《密码学》课程设计实验报告

***本次实验源文件和实验报告均已上传至 github 链接下的 AES 文件夹中：**

sterzhang/Cryptology: Jianshu Zhang's cryptological experiments (github.com)

实验序号：03　　实验项目名称：分组密码工作模式

学　　号	2021302181216	姓　　名	张鉴殊	专业、班	21 信安
实验地点	C202	指导教师	余荣威	时间	2022.12.8

一、实验目的及要求

教学目的：

(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。

实验要求：

(1)掌握分组密码的 ECB、CBC、OFB、CFB、CTR 等常用工作模式；

(2)掌握分组密码的短块加密技术；

(3)熟悉分组密码各工作模式的（数据掩盖、错误传播、效率等）特点；

(4)利用分组密码工作模式和短块处理技术实现任意长度输入的加密与解密。

二、实验设备（环境）及要求

Windows 操作系统，高级语言开发环境

三、实验内容与步骤

本次实验采用的是 AES 算法结合不同的工作模式，先来看 AES 算法的基本逻辑的笔记，接下来我的加密算法以 AES 为框架：

AES 分组算法

明文长度128位

密钥长度为128位 / 192位 / 256位

128 bit → AES算法 ← 128 bit / 192 bit / 256 bit key（10轮 / 12轮 / 14轮）→ 128 bit

将明文、密钥、明文（16字节）

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

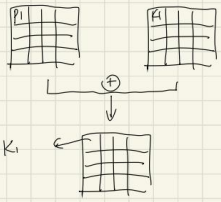
按该顺序排列

明文
↓
初始变换
↓
9轮循环运算
↓
1轮最终轮
↓
密文

1、字节代换
2、行移位
3、列混合 （最终轮不含列混合）
4、轮密钥加

初始变换

将明文块 与 密钥块 做异或运算



循环运算 1、字节代换

19			

拿出19 → (1,9) → 查 S-BOX

取出对应的值代换回去

2、行移位

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

第一行不变
第二行向左移1个字节
第三行左移2个字节
第四行左移3个字节

1	5	9	13
6	10	14	2
11	15	3	7
16	4	8	12

4、轮密钥加

得到结果和子密钥矩阵异或

密钥扩展得到的10个轮密钥

密钥扩展

W_i：
- i的倍数 $W_i = W_{i-4} \oplus T(W_{i-1})$
- 不是4的倍数 $W_i = W_{i-4} \oplus W_{i-1}$

函数T
1、字循环
2、字节轮换
3、轮常量异或

函数T
1、字循环

将

1	
2	
3	
4	

→

4	
1	
2	
3	

2、字节代换

对字循环结果进行字节代换（利用S盒）

3、轮常量异或

将前两步得到的和轮常量 RconEjJ 异或

3、列混合

左乘固定的矩阵

固定矩阵

$$\begin{bmatrix} S'_{0,0} & \cdots & \cdots & \cdots \\ S'_{1,0} & \cdots & \cdots & \cdots \\ S'_{2,0} & \cdots & \cdots & \cdots \\ S'_{3,0} & \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,0} & \cdots & \cdots & \cdots \\ S_{1,0} & \cdots & \cdots & \cdots \\ S_{2,0} & \cdots & \cdots & \cdots \\ S_{3,0} & \cdots & \cdots & \cdots \end{bmatrix}$$

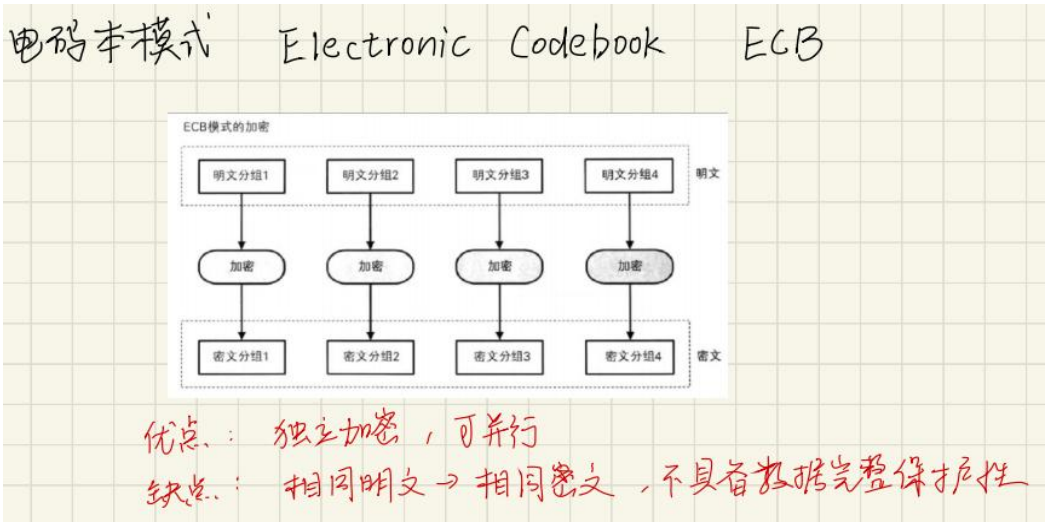
$$S'_{0,0} = (2 * S_{0,0}) \oplus (3 * S_{1,0}) \oplus S_{2,0} \oplus S_{3,0}$$

