

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



山东大学
SHANDONG UNIVERSITY

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	4
2	实验原理	4
3	实验一过程	6
3.1	格式处理	6
3.2	轮密钥扩展	7
3.3	字节代换	9
3.4	行位移	10
3.5	列混合	11
3.6	轮密钥异或	12
4	实验二过程	12
4.1	实现所要求的明文集合	12
4.2	时间比较	13
5	实验结论	14
5.1	实验一	14
5.2	实验二	15
6	实验总结分析	16
7	附录	17
7.1	任务一完整代码	17
7.2	任务二完整代码	26
7.2.1	aes.cpp	26
7.2.2	mbdtdls-aes-test.c	36
7.2.3	main.c	43

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。

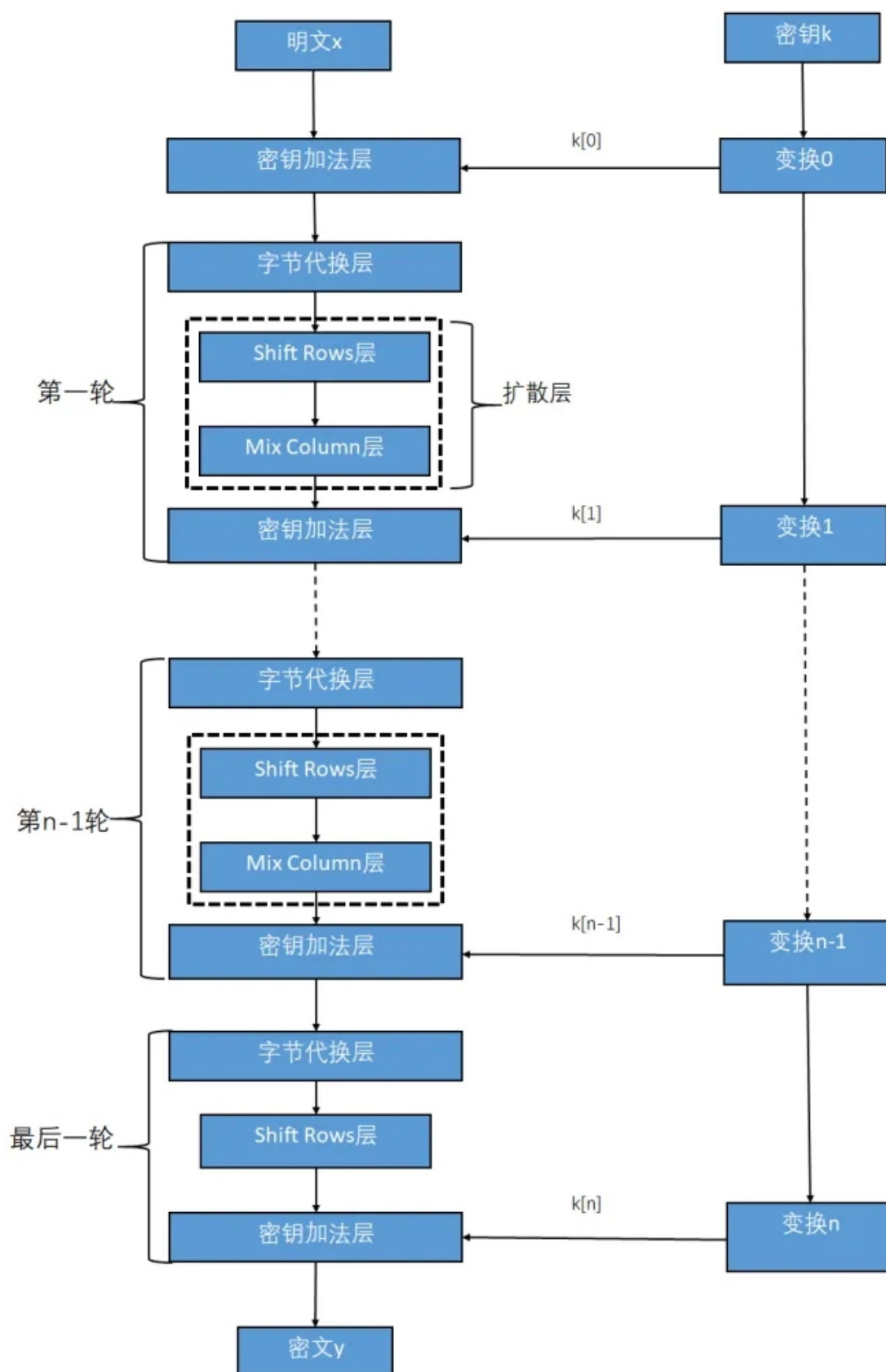
任务二：对同一个密钥，分别加密 100 组满足以下结构的 256 个明文组成的集合（提前生成好），估计算法加密一次的运行时间，并与直接调用算法库进行加密的运行时间进行比较。其中 256 个明文满足：第 0 字节遍历 256 种可能，其余字节取任意常数。

