AES 算法设计与分析

山东大学网络空间安全学院
密码学引论　实验报告

21 级　高培民　　202100460055
21 级　吴兵　　202000161241
21 级　田行健　　202100460086
21 级　王羽煊　　202100460036

2023 年 3 月 24 日

目录

1 小组简介

1.1 组员分工

高培民：负责编写第一题第二题实验相关 C 语言代码，并负责第二题代码思路部分的撰写。

吴兵：负责第一题 Python 代码编写以及第一题代码思路的撰写，并测试相关代码。

田行健：负责 AES 算法简介以及收获感悟部分的撰写，论文整体排版以及结果分析。

王羽煊：负责第一题 Python 代码编写以及第一题代码思路的撰写，并测试相关代码。

1.2 环境配置

采用平台环境配置如下：

硬件环境：

处理器：inter(R)Core(TM)i7-10750H CPU 64 位操作系统（笔者）

内存：16GB （笔者）

软件环境：

操作系统：Windows 10 专业版 x64

Python 解释器：PyCharm Community Edition 2022.2.1

C++ 编译器：Visual Studio Community 2019

2 AES 算法简介

2.1 分组密码

对称加密可分为分组密码与流密码。

分组密码 (block cipher) 是一类每次只能处理特定长度的一块数据的密码算法，这里的“一块”就称为分组（block）。一个分组的比特数就称为分组长度（block lenght）。例如 DES 和 3DES 的分组长度都是 64 比特。AES 的分组长度为 128 比特。分组密码处理完一个分组就结束了，因此不需要通过内部状态来记录加密的进度；相对地，流密码是对一串数据进行连续处理，因此需要保持内部状态。

我们此次研究的 AES 就是一种流行的分组密码。

2.2 AES 介绍

AES（Advanced Encryption Standard）是一种对称密钥加密算法，它是由美国国家标准与技术研究所（NIST）在 2001 年发布的一种加密标准。AESd 的特点在于加密和解密使用相同的密钥，的设计目标是提供高度的安全性、高速度、高效性、简洁性和易用性。它是取代 DES（Data Encryption Standard）的加密算法，目前已成为广泛使用的加密标准之一。

2.3 AES 原理

AES 算法的流程大致可分为以下四个步骤：

1. **密钥扩展** (Key Expansion)

在 AES 算法中，密钥的长度可以是 128 比特、192 比特或 256 比特。在密钥扩展阶段，算法会根据密钥的长度生成一系列的轮密钥，用于加密和解密的每一轮操作中。

2. **初始轮** (Initial Round)

在初始轮中，输入数据会被进行一次简单的处理，包括字节替换、行移位、列混淆和轮密钥加。这一步骤的目的是将输入数据进行简单的混淆，以增加加密的强度。

3. **重复轮** (Rounds)

在重复轮中，输入数据会被重复地进行字节替换、行移位、列混淆和轮密钥加等操作，以增加加密的强度。这些操作是通过轮函数 (Round Function) 实现的，轮函数是一个复杂的数学函数，包括有限域上的乘法、加法、异或等运算。

4. **最终轮** (Final Round)

在最终轮中，输入数据会进行最后一次的字节替换、行移位和轮密钥加等操作，但不包括列混淆。这一步骤是为了最后的混淆，以增加加密的强度。

而 AES 算法的核心是轮函数（Round Function），它由四个步骤组成：字节替代 (SubBytes)、行移位 (ShiftRows)、列混淆 (MixColumns) 和轮密钥加 (AddRoundKey)。

1. **字节替换** (SubBytes)

在 AES 算法中，字节替换操作是通过一个称为 S 盒 (Substitution Box) 的查找表来实现的。S 盒将一个输入字节映射到一个输出字节，这个映射关系是通过复杂的算法得出的。由于 S 盒是固定的，因此攻击者难以通过分析 S 盒来推断出密钥。

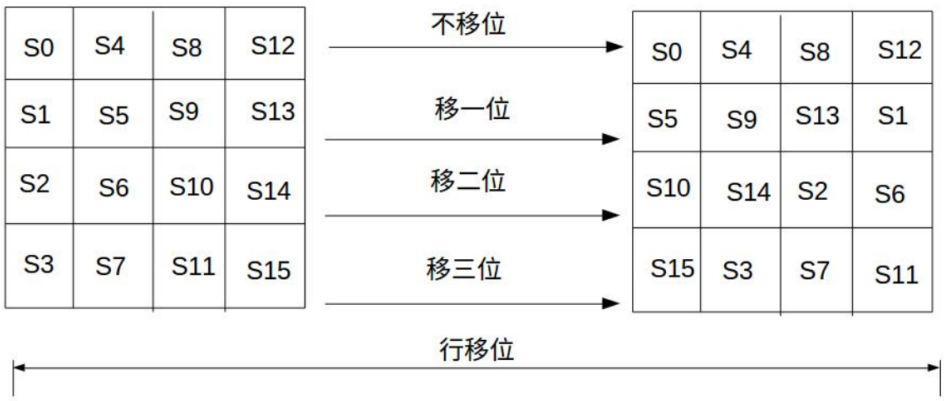
S 盒如下图所示：

行/列	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	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76
1	0xca	0x82	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0
2	0xb7	0xfd	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x71	0xd8	0x31	0x15
3	0x04	0xc7	0x23	0xc3	0x18	0x96	0x05	0x9a	0x07	0x12	0x80	0xe2	0xeb	0x27	0xb2	0x75
4	0x09	0x83	0x2c	0x1a	0x1b	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84
5	0x53	0xd1	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf
6	0xd0	0xef	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8
7	0x51	0xa3	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0x10	0xff	0xf3	0xd2
8	0xcd	0x0c	0x13	0xec	0x5f	0x97	0x44	0x17	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0x19	0x73
9	0x60	0x81	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0x14	0xde	0x5e	0x0b	0xdb
A	0xe0	0x32	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79
B	0xe7	0xc8	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08
C	0xba	0x78	0x25	0x2e	0x1c	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0x1f	0x4b	0xbd	0x8b	0x8a
D	0x70	0x3e	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0x1d	0x9e
E	0xe1	0xf8	0x98	0x11	0x69	0xd9	0x8e	0x94	0x9b	0x1e	0x87	0xe9	0xce	0x55	0x28	0xdf
F	0x8c	0xa1	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0x16

2. **行移位** (ShiftRows)

行移位操作是通过将每一行的字节向左循环移位来实现的。第一行不需要移位，第二行向左移动 1 个字节，第三行向左移动 2 个字节，第四行向左移动 3 个字节。这一操作可以增加加密的强度，使得相邻的字节位置发生变化。

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，如下图所示：



3. **列混淆** (MixColumns)

列混合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，如下图的公式所示 (代码实现时会对此处进行查找表优化)：

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

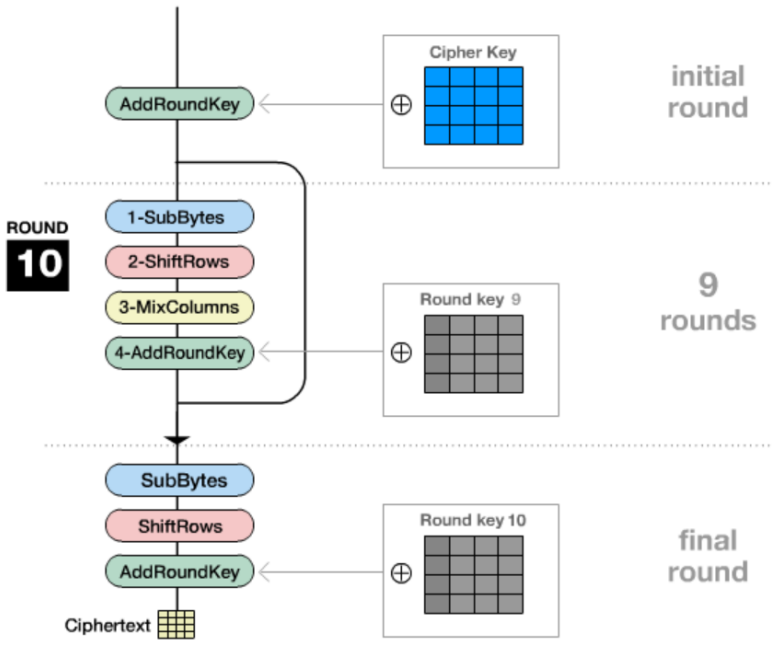
状态矩阵中的第 j 列 ($0 \leq j \leq 3$) 的列混合可以表示为下图所示：

$$\begin{aligned} s'_{0,j} &= (2 * s_{0,j}) \oplus (3 * s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 * s_{1,j}) \oplus (3 * s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 * s_{2,j}) \oplus (3 * s_{3,j}) \\ s'_{3,j} &= (3 * s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 * s_{3,j}) \end{aligned}$$

4. **轮密钥加** (AddRoundKey)