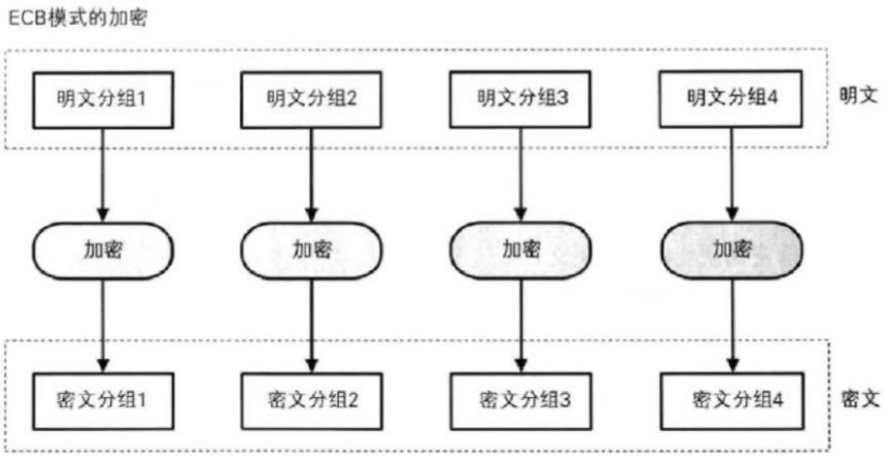
$$(00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0) = \begin{cases} (a_6a_5a_4a_3a_2a_1a_00), a_7 = 0 \\ (a_6a_5a_4a_3a_2a_1a_00) \oplus (00011011), a_7 = 1 \end{cases}$$

$$(00000011) * (a_7a_6a_5a_4a_3a_2a_1a_0) = [(00000010) \oplus (00000001)] * (a_7a_6a_5a_4a_3a_2a_1a_0)$$
$$= [(00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0)] \oplus (a_7a_6a_5a_4a_3a_2a_1a_0)$$

1. 分组密码的常用工作模式

（1）电码本模式 ECB（教材 p124 式 3-76）





这里为了测试不同的加密模式，AES 的明文我设置成 256 位的，这样就能分成 2 个 128 的块进行加密，实现各种模式的加密比对。这里我首先来实现最简单的 AES 结合 ECB 加密模式，这里可以看到明文传进来我先进行了分割，按照 16 字节为一组划分，分别存进 block 里，后续对每个 block 进行加密。由于这里是最简单的 ECB 加密，所以直接遍历依次进行加密即可，最后再拼接起来。

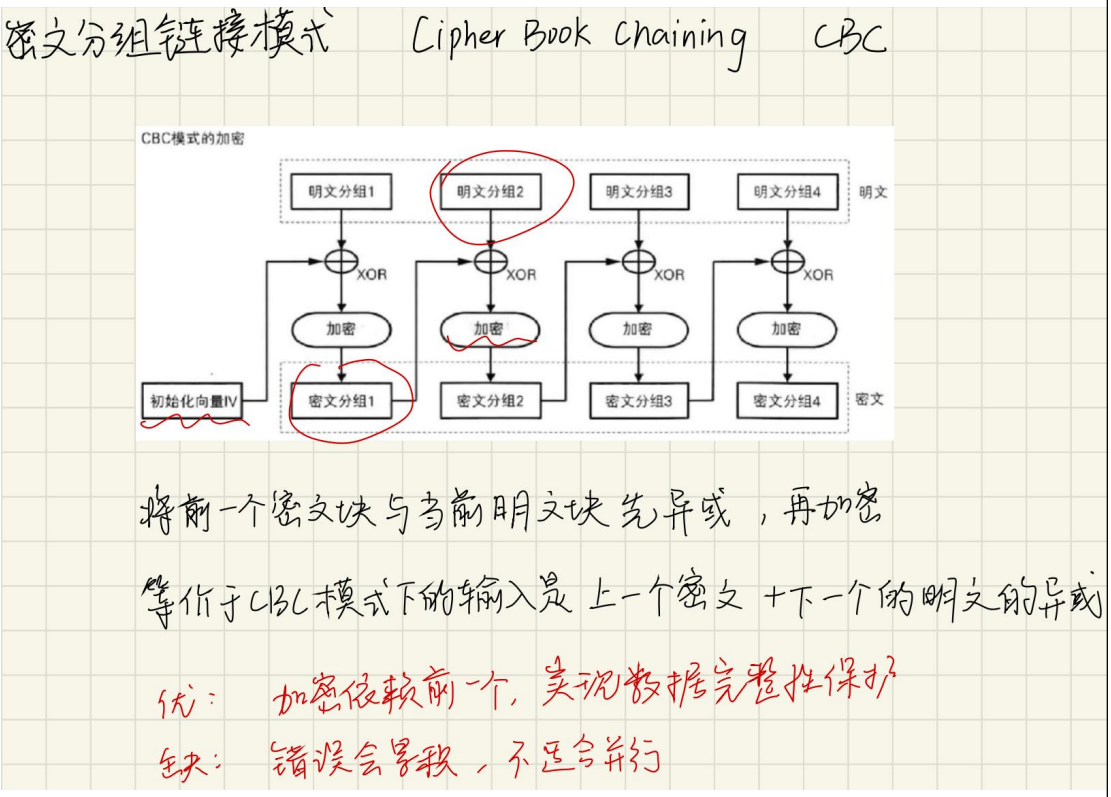
```
    # AES + ECB(电码本模式)

    def aes_encrypt(self, plaintext, RoundKeys):
        # 将明文分割成 128 位（16 字节）的块
        plaintext_blocks = [plaintext[i:i+16] for i in range(0, len(plaintext),
16)]
        encrypted_blocks = []
```

