密码学引论实验

AES 算法

2023 年 3 月 26 日

成员信息及完成部分：

靳立伟：网安二班 202100460079-AES 算法实现构思

戴方奇：网安二班 202100460092-实验报告编写与总结

王子瑞：网安二班 202100460088-密码算法的分析对比

王成盟：网安二班 202100460091-辅助进行算法分析

摘 要

AES,Advanced Encryption Standard，是一种分组对称加密算法，作为一种区块加密标准，代替 DES 算法被美国联邦政府所采用，是 2006 年后对称密钥加密中最流行的算法之一。

AES 的区块长度固定为 128 比特，密钥长度可以是 128，192 或 256 比特；本次实验中，任务一：中选取 AES-128 加密算法，对小组成员的学号进行加密（不满 128bit 在最后填充足够多 0）；任务二：选取特定随机明文集合，对于同一密钥，对实验所实现算法与算法库各种运行时间进行比较。

关键词: AES 分组加密 对称加密 状态矩阵

目录

1 问题重述

任务一：实现 AES-128 加密算法，并设明文和密钥均为学号（按 ASCII 码表示，每位数字对应 8-bit，不满 128-bit 的在最后填充足够多的 0，明文最左侧为最低位），求对应的密文。

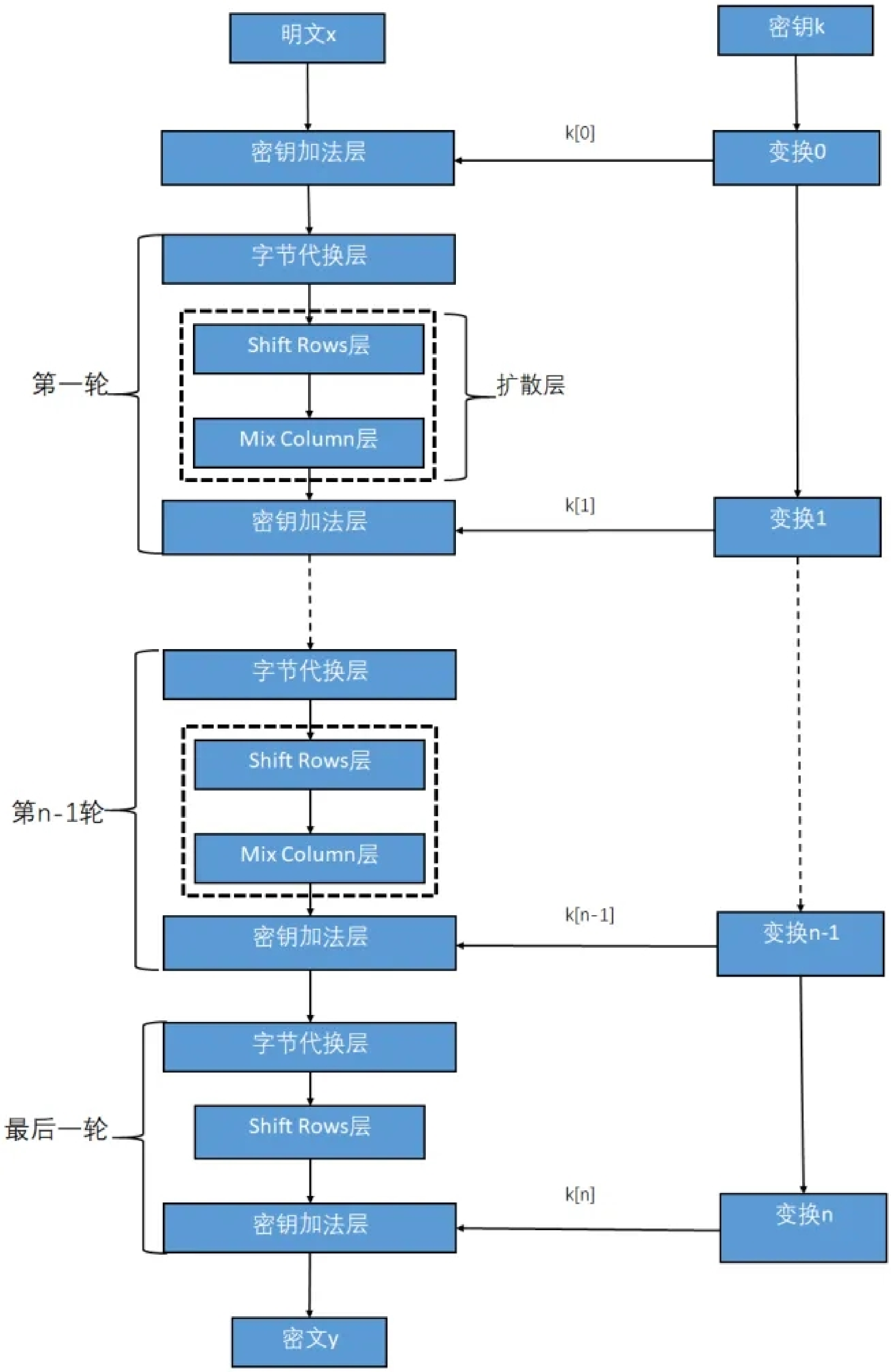
任务二：对同一个密钥，分别加密 100 组满足以下结构的 256 个明文组成的集合（提前生成好），估计算法加密一次的运行时间，并与直接调用算法库进行加密的运行时间进行比较。其中 256 个明文满足：第 0 字节遍历 256 种可能，其余字节取任意常数。

2 实验原理

AES 是可以进行多轮加密的分组密码算法，密钥长度不同，推荐加密的轮数也不同。本次实验实现的是 AES-128，加密轮数为 10。

AES 的算法核心就是实现一轮中的所有操作。在每一轮加密中包含字节代换、行位移、列混合和轮密钥异或四个步骤。其中在第一轮迭代之前，先将明文和原始密钥进行一次异或加密操作，最后一轮迭代不执行列混合。

