实验 **2 AES** 密码体制

一．实验目的  
通过实现 AES 算法，以及对实际的数据进行加密和解密，加深对 AES 密码体制原理的理解。

二．实验环境  
硬件设备： PC 机  
操作系统： windows 2000 操作系统  
编程工具： Matlab， VC++或 Eclipse

三．**AES** 简介  
 信息加密根据采用的密钥类型可以划分为对称密码算法和非对称密码算法。对称密码算法是指加密系统的加密密钥和解密密钥相同，或者虽然不同，但是可以从其中任意一个推导出另一个，更形象的说就是用同一把钥匙开锁和解锁。在对称密码算法的发展历史中曾出现过多种优秀的算法，包括 DES、 3DES、 AES 等。  
 2000年10月，NIST（美国国家标准和技术协会）宣布通过从 15 种候选算法中选出的一项新的高级加密标准（AES）规范 Rijndael。Rijndael 这个名字是从它的两个发明者 Rijmen和 Daemen 的名字得来的。  
 AES 是迭代、对称密钥分组的密码体制，它可以使用 128、192和256位密钥，并且用128 位（16 字节）分组加密和解密数据。基于代换与置换运算。置换是对数据重新进行安排，代换是将一个数据单元替换为另一个。详细的加解密过程请参考官方标准：http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf

四．实验步骤  
1.根据附录提供的算法与代码， 实现AES加密算法，使算法能够对128位的数据加密。要求能够处理密钥为128位，数据分组为128位，也即16字节的情况。

2． 使用下面数据测试你的算法：  
定义加密过程轮数为 Nr=10

密钥  
0X2B,0X7E,0X15,0X16, 0X28,0XAE,0XD2,0XA6, 0XAB,0XF7,0X15,0X88, X09,0XCF,0X4F,0X3C

明文  
0X32,0X43,0XF6,0XA8, 0X88,0X5A,0X30,0X8D, 0X31,0X31,0X98,0XA2,

0XE0,0X37,0X07,0X34

密文  
0X39,0X25,0X84,0X1D, 0X02,0XDC,0X09,0XFB, 0XDC,0X11,0X85,0X97,  
0X19,0X6A,0X0B,0X32  
3．修改加密算法，令Nr＝10，密钥的所有字节都是0。使用步骤2中的明文，计算并记录密文。  
（答案： e5 27 93 6d 04 9f 88 87 2a 49 03 30 5b 97 5b d1）  
4．将步骤2中的明文，改变1位，其它不变。记录密文中有多少字节发生变化。多重复几次，观察 AES 的扩散和混乱的效果。  
5．实现 AES 解密算法，要求同步骤 1。使用步骤2中的数据测试算法。  
6．修改解密算法，令 Nr=10，Key为全0，密文为步骤2中的密文。运行解密算法，记录明文。  
注意，要求将源代码以及每一步的结果写入报告

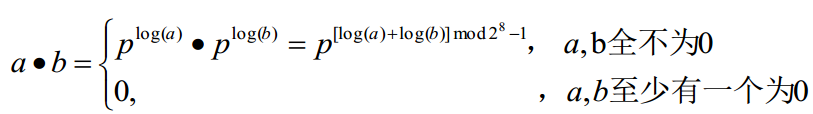
五、思考题  
研究 AES 密文的随机性，探讨使用加密算法做为随机数发生器的可行性。

六、附录  
1． GF(2)多项式乘法运算  
GF(2)上的多项式系数仅有0，1两种情况。 GF（2）的 7 次多项式可以使用 8 位二进制数表示：例如 x7+x5+1 可以表示为（10100001）。算法中的多项式的输入输出都是以这种形式表示的。不可约多项式，一般的对于有限域 GF（28） 上都是 8 次多项式，其最高阶的项的系数比为 1，因此可以省略表示，于是用一个 8 位二进制数表示。  
输入：两个最高次数为 7 次的多项式 A(x),B(x)，一个 GF(2)上的 8 次多项式 M(x)  
输出： A(x)\*B(x) modM(x)  
具体 C 语言形式代码如下：  
BYTE Multi\_P(BYTE Ax, BYTE Bx, BYTE Mx)  
{  
 BYTE Cx=0;  
 for(int i=0; i<8;i++)  
 {  
 Cx = ((Cx & 0x80) ==0)?(Cx<<1):(Mx^(Cx<<1));  
 if((Ax&0x80)!=0)  
 {  
 Cx = Cx ^ Bx;  
 }  
 Ax = Ax<<1;  
 }  
 return Cx;  
}

2．有限域 GF（28）上的运算有限域 GF（28）的元素为多项式，最高次数小于等于7

加法： 多项式加法，也就是对应的系数相加模 2。若以二进制数表示一个多项式，那么两个多项式相加相当于两个二进制数相异或。  
代码如下：  
BYTE GF\_add(BYTE x,BYTE y){ return ( x ^ y ); }

乘法：多项式 A(x),B(x),不可约多项式 M(x)。  
A(x)\*B(x) mod M(x)

计算方法有两种：  
1） 通过直接调用函数 Multi\_P 实现。  
2） 根据有限域的性质，采用指数形式计算。  
即：每个元素 a 都可以表示为 px， p 是域中的本原元, x=log(a)。元素 a,b  
相乘的结果相当于：  
  