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。

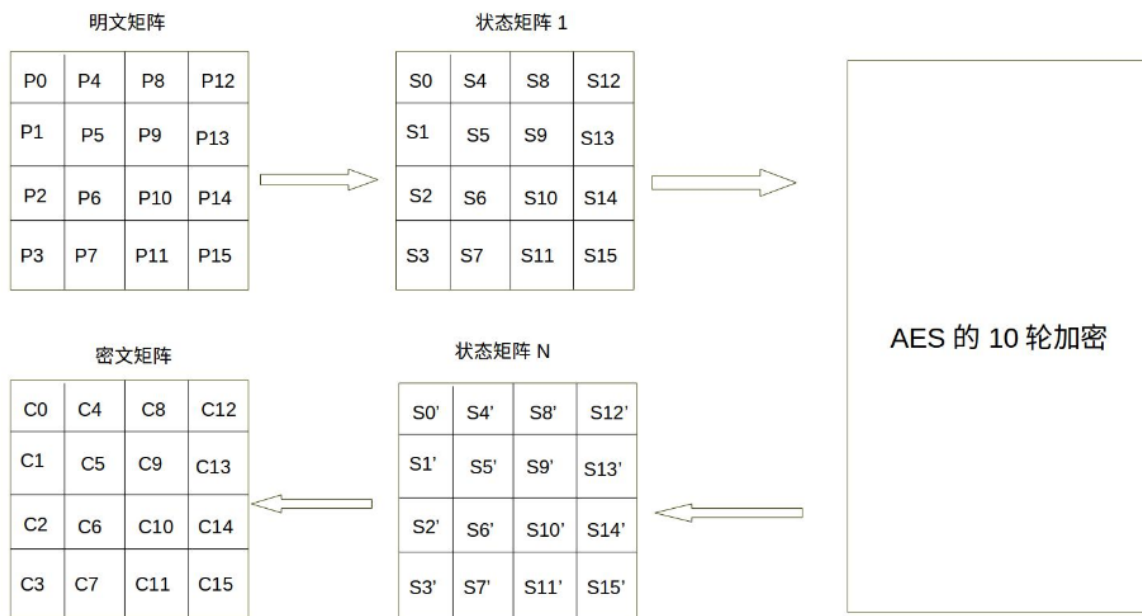
具体加密流程见下图：



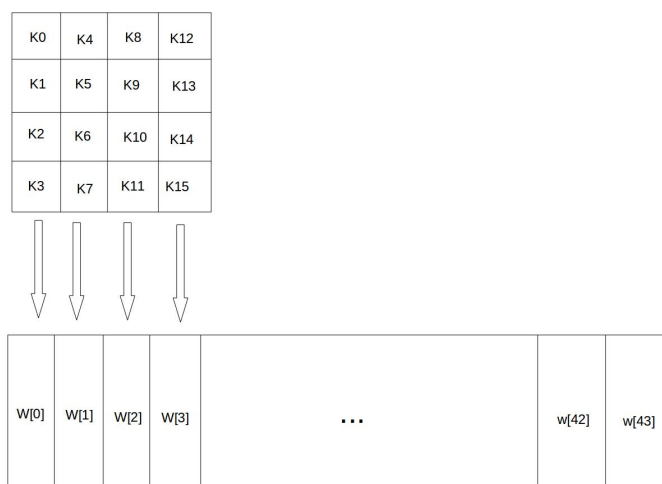
3 实验一过程

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 $P = P_0 P_1 \cdots P_{15}$ 和 $K = K_0 K_1 \cdots K_{15}$ 。如，明文分组为 $P = \text{abcdefghijklmnop}$ ，其中的字符 a 对应 P_0 ， p 对应 P_{15} 。其实我们将明文转化为 `ascii` 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 $W[0], W[1], \cdots, W[43]$ ，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：

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

其中函数 g 由 3 部分组成：字循环、字节代换和轮常量异或

```

1 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
2 int g_f(unsigned char(*key)[44], unsigned int col)
3 {
4     int ret = 0;
5     for (int i = 0; i < 4; i++) //先放到下一列，且行移位
6     {
7         key[i][col] = key[(i + 1) % 4][col - 1];
8     }
9     keyarr_S(key, col); //S盒置换
10    key[0][col] = key[0][col] ^ Rcon[col / 4]; //异或

```

```
11     return ret;
12 }
13     int Extendkey(const unsigned char(*pkey)[4], unsigned char
    (*key)[44])
14 {
15     int ret = 0;
16     for (int i = 0; i < 16; i++)//主密钥赋值
17     {
18         key[i & 0x03][i >> 2] = pkey[i & 0x03][i >> 2];
19     }
20
21     for (int i = 1; i < 11; i++)//10轮
22     {
23         g_f(key, 4 * i);//G函数
24         for (int k = 0; k < 4; k++)//每一轮的第一列得到
25         {
26             key[k][4 * i] = key[k][4 * i] ^ key[k][4 * (i - 1)];
27         }
28         for (int j = 1; j < 4; j++)//后三列
29         {
30             for (int k = 0; k < 4; k++)
31             {
32                 key[k][4 * i + j] = key[k][4 * (i - 1) + j] ^ key[k][4
    * i + j - 1];
33             }
34         }
35     }
36     return ret;
37 }
```

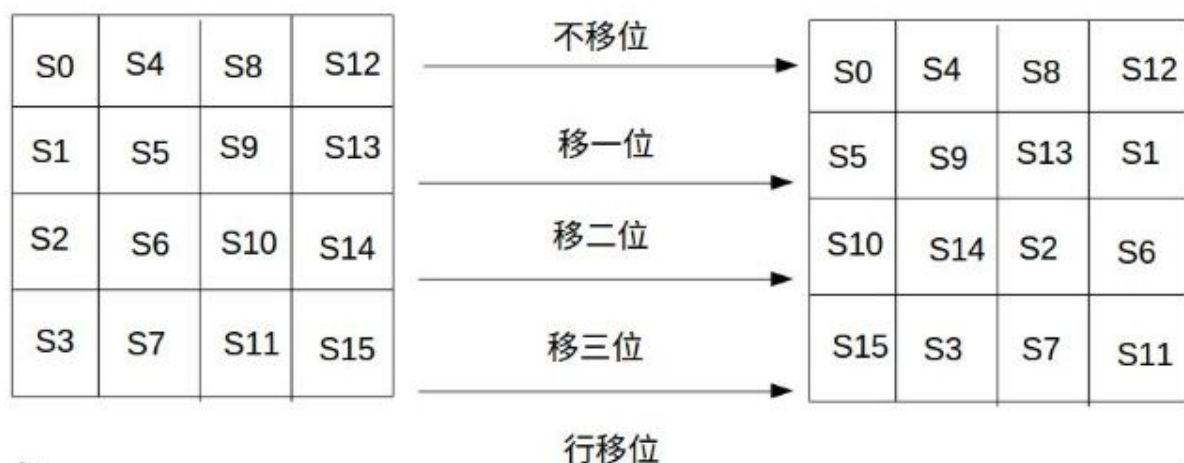

3.3 字节代换

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。

```
1 int S_ps(unsigned char* plain)//S盒代换
2 {
3     int ret = 0;
4     for (int i = 0; i < 16; i++)
5     {
6         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
7     }
8     return ret;
9 }
10 int S_cs(unsigned char* cipher)//逆S盒代换
11 {
12     int ret = 0;
13     for (int i = 0; i < 16; i++)
14     {
15         cipher[i] = ReS_Table[cipher[i] >> 4][cipher[i] & 0x0F];
16     }
17     return ret;
18 }
```

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：



具体代码如下：

```

1 int Row_shift(unsigned int* plain)
2 {
3     int ret = 0;
4     //不用移位
5     plain[1] = (plain[1] >> 24 | plain[1] << 8); //循环左移一个明文
6     plain[2] = (plain[2] >> 16 | plain[2] << 16); //循环左移两个
7     plain[3] = (plain[3] >> 8 | plain[3] << 24); //循环左移三个
8     return ret;
9 }
10 int ReRow_shift(unsigned int* cipher) //解密逆变换
11 {
12     int ret = 0;
13     cipher[1] = (cipher[1] >> 8 | cipher[1] << 24);
14     cipher[2] = (cipher[2] >> 16 | cipher[2] << 16);
15     cipher[3] = (cipher[3] >> 24 | cipher[3] << 8);

```

```

16     return ret;
17 }

```

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：

```

1 int column_mix(unsigned char(*plain)[4])
2 {
3     int ret = 0;
4     unsigned char temp[4][4];
5     memcpy(temp, plain, 16);
6     for (int i = 0; i < 4; i++)
7     {
8         for (int j = 0; j < 4; j++)
9         {
10             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]);
11         }
12     }
13     return ret;
14 }

```

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。

```
1 int and_round(unsigned char(*plain)[4], unsigned char(*key)
   [44], unsigned int n)
2 {
3     int ret = 0;
4     for (int i = 0; i < 4; i++)
5     {
6         for (int j = 0; j < 4; j++)
7         {
8             plain[i][j] ^= key[i][n + j];
9         }
10    }
11    return ret;
12 }
```

4 实验二过程

4.1 实现所要求的明文集合

```
1 for (int n = 0; n < 100; n++)//100组
2 {
3     unsigned char mes[4096]; //256个128bit的明文
4     for (int m = 0; m < 256; m++)
5     {
6         mes[m * 16] = char(m); //将256个明文的每个第0字节遍历
7     }
8     for (int r = 1; r < 4096 && r % 16 != 0; r++)
```

```
9      {
10          srand((unsigned int)(time(NULL)));
11          mes[r] = char(rand() % 256); // 其他的随机
12      }
13
14      for (int p = 0; p < 256; p++)
15      {
16          for (int q = 0; q < 16; q++)
17          {
18              message[q] = mes[p * 16 + q]; // 每128bit传一次
19          }
20          unsigned char plain[4][4];
21          unsigned char plain2[16];
22          unsigned int plain3[16];
23          for (int i = 0; i < 12; i++)
24          {
25              plain[i / 4][i % 4] = message[i];
26          }
27          for (int i = 0; i < 4; i++)
28          {
29              plain[3][i] = 0x0;
30          }
```