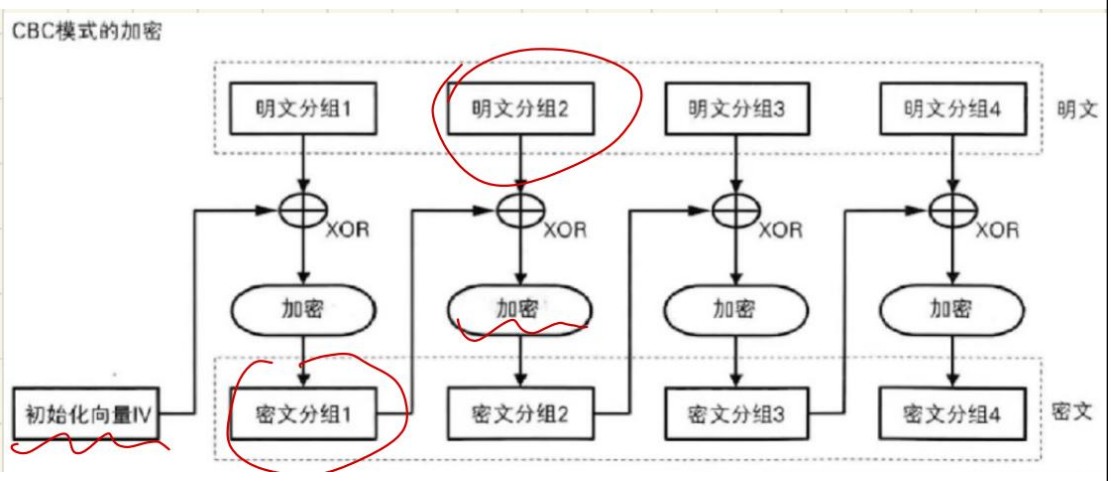
```
        for plaintext_block in plaintext_blocks:
            State = plaintext_block
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
            State = bytes(State)  # 将列表转换为字节序列
            encrypted_blocks.append(State)

        # 将加密后的块合并为一个字节序列
        encrypted_data = b''.join(encrypted_blocks)
        return encrypted_data
```

解密也是如此，只不过传进来的是密文。

```
    def aes_decrypt(self, ciphertext, RoundKeys):
        # 将密文分割成 128 位（16 字节）的块
        ciphertext_blocks = [ciphertext[i:i+16] for i in range(0,
len(ciphertext), 16)]
        decrypted_blocks = []
        for ciphertext_block in ciphertext_blocks:
            State = ciphertext_block
            State = self.AddRoundKey(State, RoundKeys, 10)
            for Round in range(1, 10):
                State = self.ShiftRows_Inv(State)
                State = self.SubBytes_Inv(State)
                State = self.AddRoundKey(State, RoundKeys, 10 - Round)
                State = self.MixColumns_Inv(State)
            State = self.ShiftRows_Inv(State)
            State = self.SubBytes_Inv(State)
            State = self.AddRoundKey(State, RoundKeys, 0)
            State = bytes(State)  # 将列表转换为字节序列
            decrypted_blocks.append(State)
        # 将解密后的块合并为一个字节序列
        decrypted_data = b''.join(decrypted_blocks)
        return decrypted_data
```

（2）密文链接模式 CBC（教材 p125 图 3-30、31）





这里我为 CBC 密文分组链接模式定义了一个新的函数 ase_encrypt_cbc，这里是将前一个密文块和当前的明文块进行异或，这里也定义了 xor_blocks 用来执行异或操作。可以看到我下面的具体实现，将异或后的结果送到加密过程中，实现了 CBC。另外这里是用随机生成的 16 字节 IV 作为初始化向量，保证了 IV 的随机性和唯一性，但是这里为了验证加密和解密正确性，我设置可以复现的 seed，以便观测结果。

```
    # 设置固定的种子
    random.seed(12345)
    # 生成一个 16 字节的伪随机 IV
    IV = bytes([random.randint(0, 255) for _ in range(16)])
```

