浙江大学

本科实验报告

课程名称：	网络与通信安全
姓　　名：	袁东琦
学　　院：	信息与电子工程学院
系：	
专　　业：	信息工程
学　　号：	3200103602
指导教师：	陈惠芳/谢磊
实验名称：	AES 与基于 AES 的 CMAC

2023 年 6 月 9 日

专业：___信息工程_____
姓名：___袁东琦_____
学号：___3200103602___
日期：____2023/6/9_____

浙江大学实验报告

课程名称：_____网络与通信安全____指导老师：_____陈惠芳/谢磊_____成绩：_____
实验名称：_____AES 与基于 AES 的 CMAC_____

一、实验要求

复习或自学相应的理论知识与工作原理，用 MATLAB 或 C 实现加解密算法及其应用：AES 和基于 AES 的 CMAC。

二、实验原理

1、AES 原理

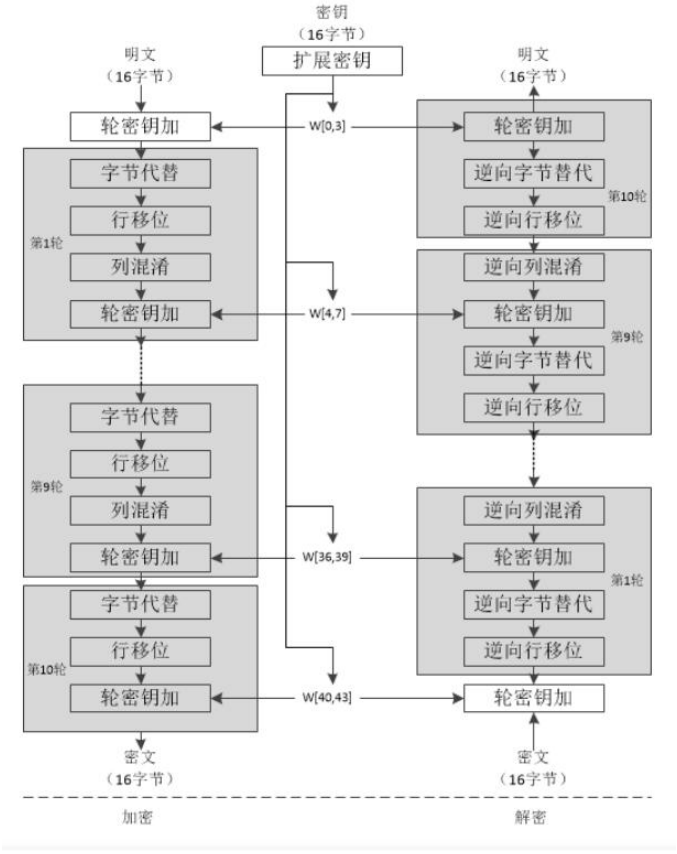

图 1　AES 框图

加密的第 1 轮到第 9 轮的轮函数一样，包括 4 个操作：字节代换、行位移、列混淆和轮密钥加。最后一轮迭代不执行列混合。

1）字节代换

AES 的字节代换通过查表实现，主要功能是通过 S 盒完成一个字节到另外一个字节的映射。AES 定义了一个 S 盒和一个逆 S 盒。矩阵每一个元素的第一个数字或字母对应 S 盒或逆 S 盒的纵轴，第二个数字或字母对应横轴，两轴交叉的点对应的十六进制数就是该字节代换的结果。