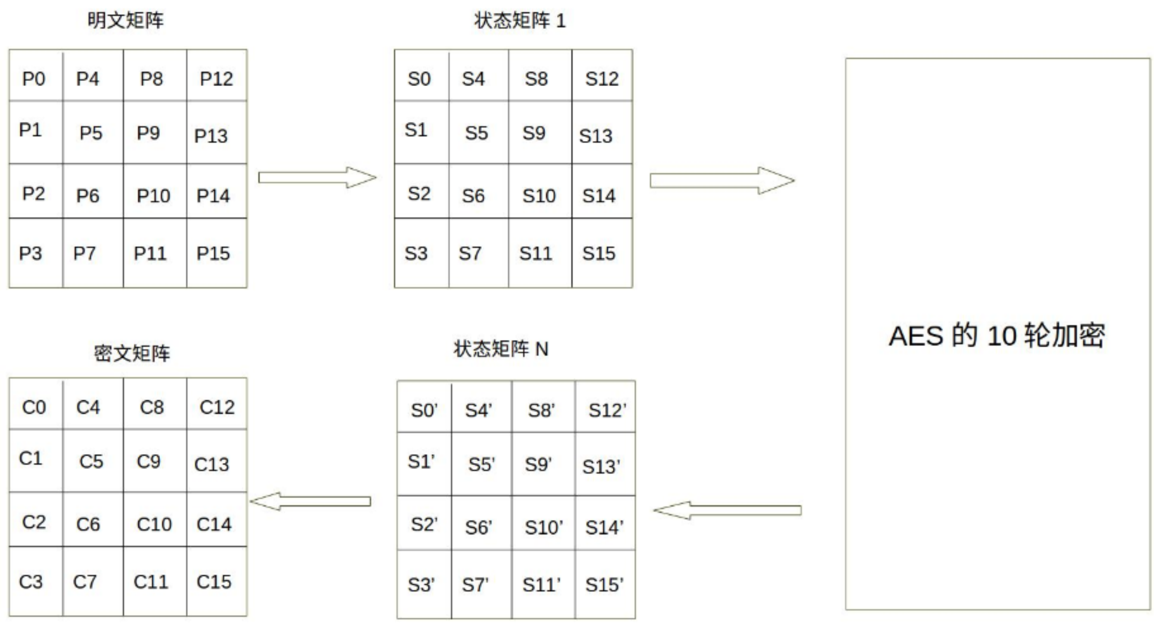
具体加密流程见下图：


明文x
密钥k
k[0]
密钥加法层
变换0
字节代换层
Shift Rows层
Mix Column层
第一轮
扩散层
k[1]
密钥加法层
变换1
字节代换层
Shift Rows层
Mix Column层
第n-1轮
k[n-1]
密钥加法层
变换n-1
字节代换层
Shift Rows层
最后一轮
k[n]
密钥加法层
变换n
密文y

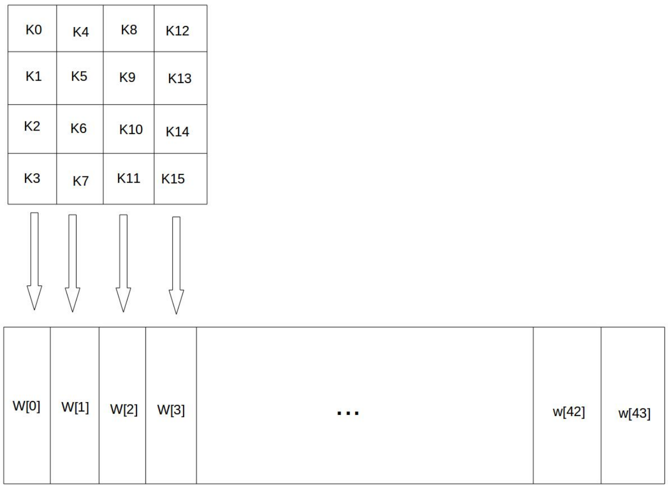
3 实验一过程

3.1 格式处理

AES 的处理单位是字节，128 位的输入明文分组 P 和输入密钥 K 都被分成 16 个字节，分别记为 P = P0 P1 ···P15 和 K = K0 K1 ···K15。如，明文分组为 P = abcdefghijklmnop, 其中的字符 a 对应 P0，p 对应 P15。其实我们将明文转化为 ascll 码表示为一个字节。一般地，明文分组用字节为单位的正方形矩阵描述，称为状态矩阵。在算法的每一轮中，状态矩阵的内容不断发生变化，最后的结果作为密文输出。该矩阵中字节的排列顺序为从上到下、从左至右依次排列，如下图所示：



类似地，128 位密钥也是用字节为单位的 4×4 大小矩阵表示，所以矩阵的每一列被称为 1 个 32 位比特字。通过密钥编排函数将该密钥矩阵被扩展成一个 44 个字组成的序列 W[0],W[1], ···,W[43], 该序列的前 4 个元素 W[0],W[1],W[2],W[3] 是原始密钥，用于加密运算中的初始密钥加（下面介绍）; 后面 40 个字分为 10 组，每组 4 个字（128 比特）分别用于 10 轮加密运算中的轮密钥加，如下图所示：



3.2 轮密钥扩展

每轮加密的密钥都是由与原始密钥变化而来的，第 i 轮加密需要用的密钥序列为 W[4i]、W[4i+1]、W[4i+2]、W[4i+3]，这时我们就需要跟据加密轮数随时扩展我们的轮密钥。

已知原始密钥 W[0],W[1]W[2],W[3]，后续密钥通过递归函数得到：

$$[w[i]] = \begin{cases} W[i] = W[i-4]\ W[i-1], & imod4 \neq 0 \\ W[i] = W[i-4]\ g(W[i-1]), & imod4 = 0. \end{cases} \tag{1}$$

其中函数 g 由 3 部分组成：字循环、字节代换和轮常量异或

```
const unsigned int Rcon[11] = { 0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36 };//G函数中异或轮密钥
int g_f(unsigned char(*key)[44], unsigned int col)
{
  int ret = 0;
  for (int i = 0; i < 4; i++)//先放到下一列，且行移位
  {
    key[i][col] = key[(i + 1) % 4][col - 1];
  }
  keyarr_S(key, col);//S盒置换
  key[0][col] = key[0][col] ^ Rcon[col / 4];//异或
```

```
    return ret;
}
    int Extendkey(const unsigned char(*pkey)[4], unsigned char (*key)[44])
{
    int ret = 0;
    for (int i = 0; i < 16; i++)//主密钥赋值
    {
        key[i & 0x03][i >> 2] = pkey[i & 0x03][i >> 2];
    }

    for (int i = 1; i < 11; i++)//10轮
    {
        g_f(key, 4 * i);//G函数
        for (int k = 0; k < 4; k++)//每一轮的第一列得到
        {
            key[k][4 * i] = key[k][4 * i] ^ key[k][4 * (i - 1)];
        }
        for (int j = 1; j < 4; j++)//后三列
        {
            for (int k = 0; k < 4; k++)
            {
                key[k][4 * i + j] = key[k][4 * (i - 1) + j] ^ key[k][4 * i + j - 1];
            }
        }
    }
    return ret;
}
```

