武汉大学国家网络安全学院

课程实验报告

分组密码工作模式

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 **15** 日

分组密码工作模式-SM4

一、 实验目的

1. 掌握分组密码的基本概念
2. 掌握 DES、AES、SMS4 密码算法
3. 了解分组密码 DES、AES、SMS4 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、 实验内容及原理

本次实验需要在完成 SM4 算法实现的基础上，探究分组密码以下 6 种工作方式。

1. 电码本模式 ECB
2. 密文链接模式 CBC
3. 输出反馈模式 OFB
4. 密文反馈模式 CFB
5. X CBC 模式
6. 计数器模式 CTR

三、 实验环境

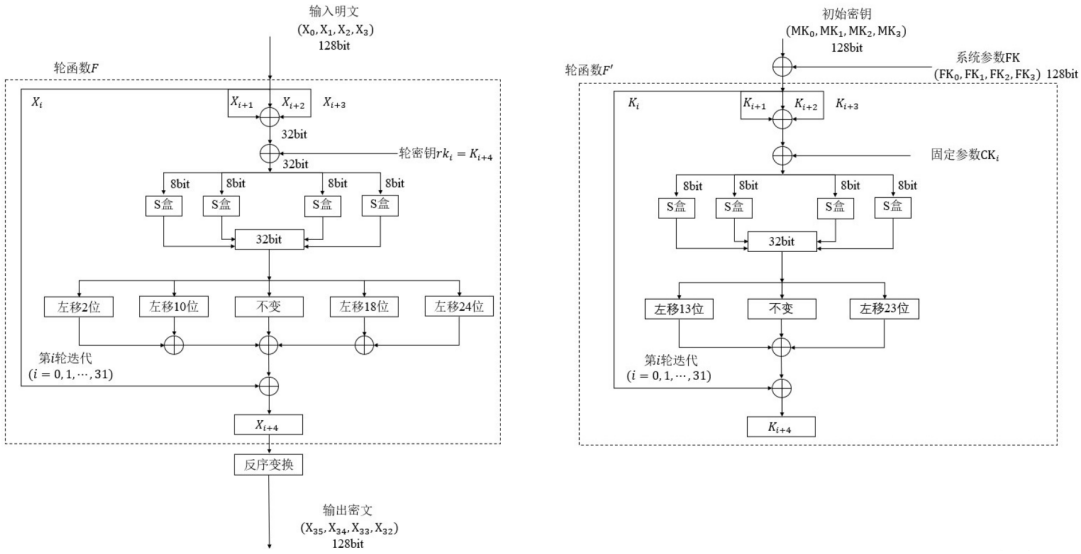
- 操作系统：Windows11 家庭版
- 基本硬件：CPU：AMD Ryzen 5 5600H, 16G 内存
- 所用软件：Pycharm 2023.2.1 专业版
- 所用语言：python3.10

四、 实验步骤与分析

在此部分，我首先给出 SM4 加解密算法的实现流程；之后分别分析实现该加密算法在 6 种不同工作模型下的工作方式。

4.1 SM4 算法的实现

SM4 算法主要包含异或、移位以及盒变换操作。它分为密钥拓展和加/解密两个模块，这两个模块的流程大同小异，其中，移位变换是指循环左移；盒变换是一个将 8bit 输入映射到 8bit 输出的变换，是一个固定的变换下图是 SM4 的加解密（左）和密钥拓展（右）的流程图：



由于 SM4 算法属于对合运算，因此解密过程与加密过程的完全相同，只需要将密文当作原文输入，颠倒密钥的使用顺序即可，最终得到的加密密文即解密所得的明文。

SM4 密码算法使用了以下基本密码部件：**S 盒、非线性变换部件、线性变换部件、合成变换部件**。在加密时又使用到了**轮函数**；以及还需要**密钥扩展算法**来生成加密所需的密钥流。

接下来首先介绍 4 种基本的部件，再分析加密所需的密钥扩展算法与轮函数的实现。

4.1.1 S 盒部件

S 盒（Substitution Box）是密码学中一种用于替代数据的重要组件，它广泛用于分组密码算法的设计中。在 SM4 算法中，S 盒是其中的一个关键部分。

S 盒的主要作用是通过对输入的每个字节进行替代，产生输出。S 盒通常被设计为一个固定的、非线性的、对抗各种密码分析方法的映射表。在 SM4 算法中，S 盒的设计是经过深思熟虑的，以满足密码学的安全性要求。