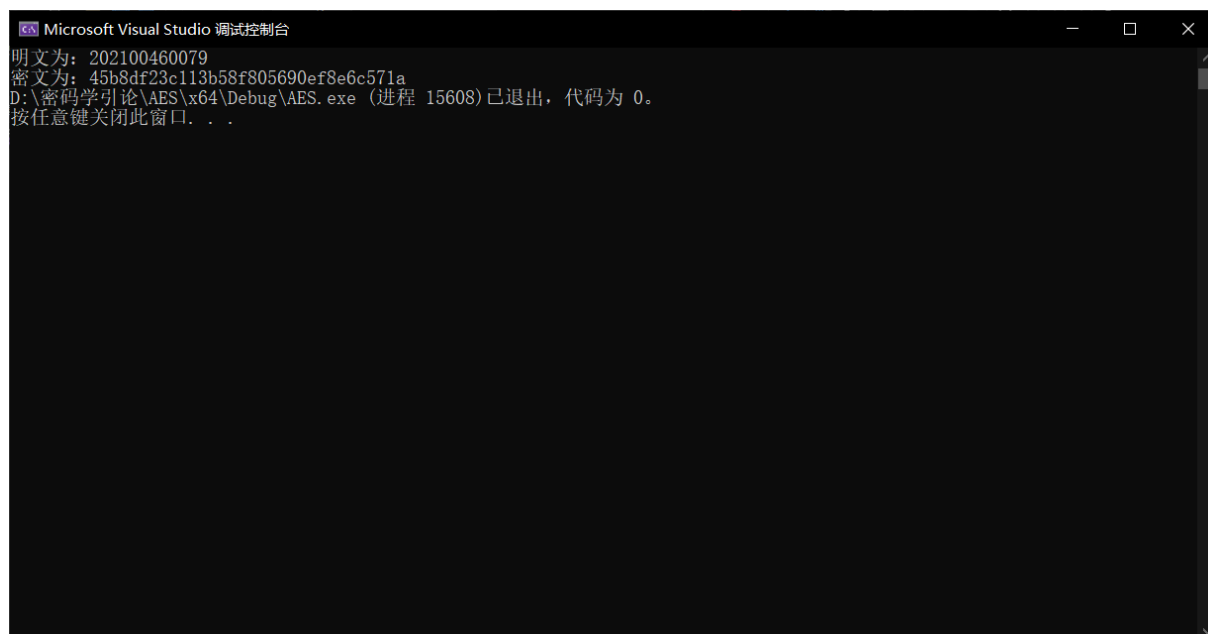
4.2 时间比较

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

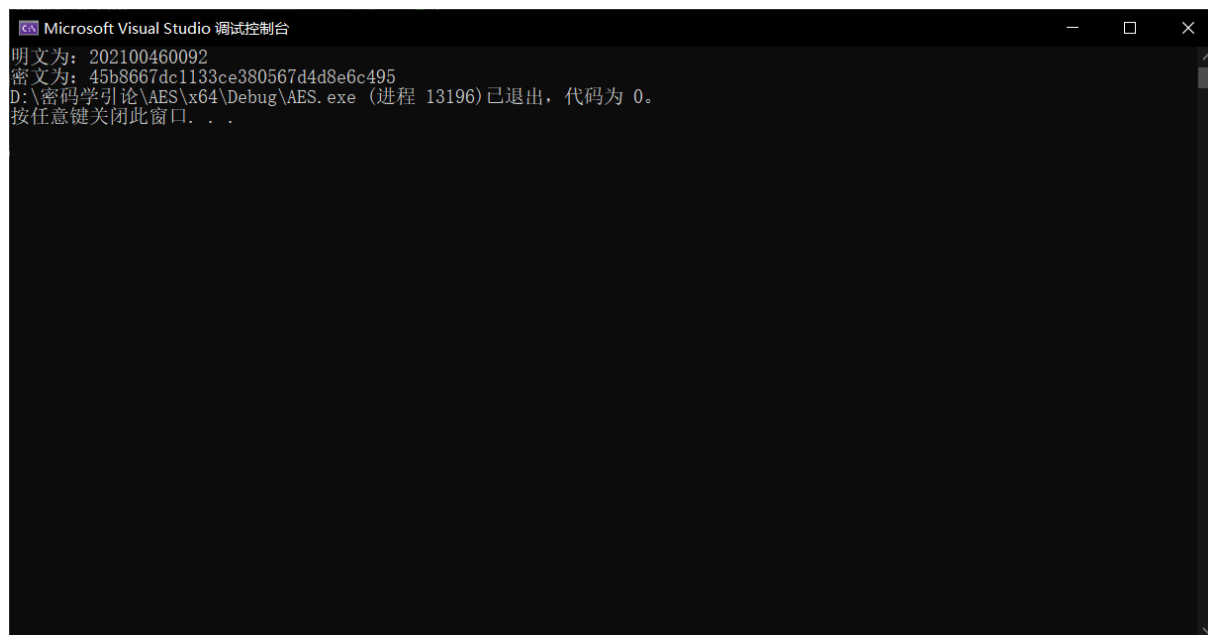
5.1 实验一

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



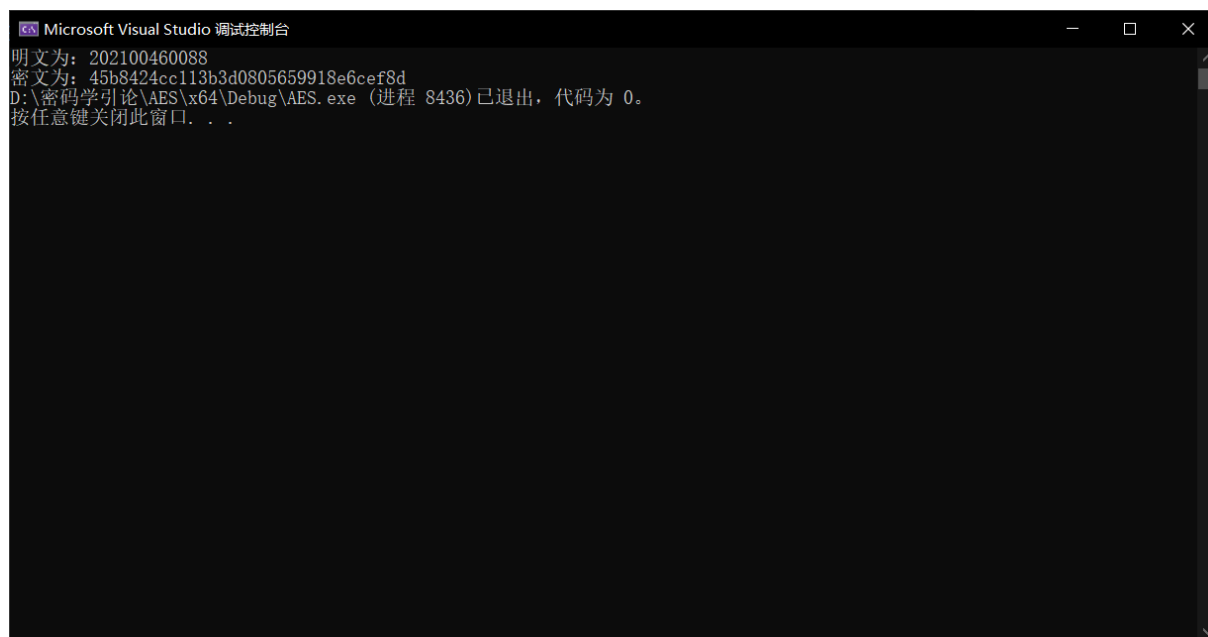
```
Microsoft Visual Studio 调试控制台
明文为: 202100460079
密文为: 45b8df23c113b58f805690ef8e6c571a
D:\密码学引论\AES\x64\Debug\AES.exe (进程 15608) 已退出, 代码为 0。
按任意键关闭此窗口. . .
```

