密码算法实验

[实验目的]

1.掌握密码学中经典的对称密码算法 AES、RC4 的算法原理。

2.掌握 AES、RC4 的算法流程和实现方法。

[实验预备]

1.AES 算法的基本原理和特点。

2.流密码 RC4 的密钥流生成以及 S 盒初始化过程。

[实验内容]

1. 分析 AES、RC4 的实现过程。
2. 用程序设计语言将算法过程编程实现。
3. 完成字符串数据的加密运算和解密运算

   输入十六进制明文：112233445566778899００AABBCCDDEEFF

   输入十六进制密钥：13579BDF02468ACE1234567890ABCDEF

[实验步骤]

1. 预习 AES、RC4 算法。
2. 写出算法流程，用程序设计语言将算法过程编程实现。
3. 输入指定的明文、密钥进行实验，验证结果。
4. 自己选择不同的输入，记录输出结果。

[问题讨论]

1.改变明文或密钥中的一个比特值可能影响 AES 值中的多少比特?

2.在 RC4 的密钥流生成中，改变初始密钥的一个比特值可能影响输出中的多少比特?

3.分析实验中在编辑、编译、运行等各环节中所出现的问题及解决方法。

一、AES 加解密算法

［AES 算法简介］

高级加密标准（缩写：AES），在密码学中又称 Rijndae 加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的 DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于 2001 年 11 月 26 日发布于 FIPS PUB 197，并在 2002 年 5 月 26 日成为有效的标准。2006 年，高级加密标准已然成为对称密钥加密中最流行的算法之一。