S 盒的变换可以通过查表来完成，输入为一个字节，输出也为一个字节。即：

$$b = S_{\text{Box}}(a) \tag{1}$$

具体的实现代码如下所示：

```
sbox = [
        0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0
            xc2, 0x28, 0xfb, 0x2c, 0x05,
        0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0
            x26, 0x49, 0x86, 0x06, 0x99,
```

```
        0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0
            x43, 0xed, 0xcf, 0xac, 0x62,
        0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0
            xfa, 0x75, 0x8f, 0x3f, 0xa6,
        0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0
            x19, 0xe6, 0x85, 0x4f, 0xa8,
        0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0
            x4b, 0x70, 0x56, 0x9d, 0x35,
        0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0
            x3b, 0x01, 0x21, 0x78, 0x87,
        0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0
            xe7, 0xa0, 0xc4, 0xc8, 0x9e,
        0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0
            xce, 0xf9, 0x61, 0x15, 0xa1,
        0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0
            x30, 0xf5, 0x8c, 0xb1, 0xe3,
        0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0
            xab, 0x0d, 0x53, 0x4e, 0x6f,
        0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0
            x72, 0x6d, 0x6c, 0x5b, 0x51,
        0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0
            x41, 0x1f, 0x10, 0x5a, 0xd8,
        0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0
            x12, 0xb8, 0xe5, 0xb4, 0xb0,
        0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0
            x09, 0xc5, 0x6e, 0xc6, 0x84,
        0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0
            x3e, 0xd7, 0xcb, 0x39, 0x48
    ]

def S_box(byte_Str):
    byte=transform(byte_Str)
    row = (byte »4) & 0x0F
    col = byte & 0x0F
    return sbox[row * 16+ col]
```

4.1.2 非线性变换部件

SM4 的非线性变换是一种以字为单位的非线性替代变换，它由 4 个 S 盒并列构成，实际上是 S 盒的一种并行运用。输入为一个字，在处理时其实是把一个字拆分为了 4 个字节，分别送到 S 盒中进行变换，因此输出也为 4 个字节，即一个字。

$$B = \tau(A) = (S_{\text{Box}}(a_0), S_{\text{Box}}(a_1), S_{\text{Box}}(a_2), S_{\text{Box}}(a_3)) \tag{2}$$

实现代码如下：

```
def tt(Word):
    a=[ hex((Word>>(8*i))&0xff)[2:] for i in range(4)]
    a.reverse()
    b=[ hex(S_box(x))[2:].zfill(2) for x in a]
    res=transform(b[0]+b[1]+b[2]+b[3])
    return res
```

4.1.3 线性变换部件

线性变换部件 L 的密码学作用为可以起到扩散。它是以字来作为处理单位的线性变换部件，输入输出的字都是 32 位。

设 L 的输入为字 B，输出为字 C，则

$$C = L(B) = B \oplus (B \ll 2) \oplus (B \ll 10) \oplus (B \ll 18) \oplus (B \ll 24) \tag{3}$$

实现代码如下：

```
def left_rotate(number, shift):
    return ((number <<shift) | (number >>(32 - shift))) & 0xFFFFFFFF

def L1(number):
    return number^(left_rotate(number,2))^(left_rotate(number,10))^
    (left_rotate(number,18))^(left_rotate(number,24))
```

4.1.4 合成变换部件

合成变换 T 的数据处理单位为字，是由非线性变换 和线性变换部件 L 两者复合而组成的。设输入的字为 X，则需要先对 X 进行一个非线性 变换，然后再进行线性 L 变换，可记为

$$T(X) = L(\tau(X)) \tag{4}$$

实现如下所示：

```
def T1(word):
    t1=tt(word)
    return L1(t1)
```

4.1.5 密钥扩展算法

SM4 密码算法采用 32 轮的迭代加密结构，拥有 128 位加密密钥，一共使用 32 轮密钥，每一轮的加密使用 32 位的一个轮密钥。SM4 算法的特点使得它需要使用一个密钥扩展算法，在加密密钥当中产生 32 个轮密钥。在这个密钥的扩展算法当中有常数 FK、固定参数 CK 这两个数值，利用这两个数值便可完成它的这一个扩展算法。

