《应用密码学》实验报告

课程：__应用密码学__ 实验名称：AES 变换操作的实现

姓名：杨佳伲 实验日期：2024.4.1

学号：2022122006 实验报告日期：2024.4.1

班级：信安实验 221

教师评语： 签名： 日期：	成绩：

一、实验名称

AES 变换操作的实现

二、实验环境（详细说明运行的系统、平台及代码等）

1．RedC2.0++

三、实验目的

（1）加深对 AES 算法的理解；

（2）阅读标准(fips-197)和文献，提高自学能力；

（3）加深对模块化设计的理解，提高编程实践能力。

四、实验内容、步骤及结果

1．实验内容

（1）按照 AES 算法，完成 AES 算法 S 盒、行移位、列混合、轮密钥加操作；

2．实验步骤

主要函数实现方法介绍

(1) 、轮密钥加

首先，明文分组与密钥都为 128 位的其实等于 16 个字节，将明文与密钥分别对应的 16 个字节行向优先填入两个 4*4 的矩阵中。先定义两个 4*4 的二维数组，并将明文分组与密钥行向优先填入这两个数组；再定义两个指针，对于使这两个数组对应的元素准确异或可以用 for 语句来实现，因为主函数要将轮密钥加作为副函数来调用，所以将数组指针作为参数来实现这个模块。具体代码：

```
void AddRoundKey_06(unsigned char state[4][4], const unsigned 
char roundKey[4][4]) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] ^= roundKey[i][j];
        }
    }
}
```

(2) 、字节代替 利用二重循环相应替代即可

```
void SubBytes_06(unsigned char state[4][4]) {
    for(int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] = sbox[state[i][j]];
        }
    }
}
```

(3) 、行移位

在 c 语言中对于 4*4 的数组的行列是从 0~3 编号的，即：
第 0 行左移 0 位、第 1 行左移 1 位、第 2 行左移 2 位、第 3 行左移 3 位。
对于左移可以先：int b[4][4]，在 b 中保留左移后的结果。通过
b[j][i]=*(p+j*4+(i+j)%4);来实现左移。最后再将 b 中的结果填入到原数组中。

```
int ShiftRows_06(uint8_t (*state)[4]) {
    uint32_t block[4] = {0};

    for (int i = 0; i < 4; ++i) {

        LOAD32H(block[i], state[i]);
        block[i] = ROF32(block[i], 8*i);
        STORE32H(block[i], state[i]);
    }

    return 0;
}
```

(4) 、列混合

列混淆是将输入的 4*4 矩阵左乘一个常熟矩阵（常数矩阵是一个常态，固定不变），在这里左乘是矩阵的左乘，但又有所不同。乘法为有限域 GF(2^8)上的乘法，加法为有限域 GF(2^8)上的加法即异或。列混淆的左乘与普通矩阵的左乘不同之处就在于乘法为有限域 GF(2^8)上的乘法，加法为有限域 GF(2^8)上的加法即异或。