于是 ，如果建立 GF（28）所有元素的对数表，及指数表，那么就可以通过查  
表得到结果。显然同第一种方法相比，这种计算方法更快。  
代码如下：  
BYTE GF\_Mul(BYTE x, BYTE y)  
{  
 if(x!=0 && y!=0)  
 {  
 // GF\_2\_8\_pow 是指数表， GF\_2\_8\_log 是对数表  
 return GF\_2\_8\_pow[(GF\_2\_8\_log[x] + GF\_2\_8\_log[y])%0xFF];  
 }  
 else  
 {  
 return 0;  
 }  
}  
至于指数表、 对数表的建立，可以通过调用 Multi\_P 来实现，比如  
p3 mod m(x)=(((p\*p) mod m(x))\*p) mod m(x)  
调用两次函数 Multi\_P 就可以得到 p3 mod m(x)。通过这种方式就可以得到指数  
表，然后由指数表反推就得到对数表。 这里需要特别注意对于 0 的处理。  
逆运算：已知任意一个非零元素 a，求其逆元 a-1，使得 a·a-1=1。  
求逆元的快速的方法，就是通过查表，先求出 log(a),然后，计算逆元的指数 log(a)  
＝，最后通过查指数表求出 p，也就是 a 的逆元。  
特别地： 0 没有逆元，直接返回 0。  
算法如下：  
BYTE GF\_Reverse(BYTE x)  
{  
 // GF\_2\_8\_pow 是指数表， GF\_2\_8\_log 是对数表  
 return x!=0?GF\_2\_8\_pow[(0xFF-GF\_2\_8\_log[x])]:0;  
}  
3． AES 基本运算  
为了简化运算， 接下来介绍的算法都是基于这样的假设：  
a) 算法密钥为 128 位，每个块为 128 位，也就是 16 个字节。  
b) 每种变换算法的输入都是一个块，也就是每次处理 16 个字节  
AES 加密过程由 4 个基本基本动作组成，包括： S 盒变换、行移位变化、列混合变换、轮密钥加法变换。 接下来，我们看看这些细节是如何实现的。  
1） S 盒变换  
算法：通过查表 SBox 实现。以 16 个字节为基本单位，分别对每个字节查表替换以  
相应的表中内容。  
void SubBytes(BYTE state[16])  
{  
 for(int i=0; i<16;i++)  
 {  
 state[i] = SBox[state[i]];  
 }  
 return;  
}  
SBox 可以通过计算生成。 也可以以全局数组的形式直接写入代码中。  
以下方法用来计算 SBox[x]  
BYTE c= 0x63;  
求字节 x 的逆： b = GF\_Reverse(x)  
然后，依次计算每一位  
 for i=0 to 7  
 //bit(b,i)返回字节 b 的第 i 位的值： 0 或 1。 ⊕为异或预算  
 B= bit(b, i) ⊕bit(b, (i+4)mod8) ⊕ bit(b, (i+5) mod8)  
 ⊕ bit(b, (i+6)mod8) ⊕bit(b, (i+7)mod8) ⊕ bit(c, i)  
 setbit(SBox[x], i, B) //设置 SBox[x]的第 i 位为 B  
endfor  
2） 行移位变换  
这个变换很简单：输入为 state[16]。state[0]-state[3]不变 state[4]-state[7]循环左移 1 个字节， state[8]-state[11] 循环左移2个字节，state[12]-state[15] 循环左移 3 个字节。算法如下：  
输入：16个字节构成的一个块 state[16]  
输出：变换后的结果保存在 state[16]  
Void ShiftRows( BYTE state[16])  
{  
 BYTE t[4];  
 for(int i=1; i<4; i++)  
 {  
 for( int j=0; j<4; j++ )  
 {  
 t[j] = state[ i\*4 + (j+i)%4 ];  
 }  
 memcpy(state+4\*i, t, 4);  
 }  
}  
3） 列混合变换  
输入： 16 个字节构成的一个块： state[16]  
输出：变换后的结果保存在 state[16]  
算法：按列处理，每列表示为 s(x) = S[3][i]x3+ S[2][i]x2+ S[1][i]x+ S[0][i]，然后计算  
[{03}x3+{01}x2+{01}X+{02}]\*s(x) mod (x4+1) 得到新的多项式，其系数对应S[0][i]-S[3][i]变换后的字节值。

Void MixColumns(BYTE state[16])  
{  
 BYTE temp[4][4];  
 memcpy(temp, state, 16); //将 state复制到 temp中，temp[i][j]=state[i\*4+j]

for(int i=0; i<4; i++) //第 i 列

{

for(int j=0; j<4; j++) //第 j 行， ⊕为异或运算  
{  
 state[j\*4+i] =  
 GF\_Mul(0x2, temp[j][i])  
 ⊕GF\_ Mul (0x3, temp[(j+1) mod 4][i])

⊕temp[ (j+2)mod 4 ][i] ⊕temp[ (j+3)mod 4 ][i] ;  
 }  
 }  
}  
  
4） 轮密钥加法变换  
每个块的加密都要经过 Nr（不妨取 Nr＝10） 轮变换。 在第 i 轮变换，使用第 i 个扩展密钥 k(i)变换是输入与密钥异或的过程  
代码如下：  
void AddRoundKey(BYTE state[16],BYTE key[16])  
{  
 for(int i=0; i<4; i++)  
 {  
 for( int j=0; j<4; j++ )  
 {  
 state[ i\*4 + j] = state[ i\*4 + j] ⊕key[j\*4+i];  
 }  
 }  
}  
5） 密钥扩展算法  
选取 128 位的密钥，也就是 16 个字节的密钥，通过扩展算法，生成 Nr 个扩展密钥。  
对于第 r 轮(r 从 1 开始计数)用的密钥  
输入： r 表示当前要扩展第 r 轮密钥， r>0  
数组 key 的前 16 字节放入第 r-1 轮密钥  
输出： 数组 key 的后 16 字节写入第 r 轮密钥  
注意： 第 0 轮的密钥也就是初始密钥  
void make\_key(BYTE key[32], int r)  
{  
 const BYTE RC[10]={0x1,0x2,0x4,0x8,0x10,0x20,0x40,0x80,0x1B,0x36};  
 BYTE \*temp = key;  
 DWORD \*dw\_key= (( DWORD\*)(temp));  
 //存储第四个双字，也就是最后 4 个字节  
 DWORD dw3= dw\_key[3];  
 //对第一个字，旋转并替换  
 BYTE t = temp[4\*3+0];  
 temp[4\*3+0] = gSBox[temp[4\*3+1]];  
 temp[4\*3+1] = gSBox[temp[4\*3+2]];  
 temp[4\*3+2] = gSBox[temp[4\*3+3]];  
 temp[4\*3+3] = gSBox[t];  
 //对第一个字，异或  
 temp[4\*3] = temp[4\*3]^RC[r-1];  
 //第 i－1 个字 与 第 i-3 个字  
 for(int i=4; i<7;i++)  
 {  
 dw\_key[i] = dw\_key[i-1]^dw\_key[i-4];  
 }  
 dw\_key[7] = dw\_key[6]^dw3;  
 dw\_key[3] = dw3;  
}  
通过反复调用函数 make\_key 就可以生成任意轮扩展密钥。 对于 128 位的密钥，可  
以取 Nr=10。  
4． AES 加密算法  
这里仅仅描述对 1 个块的加密，对更长的数据，则需要依据不同的模式，使用加密算法。算法如下：  
输入：明文 in ， 密钥 key  
输出：密文 out  
AESEncrypt(BYTE in[16],BYTE out[16]， BYTE key[16])  
{  
 BYTE state[16];  
 int Nr = 10;  
 将 in 看作 4×4 的矩阵，做转置运算： state[ i\*4+j ] = in[ j\*4+i ]  
 生成 10 轮扩展密钥，记录在数组 key\_expand[11][16]中  
 AddRoundKey（state, key\_expand[0], 1）  
 for( round = 1; round<Nr; round++)  
 {  
 SubBytes(state);  
 ShiftRows(state);  
 MixColumns(state);  
 AddRoundKey(state, key\_expand[round]);  
 }将state 看作 4×4 的矩阵，做转置运算： out[ i\*4+j ] = state[ j\*4+i ]  
}  
5． AES 解密算法  
解密算法同加密算法的过程基本相同，具体运算需要 S 盒变换、行移位、列混合的逆运算。 扩展密钥使用顺序相反，先用第 Nr 轮扩展密钥，在使用 Nr－1 轮扩展密钥，直到第 0 轮扩展密钥。 过程不再详述。  
报告要求：  
1） 要求字数不少于 1000 字  
2） 列出参考文献。包括作者，标题，出处，日期等。  
3） 可以两人合作。请在报告中注明你们是如何分工的。

