武汉大学国家网络安全学院

课程实验报告

分组密码 AES

专 业 、 班	：	信安 9 班
课 程 名 称	：	密码学实验
指 导 教 师	：	王后珍
实 验 地 点	：	C102
学 生 学 号	：	2021302141097
学 生 姓 名	：	李宇

2023 年 **12** 月 **10** 日

分组密码 AES 的实现

一、 实验目的

1. 掌握分组密码的基本概念；
2. 掌握 AES 密码算法；
3. 了解 AES 密码的安全性；
4. 掌握分组密码常用工作模式及其特点；
5. 熟悉分组密码的应用。

二、 实验内容与过程分析

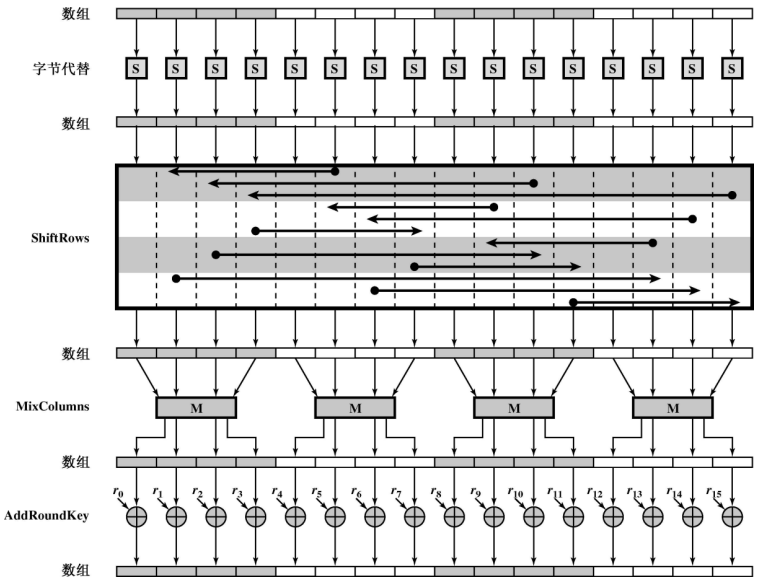
任意选用编程语言，完成分组密码 AES 的实现，并给出实现原理。

AES 的实现分为加密和解密过程，由于 AES 算法不是对合运算，因此需要分别针对加密和解密过程进行代码实现。下面给出算法实现的整体流程，具体的原理分析放在实验步骤部分，结合代码进行分析。

2.1 AES 加密过程

类似于 DES 的实现，AES 实现过程也可分为多个组件，分别是：**状态初始化**、**轮密钥产生算法**、**轮密钥加变换**、**S 盒变换**、**行移位变换**、**列混合变换过程**。

实现了上述组件后，只需要按照课本流程，向这些组件的接口中输入当前状态或密钥即可成功完成加密。实现流程如下图所示：



2.2 AES 解密过程

虽然 AES 不是对合运算，但解密过程和加密过程并不是完全不一样，需要将**轮密钥产生算法、S 盒变换、行移位变换、列混合变换过程变为其逆，同时调整密钥使用顺序即可**，具体的差异之处会在后文进行分析。

三、 实验环境

- 操作系统：Windows11 家庭版
- 基本硬件：CPU：AMD Ryzen 5 5600H, 16G 内存
- 所用软件：Pycharm 2023.2.1 专业版
- 所用语言：python3.10

