中山大學
SUN YAT-SEN UNIVERSITY

《信息安全技术》

实验报告

(Assignment 2：MD5 算法)

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 软件工程 1 班
学 生 姓 名 ：	陆记
学　　号	17343080
时　　间 ：	2019　年　11　月　18　日

一、 算法原理概述

MD5 即 Message-Digest Algorithm 5（信息-摘要算法 5），在 90 年代初由 MIT 的计算机科学实验室和 RSA Data Security Inc 发明，经 MD2、MD3 和 MD4 发展而来，用于确保信息传输的完整性和一致性。

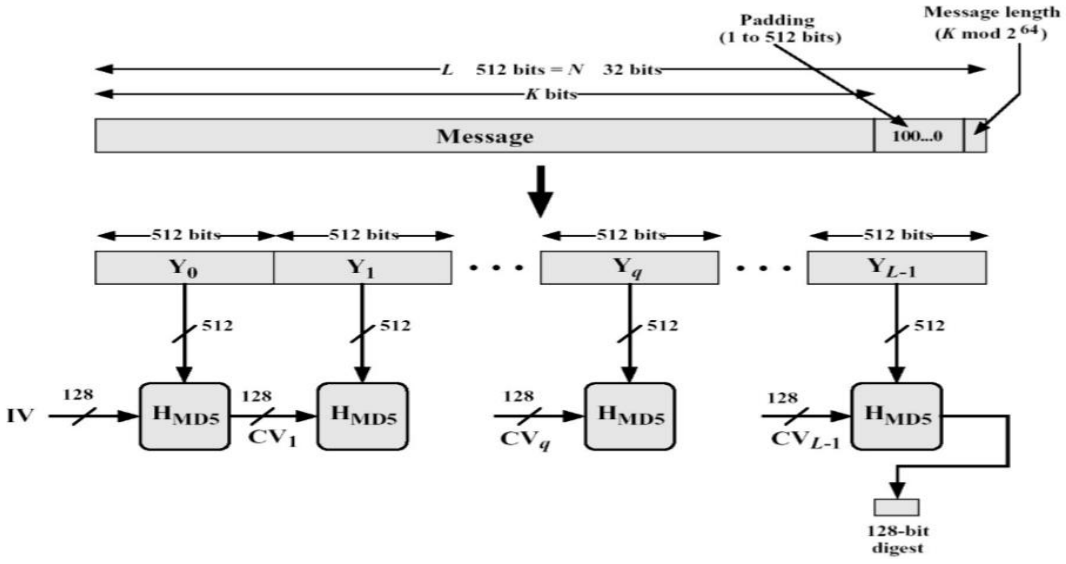
MD5 使用 little-endian(小端模式)，输入任意长度的信息，以 512-bit 进行分组。生成四个 32-bit 数据，最后联合输出固定长度(128-bit)的信息摘要。

MD5 是一个不可逆的字符串变换算法，因为从数学原理上来讲，经过 MD5 变换后的字符串逆推会有无穷多个字符串。尽管 Hans Dobbertin 在 1996 年找到了两个不同的 512-bit 块,它们 在 MD5 计算下产生相同的 hash 值。但至今还没有真正找到两个经过 MD5 变换后 hash 值相等的不同的消息，因此 MD5 算法目前为止是相对安全的。

MD5 算法的基本过程为：填充、分块、缓冲区初始化、循环压缩、得出结果。

二、 总体结构

总体流程图如下：



三、 模块分解

MD5 算法主要分为以下四个模块：

1、填充

因为 MD5 算法需要对消息进行 512-bit 分组，所以对于任意长度的输入消息字符串，首先要对其进行位扩充以保证其位数为 512-bit 的倍数（即 64-byte 的倍数，也即 16-word 的倍数）。

具体步骤：

1) 在长度为 K bits 的原始消息数据的末尾填充长度为 P bits 的标识符，注意 P 的范围为[1, 512],也就是说**至少要有 1 位**的填充,填充的内容为 1000…000(0x80…)。最终使得(P + K) mod 512 = 448，也就是让**(P + K + 64) mod 512 = 0.**

2) 在第 1) 步的基础上再在末尾填充 64 位，使得消息数据的位数为 512 的倍数，将该消息数据作为填充的结果。这一步填充的 64 位规定为 K 的低 64 位 Klow64，可以通过 **K mod 264** 得出。