1. 常数 FK:FK0=(A3B1BAC6)，FK1=(56AA3350),FK2=(677D9197), FK3=(B27022DC)
2. 固定参数 CK：一共使用有 32 个固定参数 CKi，这个 CKi，是一个字，它的产生规则是：设 cki，j 为 CKi 的第 j 字节 (i=0，1，…，31;j=0,1,2,3)，即 $CK_i = (c_{ki,0}, c_{ki,1}, c_{ki,2}, c_{ki,3})$. 最终所用的固定参数 CK 会在代码中给出。，

假设输入的加密密钥为 MK= (MK0，MK1，MK2，MK3)，输出的轮密钥为 rki, i=0,1…,30,31，中间的数据为 Ki=0,1,…,34,35。则密钥扩展算法可描述如下：

$$(K_0, K_1, K_2, K_3) = (MK_0 \oplus FK_0, MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3) \tag{5}$$

$$r_{k_i} = K(i+4) = K_i \oplus T'(K(i+1) \oplus K(i+2) \oplus K(i+3) \oplus CK_i) \tag{6}$$

实际上后面 SM4 实际的加密算法与密钥扩展过程非常相似，在密钥扩展算法中 T’ 变换与加密算法轮函数中的 T 基本相同，只将其中的线性变换 L 修改为以下的 L’：

$$L'(B) = B \oplus (B \ll 13) \oplus (B \ll 23) \tag{7}$$

实现代码如下所示：

```
def generate_key(Key):
    FK = [0xA3B1BAC6, 0x56AA3350, 0x677D9197, 0xB27022DC]
    CK = [
        0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,
        0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
        0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,
        0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
        0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,
        0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
        0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,
        0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279
    ]

    MK=[ int(Key[i]+Key[i+1]+Key[i+2]+Key[i+3],16) for i in range(0,16,4)]
    K=[ MK[i]^FK[i] for i in range(4)]
    for i in range(32):
        t=hex(K[i] ^ T2(K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i]))
        K.append(K[i]^T2(K[i+1]^K[i+2]^K[i+3]^CK[i]))
    rk=K[4:36]
    return rk
```

4.1.6 轮函数

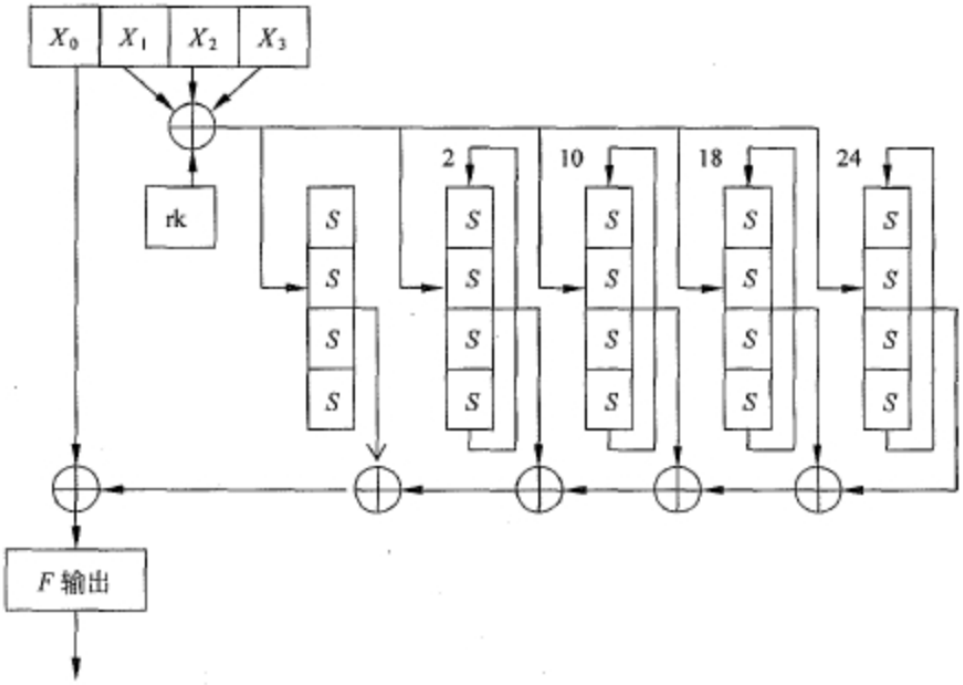
SM4 密码算法的结构采用了对基本轮函数进行迭代，它的一个轮函数可以由以上这些基本的密码部件来构成，这是一种用字来作为处理单位的密码函数。