轮密钥加操作是通过将当前输入数据和当前轮的轮密钥进行按位异或运算来实现的。这个操作可以增加加密的强度，使得每一轮的操作都受到密钥的影响。

轮密钥加是将 128 位轮密钥 K_i 同状态矩阵中的数据进行逐位异或操作，如下图所示。其中，密钥 K_i 中每个字 W[4i],W[4i+1],W[4i+2],W[4i+3] 为 32 位比特字，包含 4 个字节，他们的生成算法下面在下面介绍。轮密钥加过程可以看成是字逐位异或的结果，也可以看成字节级别或者位级别的操作。也就是说，可以看成 S0 S1 S2 S3 组成的 32 位字与 W[4i] 的异或运算。



2.4 雪崩效应

在密码学中，雪崩效应（avalanche effect）指加密算法（尤其是块密码和加密散列函数）的一种理想属性。雪崩效应是指当输入发生最微小的改变（例如，反转一个二进制位）时，也会导致输出的不可区分性改变 (输出中每个二进制位有 50% 的概率发生反转)。合格块密码中，无论密钥或明文的任何细微变化都必须引起密文的不可区分性改变。该术语最早由 Horst Feistel 使用，尽管其概念最早可以追溯到克劳德 · 香农提出的扩散（diffusion）。

若某种块密码或加密散列函数没有显示出一定程度的雪崩特性，那么它被认为具有较差的随机化特性，从而密码分析者得以仅仅从输出推测输入。这可能导致该算法部分乃至全部被破解。因此，从加密算法或加密设备的设计者角度来说，满足雪崩效应乃是必不可缺的圭臬。

构造一个具备良好雪崩效应的密码或散列是至关重要的设计目标之一。这正是绝大多数块密码采用了乘积密码的原因，也是大多数散列函数使用大数据块的原因。这些特性均使得微小的变化得以通过算法的迭代迅速增殖，造成输出的每一个二进制位在算法终止前均受到输入的每一个二进制位的影响。

3 算法设计与测试结果

3.1 第一题

3.1.1 问题描述

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 0x00，明文最左侧为最低位），求对应的密文。

3.1.2 解题思路 (Python 实现)

根据之前描述的 AES 算法具体过程，设计各个函数来实现 AES 算法。

1、用 ndarray 数组形式将明文存储下来，ciphertext 用来存储每一轮加密后的内容。因为密钥和明文一样，均为学号，所以我们用.copy() 进行复制。再定义全局变量 rcon 用来存储轮常量，这些均为字节形式，Mix_Column_Table 用来定义 x->(2x,x,x,3x) 查找表的优化，S_Box 用字典形式存储 s 盒的内容。

代码如下：

```
plaintext = np.array(
[[0x32, 0x30, 0x32, 0x31], [0x30, 0x30, 0x34, 0x36], [0x30, 0
   x30, 0x35, 0x35], [0x00, 0x00, 0x00, 0x00]],dtype=np.uint8)

ciphertext = np.zeros((4, 4), dtype=np.uint8)

key = plaintext.copy()

rcon = np.array([
[0x01, 0x00, 0x00, 0x00],
[0x02, 0x00, 0x00, 0x00],
……], dtype=np.uint8)

Mix_Columns_Table = np.zeros((256, 4), dtype=np.uint8)

keys = np.zeros((11, 4, 4), dtype=np.uint8)

S_Box = {
        '0x0': 0x63, '0x1': 0x7c, '0x2': 0x77, '0x3': 0x7b, '0
           x4': 0xf2, '0x5': 0x6b, '0x6': 0x6f, '0x7': 0xc5, '0
           x8': 0x30, ……}
```

2、mul 函数：用于计算计算有限域上的 1x,2x,3x 的值，输入变量为 x,y, 其中 x 表示的是 1，2，3 的一个，y 表示的是有限域上的元素，输出变量为在有限域上的 x*y 的值。通过观察可以优化 2*y 和 3*y 的计算如下：

$$02 * S_{i,c} = (b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 << 1) \oplus b_7 * \{1b\}$$

$$03 * Si\ c = 02 * S_{i,c} \oplus S_{i,c}$$

代码如下：

```
def mul(x, y):
    """计算有限域上的1x,2x,3x"""
    res = 0
```

```
    if x == 0x01:
        res = y
    elif x == 0x02:
        if (y & 0x80) == 0x00:
            res = y << 1 & 0b011111111
        else:
            res = (y << 1 & 0b011111111) ^ 0x1b
    elif x == 0x03:
        res = mul(0x02, y) ^ y
    return res
```

3、Generate_Table() 函数用来生成 x->(2x,x,x,3x) 查找表，Mix_Columns_Table 为二维数组，第 0 列代表 2x 的值，第 1 列代表 x 的值，第 2 列代表 x 的值，第 3 列代表 3x 的值。

代码如下：

```
def Generate_Table():
    """生成{2x,x,x,3x}的列混合查找表"""
    global plaintext, S_BOX, Mix_Columns_Table
    for i in range(256):
        j = hex(i)
        Mix_Columns_Table[i][0] = mul(0x2, S_Box.get(j))
        Mix_Columns_Table[i][1] = S_Box.get(j)
        Mix_Columns_Table[i][2] = S_Box.get(j)
        Mix_Columns_Table[i][3] = mul(0x3, S_Box.get(j))
```

4、Key_Expansion() 是轮密钥生成函数。首先将主密钥 key 和存储 11 轮加密的轮密钥的 keys 设置为全局变量。

接着先初始化第一轮轮密钥，在这一轮中的轮密钥与主密钥一致。

接着生成 2 至 11 轮的轮密钥，在这 10 轮密钥生成中，每一轮都需要经过字循环、字替换、与轮常量异或以及与前一轮的轮密钥异或。

代码如下：

```
def Key_Expansion():
    """生成轮密钥，用keys（11 X）储存"""
```

```
    global keys, key
    # 第一轮的密钥是原始密钥
    for i in range(4):
        keys[0][0][i] = key[i][0]
        keys[0][1][i] = key[i][1]
        ……
    # 剩余 10 轮密钥生成
    for i in range(1, 11):  # i 表示第 i 个轮密钥
        # 字循环 + 字代替
        arr = np.zeros((4,), dtype=np.uint8)
        arr[0] = S_Box.get(hex(keys[i - 1][1][3]))
        ……
        # 与轮常量异或
        for k in range(4):
            arr[k] = arr[k] ^ rcon[i - 1][k]
        # 再次异或
        for m in range(4):  # 求 W4
            keys[i][0][0] = arr[0] ^ keys[i - 1][0][0]
                ……
        for s in range(1, 4):  # 生成 W5, W6, W7, s 表示列, t 表示行
            for t in range(4):
                keys[i][t][s] = keys[i - 1][t][s] ^ keys[i][t][
                    s - 1]
```

5、AddRoundKey() 函数代表每一轮加密中列混合的结果与轮密钥异或，keys 为三位数组形式，有 11 个二维数组，每个二维数组代表这一轮使用的轮密钥。

代码如下：

```
def AddRoundKey(round):
    """每一轮加密中列混合的结果与轮密钥异或"""
    global keys, ciphertext
    for i in range(4):
        for j in range(4):
```

```
            ciphertext[i][j] = ciphertext[i][j] ^ keys[round][i
                ][j]
```

6、SubBytes() 函数用于输出 S 盒的结果。因为 S 盒存储在一个字典中，所以我们只需把 ciphertext 的每一个字节都用字典 get 方法替换为对应的 S 盒的输出即可。

ShiftRows() 进行行移位，第一行元素不变，从第二行开始，先用一个 brr 数组存下这一行的元素，若 i 等于 1，则左移 1 位；若 i 等于 2，则左移 2 位；若 i 等于 3，则左移 3 位。

代码如下：

```
def SubBytes():
    """最后一轮需要用到S盒"""
    global ciphertext
    for i in range(4):
        for j in range(4):
            ciphertext[i][j] = S_Box.get(hex(ciphertext[i][j]))

def ShiftRows():
    """行移位"""
    global ciphertext
    for i in range(1, 4):
        brr = np.zeros((4,), dtype=np.uint8)
        for j in range(4):
            brr[j] = ciphertext[i][j]
        if (i == 1):
            temp = ciphertext[i][0]
            ciphertext[i][0] = ciphertext[i][1]
            ciphertext[i][1] = ciphertext[i][2]
            ciphertext[i][2] = ciphertext[i][3]
            ciphertext[i][3] = temp
        ... ...
```

7、MixColumns() 函数用来进行列混合。基本思路如下：

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2(x_0 \oplus x_1) \oplus x_1 \oplus x_2 \oplus x_3 \\ x_0 \oplus 2(x_1 \oplus x_2) \oplus x_2 \oplus x_3 \\ x_0 \oplus x_1 \oplus 2(x_2 \oplus x_3) \oplus x_3 \\ 2(x_0 \oplus x_3) \oplus x_0 \oplus x_1 \oplus x_2 \end{bmatrix}$$