2、分块

经过第 1 步填充后，将输入的数据的位数变为了 512 的倍数，接下来就将填充后的数据以 512 位为一组分为 L 组（Y0、Y1、Y1…YL-1）。

3、缓冲区初始化

此过程初始化一个 128-bit 的 MD(信息摘要)缓冲区，记为 CVq。表示为 4 个 32-bit(word)寄存器（A, B, C, D），最初的缓冲区记为：CV0＝IV。在缓冲区中会进行 L 次(组数)MD5 压缩迭代，最终得到的 CVL 即为输出的信息摘要（MD）。

我们知道，寄存器是按字节寻址的，即一个地址只能存储一个字节，而上面需要的缓冲区中的每一个寄存器都要存储 32-bit（即 4 个字节）的数据，所以每个数据块需要 4 地址存储。而字节的存储分为大端和小端，MD5 算法中用的是小端存储。即将低位字节放在内存的低地址端，高位字节放在内存的高地址端。

下面为此模块初始化的 4 个寄存器的 32-bits 数据（用 8 位 16 进制数，1 位表示 4-bits）及其在寄存器中的存储结果（从左到右地址增加）：

- *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

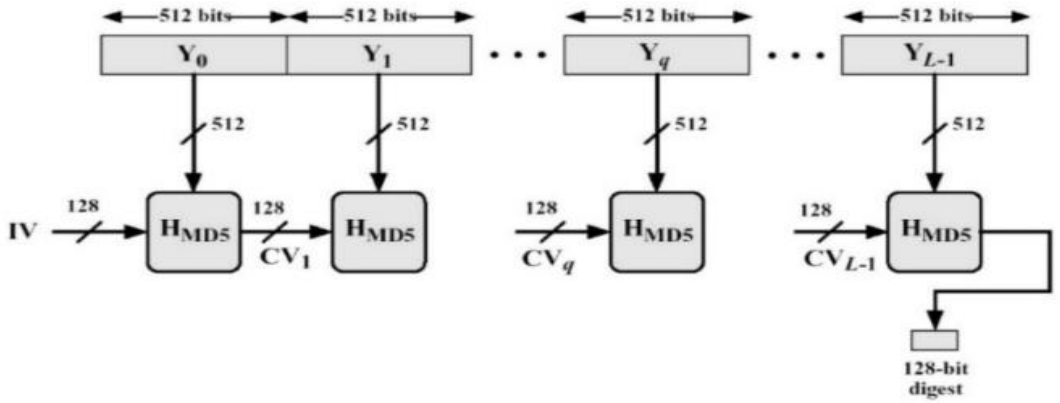
4、循环压缩

此模块为整个 MD5 算法的核心，即通过 MD5 压缩函数 H_{MD5} 对前面的 L 个分组的数据（512-bit）做 L 次压缩，每次压缩为 4 个循环，将 L 次压缩后得到的 CV_L（128-bit）作为最终的输出结果，表示为：

