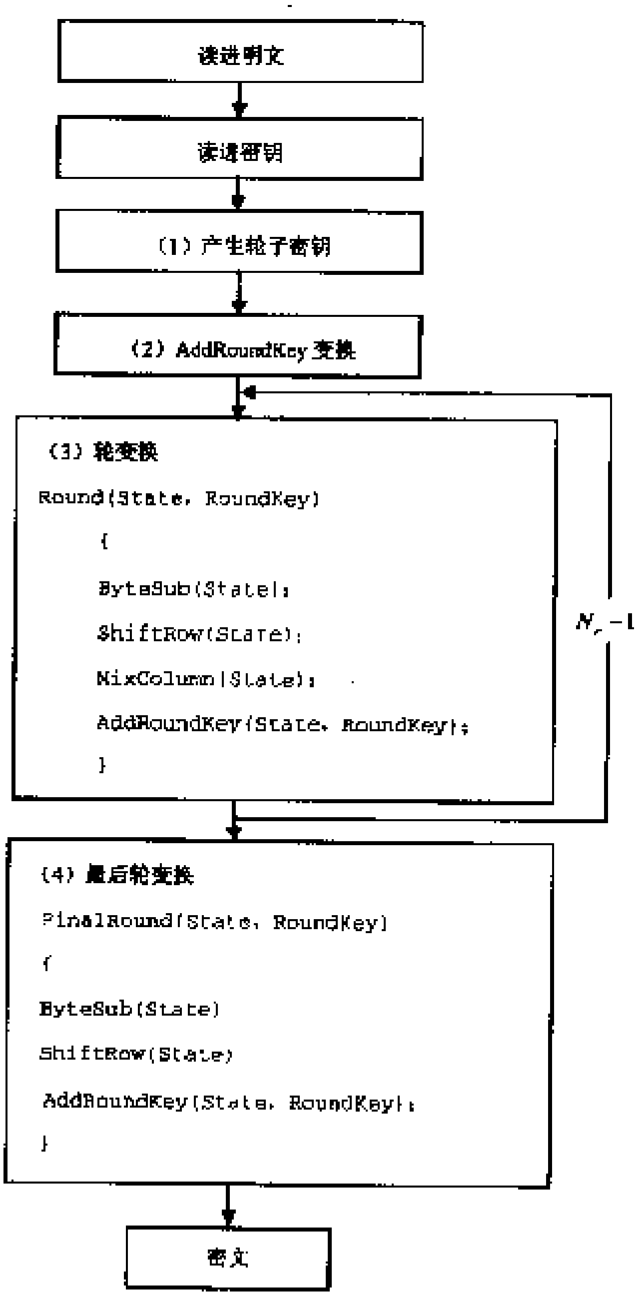
实验三 分组密码算法 AES

一、算法分析：

AES加密解密流程图如下：



二、实验原理

AES算法本质上是一种对称分组密码体制，采用代替/置换网络，每轮由三

层组成：线性混合层确保多轮之上的高度扩散，非线性层由16个S盒并置起到混淆的作用，密钥加密层将子密钥异或到中间状态。Rijndael是一个迭代分组密码，其分组长度和密钥长度都是可变的，只是为了满足AES的要求才限定处理的分组大小为128位，而密钥长度为128位、192位或256位，相应的迭代轮数N，为10轮、12轮、14轮。AES汇聚了安全性能、效率、可实现性、灵活性等优点。最大的优点是可以给出算法的最佳差分特征的概率，并分析算法抵抗差分密码分析及线性密码分析的能力。其实现的加密流程图如图-1所示。

加密的主要过程包括：对明文状态的一次密钥加，   轮轮加密和末尾轮轮加密，最后得到密文。其中 轮轮加密每一轮有四个部件，包括字节代换部件ByteSub、行移位变换ShiftRow、列混合变换NixColumn和一个密钥加AddRoundKey部件，末尾轮加密和前面轮加密类似，只是少了一个列混合变换NixColumn部件。下面具体介绍这几个部件的实现方法：