同时由于 S 盒可以和 SR 互换顺序，因此可以将 S 盒与列混合优化成一个函数，即：

$$[s'_{0,i}, s'_{1,i}, s'_{0=2,i}, s'_{3,i}] = T(s_{0,i}) \oplus T(s_{1,mod(i+1,4)}) >> 8 \oplus T(s_{2,mod(i+2,4)}) >> T(s_{3,mod(i+3,4)}) \oplus >> 24$$

因此我们遍历密文的每一列，每一列都由上述公式生成。

代码如下：

```
def MixColumns():
    """列混合"""
    global ciphertext
    for i in range(4):  #i是列循环
        temp = [ciphertext[0][i], ciphertext[1][i], ciphertext
            [2][i], ciphertext[3][i]]

        ciphertext[0][i] = Mix_Columns_Table[temp[0]][0] ^
            Mix_Columns_Table[temp[1]][3] ^ Mix_Columns_Table[
            temp[2]][
              2] ^ Mix_Columns_Table[temp[3]][1]
        ... ...
```

8、Encrypt() 函数：首先将明文转化为十六进制输出，同样将密钥转化为十六进制输出，再生成 11 组轮密钥，第 0 轮加密直接进行异或，第 1-9 论加密进行行移位，列混合，轮密钥加，因为我们这里已经在 MixColumn 里将 s 盒和列混淆查表后合并在一起，所以这里不用额外的进入 s 盒。第 10 轮加密，先进入 s 盒，再行移位，轮密钥加。最后输出密文即可。

代码如下：

```
def Encrypt():
    """加密函数"""
    #密钥生成
```

```
    Key_Expansion()
    global plaintext, key, keys, ciphertext, Mix_Columns_Table
    ciphertext = plaintext.copy()
    for i in range(4):
        ciphertext[0][i] = plaintext[i][0]
        ciphertext[1][i] = key[i][1]
        ciphertext[2][i] = key[i][2]
        ciphertext[3][i] = key[i][3]
    # 打印明文
    print("明 文: ")
    for i in range(4):
        for j in range(4):
            print(hex(plaintext[i][j]), end=' ')
        print("\n")
    # 打印密钥
    print("密 钥: ")
    for i in range(4):
        for j in range(4):
            print(hex(key[i][j]), end=' ')
        print("\n")
    # 生成 11 组轮密钥
    Key_Expansion()
    # 第一轮加密
    AddRoundKey(0)
    # 第 1 9 轮加密
    for i in range(1, 10):
        # SubBytes()
        ShiftRows()
        MixColumns()
        AddRoundKey(i)
    # 第 10 轮加密
    SubBytes()
```

```
    ShiftRows()
    AddRoundKey(10)
    #打印密文
    print("密文: ")
    for i in range(4):
        for j in range(4):
            print(hex(ciphertext[i][j]), end=' ')
        print("\n")
```

3.1.3 **解题思路** (C **实现**)

注意到在实现 Python 相关代码的时候，采用了列主序的排列方法，而此种排法会导致跨行取内存中的数据，访存效率不佳，因此在 C 语言中采用行主序的排列方法，实现取 Cache 块的访存硬件优化。部分代码如下：

```
void AddRoundKey(uint8_t* mat, uint8_t* key_mat) {
        for (int i = 0; i < 16; i++) {
                mat[i] ^= key_mat[i];
        }
}
void SubBytes(uint8_t* mat) {
        for (int i = 0; i < 16; i++) {
                mat[i] = S_BOX[mat[i]];
        }
}
```

与此同时，为了验证加密结果的正确性，还引入了解密部分，解密过程的 AES 每一步使用的函数过程次序一致，但 S 盒、SR 等步骤需要进行相应的逆运算。部分代码如下：

```
void InvSubBytes(uint8_t* mat) {
        for (int i = 0; i < 16; i++)
                mat[i] = S_INV[mat[i]];
}
```

完整代码详见附录。

3.1.4 运行结果分析

组内四人学号用 Python 的加密结果 (将列主序变换为行主序后):

202100460055	202000161241	202100460086	202100460036
0xe1 0x07 0x17 0xf7	0xaf 0x9b 0x8e 0x0e	0x42 0xa3 0xd3 0x6b	0x2b 0x3f 0xa7 0x60
0x4f 0x90 0x64 0x91	0xac 0x91 0xb4 0x2c	0x28 0x73 0x4e 0x40	0x95 0x14 0x17 0x5e
0xd4 0x06 0xef 0x92	0xc4 0xc1 0x0e 0xd1	0xd4 0x41 0x87 0x9f	0x1f 0x26 0xec 0x3d
0xd2 0xaa 0xc2 0x00	0x79 0xe2 0x18 0xba	0xb8 0x2a 0x30 0x27	0x20 0xaa 0x1c 0x4f

C 运行结果如下:

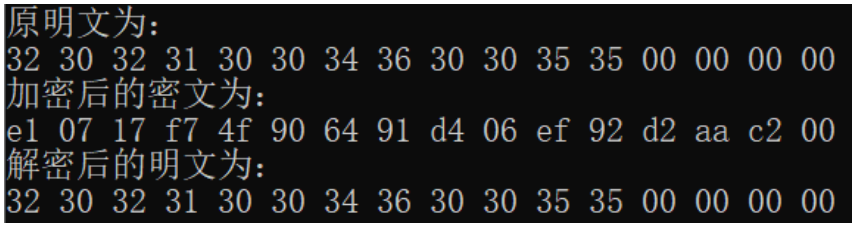


Figure 1: 202100460055

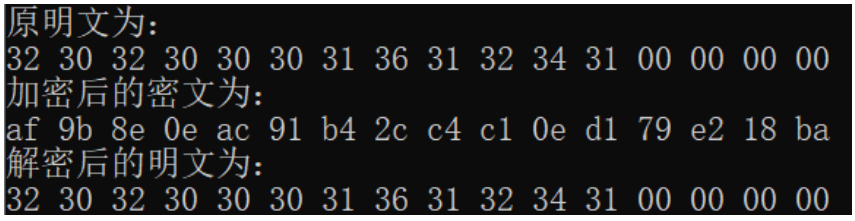


Figure 2: 202000161241

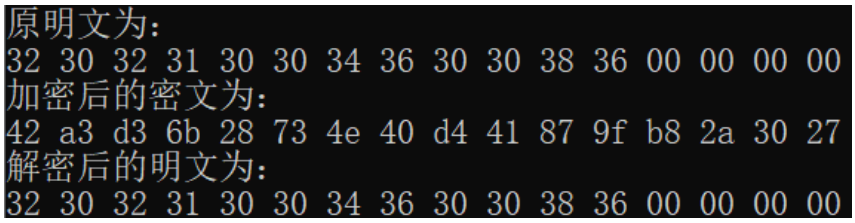


Figure 3: 2021000460086

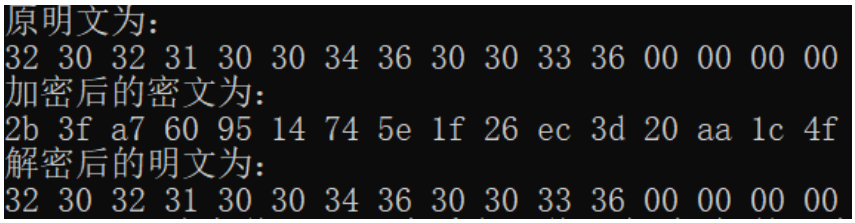


Figure 4: 202100460036

经过对比二者的运行结果相同，且 C 语言解密结果正确，第一题正确实现。

3.2 第二题

3.2.1 问题描述

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。

3.2.2 手动实现 AES

对于 AES 加密核心部分的代码仍与上文相同，并继续使用个人学号作为密钥，不同的是明文生成的过程。按照题目要求，创造一个三维矩阵，共有 100 组数据，每组数据下有 256 个长度为 16 字节的明文。

在生成每一个随机明文时，首字节数据从 0 开始计数并统计到 255，之后利用当前机器时间设立随机数种子，每一个随机字节数由 rand() 函数来生成，并利用模 255 运算使得生成的随机数大小介于 0 至 255 之间，满足一个字节数所能表示的数据范围。

```
void GenerateText() {
        srand(time(NULL));
        for (int i = 0; i < 100; i++) {
                for (int j = 0; j < 256; j++) {
                        plain_text_table[i][j][0] = j;
                        for (int k = 1; k < 16; k++) {
                                plain_text_table[i][j][k] =
                                    rand() % 255;
                        }
                }
        }
}
```