$$CV_0 = IV$$

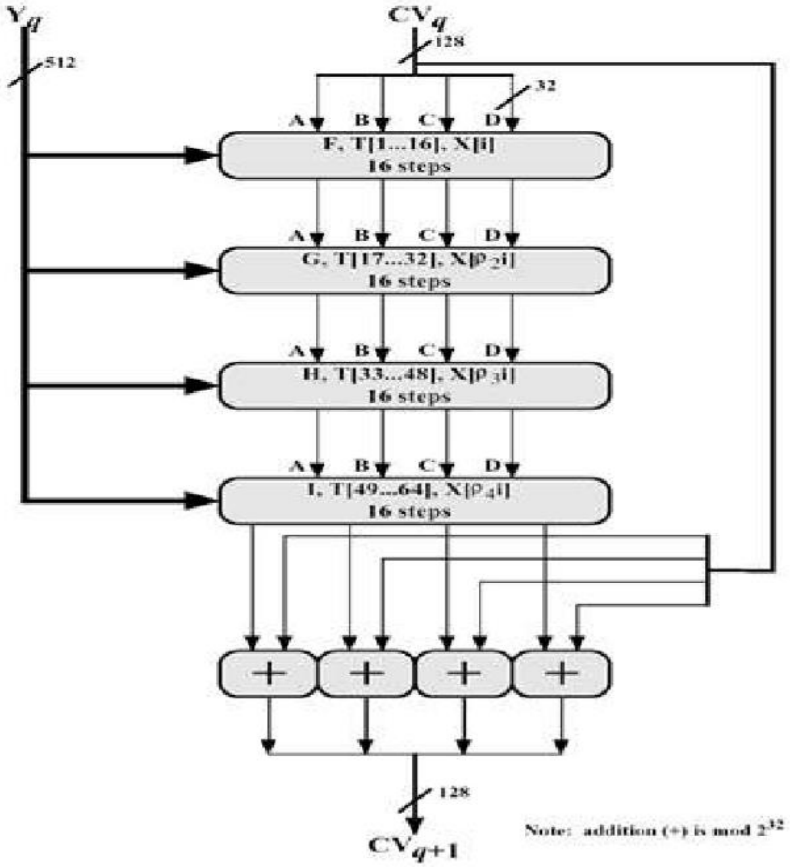
$$CV_i = H_{MD5}(CV_{i-1}, Y_{i-1})$$

即下图中的流程：



下面将介绍 MD5 算法的核心压缩函数 H_{MD5}

函数处理流程图如下：



H_{MD5} 函数先从缓冲区寄存器中获取初始的 128-bits CV_0 数据，再结合消息分组的 512-bits Y_0 数据，经过 4 轮循环（64 次迭代）的处理得到新的缓冲区 128-bits 数据 CV_1，用作下一次 H_{MD5} 函数处理。最终处理完 L 个分组的数据（到 Y_{L-1}）得到 CV_L 作为输出结果。

其中，每一轮循环分别使用不同的生成函数 F, G, H, I，结合指定的 T 表元素 T[]和从寄存器中获得的消息分组的不同部分 X[]做 16 次迭代运算，生成下一轮循环的输入，最终做完 4 次循环 64 次迭代的结果作为下一次 H_{MD5} 函数处理的输入。

4 轮循环的轮函数如下：

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

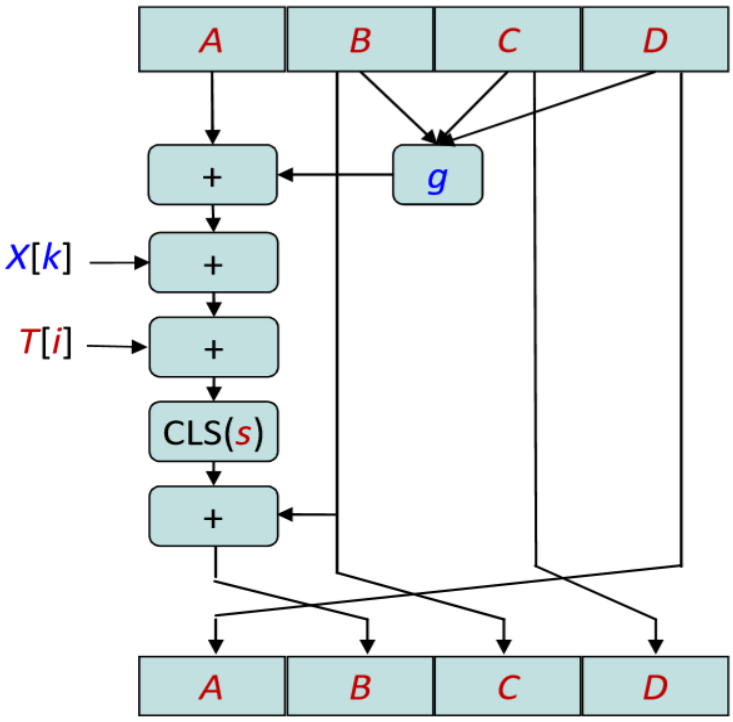
每次迭代（64 次）需要经过以下两个步骤：

1）对寄存器 A 中的数据迭代：a ← b + ((a+ g(b, c, d) + X[k] + T[i]) <<<s)

2）**对寄存器(A, B, C, D)中的值做循环右移轮换：**新 A 为原来的 D，新 B 为原来的 A，以此类推。

(A, B, C, D) -> (D, A, B, C)

图示为：



公式中各部分的解释：

① a, b, c, d：MD 缓冲区中（A, B, C, D）的当前值

② g：轮函数（根据轮次从 F, G, H, I 中选择）

③ X[k]：当前处理的消息分组 Y_q（512-bits）中的第 k 个 32-bit 字（k 的范围为 0-15），若将输入的整个消息数据（M）通过 32-bit 字节划分，则 $X[k] = M_{q\times 16+k}$（其中 16 为每个消息分组的 32-bit 字的数量）

