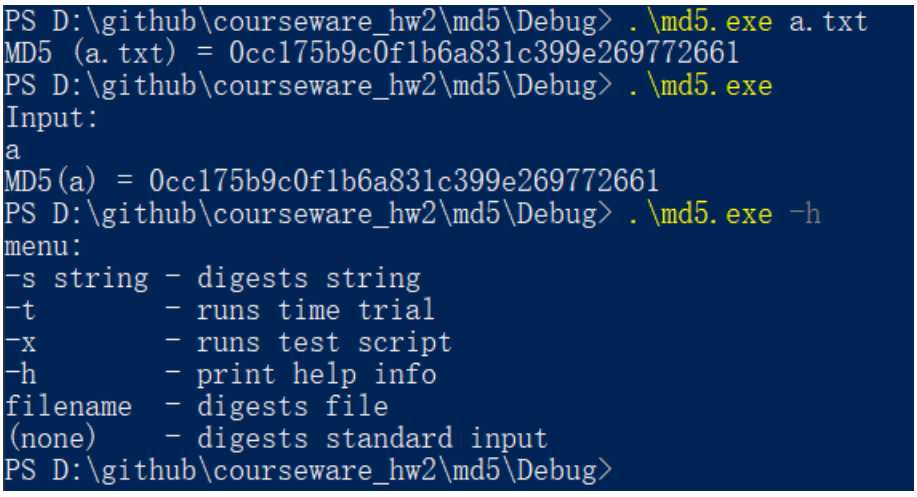


固定样例测试结果：-x

文件 MD5 转换：a.txt

空命令行后续参数直接调用 MD5Filter()

显示帮助：-h



2. How MD5 works for password protection

2.1 MD5 算法实现

MD5 算法的具体过程可以参考 1.1 中的算法描述部分，已经做出详细解释。

2.2 MD5 实现密码保护的工作原理

(1) 不可逆性

MD5 是一个安全的散列算法，输入两个不同的明文不会得到相同的输出值，根据输出值，不能得到原始的明文，即其过程不可逆。

MD5 在计算过程中丢失了原文的信息，且理论上有小概率使一个 MD5 的结果对应多个明文。（发生碰撞）

(2) 安全性

MD5 是输入不定长度信息，输出固定长度 128-bits 的算法。经过程序流程，生成四个 32 位数据，最后联合起来成为一个 128-bits 散列。基本方式为，求余、取余、调整长度、与链接变量进行循环运算。得出结果。

应用于密码保护时：用户的密码不能使用明文存入到数据库中，中间要经过一定的算法，对密码进行转换成暗文，MD5 是很好的一种选择，MD5 是一种不可逆的字符串变换算法，我们在使用的时候，将用户密码转换成 MD5 值存入到数据库中，当用户登录是同样对密码进行运算得到 MD5 值，再将该值与数据库中得 MD5 值进行比较，若相同则登录成功，否则失败。这样系统管理员不能从数据库中的 MD5 值获得用户的原密码值，保障了密码的安全性。