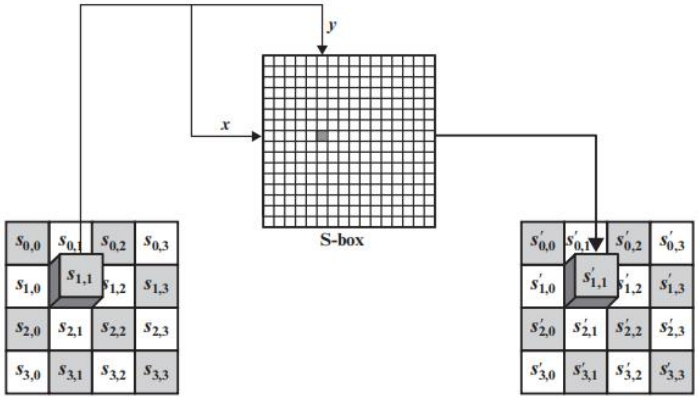

图 2　字节代换

2）行移位

第一行不移位，第二行左移一位，第三行左移两位，第四行左移三位。行移位的逆变换就是第一行不移位，第二行右移一位，第三行右移两位，第四行右移三位。

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

图 3　行移位

3）列混淆

列混淆变换是通过矩阵相乘来实现的，经行移位后的状态矩阵与固定的矩阵相乘，得到混淆后的状态矩阵。

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

图 4　列混淆

列混淆的逆变换同样通过矩阵相乘实现。

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

图 5　列混淆逆变换

4）轮密钥加

轮密钥加即当前的状态矩阵与 128 位的轮密钥进行按位异或。

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

⊕

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

图 6　轮密钥加

5）密钥扩展

AES 密钥扩展算法的输入值是 4 个字（16 字节），输出值是由 44 个字组成的一维线性数组。输入密钥直接被复制到扩展密钥数组的前 4 个字。然后每次用 5 个字填充扩展密钥数组的余下的部分。在扩展密钥数组中，每个新增的字 w[i]的值都依赖于前一个字 w[i-1]和前四个字 w[i-4]。在 4 种情形下，三种使用了异或运算。对 w 数组中下标为 4 的倍数的元素采用了更复杂的函数来计算。

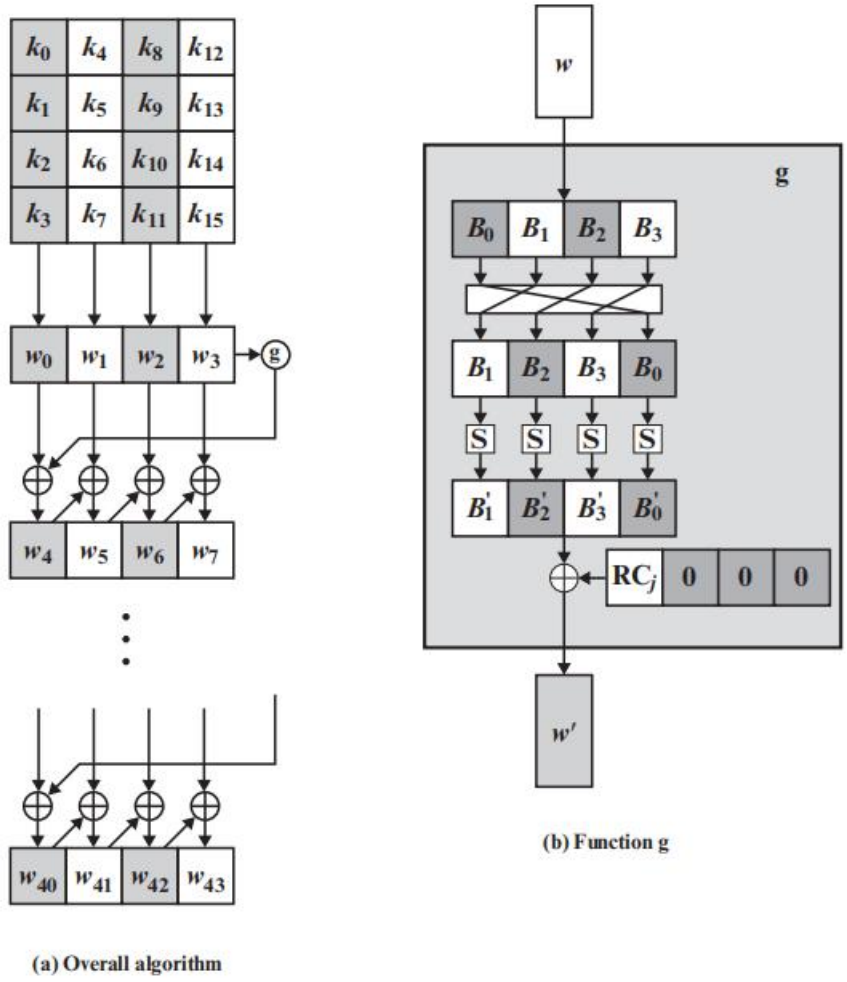

图 7　密钥扩展

（1）字循环的功能是使一个字中的 4 字节循环左移 1 字节。

（2）字替换利用 S 盒对输入字中的每个字节进行字节替换。

（3）第一步和第二步的结果再与轮常量 Rcon[j]进行异或运算。

2、CMAC 原理

采用 AES 加密算法，使用密钥 K，对明文 P 进行加密，得到的密文 C，作为明文 P 的

认证码，和明文 P 一起传输给接收方。接收方收到后，再使用自己的密钥，对明文再做一次 AES 加密，生成新的认证码，与接收到的发送方的认证码进行对比验证。如果相等，说明明文没有被篡改，接收方就可以接收明文并处理；如果不相等，说明明文被篡改，数据不安全，则丢弃。