④ T[i]：加法常数，T 表的第 i 个元素，为 32-bit 字。T 表一共有 64 个元素

⑤ +：模 2^{32} 加法

⑥ <<<s：将 32-bit 的数据循环左移 s 位

每轮循环要进行 16 次迭代（对输入消息 Y_q 的每一个 32-bit 字（共 16 个）X[k] 都要进行运算），第 i 次迭代（i = 1…16）的运算使用的 X[k]、T[i]以及移位 s 的规则如下（设 j = i － 1）：

第一轮循环：

轮函数 g 为 F;

k = j，16 轮迭代的 X[k]的使用顺序为：X[0]、X[1]、X[2]…X[15];

加法常数为 T[i]:

T[1.. 4] = { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
T[5.. 8] = { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
T[9..12] = { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
T[13..16] = { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }

移位规则为 s[i]:

s[1..16] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }

第二轮循环：

轮函数 g 为 G；

k = (1 + 5j) mod 16，16 轮迭代的 X[k]的使用顺序为：X[1]、X[6]、X[11]、X[0]、X[5]、X[10]、X[15]、X[4]、X[9]、X[14]、X[3]、X[8]、X[13]、X[2]、X[7]、X[12]；

加法常数为 T[i +16]：

T[17..20] = { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
T[21..24] = { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
T[25..28] = { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
T[29..32] = { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }

移位规则为 s[i + 16]：

s[17..32] = { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }

第三轮循环：

轮函数 g 为 H；

k = (5 + 3j) mod 16，16 轮迭代的 X[k]的使用顺序为：X[5]、X[8]、X[11]、X[14]、X[1]、X[4]、X[7]、X[10]、X[13]、X[0]、X[3]、X[6]、X[9]、X[12]、X[15]、X[2]；

加法常数为 T[i +2 * 16]：

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 }

移位规则为 s[i + 2 * 16]：

s[33..48] = { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }

第四轮循环：

轮函数 g 为 I；

k = 7j mod 16，16 轮迭代的 X[k]的使用顺序为：X[0]、X[7]、X[14]、X[5]、X[12]、X[3]、X[10]、X[1]、X[8]、X[15]、X[6]、X[13]、X[4]、X[11]、X[2]、X[9]；

加法常数为 T[i +3 * 16]：

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 }

移位规则为 s[i + 3 * 16]：

s[49..64] = { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }

（T 表的生成：T[i] = int(2^{32} × | sin(i) |）（其中 int 为取整函数，i 为弧度））

四、 数据结构

容器 p：用于储存填充后的明文（以 32-bit unsigned int 为单位），用于后面的迭代运算

```
/*
以32-bit字为单位存储的明文信息vector
每组为512-bits = 16 * 32-bits
组数为L
*/
vector<unsigned int> p(L * 16);
```

数组 X[]：16 位无符号整型数组，以 32-bit 为单位（unsigned int）临时储存分组后的每组数据(512-bit)，用于后面的迭代运算

```
unsigned int X[16]; //每一组512-bit数据以32-bit字为单位分为16组存在X中
```

T 表：64 位无符号整型（32-bit）数组（数组元素用 8 位 16 进制数表示），作为 64 次迭代运算的加法常数

```
//T表：64个32-bit字(unsigned int),用16进制数表示
const unsigned int T[64] = {
    0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
    0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
    0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
    0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
    0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
    0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
    0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
    0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
    0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
    0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
    0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
    0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
    0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
    0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
    0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
    0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391
};
```

S 表：64 位无符号整型数组，为 64 次迭代循环左移位数的规则

```
//S表：64次迭代循环左移位数表
const unsigned int s[64] = {
    7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
    5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
    4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
    6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21
};
```

X[k]表：64 位无符号整型数组，为 64 次迭代用的每组输入 512-bit 数据第 k 个字的索引

```
//X[k]表：64次迭代用的每组数据的第k个字的索引表
const unsigned int k[64] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
    1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12,
    5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2,
    0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9
};
```

Hex[]：字符串数组，用于将 32-bit 的 unsigned int 转换为 16 进制字符串

```
const char Hex[] = "0123456789abcdef";
```

五、 C++源代码

(github： https://github.com/luji17343080/Information-security-technology/tree/master/MD5)

基本变量声明及初始化

```
//缓冲区寄存器
unsigned int A = 0x67452301;
unsigned int B = 0xefcdab89;
unsigned int C = 0x98badcfe;
unsigned int D = 0x10325476;

int L; //组数
string plain = "";  //明文
string cipher;  //密文
```