六、实验代码

#include <string.h>

#include <Windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

typedef unsigned char AES\_U8;

typedef unsigned int AES\_U16;

#define AES\_LIB\_H

#define CODE\_MEM\_TYP code

#define Nc (4)

#define Nr (10)

#define Nk (4)

static void ShiftRows(AES\_U8 \*state);

static void InvShiftRows(AES\_U8 \*state);

static void MixSubColumns(AES\_U8 \*state);

static void InvMixSubColumns(AES\_U8 \*state);

static void AddRoundKey(AES\_U8 \*state, AES\_U8 \*key);

void AES\_ExpandKey(AES\_U8 \*key, AES\_U8 \*expKey);

void AES\_Encrypt(AES\_U8 \*in, AES\_U8 \*expKey, AES\_U8 \*out);

void AES\_Decrypt(AES\_U8 \*in, AES\_U8 \*expKey, AES\_U8 \*out);

AES\_U8 key[16];

AES\_U8 M[1000][16];

AES\_U8 C[1000][16];

AES\_U8 Mr[1000][16];

static const AES\_U8 Sbox[256] = {

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 };

static const AES\_U8 InvSbox[256] = {

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 };

static const AES\_U8 Rcon[11] = {

0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36 };

static const AES\_U8 Xtime2Sbox[256] = {

0xc6, 0xf8, 0xee, 0xf6, 0xff, 0xd6, 0xde, 0x91, 0x60, 0x02, 0xce, 0x56, 0xe7, 0xb5, 0x4d, 0xec,

0x8f, 0x1f, 0x89, 0xfa, 0xef, 0xb2, 0x8e, 0xfb, 0x41, 0xb3, 0x5f, 0x45, 0x23, 0x53, 0xe4, 0x9b,

0x75, 0xe1, 0x3d, 0x4c, 0x6c, 0x7e, 0xf5, 0x83, 0x68, 0x51, 0xd1, 0xf9, 0xe2, 0xab, 0x62, 0x2a,

0x08, 0x95, 0x46, 0x9d, 0x30, 0x37, 0x0a, 0x2f, 0x0e, 0x24, 0x1b, 0xdf, 0xcd, 0x4e, 0x7f, 0xea,

0x12, 0x1d, 0x58, 0x34, 0x36, 0xdc, 0xb4, 0x5b, 0xa4, 0x76, 0xb7, 0x7d, 0x52, 0xdd, 0x5e, 0x13,

0xa6, 0xb9, 0x00, 0xc1, 0x40, 0xe3, 0x79, 0xb6, 0xd4, 0x8d, 0x67, 0x72, 0x94, 0x98, 0xb0, 0x85,

0xbb, 0xc5, 0x4f, 0xed, 0x86, 0x9a, 0x66, 0x11, 0x8a, 0xe9, 0x04, 0xfe, 0xa0, 0x78, 0x25, 0x4b,

0xa2, 0x5d, 0x80, 0x05, 0x3f, 0x21, 0x70, 0xf1, 0x63, 0x77, 0xaf, 0x42, 0x20, 0xe5, 0xfd, 0xbf,

0x81, 0x18, 0x26, 0xc3, 0xbe, 0x35, 0x88, 0x2e, 0x93, 0x55, 0xfc, 0x7a, 0xc8, 0xba, 0x32, 0xe6,

0xc0, 0x19, 0x9e, 0xa3, 0x44, 0x54, 0x3b, 0x0b, 0x8c, 0xc7, 0x6b, 0x28, 0xa7, 0xbc, 0x16, 0xad,

0xdb, 0x64, 0x74, 0x14, 0x92, 0x0c, 0x48, 0xb8, 0x9f, 0xbd, 0x43, 0xc4, 0x39, 0x31, 0xd3, 0xf2,

0xd5, 0x8b, 0x6e, 0xda, 0x01, 0xb1, 0x9c, 0x49, 0xd8, 0xac, 0xf3, 0xcf, 0xca, 0xf4, 0x47, 0x10,

0x6f, 0xf0, 0x4a, 0x5c, 0x38, 0x57, 0x73, 0x97, 0xcb, 0xa1, 0xe8, 0x3e, 0x96, 0x61, 0x0d, 0x0f,

0xe0, 0x7c, 0x71, 0xcc, 0x90, 0x06, 0xf7, 0x1c, 0xc2, 0x6a, 0xae, 0x69, 0x17, 0x99, 0x3a, 0x27,