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



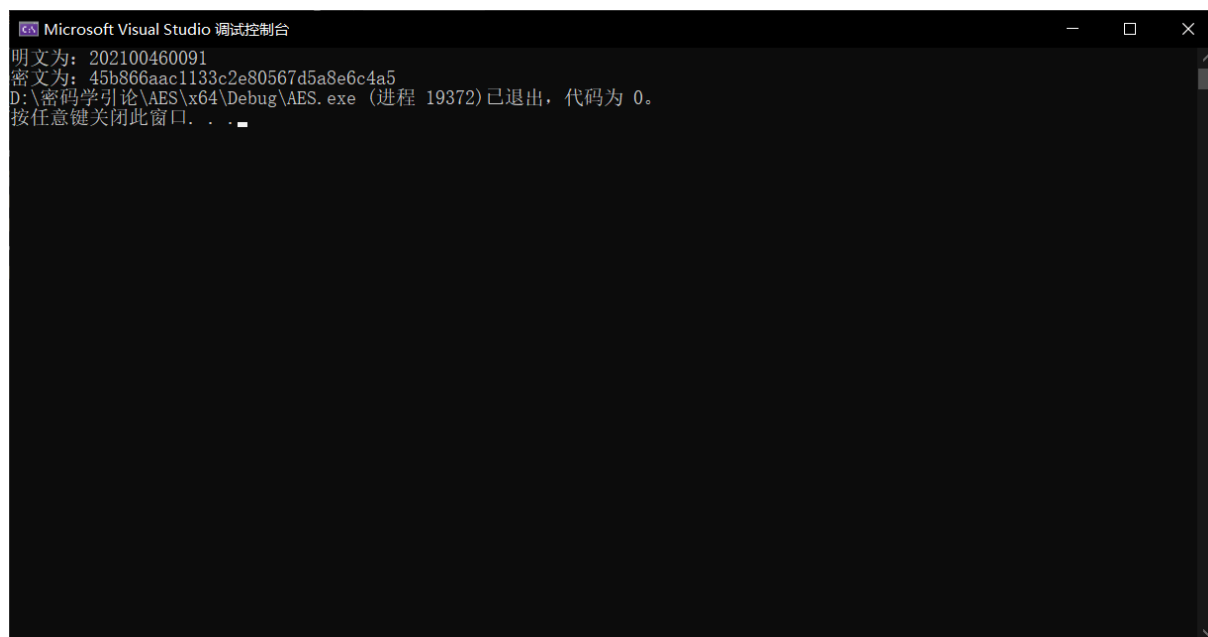
```
Microsoft Visual Studio 调试控制台
明文为: 202100460092
密文为: 45b8667dc1133ce380567d4d8e6c495
D:\密码学引论\AES\x64\Debug\AES.exe (进程 13196) 已退出, 代码为 0。
按任意键关闭此窗口. . .
```

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



```
Microsoft Visual Studio 调试控制台
明文为: 202100460088
密文为: 45b8424cc113b3d0805659918e6cef8d
D:\密码学引论\AES\x64\Debug\AES.exe (进程 8436) 已退出, 代码为 0。
按任意键关闭此窗口. . .
```

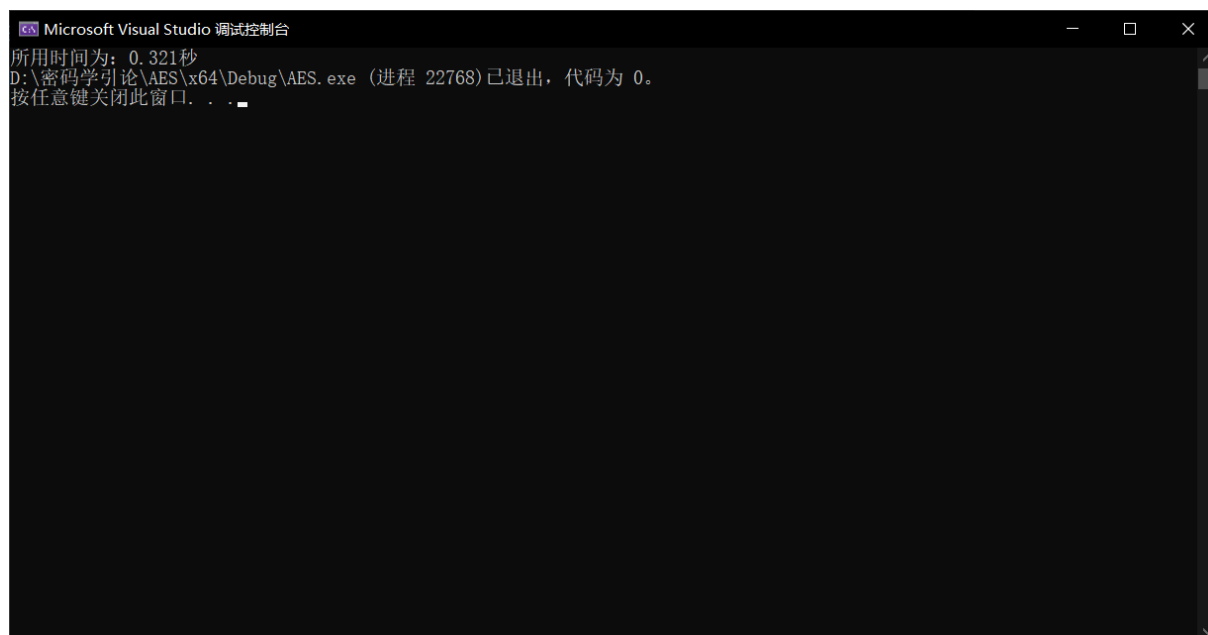
王成盟学号 202100460091, 生成密文: 45b866aac1133c2e80567d5a8e6c4a5



```
Microsoft Visual Studio 调试控制台
明文为: 202100460091
密文为: 45b866aac1133c2e80567d5a8e6c4a5
D:\密码学引论\AES\x64\Debug\AES.exe (进程 19372) 已退出, 代码为 0。
按任意键关闭此窗口. . .
```