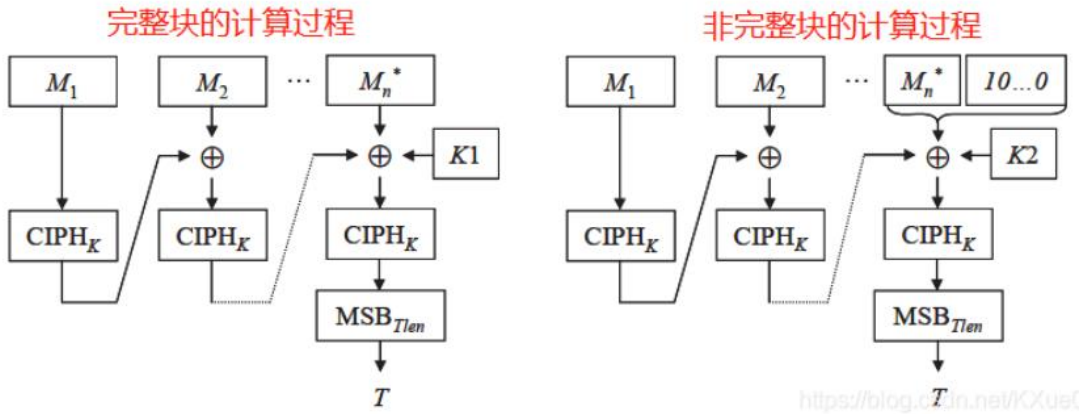

图 8 CMAC

三、实验过程与分析

1、对姓名进行 unicode 编码，得到的结果为 8881，4e1c，7426。因此选择 0x88，0x81，0x4e，0x1c，0x74，0x26 作为密钥，因为密钥需要 16 位，因此我选择重复填充的方式，最终生成的密钥如下图所示。

```
0x88,0x74,0x4e,0x88,
0x81,0x26,0x1c,0x81,
0x4e,0x88,0x74,0x4e,
0x1c,0x81,0x26,0x1c
```