1、字节代换ByteSub部件

字节代换是非线形变换，独立地对状态的每个字节进行。代换表（即S-盒）是可逆的，由以下两个变换的合成得到：

（1） 首先，将字节看作GF(28)上的元素，映射到自己的乘法逆元，'00'映射到自己。

（2）其次，对字节做如下的（GF(2)上的，可逆的）仿射变换：

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

该部件的逆运算部件就是先对自己做一个逆仿射变换，然后映射到自己的乘法逆元上。

2、行移位变换ShiftRow

行移位是将状态阵列的各行进行循环移位，不同状态行的位移量不同。第0

行不移动，第1行循环左移C1个字节，第2行循环左移C2个字节，第3行循环左移C3个字节。位移量C1、C2、C3的取值与Nb有关，由教材中表3.10给出。

ShiftRow的逆变换是对状态阵列的后3列分别以位移量Nb-C1、Nb-C2、Nb-C3进行循环移位，使得第i行第j列的字节移位到(j+Nb-Ci) mod Nb。

3、列混合变换NixColumn

在列混合变换中，将状态阵列的每个列视为系数为GF(28)上的多项式，再与

一个固定的多项式c(x)进行模x4+1乘法。当然要求c(x)是模x4+1可逆的多项式，否则列混合变换就是不可逆的，因而会使不同的输入分组对应的输出分组可能相同。Rijndael的设计者给出的c(x)为（系数用十六进制数表示）：

c(x)='03'x3+'01'x2+'01'x+'02'

c(x)是与x4+1互素的，因此是模x4+1可逆的。列混合运算也可写为矩阵

乘法。设b(x)= c(x)   a(x)，则

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

这个运算需要做GF(28)上的乘法，但由于所乘的因子是3个固定的元素02、03、01，所以这些乘法运算仍然是比较简单的。

列混合运算的逆运算是类似的，即每列都用一个特定的多项式d(x)相乘。d(x)满足

(‘03’x3+‘01’x2+‘01’x+‘02’))   d(x)=‘01’

由此可得

d(x)=‘0B’x3+‘0D’x2+‘09’x+‘0E’

4、密钥加AddRoundKey部件

密钥加是将轮密钥简单地与状态进行逐比特异或。轮密钥由种子密钥通过密钥编排算法得到，轮密钥长度等于分组长度Nb。密钥加运算的逆运算是其自身。

5、密钥编排

密钥编排指从种子密钥得到轮密钥的过程，它由密钥扩展和轮密钥选取两部分组成。其基本原则如下：

（1） 轮密钥的字数（4比特32位的数）等于分组长度乘以轮数加1；

（2） 种子密钥被扩展成为扩展密钥；

（3） 轮密钥从扩展密钥中取，其中第1轮轮密钥取扩展密钥的前Nb个字，第2轮轮密钥取接下来的Nb个字，如此下去。

密钥扩展的方法和密钥的取法具体请参考教材和ppt。

6、解密过程

AES算法的解密过程和加密过程是相似的，也是先经过一个密钥加，然后进行 轮轮解密和末尾轮轮解密，最后得到明文。和加密不同的是 轮轮解密每一轮四个部件都需要用到它们的逆运算部件，包括字节代换部件的逆运算、行移位变换的逆变换、逆列混合变换和一个密钥加部件，末尾轮加密和前面轮加密类似，只是少了一个逆列混合变换部件。

