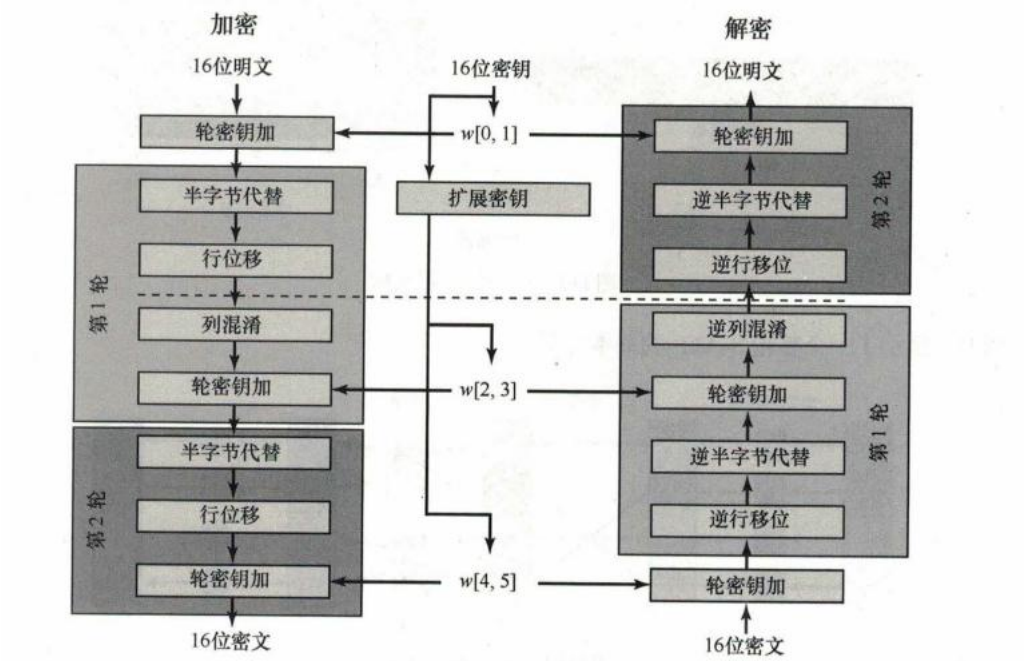
开发手册
本文用于讲解对简化 aes 的开发讲解
#运行环境
python3.11.5
numpy 1.26.0
tk 8.6.12

1. 项目介绍

这是一个简单的 AES 加密的实现，包括加解密并实现简单的图形化。明密文、密钥均为 16bit 的二进制或两字节的字符，并在此基础上进行拓展：包括两重加密、三重加密、中间相遇攻击、CBC 加密长消息等等

2. 算法介绍



2.1 密钥处理

密钥为 16bit，将被分为两个 8bit，为 w0、w1

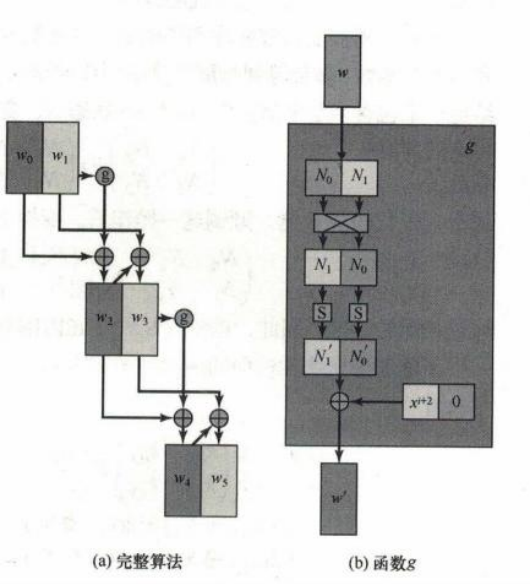
$$w_2 = w_0 \oplus g(w_1) = w_0 \oplus \mathrm{RCON}(1) \oplus \mathrm{SubNib}(\mathrm{RotNib}(w_1))$$
$$w_3 = w_2 \oplus w_1$$
$$w_4 = w_2 \oplus g(w_3) = w_2 \oplus \mathrm{RCON}(2) \oplus \mathrm{SubNib}(\mathrm{RotNib}(w_3))$$
$$w_5 = w_4 \oplus w_3$$