图 9 密钥

2、对需要加密的消息进行十六进制编码，每一位字母对应一个十六进制数，最终形成 340 个十六进制数，由于 AES 加密算法需要加密信息为 16 的倍数，因此我在最后添加了 12 个 0x00，最终形成的需要加密的信息如下图所示。

```
0x49, 0x6e, 0x66, 0x6f, 0x72, 0x6d, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x53, 0x65, 0x63, 0x75, 0x72, 0x69,
0x74, 0x79, 0x69, 0x73, 0x61, 0x6d, 0x75, 0x6c, 0x74, 0x69, 0x64, 0x69, 0x73, 0x63, 0x69, 0x70, 0x6c,
0x69, 0x6e, 0x61, 0x72, 0x79, 0x61, 0x72, 0x65, 0x61, 0x6f, 0x66, 0x73, 0x74, 0x75, 0x64, 0x79, 0x61,
0x6e, 0x64, 0x70, 0x72, 0x6f, 0x66, 0x65, 0x73, 0x73, 0x69, 0x6f, 0x6e, 0x61, 0x6c, 0x61, 0x63, 0x74,
0x69, 0x76, 0x69, 0x74, 0x79, 0x77, 0x68, 0x69, 0x63, 0x68, 0x69, 0x73, 0x63, 0x6f, 0x6e, 0x63, 0x65,
0x72, 0x6e, 0x65, 0x64, 0x77, 0x69, 0x74, 0x68, 0x74, 0x68, 0x65, 0x64, 0x65, 0x76, 0x65, 0x6c, 0x6f,
0x70, 0x6d, 0x65, 0x6e, 0x74, 0x61, 0x6e, 0x64, 0x69, 0x6d, 0x70, 0x6c, 0x65, 0x6d, 0x65, 0x6e, 0x74,
0x61, 0x74, 0x69, 0x6f, 0x6e, 0x6f, 0x66, 0x73, 0x65, 0x63, 0x75, 0x72, 0x69, 0x74, 0x79, 0x6d, 0x65,
0x63, 0x68, 0x61, 0x6e, 0x69, 0x73, 0x6d, 0x73, 0x6f, 0x66, 0x61, 0x6c, 0x6c, 0x61, 0x76, 0x61, 0x69,
0x6c, 0x61, 0x62, 0x6c, 0x65, 0x74, 0x79, 0x70, 0x65, 0x73, 0x74, 0x6f, 0x6b, 0x65, 0x65, 0x70, 0x69,
0x6e, 0x66, 0x6f, 0x72, 0x6d, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x69, 0x6e, 0x61, 0x6c, 0x6c, 0x69, 0x74,
0x73, 0x6c, 0x6f, 0x63, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x73, 0x61, 0x6e, 0x64, 0x63, 0x6f, 0x6e, 0x73,
0x65, 0x71, 0x75, 0x65, 0x6e, 0x74, 0x6c, 0x79, 0x69, 0x6e, 0x66, 0x6f, 0x72, 0x6d, 0x61, 0x74, 0x69,
0x6f, 0x6e, 0x73, 0x79, 0x73, 0x74, 0x65, 0x6d, 0x73, 0x77, 0x68, 0x65, 0x72, 0x65, 0x6e, 0x66, 0x6f,
0x72, 0x6d, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x69, 0x73, 0x63, 0x72, 0x65, 0x61, 0x74, 0x65, 0x64, 0x70,
0x72, 0x6f, 0x63, 0x65, 0x73, 0x73, 0x65, 0x64, 0x73, 0x74, 0x6f, 0x72, 0x65, 0x64, 0x74, 0x72, 0x61,
0x6e, 0x73, 0x6d, 0x69, 0x74, 0x74, 0x65, 0x64, 0x61, 0x6e, 0x64, 0x64, 0x65, 0x73, 0x74, 0x72, 0x6f,
0x79, 0x65, 0x64, 0x66, 0x72, 0x65, 0x65, 0x66, 0x72, 0x6f, 0x6d, 0x74, 0x68, 0x72, 0x65, 0x61, 0x74,
0x73, 0x54, 0x68, 0x69, 0x73, 0x70, 0x72, 0x6f, 0x6a, 0x65, 0x63, 0x74, 0x69, 0x73, 0x66, 0x69, 0x6e,
0x69, 0x73, 0x68, 0x65, 0x64, 0x62, 0x79, 0x59, 0x55, 0x41, 0x4e, 0x44, 0x4f, 0x4e, 0x47, 0x51, 0x49,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

图 10 明文

3、字节代换

根据原理编写字节代换函数和反字节代换函数。如下图所示。

```
void sub_byte(unsigned char state[4][4]){
    int i=0,j=0;
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            state[i][j]=Sbox[state[i][j]];
        }
    }
}
```

```
void inv_sub_byte(unsigned char state[4][4]){
    int i=0,j=0;
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            state[i][j]=invSBox[state[i][j]];
        }
    }
}
```

图 11　字节代换函数和反字节代换函数

字节代换结果如下图所示。经过计算验证，得到的字节代换后的矩阵是正确的。

```
字节代换:
c4 92 2f c4 0c f7 9c 0c 2f c4 92 2f 9c 0c f7 9c
字节代换逆变换:
88 74 4e 88 81 26 1c 81 4e 88 74 4e 1c 81 26 1c
```

图 12　字节代换结果

4、行移位

根据原理编写行移位函数和逆行移位函数。如下图所示。

```
void row_shift(unsigned char state[4][4]){
    int i=0;
    unsigned char idx;
    idx=state[1][0];
    for(i=0;i<3;i++){
        state[1][i]=state[1][i+1];
    }
    state[1][3]=idx;
    idx=state[2][0];
    state[2][0]=state[2][2];
    state[2][2]=idx;
    idx=state[2][1];
    state[2][1]=state[2][3];
    state[2][3]=idx;
    idx=state[3][3];
    for(i=3;i>0;i--){
        state[3][i]=state[3][i-1];
    }
    state[3][0]=idx;
}
```