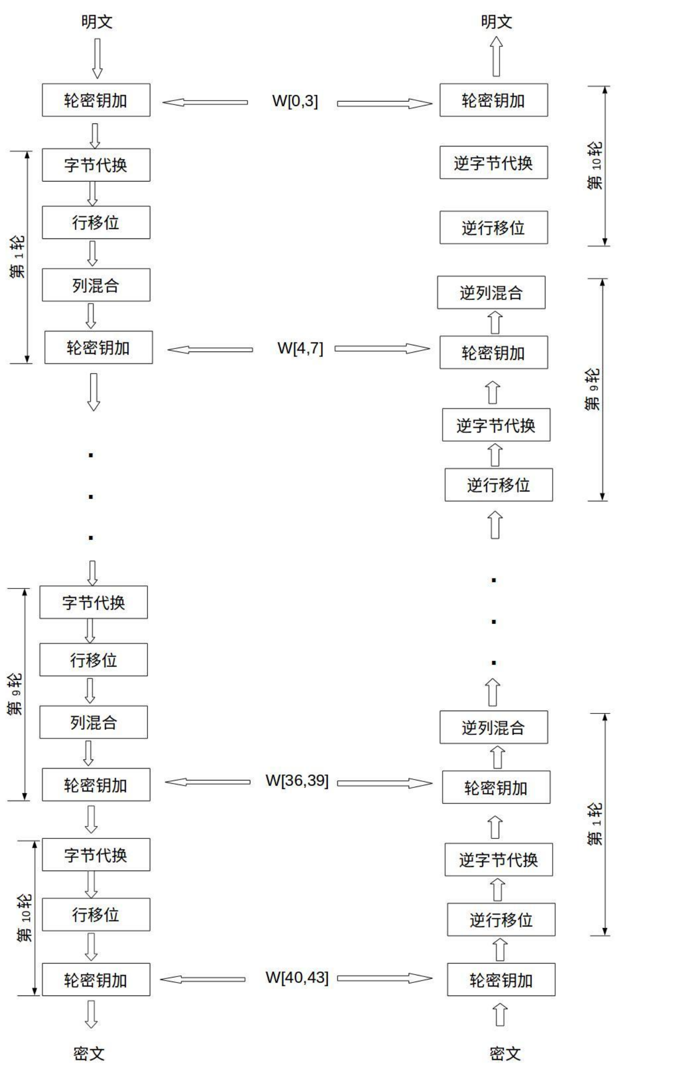
在解密的时候，还要注意轮密钥和加密密钥的区别，设加密算法的初始密钥加、第1轮、第2轮、…、第Nr轮的子密钥依次为k(0), k(1), k(2), …, k(Nr-1), k(Nr)