设轮函数 F 的输入为 (X0，X1，X2, X3)，四个 32 位字，一共有 128 位。轮密钥为 rk ，rk 也是一个 32 位的字。轮函数 F 的输出也是一个 32 位的字。

轮函数的处理过程结合之前所实现的各类部件，最终的数学表达式如下所示，此处省略了推导过程：

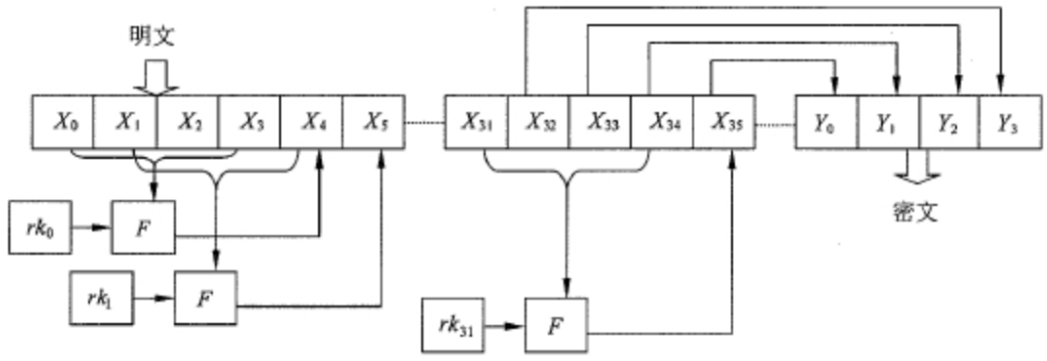
$$F(X_0, X_1, X_2, X_3, rk) = X_0 \oplus [S_{\text{Box}}(B)] \oplus [S_{\text{Box}}(B) \ll 2] \oplus [S_{\text{Box}}(B) \ll 10] \oplus [S_{\text{Box}}(B) \ll 18] \oplus [S_{\text{Box}}(B) \ll 24] \tag{8}$$

流程示意图如下所示：



4.1.7 加密算法

在实现了前面一系列部件后，终于到了明文的加密过程，加密的流程图如下所示：



实际上，SM4 密码算法的数据分组长度为 128 比特，密钥长度也是 128 比特，是分组算法当中的一种。它采用 32 轮迭代结构来作为它的加密算法，每轮使用一个轮密钥。设输入的明文为四个字 (X0，X1，X2, X3)，一共有 128 位。输入轮密钥为 rk_i,i=0,1,⋯，31，一共 32 个字。输出密文为四个字 (Y0，

Y1，Y2, Y3)，128 位。则这个加密算法可描述如下：

$$X_{i+4} = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i) \tag{9}$$

最终还需要进行一次反序处理 R：

$$R(Y_0, Y_1, Y_2, Y_3) = (X_{35}, X_{34}, X_{33}, X_{32}) \tag{10}$$

最终得到的 Y，即为加密所得的密文。若为解密过程，只需要将密文 Y 作为加密部件的输入，逆序使用密钥即可完成解密，在此也就不再赘述。上述过程的实现代码如下所示：

```
def deal(rs,Plain_text,way):

   M_word = [
      int(Plain_text[i] + Plain_text[i + 1] + Plain_text[i + 2] + Plain_text[i
          + 3], 16) for i in range(0, 16, 4)]
   X = M_word
   if way=='encrypt':
      for i in range(32):
         X.append(X[i] ^ T1(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rs[i]))
   elif way=='decrypt':
      for i in range(32):
         X.append(X[i] ^ T1(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rs[31 - i]))
   Y = X[-4:]
   Y.reverse()
   return Y
```

4.2 SM4 的电码本模式 ECB

ECB 工作模式即为最简单的工作模式，在 ECB 模式中，每个消息块都被独立加密，而密文块之间没有相互依赖关系。ECB 模式的加密过程可以简要描述为以下步骤：

- 划分消息：将长消息划分为块，每个块通常为固定长度（比如 64 或 128 比特）
- 独立加密：对每个消息块独立应用相同的加密算法和密钥。每个块的加密是相互独立的，不受其他块的影响
- 合并密文：将每个加密后的块合并为最终的密文