```
void inv_shiftrow(unsigned char state[4][4]){
    int i=0;
    unsigned char idx;
    idx=state[1][3];
    for(i=3;i>0;i--){
        state[1][i]=state[1][i-1];
    }
    state[1][0]=idx;
    idx=state[2][0];
    state[2][0]=state[2][2];
    state[2][2]=idx;
    idx=state[2][1];
    state[2][1]=state[2][3];
    state[2][3]=idx;
    idx=state[3][0];
    for(i=0;i<3;i++){
        state[3][i]=state[3][i+1];
    }
    state[3][3]=idx;
}
```

图 13　行移位和逆行移位函数

行移位结果如下图所示。经过对比验证，得到的行移位后的矩阵是正确的。

```
行移位:
88 74 4e 88 26 1c 81 81 74 4e 4e 88 1c 1c 81 26
行移位逆变换:
88 74 4e 88 81 26 1c 81 4e 88 74 4e 1c 81 26 1c
```

图 14　行移位结果

5、列混淆

根据原理编写列混淆函数以及逆列混淆函数。如下图所示。

```
void mixcol(unsigned char state[4][4]){
    int i=0,j=0;
    unsigned char col[4][4];
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            col[i][j]=state[i][j];
        }
    }
    for(j=0;j<4;j++){
        state[0][j]=x_2(col[0][j])^x_3(col[1][j])^col[2][j]^col[3][j];
        state[1][j]=col[0][j]^x_2(col[1][j])^x_3(col[2][j])^col[3][j];
        state[2][j]=col[0][j]^col[1][j]^x_2(col[2][j])^x_3(col[3][j]);
        state[3][j]=x_3(col[0][j])^col[1][j]^col[2][j]^x_2(col[3][j]);
    }
}

void inv_mixcol(unsigned char state[4][4]){
    int i=0,j=0;
    unsigned char col[4][4];
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            col[i][j]=state[i][j];
        }
    }

    for(j=0;j<4;j++){
        state[0][j]=x_e(col[0][j])^x_b(col[1][j])^x_d(col[2][j])^x_9(col[3][j]);
        state[1][j]=x_9(col[0][j])^x_e(col[1][j])^x_b(col[2][j])^x_d(col[3][j]);
        state[2][j]=x_d(col[0][j])^x_9(col[1][j])^x_e(col[2][j])^x_b(col[3][j]);
        state[3][j]=x_b(col[0][j])^x_d(col[1][j])^x_9(col[2][j])^x_e(col[3][j]);
    }
}
```

图 15　列混淆和逆列混淆函数

列混淆结果如下图所示。通过对第一列进行计算验证，得到的列混淆后的矩阵是正确的。

```
列混淆:
c1 8b ea c1 5f 3a cc 5f b1 c1 d0 b1 74 2b f6 74
列混淆逆变换:
88 74 4e 88 81 26 1c 81 4e 88 74 4e 1c 81 26 1c
```

图 16　列混淆结果

6、轮密钥加

根据原理编写轮密钥加函数。如下图所示。

```
void roundkey(unsigned char state[4][4],int k){
    int i=0,j=0;
    for(i=0;i<16;i++){
        round_key[k][i]=ex_key[16*k+i];
    }
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            state[i][j]=state[i][j]^round_key[k][4*j+i];
        }
    }
}
```

图 17　轮密钥加函数

因为轮密钥加需要明文和扩展密钥共同进行，因此我选取前十六个明文对轮密钥加函数进行测试，得到的结果如下图所示。对第一轮轮密钥进行计算验证，得到的轮密钥结果是正确的。