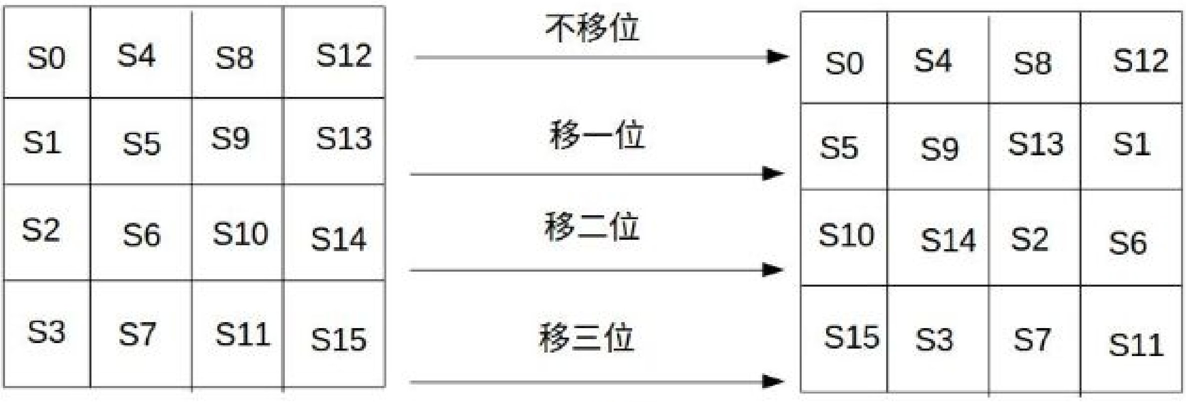
3.3 字节代换

AES 的字节代换其实就是查表代换，分别定义一个 s 盒与逆 s 盒以供加解密使用。先将明文做成一个 4×4 的明文矩阵，每个明文由 ascll 码转为一个字节，实现分组长为 128bit 每组的加密。然后将该字节转换为两位 16 进制数表示，根据高位为行值，低位作为列值，取出 S 盒或者逆 S 盒中对应的元素作为输出，就实现了字节代换。(s 盒或者说代换表是已经预设好的，不需要加命者自行设计)。在预处理阶段我们先在代码中给出预设 s 盒与逆 s 盒的数据，再进行字节处理。

```
int S_ps(unsigned char* plain)//S盒代换
{
    int ret = 0;
    for (int i = 0; i < 16; i++)
    {
        plain[i] = S_Table[plain[i] >> 4][plain[i] & 0x0F];//利用该明文的前四位作行，后四位作列，代换
    }
    return ret;
}
int S_cs(unsigned char* cipher)//逆S盒代换
{
    int ret = 0;
    for (int i = 0; i < 16; i++)
    {
        cipher[i] = ReS_Table[cipher[i] >> 4][cipher[i] & 0x0F];
    }
    return ret;
}
```

3.4 行位移

行移位是一个简单的左循环移位操作。当密钥长度为 128 比特时，状态矩阵的第 0 行左移 0 字节，第 1 行左移 1 字节，第 2 行左移 2 字节，第 3 行左移 3 字节，同理行移位的逆变换也就是将状态矩阵中的每一行执行相反的移位操作，即状态矩阵的第 0 行右移 0 字节，第 1 行右移 1 字节，第 2 行右移 2 字节，第 3 行右移 3 字节。如下图所示：



行移位

具体代码如下：

```
int Row_shift(unsigned int* plain)
{
    int ret = 0;
    //不用移位
    plain[1] = (plain[1] >> 24 | plain[1] << 8);//循环左移一个明文
    plain[2] = (plain[2] >> 16 | plain[2] << 16);//循环左移两个
    plain[3] = (plain[3] >> 8 | plain[3] << 24);//循环左移三个
    return ret;
}
int ReRow_shift(unsigned int* cipher)//解密逆变换
{
    int ret = 0;
    cipher[1] = (cipher[1] >> 8 | cipher[1] << 24);
    cipher[2] = (cipher[2] >> 16 | cipher[2] << 16);
    cipher[3] = (cipher[3] >> 24 | cipher[3] << 8);
```

```
    return ret;
}
```

3.5 列混合

为了实现加密算法的混淆扩散特性，提高安全性，列混合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，如下图的公式所示：

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

具体代码如下：

```
int column_mix(unsigned char(*plain)[4])
{
    int ret = 0;
    unsigned char temp[4][4];
    memcpy(temp, plain, 16);
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            plain[i][j] = mul(mixarr[i][0], temp[0][j]) ^ mul(mixarr[i][1], temp[1][j]) ^ mul(mixarr[i][2], temp[2][j]) ^ mul(mixarr[i][3], temp[3][j]);
        }
    }
    return ret;
}
```

3.6 轮密钥异或

就是将 128 位轮密钥 W[4i]、W[4i+1]、W[4i+2]、W[4i+3] 与同状态矩阵中的数据进行逐位异或操作得到，经过字节代换、行位移、列混合的一组 4×4 明文矩阵称为本轮的状态矩阵。

```
int and_round(unsigned char(*plain)[4], unsigned char(*key)
    [44], unsigned int n)
{
    int ret = 0;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            plain[i][j] ^= key[i][n + j];
        }
    }
    return ret;
}
```

4 实验二过程

4.1 实现所要求的明文集合

```
for (int n = 0; n < 100; n++)//100组
    {
        unsigned char mes[4096];//256个128bit的明文
        for (int m = 0; m < 256; m++)
        {
            mes[m * 16] = char(m);//将256个明文的每个第0字节遍历
        }
        for (int r = 1; r < 4096 && r % 16 != 0; r++)
```

```
    {
        srand((unsigned int)(time(NULL)));
        mes[r] = char(rand() % 256);//其他的随机
    }

    for (int p = 0; p < 256; p++)
    {
        for (int q = 0; q < 16; q++)
        {
            message[q] = mes[p * 16 + q];//每128bit传一次
        }
        unsigned char plain[4][4];
        unsigned char plain2[16];
        unsigned int plain3[16];
        for (int i = 0; i < 12; i++)
        {
            plain[i / 4][i % 4] = message[i];
        }
        for (int i = 0; i < 4; i++)
        {
            plain[3][i] = 0x0;
        }
```