0xd9, 0xeb, 0x2b, 0x22, 0xd2, 0xa9, 0x07, 0x33, 0x2d, 0x3c, 0x15, 0xc9, 0x87, 0xaa, 0x50, 0xa5,

0x03, 0x59, 0x09, 0x1a, 0x65, 0xd7, 0x84, 0xd0, 0x82, 0x29, 0x5a, 0x1e, 0x7b, 0xa8, 0x6d, 0x2c

};

static const AES\_U8 Xtime3Sbox[256] = {

0xa5, 0x84, 0x99, 0x8d, 0x0d, 0xbd, 0xb1, 0x54, 0x50, 0x03, 0xa9, 0x7d, 0x19, 0x62, 0xe6, 0x9a,

0x45, 0x9d, 0x40, 0x87, 0x15, 0xeb, 0xc9, 0x0b, 0xec, 0x67, 0xfd, 0xea, 0xbf, 0xf7, 0x96, 0x5b,

0xc2, 0x1c, 0xae, 0x6a, 0x5a, 0x41, 0x02, 0x4f, 0x5c, 0xf4, 0x34, 0x08, 0x93, 0x73, 0x53, 0x3f,

0x0c, 0x52, 0x65, 0x5e, 0x28, 0xa1, 0x0f, 0xb5, 0x09, 0x36, 0x9b, 0x3d, 0x26, 0x69, 0xcd, 0x9f,

0x1b, 0x9e, 0x74, 0x2e, 0x2d, 0xb2, 0xee, 0xfb, 0xf6, 0x4d, 0x61, 0xce, 0x7b, 0x3e, 0x71, 0x97,

0xf5, 0x68, 0x00, 0x2c, 0x60, 0x1f, 0xc8, 0xed, 0xbe, 0x46, 0xd9, 0x4b, 0xde, 0xd4, 0xe8, 0x4a,

0x6b, 0x2a, 0xe5, 0x16, 0xc5, 0xd7, 0x55, 0x94, 0xcf, 0x10, 0x06, 0x81, 0xf0, 0x44, 0xba, 0xe3,

0xf3, 0xfe, 0xc0, 0x8a, 0xad, 0xbc, 0x48, 0x04, 0xdf, 0xc1, 0x75, 0x63, 0x30, 0x1a, 0x0e, 0x6d,

0x4c, 0x14, 0x35, 0x2f, 0xe1, 0xa2, 0xcc, 0x39, 0x57, 0xf2, 0x82, 0x47, 0xac, 0xe7, 0x2b, 0x95,

0xa0, 0x98, 0xd1, 0x7f, 0x66, 0x7e, 0xab, 0x83, 0xca, 0x29, 0xd3, 0x3c, 0x79, 0xe2, 0x1d, 0x76,

0x3b, 0x56, 0x4e, 0x1e, 0xdb, 0x0a, 0x6c, 0xe4, 0x5d, 0x6e, 0xef, 0xa6, 0xa8, 0xa4, 0x37, 0x8b,

0x32, 0x43, 0x59, 0xb7, 0x8c, 0x64, 0xd2, 0xe0, 0xb4, 0xfa, 0x07, 0x25, 0xaf, 0x8e, 0xe9, 0x18,

0xd5, 0x88, 0x6f, 0x72, 0x24, 0xf1, 0xc7, 0x51, 0x23, 0x7c, 0x9c, 0x21, 0xdd, 0xdc, 0x86, 0x85,

0x90, 0x42, 0xc4, 0xaa, 0xd8, 0x05, 0x01, 0x12, 0xa3, 0x5f, 0xf9, 0xd0, 0x91, 0x58, 0x27, 0xb9,

0x38, 0x13, 0xb3, 0x33, 0xbb, 0x70, 0x89, 0xa7, 0xb6, 0x22, 0x92, 0x20, 0x49, 0xff, 0x78, 0x7a,

0x8f, 0xf8, 0x80, 0x17, 0xda, 0x31, 0xc6, 0xb8, 0xc3, 0xb0, 0x77, 0x11, 0xcb, 0xfc, 0xd6, 0x3a

};

static const AES\_U8 Xtime9[256] = {

0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53, 0x6c, 0x65, 0x7e, 0x77,

0x90, 0x99, 0x82, 0x8b, 0xb4, 0xbd, 0xa6, 0xaf, 0xd8, 0xd1, 0xca, 0xc3, 0xfc, 0xf5, 0xee, 0xe7,

0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,

0xab, 0xa2, 0xb9, 0xb0, 0x8f, 0x86, 0x9d, 0x94, 0xe3, 0xea, 0xf1, 0xf8, 0xc7, 0xce, 0xd5, 0xdc,

0x76, 0x7f, 0x64, 0x6d, 0x52, 0x5b, 0x40, 0x49, 0x3e, 0x37, 0x2c, 0x25, 0x1a, 0x13, 0x08, 0x01,

0xe6, 0xef, 0xf4, 0xfd, 0xc2, 0xcb, 0xd0, 0xd9, 0xae, 0xa7, 0xbc, 0xb5, 0x8a, 0x83, 0x98, 0x91,

0x4d, 0x44, 0x5f, 0x56, 0x69, 0x60, 0x7b, 0x72, 0x05, 0x0c, 0x17, 0x1e, 0x21, 0x28, 0x33, 0x3a,

0xdd, 0xd4, 0xcf, 0xc6, 0xf9, 0xf0, 0xeb, 0xe2, 0x95, 0x9c, 0x87, 0x8e, 0xb1, 0xb8, 0xa3, 0xaa,

0xec, 0xe5, 0xfe, 0xf7, 0xc8, 0xc1, 0xda, 0xd3, 0xa4, 0xad, 0xb6, 0xbf, 0x80, 0x89, 0x92, 0x9b,

0x7c, 0x75, 0x6e, 0x67, 0x58, 0x51, 0x4a, 0x43, 0x34, 0x3d, 0x26, 0x2f, 0x10, 0x19, 0x02, 0x0b,

0xd7, 0xde, 0xc5, 0xcc, 0xf3, 0xfa, 0xe1, 0xe8, 0x9f, 0x96, 0x8d, 0x84, 0xbb, 0xb2, 0xa9, 0xa0,