计时部分利用 C 程序所提供的 time 库，利用 clock() 函数得到 CPU 时钟周期数，再由时钟周期长度确定具体运行时间即可。

3.2.3 利用 mbedTLS 库实现 AES

在此利用先前实验课所学习过的 mbedTLS 库，调用 AES 所专有的加解密函数来实现 AES-128-ECB 模式。

在具体实现时，利用 mbedtls_aes_init() 函数完成初始化工作，利用 mbedtls_aes_setkey_enc() 函数进行密钥传递，利用 mbedtls_aes_crypt_ecb() 函数进行加密工作，并同步进行错误检测的工作。

```
mbedtls_aes_init(&aes_ctx);
ret = mbedtls_aes_setkey_enc(&aes_ctx, key, sizeof(key) * 8);
if (ret != 0) {
                goto exit;
}
ret = mbedtls_aes_crypt_ecb(&aes_ctx, MBEDTLS_AES_ENCRYPT,
   plain_text, output1);
if (ret != 0) {
                goto exit;
}
```

相应地，使用 mbedtls_aes_setkey_dec() 函数进行解密阶段的密钥传递，使用 mbedtls_aes_crypt_cbc() 函数完成解密工作。

```
ret = mbedtls_aes_setkey_dec(&aes_ctx, key, sizeof(key) * 8);
if (ret != 0) {
                goto exit;
}
ret = mbedtls_aes_crypt_ecb(&aes_ctx, MBEDTLS_AES_DECRYPT,
   output1, output2);
if (ret != 0) {
                goto exit;
}
```

最后，密钥继续使用个人密钥，而随机明文数据组生成的过程继续沿用 GenerateText() 函数，计时也是继续使用上文所述的 time 库方法即可。

3.2.4 运行结果分析

100 组明文运行 1 轮：

实现方法	时间
手动实现	0.063000s
mbedTLS 库实现	0.010000s

100 组明文运行 100 轮：

实现方法	时间
手动实现	6.461000s
mbedTLS 库实现	1.258000s

经过对比通过以上结果可以看出，只运行一次两种方式时间都很快，但是若运行 100 次，mbedTLS 库函数还是明显优于自写函数，但其效率均处于能接受的状态。

4 收获感悟

本次实验我们使用 C 语言和 Python 编程语言分别实现了 AES 加密算法，对于给定的明文与密钥，成功还原出对应的密文。在成功解密的基础上，我们对比了手动实现和 mbedTLS 库实现在时间开销上的差异，进一步地认识与理解 AES 算法，收获颇丰。

具体实现的过程中，C 语言和 Python 代码均作了查找表上的优化，即将 MR 与 SubBytes 过程合并并预设好查找表，与此同时，C 语言还考虑到了数据在内存中的排列方式，采用了行主序的操作完成相关代码，对于 Cache 块的访存效率更好。

在实现 AES 算法的过程中，我也意识到了代码实现的安全性和效率之间的平衡。为了确保加密算法的安全性，需要使用高强度的加密密钥，这会导致密钥扩展的开销很大。同时，也需要考虑加密算法的效率，因为在实际应用中，加密和解密的速度也是非常重要的。

此外，我还学到了一些代码实现中的最佳实践，例如代码的可读性、可维护性和可测试性等。这些实践可以帮助代码更好地满足实际需求，减少错误和维护成本。

总的来说，通过实现 AES 算法，我们深刻地认识到了算法实现的复杂性和细节之多。同时，我们也学到了一些代码实现的最佳实践和注意事项，这对于我们的日后编程和算法实现都是非常有帮助的。

5 附录

5.1 第一题 AES 实现 (Python)

```python
import numpy as np
import warnings
import copy

warnings.filterwarnings('ignore')

plaintext = np.array(
    [[0x32, 0x30, 0x32, 0x31], [0x30, 0x30, 0x34, 0x36], [0x30,
        0x30, 0x35, 0x35], [0x00, 0x00, 0x00, 0x00]],
    dtype=np.uint8)
ciphertext = np.zeros((4, 4), dtype=np.uint8)
key = plaintext.copy()
rcon = np.array([
    [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]
], dtype=np.uint8)

Mix_Columns_Table = np.zeros((256, 4), dtype=np.uint8)

keys = np.zeros((11, 4, 4), dtype=np.uint8)

