

**综合实验报告**

( 2020 -- 2021 年度第 一 学期)

名 称： 计算机密码学综合实验

题 目： RC4、AES、MD5的编写

院 系： 控制与计算机工程学院计算机系

班 级： 信安1802

学 号： 201809000118

学生姓名： 王楚璇

指导教师： 胡朝举、李莉

设计周数： 1周

成 绩：

日期： 2020年 12 月 10 日

**一、 目的与要求**

《计算机密码学》是一门理论和实践性均非常强的信息安全专业骨干课，是信息安全和网络安全的基础。而《计算机密码学综合实验》主要针对《计算机密码学》课上实践环节的不足而开设的综合实验，是课程的继续的提高。密码学的基础内容包括以下几个面：流密码、分组密码、公钥密码体制及消息认证、消息摘要及数字签字及与之相关的技术、安全协议等。由于公钥密码体制涉及较复杂的大整数或椭圆曲线的算法，在程序设计方面存在着非常大的困难，要实现实用意义下的算法，一般须借助于第三方库或各种高级语言所提供的相应库函数（或类）来实现。考虑到本科阶段的密码学学习，必须要知其然也要知其所以然，而一旦知其密码学的安全原理，在实践中结合相应的库函数，一般均能够很好地进行应用，但前提是对密码学的原理有很好地认识。本实验就是要解决此问题。

为此，本实验共两部分内容：

1. 根据我校《信息安全综合实验系统》的实训系统来学习密码学的算法，领会密码学的主要内容：

(1) DES单步加密实验;(2) DES算法实验;(3) 3DES算法实验;(4)AES算法实验;(5)MD5算法实验 ;(6)SHA-1算法实验;(7)RSA算法实验;(8)DSA数字签名实验;

2. 编程实践：对密码学的关键算法流密码、分组密码以及数字摘要MD5进行编程实现。

**二、 主要内容**

实验一 《信息安全综合实验系统》平台实验：

以附件 《信息安全综合实验系统》实验指导书的要求，进行实验，并写出相应的实验报告；

实验二 程序设计

对RC4,128位AES（或DES）,MD5进行编程并进行类封装，将此3个类整合到一个window程序中进行演示和运行。对于windows程序设计能力较强的同学，可以考虑图形界面。

**三、 进度计划**

1. 平台实验必须在相应的实验环境中进行，请记录完整的实验过程；

2. 星期一至星期五每天上午8:30-11:30平台或编程，下午时间不做具体要求。编程设计请灵活掌握。

**四、 实验成果要求**

1. 要求程序能正常运行，并实现任务书要求功能。

2. 完成实验报告，要求格式规范，内容具体而翔实，应体现自身所作的工作，注重对设计思路的归纳和对问题解决过程的总结。

3. 程序设计必须完成，且通过对任意指定的文件进行加、解密处理。

4. 坚决杜绝抄袭行为，一经发现，按不及格处理。

**五、 考核方式**

平时成绩+程序验收+实验报告。

学生姓名：

指导教师： 胡朝举、李莉

2020 年 11 月 10 日

**实验一 RC4 的加密与解密**

1. **实验目的及要求**

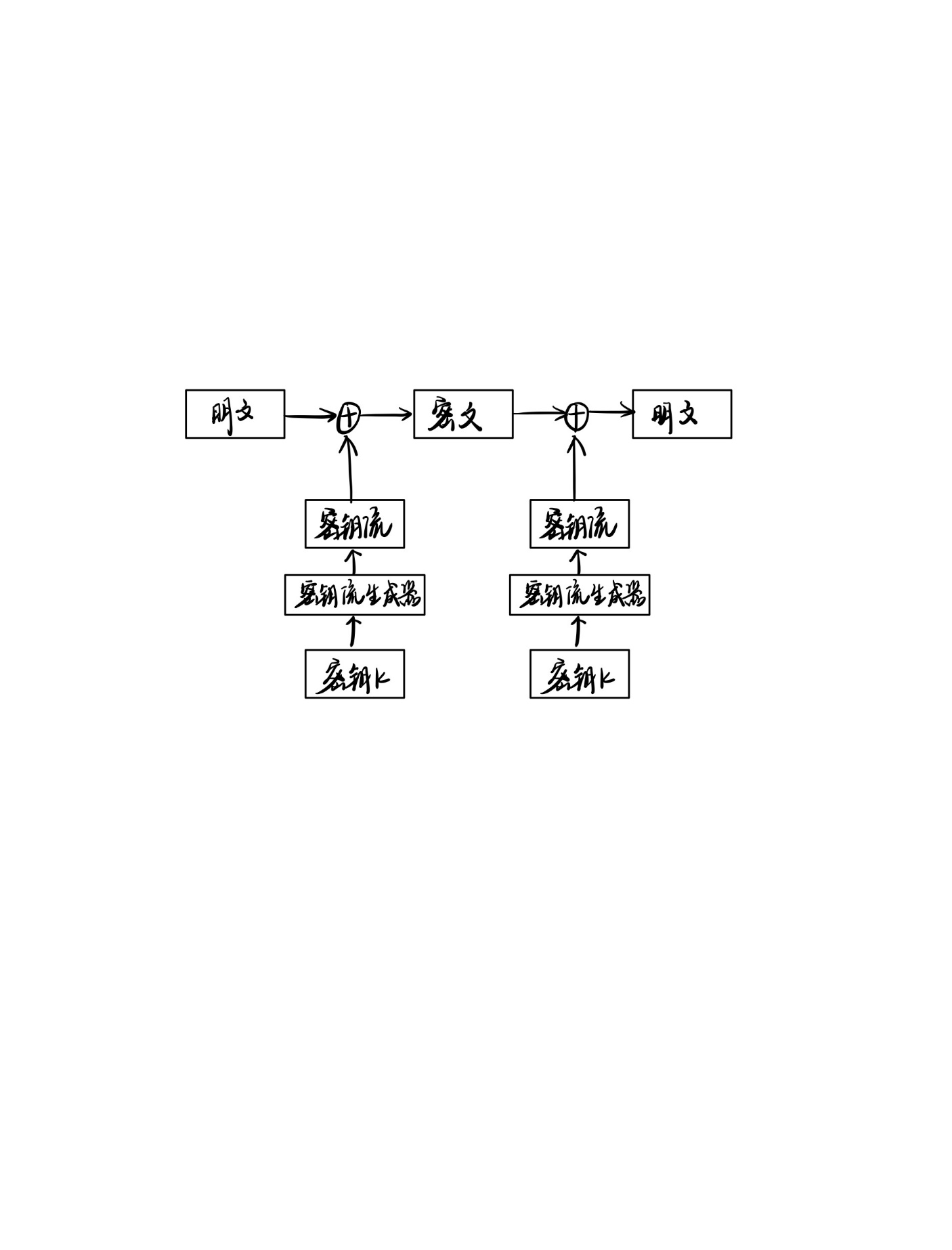
实现 RC4 的加密与解密。

1. **实验正文**

RC4 于 1987 年提出，其特点是算法简单，运行速度快，而且密钥长度是可变的，可变范围为 1-256 字节。主要由初始化算法（KSA）和伪随机子密码生成算法（PRGA）。几个主要关键变量为：

1. 状态向量 S：长度 256 字节，用于生成密钥流

2. 密钥K：长度为 1-256 字节，用于生成密钥流



typedef unsigned char BYTE;

class RC4

{

public:

RC4(char\* key, int keylen);

void setkey(char\* key, int keylen);

void getkey(char\* output,char\* data, int len);

void Encrypt(char\* cc,char\* mm,int len);

void Decrypt(char\* cc, char\* mm, int len);

void initial(char\* key, int keylen);

private:

BYTE s[256],k[256];

};

RC4::RC4(char\* key, int keylen)

{

this->setkey(key, keylen);

}

void RC4::setkey(char\* key, int keylen)

{

for (int i = 0; i < 256; i++)

{

k[i] = key[i % keylen];

}

for (int i = 0; i < 256; i++)

{

s[i] = i;

}

int j = 0;

for (int i = 0; i < 256; i++)

{

j = (j + s[i] + k[i]) % 256;

swap(s[i], s[j]);

}

}

//生成密钥流

void RC4::getkey(char\* cc, char\* mm, int len)

{

int i = 0, j = 0;

for (int k = 0; k < len; k++)

{

i = (i + 1) % 256;

j = (j + s[i]) % 256;

swap(s[i], s[j]);

BYTE key = s[(s[i] + s[j]) % 256];

cc[k] = key ^ mm[k];

}

}

void RC4::Encrypt(char\* cc, char\* mm, int len)

{

this->getkey(cc, mm, len);

}

void RC4::Decrypt(char\* cc, char\* mm, int len)

{

this->getkey(cc, mm, len);

}

void RC4::initial(char\* key, int keylen)//重设

{

for (int i = 0; i < 256; i++)

{

s[i] = i;

k[i] = key[i % keylen];

}

int j = 0;

for (int i = 0; i < 256; i++)

{

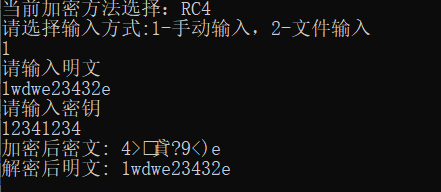
j = (j + s[i] + k[i]) % 256;

swap(s[i], s[j]);

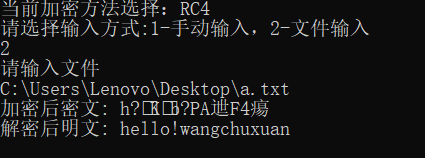
}

}

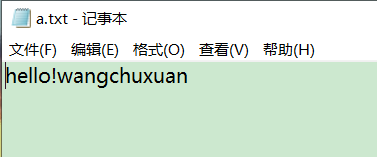
控制台输入结果:



文件输入结果:



文件：



1. **实验总结**

RC4实验较为简单，它主要考察我们对算法的理解和掌握。一开始不理解为何每次加密、解密都需要初始化密钥，在实现过程中，知道 S-box 需要在需要在初始状态被密钥打乱，生成密钥流来加密。这是我第一次通过读文件来加密，知道了对于文件读取的知识。还有要注意各数组的类型char或是BYTE，这为以后的实验打下基础。

