AES算法

1120152035 王奥博

算法简介:

和DES相比,AES也是对称分组密码算法.加密时数据的分组长度必须为128位(16个字节).

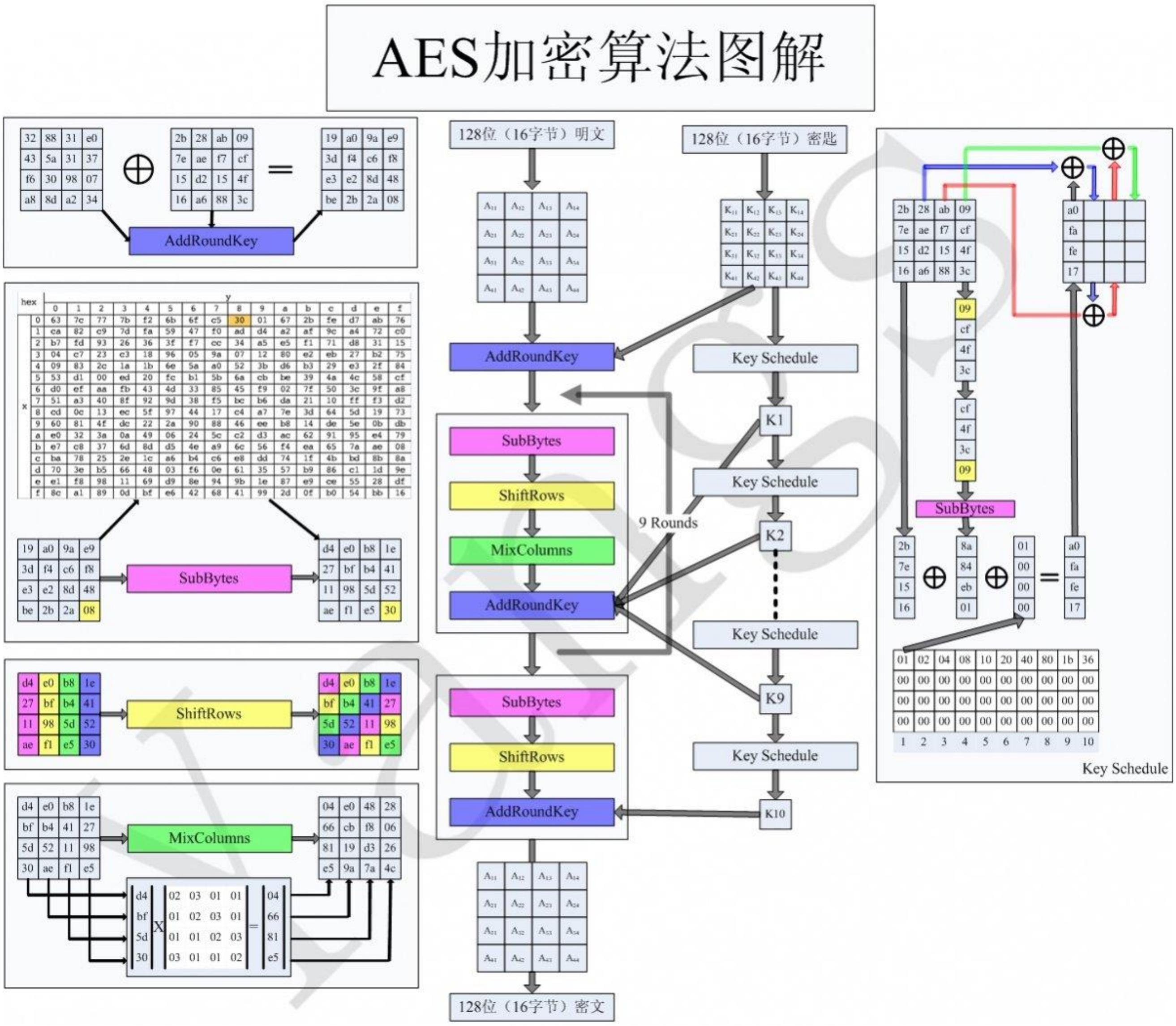
当使用128, 192, 256位三种不同长度的秘钥时,加密时的AES算法可称为"AES-128"," AES-192","AES-256".