按照上图所示依次解出 w2、w3、w4、w5

1. RCON[i]是一个轮常数，RCON[1]=x^3=100，RCON[2]=x^4mod(x^4+x+1)=x+1=0011，右半字节补 0。所以，RCON[1]=10000000，RCON[2]=00110000

2. +号为异或运算，SubNib 则为 s 盒置换，RotNib 为交换左右 4 个 bit，g(w)运算如下图所示

3. 最后使用的密钥 key1=key,key2=w2+w3,key3=w4+w5



代码中构建密钥使用的 s 盒置换函数如下：
基本思想是通过 ab 行、cd 列来获取 s 盒中的数值，然后转化为二进制进行输出

```
#s 盒置换
def s_box(w):


    left_half = w[4:]
    right_half = w[:4]
    SBOX = np.array([[9, 4, 10, 11],
                     [13, 1, 8, 5],
                     [6, 2, 0, 3],
                     [12, 14, 15, 7]])


    left_half1 = [left_half[0], left_half[1]]
    left_half2 = [left_half[2], left_half[3]]
    right_half1 = [right_half[0], right_half[1]]
    right_half2 = [right_half[2], right_half[3]]
    left_str1 = ''.join(map(str, left_half1))
    left_str2 = ''.join(map(str, left_half2))
    right_str1 = ''.join(map(str, right_half1))
    right_str2 = ''.join(map(str, right_half2))
    # 然后将每组二进制数转化为十进制数，也就是 S 盒中的行列索引
    left_decimal1 = int(left_str1,2)
    left_decimal2 = int(left_str2, 2)
    right_decimal1 = int(right_str1,2)
    right_decimal2 = int(right_str2, 2)
    # 找到对应的 s_left 值
    s_left = SBOX[left_decimal1][left_decimal2]
    s_right = SBOX[right_decimal1][right_decimal2]
```

```
    # 将 s_left 转换为两个 bit，存在 s_left_1 中
    s_left_1 = [int(b) for b in "{0:04b}".format(s_left)]
    s_right_1=[int(b) for b in "{0:04b}".format(s_right)]
    # 将 s_left_1 和 s_right_1 拼接成一个长度为 8 的二进制数
    F1 = s_left_1 + s_right_1
    return np.array(F1)
```

最后便可以得到处理密钥的函数

```
#密钥处理
def key_process(key):
    RC1=[1,0,0,0,0,0,0,0]
    RC2=[0,0,1,1,0,0,0,0]
    #密钥侧处理
    w0=key[0:8]
    w1=key[8:16]
    g1=xor(s_box(w1),RC1)
    w2=xor(w0,g1)
    w3=xor(w1,w2)
    g2=xor(s_box(w3),RC2)
    w4=xor(w2,g2)
    w5=xor(w3,w4)
    key1=key
    key2=np.concatenate((w2,w3))
    key3=np.concatenate((w4,w5))

    return key1,key2,key3
```

2.2 明文加密

简单的 aes 有两轮加密

第一轮

1. 首先把传入的明文轮密钥加，即将明文和密钥进行异或运算(此时是把明文与 key1 进行异或运算)

2. 将第一步结果进行半字节替代，即从第一个 bit 开始，每 4 个 bit(设为 abcd)作为一个单位，并且把每个单位放入 s 盒中进行置换，置换的元素的 s 盒的第 ab 行、cd 列的元素

3. 将上一步结果进行行移位，即可认为把第二个单位和第四的单位交换位置

4. 下一步是进行列混淆，可以认为是矩阵运算，如下图所示，选择一个对称矩阵，和它进行矩阵乘法，而加法则可以认为是异或，而乘法为 GF(2^4)上的乘法，具体细节可以看后面的数据