0x47, 0x4e, 0x55, 0x5c, 0x63, 0x6a, 0x71, 0x78, 0x0f, 0x06, 0x1d, 0x14, 0x2b, 0x22, 0x39, 0x30,

0x9a, 0x93, 0x88, 0x81, 0xbe, 0xb7, 0xac, 0xa5, 0xd2, 0xdb, 0xc0, 0xc9, 0xf6, 0xff, 0xe4, 0xed,

0x0a, 0x03, 0x18, 0x11, 0x2e, 0x27, 0x3c, 0x35, 0x42, 0x4b, 0x50, 0x59, 0x66, 0x6f, 0x74, 0x7d,

0xa1, 0xa8, 0xb3, 0xba, 0x85, 0x8c, 0x97, 0x9e, 0xe9, 0xe0, 0xfb, 0xf2, 0xcd, 0xc4, 0xdf, 0xd6,

0x31, 0x38, 0x23, 0x2a, 0x15, 0x1c, 0x07, 0x0e, 0x79, 0x70, 0x6b, 0x62, 0x5d, 0x54, 0x4f, 0x46 };

static const AES\_U8 XtimeB[256] = {

0x00, 0x0b, 0x16, 0x1d, 0x2c, 0x27, 0x3a, 0x31, 0x58, 0x53, 0x4e, 0x45, 0x74, 0x7f, 0x62, 0x69,

0xb0, 0xbb, 0xa6, 0xad, 0x9c, 0x97, 0x8a, 0x81, 0xe8, 0xe3, 0xfe, 0xf5, 0xc4, 0xcf, 0xd2, 0xd9,

0x7b, 0x70, 0x6d, 0x66, 0x57, 0x5c, 0x41, 0x4a, 0x23, 0x28, 0x35, 0x3e, 0x0f, 0x04, 0x19, 0x12,

0xcb, 0xc0, 0xdd, 0xd6, 0xe7, 0xec, 0xf1, 0xfa, 0x93, 0x98, 0x85, 0x8e, 0xbf, 0xb4, 0xa9, 0xa2,

0xf6, 0xfd, 0xe0, 0xeb, 0xda, 0xd1, 0xcc, 0xc7, 0xae, 0xa5, 0xb8, 0xb3, 0x82, 0x89, 0x94, 0x9f,

0x46, 0x4d, 0x50, 0x5b, 0x6a, 0x61, 0x7c, 0x77, 0x1e, 0x15, 0x08, 0x03, 0x32, 0x39, 0x24, 0x2f,

0x8d, 0x86, 0x9b, 0x90, 0xa1, 0xaa, 0xb7, 0xbc, 0xd5, 0xde, 0xc3, 0xc8, 0xf9, 0xf2, 0xef, 0xe4,

0x3d, 0x36, 0x2b, 0x20, 0x11, 0x1a, 0x07, 0x0c, 0x65, 0x6e, 0x73, 0x78, 0x49, 0x42, 0x5f, 0x54,

0xf7, 0xfc, 0xe1, 0xea, 0xdb, 0xd0, 0xcd, 0xc6, 0xaf, 0xa4, 0xb9, 0xb2, 0x83, 0x88, 0x95, 0x9e,

0x47, 0x4c, 0x51, 0x5a, 0x6b, 0x60, 0x7d, 0x76, 0x1f, 0x14, 0x09, 0x02, 0x33, 0x38, 0x25, 0x2e,

0x8c, 0x87, 0x9a, 0x91, 0xa0, 0xab, 0xb6, 0xbd, 0xd4, 0xdf, 0xc2, 0xc9, 0xf8, 0xf3, 0xee, 0xe5,

0x3c, 0x37, 0x2a, 0x21, 0x10, 0x1b, 0x06, 0x0d, 0x64, 0x6f, 0x72, 0x79, 0x48, 0x43, 0x5e, 0x55,

0x01, 0x0a, 0x17, 0x1c, 0x2d, 0x26, 0x3b, 0x30, 0x59, 0x52, 0x4f, 0x44, 0x75, 0x7e, 0x63, 0x68,

0xb1, 0xba, 0xa7, 0xac, 0x9d, 0x96, 0x8b, 0x80, 0xe9, 0xe2, 0xff, 0xf4, 0xc5, 0xce, 0xd3, 0xd8,

0x7a, 0x71, 0x6c, 0x67, 0x56, 0x5d, 0x40, 0x4b, 0x22, 0x29, 0x34, 0x3f, 0x0e, 0x05, 0x18, 0x13,

0xca, 0xc1, 0xdc, 0xd7, 0xe6, 0xed, 0xf0, 0xfb, 0x92, 0x99, 0x84, 0x8f, 0xbe, 0xb5, 0xa8, 0xa3 };

static const AES\_U8 XtimeD[256] = {

0x00, 0x0d, 0x1a, 0x17, 0x34, 0x39, 0x2e, 0x23, 0x68, 0x65, 0x72, 0x7f, 0x5c, 0x51, 0x46, 0x4b,

0xd0, 0xdd, 0xca, 0xc7, 0xe4, 0xe9, 0xfe, 0xf3, 0xb8, 0xb5, 0xa2, 0xaf, 0x8c, 0x81, 0x96, 0x9b,

0xbb, 0xb6, 0xa1, 0xac, 0x8f, 0x82, 0x95, 0x98, 0xd3, 0xde, 0xc9, 0xc4, 0xe7, 0xea, 0xfd, 0xf0,

0x6b, 0x66, 0x71, 0x7c, 0x5f, 0x52, 0x45, 0x48, 0x03, 0x0e, 0x19, 0x14, 0x37, 0x3a, 0x2d, 0x20,

0x6d, 0x60, 0x77, 0x7a, 0x59, 0x54, 0x43, 0x4e, 0x05, 0x08, 0x1f, 0x12, 0x31, 0x3c, 0x2b, 0x26,

0xbd, 0xb0, 0xa7, 0xaa, 0x89, 0x84, 0x93, 0x9e, 0xd5, 0xd8, 0xcf, 0xc2, 0xe1, 0xec, 0xfb, 0xf6,