［AES 算法流程］

1.AES 算法基本变换包括以下四个步骤

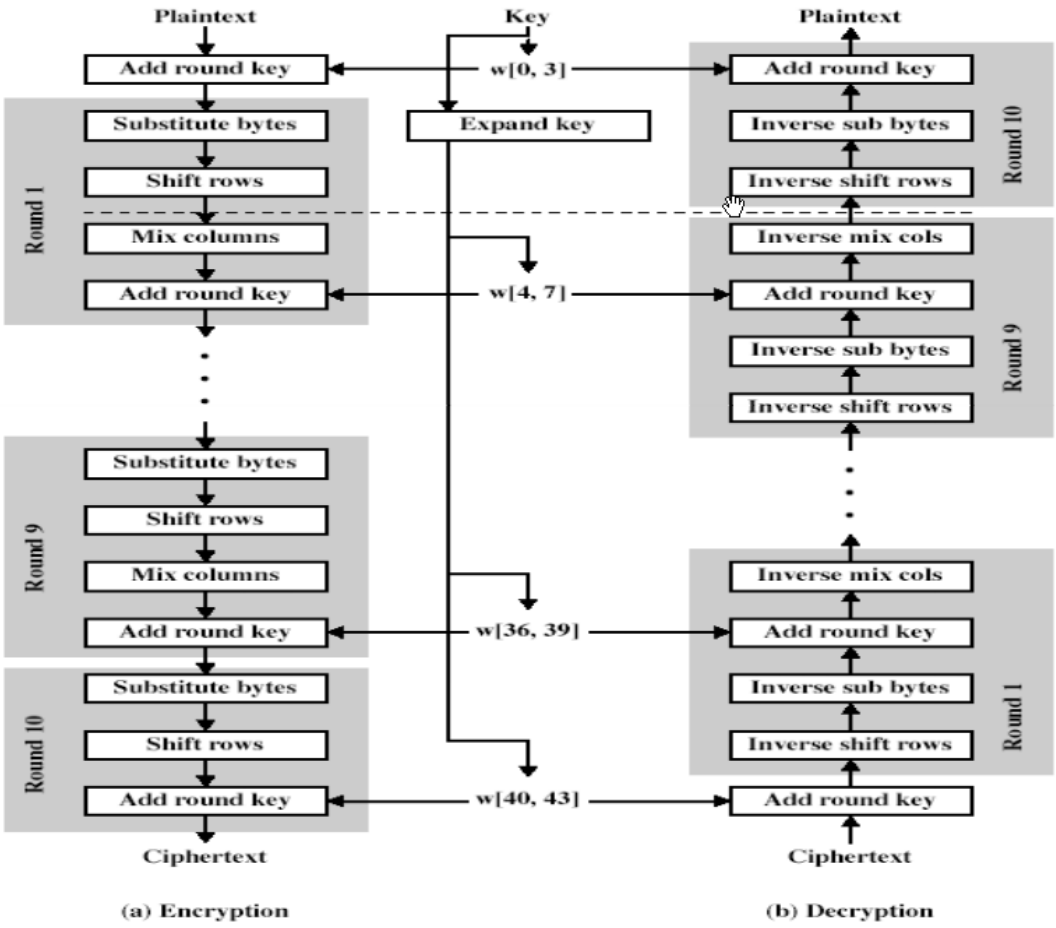
SubBytes（字节替代）

ShiftRows（行移位）

MixColumns（列混淆）

AddRoundKey(轮密钥加)

2.算法的整体描述



3.输入明文的组织排列方式

如图可以看出，明文以列优先形式存储在 4*4 的矩阵中

具体实现代码如下：

```
unsigned char* AES::Cipher(unsigned char* input)
{
    unsigned char state[4][4];
    int i,r,c;

    for(r=0; r<4; r++)
    {
        for(c=0; c<4 ;c++)
        {
            state[r][c] = input[c*4+r];//按列优先输入
        }
    }
```

4. ByteSubstitution（字节替代）

非线性的字节替代，单独处理每个字节：

求该字节在有限域 $GF(2^8)$上的乘法逆，"0"被映射为自身，即对于 $\alpha \in GF(2^8)$，求 $\beta \in GF(2^8)$，

使得 $\alpha \cdot \beta=\beta \cdot \alpha=1 \bmod(x^8+x^4+x^2+x+1)$。

对上一步求得的乘法逆作仿射变换

$$y_i=x_i + x_{(i+4)\bmod 8} + x_{(i+6)\bmod 8} + x_{(i+7)\bmod 8} + c_i$$

(其中 c_i 是 63_{10} 即 01100011_2 的第 i 位)，用矩阵表示为

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

置换表（S-Box）如下：

```
unsigned char sBox[] =
{ /*  0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f */
    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, /*1*/
    0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15, /*2*/
    0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75, /*3*/
    0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84, /*4*/
    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, /*6*/
    0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2, /*7*/
    0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73, /*8*/
    0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb, /*9*/
    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, /*b*/
    0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a, /*c*/
    0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e, /*d*/
    0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf, /*e*/
    0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16  /*f*/
};
```

逆置换表，解密时使用

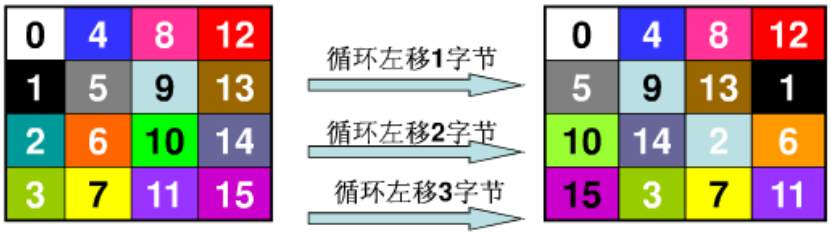
```
unsigned char invsBox[256] =
{ /*  0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f */
    0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb, /*0*/
    0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb, /*1*/
    0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e, /*2*/
    0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25, /*3*/
    0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92, /*4*/
    0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84, /*5*/
    0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06, /*6*/
    0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b, /*7*/
    0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73, /*8*/
    0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e, /*9*/
    0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b, /*a*/
    0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4, /*b*/
    0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f, /*c*/
    0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef, /*d*/
    0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61, /*e*/
    0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d  /*f*/
};
```

具体实现代码如下：

```
void AES::SubBytes(unsigned char state[][4])
{//字节替换
    int r,c;
    for(r=0; r<4; r++)
    {
        for(c=0; c<4; c++)
        {
            state[r][c] = Sbox[state[r][c]];
        }
    }
}
```