则解密算法的初始密钥加、第1轮、第2轮、…、第Nr轮的子密钥依次为

k(Nr), InvMixColumn (k(Nr-1)), InvMixColumn (k(Nr-2)), …,

InvMixColumn (k(1)), k(0)。

三、算法实现流程



四、程序代码和执行结果

aes_head.h:

```
#pragma once
#include <iostream>
#include <bitset>
#include <string>
using namespace std;
typedef bitset<8> byte;
typedef bitset<32> word;
const int Nr = 10;  // AES-128需要 10 轮加密
const int Nk = 4;   // Nk 表示输入密钥的 word 个数
byte S_Box[16][16] = {
    {0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B,
0xFE, 0xD7, 0xAB, 0x76},
    {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},
    {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},
    {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}
};
byte Inv_S_Box[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}
};
// 轮常数，密钥扩展中用到。（AES-128只需要10轮）
word Rcon[10] = { 0x01000000, 0x02000000, 0x04000000, 0x08000000, 0x10000000,
                 0x20000000, 0x40000000, 0x80000000, 0x1b000000, 0x36000000 };
/**********************************************************************/
/*                                                                    */
/*                              AES算法实现                            */
/*                                                                    */
/**********************************************************************/
/******************************下面是加密的变换函数**********************/
/**
 *  S盒变换 - 前4位为行号，后4位为列号
 */
void SubBytes(byte mtx[4 * 4])
{
    for (int i = 0; i < 16; ++i)
    {
        int row = mtx[i][7] * 8 + mtx[i][6] * 4 + mtx[i][5] * 2 + mtx[i][4];
        int col = mtx[i][3] * 8 + mtx[i][2] * 4 + mtx[i][1] * 2 + mtx[i][0];
        mtx[i] = S_Box[row][col];
    }
}
/**
 *  行变换 - 按字节循环移位
 */
void ShiftRows(byte mtx[4 * 4])
{
    // 第二行循环左移一位
    byte temp = mtx[4];
    for (int i = 0; i < 3; ++i)
        mtx[i + 4] = mtx[i + 5];
    mtx[7] = temp;
    // 第三行循环左移两位
    for (int i = 0; i < 2; ++i)
    {
        temp = mtx[i + 8];
        mtx[i + 8] = mtx[i + 10];
        mtx[i + 10] = temp;
    }
    // 第四行循环左移三位
    temp = mtx[15];
    for (int i = 3; i > 0; --i)
        mtx[i + 12] = mtx[i + 11];
    mtx[12] = temp;
}
/**
 *  有限域上的乘法 GF(2^8)
 */
byte GFMul(byte a, byte b) {
    byte p = 0;
    byte hi_bit_set;
    for (int counter = 0; counter < 8; counter++) {
        if ((b & byte(1)) != 0) {
            p ^= a;
        }
        hi_bit_set = (byte)(a & byte(0x80));
```

```
        a <<= 1;
        if (hi_bit_set != 0) {
            a ^= 0x1b; /* x^8 + x^4 + x^3 + x + 1 */
        }
        b >>= 1;
    }
    return p;
}
/**
 *  列变换
 */
void MixColumns(byte mtx[4 * 4])
{
    byte arr[4];
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            arr[j] = mtx[i + j * 4];
        mtx[i] = GFMul(0x02, arr[0]) ^ GFMul(0x03, arr[1]) ^ arr[2] ^ arr[3];
        mtx[i + 4] = arr[0] ^ GFMul(0x02, arr[1]) ^ GFMul(0x03, arr[2]) ^
arr[3];
        mtx[i + 8] = arr[0] ^ arr[1] ^ GFMul(0x02, arr[2]) ^ GFMul(0x03,
arr[3]);
        mtx[i + 12] = GFMul(0x03, arr[0]) ^ arr[1] ^ arr[2] ^ GFMul(0x02,
arr[3]);
    }
}
/**
 *  轮密钥加变换 - 将每一列与扩展密钥进行异或
 */
void AddRoundKey(byte mtx[4 * 4], word k[4])
{
    for (int i = 0; i < 4; ++i)
    {
        word k1 = k[i] >> 24;
        word k2 = (k[i] << 8) >> 24;
        word k3 = (k[i] << 16) >> 24;
        word k4 = (k[i] << 24) >> 24;
        mtx[i] = mtx[i] ^ byte(k1.to_ulong());
        mtx[i + 4] = mtx[i + 4] ^ byte(k2.to_ulong());
        mtx[i + 8] = mtx[i + 8] ^ byte(k3.to_ulong());
        mtx[i + 12] = mtx[i + 12] ^ byte(k4.to_ulong());
    }
}
/**************************下面是解密的逆变换函数***********************/
/**
 *  逆S盒变换
 */
void InvSubBytes(byte mtx[4 * 4])
{
    for (int i = 0; i < 16; ++i)
    {
        int row = mtx[i][7] * 8 + mtx[i][6] * 4 + mtx[i][5] * 2 + mtx[i][4];
        int col = mtx[i][3] * 8 + mtx[i][2] * 4 + mtx[i][1] * 2 + mtx[i][0];
        mtx[i] = Inv_S_Box[row][col];
    }
}
/**
```

```
 *  逆行变换 - 以字节为单位循环右移
 */