5. 最后一步则是进行第二次轮密钥加，即将上一步的结果和 key2 进行异或运算

第二轮

1. 接下来进行再次进行半字节替代

2. 随后进行行移位，即将第 2 个单位和第 4 个单位进行交换

3. 最后一步是第三次轮密钥加，把上一步的结果和 key3 进行异或运算，最后就可以得到密文

GF(2^4)上的乘法如下：

(b) 乘法

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A

乘法代码实现为：

显然乘法矩阵是对称矩阵，如果是 a 乘以 b，那么我们就可以按照 a 行 b 列来索引到乘法结果，并转化为二进制输出

```
#列混淆
def multiply(n,round_part):
    a=10;b=11;c=12;d=13;e=14;f=15
    xBOX = np.array([ [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                      [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f],
                      [0, 2, 4, 6, 8, a, c, e, 3, 1, 7, 5, b, 9, f, d],
                      [0, 3, 6, 5, c, f, a, 9, b, 8, d, e, 7, 4, 1, 2],
                      [0, 4, 8, c, 3, 7, b, f, 6, 2, e, a, 5, 1, d, 9],
                      [0, 5, a, f, 7, 2, d, 8, e, b, 4, 1, 9, c, 3, 6],
                      [0, 6, c, a, b, d, 7, 1, 5, 3, 9, f, e, 8, 2, 4],
                      [0, 7, e, 9, f, 8, 1, 6, d, a, 3, 4, 2, 5, c, b],
                      [0, 8, 3, b, 6, e, 5, d, c, 4, f, 7, a, 2, 9, 1],
                      [0, 9, 1, 8, 2, b, 3, a, 4, d, 5, c, 6, f, 7, e],
                      [0, a, 7, d, e, 4, 9, 3, f, 5, 8, 2, 1, b, 6, c],
                      [0, b, 5, e, a, 1, f, 4, 7, c, 2, 9, d, 6, 8, 3],
                      [0, c, b, 7, 5, 9, e, 2, a, 6, 1, d, f, 3, 4, 8],
                      [0, d, 9, 4, 1, c, 8, 5, 2, f, b, 6, 3, e, a, 7],
                      [0, e, f, 1, d, 3, 2, c, 9, 7, 6, 8, 4, a, b, 5],
                      [0, f, d, 2, 9, 6, 4, b, 1, e, c, 3, 8, 7, 5, a]])
```

```
    round_part_2 = ''.join(map(str, round_part))
    round_part_10=int(round_part_2,2)
    # 找到对应的运算值
    Sii = xBOX[n][round_part_10]
    S_mult= [int(b) for b in "{0:04b}".format(Sii)]

    return S_mult
```

加密中使用的 s 盒函数如下，与密钥中的 s 盒相似，但

```
#s 盒置换
def s_box1(round_part):

    SBOX1 = np.array([[9, 4, 10, 11],
                      [13, 1, 8, 5],
                      [6, 2, 0, 3],
                      [12, 14, 15, 7]])

    round_part1 = [round_part[0], round_part[1]]
    round_part2 = [round_part[2], round_part[3]]
    left_str1 = ''.join(map(str, round_part1))
    left_str2 = ''.join(map(str, round_part2))
    # 然后将每组二进制数转化为十进制数，也就是 S 盒中的行列索引
    round_part_decimal1 = int(left_str1,2)
    round_part_decimal2 = int(left_str2, 2)
    round_part_s = SBOX1[round_part_decimal1][round_part_decimal2]
    round_part_s11= [int(b) for b in "{0:04b}".format(round_part_s)]
    F1 = round_part_s11
    return np.array(F1)
```

完整的加密代码如下：

```
#加密函数
def encrypt(plaintext,key):

    key1,key2,key3=key_process(key)
    round_key_addition=xor(plaintext,key1)
    #第一轮
    round_part00=round_key_addition[:4]
    round_part10=round_key_addition[4:8]
    round_part01=round_key_addition[8:12]
    round_part11=round_key_addition[12:]
    #半字节替换
    round_part_s00=s_box1(round_part00)
    round_part_s10=s_box1(round_part10)
    round_part_s01=s_box1(round_part01)
```

```
    round_part_s11=s_box1(round_part11)
    #行位移
    temp=round_part_s10
    round_part_s10=round_part_s11
    round_part_s11=temp
    #列混淆
    S00=xor(multiply(1,round_part_s00),multiply(4,round_part_s10))
    S10=xor(multiply(4,round_part_s00),multiply(1,round_part_s10))
    S01=xor(multiply(1,round_part_s01),multiply(4,round_part_s11))
    S11=xor(multiply(4,round_part_s01),multiply(1,round_part_s11))
    Befoer_First_Round_Results = np.concatenate((S00, S10,S01,S11))
    First_Round_Results=xor(Befoer_First_Round_Results,key2)
    #第二轮
    round_part2_00=First_Round_Results[:4]
    round_part2_10=First_Round_Results[4:8]
    round_part2_01=First_Round_Results[8:12]
    round_part2_11=First_Round_Results[12:]
    #半字节替换
    round_part_s2_00=s_box1(round_part2_00)
    round_part_s2_10=s_box1(round_part2_10)
    round_part_s2_01=s_box1(round_part2_01)
    round_part_s2_11=s_box1(round_part2_11)
    #行位移
    temp2=round_part_s2_10
    round_part_s2_10=round_part_s2_11
    round_part_s2_11=temp2
    #轮密钥加
    Befoer_Second_Round_Results = np.concatenate((round_part_s2_00,
round_part_s2_10,round_part_s2_01,round_part_s2_11))
    Second_Round_Results=xor(Befoer_Second_Round_Results,key3)
    ciphertext=Second_Round_Results
    return np.array(ciphertext)
```