这里为加密过程，先将第一个块与 IV 异或，随后的块都和前一个密文进行异或。

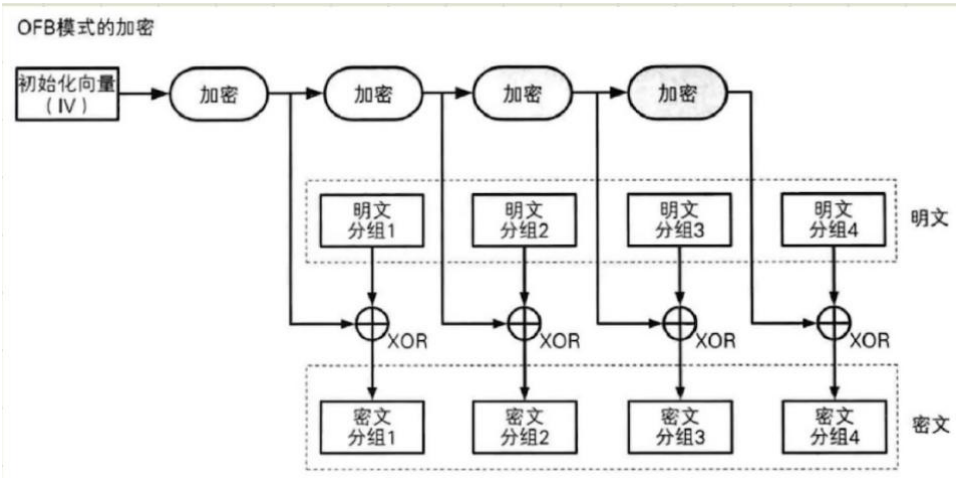
```
    # AES + CBC(密文分组模式)
    def xor_blocks(self, block1, block2):
        return bytes([b1 ^ b2 for b1, b2 in zip(block1, block2)])

    def aes_encrypt_cbc(self, plaintext, RoundKeys):
        # 初始化向量
        iv = self.IV  # 确保在类中定义了 IV
        # 将明文分割成 128 位（16 字节）的块
        plaintext_blocks = [plaintext[i:i+16] for i in range(0, len(plaintext),
16)]
        encrypted_blocks = []
        previous_block = iv
        for plaintext_block in plaintext_blocks:
            # 将当前明文块与前一个加密块（或 IV）异或
            State = self.xor_blocks(plaintext_block, previous_block)
            # 加密异或后的块
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
            # 保存加密块，用作下一个块的 IV
            previous_block = State
            State = bytes(State)  # 将列表转换为字节序列
            encrypted_blocks.append(State)

        # 将加密后的块合并为一个字节序列
        encrypted_data = b''.join(encrypted_blocks)
        return encrypted_data
```

解密过程和加密可以看做互逆的过程，最后一次再和 IV 进行异或。

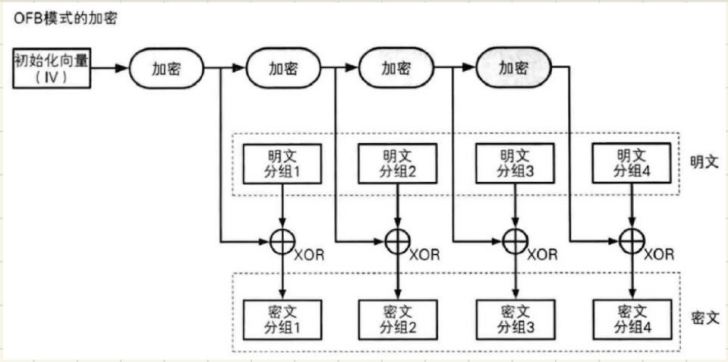
```
    def aes_decrypt_cbc(self, ciphertext, RoundKeys):
```

```
        iv = self.IV
        # 将密文分割成 128 位（16 字节）的块
        ciphertext_blocks = [ciphertext[i:i+16] for i in range(0,
len(ciphertext), 16)]
        decrypted_blocks = []
        previous_block = iv
        for ciphertext_block in ciphertext_blocks:
            # 解密块
            State = ciphertext_block
            State = self.AddRoundKey(State, RoundKeys, 10)
            for Round in range(9, 0, -1):
                State = self.ShiftRows_Inv(State)
                State = self.SubBytes_Inv(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
                State = self.MixColumns_Inv(State)
            State = self.ShiftRows_Inv(State)
            State = self.SubBytes_Inv(State)
            State = self.AddRoundKey(State, RoundKeys, 0)
            # 将解密后的块与前一个密文块（或 IV）异或
            State = self.xor_blocks(State, previous_block)
            decrypted_blocks.append(State)
            # 更新 previous_block 为当前的密文块
            previous_block = ciphertext_block

        # 将解密后的块合并为一个字节序列
        decrypted_data = b''.join(decrypted_blocks)
        return decrypted_data
```

（3）输出反馈模式 OFB（教材 p127 图 3-32）



输出反馈模式 Output Feedback OFB



将前一个加密输出作输入加密，生成密钥流，再与当前明文块异或得到密文块

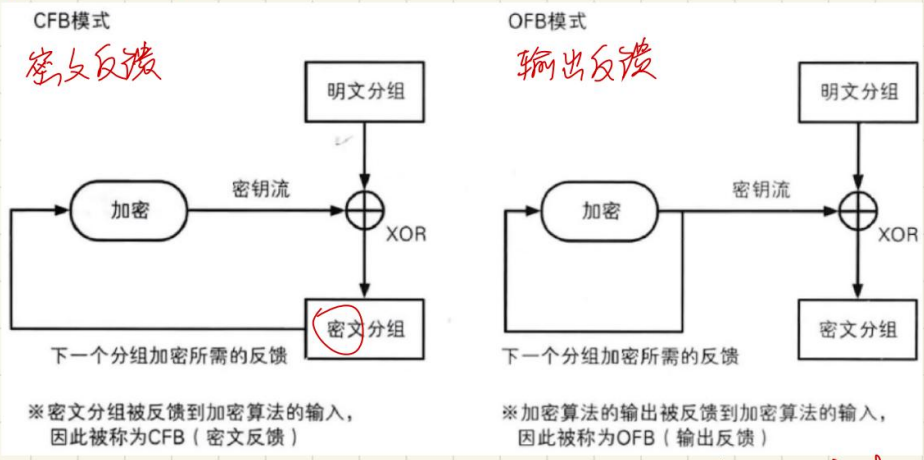
因为密钥流与明、密文无关

优：可变长度的加密操作；实时性；对密文分组的错误不敏感

缺：传输错误不可恢复；无法完整性保护；不并行

后面要介绍的 CFB 和当前的 OFB 模式都类似流密码，两者的明文均不参与加密。

都类似流密码



两者的明文均不参与加密，而是和密钥流进行异或

OFB 的加密模式较简单，逻辑也很简单，实现较容易。只用将最初的 IV 的不断地进行加密，再和明文异或即可，所以在实现中，我就把每一次加密后的 IV 作为下一次加密的输入，用输出和明文异或得到密文，最后再拼接起来即可。

```
    # AES + OFB(输出反馈模式)
```