本次试验中,以AES-128为例展开.

AES加密过程:

AES的加密经过了4种操作:

- 字节替换
- 行位移
- 列混淆
- 轮秘钥加



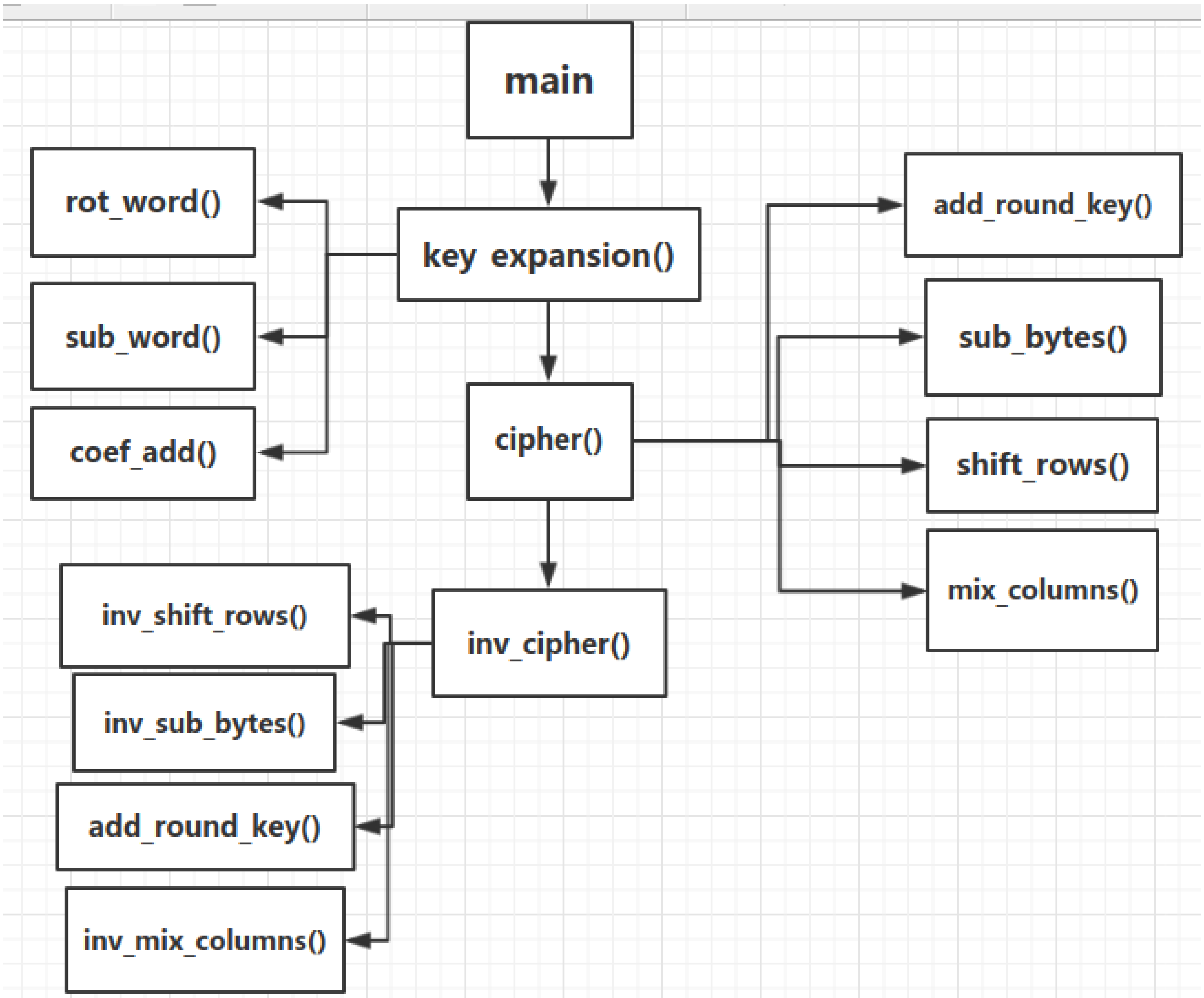
AES解密

对于对称加密AES,解密过程可以看为加密的逆操作.按加密的相反步骤即可恢复明文.每轮的秘钥由初始秘钥扩展得到.