5. ShiftRows（行移位变换）

行移位变换完成基于行的循环位移操作，变换方法：



即行移位变换作用于行上，第 0 行不变，第 1 行循环左移 1 个字节，第 2 行循环左移 2 个字节，第 3 行循环左移 3 个字节。

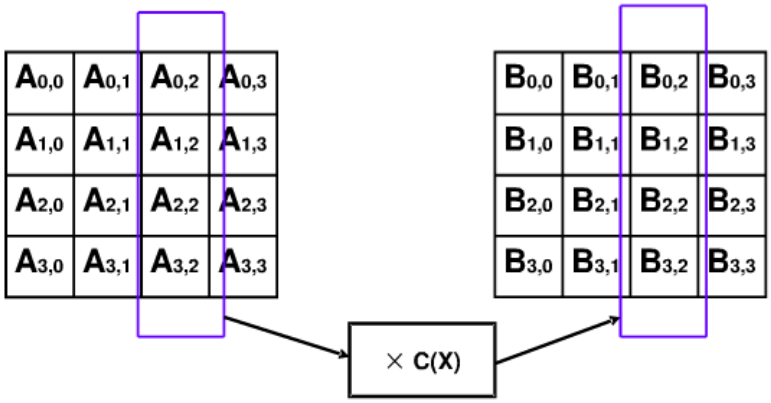
具体实现代码如下：

```
void AES::ShiftRows(unsigned char state[][4])
{//循环左移
    unsigned char t[4];
    int r,c;
    for(r=1; r<4; r++)
    {
        for(c=0; c<4; c++)
        {
            t[c] = state[r][(c+r)%4];
        }
        for(c=0; c<4; c++)
        {
            state[r][c] = t[c];
        }
    }
}
```

6.MixColumns（列混淆变换）

逐列混合，方法：

$b(x) = (03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02) \cdot a(x) \bmod (x^4 + 1)$



矩阵表示形式：

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

具体代码如下：

```
void AES::MixColumns(unsigned char state[][4])
{//列混合
    unsigned char t[4];
    int r,c;
    for(c=0; c< 4; c++)
    {
        for(r=0; r<4; r++)
        {
            t[r] = state[r][c];
        }
        for(r=0; r<4; r++)
        {
            state[r][c] = FFmul(0x02, t[r])
                        ^ FFmul(0x03, t[(r+1)%4])
                        ^ FFmul(0x01, t[(r+2)%4])
                        ^ FFmul(0x01, t[(r+3)%4]);
        }
    }
}
```

```
unsigned char AES::FFmul(unsigned char a, unsigned char b)
{
    unsigned char bw[4];
    unsigned char res=0;
    int i;
    bw[0] = b;
    for(i=1; i<4; i++)
    {
        bw[i] = bw[i-1]<<1;
        if(bw[i-1]&0x80)
        {
            bw[i]^=0x1b;
        }
    }
    for(i=0; i<4; i++)
    {
        if((a>>i)&0x01)
        {
            res ^= bw[i];
        }
    }
    return res;
}
```

其中 FFmul 为有限域 GF(2^8)上的乘法，标准算法应该是循环 8 次（b 与 a 的每一位相乘，结果相加），但这里只用到最低 2 位，解密时用到的逆列混淆也只用了低 4 位，所以在这里高 4 位的运算是多余的，只计算低 4 位。