```

```
29 S_Box = {
30     '0x0': 0x63, '0x1': 0x7c, '0x2': 0x77, '0x3': 0x7b, '0x4':
         0xf2, '0x5': 0x6b, '0x6': 0x6f, '0x7': 0xc5, '0x8': 0x30
         ,
31     '0x9': 0x01, '0xa': 0x67, '0xb': 0x2b, '0xc': 0xfe, '0xd':
         0xd7, '0xe': 0xab, '0xf': 0x76,
32     '0x10': 0xca, '0x11': 0x82, '0x12': 0xc9, '0x13': 0x7d, '0
         x14': 0xfa, '0x15': 0x59, '0x16': 0x47, '0x17': 0xf0,
33     '0x18': 0xad, '0x19': 0xd4, '0x1a': 0xa2, '0x1b': 0xaf, '0
         x1c': 0x9c, '0x1d': 0xa4, '0x1e': 0x72, '0x1f': 0xc0,
34     '0x20': 0xb7, '0x21': 0xfd, '0x22': 0x93, '0x23': 0x26, '0
         x24': 0x36, '0x25': 0x3f, '0x26': 0xf7, '0x27': 0xcc,
35     '0x28': 0x34, '0x29': 0xa5, '0x2a': 0xe5, '0x2b': 0xf1, '0
         x2c': 0x71, '0x2d': 0xd8, '0x2e': 0x31, '0x2f': 0x15,
36     '0x30': 0x04, '0x31': 0xc7, '0x32': 0x23, '0x33': 0xc3, '0
         x34': 0x18, '0x35': 0x96, '0x36': 0x05, '0x37': 0x9a,
37     '0x38': 0x07, '0x39': 0x12, '0x3a': 0x80, '0x3b': 0xe2, '0
         x3c': 0xeb, '0x3d': 0x27, '0x3e': 0xb2, '0x3f': 0x75,
38     '0x40': 0x09, '0x41': 0x83, '0x42': 0x2c, '0x43': 0x1a, '0
         x44': 0x1b, '0x45': 0x6e, '0x46': 0x5a, '0x47': 0xa0,
39     '0x48': 0x52, '0x49': 0x3b, '0x4a': 0xd6, '0x4b': 0xb3, '0
         x4c': 0x29, '0x4d': 0xe3, '0x4e': 0x2f, '0x4f': 0x84,
40     '0x50': 0x53, '0x51': 0xd1, '0x52': 0x00, '0x53': 0xed, '0
         x54': 0x20, '0x55': 0xfc, '0x56': 0xb1, '0x57': 0x5b,
41     '0x58': 0x6a, '0x59': 0xcb, '0x5a': 0xbe, '0x5b': 0x39, '0
         x5c': 0x4a, '0x5d': 0x4c, '0x5e': 0x58, '0x5f': 0xcf,
42     '0x60': 0xd0, '0x61': 0xef, '0x62': 0xaa, '0x63': 0xfb, '0
         x64': 0x43, '0x65': 0x4d, '0x66': 0x33, '0x67': 0x85,
43     '0x68': 0x45, '0x69': 0xf9, '0x6a': 0x02, '0x6b': 0x7f, '0
         x6c': 0x50, '0x6d': 0x3c, '0x6e': 0x9f, '0x6f': 0xa8,
44     '0x70': 0x51, '0x71': 0xa3, '0x72': 0x40, '0x73': 0x8f, '0
         x74': 0x92, '0x75': 0x9d, '0x76': 0x38, '0x77': 0xf5,
```

```
45      '0x78': 0xbc, '0x79': 0xb6, '0x7a': 0xda, '0x7b': 0x21, '0
          x7c': 0x10, '0x7d': 0xff, '0x7e': 0xf3, '0x7f': 0xd2,
46      '0x80': 0xcd, '0x81': 0x0c, '0x82': 0x13, '0x83': 0xec, '0
          x84': 0x5f, '0x85': 0x97, '0x86': 0x44, '0x87': 0x17,
47      '0x88': 0xc4, '0x89': 0xa7, '0x8a': 0x7e, '0x8b': 0x3d, '0
          x8c': 0x64, '0x8d': 0x5d, '0x8e': 0x19, '0x8f': 0x73,
48      '0x90': 0x60, '0x91': 0x81, '0x92': 0x4f, '0x93': 0xdc, '0
          x94': 0x22, '0x95': 0x2a, '0x96': 0x90, '0x97': 0x88,
49      '0x98': 0x46, '0x99': 0xee, '0x9a': 0xb8, '0x9b': 0x14, '0
          x9c': 0xde, '0x9d': 0x5e, '0x9e': 0x0b, '0x9f': 0xdb,
50      '0xa0': 0xe0, '0xa1': 0x32, '0xa2': 0x3a, '0xa3': 0x0a, '0
          xa4': 0x49, '0xa5': 0x06, '0xa6': 0x24, '0xa7': 0x5c,
51      '0xa8': 0xc2, '0xa9': 0xd3, '0xaa': 0xac, '0xab': 0x62, '0
          xac': 0x91, '0xad': 0x95, '0xae': 0xe4, '0xaf': 0x79,
52      '0xb0': 0xe7, '0xb1': 0xc8, '0xb2': 0x37, '0xb3': 0x6d, '0
          xb4': 0x8d, '0xb5': 0xd5, '0xb6': 0x4e, '0xb7': 0xa9,
53      '0xb8': 0x6c, '0xb9': 0x56, '0xba': 0xf4, '0xbb': 0xea, '0
          xbc': 0x65, '0xbd': 0x7a, '0xbe': 0xae, '0xbf': 0x08,
54      '0xc0': 0xba, '0xc1': 0x78, '0xc2': 0x25, '0xc3': 0x2e, '0
          xc4': 0x1c, '0xc5': 0xa6, '0xc6': 0xb4, '0xc7': 0xc6,
55      '0xc8': 0xe8, '0xc9': 0xdd, '0xca': 0x74, '0xcb': 0x1f, '0
          xcc': 0x4b, '0xcd': 0xbd, '0xce': 0x8b, '0xcf': 0x8a,
56      '0xd0': 0x70, '0xd1': 0x3e, '0xd2': 0xb5, '0xd3': 0x66, '0
          xd4': 0x48, '0xd5': 0x03, '0xd6': 0xf6, '0xd7': 0x0e,
57      '0xd8': 0x61, '0xd9': 0x35, '0xda': 0x57, '0xdb': 0xb9, '0
          xdc': 0x86, '0xdd': 0xc1, '0xde': 0x1d, '0xdf': 0x9e,
58      '0xe0': 0xe1, '0xe1': 0xf8, '0xe2': 0x98, '0xe3': 0x11, '0
          xe4': 0x69, '0xe5': 0xd9, '0xe6': 0x8e, '0xe7': 0x94,
59      '0xe8': 0x9b, '0xe9': 0x1e, '0xea': 0x87, '0xeb': 0xe9, '0
          xec': 0xce, '0xed': 0x55, '0xee': 0x28, '0xef': 0xdf,
60      '0xf0': 0x8c, '0xf1': 0xa1, '0xf2': 0x89, '0xf3': 0x0d, '0
          xf4': 0xbf, '0xf5': 0xe6, '0xf6': 0x42, '0xf7': 0x68,
```

```
61      '0xf8': 0x41, '0xf9': 0x99, '0xfa': 0x2d, '0xfb': 0x0f, '0
            xfc': 0xb0, '0xfd': 0x54, '0xfe': 0xbb, '0xff': 0x16}
62
63
64  def mul(x, y):
65      """计算有限域上的1x,2x,3x"""
66      res = 0
67      if x == 0x01:
68          res = y
69      elif x == 0x02:
70          if (y & 0x80) == 0x00:
71              res = y << 1 & 0b011111111
72          else:
73              res = (y << 1 & 0b011111111) ^ 0x1b
74      elif x == 0x03:
75          res = mul(0x02, y) ^ y
76      return res
77
78
79  def Generate_Table():
80      """生成{2x,x,x,3x}的列混合查找表"""
81      global plaintext, S_BOX, Mix_Columns_Table
82      for i in range(256):
83          j = hex(i)
84          Mix_Columns_Table[i][0] = mul(0x2, S_Box.get(j))
85          Mix_Columns_Table[i][1] = S_Box.get(j)
86          Mix_Columns_Table[i][2] = S_Box.get(j)
87          Mix_Columns_Table[i][3] = mul(0x3, S_Box.get(j))
88
89
90  def Key_Expansion():
91      """生成轮密钥, 用keys (11 X ) 储存"""
```

```
92      global keys, key
93      # 第一轮的密钥是原始密钥
94      for i in range(4):
95          keys[0][0][i] = key[i][0]
96          keys[0][1][i] = key[i][1]
97          keys[0][2][i] = key[i][2]
98          keys[0][3][i] = key[i][3]
99
100     # 剩余 10 轮密钥生成
101     for i in range(1, 11):  # i 表示第 i 个轮密钥
102         # 字循环 + 字代替
103         arr = np.zeros((4,), dtype=np.uint8)
104         arr[0] = S_Box.get(hex(keys[i - 1][1][3]))
105         arr[1] = S_Box.get(hex(keys[i - 1][2][3]))
106         arr[2] = S_Box.get(hex(keys[i - 1][3][3]))
107         arr[3] = S_Box.get(hex(keys[i - 1][0][3]))
108         # 与轮常量异或
109         for k in range(4):
110             arr[k] = arr[k] ^ rcon[i - 1][k]
111         # 再次异或
112         for m in range(4):  # 求 W4
113             keys[i][0][0] = arr[0] ^ keys[i - 1][0][0]
114             keys[i][1][0] = arr[1] ^ keys[i - 1][1][0]
115             keys[i][2][0] = arr[2] ^ keys[i - 1][2][0]
116             keys[i][3][0] = arr[3] ^ keys[i - 1][3][0]
117
118         for s in range(1, 4):  # 生成 W5, W6, W7, s 表示列, t 表示行
119             for t in range(4):
120                 keys[i][t][s] = keys[i - 1][t][s] ^ keys[i][t][
                        s - 1]
121
122
```

```
123 def AddRoundKey(round):
124     """每一轮加密中列混合的结果与轮密钥异或"""
125     global keys, ciphertext
126     for i in range(4):
127         for j in range(4):
128             ciphertext[i][j] = ciphertext[i][j] ^ keys[round][i
                ][j]
129
130
131 def SubBytes():
132     """最后一轮需要用到S盒"""
133     global ciphertext
134     for i in range(4):
135         for j in range(4):
136             ciphertext[i][j] = S_Box.get(hex(ciphertext[i][j]))
137
138
139 def ShiftRows():
140     """行移位"""
141     global ciphertext
142     for i in range(1, 4):
143         brr = np.zeros((4,), dtype=np.uint8)
144         for j in range(4):
145             brr[j] = ciphertext[i][j]
146         if (i == 1):
147             temp = ciphertext[i][0]
148             ciphertext[i][0] = ciphertext[i][1]
149             ciphertext[i][1] = ciphertext[i][2]
150             ciphertext[i][2] = ciphertext[i][3]
151             ciphertext[i][3] = temp
152         elif (i == 2):
153             ciphertext[i][0] = brr[2]
```

```
154             ciphertext[i][1] = brr[3]
155             ciphertext[i][2] = brr[0]
156             ciphertext[i][3] = brr[1]
157         elif (i == 3):
158             ciphertext[i][0] = brr[3]
159             ciphertext[i][1] = brr[0]
160             ciphertext[i][2] = brr[1]
161             ciphertext[i][3] = brr[2]
162
163
164 def MixColumns():
165     """列混合"""
166     global ciphertext
167     for i in range(4):  #i是列循环
168         temp = [ciphertext[0][i], ciphertext[1][i], ciphertext
                [2][i], ciphertext[3][i]]
169         '''
170         ciphertext[0][i] = mul(0x02, temp[0]) ^ mul(0x03, temp
                [1]) ^ mul(0x01, temp[2]) ^ mul(0x01, temp[3])
171         ciphertext[1][i] = mul(0x01, temp[0]) ^ mul(0x02, temp
                [1]) ^ mul(0x03, temp[2]) ^ mul(0x01, temp[3])
172         ciphertext[2][i] = mul(0x01, temp[0]) ^ mul(0x01, temp
                [1]) ^ mul(0x02, temp[2]) ^ mul(0x03, temp[3])
173         ciphertext[3][i] = mul(0x03, temp[0]) ^ mul(0x01, temp
                [1]) ^ mul(0x01, temp[2]) ^ mul(0x02, temp[3])'''
174
175         ciphertext[0][i] = Mix_Columns_Table[temp[0]][0] ^
                Mix_Columns_Table[temp[1]][3] ^ Mix_Columns_Table[
                temp[2]][
176                 2] ^ Mix_Columns_Table[temp[3]][1]
177         ciphertext[1][i] = Mix_Columns_Table[temp[0]][1] ^
                Mix_Columns_Table[temp[1]][0] ^ Mix_Columns_Table[
```

```
            temp[2]][
178             3] ^ Mix_Columns_Table[temp[3]][2]
179         ciphertext[2][i] = Mix_Columns_Table[temp[0]][2] ^
            Mix_Columns_Table[temp[1]][1] ^ Mix_Columns_Table[
            temp[2]][
180             0] ^ Mix_Columns_Table[temp[3]][3]
181         ciphertext[3][i] = Mix_Columns_Table[temp[0]][3] ^
            Mix_Columns_Table[temp[1]][2] ^ Mix_Columns_Table[
            temp[2]][
182             1] ^ Mix_Columns_Table[temp[3]][0]
183
184
185 def Encrypt():
186     """加密函数"""
187
188     #密钥生成
189     Key_Expansion()
190     global plaintext, key, keys, ciphertext, Mix_Columns_Table
191     ciphertext = plaintext.copy()
192     for i in range(4):
193         ciphertext[0][i] = plaintext[i][0]
194         ciphertext[1][i] = key[i][1]
195         ciphertext[2][i] = key[i][2]
196         ciphertext[3][i] = key[i][3]
197     #打印明文
198     print("明文: ")
199     for i in range(4):
200         for j in range(4):
201             print(hex(plaintext[i][j]), end=' ')
202         print("\n")
203
204     #打印密钥
```

```
205     print("密 钥: ")
206     for i in range(4):
207         for j in range(4):
208             print(hex(key[i][j]), end=' ')
209         print("\n")
210 
211     # 生成 11 组轮密钥
212     Key_Expansion()
213 
214     # 第一轮加密
215     AddRoundKey(0)
216 
217     # 第 1 9 轮加密
218     for i in range(1, 10):
219         # SubBytes()
220         ShiftRows()
221         MixColumns()
222         AddRoundKey(i)
223 
224     # 第 10 轮加密
225     SubBytes()
226     ShiftRows()
227     AddRoundKey(10)
228 
229     # 打印密文
230     print("密 文: ")
231     for i in range(4):
232         for j in range(4):
233             print(hex(ciphertext[i][j]), end=' ')
234         print("\n")
235 
236 
```

```
237 if __name__ == '__main__':
238     # 生成查找表
239     Generate_Table()
240     Encrypt()
```