**实验二 128 位 AES 的加密和解密**

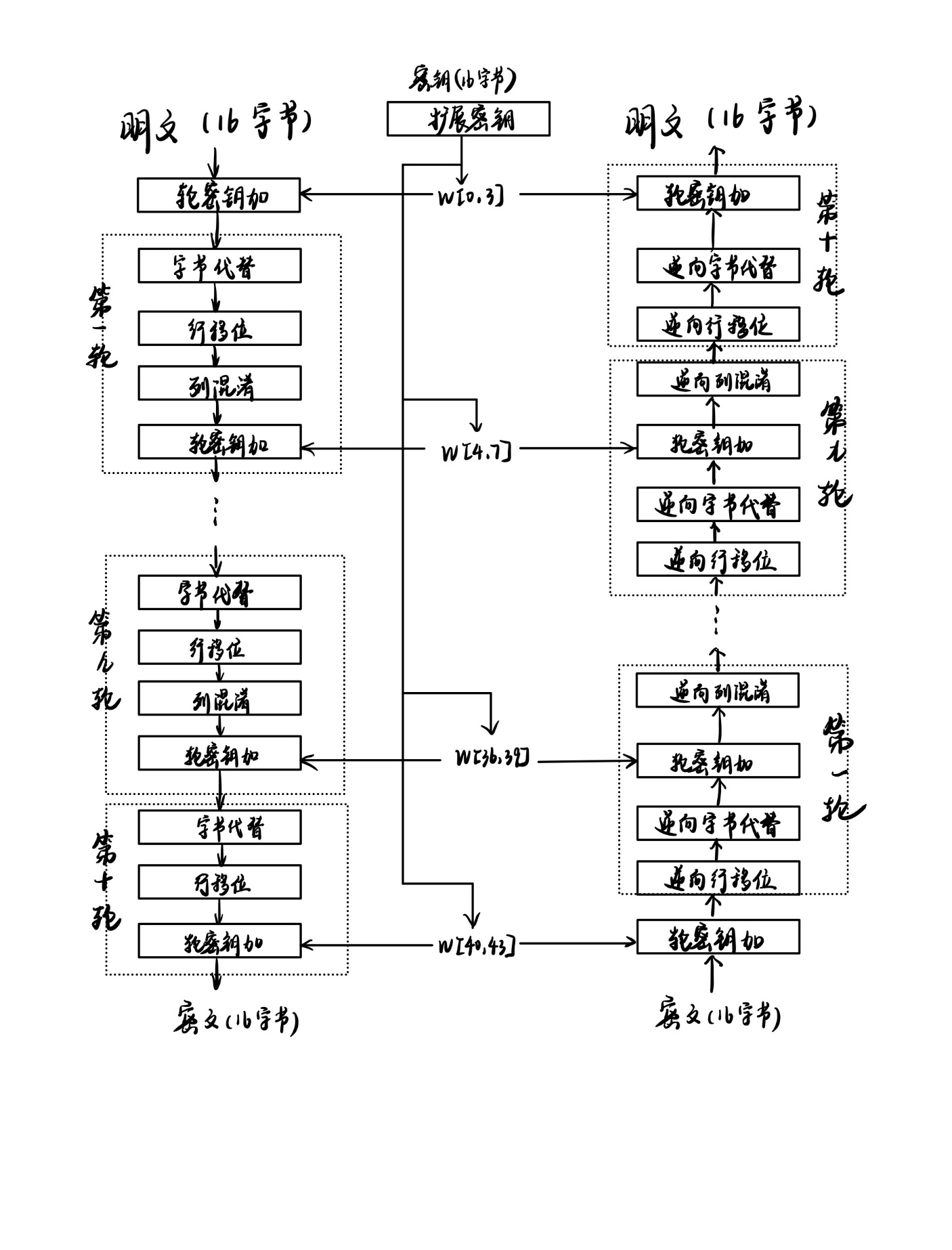
**一、实验目的及要求**

1. 实现 128 位 AES 对明文的加密解密。

2. 实现在不同工作模式（ECB、CBC、OFB）下对明文的加密解密。

**二、实验正文**

AES为分组密码，把明文分成一组一组的，每组长度相等，每次加密一组数据，直到加密完整个明文。本实验中，每个分组为16个字节，数据分组长度128bit，密钥长度为128bit，加密轮数为10轮。



定义了一个由四个BYTE(unsigned char)组成的结构体word:

struct word

{

BYTE bytechar[4];

};

创建了AES类：

class AES

{

public:

AES(BYTE\* key);

~AES() {};

int ECBEncrypt(word\* c, BYTE\* m, int length);

int ECBDecrypt(word\* c, BYTE\* m, int length);

int CBCEncrypt(word\* c, BYTE\* m, int length);

int CBCDecrypt(word\* c, BYTE\* m, int length);

int OFBEncrypt(word\* c, BYTE\* m, int length);

void setkey(BYTE\* k);

void setplain(BYTE\* p);

void Encrypt\_1(word\* c, word\* m);

void Decrypt\_1(word\* c, word\* m);

word rotWord(word w);

word subWord(word w);

word wordXor(word w1, word w2);

word plain[4];

word Rcon[11];

void setr(word\* c);

private:

void SubBytes(word\* state);

void InvSubBytes(word\* state);

void ShiftRows(word\* state);

void InvShiftRows(word\* state);

void MixColumns(word\* state);

void InvMixColumns(word\* state);

void ExpandKey(BYTE\* key,word\* w);

void RoundKeyAdd(word\* state,int round);

BYTE iv[16] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };

BYTE cipherkey[16];

word cipher[4];

word wkey[44];

};

void AES::SubBytes(word\* state)

{

BYTE l, r;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

l = state[i].bytechar[j] >> 4;

r = state[i].bytechar[j] & 0x0F;

state[i].bytechar[j] = Sbox[l][r];

}

}

}

void AES::InvSubBytes(word\* state)

{

BYTE l, r;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

l = state[i].bytechar[j] >> 4;

r = state[i].bytechar[j] & 0x0F;

state[i].bytechar[j] = InvSbox[l][r];

}

}

}

void AES::RoundKeyAdd(word\* state, int round)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

state[i].bytechar[j] ^= wkey[4 \* round + i].bytechar[j];

}

}

//cout << state[0].bytechar[0];

}

void AES::ShiftRows(word\* state)

{

BYTE temp;

//左移1

temp = state[0].bytechar[1];

state[0].bytechar[1] = state[1].bytechar[1];

state[1].bytechar[1] = state[2].bytechar[1];

state[2].bytechar[1] = state[3].bytechar[1];

state[3].bytechar[1] = temp;

//左移2

temp = state[0].bytechar[2];

state[0].bytechar[2] = state[2].bytechar[2];

state[2].bytechar[2] = temp;

temp = state[1].bytechar[2];

state[1].bytechar[2] = state[3].bytechar[2];

state[3].bytechar[2] = temp;

//左移3

temp = state[0].bytechar[3];

state[0].bytechar[3] = state[3].bytechar[3];

state[3].bytechar[3] = state[2].bytechar[3];

state[2].bytechar[3] = state[1].bytechar[3];

state[1].bytechar[3] = temp;

}

void AES::InvShiftRows(word\* state)

{

BYTE temp;

//右移1

temp = state[0].bytechar[1];

state[0].bytechar[1] = state[3].bytechar[1];

state[3].bytechar[1] = state[2].bytechar[1];

state[2].bytechar[1] = state[1].bytechar[1];

state[1].bytechar[1] = temp;

//右移2

temp = state[0].bytechar[2];

state[0].bytechar[2] = state[2].bytechar[2];

state[2].bytechar[2] = temp;

temp = state[1].bytechar[2];

state[1].bytechar[2] = state[3].bytechar[2];

state[3].bytechar[2] = temp;

//右移3

temp = state[0].bytechar[3];

state[0].bytechar[3] = state[1].bytechar[3];

state[1].bytechar[3] = state[2].bytechar[3];

state[2].bytechar[3] = state[3].bytechar[3];

state[3].bytechar[3] = temp;

}

void AES::MixColumns(word\* state)

{

word result[4];

for (int i = 0; i < 4; i++)

{

result[i].bytechar[0] = xtime2(state[i].bytechar[0]) ^ xtime3(state[i].bytechar[1]) ^ xtime1(state[i].bytechar[2]) ^ xtime1(state[i].bytechar[3]);

result[i].bytechar[1] = xtime1(state[i].bytechar[0]) ^ xtime2(state[i].bytechar[1]) ^ xtime3(state[i].bytechar[2]) ^ xtime1(state[i].bytechar[3]);

result[i].bytechar[2] = xtime1(state[i].bytechar[0]) ^ xtime1(state[i].bytechar[1]) ^ xtime2(state[i].bytechar[2]) ^ xtime3(state[i].bytechar[3]);

result[i].bytechar[3] = xtime3(state[i].bytechar[0]) ^ xtime1(state[i].bytechar[1]) ^ xtime1(state[i].bytechar[2]) ^ xtime2(state[i].bytechar[3]);

}

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

state[i].bytechar[j] = result[i].bytechar[j];

}

}

}

void AES::InvMixColumns(word\* state)

{

word result[4];

for (int i = 0; i < 4; i++)

{

result[i].bytechar[0] = xtimee(state[i].bytechar[0]) ^ xtimeb(state[i].bytechar[1]) ^ xtimed(state[i].bytechar[2]) ^ xtime9(state[i].bytechar[3]);

result[i].bytechar[1] = xtime9(state[i].bytechar[0]) ^ xtimee(state[i].bytechar[1]) ^ xtimeb(state[i].bytechar[2]) ^ xtimed(state[i].bytechar[3]);

result[i].bytechar[2] = xtimed(state[i].bytechar[0]) ^ xtime9(state[i].bytechar[1]) ^ xtimee(state[i].bytechar[2]) ^ xtimeb(state[i].bytechar[3]);

result[i].bytechar[3] = xtimeb(state[i].bytechar[0]) ^ xtimed(state[i].bytechar[1]) ^ xtime9(state[i].bytechar[2]) ^ xtimee(state[i].bytechar[3]);

}

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

state[i].bytechar[j] = result[i].bytechar[j];

}

}

}

void AES::ExpandKey(BYTE\* key, word\* w)

{

word temp;

for (int i = 0; i < Nk; i++)

{

for (int j = 0; j < 4; j++)

{

w[i].bytechar[j] = key[4 \* i + j];

}

}

for (int i = Nk; i < keylen; i++)

{

temp = w[i - 1];

if ((i % Nk) == 0)

{

temp = wordXor(subWord(rotWord(temp)),Rcon[i/Nk]);

}

for (int j = 0; j < 4; j++)