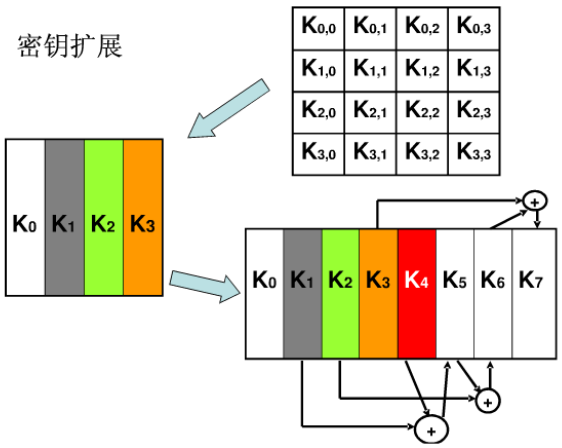
7. AddRoundKey（轮密钥加变换）

简单来说就是逐字节相加，有限域 GF(2^8)上的加法是模 2 加法，即异或

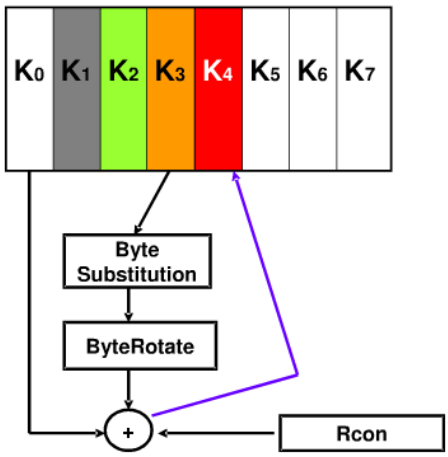
```
void AES::AddRoundKey(unsigned char state[][4], unsigned char k[][4])
{
    int r,c;
    for(c=0; c<4; c++)
    {
        for(r=0; r<4; r++)
        {
            state[r][c] ^= k[r][c];
        }
    }
}
```

8.KeyExpansion（密钥扩展）

将输入的密钥扩展为 11 组 128 位密钥组，其中第 0 组为输入密钥本身，其后第 n 组第 i 列 为 第 n-1 组第 i 列 与 第 n 组第 i-1 列之和（模 2 加法，1<= i <=3）



对于每一组 第一列即 i=0，有特殊的处理



将前一列即第 n-1 组第 3 列的 4 个字节循环左移 1 个字节，并对每个字节进行字节替代变换 SubBytes，将第一行（即第一个字节）与轮常量 rc[n]相加，最后再与前一组该列相加。

具体实现代码如下：

```
void AES::KeyExpansion(unsigned char* key, unsigned char w[][4][4])
{
    int i,j,r,c;
    unsigned char rc[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};
    for(r=0; r<4; r++)
    {
        for(c=0; c<4; c++)
            w[0][r][c] = key[r+c*4];
    }
    for(i=1; i<=10; i++)
    {
        for(j=0; j<4; j++)
        {
            unsigned char t[4];
            for(r=0; r<4; r++)
            {
                t[r] = j ? w[i][r][j-1] : w[i-1][r][3];
            }
            if(j == 0)
            {
                unsigned char temp = t[0];
                for(r=0; r<3; r++)
                {
                    t[r] = Sbox[t[(r+1)%4]];
                }
                t[3] = Sbox[temp];
                t[0] ^= rc[i-1];
            }
            for(r=0; r<4; r++)
            {
                w[i][r][j] = w[i-1][r][j] ^ t[r];
            }
        }
    }
}
```

AES 解密算法与加密类似，基本运算中除了 AddRoundKey（轮密钥加）不变外，其余的都需要进行逆变换，即 InvSubBytes（逆字节替代）、InvShiftRows（逆行移位）、InvMixColumns（逆列混淆）。

具体代码如下：

```
void AES::InvSubBytes(unsigned char state[][4])
{
    int r,c;
    for(r=0; r<4; r++)
    {
        for(c=0; c<4; c++)
        {
            state[r][c] = InvSbox[state[r][c]];
        }
    }
}

void AES::InvShiftRows(unsigned char state[][4])
{
    unsigned char t[4];
    int r,c;
    for(r=1; r<4; r++)
    {
        for(c=0; c<4; c++)
        {
            t[c] = state[r][(c-r+4)%4];
        }
        for(c=0; c<4; c++)
        {
            state[r][c] = t[c];
        }
    }
}

void AES::InvMixColumns(unsigned char state[][4])
{
    unsigned char t[4];
    int r,c;
    for(c=0; c< 4; c++)
    {
        for(r=0; r<4; r++)
        {
            t[r] = state[r][c];
        }
        for(r=0; r<4; r++)
        {
            state[r][c] = FFmul(0x0e, t[r])
                        ^ FFmul(0x0b, t[(r+1)%4])
                        ^ FFmul(0x0d, t[(r+2)%4])
                        ^ FFmul(0x09, t[(r+3)%4]);
        }
    }
}
```

[加密过程]

