

- (1) Design an executable MD5 program in C/C++ with some short samples.
- (2) Show me how MD5 works for password protection.

1. 算法原理概述

MD5 算法的基本过程为：求余、取余、调整长度、与链接变量进行循环运算、得到结果。

在对输入的明文初始化后，MD5 按每组 512 位处理输入的信息，每一分组又被划分为 16 个 32 位子分组，经过一系列处理后，算法的输出由 4 个 32 位分组组成，这 4 个 32 位分组串联后生成 128 位的散列值。

(1) 填充

在原始消息数据尾部填充标识 $100\cdots 0$ ，填充后的消息位数 $L=448 \pmod{512}$ 。至少要填充 1 个位，所以标识长度 1~512 位（至少有一位）。再向上述填充好的消息尾部附加原始消息长度值的低 64 位（填充前信息的长度转换为 64 位二进制数，如果原信息超过 64 位则取租后 64 位），最后得到一个长度 L 是 512 位整数倍的消息。

(2) 分块

把填充后的消息结果分割为 L 个 512 位的分组： $Y_0, Y_1, \cdots, Y_{L-1}$ 。结果也表示成 N 个 32 位长的字 $M_0, M_1, \cdots, M_{N-1}$ ， $N = L*16$ 。

(3) 初始化散列值

初始化一个 128 位的 MD 缓冲区，记为 CV_q ，也表示为 4 个 32 位寄存器 (A, B, C, D)； $CV_0 = IV$ 。迭代在 MD 缓冲区进行，最后一步的 128 位输出即为算法结果。

寄存器 (A, B, C, D) 置 16 进制初值作为初始向量 IV ，并采用小端存储 (little-endian) 的存储结构：

A = 0x67452301 (从左到右为低地址到高地址)

B = 0xEFCDAB89

C = 0x98BADCFE

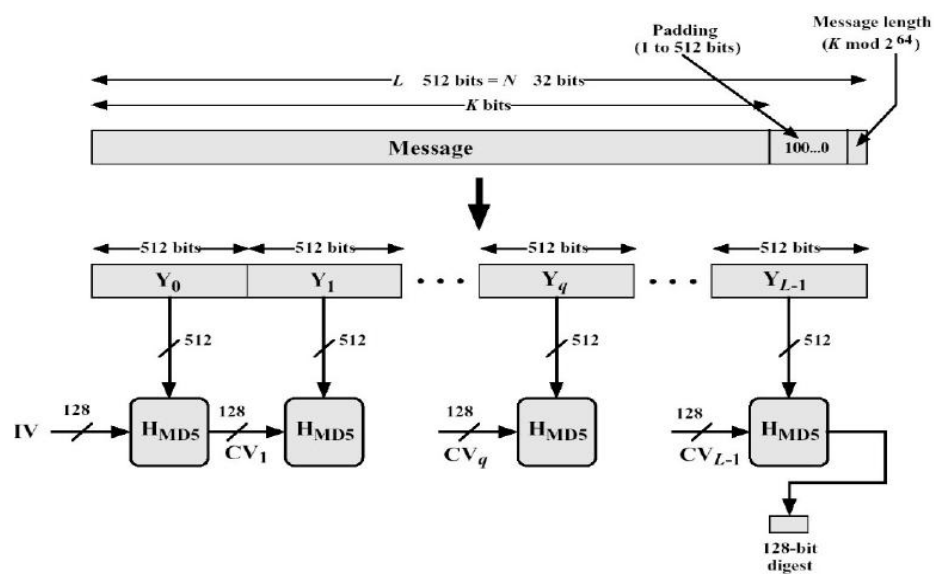
D = 0x10325476

(4) 总控流程

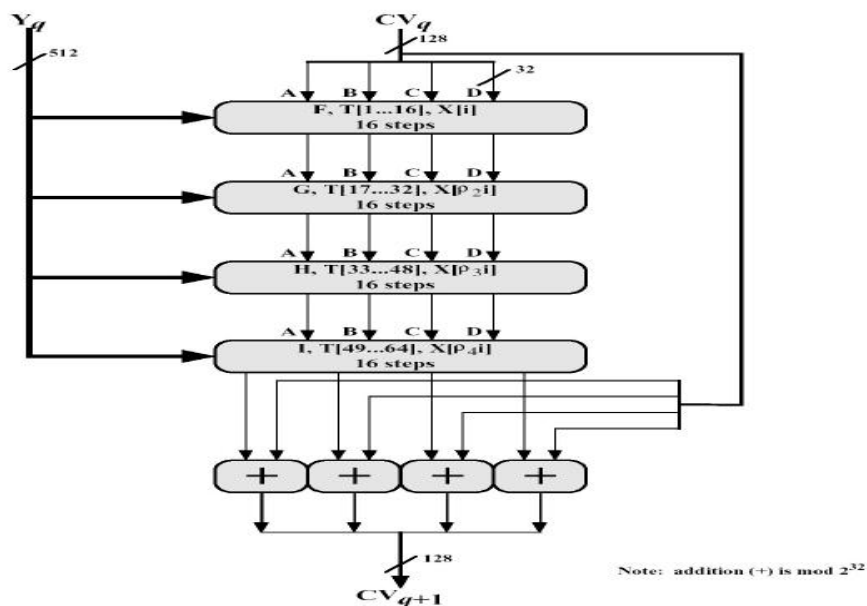
以 512 位消息分组为单位，每一分组 Y_q ($q = 0, 1, \dots, L-1$) (512 位) 经过 4 个循环的压缩算法 (得到 128 位)，表示为：