在加解密中,16字节的明文,密文和轮秘钥都已4*4的矩阵表示.

程序流程:

> 该程序函数较多,下图仅显示了部分关键函数



函数解释:

> 同样,仅解释部分主要函数

函数定义:**void key_expansion(int *key, int *w)**

函数功能:通过原始秘钥生成多重秘钥,增加比特位的扩散度

```
void key_expansion(int *key, int *w) {

    int tmp[4];
    int len = Nb*(Nr+1); // 4*(14+1) = 60, Nk=8,
    // 将种子密钥的值拷贝到密钥调度字节表  w[], 因为 w[] 表总是 4 列，假如种子密钥
    // 是 192 位（24字节），在这种情况下 KeyExapansion 将种子密钥拷贝到 w[] 的前面 6 行
    for (int i = 0; i < Nk; i++) {
        w[4*i+0] = key[4*i+0];
        w[4*i+1] = key[4*i+1];
        w[4*i+2] = key[4*i+2];
        w[4*i+3] = key[4*i+3];
    }

    // 计算剩余w[]的值
    for (int i = Nk; i < len; i++) {
        // w[j-4]
        tmp[0] = w[4*(i-1)+0];
        tmp[1] = w[4*(i-1)+1];
        tmp[2] = w[4*(i-1)+2];
        tmp[3] = w[4*(i-1)+3];

        if (i%Nk == 0) {
            //将tmp循环左移一个字节
            rot_word(tmp);
            // 分别对每个字节按 S盒进行映射
            sub_word(tmp);
            // 与32 bits的常量（RC[i/Nk],0,0,0）进行异或
            coef_add(tmp, Rcon(i/Nk), tmp);

        } else if (Nk > 6 && i%Nk == 4)
            sub_word(tmp);


        w[4*i+0] = w[4*(i-Nk)+0]^tmp[0];
        w[4*i+1] = w[4*(i-Nk)+1]^tmp[1];
        w[4*i+2] = w[4*(i-Nk)+2]^tmp[2];
        w[4*i+3] = w[4*(i-Nk)+3]^tmp[3];
    }
}
```

函数定义:**void cipher(int *in, int *out, int *w)**

函数功能:实现AES加密,函数中实现了AES加密需要的四个操作

```c
void cipher(int *in, int *out, int *w) {

    int state[4*Nb];
    // 将输入数组拷贝到 state(4*4矩阵)
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            state[Nb*i+j] = in[i+4*j];
        }
    }

    add_round_key(state, w, 0);
    // Nr=14
    for (int r = 1; r < Nr; r++) {
        // 将State矩阵中的每个字节替换成一个由 Sbox决定的新字节
        sub_bytes(state);
        // 将State矩阵中的字节向左旋转
        shift_rows(state);
        // 将State字节列的值进行数学 "域加"和"域乘"的结果代替每个字节
        mix_columns(state);
        // 轮密钥加
        add_round_key(state, w, r);
    }

    sub_bytes(state);
    shift_rows(state);
    add_round_key(state, w, Nr);

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            out[i+4*j] = state[Nb*i+j];
        }
    }
}
```

函数定义:**void inv_cipher(int *in, int *out, int *w)**

函数功能:和cipher函数作用相反,实现AES的解密:

```c
void inv_cipher(int *in, int *out, int *w) {

    int state[4*Nb];

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            state[Nb*i+j] = in[i+4*j];
        }
    }

    add_round_key(state, w, Nr);

    for (int r = Nr-1; r >= 1; r--) {
        inv_shift_rows(state);
        inv_sub_bytes(state);
        add_round_key(state, w, r);
        inv_mix_columns(state);
    }

    inv_shift_rows(state);
    inv_sub_bytes(state);
    add_round_key(state, w, 0);

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            out[i+4*j] = state[Nb*i+j];
        }
    }
}
```