四、 实验步骤与分析

4.1 AES 加密过程

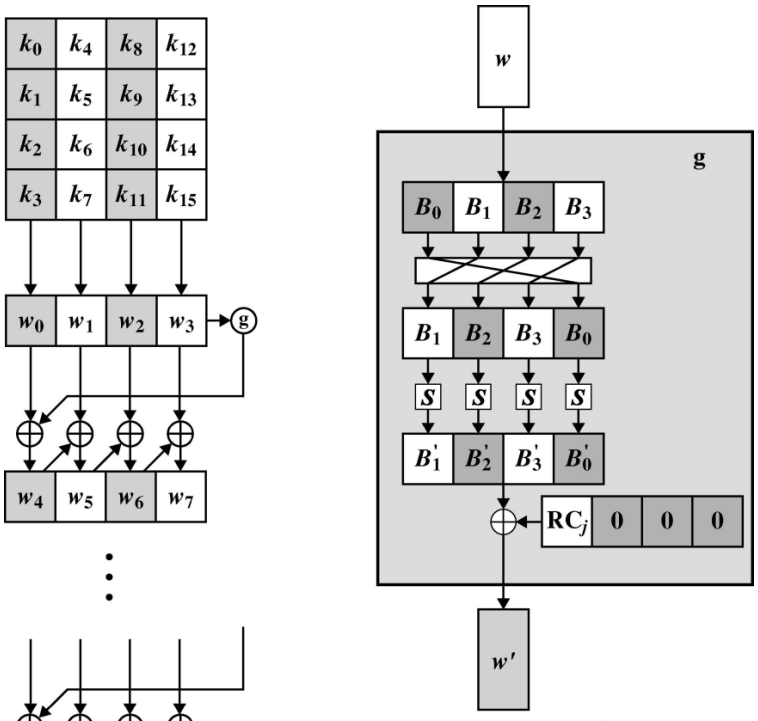
4.1.1 状态初始化

此部分为笔者为了方便后续实现，将原本的算法预处理模块整体规整为了状态初始化。由于输入的明文一般是 128 比特 2 进制数、128 比特 2 进制字符、或者是 16 个以 16 进制表示的字节等，而 AES 在处理过程中使用的是 state，也就是状态，所以在这一步对输入的原始明文进行转化，转换为对应的状态，每一列为一个字，共有数据块长度/32 数量的列。代码如下所示：

```
def init_state(plaintext,Nb):
    state = [[0] * 4for _ in range(Nb)]
    for i in range(4):
        for j in range(4):
            state[j][i] = plaintext[i * 4+ j]
    return state
```

4.1.2 轮密钥产生算法

每一轮的密钥都是通过原始密钥进行一些特定的运算生成的。这些运算包括字节替换（SubBytes）、行移位（ShiftRows）和轮常量异或（AddRoundKey）。这些运算中，字节替换是通过一个替代盒（S-Box）映射生成最终状态数组的新值，其中高四位（行数）和低四位（列数）作为索引，在 S-Box 中查找对应的字节进行替换。行移位则是交换行元素之间的位置。最后，轮常量异或使用给定的常量与字节代换后的结果进行异或。整体流程图如下所示：



(a) 总体算法

(b) 函数g

根据上述分析与流程图，编写轮密钥产生代码如下：

```
def key_expansion(key, Nk, Nb, Nr):
   W = [[0 for _ in range(4)] for _ in range(Nb * (Nr + 1))]
   for i in range(Nk):
      for j in range(4):
         W[i][j] = key[j + 4* i]

   for I in range(Nk, Nb * (Nr + 1)):
      temp = W[I - 1][:]
      if I % Nk == 0:
         temp=Rotl(temp)
         sub_bytes(temp,'key')
         temp = list_xor(temp, Rcon(I // Nk-1))
      elif I % Nk == 4and Nk > 6:
         sub_bytes(temp,'key')
      W[I] = list_xor(W[I - Nk][:], temp)

   return W
```

其中使用到了 Rotl 函数、list_xor 函数、Rcon 函数，sub_bytes 函数，前三个函数是专门为密钥产生

过程服务的，最后一个函数是 S 盒变换函数，关于它的分析后续会有专门的章节，在这里主要分析前三个函数的功能与实现。

Rotl 函数实际上完成了 Rotl 变换，本质上就是一次循环移位操作，实现代码如下：

```
def Rotl(temp):
    return [temp[1], temp[2], temp[3], temp[0]]
```

list_xor 函数是为了便利两个 list 类型变量进行异或运算而编写的，给出两个 list，list_xor 函数可完成对这两个 list 中的元素进行逐元素异或，实现代码如下：

```
def list_xor(a, b):
    if len(a) != len(b):
        raise ValueError("list长度不符")
    r = [x ^ y for x, y in zip(a, b)]
    return r
```

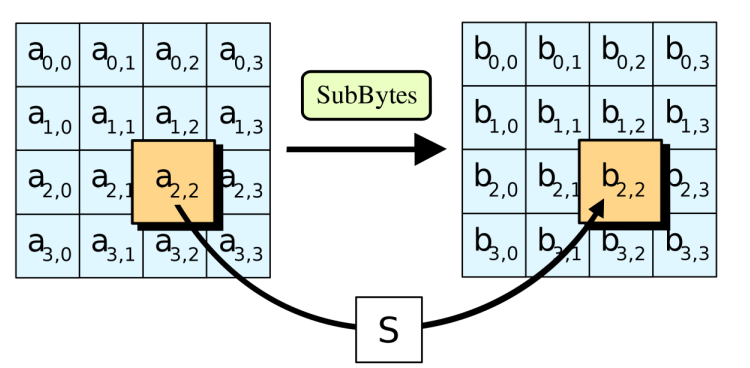
Rcon 函数是为了防止不同轮的轮密钥存在相似性，实现代码如下：

```
def Rcon(i):
    return [RC(i), 0, 0, 0]

def RC(i):
    if i == 0:
        return 1
    else:
        return xtime(RC(i - 1))
```