```
明文为:
49 6e 66 6f 72 6d 61 74 69 6f 6e 53 65 63 75 72
第0轮轮密钥为:
88 74 4e 88 81 26 1c 81 4e 88 74 4e 1c 81 26 1c
第1轮轮密钥为:
85 83 d2 14 04 a5 ce 95 4a 2d ba db 56 ac 9c c7
第2轮轮密钥为:
16 5d 14 a5 12 f8 da 30 58 d5 60 eb 0e 79 fc 2c
第3轮轮密钥为:
a4 ed 65 0e b6 15 bf 3e ee c0 df d5 e0 b9 23 f9
第4轮轮密钥为:
fa cb fc ef 4c de 43 d1 a2 1e 9c 04 42 a7 bf fd
第5轮轮密钥为:
b6 c3 a8 c3 fa 1d eb 12 58 03 77 16 1a a4 c8 eb
第6轮轮密钥为:
df 2b 41 61 25 36 aa 73 7d 35 dd 65 67 91 15 8e
第7轮轮密钥为:
1e 72 58 e4 3b 44 f2 97 46 71 2f f2 21 e0 3a 7c
第8轮轮密钥为:
7f f2 48 19 44 b6 ba 8e 02 c7 95 7c 23 27 af 00
第9轮轮密钥为:
a8 8b 2b 3f ec 3d 91 b1 ee fa 04 cd cd dd ab cd
第10轮轮密钥为:
5f e9 96 82 b3 d4 07 33 5d 2e 03 fe 90 f3 a8 33
```

图 18　轮密钥加结果

7、扩展密钥

根据原理编写扩展密钥的函数。如下图所示。