先将输入的明文按列序组合成 4*4 的矩阵，直接与第 0 组密钥（即输入的密钥）相加（异或），作为轮加密的输入

然后循环 10 次进行 SubBytes、ShiftRows、MixColumns、AddRoundKey 运算，最后恢复原序列

最后一轮并不进行 MixColumns（列混淆变换）

```
unsigned char* AES::Cipher(unsigned char* input)
{
    unsigned char state[4][4];
    int i,r,c;

    for(r=0; r<4; r++)
    {
        for(c=0; c<4 ;c++)
        {
            state[r][c] = input[c*4+r];//按列优先输入
        }
    }

    AddRoundKey(state,w[0]);

    for(i=1; i<=10; i++)
    {
        SubBytes(state);
        ShiftRows(state);
        if(i!=10)MixColumns(state);//最后一轮不用列混合
        AddRoundKey(state,w[i]);
    }

    for(r=0; r<4; r++)
    {
        for(c=0; c<4 ;c++)
        {
            input[c*4+r] = state[r][c];
        }
    }

    return input;
}
```

[解密过程]

```
unsigned char* AES::InvCipher(unsigned char* input)
{
    unsigned char state[4][4];
    int i,r,c;

    for(r=0; r<4; r++)
    {
        for(c=0; c<4 ;c++)
        {
            state[r][c] = input[c*4+r];
        }
    }

    AddRoundKey(state, w[10]);
    for(i=9; i>=0; i--)
    {
        InvShiftRows(state);
        InvSubBytes(state);
        AddRoundKey(state, w[i]);
        if(i)
        {
            InvMixColumns(state);
        }
    }

    for(r=0; r<4; r++)
    {
        for(c=0; c<4 ;c++)
        {
            input[c*4+r] = state[r][c];
        }
    }

    return input;
}
```

[输入输出测试]

输入十六进制明文：1122334455667788990AABBCCDDEEFF

输入十六进制密钥：13579BDF02468ACE1234567890ABCDEF

```
明文: 11 22 33 44 55 66 77 88 99 00 AA BB CC DD EE FF
密文: E8 98 46 59 6E D1 6C 17 C8 99 20 CF 26 51 C0 BD
解密: 11 22 33 44 55 66 77 88 99 00 AA BB CC DD EE FF
```

一、RC4 加解密算法

［RC4 算法简介］

RC4 于 1987 年提出，和 DES 算法一样，是一种对称加密算法，也就是说使用的密钥为私钥。但不同于 DES 的是，RC4 不是对明文进行分组处理，而是字节流的方式依次加密明文中的每一个字节，解密的时候也是依次对密文中的每一个字节进行解密。

RC4 算法的特点是算法简单，运行速度快，而且密钥长度是可变的，可变范围为 1-256 字节(8-2048 比特)，在如今技术支持的前提下，当密钥长度为 128 比特时，用暴力法搜索密钥已经不太可行，所以可以预见 RC4 的密钥范围任然可以在今后相当长的时间里抵御暴力搜索密钥的攻击。实际上，如今也没有找到对于 128bit 密钥长度的 RC4 加密算法的有效攻击方法。

算法中的几个关键变量：

1、密钥流：RC4 算法的关键是根据明文和密钥生成相应的密钥流，密钥流的长度和明文的长度是对应的，也就是说明文的长度是 500 字节，那么密钥流也是 500 字节。当然，加密生成的密文也是 500 字节，因为密文第 i 字节=明文第 i 字节^密钥流第 i 字节；

2、状态向量 S：长度为 256，S[0],S[1].....S[255]。每个单元都是一个字节，算法运行的任何时候，S 都包括 0-255 的 8 比特数的排列组合，只不过值的位置发生了变换；

3、临时向量 T：长度也为 256，每个单元也是一个字节。如果密钥的长度是 256 字节，就直接把密钥的值赋给 T，否则，轮转地将密钥的每个字节赋给 T；

4、密钥 K：长度为 1-256 字节，注意密钥的长度 keylen 与明文长度、密钥流的长度没有必然关系，通常密钥的长度趣味 16 字节（128 比特）。

[RC4 算法流程]