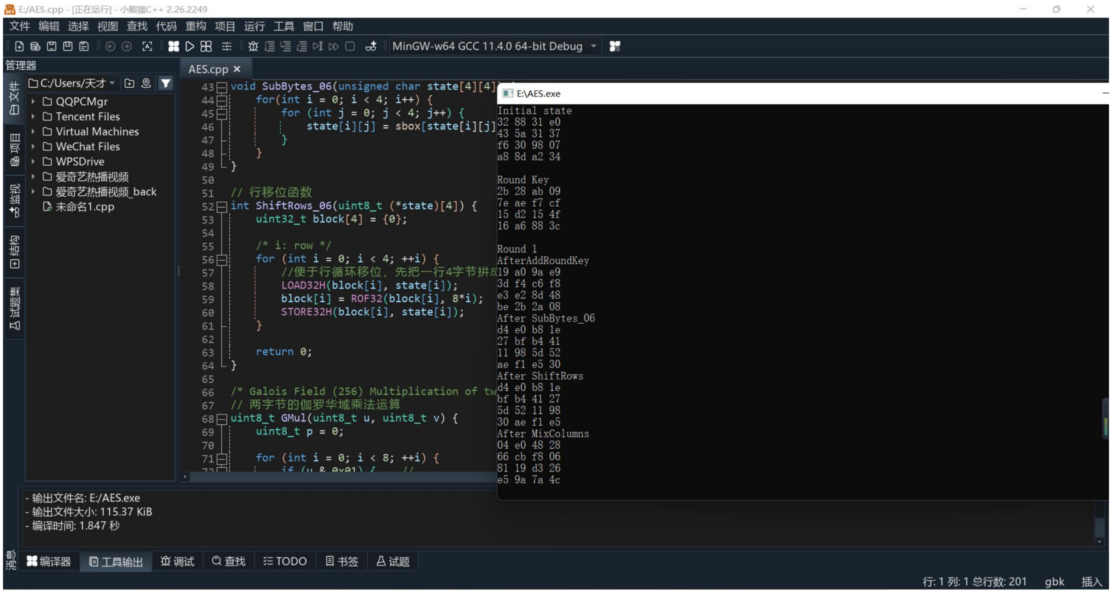
```
int MixColumns_06(uint8_t (*state)[4]) {
    uint8_t tmp[4][4];
    uint8_t M[4][4] = {{0x02, 0x03, 0x01, 0x01},
        {0x01, 0x02, 0x03, 0x01},
        {0x01, 0x01, 0x02, 0x03},
        {0x03, 0x01, 0x01, 0x02}};

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

    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            state[i][j] = GMul(M[i][0], tmp[0][j]) ^ GMul(M[i]
[1], tmp[1][j])
            ^ GMul(M[i][2], tmp[2][j]) ^ GMul(M[i][3], tmp[3][
j]);
        }
    }

    return 0;
}
```

3.实验结果



五、实验中的问题及心得

对于ＡＥＳ密钥扩展，在编写程序过程中，核心部分的程序是借鉴的优质代码，课后自己也重新模仿着敲了一遍代码，对 AES 的密钥扩展有了更深的了解和掌握，也掌握了 AES 密钥扩展的规则。

在开始编写代码之前，深入理解 AES 算法的原理是非常重要的。只有理解了算法的工作流程和每个步骤的目的，才能更好地实现代码。将算法分解成小模块，比如 S 盒替代、行移位、列混合等，有助于代码的组织和维护。每个模块都应该完成特定的功能，使得代码更易读、易懂。利用现有的函数和数据结构来简化代码编写过程。在这个例子中，我使用了现有的 S 盒和列混合的乘法函数，避免了重复造轮子。编写完代码后，进行全面的测试是必不可少的。通过提供不同的输入数据和密钥，确保代码在各种情况下都能正常工作。同时，调试过程中要注重细节，确保每一步操作都正确无误。