4.2 时间比较

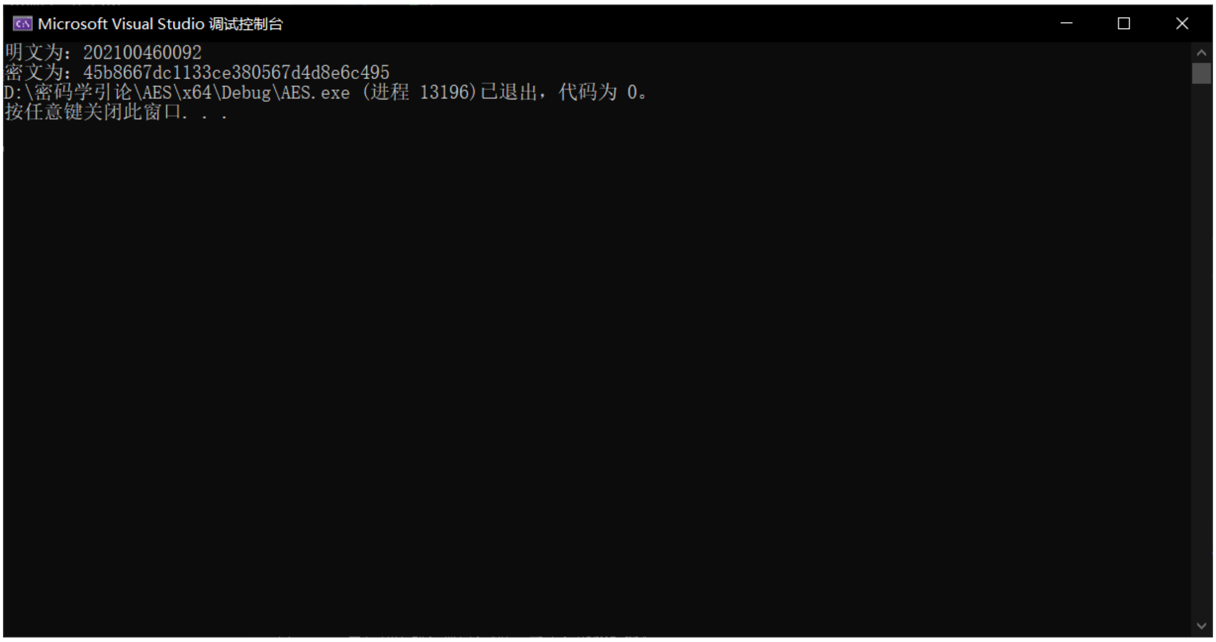
分别运行库函数和所实现函数，再进行时间比较。

5 实验结论

5.1 实验一

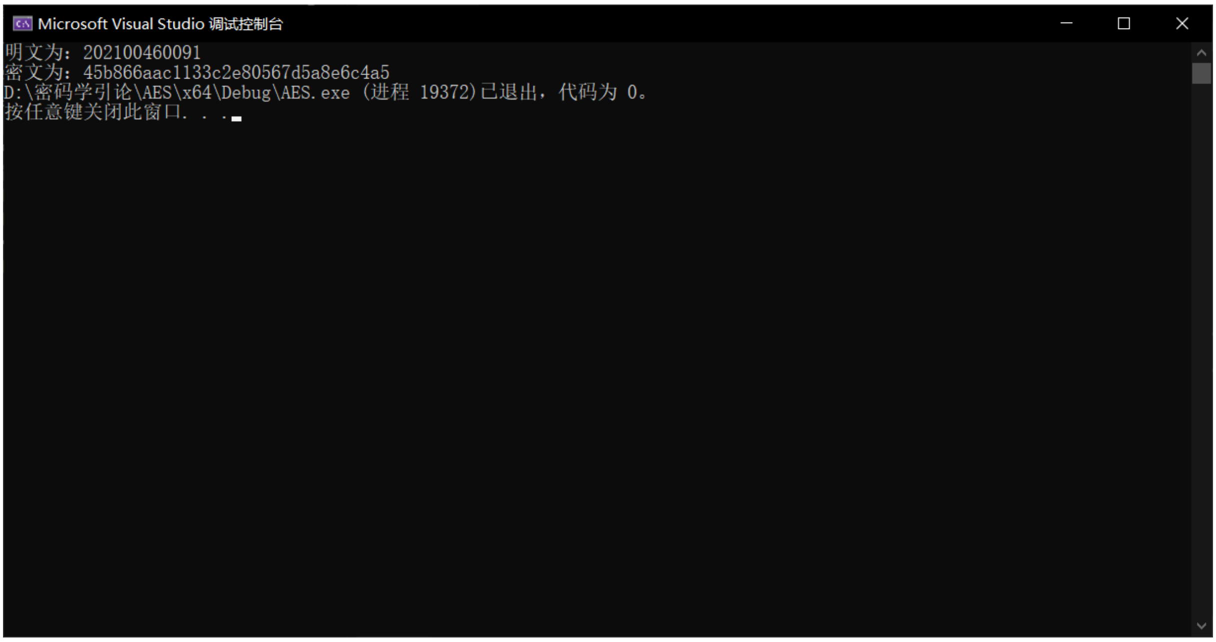
靳立伟学号 202100460079，生成密文：45b8df23c113b58f805690ef8e6c571a

戴方奇学号 202100460092，生成密文：45b8667dc1133ce380567d4d8e6c495



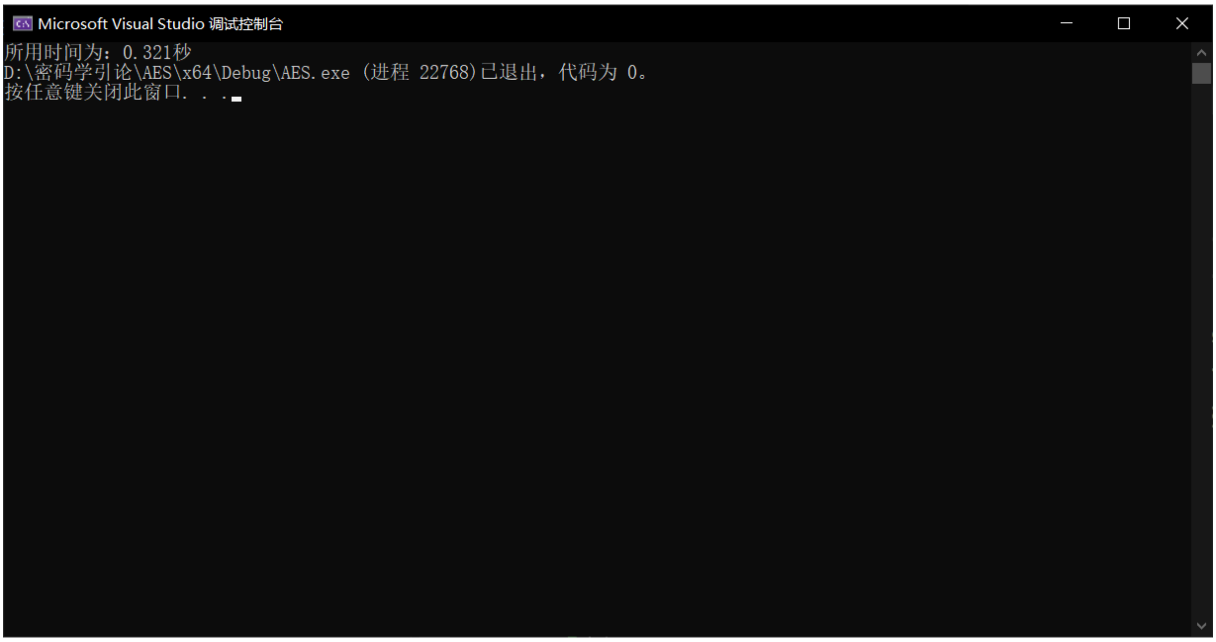
王子瑞学号 202100460088，生成密文：45b8424cc113b3d0805659918e6cef8d