4.1.3 S 盒变换

S 盒变换也就是 SubBytes，矩阵中的各字节透过一个 8 位的 S-box 进行转换。这个步骤提供了加密法非线性的变换能力。S-box 与 $GF(2^8)$ 上的乘法逆元素有关，已知具有良好的非线性特性。为了避免简单代数性质的攻击，S-box 结合了乘法反元素及一个可逆的仿射变换矩阵建构而成。



在实现中，采用空间换时间的方法，利用课本 92 页的 S 盒表进行该过程，根据输入字节的高低位

可直接获得 S 盒变换的结果，实现代码如下：

```
# AES S-Box
s_box = [..... ] #内容过多，且课本上有

def sub_bytes(state,x):
   if x=='state':
      for i in range(4):
         for j in range(4):
            state[i][j] = s_box[state[i][j]]
   elif x=='key':
      for i in range(4):
         state[i] = s_box[state[i]]
```

4.1.4 行移位变换

行移位变换也就是 ShiftRows，它描述了矩阵的列操作。在此步骤中，每一列都向左循环位移某个偏移量。具体偏移量大小需要根据 Nb(数据块字长度) 确定，如下表所示：

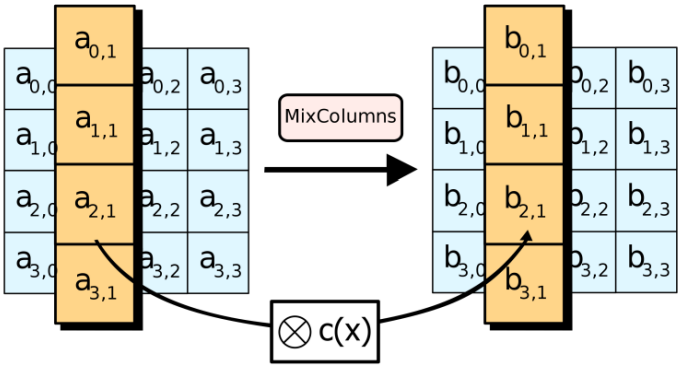
Nb	C1	C2	C3
4	1	2	3
6	1	3	3
8	1	3	4

实现代码如下所示：

```
def shift_rows(state, Nb):
   if Nb == 4or Nb == 6:
      bias = [0, 1, 2, 3]
   elif Nb == 6:
      bias = [0, 1, 3, 4]
   for i in range(1, 4):
      state[i] = state[i][bias[i]:] + state[i][:bias[i]]
```

4.1.5 列混合变换

列混合变换，也就是 MixColumn，是对状态的列进行混合变换，在 MixColumn 变换中，把状态的每一列看做 $GF(2^8)$ 撒谎给你的多项式，并与一个固定的多项式从 $c(x) =' 03'x^3 +' 01'x^2 +' 01'x +' 02'$ 相乘，之后模多项式 $x^4 + 1$。MixColumn 函数接受 4 个字节的输入，输出 4 个字节，每一个输入的字节都会对输出的四个字节造成影响。



由于 state 存在多列，每一列是一个字的 4 个字节，因此需要通过循环分别处理状态的每一列，实现如下所示：

```python
# 列混淆
def mix_columns(state):
    for i in range(4):
        col = [state[j][i] for j in range(4)]
        col = mix_single_column(col)
        for j in range(4):
            state[j][i] = col[j]

# 单列混淆
def mix_single_column(col):
    a0, a1, a2, a3 = col[0], col[1], col[2], col[3]
    b0, b1, b2, b3 = 2, 1, 1, 3
    c0 = get_c(a0,b0) ^ get_c(a3,b1) ^ get_c(a2,b2) ^ get_c(a1,b3)
    c1 = get_c(a1,b0) ^ get_c(a0,b1)^ get_c(a3,b2)^ get_c(a2,b3)
    c2 = get_c(a2,b0) ^ get_c(a1,b1) ^ get_c(a0,b2)^ get_c(a3,b3)
    c3 = get_c(a3,b0) ^ get_c(a2,b1) ^ get_c(a1,b2) ^ get_c(a0,b3)
    col = [c0, c1, c2, c3]
    return col
```