5.2 第一题 AES 实现 (C)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdint.h>
4
5  uint8_t S_BOX[256] = {
6          0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67
               ,0x2B,0xFE,0xD7,0xAB,0x76,
7          0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2
               ,0xAF,0x9C,0xA4,0x72,0xC0,
8          0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5
               ,0xF1,0x71,0xD8,0x31,0x15,
9          0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80
               ,0xE2,0xEB,0x27,0xB2,0x75,
10         0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6
               ,0xB3,0x29,0xE3,0x2F,0x84,
11         0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE
               ,0x39,0x4A,0x4C,0x58,0xCF,
12         0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02
               ,0x7F,0x50,0x3C,0x9F,0xA8,
13         0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA
               ,0x21,0x10,0xFF,0xF3,0xD2,
14         0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E
               ,0x3D,0x64,0x5D,0x19,0x73,
15         0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8
               ,0x14,0xDE,0x5E,0x0B,0xDB,
```

```
16          0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC
                ,0x62,0x91,0x95,0xE4,0x79,
17          0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4
                ,0xEA,0x65,0x7A,0xAE,0x08,
18          0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74
                ,0x1F,0x4B,0xBD,0x8B,0x8A,
19          0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57
                ,0xB9,0x86,0xC1,0x1D,0x9E,
20          0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87
                ,0xE9,0xCE,0x55,0x28,0xDF,
21          0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D
                ,0x0F,0xB0,0x54,0xBB,0x16  };
22
23  uint8_t S_INV[256] = {
24          0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3
                ,0x9e,0x81,0xf3,0xd7,0xfb,
25          0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43
                ,0x44,0xc4,0xde,0xe9,0xcb,
26          0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95
                ,0x0b,0x42,0xfa,0xc3,0x4e,
27          0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2
                ,0x49,0x6d,0x8b,0xd1,0x25,
28          0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c
                ,0xcc,0x5d,0x65,0xb6,0x92,
29          0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46
                ,0x57,0xa7,0x8d,0x9d,0x84,
30          0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58
                ,0x05,0xb8,0xb3,0x45,0x06,
31          0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd
                ,0x03,0x01,0x13,0x8a,0x6b,
32          0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf
                ,0xce,0xf0,0xb4,0xe6,0x73,
```

```
33          0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37
                ,0xe8,0x1c,0x75,0xdf,0x6e,
34          0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62
                ,0x0e,0xaa,0x18,0xbe,0x1b,
35          0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0
                ,0xfe,0x78,0xcd,0x5a,0xf4,
36          0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10
                ,0x59,0x27,0x80,0xec,0x5f,
37          0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a
                ,0x9f,0x93,0xc9,0x9c,0xef,
38          0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb
                ,0x3c,0x83,0x53,0x99,0x61,
39          0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14
                ,0x63,0x55,0x21,0x0c,0x7d };
40
41  uint8_t Rcon[40] = {
42                  0x01, 0x00, 0x00, 0x00,
43                  0x02, 0x00, 0x00, 0x00,
44                  0x04, 0x00, 0x00, 0x00,
45                  0x08, 0x00, 0x00, 0x00,
46                  0x10, 0x00, 0x00, 0x00,
47                  0x20, 0x00, 0x00, 0x00,
48                  0x40, 0x00, 0x00, 0x00,
49                  0x80, 0x00, 0x00, 0x00,
50                  0x1B, 0x00, 0x00, 0x00,
51                  0x36, 0x00, 0x00, 0x00 };   // 按列向量排
52
53  uint8_t key[16] = {
54          0x32,0x30,0x32,0x31,
55          0x30,0x30,0x34,0x36,
56          0x30,0x30,0x35,0x35,
57          0x00,0x00,0x00,0x00 };
```

```
58
59  uint8_t plain_text[16] = {
60          0x32,0x30,0x32,0x31,
61          0x30,0x30,0x34,0x36,
62          0x30,0x30,0x35,0x35,
63          0x00,0x00,0x00,0x00 };
64  uint8_t cipher_text[16];
65  uint8_t keys[11][16];
66  uint8_t mix_table[256][4];
67
68
69  uint8_t mul(uint8_t x, uint8_t y)
70  {
71          uint8_t res = 0;
72          if (x == 0x01) {
73                  res = y;
74          }
75          else if (x == 0x02) {
76                  res = (y & 0x80) ? ((y << 1) ^ 0b00011011) : (y
                        << 1);
77          }
78          else if (x == 0x03) {
79                  res = mul(0x02, y) ^ y;
80          }
81          else if (x == 0x09) {
82                  res = mul(0x02, mul(0x02, mul(0x02, y))) ^ y;
83          }
84          else if (x == 0x0B) {
85                  res = mul(0x02, mul(0x02, mul(0x02, y))) ^ mul
                        (0x02, y) ^ y;
86          }
87          else if (x == 0x0D) {
```

```
88                  res = mul(0x02, mul(0x02, mul(0x02, y))) ^ mul
                        (0x02, mul(0x02, y)) ^ y;
89          }
90          else if (x == 0x0E) {
91                  res = mul(0x02, mul(0x02, mul(0x02, y))) ^ mul
                        (0x02, mul(0x02, y)) ^ mul(0x02, y);
92          }
93          return res;
94  }
95
96  void Generate_Table() {
97          for (int i = 0; i < 256; i++) {
98                  mix_table[i][0] = mul(0x02, S_BOX[i]);
99                  mix_table[i][1] = S_BOX[i];
100                 mix_table[i][2] = S_BOX[i];
101                 mix_table[i][3] = mul(0x03, S_BOX[i]);
102         }
103 }
104
105 void KeyExpansion() {
106
107         for (int i = 0; i < 16; i++) {
108                 keys[0][i] = key[i];
109         }
110         size_t index = 0;
111         for (int i = 1; i < 11; i++) {
112                 for (int j = 0; j < 16; j++) {
113                         if (j >= 0 && j <= 3) {
114                                 uint8_t arr[4] = { keys[i -
                                    1][13],keys[i - 1][14],keys[
                                    i - 1][15],keys[i - 1][12]
                                    };
```

```
115                                 uint8_t brr[4] = { S_BOX[arr
                                        [0]],S_BOX[arr[1]],S_BOX[arr
                                        [2]],S_BOX[arr[3]] };
116                                 for (int k = 0; k < 4; k++) {
117                                         keys[i][k] = keys[i -
                                            1][k] ^ brr[k] ^
                                            Rcon[index];
118                                         index += 1;
119                                 }
120                                 j = 3;
121                         }
122                         else {
123                                 keys[i][j] = keys[i - 1][j] ^
                                    keys[i][j - 4];
124                         }
125                 }
126         }
127 }
128
129 void AddRoundKey(uint8_t* mat, uint8_t* key_mat) {
130         for (int i = 0; i < 16; i++) {
131                 mat[i] ^= key_mat[i];
132         }
133 }
134
135 void SubBytes(uint8_t* mat) {
136         for (int i = 0; i < 16; i++) {
137                 mat[i] = S_BOX[mat[i]];
138         }
139 }
140 void InvSubBytes(uint8_t* mat) {
141         for (int i = 0; i < 16; i++)
```