void InvShiftRows(byte mtx[4 * 4])
{
    // 第二行循环右移一位
    byte temp = mtx[7];
    for (int i = 3; i > 0; --i)
        mtx[i + 4] = mtx[i + 3];
    mtx[4] = temp;
    // 第三行循环右移两位
    for (int i = 0; i < 2; ++i)
    {
        temp = mtx[i + 8];
        mtx[i + 8] = mtx[i + 10];
        mtx[i + 10] = temp;
    }
    // 第四行循环右移三位
    temp = mtx[12];
    for (int i = 0; i < 3; ++i)
        mtx[i + 12] = mtx[i + 13];
    mtx[15] = temp;
}
void InvMixColumns(byte mtx[4 * 4])
{
    byte arr[4];
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            arr[j] = mtx[i + j * 4];
        mtx[i] = GFMul(0x0e, arr[0]) ^ GFMul(0x0b, arr[1]) ^ GFMul(0x0d, arr[2])
^ GFMul(0x09, arr[3]);
        mtx[i + 4] = GFMul(0x09, arr[0]) ^ GFMul(0x0e, arr[1]) ^ GFMul(0x0b,
arr[2]) ^ GFMul(0x0d, arr[3]);
        mtx[i + 8] = GFMul(0x0d, arr[0]) ^ GFMul(0x09, arr[1]) ^ GFMul(0x0e,
arr[2]) ^ GFMul(0x0b, arr[3]);
        mtx[i + 12] = GFMul(0x0b, arr[0]) ^ GFMul(0x0d, arr[1]) ^ GFMul(0x09,
arr[2]) ^ GFMul(0x0e, arr[3]);
    }
}
/*******************************下面是密钥扩展部分***********************/
/**
 * 将4个 byte 转换为一个 word.
 */
word Word(byte& k1, byte& k2, byte& k3, byte& k4)
{
    word result(0x00000000);
    word temp;
    temp = k1.to_ulong();  // K1
    temp <<= 24;
    result |= temp;
    temp = k2.to_ulong();  // K2
    temp <<= 16;
    result |= temp;
    temp = k3.to_ulong();  // K3
    temp <<= 8;
    result |= temp;
    temp = k4.to_ulong();  // K4
    result |= temp;
```

```
    return result;
}
/**
 *  按字节 循环左移一位
 *  即把[a0, a1, a2, a3]变成[a1, a2, a3, a0]
 */
word RotWord(word rw)
{
    word high = rw << 8;
    word low = rw >> 24;
    return high | low;
}
/**
 *  对输入word中的每一个字节进行S-盒变换
 */
word SubWord(word sw)
{
    word temp;
    for (int i = 0; i < 32; i += 8)
    {
        int row = sw[i + 7] * 8 + sw[i + 6] * 4 + sw[i + 5] * 2 + sw[i + 4];
        int col = sw[i + 3] * 8 + sw[i + 2] * 4 + sw[i + 1] * 2 + sw[i];
        byte val = S_Box[row][col];
        for (int j = 0; j < 8; ++j)
            temp[i + j] = val[j];
    }
    return temp;
}
/**
 *  密钥扩展函数 - 对128位密钥进行扩展得到 w[4*(Nr+1)]
 */
void KeyExpansion(byte key[4 * Nk], word w[4 * (Nr + 1)])
{
    word temp;
    int i = 0;
    // w[]的前4个就是输入的key
    while (i < Nk)
    {
        w[i] = Word(key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3]);
        ++i;
    }
    i = Nk;
    while (i < 4 * (Nr + 1))
    {
        temp = w[i - 1]; // 记录前一个word
        if (i % Nk == 0)
            w[i] = w[i - Nk] ^ SubWord(RotWord(temp)) ^ Rcon[i / Nk - 1];
        else
            w[i] = w[i - Nk] ^ temp;
        ++i;
    }
}
/******************************下面是加密和解密函数**************************/
/*
 *  加密
 */
byte* encrypt(byte in[4 * 4], word w[4 * (Nr + 1)])
{
```

```
    byte temp[16];
    word key[4];
    for (int i = 0; i < 4; ++i)
        key[i] = w[i];
    AddRoundKey(in, key);
    for (int round = 1; round < Nr; ++round)
    {
        SubBytes(in);
        ShiftRows(in);
        MixColumns(in);
        for (int i = 0; i < 4; ++i)
            key[i] = w[4 * round + i];
        AddRoundKey(in, key);
    }
    SubBytes(in);
    ShiftRows(in);
    for (int i = 0; i < 4; ++i)
        key[i] = w[4 * Nr + i];
    AddRoundKey(in, key);
    for (int i = 0; i < 16; i++)
        temp[i] = in[i];
    return temp;
}
/**

 *  解密

 */
byte* decrypt(byte in[4 * 4], word w[4 * (Nr + 1)])
{
    byte temp[16];
    word key[4];
    for (int i = 0; i < 4; ++i)
        key[i] = w[4 * Nr + i];
    AddRoundKey(in, key);
    for (int round = Nr - 1; round > 0; --round)
    {
        InvShiftRows(in);
        InvSubBytes(in);
        for (int i = 0; i < 4; ++i)
            key[i] = w[4 * round + i];
        AddRoundKey(in, key);
        InvMixColumns(in);
    }
    InvShiftRows(in);
    InvSubBytes(in);
    for (int i = 0; i < 4; ++i)
        key[i] = w[i];
    AddRoundKey(in, key);
    for (int i = 0; i < 16; i++)
        temp[i] = in[i];
    return temp;
}
```