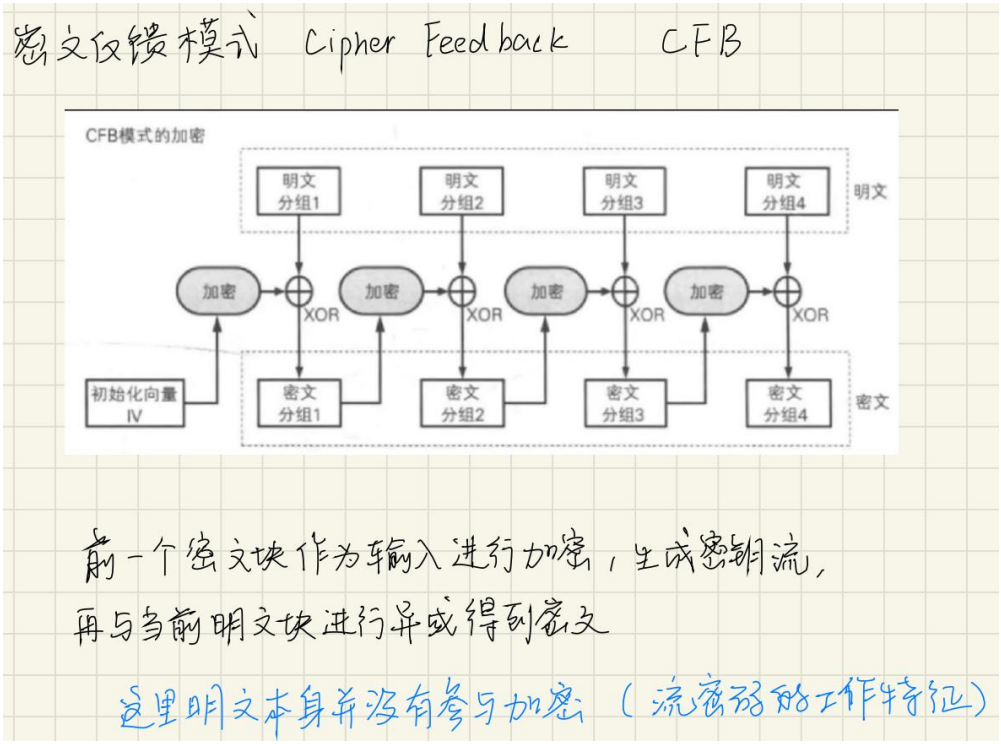
```
    def aes_encrypt_ofb(self, plaintext, RoundKeys):

        # 将明文分割成 128 位（16 字节）的块
        plaintext_blocks = [plaintext[i:i+16] for i in range(0, len(plaintext),
16)]
        encrypted_blocks = []
        output_block = self.IV
        for plaintext_block in plaintext_blocks:
            State = output_block
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
            State = bytes(State)  # 将列表转换为字节序列
            encrypted_block = self.xor_blocks(State, plaintext_block)
            encrypted_blocks.append(encrypted_block)

        # 将加密后的块合并为一个字节序列
        encrypted_data = b''.join(encrypted_blocks)
        return encrypted_data
```