总的来说，编写 AES 算法的代码需要耐心和细心，但通过理解算法原理、模块化设计和充分测试，可以写出高质量的代码。

附件：程序代码

```
#include <stdio.h>
#include <stdint.h>

#define LOAD32H(x, y) \
do { (x) = ((uint32_t)((y)[0] & 0xff)<<24) | ((uint32_t)((y)[1]
 & 0xff)<<16) | \
((uint32_t)((y)[2] & 0xff)<<8)  | ((uint32_t)((y)[3] & 0xff));}
 while(0)

// uint32_t x -> uint8_t y[4]
#define STORE32H(x, y) \
do { (y)[0] = (uint8_t)(((x)>>24) & 0xff); (y)[1] = (uint8_t)(
((x)>>16) & 0xff);   \
(y)[2] = (uint8_t)(((x)>>8) & 0xff); (y)[3] = (uint8_t)((x) &
0xff); } while(0)

// uint32_t x 循环左移n 位
#define ROF32(x, n)  (((x) << (n)) | ((x) >> (32-(n))))
// uint32_t x 循环右移n 位
#define ROR32(x, n)  (((x) >> (n)) | ((x) << (32-(n))))



// AES 的S 盒
static unsigned char sbox[16*16]=
{// populate the Sbox matrix
/* 0 1 2 3 4 5 6 7 8 9 a b c d e f */
/*0*/ 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x
01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
/*1*/ 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0x
d4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
/*2*/ 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0x
a5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
/*3*/ 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x
12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
/*4*/ 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x
3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
/*5*/ 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0x
cb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
/*6*/ 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0x
f9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
/*7*/ 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0x
```

```
b6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
/*8*/ 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0x
a7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
/*9*/ 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0x
ee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
/*a*/ 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0x
d3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
/*b*/ 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x
56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
/*c*/ 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0x
dd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
/*d*/ 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x
35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
/*e*/ 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x
1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
/*f*/ 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x
99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16};


// 字节替换函数
void SubBytes_06(unsigned char state[4][4]) {
    for(int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] = sbox[state[i][j]];
        }
    }
}

// 行移位函数
int ShiftRows_06(uint8_t (*state)[4]) {
    uint32_t block[4] = {0};

    /* i: row */
    for (int i = 0; i < 4; ++i) {
        //便于行循环移位，先把一行 4 字节拼成 uint_32 结构，移位后再转
成独立的 4 个字节 uint8_t
        LOAD32H(block[i], state[i]);
        block[i] = ROF32(block[i], 8*i);
        STORE32H(block[i], state[i]);
    }

    return 0;
}
```

```c
/* Galois Field (256) Multiplication of two Bytes */
// 两字节的伽罗华域乘法运算
uint8_t GMul(uint8_t u, uint8_t v) {
    uint8_t p = 0;

    for (int i = 0; i < 8; ++i) {
        if (u & 0x01) {    //
            p ^= v;
        }

        int flag = (v & 0x80);
        v <<= 1;
        if (flag) {
            v ^= 0x1B; /* x^8 + x^4 + x^3 + x + 1 */
        }

        u >>= 1;
    }

    return p;
}
// 列混合
int MixColumns_06(uint8_t (*state)[4]) {
    uint8_t tmp[4][4];
    uint8_t M[4][4] = {{0x02, 0x03, 0x01, 0x01},
        {0x01, 0x02, 0x03, 0x01},
        {0x01, 0x01, 0x02, 0x03},
        {0x03, 0x01, 0x01, 0x02}};

    /* copy state[4][4] to tmp[4][4] */
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j){
            tmp[i][j] = state[i][j];
        }
    }

    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {  //伽罗华域加法和乘法
            state[i][j] = GMul(M[i][0], tmp[0][j]) ^ GMul(M[i]
[1], tmp[1][j])
            ^ GMul(M[i][2], tmp[2][j]) ^ GMul(M[i][3], tmp[3][
j]);
        }
    }
```

```c
    return 0;
}


// 轮密钥加函数
void AddRoundKey_06(unsigned char state[4][4], const unsigned
char roundKey[4][4]) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] ^= roundKey[i][j];
        }
    }
}

// 主函数
int main() {
    // 初始状态矩阵
    unsigned char initial_state[4][4] = {
        {0x32, 0x88, 0x31, 0xe0},
        {0x43, 0x5a, 0x31, 0x37},
        {0xf6, 0x30, 0x98, 0x07},
        {0xa8, 0x8d, 0xa2, 0x34}
    };

    // 轮密钥
    unsigned char round_key[4][4] = {
        {0x2b, 0x28, 0xab, 0x09},
        {0x7e, 0xae, 0xf7, 0xcf},
        {0x15, 0xd2, 0x15, 0x4f},
        {0x16, 0xa6, 0x88, 0x3c}
    };

    // 打印初始状态
    printf("Initial state\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02x ", initial_state[i][j]);
        }
        printf("\n");
    }

    // 第一轮加密
    printf("\nRound Key\n");
```

```c
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02x ", round_key[i][j]);
        }
        printf("\n");
    }

    printf("\nRound 1\n");

// 轮密钥加
AddRoundKey_06(initial_state, round_key);
printf("AfterAddRoundKey\n");
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        printf("%02x ", initial_state[i][j]);
    }
    printf("\n");
}

// 字节替换
SubBytes_06(initial_state);
printf("After SubBytes_06\n");
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        printf("%02x ", initial_state[i][j]);
    }
    printf("\n");
}

// 行移位
ShiftRows_06(initial_state);
printf("After ShiftRows\n");
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        printf("%02x ", initial_state[i][j]);
    }
    printf("\n");
}

// 列混合
MixColumns_06(initial_state);
printf("After MixColumns\n");
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
```

```c
            printf("%02x ", initial_state[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```