aes_main.cpp:

```
/*
```

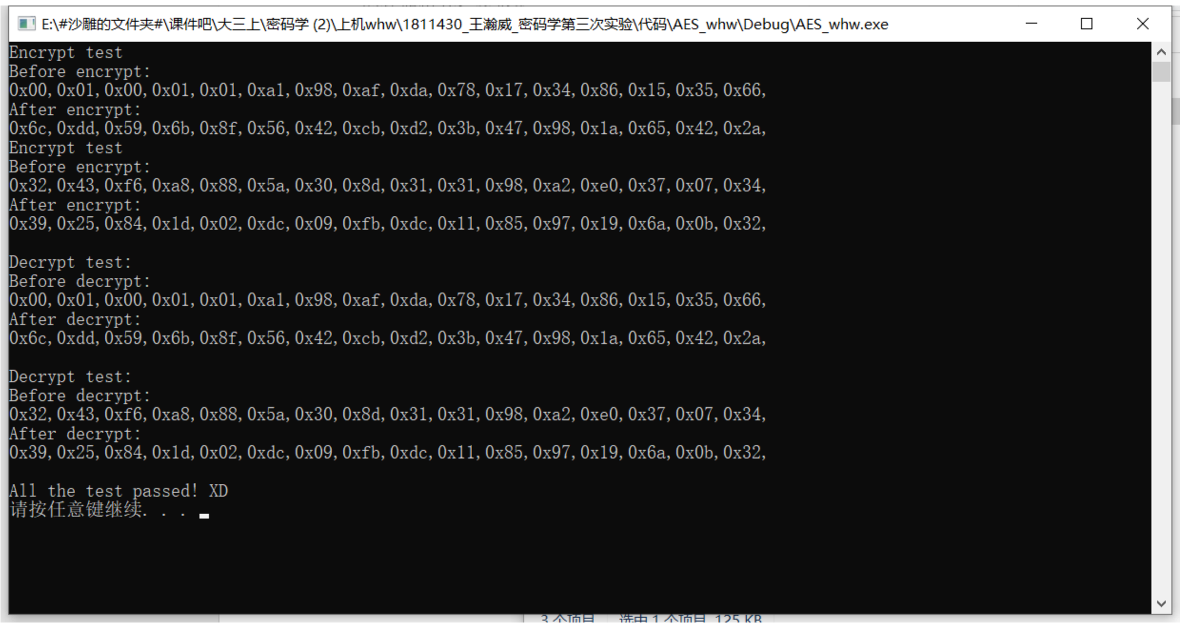
```
aes操作对象主要是字节，getBytesfromcases从样例形式转化为字节形式的string
*/
#pragma once
#include "aes_head.h"
#define CASE_LEN 4
#ifndef string
#include<string>
#endif
bool isEqual(byte* str1, byte* str2) {
    for (int i = 0; i < 16; i++) {
        if (str1[i] != str2[i])
            return false;
    }
    return true;
}
void print_String(string str) {
    char ch, ch1;
    for (int i = 0; i < str.length(); i++) {
        ch = (str[i] >> 4 & 0xf);
        ch1 = str[i] & 0x0f;
        cout << "0x" << hex << int(ch) << hex << int(ch1) << ',';
    }
    cout << endl;
}
void pring_Bytes(byte* key, bool flag = 1) {
    for (int i = 0; i < 16; ++i)
    {
        cout << hex << key[i].to_ulong() << " ";
        if (flag) {
            if ((i + 1) % 4 == 0)
                cout << endl;
        }
    }
    cout << endl;
}
string  getBytesfromcases(string txt);
//从string里取出各个char，赋值给byte数据类型
void convert_type(string str, byte* mm, bool flag = 1);
static const struct aes_test_case {
    int num, mode; // mode 1 = encrypt
    string txt, key, out;//16
} cases[] = {
    { 1, 1,    "0001, 0001, 01a1, 98af, da78, 1734, 8615, 3566",
               "0001, 2001, 7101, 98ae, da79, 1714, 6015, 3594",
               "6cdd, 596b, 8f56, 42cb, d23b, 4798, 1a65, 422a"
    },
    { 2, 1,   "3243, f6a8, 885a, 308d, 3131, 98a2, e037, 0734",
              "2b7e, 1516, 28ae, d2a6, abf7, 1588, 09cf, 4f3c",
              "3925, 841d, 02dc, 09fb, dc11, 8597, 196a, 0b32"
    },//加密用测试数据

    { 3, 0,     "6cdd, 596b, 8f56, 42cb, d23b, 4798, 1a65, 422a",
                "0001, 2001, 7101, 98ae, da79, 1714, 6015, 3594",
                "0001, 0001, 01a1, 98af, da78, 1734, 8615, 3566"
    },
    { 4, 0,     "3925, 841d, 02dc, 09fb, dc11, 8597, 196a, 0b32",
                "2b7e, 1516, 28ae, d2a6, abf7, 1588, 09cf, 4f3c",
                "3243, f6a8, 885a, 308d, 3131, 98a2, e037, 0734"}//解密用测试数据
```

```
};
void test_grade() {
    int sum = 0;
    string txt, out, key0;

    for (int i = 0; i < CASE_LEN; i++)
    {
        byte plain[16], key[16], cipher[16];
        if (cases[i].mode == 1) {
            //兼容函数
            txt = getBytesfromcases(cases[i].txt);
            out = getBytesfromcases(cases[i].out);
            key0 = getBytesfromcases(cases[i].key);
            /*if (i == 0)
                print_String(key0);
            */
            convert_type(txt, plain);
            convert_type(out, cipher);
            convert_type(key0, key, 0);

            word w[4 * (Nr + 1)];
            KeyExpansion(key, w);
            encrypt(plain, w);
            cout << "Encrypt test" << endl;
            if (isEqual(plain, cipher)) {
                cout << "Before encrypt:" << endl;
                print_String(txt);

                //cout << "HEX:" << hex<<plain<<endl;
                cout <<"After encrypt:" << endl;
                print_String(out);
                //printf("%0x", out);
                //cout << "HEX:" << hex<<cipher << endl;
                sum++;