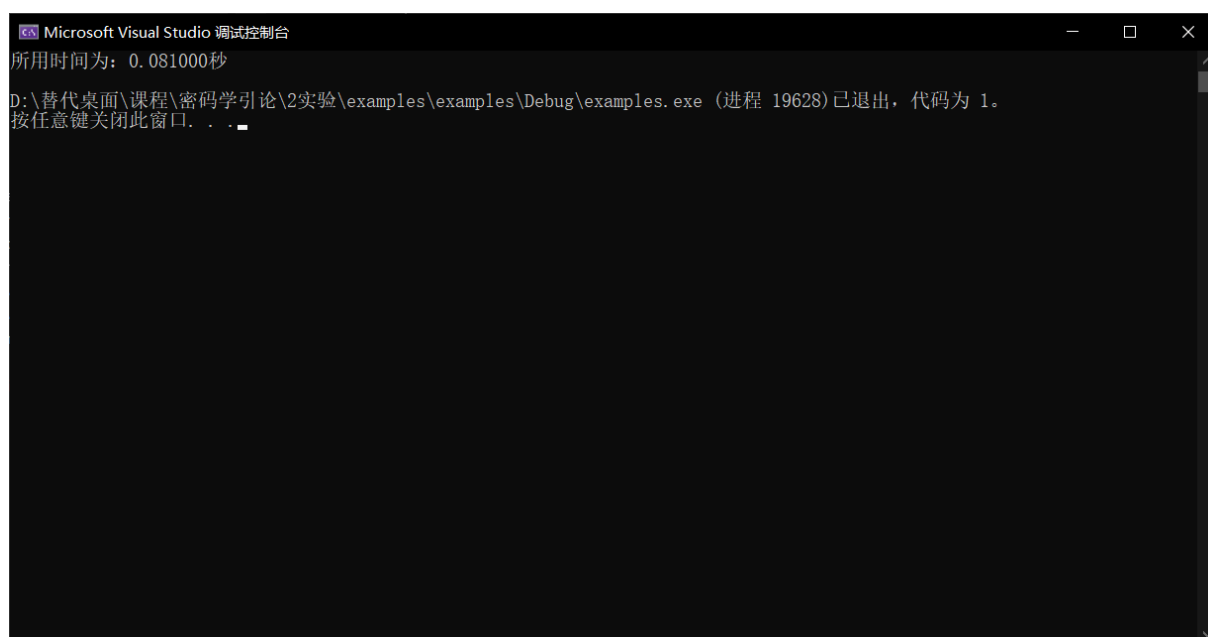
5.2 实验二

自己实习的 aes 加密所要求的明文集合时间:



```
Microsoft Visual Studio 调试控制台
所用时间为: 0.321秒
D:\密码学引论\AES\x64\Debug\AES.exe (进程 22768) 已退出, 代码为 0。
按任意键关闭此窗口. . .
```

库函数 aes 加密所要求的明文集合时间:



```
Microsoft Visual Studio 调试控制台
所用时间为: 0.081000秒
D:\替代桌面\课程\密码学引论\2实验\examples\examples\Debug\examples.exe (进程 19628) 已退出, 代码为 1。
按任意键关闭此窗口. . .
```

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

```
1
2 //AES-128 bit
3 #include <iostream>
4 using namespace std;
5 const unsigned char S_Table[16][16] =
6 {
7 0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0
8   x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
9 0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0
10  xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
11 0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0
12  xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
13 0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0
14  x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
15 0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0
16  xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
17 0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0
18  xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
19 0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0
20  x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
21 0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0
22  xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
```

```

15 0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0
    x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
16 0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0
    xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
17 0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0
    xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
18 0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0
    xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
19 0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0
    x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
20 0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0
    x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
21 0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0
    x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
22 0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0
    x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
23 }; //S盒
24 const unsigned char ReS_Table[16][16] =
25 {
26 0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0
    xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
27 0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0
    x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
28 0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0
    x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
29 0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0
    xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
30 0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0
    x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
31 0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0
    x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
32 0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0

```

```

    x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
33 0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0
    xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
34 0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0
    xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
35 0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0
    x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
36 0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0
    x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
37 0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0
    xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
38 0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0
    x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
39 0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0
    x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
40 0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0
    xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
41 0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0
    x14, 0x63, 0x55, 0x21, 0x0C, 0x7D
42 }; // 逆S盒
43 const unsigned char mixarr[4][4] =
44 {
45 0x02, 0x03, 0x01, 0x01,
46 0x01, 0x02, 0x03, 0x01,
47 0x01, 0x01, 0x02, 0x03,
48 0x03, 0x01, 0x01, 0x02
49 }; // 列混合的矩阵
50 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
51 int S_ps(unsigned char*);
52 int S_cs(unsigned char*);
53 int Row_shift(unsigned int*);