{

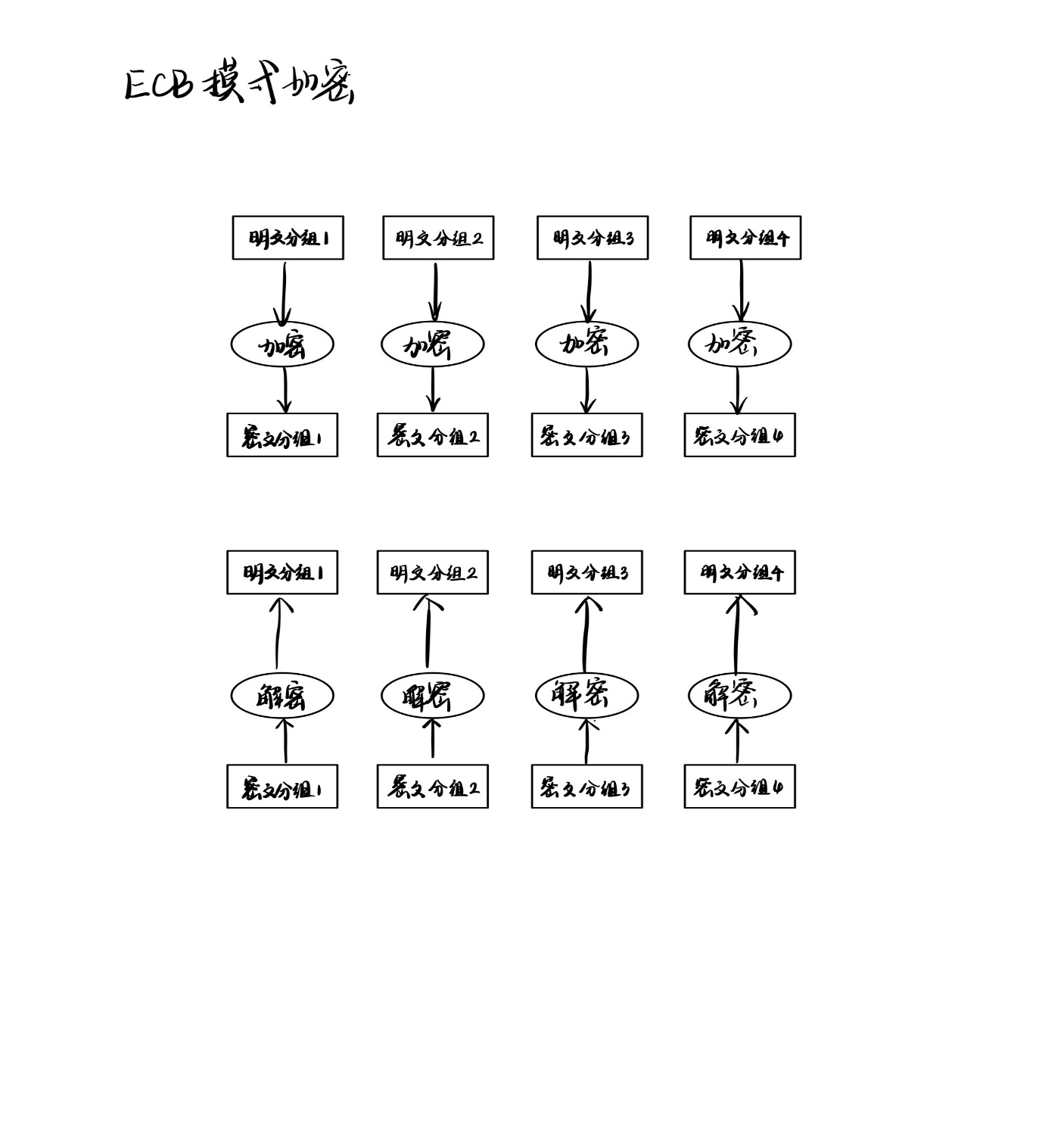
w[i].bytechar[j] = w[i - Nk].bytechar[j] ^ temp.bytechar[j];