$CV_0 = IV$

$CV_i = H_{MD5}(CV_{i-1}, Y_i)$



(5) MD5 主循环



输入是 512 位的分组 Y_q ，经过 4 轮迭代，每轮迭代有 16 步后输出 A、B、C、D，将 ABCD 串联后得到 128 位的 CV_{q+1} 。

① 初始 ABCD 为 CV_q 划分为 32 位得到。每一轮需要 16 步操作，分别用到该分组中的 16 个 32 位消息。

② 每轮运算中的一步操作为： $A \leftarrow b + ((a + g(b,c,d) + X[k] + T[i]) \lll s)$

a,b,c,d:MD 缓冲区的当前值。

g:轮函数 (FGHI) 中的一个

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	$I(b,c,d)$	$c \oplus (b \vee \neg d)$

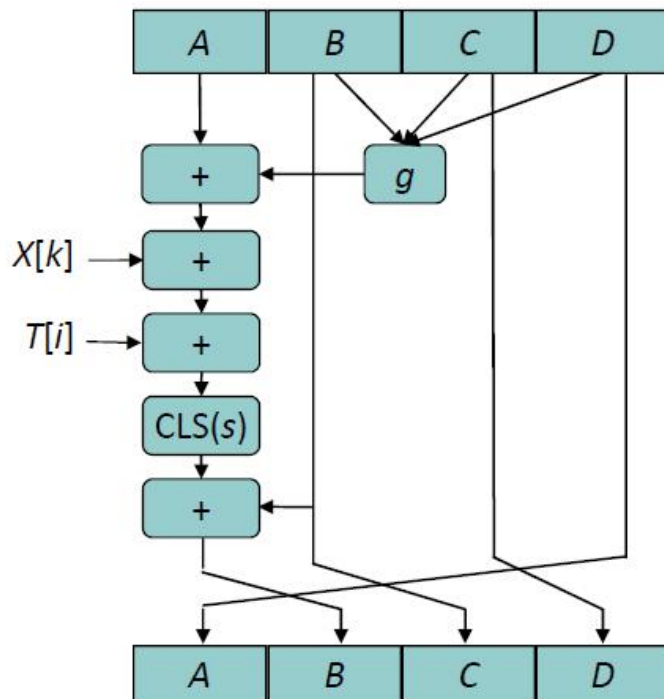
$\lll s$:循环左移 s 位

$X[k]$:当前处理消息的第 k 个 32 字，由相应的计算公式得到 k

$T[i]$:t 表中的第 i 个元素，32 位字，i 位当前进行到第几步

+:模 2^{32} 加法

每一轮迭代中的一步



MD5 每轮迭代中的一步运算逻辑

③ X[K]:

第一轮迭代: $X[j]$, $j = 1..16$.

第二轮迭代: $X[2(j)]$, $2(j) = (1 + 5j) \bmod 16$, $j = 1..16$.

第三轮迭代: $X[3(j)]$, $3(j) = (5 + 3j) \bmod 16$, $j = 1..16$.

第四轮迭代: $X[4(j)]$, $4(j) = 7j \bmod 16$, $j = 1..16$.

④ $T[i] = \text{int}(2^{32} \cdot |\sin(i)|)$

⑤ 最后将得到的 ABCD 和原始的 abcd 分别进行相加就完成了 MD5 的一

次运算

2. 代码实现

总体结构:

```

9  vector<string> userData;
10 string decToHexString(UINT32 ans);
11 void loadData();
12 void printResult(UINT32 result[4]);
13 void login();
14 void signin();