```

```
54 int ReRow_shift(unsigned int*);
55 int column_mix(unsigned char(*plain)[4]);
56 char mul(unsigned char, unsigned char);
57 int keyarr_S(unsigned char(*key)[44], unsigned int col);
58 int g_f(unsigned char(*key)[44], unsigned int col);
59 int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key)
    [44]);
60 int and_round(unsigned char(*plain)[4], unsigned char(*key)
    [44], unsigned int n);
61 int main()
62 {
63     unsigned char pkey[4][4];
64     unsigned char str[13] = "202100460079";
65     for (int i = 0; i < 12; i++)
66     {
67         pkey[i / 4][i % 4] = str[i];
68     }
69     for (int i = 0; i < 4; i++)
70     {
71         pkey[3][i] = 0x0; // 不够的补0
72     }
73
74     unsigned char key[4][44]; // 11轮密钥
75     Extendkey(pkey, key); // 由主密钥生成10轮密钥
76     // for (int m = 0; m < 100; m++)
77     // {
78
79     unsigned char message[13] = "202100460079";
80     unsigned char plain[4][4];
81     unsigned char plain2[16];
82     unsigned int plain3[16];
83     for (int i = 0; i < 12; i++)
```

```
84 {
85     plain[i / 4][i % 4] = message[i];
86 }
87 for (int i = 0; i < 4; i++)
88 {
89     plain[3][i] = 0x0; // 不够的补0
90 }
91 and_round(plain, key, 0); // 明文与密钥异或
92 for (int i = 0; i < 10; i++) // 10轮
93 {
94     for (int j = 0; j < 16; j++)
95     {
96         plain2[j] = plain[j / 4][j % 4];
97     }
98     S_ps(plain2); // S盒置换
99     for (int j = 0; j < 16; j++)
100     {
101         plain3[j] = int(plain2[j]);
102     }
103     Row_shift(plain3); // 行移位
104     for (int j = 0; j < 16; j++)
105     {
106         plain[j / 4][j % 4] = char(plain3[j]);
107     }
108     column_mix(plain); // 列混合
109     and_round(plain, key, 4 * (i + 1)); // 异或运算
110 }
111 cout << "密文为: ";
112 for (int i = 0; i < 4; i++)
113 {
114     for (int j = 0; j < 4; j++)
115     {
```

```
116     cout << hex << int(plain[i][j]);
117 }
118 }
119
120
121
122 //}
123
124
125 return 0;
126 }
127 int S_ps(unsigned char* plain)//S盒代换
128 {
129     int ret = 0;
130     for (int i = 0; i < 16; i++)
131     {
132         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
133     }
134     return ret;
135 }
136 int S_cs(unsigned char* cipher)//逆S盒代换
137 {
138     int ret = 0;
139     for (int i = 0; i < 16; i++)
140     {
141         cipher[i] = ReS_Table[cipher[i] >> 4][cipher[i] & 0x0F];
142     }
143     return ret;
144 }
145 int Row_shift(unsigned int* plain)
146 {
```

```
147  int ret = 0;
148  //不用移位
149  plain[1] = (plain[1] >> 24 | plain[1] << 8); //循环左移一个明文
150  plain[2] = (plain[2] >> 16 | plain[2] << 16); //循环左移两个
151  plain[3] = (plain[3] >> 8 | plain[3] << 24); //循环左移三个
152  return ret;
153 }
154 int ReRow_shift(unsigned int* cipher)
155 {
156     int ret = 0;
157     cipher[1] = (cipher[1] >> 8 | cipher[1] << 24);
158     cipher[2] = (cipher[2] >> 16 | cipher[2] << 16);
159     cipher[3] = (cipher[3] >> 24 | cipher[3] << 8);
160     return ret;
161 }
162 char mul(unsigned char arr, unsigned char plain)
163 {
164     unsigned char result = 0;
165     while (arr)
166     {
167         if (arr & 0x01)
168         {
169             result ^= plain;
170         }
171         arr = arr >> 1;
172         if (plain & 0x80)
173         {
174             plain = plain << 1;
175             plain ^= 0x1B;
176         }
177         else
```

```
178     plain = plain << 1;
179 }
180 return result;
181 }
182 int column_mix(unsigned char(*plain)[4])
183 {
184     int ret = 0;
185     unsigned char temp[4][4];
186     memcpy(temp, plain, 16);
187     for (int i = 0; i < 4; i++)
188     {
189         for (int j = 0; j < 4; j++)
190         {
191             plain[i][j] = mul(mixarr[i][0], temp[0][j]) ^ mul(mixarr[
192             i][1], temp[1][j]) ^ mul(mixarr[i][2], temp[2][j]) ^ mul(
193             mixarr[i][3], temp[3][j]);
194         }
195     }
196     return ret;
197 }
198 int keyarr_S(unsigned char(*key)[44], unsigned int col)
199 {
200     int ret = 0;
201     for (int i = 0; i < 4; i++)
202     {
203         key[i][col] = S_Table[key[i][col] >> 4][key[i][col] & 0x0F
204         ];
205     }
206     return ret;
207 }
208 int g_f(unsigned char(*key)[44], unsigned int col)
209 {
210     int ret = 0;
211     for (int i = 0; i < 4; i++)
212     {
213         key[i][col] = S_Table[key[i][col] >> 4][key[i][col] & 0x0F
214         ];
215     }
216     return ret;
217 }
```



```
207 int ret = 0;
208 for (int i = 0; i < 4; i++)//先放到下一列，且行移位
209 {
210     key[i][col] = key[(i + 1) % 4][col - 1];
211 }
212 keyarr_S(key, col);//S盒置换
213 key[0][col] = key[0][col] ^ Rcon[col / 4];//异或
214 return ret;
215 }
216 int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key
    )[44])
217 {
218     int ret = 0;
219     for (int i = 0; i < 16; i++)//主密钥赋值
220     {
221         key[i & 0x03][i >> 2] = pkey[i & 0x03][i >> 2];
222     }
223
224     for (int i = 1; i < 11; i++)//10轮
225     {
226         g_f(key, 4 * i);//G函数
227         for (int k = 0; k < 4; k++)//每一轮的第一列得到
228         {
229             key[k][4 * i] = key[k][4 * i] ^ key[k][4 * (i - 1)];
230         }
231         for (int j = 1; j < 4; j++)//后三列
232         {
233             for (int k = 0; k < 4; k++)
234             {
235                 key[k][4 * i + j] = key[k][4 * (i - 1) + j] ^ key[k][4
                    * i + j - 1];
236             }

```

```
237     }
238 }
239 return ret;
240 }
241 int and_round(unsigned char(*plain)[4], unsigned char(*key)
    [44], unsigned int n)
242 {
243     int ret = 0;
244     for (int i = 0; i < 4; i++)
245     {
246         for (int j = 0; j < 4; j++)
247         {
248             plain[i][j] ^= key[i][n + j];
249         }
250     }
251     return ret;
252 }
```

7.2 任务二完整代码

7.2.1 aes.cpp

```
1 //AES-128 bit
2 #include <iostream>
3 #include <stdlib.h>
4 #include <ctime>
5 using namespace std;
6 const unsigned char S_Table[16][16] =
7 {
8 0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0
    x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
```

```

9 0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0
    xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
10 0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0
    xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
11 0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0
    x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
12 0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0
    xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
13 0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0
    xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
14 0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0
    x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
15 0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0
    xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
16 0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0
    x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
17 0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0
    xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
18 0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0
    xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
19 0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0
    xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
20 0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0
    x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
21 0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0
    x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
22 0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0
    x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
23 0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0
    x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
24 }; //S盒
25 const unsigned char ReS_Table[16][16] =

```

```

26 {
27 0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0
    xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
28 0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0
    x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
29 0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0
    x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
30 0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0
    xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
31 0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0
    x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
32 0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0
    x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
33 0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0
    x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
34 0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0
    xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
35 0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0
    xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
36 0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0
    x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
37 0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0
    x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
38 0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0
    xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
39 0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0
    x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
40 0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0
    x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
41 0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0
    xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
42 0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0

```

```

    x14, 0x63, 0x55, 0x21, 0x0C, 0x7D
43 }; // 逆S盒
44 const unsigned char mixarr[4][4] =
45 {
46 0x02, 0x03, 0x01, 0x01,
47 0x01, 0x02, 0x03, 0x01,
48 0x01, 0x01, 0x02, 0x03,
49 0x03, 0x01, 0x01, 0x02
50 }; // 列混合的矩阵
51 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
52 int S_ps(unsigned char*);
53 int S_cs(unsigned char*);
54 int Row_shift(unsigned int*);
55 int ReRow_shift(unsigned int*);
56 int column_mix(unsigned char(*plain)[4]);
57 char mul(unsigned char, unsigned char);
58 int keyarr_S(unsigned char(*key)[44], unsigned int col);
59 int g_f(unsigned char(*key)[44], unsigned int col);
60 int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key
    )[44]);
61 int and_round(unsigned char(*plain)[4], unsigned char(*key)
    [44], unsigned int n);
62 int main()
63 {
64     clock_t start, end;
65
66     unsigned char str[13] = "202100460079";
67     unsigned char pkey[4][4];
68     for (int i = 0; i < 3; i++)
69     {
70         for (int j = 0; j < 4; j++)