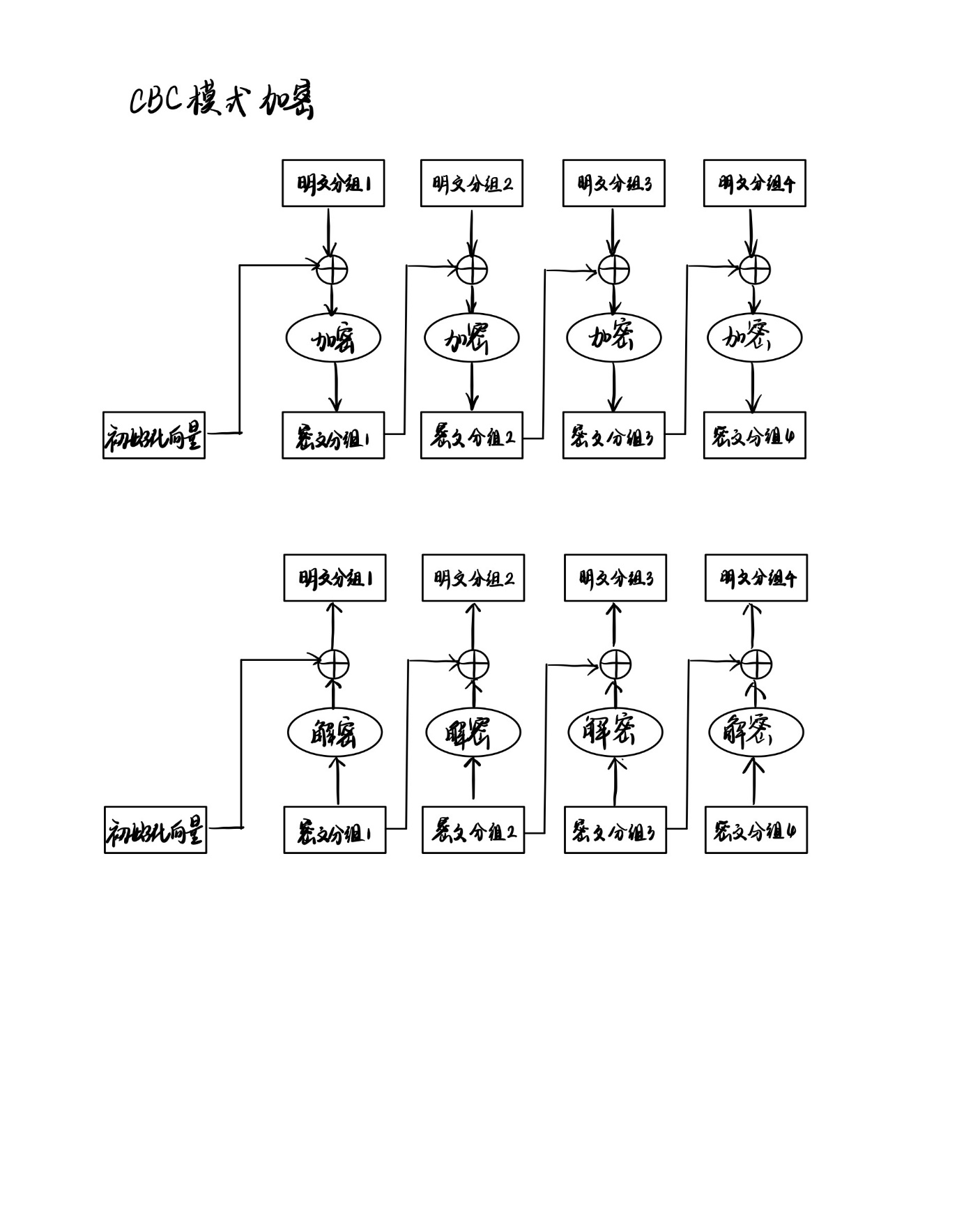
}

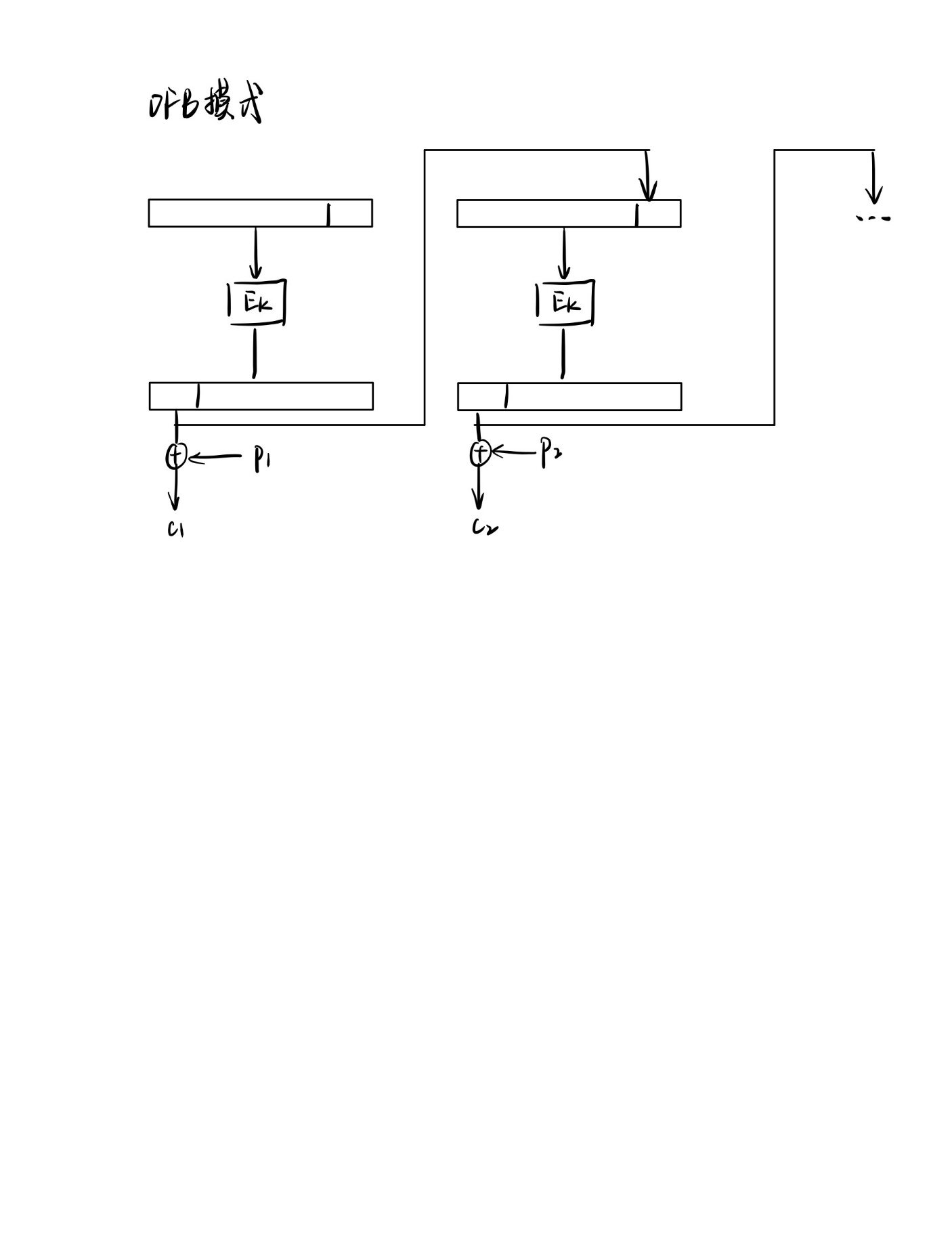
}

}

实验中采用ECB，CBC，OFB三种模式，其相应的工作过程如图所示：

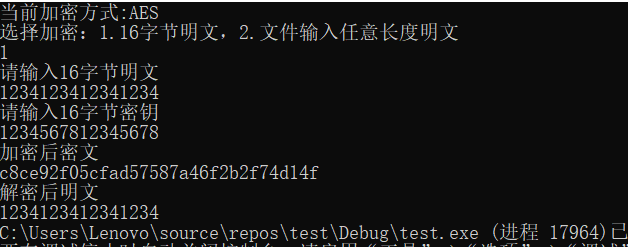






实验结果：

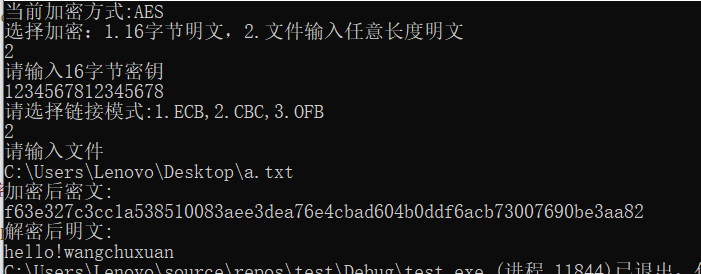
Encrypt方式加密结果：

****

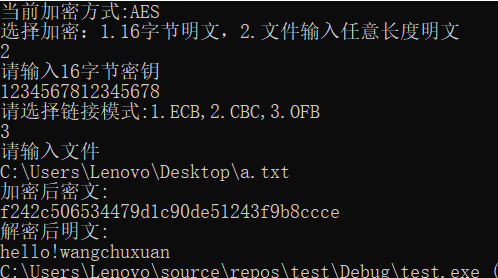
EncryptFileECB方式加密结果：

****

EncryptFileCBC方式加密结果：

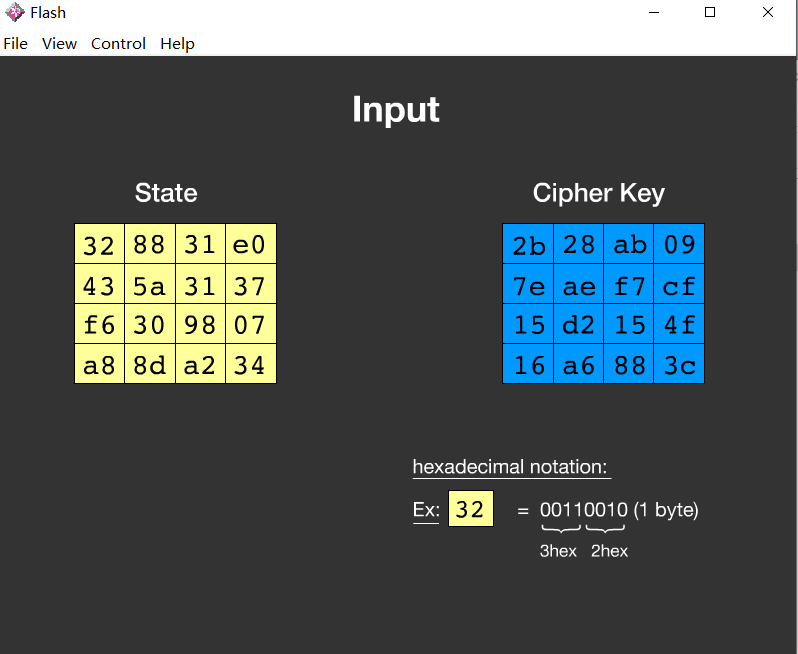
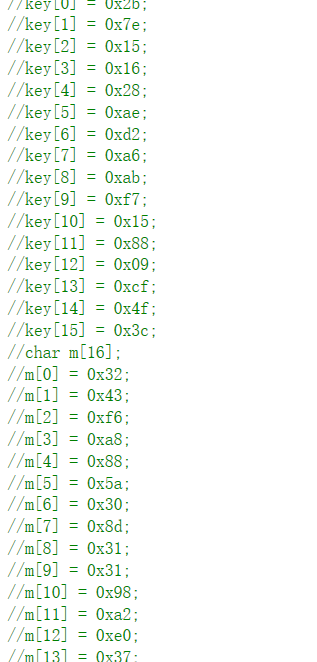
****

EncryptFileOFB方式加密结果：



**三、心得体会**

AES实验比RC4难度更大，复杂程度更高，在一开始写的时候我一头雾水，完全不知道怎么下手，幸好我去图书馆借阅了一些有关密码学实现的书籍，我参考书中所给的形式来着手编写AES，其中我还参考老师给的代码，发现相同的函数，有些书中的实现形式较为繁琐，而老师上课所讲解的伪代码较易于理解，我先编写块加密函数Encrypt\_1，在发现能对一个分组正确加密后再编写后续的几种链接模式，在编写过程中我发现该函数的编写十分难以验证，对于一个错误不知道他是在哪里出错了，这让我很头大，但是我发现老师给的资料里有一个Rijndael.exe的flash文件，里面详细给出了样例是如何一步一步加密成功的，我就通过这个，先给自己的测试函数赋值成样例中所给的值，然后再跟着flash演示进行单步调试，最后发现错误所在。虽然可能是个笨办法，但是最终还是成功帮我解决了问题，所以在以后我要多多查阅资料来帮助我更好的学习。



**实验三 MD5 数字摘要**

1. **实验目的及要求**

实现 MD5 对字符串和文件的摘要。

1. **实验正文**

MD5 算法分为如下步骤：

1. 初始化。在 MD5 算法中，首先需要对信息进行填充，这个数据按位(bit)补充，要求最终的位数对 512 求模的结果为 448。也就是说数据补位后，其位数长度只差 64 位(bit)就是512 的整数倍。即便是这个数据的位数对 512 求模的结果正好是 448 也必须进行补位。

2. 扩展长度。在完成补位工作后，又将一个表示数据原始长度的 64 bit 数(这是对原始数据没有补位前长度的描述，用二进制来表示)补在最后。当完成补位及补充数据的描述后，得到的结果数据长度正好是 512 的整数倍。也就是说长度正好是 16 个(32bit) 字的整数倍。

3. 初始化 MD 缓存器。MD5 运算要用到一个 128 位的 MD5 缓存器，用来保存中间变量和最终结果。该缓存器又可看成是 4 个 32 位的寄存器 A、B、C、D，初始化为:

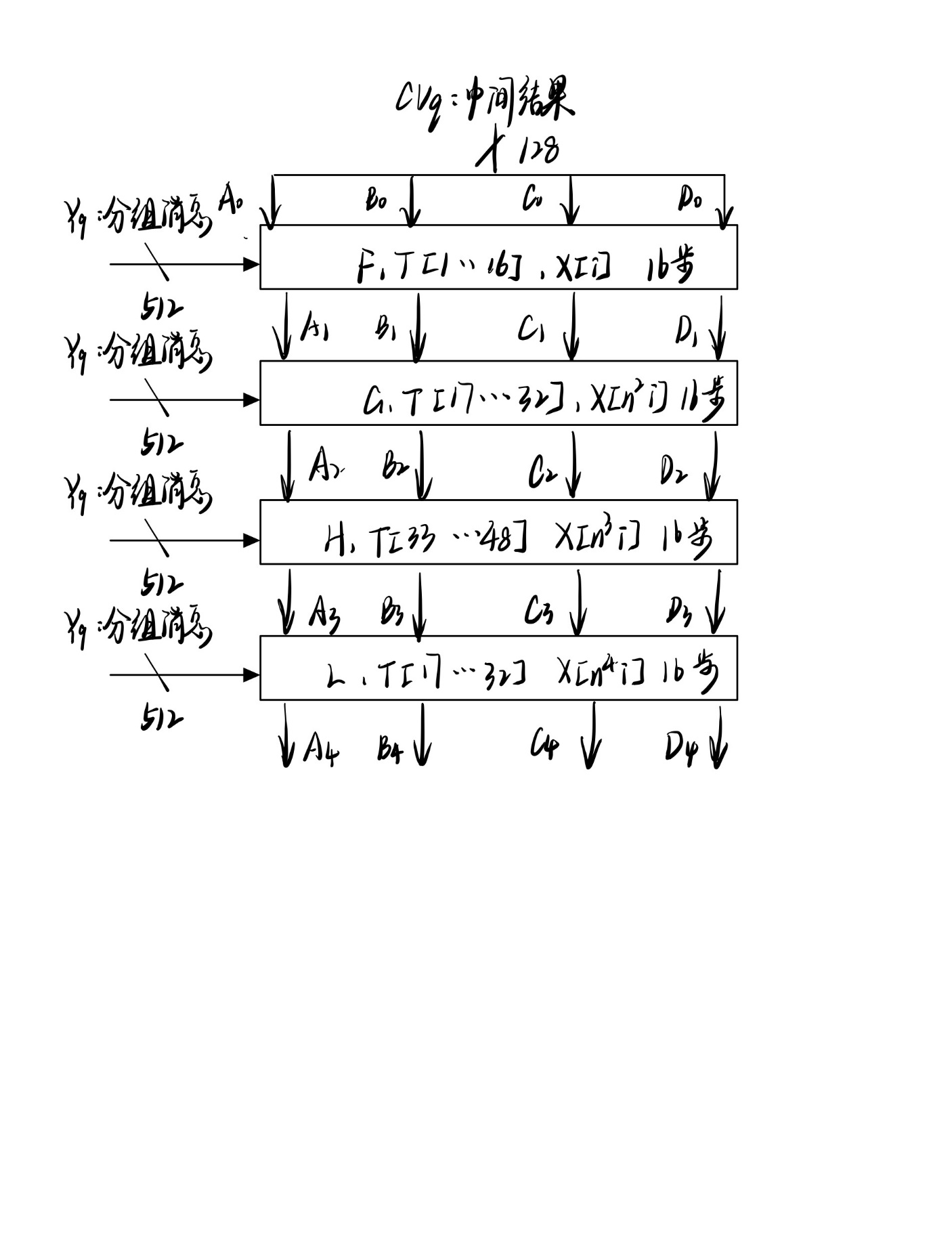
A:01 23 45 67

B:89 ab cd ef

C:fe dc ba 98

D:76 54 32 10

4. 处理数据段。首先定义 4 个非线性函数 F、G、H、I，对输入的报文运算以 512 位数据段为单位进行处理。对每个数据段都要进行 4 轮的逻辑处理，在 4 轮中分别使用 4 个不同的函数 F、G、H、I。每一轮以 ABCD 和当前的 512 位的块为输入，处理后送入 ABCD(128位)。