其中 get_c 函数实际上就是完成两个 $GF(2^8)$ 域上多项式的相乘运算，在这里借助了 xtime 函数，例如当 b0 为 “01” 时，a0 与 b0 相乘的结果一定还是 a0；当 b0 为 “02” 时，a0 与 b0 相乘实际上就是 a0 与 x 相乘，结果等同于 xtime(a)；当 b0 为 “03” 时，a0 与 b0 相乘实际上就是 a0 与 x 相乘的结果加上 a0 本身，结果等同于 xtime(a) 异或 a。按照此思路，无论 b0 等于多少，都可以通过 xtime 函数完成 $GF(2^8)$ 域上多项式的相乘运算。

```python
def get_c(a,b):
    if b==1:
        return a
    elif b==2:
        return xtime(a)
```

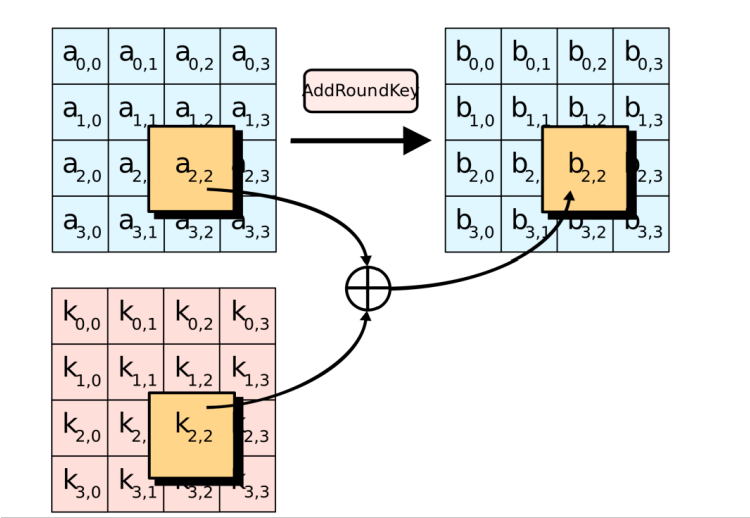
```
    elif b==3:
        return xtime(a)^a
    elif b==9:
        return xtime(xtime(xtime(a)))^a
    elif b==11:
        return xtime(xtime(xtime(a)))^a^xtime(a)
    elif b==13:
        return xtime(xtime(xtime(a))) ^ xtime(xtime(a)) ^a
    elif b==14:
        return xtime(xtime(xtime(a))) ^ xtime(xtime(a)) ^ xtime(a)
```

其中 xtime 的实现如下所示：

```
# 有限域上的乘法
def xtime(x):
    return ((x «1) ^ (0x11B if (x & 0x80) else 0x00)) & 0xFF
```

4.1.6 轮密钥加变换

轮密钥加变换（AddRoundKey）需要使用准备好的密钥与状态进行异或运算，并作为下一轮的新状态，轮密钥长度等于数据块长度。在这个操作，轮密钥被简单地异或到状态中去。



实现代码如下所示：

```
def add_round_key(state, round_key):
    for i in range(4):
        for j in range(4):
            state[j][i] ^= round_key[i][j]
```

4.1.7 AES 加密整体流程

在完成了上述所有组件后，按照课本流程将他们依次组合即可完成加密，如下代码所示：

```
def aes_encrypt(plaintext, key):
    Nb = int(len(plaintext) / 4)
    Nk = int(len(key) / 4)
    Nr = get_Nr(Nb, Nk)

    state=init_state(plaintext,Nb)
    round_keys = key_expansion(key, Nk, Nb, Nr)

    add_round_key(state, round_keys[0:4][:])

    for round_num in range(1, Nr):
        sub_bytes(state,'state')
        shift_rows(state, Nb)
        mix_columns(state)
        add_round_key(state, round_keys[4 * round_num:4 * round_num + 4][:])

    # 最后一轮
    round_num += 1
    sub_bytes(state,'state')
    shift_rows(state, Nb)
    add_round_key(state, round_keys[4 * round_num:4 * round_num + 4][:])

    ciphertext = [hex(state[j][i])[2:].zfill(2).zfill(2)for i in range(4) for j
        in range(Nb)]

    return ciphertext
```