完整代码:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
 /*
    实现功能： 将a的值乘以b
    如果该值的最高位为1（表示该数值不小于128），则还需要将移位后的结果异或 00011011(0x1b)
    注意：a和b都是传值调用，返回结果保存在 p中（也即a左移1位的结果）
    @param:
        a=R[0],
        b=0x02(乘以2，也即将该值的二进制位左移 1位)
 */
int gmult(int a, int b) {

    int p = 0, i = 0, hbs = 0;

    for (i = 0; i < 8; i++) {
        if (b & 1) {
            p ^= a;
        }

        hbs = a & 0x80;
        a <<= 1;
        if (hbs) a ^= 0x1b; // 0000 0001 0001 1011
        b >>= 1;
    }

    return (int)p;
}

// 数组异或
void coef_add(int a[], int b[], int d[]) {

    d[0] = a[0]^b[0];
    d[1] = a[1]^b[1];
    d[2] = a[2]^b[2];
    d[3] = a[3]^b[3];
}

/*
coef_mult函数执行的操作就是：将矩阵 T和b的每一元素分别执行 gmult，在把结果进行异或运算
运算结果存放在 d数组里面
*/
void coef_mult(int *a, int *b, int *d) {

    d[0] = gmult(a[0],b[0])^gmult(a[3],b[1])^gmult(a[2],b[2])^gmult(a[1],b[3]);
    d[1] = gmult(a[1],b[0])^gmult(a[0],b[1])^gmult(a[3],b[2])^gmult(a[2],b[3]);
    d[2] = gmult(a[2],b[0])^gmult(a[1],b[1])^gmult(a[0],b[2])^gmult(a[3],b[3]);
    d[3] = gmult(a[3],b[0])^gmult(a[2],b[1])^gmult(a[1],b[2])^gmult(a[0],b[3]);
}

/*
 * Number of columns (32-bit words) comprising the State. For this

 * standard, Nb = 4.
```

```
 */
int Nb = 4;

/*
 * Number of 32-bit words comprising the Cipher Key. For this
 * standard, Nk = 4, 6, or 8.
 */
int Nk;

/*
 * Number of rounds, which is a function of  Nk  and  Nb (which is
 * fixed). For this standard, Nr = 10, 12, or 14.
 */
// Nr 为给定密钥大小的轮数减 1。加密算法使用的轮数要么是 10，12，要
// 么是14，这依赖于种子密钥长度是 128位、192位还是256位
int Nr;

/*
 * S-box transformation table
 */
static int s_box[256] = {
     // 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

/*
 * Inverse S-box transformation table
 */
static int inv_s_box[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


/*
 * Generates the round constant Rcon[i] // 轮常数表
 */
/* 每个轮常数的最左边的字节是 GF(28)域中2的幂次方。它的另一个表示方法是其每个
    值是前一个值乘上0x02。注意 0x80 × 0x02 = 0x1b 是 0x80 左移1个比特位
    后紧接着与 0x1b 进行异或。
*/
int R[] = {0x02, 0x00, 0x00, 0x00};
```

```
/*
 RC = {00, 01, 02, 04, 08, 10, 20, 40, 80, 1B, 36}
 这里通过Rcon函数计算，所得的值和直接索引 RC表一样。
*/
int * Rcon(int i) {

    if (i == 1) {
        R[0] = 0x01; // x^(1-1) = x^0 = 1
    } else if (i > 1) {
        R[0] = 0x02;
        i--;
        while (i-1 > 0) {
            R[0] = gmult(R[0], 0x02);
            i--;
        }
    }

    return R;
}

 /*
 轮密钥加
 @param
    state:需要处理的状态数据（ 4*4矩阵）
    w:轮密钥
    r:使用第r轮密钥（ 4*4矩阵为一轮）
 */
void add_round_key(int *state, int *w, int r) {
    for (int c = 0; c < Nb; c++) {
        // 按列循环，计算第 c列的值
        // state[row,col] = state[row,col]^w[col,row] （r=0）
        state[Nb*0+c] = state[Nb*0+c]^w[4*Nb*r+4*c+0];   //debug, so it works for Nb !=4
        state[Nb*1+c] = state[Nb*1+c]^w[4*Nb*r+4*c+1];
        state[Nb*2+c] = state[Nb*2+c]^w[4*Nb*r+4*c+2];
        state[Nb*3+c] = state[Nb*3+c]^w[4*Nb*r+4*c+3];
    }
}