```

```
71     {
72         pkey[i][j] = str[i * 4 + j];
73     }
74 }
75 for (int i = 0; i < 4; i++)
76 {
77     pkey[3][i] = 0x0;
78 }
79
80 start = clock();
81 unsigned char key[4][44];
82 unsigned char message[16];
83 Extendkey(pkey, key);
84 for (int n = 0; n < 100; n++)//100组
85 {
86     unsigned char mes[4096];//256个128bit的明文
87     for (int m = 0; m < 256; m++)
88     {
89         mes[m * 16] = char(m);//将256个明文的每个第0字节遍历
90     }
91     for (int r = 1; r < 4096 && r % 16 != 0; r++)
92     {
93         srand((unsigned int)(time(NULL)));
94         mes[r] = char(rand() % 256);//其他的随机
95     }
96
97     for (int p = 0; p < 256; p++)
98     {
99         for (int q = 0; q < 16; q++)
100         {
101             message[q] = mes[p * 16 + q];//每128bit传一次
102         }
```

```
103     unsigned char plain[4][4];
104     unsigned char plain2[16];
105     unsigned int plain3[16];
106     for (int i = 0; i < 12; i++)
107     {
108         plain[i / 4][i % 4] = message[i];
109     }
110     for (int i = 0; i < 4; i++)
111     {
112         plain[3][i] = 0x0;
113     }
114     and_round(plain, key, 0);
115     for (int i = 0; i < 10; i++)
116     {
117         for (int j = 0; j < 16; j++)
118         {
119             plain2[j] = plain[j / 4][j % 4];
120         }
121         S_ps(plain2);
122         for (int j = 0; j < 16; j++)
123         {
124             plain3[j] = int(plain2[j]);
125         }
126         Row_shift(plain3);
127         for (int j = 0; j < 16; j++)
128         {
129             plain[j / 4][j % 4] = char(plain3[j]);
130         }
131         column_mix(plain);
132         and_round(plain, key, 4 * (i + 1));
133     }
134     //cout << "密文为: ";
```

```
135     /*for (int i = 0; i < 4; i++)
136     {
137         for (int j = 0; j < 4; j++)
138         {
139             cout << hex << int(plain[i][j]);
140         }
141     }*/
142
143 }
144
145
146 }
147 end = clock();
148 double time1 = double(end - start) / 1000;
149 cout << "所用时间为: " << time1 << "秒";
150
151 return 0;
152 }
153 int S_ps(unsigned char* plain)//S盒代换
154 {
155     int ret = 0;
156     for (int i = 0; i < 16; i++)
157     {
158         plain[i] = S_Table[plain[i] >> 4][plain[i] & 0x0F];
159     }
160     return ret;
161 }
162 int S_cs(unsigned char* cipher)//逆S盒代换
163 {
164     int ret = 0;
165     for (int i = 0; i < 16; i++)
166     {
```



```
167     cipher[i] = ReS_Table[cipher[i] >> 4][cipher[i] & 0x0F];
168 }
169 return ret;
170 }
171 int Row_shift(unsigned int* plain)
172 {
173     int ret = 0;
174     plain[1] = (plain[1] >> 24 | plain[1] << 8);
175     plain[2] = (plain[2] >> 16 | plain[2] << 16);
176     plain[3] = (plain[3] >> 8 | plain[3] << 24);
177     return ret;
178 }
179 int ReRow_shift(unsigned int* cipher)
180 {
181     int ret = 0;
182     cipher[1] = (cipher[1] >> 8 | cipher[1] << 24);
183     cipher[2] = (cipher[2] >> 16 | cipher[2] << 16);
184     cipher[3] = (cipher[3] >> 24 | cipher[3] << 8);
185     return ret;
186 }
187 char mul(unsigned char arr, unsigned char plain)
188 {
189     unsigned char result = 0;
190     while (arr)
191     {
192         if (arr & 0x01)
193         {
194             result ^= plain;
195         }
196         arr = arr >> 1;
197         if (plain & 0x80)
198         {
```

```
199     plain = plain << 1;
200     plain ^= 0x1B;
201 }
202 else
203     plain = plain << 1;
204 }
205 return result;
206 }
207 int column_mix(unsigned char(*plain)[4])
208 {
209     int ret = 0;
210     unsigned char temp[4][4];
211     memcpy(temp, plain, 16);
212     for (int i = 0; i < 4; i++)
213     {
214         for (int j = 0; j < 4; j++)
215         {
216             plain[i][j] = mul(mixarr[i][0], temp[0][j]) ^ mul(mixarr[
217 i][1], temp[1][j]) ^ mul(mixarr[i][2], temp[2][j]) ^ mul(
218 mixarr[i][3], temp[3][j]);
219         }
220     }
221     return ret;
222 }
223 int keyarr_S(unsigned char(*key)[44], unsigned int col)
224 {
225     int ret = 0;
226     for (int i = 0; i < 4; i++)
227     {
228         key[i][col] = S_Table[key[i][col] >> 4][key[i][col] & 0x0F
229 ];
230     }
```

```
228     return ret;
229 }
230 int g_f(unsigned char(*key)[44], unsigned int col)
231 {
232     int ret = 0;
233     for (int i = 0; i < 4; i++)//先放到下一列, 且行移位
234     {
235         key[i][col] = key[(i + 1) % 4][col - 1];
236     }
237     keyarr_S(key, col);//S盒置换
238     key[0][col] = key[0][col] ^ Rcon[col / 4];//异或
239     return ret;
240 }
241 int Extendkey(const unsigned char(*pkey)[4], unsigned char(*key
    )[44])
242 {
243     int ret = 0;
244     for (int i = 0; i < 16; i++)//主密钥赋值
245     {
246         key[i & 0x03][i >> 2] = pkey[i & 0x03][i >> 2];
247     }
248
249     for (int i = 1; i < 11; i++)//10轮
250     {
251         g_f(key, 4 * i);//G函数
252         for (int k = 0; k < 4; k++)//每一轮的第一列得到
253         {
254             key[k][4 * i] = key[k][4 * i] ^ key[k][4 * (i - 1)];
255         }
256         for (int j = 1; j < 4; j++)//后三列
257         {
258             for (int k = 0; k < 4; k++)
```

```

259     {
260         key[k][4 * i + j] = key[k][4 * (i - 1) + j] ^ key[k][4
    * i + j - 1];
261     }
262 }
263 }
264 return ret;
265 }
266 int and_round(unsigned char(*plain)[4], unsigned char(*key)
    [44], unsigned int n)
267 {
268     int ret = 0;
269     for (int i = 0; i < 4; i++)
270     {
271         for (int j = 0; j < 4; j++)
272         {
273             plain[i][j] ^= key[i][n + j];
274         }
275     }
276     return ret;
277 }

```

7.2.2 mbedtls-aes-test.c

```

1     /**
2     * @brief   AES Function demo
3     * @author  mculover666
4     * @date    2020/09/23
5     */
6
7 #if !defined(MBEDTLS_CONFIG_FILE)