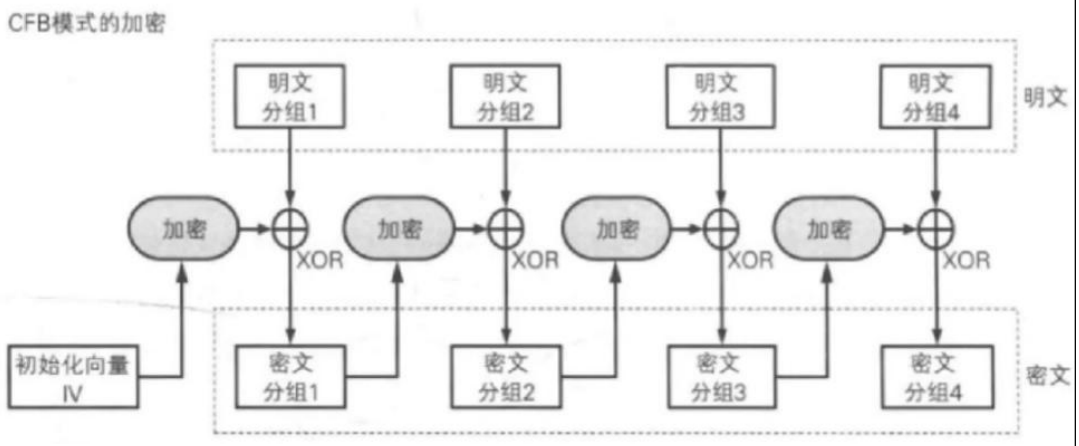
解密过程和加密差不多，也是用一个 IV 一路加密到底

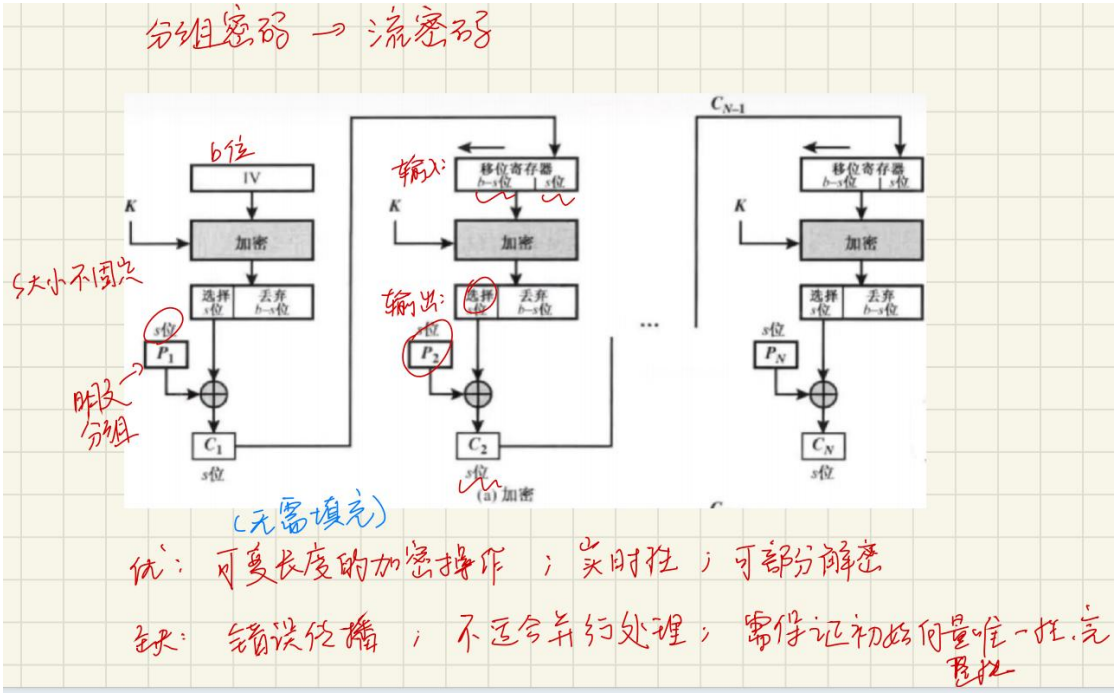
```
    def aes_decrypt_ofb(self, ciphertext, RoundKeys):
        # 将明文分割成 128 位（16 字节）的块
        ciphertext_blocks = [ciphertext[i:i+16] for i in range(0,
len(ciphertext), 16)]
        decrypted_blocks = []
        output_block = self.IV
        for ciphertext_block in ciphertext_blocks:
            State = output_block
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
```

```
            State = bytes(State)  # 将列表转换为字节序列
            decrypted_block = self.xor_blocks(State, ciphertext_block)
            decrypted_blocks.append(decrypted_block)

        # 将加密后的块合并为一个字节序列
        decrypted_data = b''.join(decrypted_blocks)
        return decrypted_data
```

（4）密文反馈模式 CFB（教材 p128 图 3-33）







CFB 的加密模式最开始是用 IV 先加密，再和明文做异或，得到的密文作为下一轮的加密的输入，实现如下。

```
# AES + CFB(密文反馈模式)
    def aes_encrypt_cfb(self, plaintext, RoundKeys):

        # 将明文分割成 128 位（16 字节）的块
        plaintext_blocks = [plaintext[i:i+16] for i in range(0, len(plaintext),
16)]
        encrypted_blocks = []
        output_block = self.IV
        for plaintext_block in plaintext_blocks:
            State = output_block
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
            State = bytes(State)  # 将列表转换为字节序列
            encrypted_block = self.xor_blocks(State, plaintext_block)
```

```
            encrypted_blocks.append(encrypted_block)
            output_block = encrypted_block

        # 将加密后的块合并为一个字节序列
        encrypted_data = b''.join(encrypted_blocks)
        return encrypted_data
```

解密的时候和加密的过程有点区别，用的是每一轮的密文作为加密的输入，而不是得到的明文，这里我一开始理解的有点问题，卡了一段时间。注意第一次还是用的 IV，然后用第一次对应解密的密文去作为下一次加密的输入。