/*
将State字节列的值进行数学 "域加"和"域乘"的结果代替每个字节
假设State[0,1]的值是0x09，并且列1上的其它值分别为 0x60，
0xe1和0x04，那么State[0,1]的新值计算如下：
State[0,1]=(State[0,1]*0x01)+(State[1,1]*0x02)+(State[2,1]*0x03)+(State[3,1]*0x01)
            =(0x09*0x01)+(0x60*0x02)+(0xe1*0x03)+(0x04*0x01)
            =0x57
此处加法和乘法是专门的数学域操作，而不是平常整数的加法和乘法
数学域操作 查阅相关资料
*/
void mix_columns(int *state) {

    int a[] = {0x02, 0x01, 0x01, 0x03}; // a(x) = {02} + {01}x + {01}x2 + {03}x3
    int i, j, col[4], res[4];

```

```
    for (j = 0; j < Nb; j++) {
        // 按列取出state数据
        for (i = 0; i < 4; i++) {
            col[i] = state[Nb*i+j];
        }
        // 对state的每一列，执行“域运算”
        coef_mult(a, col, res);
        // 将“域运算”的结果存放回state
        for (i = 0; i < 4; i++) {
            state[Nb*i+j] = res[i];
        }
    }
}

void inv_mix_columns(int *state) {

    int a[] = {0x0e, 0x09, 0x0d, 0x0b}; // a(x) = {0e} + {09}x + {0d}x2 + {0b}x3
    int i, j, col[4], res[4];

    for (j = 0; j < Nb; j++) {
        for (i = 0; i < 4; i++) {
            col[i] = state[Nb*i+j];
        }

        coef_mult(a, col, res);

        for (i = 0; i < 4; i++) {
            state[Nb*i+j] = res[i];
        }
    }
}

 /*
 ShiftRows是一个置换操作，它将State矩阵中的字节向左旋转
 State的第0行被向左旋转0个位置，
 State的第1行被向左旋转1个位置，
 State的第2行被向左旋转2个位置，
 State的第3行被向左旋转3个位置
 */
void shift_rows(int *state) {
    for (int i = 1; i < 4; i++) {
        int s = 0;
        while (s < i) {
            int tmp = state[Nb*i+0];

            for (int k = 1; k < Nb; k++) {
                state[Nb*i+k-1] = state[Nb*i+k];
            }

            state[Nb*i+Nb-1] = tmp;
            s++;
        }

    }
```

```
}

void inv_shift_rows(int *state) {

    for (int i = 1; i < 4; i++) {
        int s = 0;
        while (s < i) {
            int tmp = state[Nb*i+Nb-1];

            for (int k = Nb-1; k > 0; k--) {
                state[Nb*i+k] = state[Nb*i+k-1];
            }

            state[Nb*i+0] = tmp;
            s++;
        }
    }
}

 /*
 将State矩阵中的每个字节替换成一个由 Sbox决定的新字节
 比如，State[0,1]的值是0x40如果你想找到它的代替者，你取 State[0,1]的值(0x40)并
 让x等于左边的数字(4)并让y等于右边的数字(0)。然后你用x和y作为索引进
 到Sbox表中寻找代替值，Sbox[x,y]
 */
void sub_bytes(int *state) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            int row = (state[Nb*i+j] & 0xf0) >> 4;
            int col = state[Nb*i+j] & 0x0f;
            state[Nb*i+j] = s_box[16*row+col];
        }
    }
}

void inv_sub_bytes(int *state) {

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            int row = (state[Nb*i+j] & 0xf0) >> 4;
            int col = state[Nb*i+j] & 0x0f;
            state[Nb*i+j] = inv_s_box[16*row+col];
        }
    }
}