            }
            else
            {
                cout << "Error occured in Encrypt\t" << cases[i].num << "\t
testing case" << endl;
                cout << "should be " << endl; printf("%0x", out); cout << endl;
                    cout<<"result is "<< hex << plain << endl;
                 cout << endl; printf("%0x", plain);
            }
            cout << endl;
        }

        else
        {

            //兼容函数
            txt = getBytesfromcases(cases[i].txt);
            out = getBytesfromcases(cases[i].out);
            key0 = getBytesfromcases(cases[i].key);
            convert_type(txt, cipher);
            convert_type(out, plain);
            convert_type(key0, key, 0);
            word w[4 * (Nr + 1)];
            KeyExpansion(key, w);
```

```
            decrypt(cipher, w);
            cout << endl << "Decrypt test:" << endl;
            if (isEqual(cipher, plain))
            {
                sum++;
                cout << "Before decrypt:" << endl;
                print_String(out);
                cout << "After decrypt:" << endl;
                print_String(txt);
            }
            else
                cout << "Error occured in Decrypt\t" << cases[i].num << "\t
testing case" << endl;
        }
    }
    if (sum != 4) {
        cout << "Success tims is:" << sum << "/4" << endl;
    }
    else
        cout << endl << "All the test passed! XD" << endl;
}
//key正常赋值，而plain和cipher要转置,此时flag==1
void convert_type(string str, byte* mm, bool flag) {
    if (flag) {
        for (int j = 0; j < 4; j++)
            for (int i = 0; i < 4; i++)
                mm[4 * j + i] = str[4 * i + j];//转置

    }
    else {
        for (int i = 0; i < 16; i++)
            mm[i] = str[i];
    }

};

//输入2byte的string，输出char
char getCharfromHex_0x(string str_0x) {
    //大写转小写，未实现
    int result = 0;
    for (int i = 0; i < 2; i++) {
        if (str_0x[i] <= '9' && str_0x[i] >= '0') {
            if (i == 0)
                result += (str_0x[i] - '0') * 16;
            else
                result += (str_0x[i] - '0');
        }
        else if (str_0x[i] <= 'f' && str_0x[i] >= 'a') {
            if (i == 0)
                result += ((str_0x[i] - 'a') + 10) * 16;
            else
                result += (str_0x[i] - 'a') + 10;
        }
    }
    return char(result);
}//checked

//输入形式是去除了0x的16进制，输出byte构成的string
```

```
string  getBytesfromcases(string txt) {
    //txt.erase(std::remove(txt.begin(), txt.end(), ','), txt.end());//去除,号,
用' '代替
    string mid = "";
    string result = "";
    for (int i = 0; i < txt.length(); i++) {
        if ((txt[i] >= '0' && txt[i] <= '9') || (txt[i] <= 'z' && txt[i] >=
'a'))
            result += txt[i];
        else
            continue;
    }
    txt = result;
    result = "";
    for (int i = 0; i < txt.length(); i += 2) {
        mid += txt[i];
        mid += txt[i + 1];
        result += getCharfromHex_0x(mid);
        mid = "";
    }
    return result;
};
int main()

{
    test_grade();
    system("pause");
    return 0;
}
```