王成盟学号 202100460091，生成密文：45b866aac1133c2e80567d5a8e6c4a5

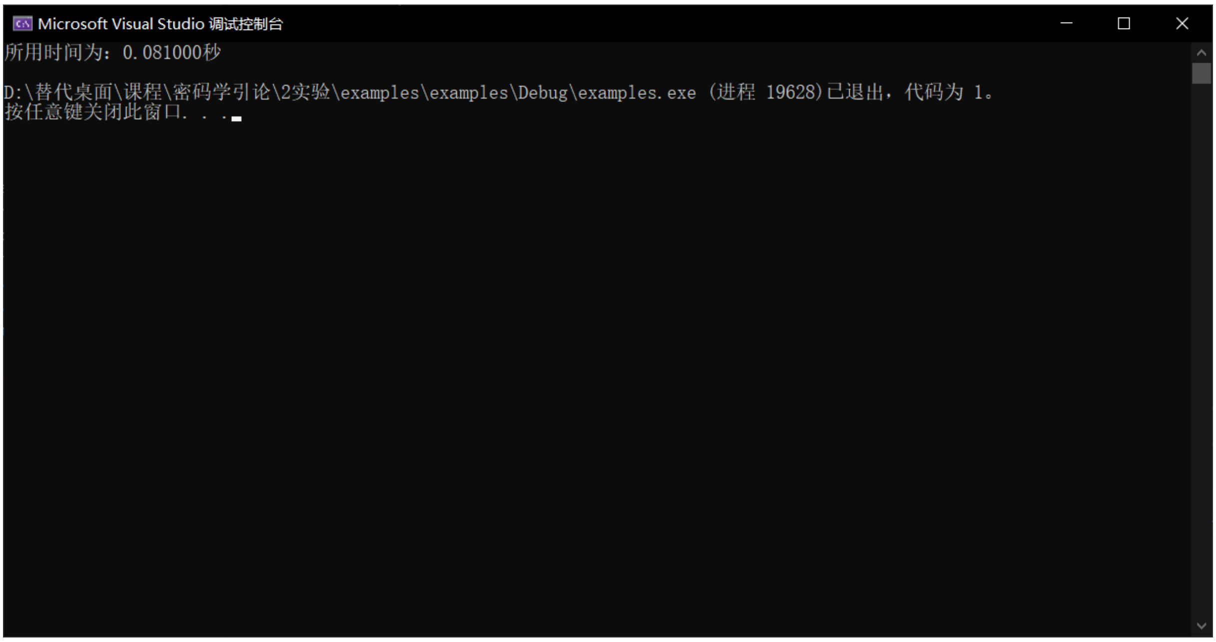


5.2 实验二

自己实习的 aes 加密所要求的明文集合时间：



库函数 aes 加密所要求的明文集合时间：



6 实验总结分析

可以看出我组实现 aes 的加密时间几乎是库函数的四倍，并没有太过于悬殊，推测库函数相比我们的函数在代码层面可能在读写方面的有相关的优化，或者在实现密码加密一轮的各个步骤中做出了优化。