/*
    使用替换表 Sbox 对一个给定的一行密钥调度表 w[] 进行逐字节替换
    比如0x27的代替结果是 x = 2 和 y = 7，并且 Sbox[2,7] 返回 0xcc
*/
void sub_word(int *w) {
    for (int i = 0; i < 4; i++) {

        w[i] = s_box[16*((w[i] & 0xf0) >> 4) + (w[i] & 0x0f)];
```

```
    }
}

// 接受一个4个字节的数组并将它们向左旋转一个位置
void rot_word(int *w) {
    int tmp = w[0];

    for (int i = 0; i < 3; i++) {
        w[i] = w[i+1];
    }

    w[3] = tmp;
}

/*
AES加密和解密算法使用了一个由种子密钥字节数组生成的密钥调度表。 AES规范
中称之为密钥扩展例程（KeyExpansion）。从本质上讲，从一个原始密钥中生成多
重密钥以代替使用单个密钥大大增加了比特位的扩散 .
*/
void key_expansion(int *key, int *w) {

    int tmp[4];
    int len = Nb*(Nr+1); // 4*(14+1) = 60, Nk=8,
    // 将种子密钥的值拷贝到密钥调度字节表 w[], 因为 w[] 表总是 4 列，假如种子密钥
    // 是 192 位（24字节），在这种情况下 KeyExapansion 将种子密钥拷贝到 w[] 的前面 6 行
    for (int i = 0; i < Nk; i++) {
        w[4*i+0] = key[4*i+0];
        w[4*i+1] = key[4*i+1];
        w[4*i+2] = key[4*i+2];
        w[4*i+3] = key[4*i+3];
    }

    // 计算剩余w[]的值
    for (int i = Nk; i < len; i++) {
        // w[j-4]
        tmp[0] = w[4*(i-1)+0];
        tmp[1] = w[4*(i-1)+1];
        tmp[2] = w[4*(i-1)+2];
        tmp[3] = w[4*(i-1)+3];

        if (i%Nk == 0) {
            //将tmp循环左移一个字节
            rot_word(tmp);
            // 分别对每个字节按 S盒进行映射
            sub_word(tmp);
            // 与32 bits的常量（RC[i/Nk],0,0,0）进行异或
            coef_add(tmp, Rcon(i/Nk), tmp);

        } else if (Nk > 6 && i%Nk == 4)
            sub_word(tmp);



        w[4*i+0] = w[4*(i-Nk)+0]^tmp[0];
```

```
        w[4*i+1] = w[4*(i-Nk)+1]^tmp[1];
        w[4*i+2] = w[4*(i-Nk)+2]^tmp[2];
        w[4*i+3] = w[4*(i-Nk)+3]^tmp[3];
    }
}

void cipher(int *in, int *out, int *w) {

    int state[4*Nb];
    // 将输入数组拷贝到state(4*4矩阵)
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            state[Nb*i+j] = in[i+4*j];
        }
    }

    add_round_key(state, w, 0);
    // Nr=14
    for (int r = 1; r < Nr; r++) {
        // 将State矩阵中的每个字节替换成一个由 Sbox决定的新字节
        sub_bytes(state);
        // 将State矩阵中的字节向左旋转
        shift_rows(state);
        // 将State字节列的值进行数学 "域加"和"域乘"的结果代替每个字节
        mix_columns(state);
        // 轮密钥加
        add_round_key(state, w, r);
    }

    sub_bytes(state);
    shift_rows(state);
    add_round_key(state, w, Nr);

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            out[i+4*j] = state[Nb*i+j];
        }
    }
}

void inv_cipher(int *in, int *out, int *w) {

    int state[4*Nb];