5. 输出。信息摘要最终处理成以 A, B, C, D 的形式输出。也就是开始于 A 的低位在前的顺序字节，结束于 D 的高位在前的顺序字节。

class MD5

{

public:

MD5();

void initial(Context\* context);

DWORD F(DWORD x, DWORD y, DWORD z);

DWORD G(DWORD x, DWORD y, DWORD z);

DWORD H(DWORD x, DWORD y, DWORD z);

DWORD I(DWORD x, DWORD y, DWORD z);

void FF(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

void GG(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

void HH(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

void II(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

DWORD ROL(DWORD x, DWORD n);

void decode(DWORD\* output, BYTE\* input, DWORD len);

void encode(BYTE\* output, DWORD\* input, DWORD len);

void transformHash(DWORD\* state, BYTE\* block);

void update(Context\* context, BYTE\* input, DWORD inputlen);

void final(BYTE\* digest, Context\* context);

BYTE\* hashProcess(const char\* s);

void setBuffer(BYTE\* output, BYTE\* input, DWORD len);

void setMem(BYTE\* output, int value, DWORD len);

BYTE\* display(BYTE\* digest);

void MD5way1(const char\* fileName);

void MD5way2(const char\* s);

void base64();

private:

BYTE Padding[64];

static const DWORD FS[16];

static const DWORD GS[16];

static const DWORD HS[16];

static const DWORD IS[16];

static const DWORD FAC[16];

static const DWORD GAC[16];

static const DWORD HAC[16];

static const DWORD IAC[16];

};

void MD5::transformHash(DWORD\* state, BYTE\* block) {

DWORD A, B, C, D;

DWORD x[16];

A = state[0];

B = state[1];

C = state[2];

D = state[3];

decode(x, block, 64);

DWORD temp;

DWORD i;

for (i = 0; i < 16; i++) {

FF(A, B, C, D, x[i], FS[i], FAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

for (i = 0; i < 16; i++) {

GG(A, B, C, D, x[(1 + i \* 5) % 16], GS[i], GAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

for (i = 0; i < 16; i++) {

HH(A, B, C, D, x[(5 + 3 \* i) % 16], HS[i], HAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

for (i = 0; i < 16; i++) {

II(A, B, C, D, x[(i \* 7) % 16], IS[i], IAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

state[0] += A;

state[1] += B;

state[2] += C;

state[3] += D;

}

void MD5::update(Context\* context, BYTE\* input, DWORD inputlen) {

DWORD i, index, partLength;

index = (DWORD)((context->count[0] >> 3) & 0x3F); //count是计算已输入数据的总位数，除以8，模64运算，即输入数据的字节数%64 （尾巴长度）

context->count[0] += ((DWORD)inputlen << 3);

//下面的if是用来处理进位的

if (context->count[0] < ((DWORD)inputlen << 3)) {

context->count[1]++;

}

context->count[1] += ((DWORD)inputlen >> 29);

partLength = 64 - index;//距离64bit的距离，先凑齐，用于1）大于64b消息或是2）说明2；

if (inputlen >= partLength) {

setBuffer(&context->buffer[index], input, partLength);//填齐用的

transformHash(context->state, context->buffer);

for (i = partLength; i + 63 < inputlen; i += 64) {

transformHash(context->state, &input[i]);

}

index = 0;

}

else {

i = 0;

}

setBuffer((BYTE\*)&context->buffer[index], (BYTE\*)&input[i], inputlen - i);//填充剩余的

}

//hash的计算过程

void MD5::final(BYTE\* digest, Context\* context) {

BYTE bits[8];

DWORD index, padLength;

encode(bits, context->count, 8);//将消息长度传入到bits中

index = (DWORD)((context->count[0] >> 3) & 0x3F);//计算已插入了多少，距离64B

padLength = (index < 56) ? (56 - index) : (120 - index); //确定需要填充的数据长度，有8字节用于附加输入数据的总长度 64+56=120

update(context, Padding, padLength);//加入填充字节

update(context, bits, 8);//加入长度

encode(digest, context->state, 16);//将算好的MD5传入到digest中 16\*8

setMem((BYTE\*)context, 0, sizeof(\*context));//重置contex

}

//len由sizeof得到

//将要计算的数据转换到context中的buffer[64]中

void MD5::setBuffer(BYTE\* output, BYTE\* input, DWORD len) {

DWORD i;

for (i = 0; i < len; i++) {

output[i] = input[i];

}

}

//完成计算后，用此函数重置context

void MD5::setMem(BYTE\* output, int value, DWORD len) {

DWORD i;

for (i = 0; i < len; i++) {

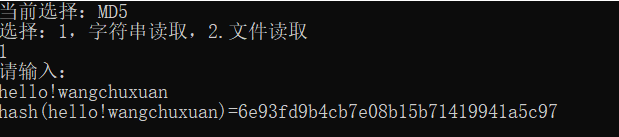
((char\*)output)[i] = (char)value;

}

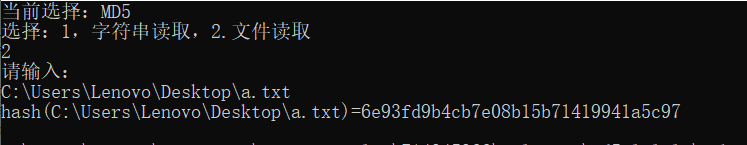
}

实验结果：

字符串输入



文件输入



1. **心得体会**

MD5算法在课上讲的时间不多，对他的了解不如RC4和AES一般多，故我参考着网上的资料，按照老师所要求的形式来编写MD5算法。

在实现MD5的算法时，我觉得最大的难点是对消息的填充过程。MD5算法中的四个非线性函数和四个复合函数按照书中给的算法就可以写出来。通过前两个实验，本人对字符串，按位运算等操作均比较熟练，在编码实现上没有太大的问题。本程序在对消息进行计算MD5时采用的是先计算消息前面的n\*512位消息的MD5值，然后再对剩余的内容进行填充，之后再接着计算。本程序在设计中在结构体context中设置了(uchar)buffer[64]来充当缓冲区，记录消息长度的(uint)count[2]，还有用来存放Hash值的ABCD（ABCD均为32位的二进制）。在编程中需要注意的是，在使用复合操作的时候，会改变A的值，故要对其进行引用&。

**参考文献**

1. 杨波.现代密码学.清华大学出版社.2017
2. 王静文,吴晓艺.密码编码与信息安全-C++实践.清华大学出版社.2015
3. 李子臣,杨亚涛.典型密码算法 C语言实现.国防工业出版社.2013

**附录：实验源码**

**RC4**

using namespace std;

typedef unsigned char BYTE;

class RC4

{

public:

RC4(char\* key, int keylen);

void setkey(char\* key, int keylen);

void getkey(char\* output,char\* data, int len);

void Encrypt(char\* cc,char\* mm,int len);

void Decrypt(char\* cc, char\* mm, int len);

private:

BYTE s[256],k[256];

};

RC4::RC4(char\* key, int keylen)

{

this->setkey(key, keylen);

}

void RC4::setkey(char\* key, int keylen)

{

for (int i = 0; i < 256; i++)

{

k[i] = key[i % keylen];

}

for (int i = 0; i < 256; i++)

{

s[i] = i;

}

int j = 0;

for (int i = 0; i < 256; i++)

{

j = (j + s[i] + k[i]) % 256;

swap(s[i], s[j]);

}

}

//生成密钥流

void RC4::getkey(char\* cc, char\* mm, int len)

{

int i = 0, j = 0;

for (int k = 0; k < len; k++)

{

i = (i + 1) % 256;

j = (j + s[i]) % 256;

swap(s[i], s[j]);

BYTE key = s[(s[i] + s[j]) % 256];

cc[k] = key ^ mm[k];

}

}

void RC4::Encrypt(char\* cc, char\* mm, int len)

{

this->getkey(cc, mm, len);

}

void RC4::Decrypt(char\* cc, char\* mm, int len)

{

this->getkey(cc, mm, len);

}

void way1()

{

cout << "请输入明文" << endl;

char msg[256];

cin >> msg;

int lenM = strlen(msg);

char cipher[256] = "";

cout << "请输入密钥" << endl;

char key[256];

cin >> key;

int keylen = strlen(key);

RC4 rc4(key, keylen);

rc4.Encrypt(cipher, msg, lenM);

printf("加密后密文: %s\n", cipher);

rc4.initial(key, keylen);

rc4.Decrypt(cipher, cipher, lenM);

printf("解密后明文: %s\n", cipher);

}

void way2()

{

char msg[256] = "";

cout << "请输入文件" << endl;

FILE\* file;

char f[50];

cin >> f;

if ((file = fopen(f, "rb")) == NULL)

{

cout << "不存在该文件！" << endl;

exit(0);

}

char a;

int i = 0;

while ((a = fgetc(file)) != EOF)

{

msg[i++] = a;

}

msg[i] = '\0';

fclose(file);

int lenM = strlen(msg);

char cipher[256] = "";

char key[256];

cin >> key;

int keylen = strlen(key);

RC4 rc4(key, keylen);

rc4.Encrypt(cipher, msg, lenM);

printf("加密后密文: %s\n", cipher);

rc4.initial(key, keylen);

rc4.Decrypt(cipher, cipher, lenM);

printf("解密后明文: %s\n", cipher);

}

**AES**

typedef unsigned char BYTE;