执行结果:



五、雪崩实验

对两组数据分别进行进行雪崩实验，初始数据如下：

明文（16进制）：0001, 0001, 01a1, 98af, da78, 1734, 8615, 3566

密钥（16进制）：0001, 2001, 7101, 98ae, da79, 1714, 6015, 3594

密文（16进制）：6cdd, 596b, 8f56, 42cb, d23b, 4798, 1a65, 422a

明文（16进制）：3243, f6a8, 885a, 308d, 3131, 98a2, e037, 0734

密钥（16进制）：2b7e, 1516, 28ae, d2a6, abf7, 1588, 09cf, 4f3c

密文（16进制）：3925, 841d, 02dc, 09fb, dc11, 8597, 196a, 0b32

每次修改8位，得到改变结果对比如下：

```
should be
147f320
result is 012FF6D4

12ff6d4
```

```
should be
13cf7a8
result is 0117FC40

117fc40
```

```
should be
89f6a0
result is 006FF578

6ff578
```

```
should be
d9f5c8
result is 00AFF798

aff798
```

（5+5+5+3）/4*16=5.5 改变明文时平均每次改变了72位

改变密钥进行测试：

```
should be
caf3f8
result is 007BFBA0

7bfba0
```

```
should be
def8a0
result is 00CFF994

cff994
```

```
should be
e6f788
result is 00AFF7D8

aff7d8
```

```
should be
cff4b0
result is 00AFFC74

affc74
```

（5+5+3+4）/4*16=72 改变密钥时平均每次改变了72位。

理论应为64位，与实验结果差异并不大。