```
142                 mat[i] = S_INV[mat[i]];
143 }
144 void ShiftRows(uint8_t* mat) {
145         uint8_t temp[16];
146         temp[0] = mat[0]; temp[4] = mat[4]; temp[8] = mat[8];
              temp[12] = mat[12];
147         temp[1] = mat[5]; temp[5] = mat[9];       temp[9] = mat
              [13]; temp[13] = mat[1];
148         temp[2] = mat[10]; temp[6] = mat[14]; temp[10] = mat
              [2]; temp[14] = mat[6];
149         temp[3] = mat[15]; temp[7] = mat[3];      temp[11] = mat
              [7]; temp[15] = mat[11];
150         for (int i = 0; i < 16; i++) {
151                 mat[i] = temp[i];
152         }
153 }
154 void InvShiftRows(uint8_t* mat) {
155         uint8_t temp[16];
156         temp[0] = mat[0];         temp[4] = mat[4];         temp[8]
              = mat[8];         temp[12] = mat[12];
157         temp[1] = mat[13];        temp[5] = mat[1];         temp[9]
              = mat[5];         temp[13] = mat[9];
158         temp[2] = mat[10];        temp[6] = mat[14];        temp
              [10] = mat[2];        temp[14] = mat[6];
159         temp[3] = mat[7];         temp[7] = mat[11];        temp
              [11] = mat[15];       temp[15] = mat[3];
160         for (int i = 0; i < 16; i++)
161                 mat[i] = temp[i];
162 }
163 void MixColumns(uint8_t* mat) {
164         uint8_t temp[16];
165         for (int i = 0; i < 16; i++) {
```

```
166                 uint8_t index1 = (i + 1) % 4 == 0 ? (i + 1 - 4)
                        : (i + 1);
167                 uint8_t index2 = (index1 + 1) % 4 == 0 ? (
                        index1 + 1 - 4) : (index1 + 1);
168                 uint8_t index3 = (index2 + 1) % 4 == 0 ? (
                        index2 + 1 - 4) : (index2 + 1);
169                 temp[i] = mix_table[mat[i]][0] ^ mix_table[mat[
                        index1]][3] ^ mix_table[mat[index2]][2] ^
                        mix_table[mat[index3]][1];
170         }
171         for (int i = 0; i < 16; i++)
172                 mat[i] = temp[i];
173 }
174 void InvMixColumns(uint8_t* mat)
175 {
176         uint8_t temp[16];
177         for (int i = 0; i < 16; i++) {
178                 uint8_t index1 = (i + 1) % 4 == 0 ? (i + 1 - 4)
                        : (i + 1);
179                 uint8_t index2 = (index1 + 1) % 4 == 0 ? (
                        index1 + 1 - 4) : (index1 + 1);
180                 uint8_t index3 = (index2 + 1) % 4 == 0 ? (
                        index2 + 1 - 4) : (index2 + 1);
181                 temp[i] = mul(0x0E, mat[i]) ^ mul(0x0B, mat[
                        index1]) ^ mul(0x0D, mat[index2]) ^ mul(0x09
                        , mat[index3]);
182         }
183         for (int i = 0; i < 16; i++)
184                 mat[i] = temp[i];
185 }
186 void Encrypt() {
187         printf("原明文为: \n");
```

```
188         for (int i = 0; i < 16; i++) {
189                 printf("%02x ", plain_text[i]);
190         }printf("\n");
191 
192         for (int i = 0; i < 16; i++) {
193                 cipher_text[i] = plain_text[i];
194         }
195         KeyExpansion();
196         AddRoundKey(cipher_text, keys[0]);
197         for (int i = 1; i < 10; i++) {
198                 ShiftRows(cipher_text);
199                 MixColumns(cipher_text);//优化
200                 AddRoundKey(cipher_text, keys[i]);
201         }
202         SubBytes(cipher_text);
203         ShiftRows(cipher_text);
204         AddRoundKey(cipher_text, keys[10]);
205         printf("加密后的密文为: \n");
206         for (int i = 0; i < 16; i++) {
207                 printf("%02x ", cipher_text[i]);
208         }
209 }
210 void Decrypt() {
211         for (int i = 0; i < 16; i++) {
212                 plain_text[i] = cipher_text[i];
213         }
214         AddRoundKey(plain_text, keys[10]);
215         InvShiftRows(plain_text);
216         InvSubBytes(plain_text);
217         for (int i = 9; i > 0; i--) {
218                 AddRoundKey(plain_text, keys[i]);
219                 InvMixColumns(plain_text);
```

```
220                     InvShiftRows(plain_text);
221                     InvSubBytes(plain_text);
222             }
223             AddRoundKey(plain_text, keys[0]);
224             printf("解密后的明文为: \n");
225             for (int i = 0; i < 16; i++) {
226                     printf("%02x ", plain_text[i]);
227             }
228     }
229
230
231     int main() {
232             Generate_Table();
233
234             Encrypt();
235             printf("\n");
236             Decrypt();
237             return 0;
238     }
```

5.3 第二题 AES 手动实现 (C)

```
1   #include <stdio.h>
2   #include <string.h>
3   #include <stdint.h>
4   #include <stdlib.h>
5   #include <time.h>
6
7   uint8_t S_BOX[256] = {
8           0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67
                ,0x2B,0xFE,0xD7,0xAB,0x76,
9           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 };

uint8_t S_INV[256] = {
    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,0x77,0xd6,0x26,0xe1,0x69,0x14
    ,0x63,0x55,0x21,0x0c,0x7d };
```

```
43 uint8_t Rcon[40] = {
44                 0x01, 0x00, 0x00, 0x00,
45                 0x02, 0x00, 0x00, 0x00,
46                 0x04, 0x00, 0x00, 0x00,
47                 0x08, 0x00, 0x00, 0x00,
48                 0x10, 0x00, 0x00, 0x00,
49                 0x20, 0x00, 0x00, 0x00,
50                 0x40, 0x00, 0x00, 0x00,
51                 0x80, 0x00, 0x00, 0x00,
52                 0x1B, 0x00, 0x00, 0x00,
53                 0x36, 0x00, 0x00, 0x00 };   // 按列向量排
54
55 uint8_t key[16] = {
56         0x32,0x30,0x32,0x31,
57         0x30,0x30,0x34,0x36,
58         0x30,0x30,0x35,0x35,
59         0x00,0x00,0x00,0x00 };
60
61 uint8_t plain_text_table[100][256][16];
62
63 uint8_t keys[11][16];
64 uint8_t mix_table[256][4];
65 uint8_t cipher_text[16];
66
67 uint8_t mul(uint8_t x, uint8_t y)
68 {
69         uint8_t res = 0;
70         if (x == 0x01) {
71                 res = y;
72         }
73         else if (x == 0x02) {
74                 res = (y & 0x80) ? ((y << 1) ^ 0b00011011) : (y
```

```
                    << 1);
        }
        else if (x == 0x03) {
                res = mul(0x02, y) ^ y;
        }
        else if (x == 0x09) {
                res = mul(0x02, mul(0x02, mul(0x02, y))) ^ y;
        }
        else if (x == 0x0B) {
                res = mul(0x02, mul(0x02, mul(0x02, y))) ^ mul
                    (0x02, y) ^ y;
        }
        else if (x == 0x0D) {
                res = mul(0x02, mul(0x02, mul(0x02, y))) ^ mul
                    (0x02, mul(0x02, y)) ^ y;
        }
        else if (x == 0x0E) {
                res = mul(0x02, mul(0x02, mul(0x02, y))) ^ mul
                    (0x02, mul(0x02, y)) ^ mul(0x02, y);
        }
        return res;
}

void GenerateText() {
        srand(time(NULL));
        for (int i = 0; i < 100; i++) {
                for (int j = 0; j < 256; j++) {
                        plain_text_table[i][j][0] = j;
                        for (int k = 1; k < 16; k++) {
                                plain_text_table[i][j][k] =
                                    rand() % 255;
                        }
```

```
102                     }
103             }
104     }
105
106     void Generate_Table() {
107             for (int i = 0; i < 256; i++) {
108                     mix_table[i][0] = mul(0x02, S_BOX[i]);
109                     mix_table[i][1] = S_BOX[i];
110                     mix_table[i][2] = S_BOX[i];
111                     mix_table[i][3] = mul(0x03, S_BOX[i]);
112             }
113     }
114
115     void KeyExpansion() {
116
117             for (int i = 0; i < 16; i++) {
118                     keys[0][i] = key[i];
119             }
120             size_t index = 0;
121             for (int i = 1; i < 11; i++) {
122                     for (int j = 0; j < 16; j++) {
123                             if (j >= 0 && j <= 3) {
124                                     uint8_t arr[4] = { keys[i -
                                        1][13],keys[i - 1][14],keys[
                                        i - 1][15],keys[i - 1][12]
                                        };
125                                     uint8_t brr[4] = { S_BOX[arr
                                        [0]],S_BOX[arr[1]],S_BOX[arr
                                        [2]],S_BOX[arr[3]] };
126                                     for (int k = 0; k < 4; k++) {
127                                             keys[i][k] = keys[i -
                                                1][k] ^ brr[k] ^
```