每组数据 4 轮循环的轮函数（F、G、H、I）： 每个函数的传入参数都是当前的 B、C、D 缓冲区寄存器的值

```
/* 第一轮循环函数F */
unsigned int F(unsigned int b, unsigned int c, unsigned int d) {
    return (b & c) | ((~b) & d);
}
/* 第二轮循环函数G */
unsigned int G(unsigned int b, unsigned int c, unsigned int d) {
    return (b & d) | (c & (~d));
}
/* 第三轮循环函数H */
unsigned int H(unsigned int b, unsigned int c, unsigned int d) {
    return b ^ c ^ d;
}
/* 第四轮循环函数I */
unsigned int I(unsigned int b, unsigned int c, unsigned int d) {
    return c ^ (b | (~d));
}
```

填充分组函数 padding： 传入输入的字符串，修改填充明文容器 p

```
/* 填充分组 */
void padding(string s) {
    /*
    512-bit为一组，每一个字符为8-bits
    字符串的位数为字符串长度乘以8
    填充后的字符串分为三部分：原来的K-bits + 初填充的P-bits + 64 bits
    最后将字符串以32-bit字为单位（unsigned int）储存在明文容器p中
    */
    int bitCount = (s.length() * 8 + 64) % 512 == 0 ? 512 : 448 - (s.length() * 8) % 512; //最开始填充的P的位数
    L = ((bitCount + s.length() * 8) + 64) / 512; //组数为填充后的总位数除以512
    p.resize(L * 16); //需要重新定义p的大小
    for (int i = 0; i < s.length(); i++) {
        p[i >> 2] |= (int)(s[i]) << ((i % 4) * 8); //1个unsigned int对应4个char
    }
    /* 填充1000...000 */
    p[s.length() >> 2] |= 0x80 << ((s.length() % 4) * 8);

    /* 填充后64位 */
    p[p.size() - 2] = s.length() * 8;
}
```

循环压缩函数 cyclic_compress（在次之前先写一个循环左移位的函数 left_shift）：每一次迭代根据公式修改寄存器 A 的值，然后将寄存器中的数据循环右移（具体前面已经阐述）

```
/* 循环左移位函数 */
unsigned int left_shift(unsigned int num, unsigned int digits) {
    return (num << digits) | (num >> (32 - digits));
}
/* 循环压缩 */
void cyclic_compress(unsigned int *X) {
    unsigned int a = A;
    unsigned int b = B;
    unsigned int c = C;
    unsigned int d = D;
    unsigned int tmp = 0;

    /* 4轮循环，64次迭代 */
    for (int i = 0; i < 64; i++) {
        unsigned int g[4] = { F(b, c, d), G(b, c, d), H(b, c, d), I(b, c, d) }; //4轮循环分别使用的轮函数g

        /* 缓冲区循环右移轮换 */
        tmp = d;
        d = c;
        c = b;
        b = b + left_shift(a + g[i / 16] + X[k[i]] + T[i], s[i]);
        a = tmp;
    }
    A += a;
    B += b;
    C += c;
    D += d;
}
```

加密函数 encrypt：先对输入明文字符串进行 padding，再执行 L 次（数据组数）循环压缩函数。在执行循环压缩之前，要对每组数据以 32-bit 为单位存入无符号整型数组 X 中。最后调用 Unit32toHexStr 函数将缓冲区寄存器 A、B、C、D 里的无符号整型数（8 位 16 进制表示）转换为字符串，将字符串依次相加即为密文。

```
/* 32-bit无符号整型转变为8位16进制字符串 */
string Uint32toHexStr(unsigned int num) {
    string result = "";
    for (int i = 0; i < 4; i++) {
        string tmp = "";
        unsigned int hex_ = (num >> (i * 8)) % (1 << 8) & 0xff; //16进制转换
        for (int j = 0; j < 2; j++) {
            tmp = Hex[hex_ % 16] + tmp; //char转string,注意加的顺序不能相反
            hex_ /= 16;
        }
        result += tmp;
    }
    return result;
}

/* 加密函数 */
void encrypt(string s) {
    padding(s); //填充
    /* 对每个分组的数据（共L组）进行4次循环（64次迭代）压缩，结果作为下一次迭代的输入 */
    for (unsigned int i = 0; i < L; i++) {
        for (int j = 0; j < 16; j++) {
            X[j] = p[i * 16 + j];
        }
        cyclic_compress(X);  //循环压缩
    }
    cipher = Uint32toHexStr(A) + Uint32toHexStr(B) + Uint32toHexStr(C) + Uint32toHexStr(D); //密文为32位16进制字符串
}
```