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            state[Nb*i+j] = in[i+4*j];
        }
    }

    add_round_key(state, w, Nr);


    for (int r = Nr-1; r >= 1; r--) {
```

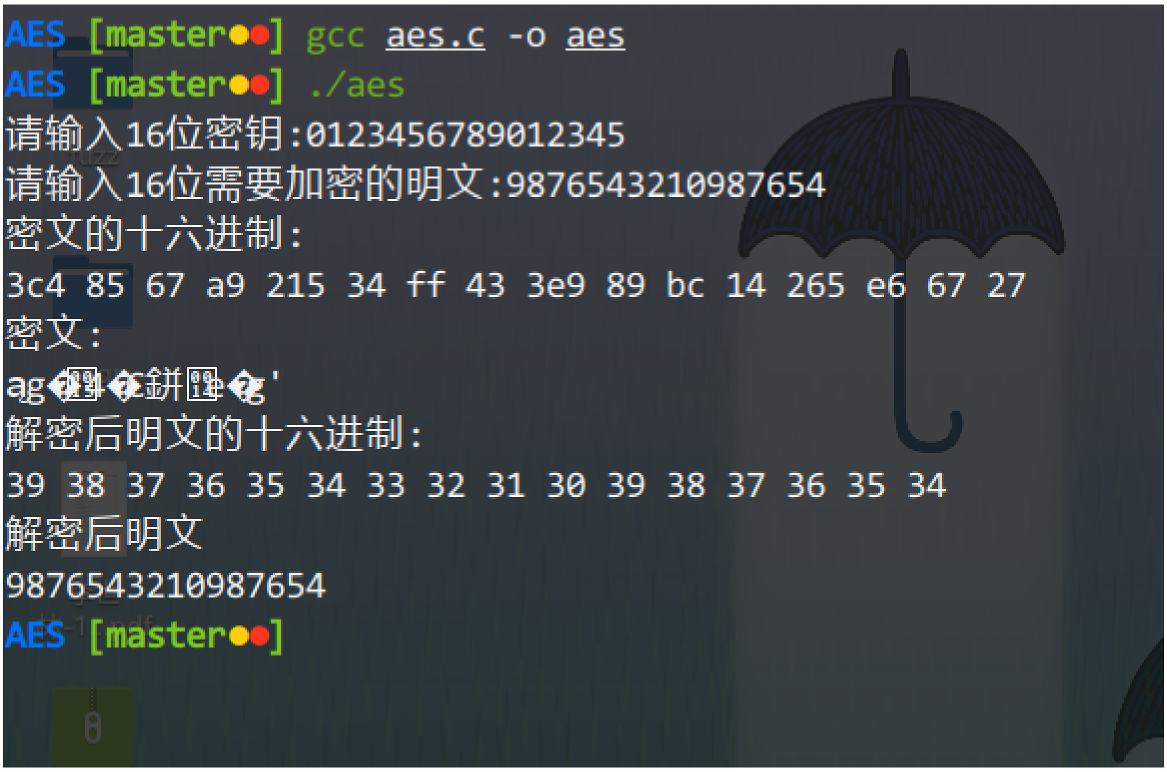
```
        inv_shift_rows(state);
        inv_sub_bytes(state);
        add_round_key(state, w, r);
        inv_mix_columns(state);
    }

    inv_shift_rows(state);
    inv_sub_bytes(state);
    add_round_key(state, w, 0);

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < Nb; j++) {
            out[i+4*j] = state[Nb*i+j];
        }
    }
}

int main() {

    int key[16];
    int in[16];
    char inkey[32];
    char inming[16];
    printf("请输入16位密钥:");
    scanf("%s",inkey);
    printf("请输入16位需要加密的明文:");
    scanf("%s",inming);
    for(int k=0;k<16;k++)
        key[k]=(int)inkey[k];
    for(int k=0;k<16;k++)
        in[k]=(int)inming[k];
    int out[16]; // 128 bits

    switch (sizeof(key)) {
        default:
        case 16: Nk = 4; Nr = 10; break;
        case 24: Nk = 6; Nr = 12; break;
        case 32: Nk = 8; Nr = 14; break; // Nk=8, Nr=14
    }

    int *w = malloc(Nb*(Nr+1)*4);//秘钥扩展
    // 密钥扩展
    key_expansion(key, w);
    // 加密
    cipher(in, out, w);
    printf("密文的十六进制:\n");
    for (int i = 0; i < 4; i++) {
        printf("%x %x %x %x ", out[4*i+0], out[4*i+1], out[4*i+2], out[4*i+3]);
    }
    printf("\n");
    printf("密文:\n");
        for (int i = 0; i < 4; i++) {