0xd6, 0xdb, 0xcc, 0xc1, 0xe2, 0xef, 0xf8, 0xf5, 0xbe, 0xb3, 0xa4, 0xa9, 0x8a, 0x87, 0x90, 0x9d,

0x06, 0x0b, 0x1c, 0x11, 0x32, 0x3f, 0x28, 0x25, 0x6e, 0x63, 0x74, 0x79, 0x5a, 0x57, 0x40, 0x4d,

0xda, 0xd7, 0xc0, 0xcd, 0xee, 0xe3, 0xf4, 0xf9, 0xb2, 0xbf, 0xa8, 0xa5, 0x86, 0x8b, 0x9c, 0x91,

0x0a, 0x07, 0x10, 0x1d, 0x3e, 0x33, 0x24, 0x29, 0x62, 0x6f, 0x78, 0x75, 0x56, 0x5b, 0x4c, 0x41,

0x61, 0x6c, 0x7b, 0x76, 0x55, 0x58, 0x4f, 0x42, 0x09, 0x04, 0x13, 0x1e, 0x3d, 0x30, 0x27, 0x2a,

0xb1, 0xbc, 0xab, 0xa6, 0x85, 0x88, 0x9f, 0x92, 0xd9, 0xd4, 0xc3, 0xce, 0xed, 0xe0, 0xf7, 0xfa,

0xb7, 0xba, 0xad, 0xa0, 0x83, 0x8e, 0x99, 0x94, 0xdf, 0xd2, 0xc5, 0xc8, 0xeb, 0xe6, 0xf1, 0xfc,

0x67, 0x6a, 0x7d, 0x70, 0x53, 0x5e, 0x49, 0x44, 0x0f, 0x02, 0x15, 0x18, 0x3b, 0x36, 0x21, 0x2c,

0x0c, 0x01, 0x16, 0x1b, 0x38, 0x35, 0x22, 0x2f, 0x64, 0x69, 0x7e, 0x73, 0x50, 0x5d, 0x4a, 0x47,

0xdc, 0xd1, 0xc6, 0xcb, 0xe8, 0xe5, 0xf2, 0xff, 0xb4, 0xb9, 0xae, 0xa3, 0x80, 0x8d, 0x9a, 0x97 };

static const AES\_U8 XtimeE[256] = {

0x00, 0x0e, 0x1c, 0x12, 0x38, 0x36, 0x24, 0x2a, 0x70, 0x7e, 0x6c, 0x62, 0x48, 0x46, 0x54, 0x5a,

0xe0, 0xee, 0xfc, 0xf2, 0xd8, 0xd6, 0xc4, 0xca, 0x90, 0x9e, 0x8c, 0x82, 0xa8, 0xa6, 0xb4, 0xba,

0xdb, 0xd5, 0xc7, 0xc9, 0xe3, 0xed, 0xff, 0xf1, 0xab, 0xa5, 0xb7, 0xb9, 0x93, 0x9d, 0x8f, 0x81,

0x3b, 0x35, 0x27, 0x29, 0x03, 0x0d, 0x1f, 0x11, 0x4b, 0x45, 0x57, 0x59, 0x73, 0x7d, 0x6f, 0x61,

0xad, 0xa3, 0xb1, 0xbf, 0x95, 0x9b, 0x89, 0x87, 0xdd, 0xd3, 0xc1, 0xcf, 0xe5, 0xeb, 0xf9, 0xf7,

0x4d, 0x43, 0x51, 0x5f, 0x75, 0x7b, 0x69, 0x67, 0x3d, 0x33, 0x21, 0x2f, 0x05, 0x0b, 0x19, 0x17,

0x76, 0x78, 0x6a, 0x64, 0x4e, 0x40, 0x52, 0x5c, 0x06, 0x08, 0x1a, 0x14, 0x3e, 0x30, 0x22, 0x2c,

0x96, 0x98, 0x8a, 0x84, 0xae, 0xa0, 0xb2, 0xbc, 0xe6, 0xe8, 0xfa, 0xf4, 0xde, 0xd0, 0xc2, 0xcc,

0x41, 0x4f, 0x5d, 0x53, 0x79, 0x77, 0x65, 0x6b, 0x31, 0x3f, 0x2d, 0x23, 0x09, 0x07, 0x15, 0x1b,

0xa1, 0xaf, 0xbd, 0xb3, 0x99, 0x97, 0x85, 0x8b, 0xd1, 0xdf, 0xcd, 0xc3, 0xe9, 0xe7, 0xf5, 0xfb,

0x9a, 0x94, 0x86, 0x88, 0xa2, 0xac, 0xbe, 0xb0, 0xea, 0xe4, 0xf6, 0xf8, 0xd2, 0xdc, 0xce, 0xc0,

0x7a, 0x74, 0x66, 0x68, 0x42, 0x4c, 0x5e, 0x50, 0x0a, 0x04, 0x16, 0x18, 0x32, 0x3c, 0x2e, 0x20,

0xec, 0xe2, 0xf0, 0xfe, 0xd4, 0xda, 0xc8, 0xc6, 0x9c, 0x92, 0x80, 0x8e, 0xa4, 0xaa, 0xb8, 0xb6,

0x0c, 0x02, 0x10, 0x1e, 0x34, 0x3a, 0x28, 0x26, 0x7c, 0x72, 0x60, 0x6e, 0x44, 0x4a, 0x58, 0x56,

0x37, 0x39, 0x2b, 0x25, 0x0f, 0x01, 0x13, 0x1d, 0x47, 0x49, 0x5b, 0x55, 0x7f, 0x71, 0x63, 0x6d,

0xd7, 0xd9, 0xcb, 0xc5, 0xef, 0xe1, 0xf3, 0xfd, 0xa7, 0xa9, 0xbb, 0xb5, 0x9f, 0x91, 0x83, 0x8d };

static void ShiftRows(AES\_U8 \*state)