参考文献

[1] TimeShatter：AES 加密算法的详细介绍与实现

[2] 知乎看雪：《密码学基础：AES 加密算法》

7 附录

7.1 任务一完整代码

```

    //AES-128 bit
#include <iostream>
using namespace std;
const unsigned char S_Table[16][16] =
{
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
};//S盒
const unsigned char ReS_Table[16][16] =
{
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, 0
```

```
x58, 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
};//逆S盒
const unsigned char mixarr[4][4] =
{
0x02, 0x03, 0x01, 0x01,
0x01, 0x02, 0x03, 0x01,
0x01, 0x01, 0x02, 0x03,
0x03, 0x01, 0x01, 0x02
};//列混合的矩阵
const unsigned int Rcon[11] = { 0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36 };//G函数中异或轮密钥
int S_ps(unsigned char*);
int S_cs(unsigned char*);
int Row_shift(unsigned int*);
```

```c
int ReRow_shift(unsigned int*);
int column_mix(unsigned char(*plain)[4]);
char mul(unsigned char, unsigned char);
int keyarr_S(unsigned char(*key)[44], unsigned int col);
int g_f(unsigned char(*key)[44], unsigned int col);
int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key)[44]);
int and_round(unsigned char(*plain)[4], unsigned char(*key)[44], unsigned int n);
int main()
{
  unsigned char pkey[4][4];
  unsigned char str[13] = "202100460079";
  for (int i = 0; i < 12; i++)
  {
    pkey[i / 4][i % 4] = str[i];
  }
  for (int i = 0; i < 4; i++)
  {
    pkey[3][i] = 0x0;//不够的补0
  }

  unsigned char key[4][44];//11轮密钥
  Extendkey(pkey, key);//由主密钥生成10轮密钥
  //for (int m = 0; m < 100; m++)
  //{

  unsigned char message[13] = "202100460079";
  unsigned char plain[4][4];
  unsigned char plain2[16];
  unsigned int plain3[16];
  for (int i = 0; i < 12; i++)
```

```
{
    plain[i / 4][i % 4] = message[i];
}
for (int i = 0; i < 4; i++)
{
    plain[3][i] = 0x0;//不够的补0
}
and_round(plain, key, 0);//明文与密钥异或
for (int i = 0; i < 10; i++)//10轮
{
    for (int j = 0; j < 16; j++)
    {
        plain2[j] = plain[j / 4][j % 4];
    }
    S_ps(plain2);//S盒置换
    for (int j = 0; j < 16; j++)
    {
        plain3[j] = int(plain2[j]);
    }
    Row_shift(plain3);//行移位
    for (int j = 0; j < 16; j++)
    {
        plain[j / 4][j % 4] = char(plain3[j]);
    }
    column_mix(plain);//列混合
    and_round(plain, key, 4 * (i + 1));//异或运算
}
cout << "密文为: ";
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 4; j++)
    {
```

```
            cout << hex << int(plain[i][j]);
        }
    }



    //}


    return 0;
}
int S_ps(unsigned char* plain)//S盒代换
{
    int ret = 0;
    for (int i = 0; i < 16; i++)
    {
        plain[i] = S_Table[plain[i] >> 4][plain[i] & 0x0F];//利用该明文的前四位作行，后四位作列，代换
    }
    return ret;
}
int S_cs(unsigned char* cipher)//逆S盒代换
{
    int ret = 0;
    for (int i = 0; i < 16; i++)
    {
        cipher[i] = ReS_Table[cipher[i] >> 4][cipher[i] & 0x0F];
    }
    return ret;
}
int Row_shift(unsigned int* plain)
{
```

```
    int ret = 0;
    //不用移位
    plain[1] = (plain[1] >> 24 | plain[1] << 8);//循环左移一个明
      文
    plain[2] = (plain[2] >> 16 | plain[2] << 16);//循环左移两个
    plain[3] = (plain[3] >> 8 | plain[3] << 24);//循环左移三个
    return ret;
}
int ReRow_shift(unsigned int* cipher)
{
    int ret = 0;
    cipher[1] = (cipher[1] >> 8 | cipher[1] << 24);
    cipher[2] = (cipher[2] >> 16 | cipher[2] << 16);
    cipher[3] = (cipher[3] >> 24 | cipher[3] << 8);
    return ret;
}
char mul(unsigned char arr, unsigned char plain)
{
    unsigned char result = 0;
    while (arr)
    {
        if (arr & 0x01)
        {
            result ^= plain;
        }
        arr = arr >> 1;
        if (plain & 0x80)
        {
            plain = plain << 1;
            plain ^= 0x1B;
        }
        else
```

```
        plain = plain << 1;
    }
    return result;
}
int column_mix(unsigned char(*plain)[4])
{
    int ret = 0;
    unsigned char temp[4][4];
    memcpy(temp, plain, 16);
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            plain[i][j] = mul(mixarr[i][0], temp[0][j]) ^ mul(mixarr[i][1], temp[1][j]) ^ mul(mixarr[i][2], temp[2][j]) ^ mul(mixarr[i][3], temp[3][j]);
        }
    }
    return ret;
}
int keyarr_S(unsigned char(*key)[44], unsigned int col)
{
    int ret = 0;
    for (int i = 0; i < 4; i++)
    {
        key[i][col] = S_Table[key[i][col] >> 4][key[i][col] & 0x0F];
    }
    return ret;
}
int g_f(unsigned char(*key)[44], unsigned int col)
{
```

```
    int ret = 0;
    for (int i = 0; i < 4; i++)//先放到下一列，且行移位
    {
        key[i][col] = key[(i + 1) % 4][col - 1];
    }
    keyarr_S(key, col);//S盒置换
    key[0][col] = key[0][col] ^ Rcon[col / 4];//异或
    return ret;
}
int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key)[44])
{
    int ret = 0;
    for (int i = 0; i < 16; i++)//主密钥赋值
    {
        key[i & 0x03][i >> 2] = pkey[i & 0x03][i >> 2];
    }

    for (int i = 1; i < 11; i++)//10轮
    {
        g_f(key, 4 * i);//G函数
        for (int k = 0; k < 4; k++)//每一轮的第一列得到
        {
            key[k][4 * i] = key[k][4 * i] ^ key[k][4 * (i - 1)];
        }
        for (int j = 1; j < 4; j++)//后三列
        {
            for (int k = 0; k < 4; k++)
            {
                key[k][4 * i + j] = key[k][4 * (i - 1) + j] ^ key[k][4 * i + j - 1];
            }
```

```
        }
    }
    return ret;
}
int and_round(unsigned char(*plain)[4], unsigned char(*key)[44], unsigned int n)
{
    int ret = 0;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            plain[i][j] ^= key[i][n + j];
        }
    }
    return ret;
}
```