#define Nk 4//密钥分组长度/32

#define Nb 4//数据分组长度/32

#define Nr 10//轮数

#define keylen (Nb\*(Nr+1))

struct word

{

BYTE bytechar[4];

};

class AES

{

public:

AES(BYTE\* key);

~AES() {};

int ECBEncrypt(word\* c, BYTE\* m, int length);

int ECBDecrypt(word\* c, BYTE\* m, int length);

int CBCEncrypt(word\* c, BYTE\* m, int length);

int CBCDecrypt(word\* c, BYTE\* m, int length);

int OFBEncrypt(word\* c, BYTE\* m, int length);

void setkey(BYTE\* k);

void setplain(BYTE\* p);

void Encrypt\_1(word\* c, word\* m);

void Decrypt\_1(word\* c, word\* m);

word rotWord(word w);

word subWord(word w);

word wordXor(word w1, word w2);

word plain[4];

word Rcon[11];

void setr(word\* c);

private:

void SubBytes(word\* state);

void InvSubBytes(word\* state);

void ShiftRows(word\* state);

void InvShiftRows(word\* state);

void MixColumns(word\* state);

void InvMixColumns(word\* state);

void ExpandKey(BYTE\* key,word\* w);

void RoundKeyAdd(word\* state,int round);

BYTE iv[16] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };

BYTE cipherkey[16];

word cipher[4];

word wkey[44];

};

const BYTE Sbox[16][16] = {

{0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76},//0

{0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0},

{0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15},

{0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75},

{0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84},

{0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf},//5

{0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8},

{0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2},

{0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73},

{0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb},

{0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79},//a

{0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08},

{0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a},

{0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e},

{0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf},

{0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16}

};

const BYTE InvSbox[16][16] = {

{0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb},

{0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb},

{0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e},

{0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25},

{0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92},

{0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84},

{0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06},

{0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b},

{0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73},

{0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e},

{0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b},

{0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4},

{0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f},

{0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef},

{0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61},

{0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d}

};

BYTE xtime(BYTE a)//计算a\*2(即a\*x)

{

if (a & 0x80)//最高位非零

{

return (a << 1) ^ 0x1B;

}

return (a << 1);

}

BYTE xtime1(BYTE a)

{

return a;

}

BYTE xtime2(BYTE a)

{

return xtime(a);

}

BYTE xtime3(BYTE a)

{

return a ^ xtime(a);

}

BYTE xtime9(BYTE a)//00001001

{

return xtime(xtime(xtime(a))) ^ a;

}

BYTE xtimeb(BYTE a)//00001011

{

return xtime(xtime(xtime(a))) ^ xtime(a) ^ a;

}

BYTE xtimed(BYTE a)//00001101

{

return xtime(xtime(xtime(a))) ^ xtime(xtime(a)) ^ a;

}

BYTE xtimee(BYTE a)//00001110

{

return xtime(xtime(xtime(a))) ^ xtime(xtime(a)) ^ xtime(a);

}

AES::AES(BYTE\* key)

{

setr(this->Rcon);//设置Rcon值

setkey(key);//设置密钥

}

void AES::setkey(BYTE\* k)

{

for (int i = 0; i < 16; i++)

{

cipherkey[i] = k[i];

}

ExpandKey(cipherkey, wkey);//扩展密钥

}

void AES::setplain(BYTE\* p)

{

for (int i = 0; i < 16; i++)

{

plain[i / 4].bytechar[i % 4] = p[i];//设置要加密的内容

}

}

void AES::SubBytes(word\* state)

{

BYTE l, r;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

l = state[i].bytechar[j] >> 4;

r = state[i].bytechar[j] & 0x0F;

state[i].bytechar[j] = Sbox[l][r];

}

}

}

void AES::InvSubBytes(word\* state)

{

BYTE l, r;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

l = state[i].bytechar[j] >> 4;

r = state[i].bytechar[j] & 0x0F;

state[i].bytechar[j] = InvSbox[l][r];

}

}

}

void AES::RoundKeyAdd(word\* state, int round)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

state[i].bytechar[j] ^= wkey[4 \* round + i].bytechar[j];

}

}

}

void AES::ShiftRows(word\* state)

{

BYTE temp;

//左移1

temp = state[0].bytechar[1];

state[0].bytechar[1] = state[1].bytechar[1];

state[1].bytechar[1] = state[2].bytechar[1];

state[2].bytechar[1] = state[3].bytechar[1];

state[3].bytechar[1] = temp;

//左移2

temp = state[0].bytechar[2];

state[0].bytechar[2] = state[2].bytechar[2];

state[2].bytechar[2] = temp;

temp = state[1].bytechar[2];

state[1].bytechar[2] = state[3].bytechar[2];

state[3].bytechar[2] = temp;

//左移3

temp = state[0].bytechar[3];

state[0].bytechar[3] = state[3].bytechar[3];

state[3].bytechar[3] = state[2].bytechar[3];

state[2].bytechar[3] = state[1].bytechar[3];

state[1].bytechar[3] = temp;

}

void AES::InvShiftRows(word\* state)

{

BYTE temp;

//右移1

temp = state[0].bytechar[1];

state[0].bytechar[1] = state[3].bytechar[1];

state[3].bytechar[1] = state[2].bytechar[1];

state[2].bytechar[1] = state[1].bytechar[1];

state[1].bytechar[1] = temp;

//右移2

temp = state[0].bytechar[2];

state[0].bytechar[2] = state[2].bytechar[2];

state[2].bytechar[2] = temp;

temp = state[1].bytechar[2];

state[1].bytechar[2] = state[3].bytechar[2];

state[3].bytechar[2] = temp;

//右移3

temp = state[0].bytechar[3];

state[0].bytechar[3] = state[1].bytechar[3];

state[1].bytechar[3] = state[2].bytechar[3];

state[2].bytechar[3] = state[3].bytechar[3];

state[3].bytechar[3] = temp;

}

void AES::MixColumns(word\* state)

{

word result[4];

for (int i = 0; i < 4; i++)

{

result[i].bytechar[0] = xtime2(state[i].bytechar[0]) ^ xtime3(state[i].bytechar[1]) ^ xtime1(state[i].bytechar[2]) ^ xtime1(state[i].bytechar[3]);

result[i].bytechar[1] = xtime1(state[i].bytechar[0]) ^ xtime2(state[i].bytechar[1]) ^ xtime3(state[i].bytechar[2]) ^ xtime1(state[i].bytechar[3]);

result[i].bytechar[2] = xtime1(state[i].bytechar[0]) ^ xtime1(state[i].bytechar[1]) ^ xtime2(state[i].bytechar[2]) ^ xtime3(state[i].bytechar[3]);

result[i].bytechar[3] = xtime3(state[i].bytechar[0]) ^ xtime1(state[i].bytechar[1]) ^ xtime1(state[i].bytechar[2]) ^ xtime2(state[i].bytechar[3]);

}

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

state[i].bytechar[j] = result[i].bytechar[j];

}

}

}

void AES::InvMixColumns(word\* state)

{

word result[4];

for (int i = 0; i < 4; i++)

{

result[i].bytechar[0] = xtimee(state[i].bytechar[0]) ^ xtimeb(state[i].bytechar[1]) ^ xtimed(state[i].bytechar[2]) ^ xtime9(state[i].bytechar[3]);

result[i].bytechar[1] = xtime9(state[i].bytechar[0]) ^ xtimee(state[i].bytechar[1]) ^ xtimeb(state[i].bytechar[2]) ^ xtimed(state[i].bytechar[3]);

result[i].bytechar[2] = xtimed(state[i].bytechar[0]) ^ xtime9(state[i].bytechar[1]) ^ xtimee(state[i].bytechar[2]) ^ xtimeb(state[i].bytechar[3]);

result[i].bytechar[3] = xtimeb(state[i].bytechar[0]) ^ xtimed(state[i].bytechar[1]) ^ xtime9(state[i].bytechar[2]) ^ xtimee(state[i].bytechar[3]);

}

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

state[i].bytechar[j] = result[i].bytechar[j];

}

}

}

void AES::ExpandKey(BYTE\* key, word\* w)

{

word temp;

for (int i = 0; i < Nk; i++)

{

for (int j = 0; j < 4; j++)

{

w[i].bytechar[j] = key[4 \* i + j];

}

}

for (int i = Nk; i < keylen; i++)

{

temp = w[i - 1];

if ((i % Nk) == 0)

{

temp = wordXor(subWord(rotWord(temp)),Rcon[i/Nk]);

}

for (int j = 0; j < 4; j++)

{

w[i].bytechar[j] = w[i - Nk].bytechar[j] ^ temp.bytechar[j];

}

}

}

int AES::ECBEncrypt(word\* c, BYTE\* m, int length)

{

int insertnum = 0;

insertnum = 16 - length % 16;

int n = 0;

while (length >= 16)

{

this->setplain(&m[n \* 16]);

this->Encrypt\_1(&c[n \* 4], &plain[0]);

length -= 16;

n++;

}//对前面的加密

if (insertnum > 0)

{

BYTE\* temp = new BYTE[16];

for (int i = 0; i < length; i++)

{

temp[i] = m[n \* 16 + i];

}

for (int i = 16 - insertnum; i < 16; i++)

{

temp[i] = insertnum;

}

this->setplain(temp);

Encrypt\_1(&c[n \* 4],&plain[0]);

delete[] temp;

}

return n+1;

}

int AES::ECBDecrypt(word\* c, BYTE\* m, int length)

{

int n = 0;

while (length > 0)

{

this->setplain(&m[n \* 16]);

Decrypt\_1(&c[n \* 4], &plain[0]);

length -= 16;

n++;

}

int filln = (int)(c[n\*4-1].bytechar[3] - 0x00);

//转化成十进制

return filln;

}

int AES::CBCEncrypt(word\* c, BYTE\* m, int length)

{

int insertnum = 0;

insertnum = 16 - length % 16;

int n = 0;

BYTE\* tran=NULL;

while (length >= 16)

{

if (tran == NULL && !n)

{

for (int i = 0; i < 16; i++)

{

m[i] ^= iv[i];

}

this->setplain(m);

Encrypt\_1(c, &plain[0]);

tran = new BYTE[16];

memcpy(tran, &c[0], 16);

}

else

{

if (tran)

{

for (int i = 0; i < 16; i++)

{

m[n \* 16 + i] ^= tran[i];

}

}

this->setplain(&m[n \* 16]);

Encrypt\_1(&c[n \* 4], &plain[0]);

memcpy(tran, &c[n \* 4], 16);

}

length -= 16;

n++;

}

if (insertnum)