{

AES\_U8 tmp;

state[0] = Sbox[state[0]];

state[4] = Sbox[state[4]];

state[8] = Sbox[state[8]];

state[12] = Sbox[state[12]];

tmp = Sbox[state[1]];

state[1] = Sbox[state[5]];

state[5] = Sbox[state[9]];

state[9] = Sbox[state[13]];

state[13] = tmp;

tmp = Sbox[state[2]];

state[2] = Sbox[state[10]];

state[10] = tmp;

tmp = Sbox[state[6]];

state[6] = Sbox[state[14]];

state[14] = tmp;

tmp = Sbox[state[15]];

state[15] = Sbox[state[11]];

state[11] = Sbox[state[7]];

state[7] = Sbox[state[3]];

state[3] = tmp;

}

static void InvShiftRows(AES\_U8 \*state)

{

AES\_U8 tmp;

state[0] = InvSbox[state[0]];

state[4] = InvSbox[state[4]];

state[8] = InvSbox[state[8]];

state[12] = InvSbox[state[12]];

tmp = InvSbox[state[13]];

state[13] = InvSbox[state[9]];

state[9] = InvSbox[state[5]];

state[5] = InvSbox[state[1]];

state[1] = tmp;

tmp = InvSbox[state[2]];

state[2] = InvSbox[state[10]];

state[10] = tmp;

tmp = InvSbox[state[6]];

state[6] = InvSbox[state[14]];

state[14] = tmp;

tmp = InvSbox[state[3]];

state[3] = InvSbox[state[7]];

state[7] = InvSbox[state[11]];

state[11] = InvSbox[state[15]];

state[15] = tmp;

}

static void MixSubColumns(AES\_U8 \*state)

{

AES\_U8 newstate[4 \* Nc];

newstate[0] = Xtime2Sbox[state[0]] ^ Xtime3Sbox[state[5]] ^ Sbox[state[10]] ^ Sbox[state[15]];

newstate[1] = Sbox[state[0]] ^ Xtime2Sbox[state[5]] ^ Xtime3Sbox[state[10]] ^ Sbox[state[15]];

newstate[2] = Sbox[state[0]] ^ Sbox[state[5]] ^ Xtime2Sbox[state[10]] ^ Xtime3Sbox[state[15]];

newstate[3] = Xtime3Sbox[state[0]] ^ Sbox[state[5]] ^ Sbox[state[10]] ^ Xtime2Sbox[state[15]];

newstate[4] = Xtime2Sbox[state[4]] ^ Xtime3Sbox[state[9]] ^ Sbox[state[14]] ^ Sbox[state[3]];

newstate[5] = Sbox[state[4]] ^ Xtime2Sbox[state[9]] ^ Xtime3Sbox[state[14]] ^ Sbox[state[3]];

newstate[6] = Sbox[state[4]] ^ Sbox[state[9]] ^ Xtime2Sbox[state[14]] ^ Xtime3Sbox[state[3]];

newstate[7] = Xtime3Sbox[state[4]] ^ Sbox[state[9]] ^ Sbox[state[14]] ^ Xtime2Sbox[state[3]];

newstate[8] = Xtime2Sbox[state[8]] ^ Xtime3Sbox[state[13]] ^ Sbox[state[2]] ^ Sbox[state[7]];

newstate[9] = Sbox[state[8]] ^ Xtime2Sbox[state[13]] ^ Xtime3Sbox[state[2]] ^ Sbox[state[7]];

newstate[10] = Sbox[state[8]] ^ Sbox[state[13]] ^ Xtime2Sbox[state[2]] ^ Xtime3Sbox[state[7]];

newstate[11] = Xtime3Sbox[state[8]] ^ Sbox[state[13]] ^ Sbox[state[2]] ^ Xtime2Sbox[state[7]];

newstate[12] = Xtime2Sbox[state[12]] ^ Xtime3Sbox[state[1]] ^ Sbox[state[6]] ^ Sbox[state[11]];

newstate[13] = Sbox[state[12]] ^ Xtime2Sbox[state[1]] ^ Xtime3Sbox[state[6]] ^ Sbox[state[11]];

newstate[14] = Sbox[state[12]] ^ Sbox[state[1]] ^ Xtime2Sbox[state[6]] ^ Xtime3Sbox[state[11]];

newstate[15] = Xtime3Sbox[state[12]] ^ Sbox[state[1]] ^ Sbox[state[6]] ^ Xtime2Sbox[state[11]];

memcpy(state, newstate, sizeof(newstate));

}

static void InvMixSubColumns(AES\_U8 \*state)

{

AES\_U8 newstate[4 \* Nc];

AES\_U16 i;

newstate[0] = XtimeE[state[0]] ^ XtimeB[state[1]] ^ XtimeD[state[2]] ^ Xtime9[state[3]];

newstate[5] = Xtime9[state[0]] ^ XtimeE[state[1]] ^ XtimeB[state[2]] ^ XtimeD[state[3]];

newstate[10] = XtimeD[state[0]] ^ Xtime9[state[1]] ^ XtimeE[state[2]] ^ XtimeB[state[3]];

newstate[15] = XtimeB[state[0]] ^ XtimeD[state[1]] ^ Xtime9[state[2]] ^ XtimeE[state[3]];

newstate[4] = XtimeE[state[4]] ^ XtimeB[state[5]] ^ XtimeD[state[6]] ^ Xtime9[state[7]];

newstate[9] = Xtime9[state[4]] ^ XtimeE[state[5]] ^ XtimeB[state[6]] ^ XtimeD[state[7]];

newstate[14] = XtimeD[state[4]] ^ Xtime9[state[5]] ^ XtimeE[state[6]] ^ XtimeB[state[7]];

newstate[3] = XtimeB[state[4]] ^ XtimeD[state[5]] ^ Xtime9[state[6]] ^ XtimeE[state[7]];

newstate[8] = XtimeE[state[8]] ^ XtimeB[state[9]] ^ XtimeD[state[10]] ^ Xtime9[state[11]];

newstate[13] = Xtime9[state[8]] ^ XtimeE[state[9]] ^ XtimeB[state[10]] ^ XtimeD[state[11]];

newstate[2] = XtimeD[state[8]] ^ Xtime9[state[9]] ^ XtimeE[state[10]] ^ XtimeB[state[11]];

newstate[7] = XtimeB[state[8]] ^ XtimeD[state[9]] ^ Xtime9[state[10]] ^ XtimeE[state[11]];

newstate[12] = XtimeE[state[12]] ^ XtimeB[state[13]] ^ XtimeD[state[14]] ^ Xtime9[state[15]];

newstate[1] = Xtime9[state[12]] ^ XtimeE[state[13]] ^ XtimeB[state[14]] ^ XtimeD[state[15]];

newstate[6] = XtimeD[state[12]] ^ Xtime9[state[13]] ^ XtimeE[state[14]] ^ XtimeB[state[15]];

newstate[11] = XtimeB[state[12]] ^ XtimeD[state[13]] ^ Xtime9[state[14]] ^ XtimeE[state[15]];

for (i = 0; i<(4 \* Nc); i++)