```
void key_expand(unsigned char key[16])
{
    unsigned char RCon[10][4]={
                            {0x01,0x00,0x00,0x00},
                            {0x02,0x00,0x00,0x00},
                            {0x04,0x00,0x00,0x00},
                            {0x08,0x00,0x00,0x00},
                            {0x10,0x00,0x00,0x00},
                            {0x20,0x00,0x00,0x00},
                            {0x40,0x00,0x00,0x00},
                            {0x80,0x00,0x00,0x00},
                            {0x1b,0x00,0x00,0x00},
                            {0x36,0x00,0x00,0x00}};
    int i=0,j=0,l=0;
    unsigned char k;
    unsigned char idx[4];
    for(i=0;i<16;i++){
        ex_key[i]=key[i];
    }
    for(i=16;i<176;i+=4){
        for(j=0;j<4;j++){
            idx[j]=ex_key[i-4+j];
        }
        if(i%16==0){
            k=idx[0];
            for(l=0;l<3;l++){
                idx[l]=idx[l+1];
            }
            idx[3]=k;
            for(l=0;l<4;l++){
                idx[l]=Sbox[idx[l]];
            }
            for(l=0;l<4;l++){
                idx[l]=idx[l]^RCon[i/16-1][l];
            }
        }
        for(l=i;l<i+4;l++){
            ex_key[l]=ex_key[l-16]^idx[l-i];
        }
    }
}
```

图 19　密钥扩展函数

扩展密钥得到的结果如下图所示。经过手动计算验证第二行的密钥，证明结果是正确的。

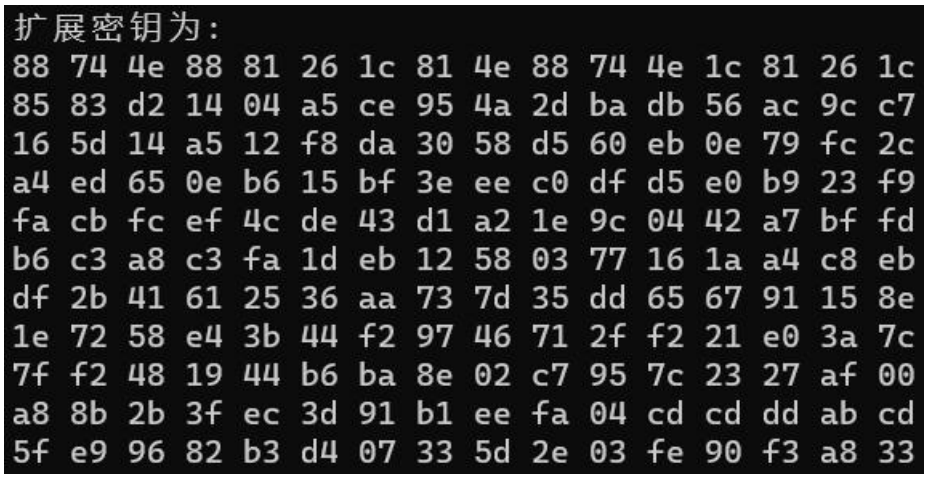

图 20　密钥扩展结果

8、AES 加密与解密

通过对上述函数的验证，均得到正确的结果，因此使用上述函数完成对 AES 加密函数以及解密函数的编写。由于 AES 加密过程中前九轮需要通过字节代换、行移位、列混淆以及轮密钥加四个步骤，而第十轮只需要通过字节代换、行移位以及轮密钥加三个步骤，因此对两种不同的加密过程通过编写两个函数进行区分。最终得到的加密与解密过程如下图所示。

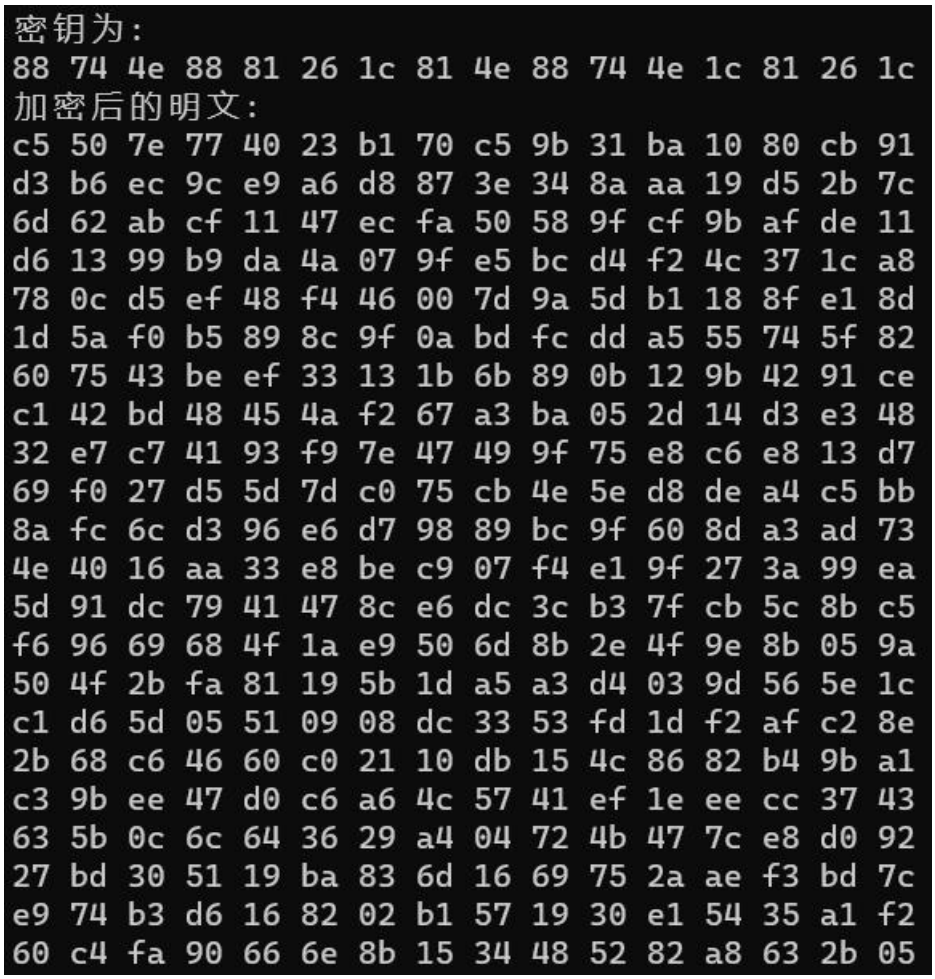

图 21　AES 加密结果