```

```

22 public:
23     MD5(string str);
24     // expand the initial message to be times of 512 bits
25     void generateT();
26     BYTE* expand();
27     // the index of X[j]
28     UINT32 getIndexOfX(UINT32 j, UINT32 k);
29     // get result of function FGHI
30     UINT32 getFunctionRes(UINT32 cycleN, UINT32 B, UINT32 C, UINT32 D);
31     UINT32 shiftLeft(UINT32 src, UINT32 shiftN);
32     void rotate(UINT32 &A, UINT32 &B, UINT32 &C, UINT32 &D);
33     void getResult();
34     // little endian int32 to byte at the last
35     void processEndian();
36     UINT32 F(UINT32 B, UINT32 C, UINT32 D);
37     UINT32 G(UINT32 B, UINT32 C, UINT32 D);
38     UINT32 H(UINT32 B, UINT32 C, UINT32 D);
39     UINT32 I(UINT32 B, UINT32 C, UINT32 D);

```

填充实现：

```

43 BYTE* MD5::expand() {
44     const BYTE* str = message.c_str();
45     int len = strlen(str);
46     // 1(byte) is first expand 10000000 and promise that it expands at least 1
47     // 8(bytes) is the binary bits(64bits) of initial message len
48     UINT32 zeroN = 64 - (len + 1 + 8) % 64;
49     BYTE* res = new BYTE[len + 1 + zeroN + 8 + 1];
50     for (int i = 0; i < len; ++i)
51         res[i] = str[i];
52     res[len] = 0x80; // 10000000
53     for (int i = len + 1; i < len + 1 + zeroN; ++i)
54         res[i] = 0;
55     // len is calculate as bytes and when expand it should be calculate as bits
56     UINT64 binaryLen = (UINT64)len * 8;
57     memcpy(res + len + 1 + zeroN, &binaryLen, 8);
58     return res;
59 }

```

主循环的实现：

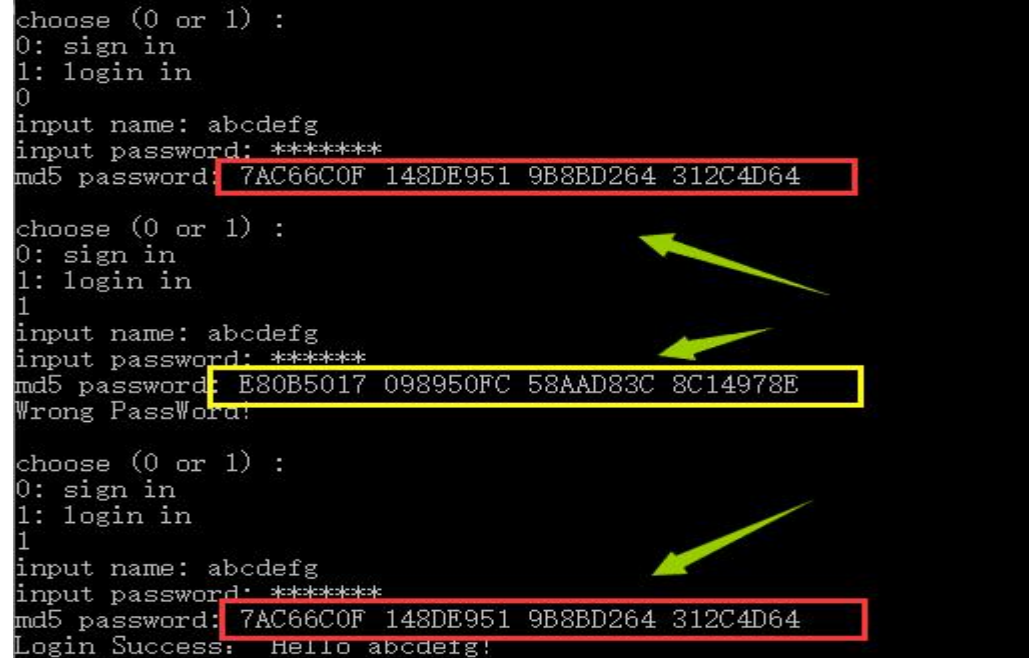
```

for (int i = 0; i < n - 1; i += 64) {
    // Lth block
    int L = i / 64;
    // result(q) is the value for CV(q+1)
    A = result[0];
    B = result[1];
    C = result[2];
    D = result[3];
    // 4 cycles
    for (int j = 0; j < 4; ++j) {
        // 16 steps
        for (int k = 0; k < 16; ++k) {
            UINT32 funRes = getFunctionRes(j, B, C, D);
            // *4 is because 4bytes is UINT32
            UINT32 indexOfX = i + getIndexOfX(j, k) * 4;
            // when calculate, all use the little endian and swap the result at the end
            UINT32 shiftRes = shiftLeft((A + funRes + *(UINT32*)(expandMessage + indexOfX)
                + T[j * 16 + k]), s[j][k % 4]);
            A = B + shiftRes;
            rotate(A, B, C, D);
        }
    }
    // UINT32 r = (UINT32)pow(2, 32);
    result[0] = result[0] + A;
    result[1] = result[1] + B;
    result[2] = result[2] + C;
    result[3] = result[3] + D;
}
}

```

注：具体代码见.cpp 文件。

实验结果截图：



```

choose (0 or 1) :
0: sign in
1: login in
0
input name: abcdefg
input password: *****
md5 password: 7AC66C0F 148DE951 9B8BD264 312C4D64

choose (0 or 1) :
0: sign in
1: login in
1
input name: abcdefg
input password: *****
md5 password: E80B5017 098950FC 58AAD83C 8C14978E
Wrong PassWord!

choose (0 or 1) :
0: sign in
1: login in
1
input name: abcdefg
input password: *****
md5 password: 7AC66C0F 148DE951 9B8BD264 312C4D64
Login Success: Hello abcdeig!