```
                                        Rcon[index];
128                                     index += 1;
129                                 }
130                                 j = 3;
131                             }
132                             else {
133                                 keys[i][j] = keys[i - 1][j] ^
                                        keys[i][j - 4];
134                             }
135                         }
136                 }
137 }
138
139 void AddRoundKey(uint8_t* mat, uint8_t* key_mat) {
140         for (int i = 0; i < 16; i++) {
141                 mat[i] ^= key_mat[i];
142         }
143 }
144
145 void SubBytes(uint8_t* mat) {
146         for (int i = 0; i < 16; i++) {
147                 mat[i] = S_BOX[mat[i]];
148         }
149 }
150 void InvSubBytes(uint8_t* mat) {
151         for (int i = 0; i < 16; i++)
152                 mat[i] = S_INV[mat[i]];
153 }
154 void ShiftRows(uint8_t* mat) {
155         uint8_t temp[16];
156         temp[0] = mat[0]; temp[4] = mat[4]; temp[8] = mat[8];
            temp[12] = mat[12];
```

```
157         temp[1] = mat[5]; temp[5] = mat[9];      temp[9] = mat
                [13]; temp[13] = mat[1];
158         temp[2] = mat[10]; temp[6] = mat[14]; temp[10] = mat
                [2]; temp[14] = mat[6];
159         temp[3] = mat[15]; temp[7] = mat[3];     temp[11] = mat
                [7]; temp[15] = mat[11];
160         for (int i = 0; i < 16; i++) {
161                 mat[i] = temp[i];
162         }
163 }
164 void InvShiftRows(uint8_t* mat) {
165         uint8_t temp[16];
166         temp[0] = mat[0];       temp[4] = mat[4];       temp[8]
                = mat[8];       temp[12] = mat[12];
167         temp[1] = mat[13];      temp[5] = mat[1];       temp[9]
                = mat[5];       temp[13] = mat[9];
168         temp[2] = mat[10];      temp[6] = mat[14];      temp
                [10] = mat[2];      temp[14] = mat[6];
169         temp[3] = mat[7];       temp[7] = mat[11];      temp
                [11] = mat[15];     temp[15] = mat[3];
170         for (int i = 0; i < 16; i++)
171                 mat[i] = temp[i];
172 }
173 void MixColumns(uint8_t* mat) {
174
175         uint8_t temp[16];
176         for (int i = 0; i < 16; i++) {
177                 uint8_t index1 = (i + 1) % 4 == 0 ? (i + 1 - 4)
                        : (i + 1);
178                 uint8_t index2 = (index1 + 1) % 4 == 0 ? (
                    index1 + 1 - 4) : (index1 + 1);
179                 uint8_t index3 = (index2 + 1) % 4 == 0 ? (
```

```
                    index2 + 1 - 4) : (index2 + 1);
180                 temp[i] = mix_table[mat[i]][0] ^ mix_table[mat[
                    index1]][3] ^ mix_table[mat[index2]][2] ^
                    mix_table[mat[index3]][1];
181         }
182         for (int i = 0; i < 16; i++)
183                 mat[i] = temp[i];
184
185 }
186 void InvMixColumns(uint8_t* mat)
187 {
188         uint8_t temp[16];
189         for (int i = 0; i < 16; i++) {
190                 uint8_t index1 = (i + 1) % 4 == 0 ? (i + 1 - 4)
                        : (i + 1);
191                 uint8_t index2 = (index1 + 1) % 4 == 0 ? (
                    index1 + 1 - 4) : (index1 + 1);
192                 uint8_t index3 = (index2 + 1) % 4 == 0 ? (
                    index2 + 1 - 4) : (index2 + 1);
193                 temp[i] = mul(0x0E, mat[i]) ^ mul(0x0B, mat[
                    index1]) ^ mul(0x0D, mat[index2]) ^ mul(0x09
                    , mat[index3]);
194         }
195         for (int i = 0; i < 16; i++)
196                 mat[i] = temp[i];
197 }
198 void Encrypt(uint8_t* plain_text) {
199         /*
200         printf("原明文为: \n");
201         for (int i = 0; i < 16; i++) {
202                 printf("%02x ", plain_text[i]);
203         }printf("\n");*/
```

```
204
205         for (int i = 0; i < 16; i++) {
206                 cipher_text[i] = plain_text[i];
207         }
208         KeyExpansion();
209         AddRoundKey(cipher_text, keys[0]);
210         for (int i = 1; i < 10; i++) {
211                 ShiftRows(cipher_text);
212                 MixColumns(cipher_text);//优化
213                 AddRoundKey(cipher_text, keys[i]);
214         }
215         SubBytes(cipher_text);
216         ShiftRows(cipher_text);
217         AddRoundKey(cipher_text, keys[10]);
218         /*
219         printf("加密后的密文为: \n");
220         for (int i = 0; i < 16; i++) {
221                 printf("%02x ", cipher_text[i]);
222         }*/
223 }
224 void Decrypt(uint8_t* plain_text) {
225         for (int i = 0; i < 16; i++) {
226                 plain_text[i] = cipher_text[i];
227         }
228         AddRoundKey(plain_text, keys[10]);
229         InvShiftRows(plain_text);
230         InvSubBytes(plain_text);
231         for (int i = 9; i > 0; i--) {
232                 AddRoundKey(plain_text, keys[i]);
233                 InvMixColumns(plain_text);
234                 InvSubBytes(plain_text);
235                 InvShiftRows(plain_text);
```

```
236         }
237         AddRoundKey(plain_text, keys[0]);
238         printf("解密后的明文为: \n");
239         for (int i = 0; i < 16; i++) {
240                 printf("%02x ", plain_text[i]);
241         }
242 }
243
244
245
246 int main() {
247         Generate_Table();
248         GenerateText();
249
250         clock_t t;
251         t = clock();
252         for (int i = 0; i < 100; i++) {
253                 for (int j = 0; j < 256; j++) {
254                         Encrypt(plain_text_table[i][j]);
255                 }
256         }
257         printf("time:%fs\n", (double)(clock() - t) /
              CLOCKS_PER_SEC);
258         return 0;
259 }
```

5.4 第二题 mbedTLS 库 (C)

```
1 #include <string.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
```

```
5  #include "mbedtls/aes.h"
6
7
8  uint8_t plain_text_table[100][256][16];
9
10 uint8_t key[16] = {
11         0x32,0x30,0x32,0x31,
12         0x30,0x30,0x34,0x36,
13         0x30,0x30,0x35,0x35,
14         0x00,0x00,0x00,0x00 };
15
16
17 void GenerateText() {
18         srand(time(NULL));
19         for (int i = 0; i < 100; i++) {
20                 for (int j = 0; j < 256; j++) {
21                         plain_text_table[i][j][0] = j;
22                         for (int k = 1; k < 16; k++) {
23                                 plain_text_table[i][j][k] =
                                       rand() % 255;
24                         }
25                 }
26         }
27 }
28
29 void AES_mbedtls(uint8_t* plain_text) {
30         uint8_t output1[16];
31         uint8_t output2[16];
32         mbedtls_aes_context aes_ctx;
33         int ret;
34
35         /*
```

```
36          //Encrypt
37          printf("原明文为: \n");
38          for (int i = 0; i < 16; i++) {
39                  printf("%02x ", plain_text[i]);
40          }printf("\n");*/
41
42
43          mbedtls_aes_init(&aes_ctx);
44          ret = mbedtls_aes_setkey_enc(&aes_ctx, key, sizeof(key)
                * 8);
45          if (ret != 0) {
46                  goto exit;
47          }
48          ret = mbedtls_aes_crypt_ecb(&aes_ctx,
              MBEDTLS_AES_ENCRYPT, plain_text, output1);
49          if (ret != 0) {
50                  goto exit;
51          }
52          /*
53          printf("加密后的密文为: \n");
54          for (int i = 0; i < 16; i++) {
55                  printf("%02x ", output1[i]);
56          }printf("\n");
57
58          //Decrypt
59          ret = mbedtls_aes_setkey_dec(&aes_ctx, key, sizeof(key)
                * 8);
60          if (ret != 0) {
61                  goto exit;
62          }
63          ret = mbedtls_aes_crypt_ecb(&aes_ctx,
              MBEDTLS_AES_DECRYPT, output1, output2);
```

```
64          if (ret != 0) {
65                  goto exit;
66          }
67
68          printf("解密后的密文为: \n");
69          for (int i = 0; i < 16; i++) {
70                  printf("%02x ", output2[i]);
71          }*/
72  exit:
73          mbedtls_aes_free(&aes_ctx);
74  }
75
76  int main() {
77          GenerateText();
78
79          clock_t t;
80          t = clock();
81          for (int i = 0; i < 100; i++) {
82                  for (int j = 0; j < 256; j++) {
83                          AES_mbedtls(plain_text_table[i][j]);
84                  }
85          }
86          printf("time:%fs\n", (double)(clock() - t) /
               CLOCKS_PER_SEC);
87          return 0;
88  }
```

参考文献

[1]. Douglas R. Stinson 道格拉斯 R. 斯廷森著，冯登国等译，密码学原理与实践（第三版），电子工业出版社，2016.01

[2]. William Stallings 威廉 · 斯托林斯，王张宜、杨敏、杜瑞颖等译，密码编码学与网络安全：原理与实践（第五版），电子工业出版社，2014.01

[3]. 冯登国裴定一，密码学导引，科学出版社，1999.04