```
解密后的明文:
49 6e 66 6f 72 6d 61 74 69 6f 6e 53 65 63 75 72
69 74 79 69 73 61 6d 75 6c 74 69 64 69 73 63 69
70 6c 69 6e 61 72 79 61 72 65 61 6f 66 73 74 75
64 79 61 6e 64 70 72 6f 66 65 73 73 69 6f 6e 61
6c 61 63 74 69 76 69 74 79 77 68 69 63 68 69 73
63 6f 6e 63 65 72 6e 65 64 77 69 74 68 74 68 65
64 65 76 65 6c 6f 70 6d 65 6e 74 61 6e 64 69 6d
70 6c 65 6d 65 6e 74 61 74 69 6f 6e 6f 66 73 65
63 75 72 69 74 79 6d 65 63 68 61 6e 69 73 6d 73
6f 66 61 6c 6c 61 76 61 69 6c 61 62 6c 65 74 79
70 65 73 74 6f 6b 65 65 70 69 6e 66 6f 72 6d 61
74 69 6f 6e 69 6e 61 6c 6c 69 74 73 6c 6f 63 61
74 69 6f 6e 73 61 6e 64 63 6f 6e 73 65 71 75 65
6e 74 6c 79 69 6e 66 6f 72 6d 61 74 69 6f 6e 73
79 73 74 65 6d 73 77 68 65 72 65 6e 66 6f 72 6d
61 74 69 6f 6e 69 73 63 72 65 61 74 65 64 70 72
6f 63 65 73 73 65 64 73 74 6f 72 65 64 74 72 61
6e 73 6d 69 74 74 65 64 61 6e 64 65 73 74 72 6f
79 65 64 66 72 65 65 66 72 6f 6d 74 68 72 65 61
74 73 54 68 69 73 70 72 6f 6a 65 63 74 69 73 66
69 6e 69 73 68 65 64 62 79 59 55 41 4e 44 4f
4e 47 51 49 00 00 00 00 00 00 00 00 00 00 00 00
I n f o r m a t i o n S e c u r i t y i s a m u l t i d i s c i p l
i n a r y a r e a o f s t u d y a n d p r o f e s s i o n a l a c t
i v i t y w h i c h i s c o n c e r n e d w i t h t h e d e v e l o
p m e n t a n d i m p l e m e n t a t i o n o f s e c u r i t y m e
c h a n i s m s o f a l l a v a i l a b l e t y p e s t o k e e p i
n f o r m a t i o n i n a l l i t s l o c a t i o n s a n d c o n s
e q u e n t l y i n f o r m a t i o n s y s t e m s w h e r e n f o
r m a t i o n i s c r e a t e d p r o c e s s e d s t o r e d t r a
n s m i t t e d a n d d e s t r o y e d f r e e f r o m t h r e a t
s T h i s p r o j e c t i s f i n i s h e d b y Y U A N D O N G Q I
```

图 22　AES 解密结果

从解密结果可以看出，解密得到的信息与明文是一致的，说明加密函数和解密函数的编写均是正确的。

9、CMAC 认证

因为输入的明文长度为 340，并不是 16 的倍数，因此需要通过在最后补一个 1 和若干个 0 的方式使其长度变为 16 的倍数，且在最后应使用 k2 密钥进行加密。产生 K1、K2 密钥的函数以及填充明文使其长度为 16 的倍数的函数如下图所示。

```
void k12(unsigned char key[16],unsigned char k1[16],unsigned char k2[16]){
    unsigned char l[16];
    unsigned char zero[16]={0x00};
    int i=0;
    encrypt(zero,key,l);
    if(l[0]>=0x80){
        for(i=0;i<16;i++){
            k1[i]=l[i]<<1;
        }
        k1[15]=k1[15]^0x87;
    }
    else{
        for(i=0;i<16;i++){
            k1[i]=l[i]<<1;
        }
    }
    if(k1[0]>=0x80){
        for(i=0;i<16;i++){
            k2[i]=k1[i]<<1;
        }
        k2[15]=k2[15]^0x87;
    }
    else{
        for(i=0;i<16;i++){
            k2[i]=k1[i]<<1;
        }
    }
}
```

图 23　K1、K2 产生函数

```
void fill_n(unsigned char* last_text,int len,unsigned char out_text[16]){
    int i=0;
    for(i=0;i<16;i++){
        if(i<len){
            out_text[i]=last_text[i];
        }
        else if(i==len){
            out_text[i]=0x80;
        }
        else{
            out_text[i]=0x00;
        }
    }
}
```

图 24　填充函数

根据上面给出的 K1、K2 产生函数以及填充函数编写 CMAC 函数，并产生认证码。最终得到的认证码结果如下图所示。

```
认证码为：
80 1d 08 a8 32 81 b2 28 df ae c2 29 e0 22 cb 38
```

图 25　认证码结果

四、实验总结

本次实验主要通过学习了解 AES 加密算法的原理以及基于 AES 的 CMAC 的原理，自己编写代码实现对一段消息的加密以及认证。通过本次实验，我对 AES 加密的过程有了更深入的理解，对 CMAC 的加密认证过程有了更全面的了解。