{

BYTE\* temp = new BYTE[16];

for (int i = 0; i < length; i++)

{

temp[i] = m[n \* 16 + i];

}

for (int i = 16 - insertnum; i < 16; i++)

{

temp[i] = insertnum;

}

if (tran)

{

for (int i = 0; i < 16; i++)

{

temp[i] ^= tran[i];

}

delete[] tran;

}

else

{

for (int i = 0; i < 16; i++)

{

temp[i] ^= iv[i];

}

}

this->setplain(temp);

Encrypt\_1(&c[n \* 4], & plain[0]);

delete[] temp;

tran = NULL;

}

return n+1;

}

int AES::CBCDecrypt(word\* c, BYTE\* m, int length)

{

BYTE\* tran=NULL;

BYTE\* temp=new BYTE[16];

int n = 0;

while (length>0)

{

if (tran == NULL && !n) //第一组

{

tran = new BYTE[16];

memcpy(tran, &m[0], 16);

this->setplain(&m[0]);

Decrypt\_1(&c[0], &plain[0]);

for (int i = 0; i < 16; i++)

{

c[i/4].bytechar[i%4] ^= iv[i];

}

}

else

{

memcpy(temp, &m[n \* 16], 16);

this->setplain(&m[n \* 16]);

Decrypt\_1(&c[n \* 4], &plain[0]);

for (int i = 0; i < 16; i++)

{

c[n \* 4 + (i/4)].bytechar[i% 4] ^= tran[i];

}

memcpy(tran, temp, 16);

}

length -= 16;

n++;

}

delete[] temp;

delete[] tran;

int filln = (int)(c[n \* 4 - 1].bytechar[3] - 0x00);

return filln;

}

int AES::OFBEncrypt(word\* c, BYTE\* m, int length)

{

BYTE temp[16];

word tempp[4];

memcpy(temp,iv,16);

for (int i = 0; i < length; i++) {

this->setplain(temp);

Encrypt\_1(tempp,&plain[0]);

c[i/4].bytechar[i%4] = m[i] ^ tempp[0].bytechar[0];

for (int j = 0; j < 15; j++)

{

temp[j] = temp[j + 1];

}

temp[15] = tempp[0].bytechar[0];

}

return 1;

}

void AES::Encrypt\_1(word\* c, word\* m)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

c[i].bytechar[j] = m[i].bytechar[j];

}

}

RoundKeyAdd(c, 0);

for (int i = 1; i < 10; i++)

{

SubBytes(c);

ShiftRows(c);

MixColumns(c);

RoundKeyAdd(c, i);

}

SubBytes(c);

ShiftRows(c);

RoundKeyAdd(c, 10);

}

void AES::Decrypt\_1(word\* c, word\* m)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

c[i].bytechar[j] = m[i].bytechar[j];

}

}

RoundKeyAdd(c, 10);

for(int i = 9; i > 0; i--)

{

InvShiftRows(c);

InvSubBytes(c);

RoundKeyAdd(c, i);

InvMixColumns(c);

}

InvShiftRows(c);

InvSubBytes(c);

RoundKeyAdd(c, 0);

}

word AES::rotWord(word w)

{

word temp;

for (int i = 0; i < 4; i++)

{

temp.bytechar[(i + 3) % 4] = w.bytechar[i];

}

return temp;

}

word AES::subWord(word w)

{

BYTE l, r;

for (int i = 0; i < 4; i++)

{

l = w.bytechar[i] >> 4;

r = w.bytechar[i] & 0x0F;

w.bytechar[i] = Sbox[(int)l][(int)r];

}

return w;

}

word AES::wordXor(word w1, word w2)

{

word temp;

for (int i = 0; i < 4; i++)

{

temp.bytechar[i] = w1.bytechar[i] ^ w2.bytechar[i];

}

return temp;

}

void AES::setr(word\* Rcon)//

{

Rcon[0].bytechar[0] = 0x00;

Rcon[0].bytechar[1] = 0x00;

Rcon[0].bytechar[2] = 0x00;

Rcon[0].bytechar[3] = 0x00;

Rcon[1].bytechar[0] = 0x01;

Rcon[1].bytechar[1] = 0x00;

Rcon[1].bytechar[2] = 0x00;

Rcon[1].bytechar[3] = 0x00;

Rcon[2].bytechar[0] = 0x02;

Rcon[2].bytechar[1] = 0x00;

Rcon[2].bytechar[2] = 0x00;

Rcon[2].bytechar[3] = 0x00;

Rcon[3].bytechar[0] = 0x04;

Rcon[3].bytechar[1] = 0x00;

Rcon[3].bytechar[2] = 0x00;

Rcon[3].bytechar[3] = 0x00;

Rcon[4].bytechar[0] = 0x08;

Rcon[4].bytechar[1] = 0x00;

Rcon[4].bytechar[2] = 0x00;

Rcon[4].bytechar[3] = 0x00;

Rcon[5].bytechar[0] = 0x10;

Rcon[5].bytechar[1] = 0x00;

Rcon[5].bytechar[2] = 0x00;

Rcon[5].bytechar[3] = 0x00;

Rcon[6].bytechar[0] = 0x20;

Rcon[6].bytechar[1] = 0x00;

Rcon[6].bytechar[2] = 0x00;

Rcon[6].bytechar[3] = 0x00;

Rcon[7].bytechar[0] = 0x40;

Rcon[7].bytechar[1] = 0x00;

Rcon[7].bytechar[2] = 0x00;

Rcon[7].bytechar[3] = 0x00;

Rcon[8].bytechar[0] = 0x80;

Rcon[8].bytechar[1] = 0x00;

Rcon[8].bytechar[2] = 0x00;

Rcon[8].bytechar[3] = 0x00;

Rcon[9].bytechar[0] = 0x1b;

Rcon[9].bytechar[1] = 0x00;

Rcon[9].bytechar[2] = 0x00;

Rcon[9].bytechar[3] = 0x00;

Rcon[10].bytechar[0] = 0x36;

Rcon[10].bytechar[1] = 0x00;

Rcon[10].bytechar[2] = 0x00;

Rcon[10].bytechar[3] = 0x00;

}

**MD5**

typedef unsigned int DWORD;

typedef unsigned char BYTE;

struct Context {

DWORD state[4]; //存放哈希值

DWORD count[2]; //存放待计算哈希函数的数据的长度

BYTE buffer[64]; //存放临时数据

};

class MD5

{

public:

MD5();

void initial(Context\* context);

DWORD F(DWORD x, DWORD y, DWORD z);

DWORD G(DWORD x, DWORD y, DWORD z);

DWORD H(DWORD x, DWORD y, DWORD z);

DWORD I(DWORD x, DWORD y, DWORD z);

void FF(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

void GG(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

void HH(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

void II(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac);

DWORD ROL(DWORD x, DWORD n);

void decode(DWORD\* output, BYTE\* input, DWORD len);

void encode(BYTE\* output, DWORD\* input, DWORD len);

void transformHash(DWORD\* state, BYTE\* block);

void update(Context\* context, BYTE\* input, DWORD inputlen);

void final(BYTE\* digest, Context\* context);

BYTE\* hashProcess(const char\* s);

void setBuffer(BYTE\* output, BYTE\* input, DWORD len);

void setMem(BYTE\* output, int value, DWORD len);

BYTE\* display(BYTE\* digest);

void MD5way1(const char\* fileName);

void MD5way2(const char\* s);

void base64();

private:

BYTE Padding[64];

static const DWORD FS[16];

static const DWORD GS[16];

static const DWORD HS[16];

static const DWORD IS[16];

static const DWORD FAC[16];

static const DWORD GAC[16];

static const DWORD HAC[16];

static const DWORD IAC[16];

};

const DWORD MD5::FS[16] = {7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22};

const DWORD MD5::GS[16] = {5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20};

const DWORD MD5::HS[16] = {4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23};

const DWORD MD5::IS[16] = {6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21};

const DWORD MD5::FAC[16] = {

0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,

0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,

0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,

0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821

};

const DWORD MD5::GAC[16] = {

0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,

0xd62f105d, 0x2441453, 0xd8a1e681, 0xe7d3fbc8,

0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,

0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a

};

const DWORD MD5::HAC[16] = {

0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,

0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,

0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x4881d05,

0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665

};

const DWORD MD5::IAC[16] = {

0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,

0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,

0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,

0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391

};

//填充数据的初始化

MD5::MD5()

{

Padding[0] = 0x80;

DWORD i;

for (i = 1; i < 64; i++)

{

Padding[i] = 0;

}

}

//装入幻数

void MD5::initial(Context\* context)

{

context->state[0] = 0x67452301;

context->state[1] = 0xefcdab89;

context->state[2] = 0x98badcfe;

context->state[3] = 0x10325476;

context->count[0] = 0;

context->count[1] = 0;

}

//4个算子

DWORD MD5::F(DWORD x, DWORD y, DWORD z)

{

return (((x) & (y)) | ((~x) & (z)));

}

DWORD MD5::G(DWORD x, DWORD y, DWORD z)

{

return (((x) & (z)) | ((y) & (~z)));

}

DWORD MD5::H(DWORD x, DWORD y, DWORD z)

{

return ((x) ^ (y) ^ (z));

}

DWORD MD5::I(DWORD x, DWORD y, DWORD z)

{

return ((y) ^ ((x) | (~z)));

}

//4个辅助类算子

//A= B+((A+Func(B,C,D)+X[k]+T[i])<<<s)

void MD5::FF(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac)

{

a += F(b, c, d) + x + ac;

a = ROL(a, s);

a += b;

}

void MD5::GG(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac)

{

a += G(b, c, d) + x + ac;

a = ROL(a, s);

a += b;

}

void MD5::HH(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac)

{

a += H(b, c, d) + x + ac;

a = ROL(a, s);

a += b;

}

void MD5::II(DWORD& a, DWORD b, DWORD c, DWORD d, DWORD x, const DWORD s, const DWORD ac)

{

a += I(b, c, d) + x + ac;

a = ROL(a, s);

a += b;

}

//循环左移函数

DWORD MD5::ROL(DWORD x, DWORD n)

{

return (((x) << (n)) | ((x) >> (32 - (n))));

}

//word32与uchar类型相互转化 len为char的长度

void MD5::decode(DWORD\* output, BYTE\* input, DWORD len)

{

DWORD i, j;

for (i = 0, j = 0; j < len; i++, j += 4)

{

output[i] = ((DWORD)input[j]) | (((DWORD)input[j + 1]) << 8)

| (((DWORD)input[j + 2]) << 16) | (((DWORD)input[j + 3]) << 24);