2.3 密文解密

相较于加密来说，解密则是其逆向的过程，代入子密钥的顺序则是相反的，同样分为两轮

第一轮：

1. 首先将密文与 key1 进行异或

2. 对上一步的结果进行逆行移位，和行移位相同，同样是把第 2 个单位和第 4 个单位进行位置互换

3. 下一步是逆半字节替代，方法和半字节替代相同，但是需要使用逆 s 盒

4. 然后再次进行轮密钥加，即将上一步的结果和 key2 异或

5. 然后进行逆列混淆，需要选择一个和列混淆矩阵相乘后为单位阵的矩阵

第二轮：

1. 将上一步结果进行逆行移位

2. 将上一步结果进行逆字节替代，和第 1 轮相同

3. 将上一步结果与 key1 进行异或，则得到明文

逆 s 盒函数如下，与 s 盒函数原理相同，但使用的盒不同

```
#逆 s 盒
def s_box2(round_part):


    SBOX2 = np.array([[10, 5, 9, 11],
                      [1, 7, 8, 15],
                      [6, 0, 2, 3],
                      [12, 4, 13, 14]])


    round_part1 = [round_part[0], round_part[1]]
    round_part2 = [round_part[2], round_part[3]]
    left_str1 = ''.join(map(str, round_part1))
    left_str2 = ''.join(map(str, round_part2))
    # 然后将每组二进制数转化为十进制数，也就是 S 盒中的行列索引
    round_part_decimal1 = int(left_str1,2)
    round_part_decimal2 = int(left_str2, 2)
    round_part_s = SBOX2[round_part_decimal1][round_part_decimal2]
    round_part_s11= [int(b) for b in "{0:04b}".format(round_part_s)]
    F1 = round_part_s11
    return np.array(F1)
```

完整的解密函数如下：

```
def decrypt(ciphertext, key):


    key1,key2,key3=key_process(key)
    round_key_addition=xor(ciphertext,key3)
    #第一轮
    round_part00=round_key_addition[:4]
    round_part10=round_key_addition[4:8]
    round_part01=round_key_addition[8:12]
    round_part11=round_key_addition[12:]
    #逆行位移
    temp=round_part10
```

```
    round_part10=round_part11
    round_part11=temp
    #逆半字节替换
    round_part_s00=s_box2(round_part00)
    round_part_s10=s_box2(round_part10)
    round_part_s01=s_box2(round_part01)
    round_part_s11=s_box2(round_part11)
    #轮密钥加
    round_hole_part=np.concatenate((round_part_s00, round_part_s10,round_part_s01,round_part_s11))
    round_hole_part_xor=xor(round_hole_part,key2)
    round_part_s00=round_hole_part_xor[:4]
    round_part_s10=round_hole_part_xor[4:8]
    round_part_s01=round_hole_part_xor[8:12]
    round_part_s11=round_hole_part_xor[12:]
    #逆列混淆
    S00=xor(multiply(9,round_part_s00),multiply(2,round_part_s10))
    S01=xor(multiply(2,round_part_s00),multiply(9,round_part_s10))
    S10=xor(multiply(9,round_part_s01),multiply(2,round_part_s11))
    S11=xor(multiply(2,round_part_s01),multiply(9,round_part_s11))
    #第二轮
    round_part2_00=S00
    round_part2_10=S01
    round_part2_01=S10
    round_part2_11=S11
    #逆行位移
    temp2=round_part2_10
    round_part2_10=round_part2_11
    round_part2_11=temp2
    #逆半字节替换
    round_part_s2_00=s_box2(round_part2_00)
    round_part_s2_10=s_box2(round_part2_10)
    round_part_s2_01=s_box2(round_part2_01)
    round_part_s2_11=s_box2(round_part2_11)
    #轮密钥加
    Befoer_Second_Round_Results = np.concatenate((round_part_s2_00,
round_part_s2_10,round_part_s2_01,round_part_s2_11))
    Second_Round_Results=xor(Befoer_Second_Round_Results,key1)
    plaintext=Second_Round_Results
    return np.array(plaintext)
```