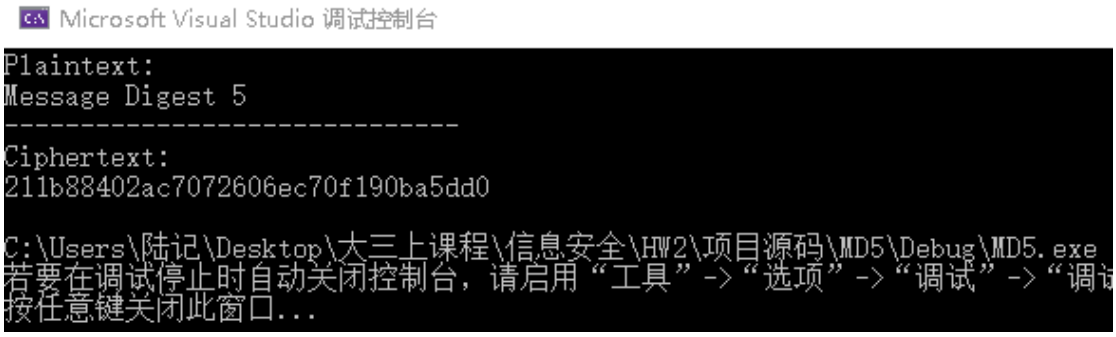
main 函数：在终端输入明文、输出密文，并将明文和密文写入指定文件中

```
int main() {
    ofstream oi, oo;    //创建文件输出流对象
    oi.open("plain.txt");
    oo.open("cipher.txt");
    cout << "Plaintext: " << endl;
    getline(cin, plain);
    oi << plain;   //将明文写入plain.txt中
    encrypt(plain);
    cout <<"------------------------------"<< endl << "Ciphertext: " << endl << cipher << endl;
    oo << cipher;   //将密文写入cipher.txt中
    oi.close();
    oo.close();
}
```

六、 编译运行结果

Test1

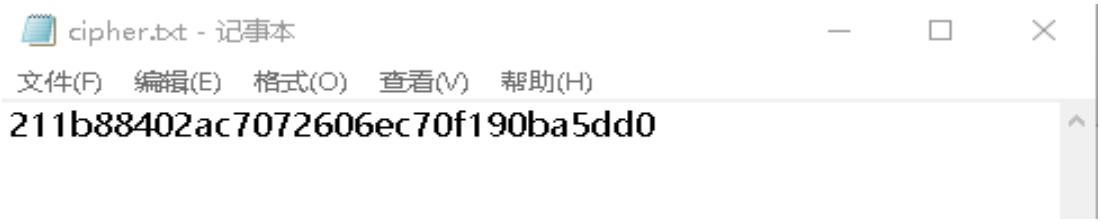
控制台结果：



在线 MD5 加密测试结果：

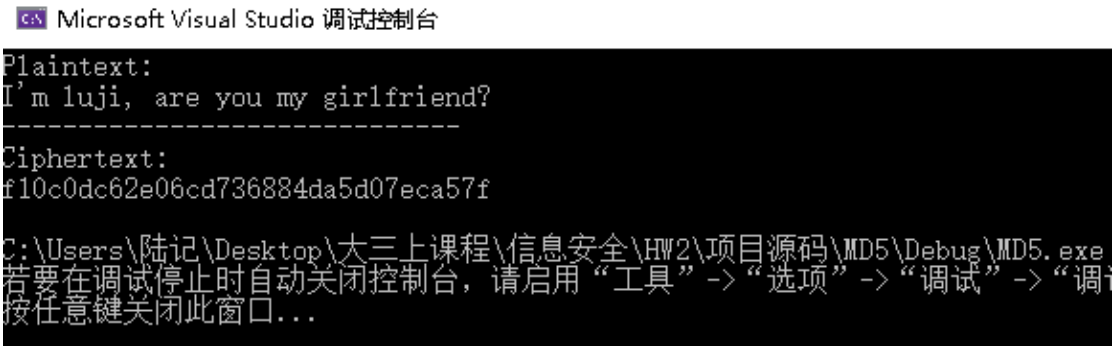
plain.txt：

cipher.txt：



Test2

控制台结果：



在线 MD5 加密测试结果：

plain.txt：

cipher.txt：