```

```
8 // #include "mbedtls/config.h"
9 #else
10 #include MBEDTLS_CONFIG_FILE
11 #endif
12
13 // #if defined(MBEDTLS_CIPHER_C)
14
15 #include <stdio.h>
16 #include <string.h>
17 #include <stdint.h>
18 #include "mbedtls/cipher.h"
19
20 /* Private Key */ // 采用不同加密算法时，注意修改密钥及分组长度。
21 //
22 // uint8_t key[8] = {
23 //     0x06, 0xa9, 0x21, 0x40, 0x36, 0xb8, 0xa1, 0x5b
24 //     /*0x51, 0x2e, 0x03, 0xd5, 0x34, 0x12, 0x00, 0x06*/
25 // };
26 uint8_t key[16] = {
27     0x32, 0x30, 0x32, 0x31, 0x30, 0x30, 0x34, 0x36,
28     0x30, 0x30, 0x37, 0x39, 0x0, 0x0, 0x0, 0x0
29 };
30
31 /* Initialization Vector */
32 // uint8_t iv[8] = {
33 //     0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30
34 //     // 0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41
35 // };
36 uint8_t iv[16] = {
37     0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30,
38     0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41
```

```
39 };
40
41 static void dump_buf(uint8_t* buf, uint32_t len)
42 {
43     int i;
44
45     printf("buf:");
46
47     for (i = 0; i < len; i++) {
48         printf("%s%02X%s", i % 16 == 0 ? "\r\n\t" : " ",
49             buf[i],
50             i == len - 1 ? "\r\n" : "");
51     }
52 }
53
54 int aes_test(mbedtls_cipher_type_t cipher_type, uint8_t temp
55 [16]) //可选对称密码算法
56 {
57     int ret;
58     size_t len;
59     int olen = 0;
60     uint8_t output_buf[64];
61     uint8_t o_buf[64];
62     const char *input = temp;
63     //const char input[] = "202100460079";
64     //const char input[] = "HelloWorld123456";
65     //const char input[] = {'m', 'c', 'u', 'l', 'o', 'v', 'e',
66     'r', '6', '6', '6', ' ', 'i', 's'};
67
68     mbedtls_cipher_context_t ctx;
69     const mbedtls_cipher_info_t* info;
```

```
69
70  /* 1. init cipher structuer */
71  mbedtls_cipher_init(&ctx);
72
73  /* 2. get info structuer from type */
74  info = mbedtls_cipher_info_from_type(cipher_type);
75
76  /* 3. setup cipher structuer */
77  ret = mbedtls_cipher_setup(&ctx, info);
78  if (ret != 0) {
79      goto exit;
80  }
81
82  ////* 4. set key */
83  /* ret = mbedtls_cipher_setkey(&ctx, key, sizeof(key) * 8,
MBEDTLS_ENCRYPT);
84      if (ret != 0) {
85          goto exit;
86      }*/
87  ret = mbedtls_cipher_setkey(&ctx, key, 128, MBEDTLS_ENCRYPT
);
88  if (ret != 0) {
89      goto exit;
90  }
91
92  /* 5. set iv */
93  ret = mbedtls_cipher_set_iv(&ctx, iv, sizeof(iv));
94  if (ret != 0) {
95      goto exit;
96  }
97
98  /* 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

```
99     ret = mbedtls_cipher_update(&ctx, (unsigned char*)input,
16, output_buf, &len);
100     if (ret != 0) {
101         goto exit;
102     }
103     olen += len;
104
105     /* 6. update cipher */
106 /*     ret = mbedtls_cipher_update(&ctx, (unsigned char*)input,
strlen(input), output_buf, &len);
107     if (ret != 0) {
108         goto exit;
109     }
110     olen += len;*/
111
112
113
114     /* 7. finish cipher */
115     ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
116     if (ret != 0) {
117         goto exit;
118     }
119     olen += len;
120
121     /* show */
122 // printf("\r\nsource_context: %s\r\n", input);
123 /*dump_buf((uint8_t*)input, strlen(input));
124 printf("cipher name: %s block size is: %d\r\n",
mbedtls_cipher_get_name(&ctx), mbedtls_cipher_get_block_size
(&ctx));
125     dump_buf(output_buf, olen);*/
```



```
126 exit:
127     /* 8. free cipher structure */
128     mbedtls_cipher_free(&ctx);
129
130
131
132
133
134
135 //     /* 1. init cipher structuer */
136 //     mbedtls_cipher_init(&ctx);
137 //
138 //     /* 2. get info structuer from type */
139 //     info = mbedtls_cipher_info_from_type(cipher_type);
140 //
141 //     /* 3. setup cipher structuer */
142 //     ret = mbedtls_cipher_setup(&ctx, info);
143 //     if (ret != 0) {
144 //         goto exit1;
145 //     }
146 //
147 //     /* 4. set key */
148 //     /*ret = mbedtls_cipher_setkey(&ctx, key, sizeof(key) * 8,
149 //         MBEDTLS_DECRYPT);
150 //     if (ret != 0) {
151 //         goto exit1;
152 //     }*/
153 //     ret = mbedtls_cipher_setkey(&ctx, key, 128,
154 //         MBEDTLS_DECRYPT);
155 //     if (ret != 0) {
156 //         goto exit1;
157 //     }
```

```
156 //
157 //      /* 5. set iv */
158 //      ret = mbedtls_cipher_set_iv(&ctx, iv, sizeof(iv));
159 //      if (ret != 0) {
160 //          goto exit1;
161 //      }
162 //
163 //      ///  
/* 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
164 //      ret = mbedtls_cipher_update(&ctx, (unsigned char*)  
output_buf, 16, o_buf, &len);
165 //      if (ret != 0) {
166 //          goto exit1;
167 //      }
168 //      olen += len;
169 //
170 //      /* 6. update cipher */
171 //      /*ret = mbedtls_cipher_update(&ctx, (unsigned char*)input  
, strlen(input), output_buf, &len);
172 //      if (ret != 0) {
173 //          goto exit1;
174 //      }
175 //      olen += len;*/
176 //
177 //
178 //
179 //      /* 7. finish cipher */
180 //      ret = mbedtls_cipher_finish(&ctx, o_buf, &len);
181 //      if (ret != 0) {
182 //          goto exit1;
183 //      }
184 //      olen += len;
```

```

185 //
186 //     /* show */
187 //     printf("\r\nsource_context: %s\r\n", output_buf);
188 //     dump_buf((uint8_t*)output_buf, strlen(output_buf));
189 //     printf("cipher name: %s block size is: %d\r\n",
190 //           mbedtls_cipher_get_name(&ctx), mbedtls_cipher_get_block_size
191 //           (&ctx));
192 //     dump_buf(o_buf, olen);
193 //exit1:
194 //     /* 8. free cipher structure */
195 //     mbedtls_cipher_free(&ctx);
196
197     return ret;
198 }
199
200 //int main()
201 //{
202 //     aes_test(MBEDTLS_CIPHER_AES_128_ECB);
203 //     return 1;
204 //}
205
206 //endif /* MBEDTLS_CIPHER_C */

```

7.2.3 main.c

```

1     #include "mbedtls/cipher.h"
2 #include "mbedtls/dhm.h"
3 #include "mbedtls/md.h"
4 #include "stdlib.h"
5 #include "time.h"
6 int main()
7 {

```

```
8   clock_t start, end;
9   start = clock();
10  uint8_t temp[17] = { 0 };
11  double time1;
12  for (int i = 0; i < 100; i++)//100组
13  {
14      for (int j = 0; j < 256; j++)//256个128bit明文
15      {
16          temp[0] = j + 48;//第0字节遍历256种
17          temp[16] = '\0';
18          for (int k = 1; k < 16; k++)
19          {
20              srand((unsigned)time(NULL));
21              temp[k] = rand() % 256;//其他15字节随机
22          }
23          aes_test(MBEDTLS_CIPHER_AES_128_ECB, temp); /*无填充, 修改输入长度为分组长度倍数 */
24              //传入函数
25      }
26
27
28  }
29  end = clock();
30  time1 = (double)(end - start) / CLOCKS_PER_SEC;
31  printf("所用时间为: %f秒\n", time1);
32  //aes_test(MBEDTLS_CIPHER_DES_CBC);
33
34  //mbedtls_rsa_test();
35  //mbedtls_rsa_sign_test();
36  //mbedtls_dhm_test();
37  //mbedtls_shax_test(MBEDTLS_MD_SHA512);
38  //aes_test(MBEDTLS_CIPHER_ARIA_128_ECB);
```

39

40

41

`return 1;``}`