{

state[i] = InvSbox[newstate[i]];

}

}

static void AddRoundKey(AES\_U8 \*state, AES\_U8 \*key)

{

AES\_U16 idx;

for (idx = 0; idx<4; idx++)

{

state[idx] ^= key[idx];

}

}

void AES\_ExpandKey(AES\_U8 \*key, AES\_U8 \*expKey)

{

AES\_U8 tmp0, tmp1, tmp2, tmp3, tmp4;

AES\_U16 idx;

for (idx = 0; idx<Nk; idx++)

{

expKey[4 \* idx + 0] = key[4 \* idx + 0];

expKey[4 \* idx + 1] = key[4 \* idx + 1];

expKey[4 \* idx + 2] = key[4 \* idx + 2];

expKey[4 \* idx + 3] = key[4 \* idx + 3];

}

for (idx = Nk; idx<(Nc \* (Nr + 1)); idx++)

{

tmp0 = expKey[4 \* idx - 4];

tmp1 = expKey[4 \* idx - 3];

tmp2 = expKey[4 \* idx - 2];

tmp3 = expKey[4 \* idx - 1];

if (!(idx % Nk))

{

tmp4 = tmp3;

tmp3 = Sbox[tmp0];

tmp0 = Sbox[tmp1] ^ Rcon[idx / Nk];

tmp1 = Sbox[tmp2];

tmp2 = Sbox[tmp4];

}

expKey[4 \* idx + 0] = expKey[4 \* idx - 4 \* Nk + 0] ^ tmp0;

expKey[4 \* idx + 1] = expKey[4 \* idx - 4 \* Nk + 1] ^ tmp1;

expKey[4 \* idx + 2] = expKey[4 \* idx - 4 \* Nk + 2] ^ tmp2;

expKey[4 \* idx + 3] = expKey[4 \* idx - 4 \* Nk + 3] ^ tmp3;

}

}

void AES\_Encrypt(AES\_U8 \*in, AES\_U8 \*expKey, AES\_U8 \*out)

{

AES\_U8 state[Nc \* 4];

AES\_U16 round;

AES\_U16 idx;

for (idx = 0; idx<Nc; idx++)

{

state[4 \* idx + 0] = \*in++;

state[4 \* idx + 1] = \*in++;

state[4 \* idx + 2] = \*in++;

state[4 \* idx + 3] = \*in++;

}

AddRoundKey((AES\_U8 \*)state, (AES\_U8 \*)expKey);

for (round = 1; round<(Nr + 1); round++)

{

if (round < Nr)

{

MixSubColumns(state);

}

else

{

ShiftRows(state);

}

AddRoundKey((AES\_U8 \*)state, (AES\_U8 \*)expKey + round \* Nc);

}

for (idx = 0; idx<Nc; idx++)

{

\*out++ = state[4 \* idx + 0];

\*out++ = state[4 \* idx + 1];

\*out++ = state[4 \* idx + 2];

\*out++ = state[4 \* idx + 3];

}

}

void AES\_Decrypt(AES\_U8 \*in, AES\_U8 \*expKey, AES\_U8 \*out)

{

AES\_U8 state[Nc \* 4];

AES\_U16 round;

AES\_U16 idx;

for (idx = 0; idx<Nc; idx++)

{

state[4 \* idx + 0] = \*in++;

state[4 \* idx + 1] = \*in++;

state[4 \* idx + 2] = \*in++;

state[4 \* idx + 3] = \*in++;

}

AddRoundKey((AES\_U8 \*)state, (AES\_U8 \*)expKey + Nr \* Nc);

round = Nr;

InvShiftRows(state);

while (round--)

{

AddRoundKey((AES\_U8 \*)state, (AES\_U8 \*)expKey + round \* Nc);

if (round)

{

InvMixSubColumns(state);

}

}

for (idx = 0; idx<Nc; idx++)

{

\*out++ = state[4 \* idx + 0];

\*out++ = state[4 \* idx + 1];

\*out++ = state[4 \* idx + 2];

\*out++ = state[4 \* idx + 3];

}

}

int main(int argc, char \*argv[])

{

AES\_U8 expKey[4 \* Nc \* (Nr + 1)];

AES\_U8 temp[16];

int i, j;

FILE \*fc = NULL;

FILE \*fp = NULL;

FILE \*fd = NULL;

fopen\_s(&fc, "ciphertext.txt", "wb");

fopen\_s(&fp, "plaintext.txt", "wb");

fopen\_s(&fd, "decry.txt", "wb");

printf("Please input keys: (16 hexadecimal numbers, splitted by space)\n\n");

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

scanf\_s("%x", &key[i]);

AES\_ExpandKey(key, expKey);

srand((unsigned)time(NULL));

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

C[0][j] = rand() % 256;

for (i = 0; i<1000; i++)

{

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

{

M[i][j] = rand() % 256;

fprintf(fp, "%x", M[i][j]);

temp[j] = M[i][j] ^ C[i - 1][j];

}

fprintf(fp, "\n");

AES\_Encrypt(temp, expKey, C[i]);

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

fprintf(fc, "%x", C[i][j]);

fprintf(fc, "\n");

}

for (i = 0; i<1000; i++)

{

AES\_Decrypt(C[i], expKey, Mr[i]);

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

{

Mr[i][j] = Mr[i][j] ^ C[i - 1][j];

fprintf(fd, "%x", Mr[i][j]);

}

fprintf(fd, "\n");

}

fclose(fp);

fclose(fc);

fclose(fd);

system("PAUSE");

return 0;

}