测试用例如下：

```
if __name__ == "__main__":


    ciphertext = [1,1,0,0,1,1,1,0,0,1,0,1,0,1,1,1]  # 示例密文
```

```
plaintext  = [1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1]  # 示例明文
key = [0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0]  # 示例密钥
ciphertext1=encrypt(plaintext,key)
plaintext1=decrypt(ciphertext,key)
```

2.4 字符处理

主要考虑传进加密运算时先转化为 ASCii 值，然后转化为二进制的；而输出时则相反，把二进制 bit 转化为 10 进制的值，随后再转化为对应 ASCii 字符

字符处理函数如下

```
#字符处理
def character_process(kerOrtext):
    kerOrtext_ascii1 = ord(kerOrtext[0])
    kerOrtext_ascii2 = ord(kerOrtext[1])
    kerOrtext_binary1 = format(kerOrtext_ascii1, '08b')# 将密钥转化为 16bit 的二进制字符串
    kerOrtext_binary2 = format(kerOrtext_ascii2, '08b')
    kerOrtext1 = [int(bit) for bit in kerOrtext_binary1]
    kerOrtext2= [int(bit) for bit in kerOrtext_binary2]
    kerOrtext_arr=np.concatenate((kerOrtext1,kerOrtext2))
    kerOrtext=[int(bit) for bit in kerOrtext_arr]


    return kerOrtext
```

2.5 用户界面

搭建的 ui 界面如下，在设计 ui 界面时考虑根据用户的输入形式来判断是否需要字符形式的输出，示例代码如下


```
    # 判断传入明文是字符（2 字节）且密钥是字符
    elif len(plaintext) == 2 and len(key) == 2:
        plaintext_ascii1 = ord(plaintext[0])
        plaintext_ascii2 = ord(plaintext[1])
        key_ascii1 = ord(key[0])
        key_ascii2 = ord(key[1])
```

```
        # 判断如果明文的 ascii 码不大于 255 且密钥只由 0 和 1 构成，则将明文的 ascii 码转化为 8bit 的二进制字符串，再调用加
密函数
        if plaintext_ascii1 > 255 or plaintext_ascii2 > 255:
            messagebox.showerror("错误", "请输入正确的 ascii 码")
        else:
            plaintext=encry.character_process(plaintext)
            key=encry.character_process(key)
            ciphertext = encry.encrypt(plaintext, key)
            ciphertext1=ciphertext[:8]
            ciphertext2=ciphertext[8:]
            ciphertext_chars = ""
            for i in range(0, len(ciphertext1), 8):
                binary1 = ciphertext1[i:i+8]
                binary2 = ciphertext2[i:i+8]
                decimal1 = int("".join(str(bit) for bit in binary1), 2)
                decimal2 = int("".join(str(bit) for bit in binary2), 2)
                if decimal1 > 255 or decimal2>255:
                    # 如果超出了 ascii 码能表示的范围，直接显示二进制形式的密文
                    ciphertext_entry.delete(0, tk.END)
                    ciphertext_entry.insert(0, "".join(
                        str(bit) for bit in ciphertext))
                    return
                character1 = chr(decimal1)
                character2 = chr(decimal2)
                ciphertext_chars += character1+character2
            ciphertext_entry.delete(0, tk.END)
            ciphertext_entry.insert(0, ciphertext_chars)
```