4.2 AES 解密过程

虽然 AES 不具有对合性，但解密过程只需要在加密过程的基础上对一些组件进行修改即可，下面只给出存在差异的组件的实现过程。

4.2.1 解密轮密钥的产生

解密密钥的生成过程相比加密密钥多了一步逆列混合变换过程。需要先按照加密过程生成所有轮密钥，再把列混合变换应用到除第一个轮密钥盒最后一个轮密钥之外的所有轮密钥上，实现代码如下所示：

```
def Inv_key_expansion(key, Nk, Nb, Nr):
    round_keys = key_expansion(key, Nk, Nb, Nr)
```

```
    for i in range(4,Nr*Nb):
          round_keys[i][:]=Inv_mix_single_column(round_keys[i][:])
    return round_keys
```

这里涉及到了逆列混合变换函数 Inv_mix_single_column，关于它的分析笔者会放在后面章节中进行单独分析。

4.2.2 逆 S 盒变换

类似 S 盒变换，此处也是输入一个 8bit 字节，输出一个变换后的 8bit 字节，采用的也是空间换时间的策略，根据输入字节的高低位直接查表，就能获得逆 S 盒变换后的结果，逆 S 盒于课本的 92 页处，实现代码如下所示：

```
inverse_s_box = [ .... ]

def Inv_sub_bytes(state,x):
   if x=='state':
      for i in range(4):
         for j in range(4):
            state[i][j] = inverse_s_box[state[i][j]]
   elif x=='key':
      for i in range(4):
         state[i] = inverse_s_box[state[i]]
```

4.2.3 逆行位移变换

与行位移变换类似，根据 Nb 判断状态字段中每一行需要位移的位数：

```
def Inv_shift_rows(state, Nb):
   if Nb == 4or Nb == 6:
      bias = [0, Nb-1, Nb-2, Nb-3]
   elif Nb == 6:
      bias = [0, Nb-1, Nb-3, Nb-4]
   for i in range(1, 4):
      state[i] = state[i][bias[i]:] + state[i][:bias[i]]
```

4.2.4 逆列混合变换

此处只是将原本相乘的多项式更换为多项式从 $c(x) =' 0B'x^3 +' 0D'x^2 +' 09'x +' 0E'$ 相乘，因此只需调整原本函数中的各项系数即可：

```
def Inv_mix_columns(state):
   for i in range(4):
```

```
        col = [state[j][i] for j in range(4)]
        col = Inv_mix_single_column(col)
        for j in range(4):
            state[j][i] = col[j]

def Inv_mix_single_column(col):
    a0, a1, a2, a3 = col[0], col[1], col[2], col[3]
    b0, b1, b2, b3 = 14, 9, 13, 11
    c0 = get_c(a0,b0) ^ get_c(a3,b1) ^ get_c(a2,b2) ^ get_c(a1,b3)
    c1 = get_c(a1,b0) ^ get_c(a0,b1)^ get_c(a3,b2)^ get_c(a2,b3)
    c2 = get_c(a2,b0) ^ get_c(a1,b1) ^ get_c(a0,b2)^ get_c(a3,b3)
    c3 = get_c(a3,b0) ^ get_c(a2,b1) ^ get_c(a1,b2) ^ get_c(a0,b3)
    col = [c0, c1, c2, c3]
    return col
```

4.2.5 AES 解密整体流程

AES 解密整体流程与加密相似，将其中的组件换为逆变换组件后，还需要调整密钥的使用顺序，由原本的顺序更改为倒序，整体流程代码如下所示：

```
def aes_decrypt(ciphertext, key):
    Nb = int(len(ciphertext) / 4)
    Nk = int(len(key) / 4)
    Nr = get_Nr(Nb, Nk)

    state = init_state(ciphertext, Nb)

    round_keys = Inv_key_expansion(key, Nk, Nb, Nr)

    add_round_key(state, round_keys[Nr*4:Nr*4+4][:])

    for round_num in range(Nr-1, 0,-1):
        Inv_sub_bytes(state, 'state')
        Inv_shift_rows(state, Nb)
        Inv_mix_columns(state)
        add_round_key(state, round_keys[4 * round_num:4 * round_num + 4][:])

    # 最后一轮
    round_num -=1
    Inv_sub_bytes(state, 'state')
    Inv_shift_rows(state, Nb)
    add_round_key(state, round_keys[4 * round_num:4 * round_num + 4][:])
    plaintext = [hex(state[j][i])[2:].zfill(2) for i in range(4) for j in
```

```
        range(Nb)]
    return plaintext
```