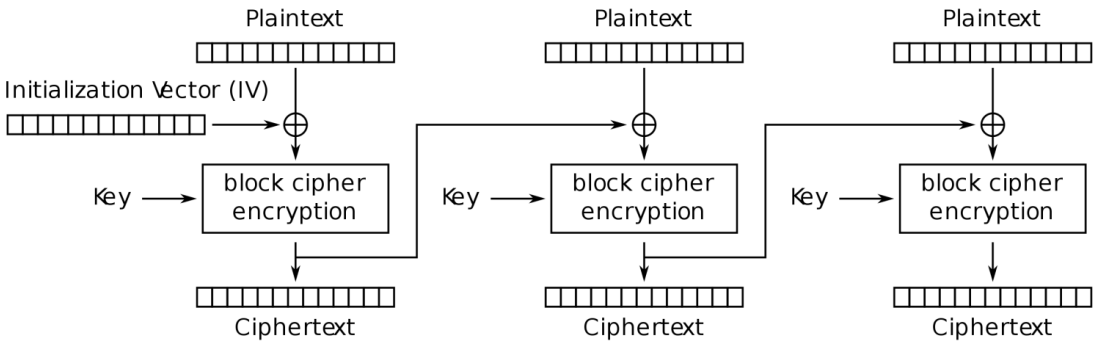
实现代码如下所示，实际上只需要将原始明文进行简单的分块处理即可，每个明文块之间没有任何关联。

```
def ECB(rs,M,way):
   M2=[M[i:i + 16] for i in range(0, len(M), 16)]
   res=[]
   for m in M2:
      t=deal(rs, m, way)
```

```
    for i in range(4):
        t2=hex(t[i])[2:].zfill(8)
        for j in range(0,len(t2),2):
            res.append(t2[j:j+2])

  return res
```

4.3 SM4 的密文链接模式 CBC

CBC 工作模式如下所示：



Cipher Block Chaining (CBC) mode encryption

CBC 采用密文参与链接的方式，提高了加密过程的复杂程度，相比 CBC 具有以下优点：

- 隐藏重复块：在 ECB 模式中，相同的明文块会得到相同的密文块，这可能导致一些安全性问题。而在 CBC 模式中，前一个密文块的影响会传递到后续块，因此即使相同的明文块在不同的位置出现，它们的密文块也会不同，提高了安全性
- 抵抗选择性位翻转攻击：在 ECB 模式中，攻击者可以选择性地翻转密文块，而不影响解密的正确性。在 CBC 模式中，由于密文块的传递性，翻转一个密文块会影响整个解密过程，增加了攻击的难度
- 随机化初始向量：CBC 模式使用一个初始向量（IV）来初始化第一个块的加密过程，这个 IV 的随机性增加了加密的随机性，提高了安全性

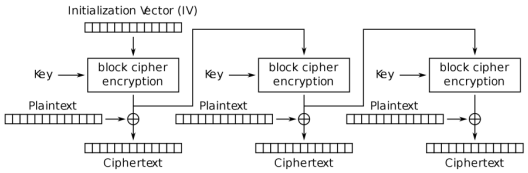
实现代码如下所示：

```
def CBC(rs,M,way,Z):
    M2=[M[i:i + 16] for i in range(0, len(M), 16)]
    res=[]
    q=0
    tlist=[]
    for m in M2:
        if way=='encrypt':
            if q==0:
```

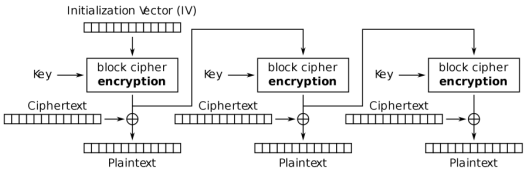
```
            m=XOR(m,Z)
        else:
            m=XOR(m,tlist[q-1])
        q+=1

    t = deal(rs, m, way)
    tlist.append(t)

    if way == 'decrypt':
        if q == 0:
            t = [x^y for x,y in zip(t,Z) ]
        else:
            t = XOR(M2[q-1],t)
            t2=[]
            for i in range(0, len(t), 4):
                sub_list = t[i:i + 4]
                combined_String = ''.join(sub_list)
                number = int(combined_String, 16)
                t2.append(number)
            t=t2
        q += 1

    for i in range(4):
        t2=hex(t[i])[2:].zfill(8)
        for j in range(0,len(t2),2):
            res.append(t2[j:j+2])

  return res
```