```

注：在测试的时候，输入的密码与用户名一致


```

choose (0 or 1) :
0: sign in
1: login in
1
input name: john
input password: ****
Sign in first please!

choose (0 or 1) :
0: sign in
1: login in
0
input name: john
input password: ****
md5 password: 527BD5B5 D689E2C3 2AE974C6 229FF785

choose (0 or 1) :
0: sign in
1: login in
1
input name: john
input password: ****
md5 password: 527BD5B5 D689E2C3 2AE974C6 229FF785
Login Success: Hello john!

```



test.txt(存储用户信息)

```

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
-----
abcdefg|7AC66C0F 148DE951 9B8BD264 312C4D64
john|527BD5B5 D689E2C3 2AE974C6 229FF785

```

输入的密码长度没有限定

```

choose (0 or 1) :
0: sign in
1: login in
0
input name: vivian
input password: ****
md5 password: 8A43D6CD 08002255 BEAA5C9A 57A1A3E3

```

```

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
-----
abcdefg|7AC66C0F 148DE951 9B8BD264 312C4D64
john|527BD5B5 D689E2C3 2AE974C6 229FF785
vivian|8A43D6CD 08002255 BEAA5C9A 57A1A3E3

```

输入的密码字符也没有限定，比如下面的密码是一个空格


```
choose (0 or 1) :
0: sign in
1: login in
0
input name: space
input password: *
md5 password: 7215EE9C 7D9DC229 D2921A40 E899EC5F

choose (0 or 1) :
0: sign in
1: login in
1
input name: space
input password: *
md5 password: 7215EE9C 7D9DC229 D2921A40 E899EC5F
Login Success: Hello space!
```

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
abcdefg	7AC66C0F	148DE951	9B8BD264	312C4D64
john	527BD5B5	D689E2C3	2AE974C6	229FF785
vivian	8A43D6CD	08002255	BEAA5C9A	57A1A3E3
zero	7215EE9C	7D9DC229	D2921A40	E899EC5F
space	7215EE9C	7D9DC229	D2921A40	E899EC5F

用 md5 加密来验证密码，相同则登录成功。

3. 应用

MD5 被广泛用于加密和解密技术中，在很多操作系统（如 RedHat Linux）中，用户的密码是以 MD5（或其他类似算法）的方式保存的。用户在进行登录验证时，系统先把用户输入的密码计算成 MD5 值，然后再和系统中保存的初始密码的 MD5 值进行比较，使得系统在不知道用户密码的情况下就可以确定用户登录系统的合法性。不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度。

4. 实验小结

本次实验比较难的部分在于填充和初始化，因为使用的是 byte，所以扩充的数目的计算要特别注意，一不小心就会出错；填充的时候至少要填充一位，所以相当于是确定了至少填充一个 byte(10000000); 另外因为输入采用的是 string，

并将其转换为 char，但是在计算 A 的时候需要将 char 转换为 32 位 int；还有是在主循环中涉及的运算次数很多，总共有 L 个分组，并且每个分组有 4 轮迭代，每轮迭代有 16 次循环，这个过程中相关数据的下标容易出错，最终会导致整个计算结果的出错；在整个过程中涉及到大小端问题的处理是：将 message 由 char 转换为 uint32 和其他 uint32 进行运算，因为都是 uint32，即都是小端存储，所以整个过程是一致的，到最后得到最终结果 result，再将 result 转换为大端，即字节原本的表示顺序，并将大端存储的 int 转换为十六进制得到最后的输出结果。

在实现的时候，为了模拟密码输入不显示的情况，我使用了 `_getch()` 和 `_putch('*')`，并且将用户的相关数据保存到电脑的文件中，保存时密码采用 md5 格式进行保存，防止用户的密码被管理员窃取。

通过这次实验，加深了我对 MD5 算法整个过程的理解，并且掌握了 MD5 算法的具体实现，在实现过程中比较难也比较重要的是细节的处理，调试过程我采用打印输出每个阶段的结果，最终得以对输入的密码进行正确的加密。通过这次实验，除了掌握算法的具体实现外，更重要的是了解了 MD5 算法的具体应用，也解开了之前对 MD5 算法只能加密不能解密的应用的不解。