        printf("%c%c%c%c", out[4*i+0], out[4*i+1], out[4*i+2], out[4*i+3]);
```

```
    }
    printf("\n");
    // 解密
    inv_cipher(out, in, w);
    printf("解密后明文的十六进制:\n");
    for (int i = 0; i < 4; i++) {
        printf("%x %x %x %x ", in[4*i+0], in[4*i+1], in[4*i+2], in[4*i+3]);
    }
     printf("\n");
    printf("解密后明文\n");
        for (int i = 0; i < 4; i++) {
        printf("%c%c%c%c", in[4*i+0], in[4*i+1], in[4*i+2], in[4*i+3]);
    }

    printf("\n");
    exit(0);

}
```

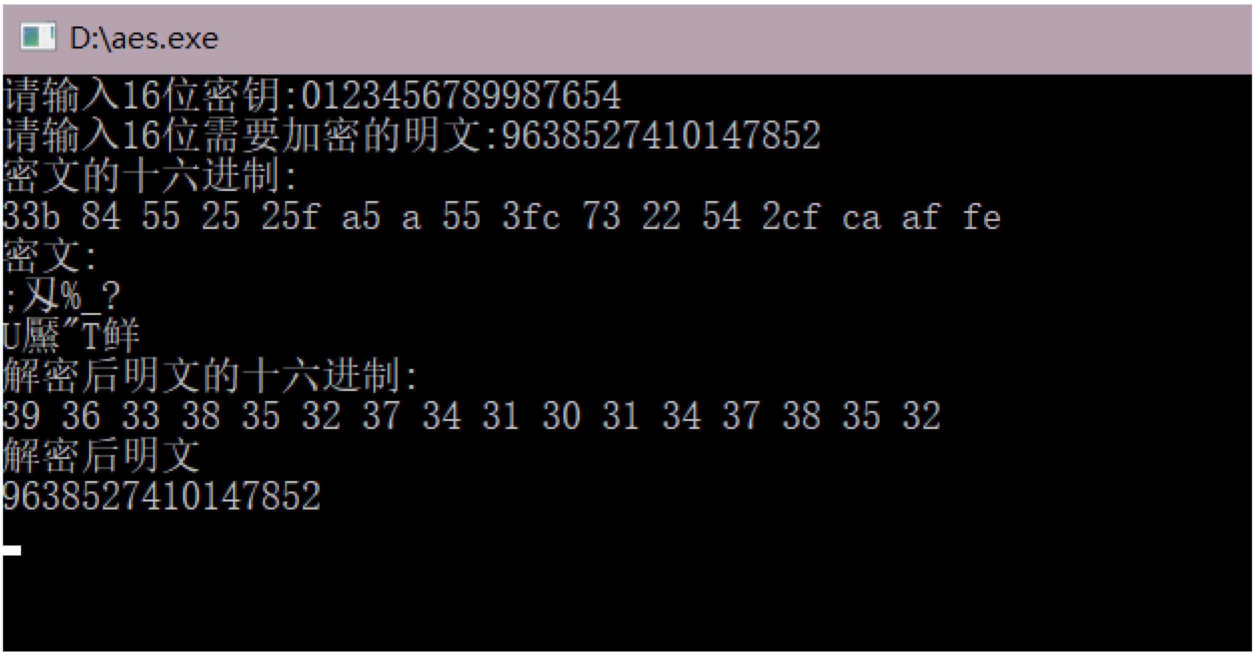
测试:

完成了在linux和windows两个平台上的测试:

Linux平台:



Windows平台:


D:\aes.exe
请输入16位密钥:0123456789987654
请输入16位需要加密的明文:9638527410147852
密文的十六进制:
33b 84 55 25 25f a5 a 55 3fc 73 22 54 2cf ca af fe
密文:
解密后明文的十六进制:
39 36 33 38 35 32 37 34 31 30 31 34 37 38 35 32
解密后明文
9638527410147852