7.2 任务二完整代码

7.2.1 aes.cpp

```
    //AES-128bit
#include <iostream>
#include <stdlib.h>
#include <ctime>
using namespace std;
const unsigned char S_Table[16][16] =
{
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
};//S盒
const unsigned char ReS_Table[16][16] =
```

```
{
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, 0
```

```
    x14, 0x63, 0x55, 0x21, 0x0C, 0x7D
};//逆S盒
const unsigned char mixarr[4][4] =
{
0x02, 0x03, 0x01, 0x01,
0x01, 0x02, 0x03, 0x01,
0x01, 0x01, 0x02, 0x03,
0x03, 0x01, 0x01, 0x02
};//列混合的矩阵
const unsigned int Rcon[11] = { 0x00, 0x01, 0x02, 0x04, 0x08, 0
    x10, 0x20, 0x40, 0x80, 0x1B, 0x36 };//G函数中异或轮密钥
int S_ps(unsigned char*);
int S_cs(unsigned char*);
int Row_shift(unsigned int*);
int ReRow_shift(unsigned int*);
int column_mix(unsigned char(*plain)[4]);
char mul(unsigned char, unsigned char);
int keyarr_S(unsigned char(*key)[44], unsigned int col);
int g_f(unsigned char(*key)[44], unsigned int col);
int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key
    )[44]);
int and_round(unsigned char(*plain)[4], unsigned char(*key)
    [44], unsigned int n);
int main()
{
  clock_t start, end;

  unsigned char str[13] = "202100460079";
  unsigned char pkey[4][4];
  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < 4; j++)
```

```
    {
        pkey[i][j] = str[i * 4 + j];
    }
}
for (int i = 0; i < 4; i++)
{
    pkey[3][i] = 0x0;
}

start = clock();
unsigned char key[4][44];
unsigned char message[16];
Extendkey(pkey, key);
for (int n = 0; n < 100; n++)//100组
{
    unsigned char mes[4096];//256个128bit的明文
    for (int m = 0; m < 256; m++)
    {
        mes[m * 16] = char(m);//将256个明文的每个第0字节遍历
    }
    for (int r = 1; r < 4096 && r % 16 != 0; r++)
    {
        srand((unsigned int)(time(NULL)));
        mes[r] = char(rand() % 256);//其他的随机
    }

    for (int p = 0; p < 256; p++)
    {
        for (int q = 0; q < 16; q++)
        {
            message[q] = mes[p * 16 + q];//每128bit传一次
        }
```

```
    unsigned char plain[4][4];
    unsigned char plain2[16];
    unsigned int plain3[16];
    for (int i = 0; i < 12; i++)
    {
      plain[i / 4][i % 4] = message[i];
    }
    for (int i = 0; i < 4; i++)
    {
      plain[3][i] = 0x0;
    }
    and_round(plain, key, 0);
    for (int i = 0; i < 10; i++)
    {
      for (int j = 0; j < 16; j++)
      {
        plain2[j] = plain[j / 4][j % 4];
      }
      S_ps(plain2);
      for (int j = 0; j < 16; j++)
      {
        plain3[j] = int(plain2[j]);
      }
      Row_shift(plain3);
      for (int j = 0; j < 16; j++)
      {
        plain[j / 4][j % 4] = char(plain3[j]);
      }
      column_mix(plain);
      and_round(plain, key, 4 * (i + 1));
    }
    //cout << "密文为: ";
```

```
        /*for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                cout << hex << int(plain[i][j]);
            }
        }*/

    }


  }
  end = clock();
  double time1 = double(end - start) / 1000;
  cout << "所用时间为: " << time1 << "秒";

  return 0;
}
int S_ps(unsigned char* plain)//S盒代换
{
  int ret = 0;
  for (int i = 0; i < 16; i++)
  {
    plain[i] = S_Table[plain[i] >> 4][plain[i] & 0x0F];
  }
  return ret;
}
int S_cs(unsigned char* cipher)//逆S盒代换
{
  int ret = 0;
  for (int i = 0; i < 16; i++)
  {
```

```
    cipher[i] = ReS_Table[cipher[i] >> 4][cipher[i] & 0x0F];
  }
  return ret;
}
int Row_shift(unsigned int* plain)
{
  int ret = 0;
  plain[1] = (plain[1] >> 24 | plain[1] << 8);
  plain[2] = (plain[2] >> 16 | plain[2] << 16);
  plain[3] = (plain[3] >> 8 | plain[3] << 24);
  return ret;
}
int ReRow_shift(unsigned int* cipher)
{
  int ret = 0;
  cipher[1] = (cipher[1] >> 8 | cipher[1] << 24);
  cipher[2] = (cipher[2] >> 16 | cipher[2] << 16);
  cipher[3] = (cipher[3] >> 24 | cipher[3] << 8);
  return ret;
}
char mul(unsigned char arr, unsigned char plain)
{
  unsigned char result = 0;
  while (arr)
  {
    if (arr & 0x01)
    {
      result ^= plain;
    }
    arr = arr >> 1;
    if (plain & 0x80)
    {
```

```
            plain = plain << 1;
            plain ^= 0x1B;
        }
        else
            plain = plain << 1;
    }
    return result;
}
int column_mix(unsigned char(*plain)[4])
{
    int ret = 0;
    unsigned char temp[4][4];
    memcpy(temp, plain, 16);
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            plain[i][j] = mul(mixarr[i][0], temp[0][j]) ^ mul(mixarr[
    i][1], temp[1][j]) ^ mul(mixarr[i][2], temp[2][j]) ^ mul(
    mixarr[i][3], temp[3][j]);
        }
    }
    return ret;
}
int keyarr_S(unsigned char(*key)[44], unsigned int col)
{
    int ret = 0;
    for (int i = 0; i < 4; i++)
    {
        key[i][col] = S_Table[key[i][col] >> 4][key[i][col] & 0x0F
    ];
    }
```

```
    return ret;
}
int g_f(unsigned char(*key)[44], unsigned int col)
{
    int ret = 0;
    for (int i = 0; i < 4; i++)//先放到下一列，且行移位
    {
        key[i][col] = key[(i + 1) % 4][col - 1];
    }
    keyarr_S(key, col);//S盒置换
    key[0][col] = key[0][col] ^ Rcon[col / 4];//异或
    return ret;
}
int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key
    )[44])
{
    int ret = 0;
    for (int i = 0; i < 16; i++)//主密钥赋值
    {
        key[i & 0x03][i >> 2] = pkey[i & 0x03][i >> 2];
    }

    for (int i = 1; i < 11; i++)//10轮
    {
        g_f(key, 4 * i);//G函数
        for (int k = 0; k < 4; k++)//每一轮的第一列得到
        {
            key[k][4 * i] = key[k][4 * i] ^ key[k][4 * (i - 1)];
        }
        for (int j = 1; j < 4; j++)//后三列
        {
            for (int k = 0; k < 4; k++)
```

```
        {
          key[k][4 * i + j] = key[k][4 * (i - 1) + j] ^ key[k][4
   * i + j - 1];
        }
      }
    }
    return ret;
}
int and_round(unsigned char(*plain)[4], unsigned char(*key)
   [44], unsigned int n)
{
    int ret = 0;
    for (int i = 0; i < 4; i++)
    {
      for (int j = 0; j < 4; j++)
      {
        plain[i][j] ^= key[i][n + j];
      }
    }
    return ret;
}
```