4.4 SM4 的输出反馈模式 OFB

OFB（Output Feedback）与其他模式（如 ECB、CBC、CFB）不同，它将分组密码算法应用于位而不是块。OFB 模式产生一个伪随机的密钥流，该密钥流然后被用于对明文进行加密。示意图如下：



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

以下是 OFB 模式的基本工作原理：

1. **初始阶段：**

- 一个初始向量（IV）被输入到分组密码算法的加密函数中。

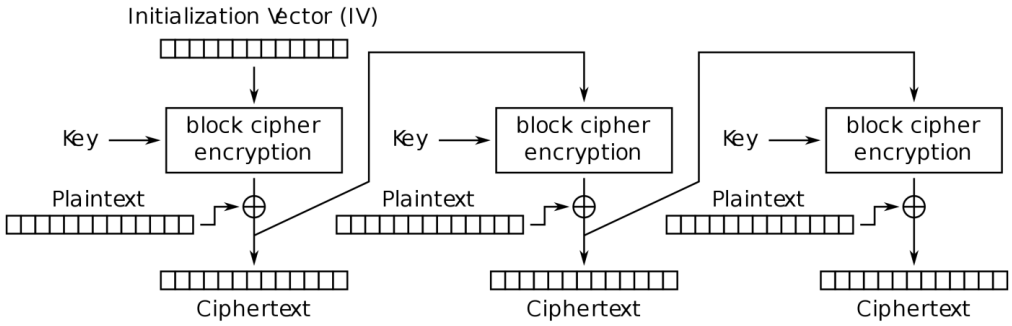
2. **加密阶段：**
   - 将初始向量输入到分组密码算法的加密函数中，得到一个伪随机序列（密钥流）。
   - 将伪随机序列与明文进行异或运算，生成密文。
3. **反馈阶段：**
   - 将伪随机序列输入到分组密码算法的加密函数中，得到新的伪随机序列。
   - 将新的伪随机序列与下一个明文进行异或运算，生成新的密文。
4. **重复：**
   - 重复上述过程，直到所有的明文都被加密。

实现代码如下：

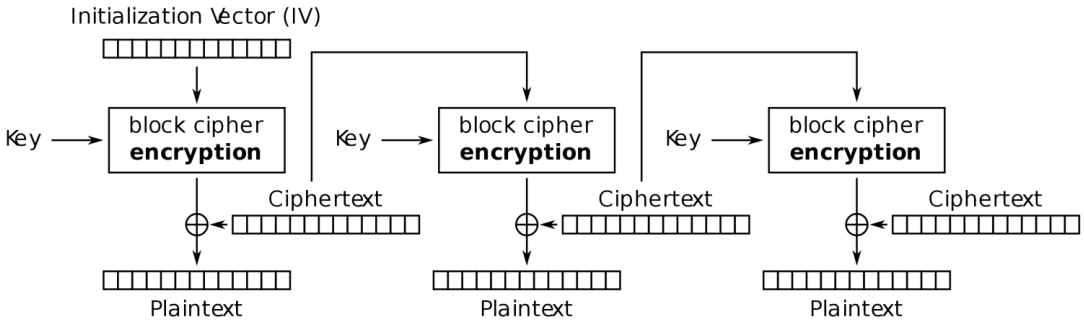
```
def OFB(state,rs,way,M):
   M2 = [M[i:i + 16] for i in range(0, len(M), 16)]
   res = []
   for m in M2:
      state = deal(rs, state, way) #结果为十进制
      res += XOR(m, state)
      #把state转化为下次方便加密的形式
      new_state=[]
      for i in range(4):
         t = hex(state[i])[2:].zfill(8)
         for j in range(0, len(t), 2):
            new_state.append(t[j:j + 2])
      state=new_state
   return res
```

4.5 SM4 的密文反馈模式 CFB

CFB 实际上与 OFB 非常相似，从下图的示意图中便可看出，在 OFB 中作为新状态的内容为上一状态与密钥加密的结果；而在 CFB 中作为新状态的内容为上一状态与密钥加密的结果再与明文异或后的结果。