五、 实验结果与总结

在测试中笔者选择了课本 97 页的示例进行验证，因此初始的明文和密钥如下所示：

```
key = [
      0x00, 0x01, 0x20, 0x01,
      0x71, 0x01, 0x98, 0xae,
      0xda, 0x79, 0x17, 0x14,
      0x60, 0x15, 0x35, 0x94
    ]

    # 明文（16字节）
    plaintext = [
       0x00, 0x01, 0x00, 0x01,
       0x01, 0xa1, 0x98, 0xaf,
       0xda, 0x78, 0x17, 0x34,
       0x86, 0x15, 0x35, 0x66
    ]
```

调用函数 aes_encrypt 和 aes_decrypt 可分别得到密文结果和解密内容，运行代码，结果如下所示：

```
D:\Anaconda\envs\pytorch\python.exe D:\python\密码学\AES\main.py
加密后的结果:
['6c', 'dd', '59', '6b', '8f', '56', '42', 'cb', 'd2', '3b', '47', '98', '1a', '65', '42', '2a']
解密后的结果:
['00', '01', '00', '01', '01', 'a1', '98', 'af', 'da', '78', '17', '34', '86', '15', '35', '66']

进程已结束，退出代码为 0
```

将代码运行结果和课本示例对比，发现成功完成加密与解密。

5.1 算法效率

借用 python 中的 time 包，对算法运行时间进行统计，选择的密钥长度和明文长度仍均为 128bit；重复运行加解密过程 1000 次，计算运行时间的平均值如下图所示：

```
D:\Anaconda\envs\pytorch\python.exe D:\python\密码学\AES\main.py
 encryption speed: 304.3796827453304 mb/s
 decryption speed: 80.22655980519052 mb/s

进程已结束，退出代码为 0
```

可以看到算法加密的速度约为 300mb/s，而解密速度约为 80mb/s，相差较大。考虑加解密过程，猜测可能是解密过程的列混淆运算造成的，在解密过程的列混淆需要多次调用 xtime 函数来完成，实际上这里还有很大的优化空间，将其放在大作业里继续优化。

复现代码已经上传至 Github 之中：AES 加密算法 python 复现代码

5.2 分析与讨论

对比前一周实现的 DES 算法，下面列出 AES 算法的优缺点：

优点：

* 分组长度和密钥长度的灵活性：AES 算法可以设定为 32 比特的任意倍数，最小值为 128 比特，最大值为 256 比特。这种设计使得 AES 在应用中能够适应不同的需求。
* 运算速度快：AES 算法的运算速度相对较快，这使得它在一些需要高性能的应用场景中具有优势。
* 对内存的需求低：AES 算法在内存需求方面相对较低，这使得它在一些资源受限的环境中也能够得到应用。
* 高安全性：AES 在加密算法的首尾都使用了初始轮密钥加函数 AddRoundKey，克服了 DES 中初始置换和逆置换都没有密钥参与的缺点。

缺点：

* 分组较短：相比 DES 算法，AES 算法的分组较短。这可能会在某些应用场景中导致性能问题。
* 密钥长度可能太短：虽然 AES 算法可以设定为 32 比特的任意倍数，但它的密钥长度在某些情况下可能仍然较短。这可能会降低算法的安全性。
* 密码生命周期短：由于 AES 算法的密钥长度较短，因此它的密码生命周期也相对较短。这意味着需要更频繁地更换密钥，以保持算法的安全性。
* AES 运算不具有对合性，相比 DES 实现稍显复杂。

5.3 个人收获

通过此次实验，我加深了对 AES 算法的理解，并对其中各个组件有了更清晰的认识，能够使用 python 编程实现各个组件；但实际编写代码的过程中，并不是那么顺利，课本相关章节在这一部分编写的十分模糊，我在实现 S 盒变化时遇到了很大的问题，由于课本 86 页在讲述 S 盒变化时分了两步，分别是求字节的乘法逆和进行仿射变换，因此在开始做的时候，我理解成了先查询 S 盒，再进行仿射变换，后来发现是理解出错了。另外是课本在 88 页的轮密钥生成过程中的描述存在问题，原本应该是 Rcon[I/Nk-1], 但课本上描述的是 Rcon[I/Nk]，这样做了之后发现生成的密钥与课本示例不符，通过查阅相关资料，最终才发现此处课本存在编写错误。

参考文献

[1] 密码学引论 (第三版)

[2] AES 加密算法原理的详细介绍与实现（csdn）：http://t.csdnimg.cn/aScMx

[3] 高级加密标准（AES）（知乎）https://zhuanlan.zhihu.com/p/360393988

教师评语评分

评语：

评　　分：

评 阅 人：

评阅时间：