7.2.2 mbedtls-aes-test.c

```
    /**
 * @brief    AES Function demo
 * @author   mculover666
 * @date     2020/09/23
*/

#if !defined(MBEDTLS_CONFIG_FILE)
```

```
//#include "mbedtls/config.h"
#else
#include MBEDTLS_CONFIG_FILE
#endif

//#if defined(MBEDTLS_CIPHER_C)

#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include "mbedtls/cipher.h"

/* Private Key *///采用不同加密算法时，注意修改密钥及分组长度。
//
//uint8_t key[8] = {
//    0x06, 0xa9, 0x21, 0x40, 0x36, 0xb8, 0xa1, 0x5b
//    /*0x51, 0x2e, 0x03, 0xd5, 0x34, 0x12, 0x00, 0x06*/
//};
uint8_t key[16] = {
    0x32, 0x30, 0x32, 0x31, 0x30, 0x30, 0x34, 0x36,
    0x30, 0x30, 0x37, 0x39, 0x0, 0x0, 0x0, 0x0
};

/* Intialization Vector */
//uint8_t iv[8] = {
//    0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30
//    //0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41
//};
uint8_t iv[16] = {
    0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30,
    0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41
```

```c
};

static void dump_buf(uint8_t* buf, uint32_t len)
{
    int i;

    printf("buf:");

    for (i = 0; i < len; i++) {
        printf("%s%02X%s", i % 16 == 0 ? "\r\n\t" : " ",
            buf[i],
            i == len - 1 ? "\r\n" : "");
    }
}

int aes_test(mbedtls_cipher_type_t cipher_type,uint8_t temp[16]) //可选对称密码算法
{
    int ret;
    size_t len;
    int olen = 0;
    uint8_t output_buf[64];
    uint8_t o_buf[64];
    const char *input = temp;
    //const char input[] = "202100460079";
    //const char input[] = "HelloWorld123456";
    //const char input[] = {'m', 'c', 'u', 'l', 'o', 'v', 'e', 'r', '6', '6', '6', ' ', 'i', 's'};


    mbedtls_cipher_context_t ctx;
    const mbedtls_cipher_info_t* info;
```

```

    /* 1. init cipher structuer */
    mbedtls_cipher_init(&ctx);

    /* 2. get info structuer from type */
    info = mbedtls_cipher_info_from_type(cipher_type);

    /* 3. setup cipher structuer */
    ret = mbedtls_cipher_setup(&ctx, info);
    if (ret != 0) {
        goto exit;
    }

    ///* 4. set key */
   /* ret = mbedtls_cipher_setkey(&ctx, key, sizeof(key) * 8, MBEDTLS_ENCRYPT);
    if (ret != 0) {
        goto exit;
    }*/
    ret = mbedtls_cipher_setkey(&ctx, key, 128, MBEDTLS_ENCRYPT);
    if (ret != 0) {
        goto exit;
    }

    /* 5. set iv */
    ret = mbedtls_cipher_set_iv(&ctx, iv, sizeof(iv));
    if (ret != 0) {
        goto exit;
    }

    /* 6. update cipher *///采用ECB模式时，无自动填充，此处输
```

```
入长度只能为分组的整数倍
    ret = mbedtls_cipher_update(&ctx, (unsigned char*)input, 16, output_buf, &len);
    if (ret != 0) {
        goto exit;
    }
    olen += len;

    /* 6. update cipher */
/*    ret = mbedtls_cipher_update(&ctx, (unsigned char*)input, strlen(input), output_buf, &len);
    if (ret != 0) {
        goto exit;
    }
    olen += len;*/



    /* 7. finish cipher */
    ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
    if (ret != 0) {
        goto exit;
    }
    olen += len;

    /* show */
//  printf("\r\nsource_context: %s\r\n", input);
    /*dump_buf((uint8_t*)input, strlen(input));
    printf("cipher name: %s block size is: %d\r\n", mbedtls_cipher_get_name(&ctx), mbedtls_cipher_get_block_size(&ctx));
    dump_buf(output_buf, olen);*/
```

```
exit:
    /* 8. free cipher structure */
    mbedtls_cipher_free(&ctx);






//     /* 1. init cipher structuer */
//     mbedtls_cipher_init(&ctx);
//
//     /* 2. get info structuer from type */
//     info = mbedtls_cipher_info_from_type(cipher_type);
//
//     /* 3. setup cipher structuer */
//     ret = mbedtls_cipher_setup(&ctx, info);
//     if (ret != 0) {
//         goto exit1;
//     }
//
//     /* 4. set key */
//     /*ret = mbedtls_cipher_setkey(&ctx, key, sizeof(key) * 8,
    MBEDTLS_DECRYPT);
//     if (ret != 0) {
//         goto exit1;
//     }*/
//     ret = mbedtls_cipher_setkey(&ctx, key, 128,
    MBEDTLS_DECRYPT);
//     if (ret != 0) {
//         goto exit1;
//     }
```

```
//
//     /* 5. set iv */
//     ret = mbedtls_cipher_set_iv(&ctx, iv, sizeof(iv));
//     if (ret != 0) {
//         goto exit1;
//     }
//
//     ///* 6. update cipher *///  采用ECB模式时，无自动填充，此处输入长度只能为分组的整数倍
//     ret = mbedtls_cipher_update(&ctx, (unsigned char*)output_buf, 16, o_buf, &len);
//     if (ret != 0) {
//         goto exit1;
//     }
//     olen += len;
//
//     /* 6. update cipher */
//     /*ret = mbedtls_cipher_update(&ctx, (unsigned char*)input, strlen(input), output_buf, &len);
//     if (ret != 0) {
//         goto exit1;
//     }
//     olen += len;*/
//
//
//
//     /* 7. finish cipher */
//     ret = mbedtls_cipher_finish(&ctx, o_buf, &len);
//     if (ret != 0) {
//         goto exit1;
//     }
//     olen += len;
```

```
//
//      /* show */
//      printf("\r\nsource_context: %s\r\n", output_buf);
//      dump_buf((uint8_t*)output_buf, strlen(output_buf));
//      printf("cipher name: %s block size is: %d\r\n", mbedtls_cipher_get_name(&ctx), mbedtls_cipher_get_block_size(&ctx));
//      dump_buf(o_buf, olen);
//exit1:
//      /* 8. free cipher structure */
//      mbedtls_cipher_free(&ctx);
    return ret;
}

//int main()
//{
//      aes_test(MBEDTLS_CIPHER_AES_128_ECB);
//      return 1;
//}

//#endif /* MBEDTLS_CIPHER_C */
```

7.2.3 main.c

```
    #include "mbedtls/cipher.h"
#include "mbedtls/dhm.h"
#include "mbedtls/md.h"
#include "stdlib.h"
#include "time.h"
int main()
{
```

```
    clock_t start, end;
    start = clock();
    uint8_t temp[17] = { 0 };
    double time1;
    for (int i = 0; i < 100; i++)//100组
    {
        for (int j = 0; j < 256; j++)//256个128bit明文
        {
            temp[0] = j + 48;//第0字节遍历256种
            temp[16] = '\0';
            for (int k = 1; k < 16; k++)
            {
                srand((unsigned)time(NULL));
                temp[k] = rand() % 256;//其他15字节随机
            }
            aes_test(MBEDTLS_CIPHER_AES_128_ECB,temp); /*无填
充， 修改输入长度为分组长度倍数 */
            //传入函数
        }


    }
    end = clock();
    time1 = (double)(end - start)/ CLOCKS_PER_SEC;
    printf("所用时间为：%f秒\n", time1);
  //aes_test(MBEDTLS_CIPHER_DES_CBC);

    //mbedtls_rsa_test();
    //mbedtls_rsa_sign_test();
    //mbedtls_dhm_test();
    //mbedtls_shax_test(MBEDTLS_MD_SHA512);
    //aes_test(MBEDTLS_CIPHER_ARIA_128_ECB);
```

```
        return 1;
}
```