3. 多重加密

3.1 双重加密

将 S-AES 算法通过双重加密进行扩展，分组长度仍然是 16 bits，但密钥长度为 32 bits

双重加密时使用的密钥为 32bit，那么则可以通过两次简单的加密，得到密文，再类似的，通过两次解密得到明文，双重加密示例代码如下

```
def double_encrypt(plaintext,key):
    K2=key[16:]
    first_ciphertext=encrypt(plaintext,key)
    ciphertext=encrypt(first_ciphertext,K2)

    return ciphertext
```

```
def double_decrypt(ciphertext,key):
    K1=key[:16]
```

```
    K2=key[16:]
    first_plaintext=decrypt(ciphertext,K2)
    plaintext=decrypt(first_plaintext,K1)

    return plaintext
```

3.2 三重加密

采用 48bit 密钥进行三重加密，则相当于使用 K1、K2、K3 各加密一次，经过三重加密获得明文，而密文则过程相反，简单的示例代码如下

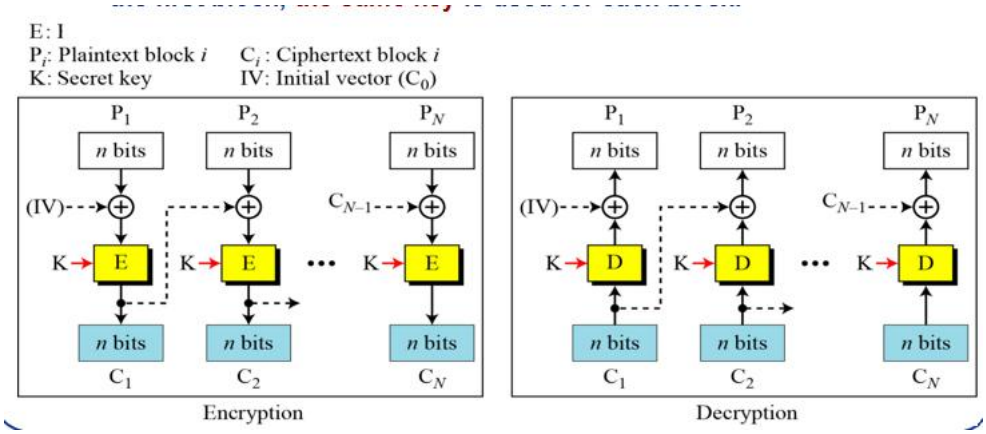
```
def triple_encrypt(plaintext,key):
    K1=key[:16]
    K2=key[16:32]
    K3=key[32:]
    first_ciphertext=encrypt(plaintext,K1)
    second_ciphertext=encrypt(first_ciphertext,K2)
    ciphertext=encrypt(second_ciphertext,K3)

    return ciphertext
def triple_decrypt(ciphertext,key):
    K1=key[:16]
    K2=key[16:32]
    K3=key[32:]
    first_plaintext=decrypt(ciphertext,K3)
    second_plaintext=decrypt(first_plaintext,K2)
    plaintext=decrypt(second_plaintext,K1)

    return plaintext
```

4. 工作模式

4.1CBC

CBC 中包括待加密的明文、密钥、以及双方共有的一个初始向量，加密流程如下

1. 将明文划分为每个 nbit 明文块(P1-Pn)进行加密
2. 将初始向量与 P1 进行异或，然后将得到的结果和 key 进行加密得到密文块 C1
3. 将 C2 与 P2 进行异或，然后将得到的结果和 key 进行加密得到密文块 C2
4. 依次类推得到 C1-Cn
5. 解密则同理，把密文 C1-Cn，从头开始解密，将 C1 与 key 进行解密，然后与初始向量异或得到明文块 P1
6. 而把 C2 与 key 进行解密后，与 C1 进行异或则得到明文块 P2，依次类推即可