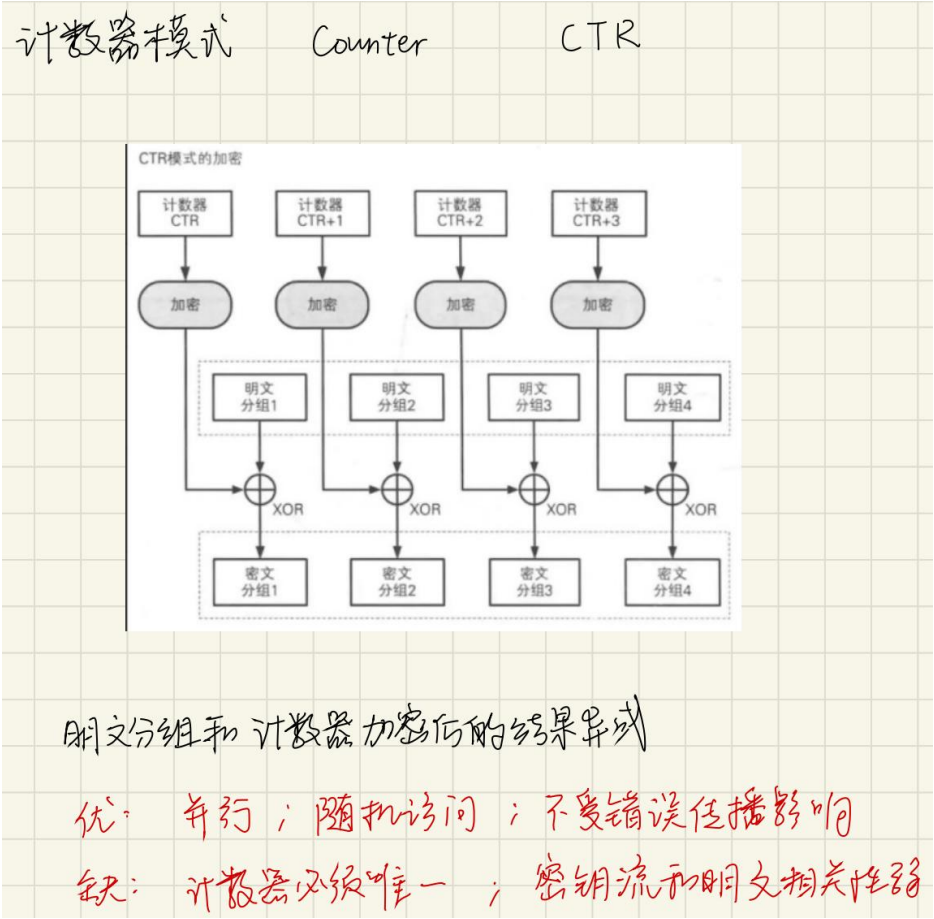
```
    def aes_decrypt_cfb(self, ciphertext, RoundKeys):
        # 将明文分割成 128 位（16 字节）的块
        ciphertext_blocks = [ciphertext[i:i+16] for i in range(0,
len(ciphertext), 16)]
        decrypted_blocks = []
        output_block = self.IV
        for ciphertext_block in ciphertext_blocks:
            State = output_block
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
            State = bytes(State)  # 将列表转换为字节序列
            decrypted_block = self.xor_blocks(State, ciphertext_block)
            decrypted_blocks.append(decrypted_block)
            output_block = ciphertext_block

        # 将加密后的块合并为一个字节序列
        decrypted_data = b''.join(decrypted_blocks)
        return decrypted_data
```

（5）X CBC 模式（教材 p128 式 3-81-83）

X CBC 模式和 CBC 的主要区别主要是明文数据的长度是否受到分组整数倍的限制。X CBC 可以处理任意长的数据，而且关键是在于处理最后一个数据块的不同，CBC 要求最后一个数据块是标准块，不能是短块。该方法就是在短块最后的第一位填充上 0x80，其它位填充 0。由于在这里前面已经实现过 CBC 了，所以主要

来实现填充部分，写在 2. 分组密码的短块处理技术（1）填充法中。

（6）计数器模式（教材 p128 式 3-84、85）



这里 CTR 首先置为 0，每轮后自增 1，且在最前面定义了如下的 Nounce 作为与计时器拼接的固定数值。在 CTR（计数器）模式下，不是直接加密数据，而是加密一个连续递增的计数器值。每个块的加密都基于这个计数器的值。为了保证加密的安全性，计数器的初始值应该是唯一的。这通常通过一个称为“Nonce”（Number used once，即一次性数字）的值来实现。Nonce 与计数器合起来构成了每个块的输入。

```
Nounce = bytes([random.randint(0, 255) for _ in range(8)])
```

```
# AES + CTR(计数器模式)
    def aes_encrypt_ctr(self, plaintext, RoundKeys):
        # 将明文分割成 128 位（16 字节）的块
```

```
        plaintext_blocks = [plaintext[i:i+16] for i in range(0, len(plaintext),
16)]

        encrypted_blocks = []
        counter = 0
        for plaintext_block in plaintext_blocks:
            # 构造计数器块
            counter_block = self.Nounce + counter.to_bytes(16 -
len(self.Nounce), byteorder='big')
            # 加密计数器块而不是明文块
            State = counter_block
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
            State = bytes(State)  # 将列表转换为字节序列
            # 将加密后的计数器块与明文块进行异或操作
            encrypted_block = self.xor_blocks(State, plaintext_block)
            encrypted_blocks.append(encrypted_block)
            # 增加计数器
            counter += 1
        # 将加密后的块合并为一个字节序列
        encrypted_data = b''.join(encrypted_blocks)
        return encrypted_data
```

解密还是一样的，实现逻辑不难。

```
    def aes_decrypt_ctr(self, ciphertext, RoundKeys):
        # 将密文分割成 128 位（16 字节）的块
        ciphertext_blocks = [ciphertext[i:i+16] for i in range(0,
len(ciphertext), 16)]
        decrypted_blocks = []
        counter = 0
        for ciphertext_block in ciphertext_blocks:
            # 构造计数器块
            counter_block = self.Nounce + counter.to_bytes(16 -
len(self.Nounce), byteorder='big')
            # 加密计数器块
            State = counter_block
```

```
            State = self.AddRoundKey(State, RoundKeys, 0)
            for Round in range(1, 10):
                State = self.SubBytes(State)
                State = self.ShiftRows(State)
                State = self.MixColumns(State)
                State = self.AddRoundKey(State, RoundKeys, Round)
            State = self.SubBytes(State)
            State = self.ShiftRows(State)
            State = self.AddRoundKey(State, RoundKeys, 10)
            State = bytes(State)  # 将列表转换为字节序列
            # 将加密后的计数器块与密文块进行异或操作
            decrypted_block = self.xor_blocks(State, ciphertext_block)
            decrypted_blocks.append(decrypted_block)
            # 增加计数器
            counter += 1
        # 将解密后的块合并为一个字节序列
        decrypted_data = b''.join(decrypted_blocks)
        return decrypted_data
```