S 盒初始化

密钥流生成

使用密钥流加密

[S 盒初始化]

- $S_0=0, S_1=1, \ldots, S_{255}=255$
- 然后用初始密钥key填充另一个256字节的数组K，不断重复密钥直至填充整个数组

  $j=0$

  对于i=0至255

  $j=(j+S_i+K_i) \bmod 256$

  交换S_i和S_j
- 最后得到用密钥初始化的S盒

具体代码如下：

```
int[] S = new int[256];
int[] K = new int[256];

//S盒初始化，0～255填充
for (int i=0;i<256;i++)
   S[i]=i;

for (int i= 0;i<256;i++)
{
   //用初始密钥key填充另一个256字节的数组Key,不断重复直到填充完
   K[i]=aKey[i % aKey.length];
}


//初始化S盒
int  j=0;
for (int i=0;i<255;i++)
{
   j=(j+S[i]+K[i]) % 256;
   int temp = S[i]; //交换Si和Sj
   S[i]=S[j];
   S[j]=temp;
}
```

[密钥流生成]

$$i=j=0$$

$$i=(i+1) \bmod 256$$

$$j=(j+S_i) \bmod 256$$

交换S_i和S_j

$$t=(S_i+S_j) \bmod 256$$

$$k=S_t$$

具体代码如下：

```
int i=0;
j=0;
int[] iInputChar = aInput;
int[] iOutputChar = new int[iInputChar.length];
for(int x = 0;x<iInputChar.length;x++)
{
    i = (i+1) % 256;
    j = (j+S[i]) % 256;

    int temp = S[i]; //交换Si和Sj
    S[i]=S[j];
    S[j]=temp;

    int t = (S[i]+S[j] ) % 256;
    int iY = S[t];
```

[使用密钥流加密]

```
iOutputChar[x] =(byte) (iInputChar[x] ^ iY);
```

本质上为求异或

[输入与输出测试]

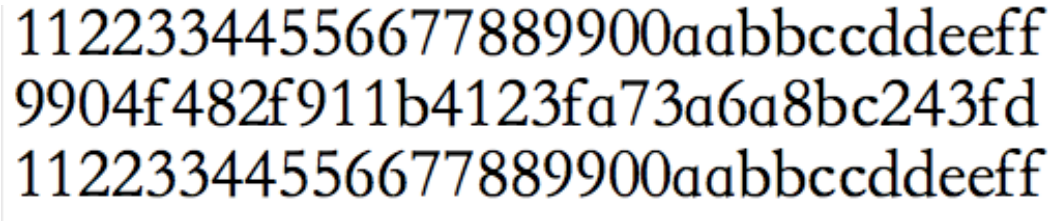
输入十六进制明文：11223344556677889900AABBCCDDEEFF

输入十六进制密钥：13579BDF02468ACE1234567890ABCDEF

```
int[] Text = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
        0x00, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
String aInput =BytesToString(Text);
int[] keys = {0x13, 0x57, 0x9B, 0xDF, 0x02, 0x46, 0x8A, 0xCE, 0x12,
        0x34, 0x56,0x78, 0x90, 0xAB, 0xCD, 0xEF};
String aKey = keys.toString();

int[] encrypt = myRC4(Text, keys);
int[] decrypt = myRC4(encrypt, keys);
```

三行从上到下依次为加密之前，加密之后，解密之后的结果



[问题讨论]

1. 改变明文或密钥中的一个比特值可能影响 AES 值中的多少比特?

   明文和密钥在 S 盒置换时，不同的字节会替换出不同的结果。算法过程中一共进行了 10 轮加密,所以改变一个比特值可能影响 AES 值中的 80 比特(8 x 10)。

2. 在 RC4 的密钥流生成中，改变初始密钥的一个比特值可能影响输出中的多少比特?

   初始密钥的不同会导致 S 盒的不同，所以可能影响输出中的 256 比特。

3. 分析实验中在编辑、编译、运行等各环节中所出现的问题及解决方法。

   首先要求输入的是 16 进制，以前没遇到过这类问题，在这里查了很多解决方法；其次是 AES 算法中明文和密钥编排都是列优先，我开始的时候弄成了行优先导致全部出错。