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

CFB 与 OFB 的实现流程也大同小异，只需更改新状态的赋值时刻即可，如下所示：

```
def CFB(state,rs,way,M): #加密
    M2 = [M[i:i + 16] for i in range(0, len(M), 16)]
    res = []
    for m in M2:
        state = deal(rs, state, way) #结果为十进制
        state=XOR(m, state)
        res += state

    return res

def CFB_D(state,rs,way,M): #解密
    M2 = [M[i:i + 16] for i in range(0, len(M), 16)]
    res = []
    q=0
    for m in M2:
        if q==0:
            state = deal(rs, state, way) #结果为十进制
            state=XOR(m, state)
```

```
            res += state
        else:
            state = deal(rs, M2[q-1], way) # 结果为十进制
            state = XOR(m, state)
            res += state
        q+=1

    return res
```

4.6 SM4 的 X CBC 模式

XCBC（Extended Cipher Block Chaining）和 CBC（Cipher Block Chaining）都是分组密码模式，但 XCBC 在一些方面对 CBC 进行了扩展以提供更好的安全性。**XCBC 工作过程如下：**

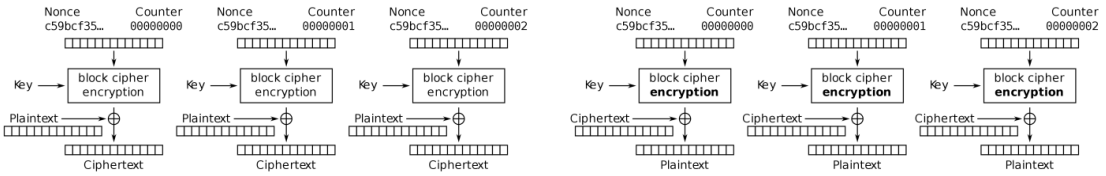
1. **初始阶段：**
   - 使用三个密钥：K_1, K_2, K_3。
   - 使用两个初始向量：IV_1（第一个块）和 IV_2（后续的块）。
2. **加密阶段：**
   - 第一个块：$C_1 = E_{K_1}(IV_1 \oplus P_1)$
   - 后续块：$C_i = E_{K_1}(IV_2 \oplus P_i \oplus C_{i-1})$，其中 $i > 1$。
3. **重复：**
   - 重复上述过程，直到所有的明文块都被加密。

其中 XCBC 还可以处理任意长度的数据，当最后一个明文块的为**短块**，即不是 128bit 时，采用填充算法，先填充 1 一个 1，剩下的位置全部填 0。代码实现如下：

```
def pad(m):
    if len(m)==16:
        return m
    elif len(m)>16:
        raise ValueError("明文切割出错")
    else:
        for i in range(16-len(m)):
            if i==0:
                m.append('10')
            else :
                m.append('00')
        return m
```

4.7 SM4 的计数器模式 CTR

CTR 模式与 OFB 十分相似，甚至还更为简单。主要的区别即在状态值上变化的体现（在 CTR 中状态值实际上就是计数器 counter 的值），如下所示：



代码实现如下：

```
def CTR(M,rs,counter,way):
    CP=copy.deepcopy(counter)
    # last_value=counter[-1]
    M2 = [M[i:i + 16] for i in range(0, len(M), 16)]
    res = []

    for m in M2:
        res += XOR(m, deal(rs, CP, way))
        CP[-1]=hex(int(CP[-1],16)+1)[2:].zfill(2)
    #counter[-1]=last_value
    return res
```

在代码实现中需要注意的是，在加解密过程中 counter 的初始值应该相同。在 counter 的类型上我选用 python 中的 list 类型，但 list 类型在函数处理过程中会修改原始的变量值，因此需要使用**深 copy** 或**函数返回前恢复初始值**的方式来确保 counter 的初始值不变；否则解密会得到错误结果。

五、 实验结果与总结

在结果展示阶段，使用的明文、密钥、初始状态值 (OFB,CFB 模式下使用) 如下所示：

```
M=
['01','23','45','67','89','ab','cd','ef','fe','dc','ba','98','76','54','32',
'10','ab','cd','12','34','ef','34','ab','fa','fe','dc','ba','98','76','54',
'32','10']

K=
['01','23','45','67','89','ab','cd','ef','fe','dc','ba','98','76','54','32','10']

state=
['ee','aa','47','a7','bf','ff','fd','1f','9e','dc','b6','78','66','e4','d2','1b']
```