示例代码如下：

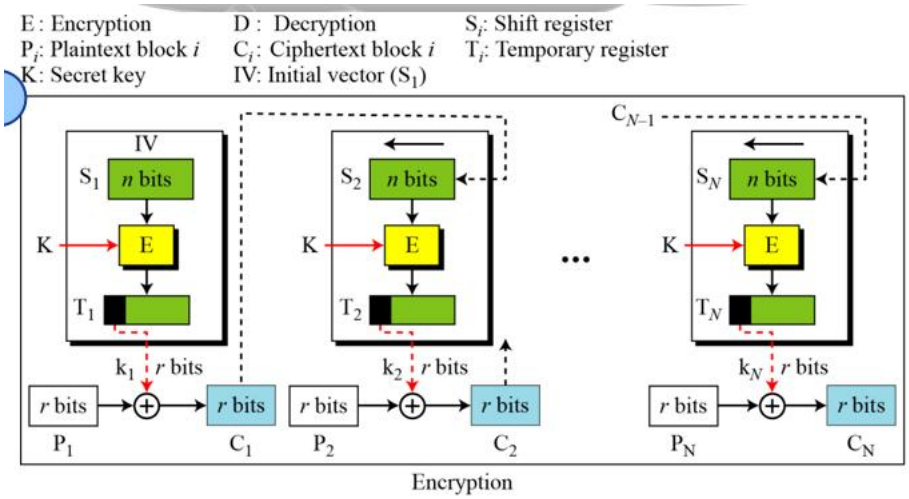
```
# CBC 加密
def cbc_encrypt(plaintext, i_v, key):
    plaintext1, plaintext2, plaintext3, plaintext4 = text_cut(plaintext)  # 切割
    K1 = xor(plaintext1, i_v)
    C1 = encry.encrypt(K1, key)  #
    # 以此类推
    K2 = xor(plaintext2, C1)
    C2 = encry.encrypt(K2, key)
    K3 = xor(plaintext3, C2)
    C3 = encry.encrypt(K3, key)
    K4 = xor(plaintext4, C3)
    C4=encry.encrypt(K4, key)

    ciphertext = np.concatenate((C1, C2, C3, C4))

    return ciphertext
```

```
# CBC 解密
def cbc_decrypt(ciphertext, i_v, key):
    ciphertext1, ciphertext2, ciphertext3, ciphertext4 = text_cut(ciphertext)  # 解密按照相同顺序一段一段解密就
好
    k1 = encry.decrypt(ciphertext1, key)
    plaintext1 = xor(i_v, k1)
    k2 = encry.decrypt(ciphertext2, key)
    plaintext2 = xor(ciphertext1, k2)
    k3 = encry.decrypt(ciphertext3, key)
    plaintext3 = xor(ciphertext2, k3)
    k4 = encry.decrypt(ciphertext4, key)
    plaintext4 = xor(ciphertext3, k4)
```

```
    plaintext = np.concatenate((plaintext1, plaintext2, plaintext3, plaintext4))
    return plaintext
```

4.2CBF

CFB 与 CBC 相似，主要区别是 CFB 是让初始向量与 key 进行加密后与明文块异或得到密文块，而将本次得到的密文块作为下一步的初始向量，依次类推得到所有的密文块



示例代码如下：

```
#CFB 加密
def cbc_encrypt(plaintext,i_v,key):
    plaintext1,plaintext2,plaintext3,plaintext4=text_cut(plaintext)#切割
    k1=encry.encrypt(i_v,key)#初始向量和密钥加密得到 k1
    ciphertext1=xor(plaintext1,k1)#异或得到第一段加密文，并作为下一段的向量
    #以此类推
    k2=encry.encrypt(ciphertext1,key)
    ciphertext2=xor(plaintext2,k2)

    k3=encry.encrypt(ciphertext2,key)
    ciphertext3=xor(plaintext3,k3)
    k4=encry.encrypt(ciphertext3,key)
    ciphertext4=xor(plaintext4,k4)
    ciphertext=np.concatenate((ciphertext1,ciphertext2,ciphertext3,ciphertext4))
    return ciphertext
```

```
#CFB 解密
def cbc_decrypt(ciphertext,i_v,key):
    ciphertext1,ciphertext2,ciphertext3,ciphertext4=text_cut(ciphertext)#解密安装相同顺序一段一段解就好
    k1=encry.decrypt(i_v,key)
    plaintext1=xor(ciphertext1,k1)
```

```
    k2=encry.decrypt(plaintext1,key)
    plaintext2=xor(ciphertext2,k2)
```

```
    k3=encry.decrypt(plaintext2,key)
    plaintext3=xor(ciphertext3,k3)
    k4=encry.decrypt(plaintext3,key)
    plaintext4=xor(ciphertext4,k4)
    ciphertext=np.concatenate((plaintext1,plaintext2,plaintext3,plaintext4))
    return plaintext
```