}

}

void MD5::encode(BYTE\* output, DWORD\* input, DWORD len)

{

DWORD i, j;

for (i = 0, j = 0; j < len; i++, j += 4)

{

output[j] = (BYTE)(input[i] & 0xFF);

output[j + 1] = (BYTE)(input[i] >> 8 & 0xFF);

output[j + 2] = (BYTE)(input[i] >> 16 & 0xFF);

output[j + 3] = (BYTE)(input[i] >> 24 & 0xFF);

}

}

//计算hash的核心函数 block为待计算MD5的串，state为缓冲区ABCD

void MD5::transformHash(DWORD\* state, BYTE\* block)

{

DWORD A, B, C, D;

DWORD x[16];

A = state[0];

B = state[1];

C = state[2];

D = state[3];

decode(x, block, 64);

DWORD temp;

DWORD i;

for (i = 0; i < 16; i++)

{

FF(A, B, C, D, x[i], FS[i], FAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

for (i = 0; i < 16; i++)

{

GG(A, B, C, D, x[(1 + i \* 5) % 16], GS[i], GAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

for (i = 0; i < 16; i++)

{

HH(A, B, C, D, x[(5 + 3 \* i) % 16], HS[i], HAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

for (i = 0; i < 16; i++)

{

II(A, B, C, D, x[(i \* 7) % 16], IS[i], IAC[i]);

temp = D;

D = C;

C = B;

B = A;

A = temp;

}

state[0] += A;

state[1] += B;

state[2] += C;

state[3] += D;

}

//填充

//说明1：每64B计算一下，直到小于64B；

//说明2：对剩余部分进行填充，如果填充后大于64B则计算64B的hash，否则说明3；

//说明3：填充，算hash

void MD5::update(Context\* context, BYTE\* input, DWORD inputlen)

{

DWORD i, index, partLength;

index = (DWORD)((context->count[0] >> 3) & 0x3F); //count是计算已输入数据的总位数，除以8，模64运算，即输入数据的字节数%64 （尾巴长度）

context->count[0] += ((DWORD)inputlen << 3);

//下面的if是用来处理进位的

if (context->count[0] < ((DWORD)inputlen << 3))

{

context->count[1]++;

}

context->count[1] += ((DWORD)inputlen >> 29);

partLength = 64 - index;//距离64bit的距离，先凑齐，用于1）大于64b消息或是2）说明2；

if (inputlen >= partLength)

{

setBuffer(&context->buffer[index], input, partLength);//填齐用的

transformHash(context->state, context->buffer);

for (i = partLength; i + 63 < inputlen; i += 64)

{

transformHash(context->state, &input[i]);

}

index = 0;

}

else

{

i = 0;

}

setBuffer((BYTE\*)&context->buffer[index], (BYTE\*)&input[i], inputlen - i);//填充剩余的

}

//hash的计算过程

void MD5::final(BYTE\* digest, Context\* context)

{

BYTE bits[8];

DWORD index, padLength;

encode(bits, context->count, 8);//将消息长度传入到bits中

index = (DWORD)((context->count[0] >> 3) & 0x3F);//计算已插入了多少，距离64B

padLength = (index < 56) ? (56 - index) : (120 - index); //确定需要填充的数据长度，有8字节用于附加输入数据的总长度 64+56=120

update(context, Padding, padLength);//加入填充字节

update(context, bits, 8);//加入长度

encode(digest, context->state, 16);//将算好的MD5传入到digest中 16\*8

setMem((BYTE\*)context, 0, sizeof(\*context));//重置contex

}

//len由sizeof得到

//将要计算的数据转换到context中的buffer[64]中

void MD5::setBuffer(BYTE\* output, BYTE\* input, DWORD len)

{

DWORD i;

for (i = 0; i < len; i++) {

output[i] = input[i];

}

}

//完成计算后，用此函数重置context

void MD5::setMem(BYTE\* output, int value, DWORD len)

{

DWORD i;

for (i = 0; i < len; i++)

{

((char\*)output)[i] = (char)value;

}

}

//输出的

BYTE\* MD5::display(BYTE\* digest)

{

DWORD i;

BYTE result[32];

for (i = 0; i < 16; i++) {

result[2\*i] = (digest[i] >> 4) & 0x0F;

result[2\*i+1] = digest[i] & 0x0F;

}

return result;

}

//入口函数

BYTE\* MD5::hashProcess(const char\* s)

{

Context context;

initial(&context);

BYTE digest[16];

DWORD len = strlen(s);

update(&context, (BYTE\*)s, len);

final(digest, &context);

BYTE\* MD5\_result = &display(digest)[0];

for (int i = 0; i < 32; i++)

{

cout <<hex<<int(display(digest)[i]);

}

return digest;

}

void MD5::MD5way1(const char\* fileName)

{

/\*ifstream readFile;

readFile.open(fileName);

char buffer[1024] = { 0 };

while (!readFile.eof())

{

readFile.getline(buffer,64);

cout << buffer<<endl;

}

readFile.close();\*/

char msg[256] = "";

FILE\* file;

if ((file = fopen(fileName, "rb")) == NULL)

{

cout << "不存在该文件！" << endl;

exit(0);

}

char a;

int i = 0;

while ((a = fgetc(file)) != EOF)

{

msg[i++] = a;

}

msg[i] = '\0';

fclose(file);

int lenM = strlen(msg);

cout << "hash(" << fileName << ")=";

hashProcess(msg);

cout << endl;

}

void MD5::MD5way2(const char\* s)

{

cout << "hash(" << s << ")=";

hashProcess(s);

cout << endl;

}

**main函数**

**Rc4部分：**

cout << "当前加密方法选择：RC4"<<endl;

cout << "请选择输入方式:1-手动输入，2-文件输入" << endl;

int way;

cin >> way;

switch (way)

{

case 1:

way1();

break;

case 2:

way2();

break;

default:

break;

}

**Aes部分：**

cout << "当前加密方式:AES" << endl;

cout << "选择加密：1.16字节明文，2.文件输入任意长度明文" << endl;

int k;

cin >> k;

if (k == 1) {

cout << "请输入16字节明文" << endl;

char key[17];//’\0’

char m[17];

cin >> m;

cout << "请输入16字节密钥" << endl;

cin >> key;

AES aes((BYTE\*)key);

aes.setplain((BYTE\*)m);

word\* c = new word[4];

aes.Encrypt\_1(c, aes.plain);

cout << "加密后密文" << endl;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

cout << hex << int(c[i].bytechar[j] >> 4) << int(c[i].bytechar[j] & 0x0f);

}

}

cout << endl;

cout << "解密后明文" << endl;

aes.setplain((BYTE\*)c);

aes.Decrypt\_1(c, aes.plain);

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

cout << c[i].bytechar[j];

}

}

}

if (k == 2) {

cout << "请输入16字节密钥" << endl;

char key[17];

cin >> key;

AES aes((BYTE\*)key);

word\* c = new word[16];

cout << "请选择链接模式:1.ECB,2.CBC,3.OFB" << endl;

int p;

cin >> p;

char m[256]="";

cout << "请输入文件" << endl;

FILE\* file;

char f[100];

cin >> f;

if ((file = fopen(f, "rb")) == NULL)

{

cout << "不存在该文件！" << endl;

exit(0);

}

char a;

int i = 0;

while ((a = fgetc(file)) != EOF)

{

m[i++] = a;

}

m[i] = '\0';

fclose(file);

int len = strlen(m);

if (p == 1) {

cout << "加密后密文:" << endl;

BYTE mm[256] = "";

int a = aes.ECBEncrypt(c, (BYTE\*)m, len);//总块数

for (int i = 0; i < 4 \* a; i++) {

for (int j = 0; j < 4; j++) {

cout << hex << int(c[i].bytechar[j] >> 4) << int(c[i].bytechar[j] & 0x0f);

mm[4 \* i + j] = c[i].bytechar[j];

}

}

cout << endl;

cout << "解密后明文:" << endl;

int b = aes.ECBDecrypt(c, mm, 16\*a);//填充个数

int nn = 16\*a-b;

for (int i = 0; i < 4 \* a; i++) {

for (int j = 0; j < 4; j++) {

//cout << hex << int(c[i].bytechar[j] >> 4) << int(c[i].bytechar[j] & 0x0f);

if (4 \* i + j < nn)

{

cout << c[i].bytechar[j];

}

}

}

}

if (p == 2) {

cout << "加密后密文:" << endl;

BYTE mm[256] = "";

int a = aes.CBCEncrypt(c, (BYTE\*)m, len);//总块数

for (int i = 0; i < 4 \* a; i++) {

for (int j = 0; j < 4; j++) {

cout << hex << int(c[i].bytechar[j] >> 4) << int(c[i].bytechar[j] & 0x0f);

mm[4 \* i + j] = c[i].bytechar[j];

}

}

cout << endl;

cout << "解密后明文:" << endl;

int b = aes.CBCDecrypt(c, mm, 16 \* a);//填充个数

int nn = 16 \* a - b;

for (int i = 0; i < 4 \* a; i++) {

for (int j = 0; j < 4; j++) {

if (4 \* i + j < nn)

{

cout << c[i].bytechar[j];

}

}

}

}

if (p == 3) {

cout << "加密后密文:" << endl;

BYTE mm[256] = "";

//int a = aes.OFBEncrypt(c, (BYTE\*)m, len);//总块数

aes.OFBEncrypt(c, (BYTE\*)m, len);

for (int i = 0; i <= len/4; i++) {

for (int j = 0; j < 4; j++) {

if (4\*i+j<len)

{

cout << hex << int(c[i].bytechar[j] >> 4) << int(c[i].bytechar[j] & 0x0f);

mm[4 \* i + j] = c[i].bytechar[j];

}

}

}

cout << endl;

cout << "解密后明文:" << endl;

aes.OFBEncrypt(c, (BYTE\*)mm, len);

for (int i = 0; i <=len/4; i++) {

for (int j = 0; j < 4; j++) {

if(4\*i+j<len)

cout << c[i].bytechar[j];

}

} delete[]c;

}

}

**MD5部分**

cout << "当前选择：MD5" << endl;

MD5 md5;

cout << "选择：1，字符串读取，2.文件读取" << endl;

int k;

cin >> k;

if (k == 1)

{

cout << "请输入：" << endl;

char a[513];

cin >> a;

md5.MD5way2(a);

}

if (k == 2) {

cout << "请输入：" << endl;

char f[100];

cin >> f;

md5.MD5way1(f);

}