而 CTR 中使用的计数器采用随机生成的方式：

```
counter=[]
for i in range(16):
   counter.append(hex(random.randint(0, 2**8- 1))[2:].zfill(2))
```

运行代码得到各组模式的加解密结果如下图所示：

```
D:\Anaconda\envs\pytorch\python.exe D:\python\密码学\SM4\sm4.py
ECB工作模型加解密结果
加密681edf34d206965e86b3e94f536e42469493b356e8ae1eaad324a6de81726b0b
解密0123456789abcdeffedcba9876543210abcd1234ef34abfafedcba9876543210
--------------------------------------------------------------------------------------------------------
CBC工作模型加解密结果
加密491ec62ab79cbf2f851b49b5339c44c89f9648d72551ae74001cce1808c2a8cd
解密0123456789abcdeffedcba9876543210abcd1234ef34abfafedcba9876543210
--------------------------------------------------------------------------------------------------------
OFB工作模型加解密结果
加密f2790b9e4b04049114d05134b75925391c39377539c2a58f00199941209ca355
解密0123456789abcdeffedcba9876543210abcd1234ef34abfafedcba9876543210
--------------------------------------------------------------------------------------------------------
CFB工作模型加解密结果
加密f2790b9e4b04049114d05134b7592539c44bf6ab91ea95965a46a35ff30ed707
解密0123456789abcdeffedcba9876543210abcd1234ef34abfafedcba9876543210
--------------------------------------------------------------------------------------------------------
CTR工作模型加解密结果
加密dbbb6fa29f2e83443e9d7e365f878a48273f37d3c1a766dca3b6633b10313476
解密0123456789abcdeffedcba9876543210abcd1234ef34abfafedcba9876543210
--------------------------------------------------------------------------------------------------------

进程已结束，退出代码为 0
```

同时为了展示 XCBC 中明文填充效果，选用明文 M 如下：

```
M=['01','23','45','67','89','ab','cd','ef','fe','dc','ba','98']
```

可以看到此时 M 的长度并不是 128bit 的倍数，运行填充代码得到结果如下所示；

```
['01', '23', '45', '67', '89', 'ab', 'cd', 'ef', 'fe', 'dc', 'ba', '98', '10', '00', '00', '00']
```

此时 M 已经被填充为了 128bit 的倍数 (由于此处采用字节为单位存储，因此 list 的长度现在被填充为了 16 的倍数)。

复现代码已经上传至 Github 之中：SM4 以及 6 种工作模式 python 复现代码

各种分组密码工作方式的特点可总结为下表所示：

工作模式	电码本模式 ECB	明密文链接模式	密文链接模式 CBC	输出反馈模式 OFB	密文反馈模式 CFB	XCBC 模式	计数器模式 CTR
能否掩盖数据模式	不能	能	能	能	能	能	能
加密错误传播无界	否	是	是	否	是	是	否
解密错误传播无界	否	是	否	否	是	否	否
是否改变消息长度	否	否	否	否	否	否	否
能否处理消息短块	不能	不能	不能	能	能	能	能
执行速度	快	适中	适中	快	快	慢	快

5.1 个人收获

通过此次实验，我学习了 SM4 加密算法的实现过程，学习了其中各类部件的工作原理，并且能够使用代码进行实现；其次，学习了 6 种分组密码常用的工作模式，明确不同工作模式下的优缺点，并且以 SM4 为示例展现了 6 种工作模式的加解密结果，加深了对分组密码的理解。

在本次实验中也踩了一些坑，例如一开始理解 SM4 解密算法出错，进行了两次逆转操作、错误理解工作模式等，并且代码中使用了较多篇幅对明文格式与加密结果格式进行转换，下次尝试能否直接重载运算符来简化这一过程；在 CTR 工作模式的实现中还踩了 list 深浅拷贝的坑。

参考文献

[1] SM4 简介（csdn）：http://t.csdnimg.cn/NqUZu

[2] SM4 加密算法原理和简单实现（java）（csdn）：http://t.csdnimg.cn/HTALK

[3] 密码学引论 (第三版)

[4] DES 加密算法中，ECB 和 CBC 模式有什么区别?（csdn）：http://t.csdnimg.cn/iFdWB

[5] 分组密码算法的工作模式——CTR 模式（csdn）：http://t.csdnimg.cn/PR54i

教师评语评分

评语：

评　　分：

评 阅 人：

评阅时间：