2. 分组密码的短块处理技术

（1）填充法

参考 X CBC 模式的填充方案

这里只需要先进行判断，然后最后一个段是否满了，如果没满则在首位补 0x80，后面补 0.

```
def aes_encrypt_cbc(self, plaintext, RoundKeys):
        # 初始化向量
        iv = self.IV  # 确保在类中定义了 IV

        # 将明文分割成 128 位（16 字节）的块
        plaintext_blocks = [plaintext[i:i+16] for i in range(0, len(plaintext),
16)]
        # 检查最后一个块的长度并进行特殊填充
        if len(plaintext_blocks[-1]) < 16:
            last_block = plaintext_blocks[-1]
            padding_needed = 16 - len(last_block)
            last_block += b'\x80' + b'\x00' * (padding_needed - 1)
            plaintext_blocks[-1] = last_block
```

（2）序列密码加密法（教材 p130 图 3-34）

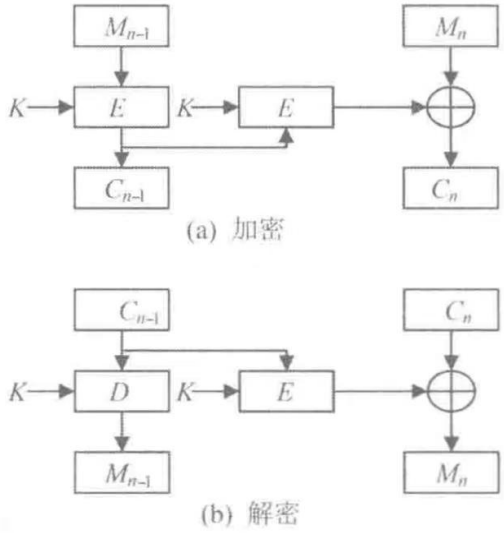
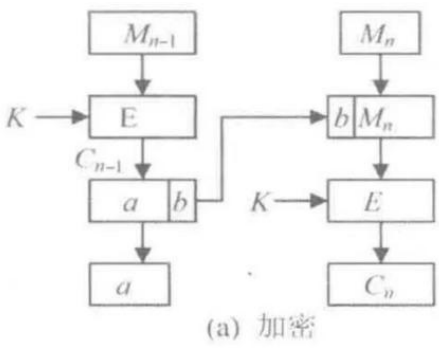


图 3-34 用序列密码加密短块

序列密码（也称为流密码）是一种加密方法，它将明文消息中的每个位或每个字符单独加密。与块密码不同，后者将数据处理成固定大小的块（例如 AES 中的 128 位），流密码在处理时没有这种块的概念，而是以连续的、位接位的方式工作。这里序列密码加密法是一种混合加密，使用分组密码和序列密码两种技术，对标准块采用分组密码加密，对短块采用序列密码加密。

（3）密文挪用技术（教材 p130 图 3-35）



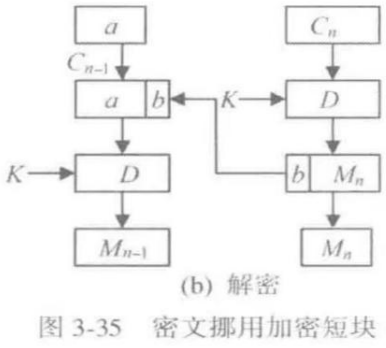


图 3-35 密文挪用加密短块

用密文挪用技术加密的短块，在加密前，首先从上一轮的密文中挪出刚好够填充

的位数，填充进去后，得到了一个标准的明文块。但此时上一轮的密文成了短块。然后在对填充后的明文加密，得到密文。这是虽然上一轮的密文成了短块，但这一轮的密文确实标准块，两者的总位数为这两轮对应明文的总位数，并没有数据扩张。解密时，先对这一轮的密文解密，还原出这一轮的明文以及从上一轮的密文中挪用的数据，再把这部分数据挪回上一轮的密文，这时再对上一轮的密文进行解密就正常了。当明文本身就是短块的时候，用初始向量 Z 代替上一轮的密文。

3. 各工作模式的特点比较

设明文 $M=(M_1, M_2, \ldots, M_n)$，相应的密文 $C=(C_1, C_2, \ldots, C_n)$。试完成下列实验，总结各工作模式的特点，并完成表格 1：

（1）选择输入消息 $M_i = M_j$，判断是否满足 $C_i=C_j$？对于不同的工作模式分别进行上述实验，得出各工作模式是否能够**掩盖明文中的数据模式的判断**。

（2）选择**篡改输入明文**中的某个分块 M_i，并**将加密后的结果与正确的密文之间进行对比**。对于不同的工作模式分别进行上述实验，得出各工作模式是否具有加密错误传播无界特性的判断。

（3）选择**篡改输入密文**中的某个分块 C_i，并将**解密后的结果与正确的明文之间进行对比**。对于不同的工作模式分别进行上述实验，得出各工作模式是否具有解密错误传播无界特性的判断。

（4）比较不同的工作模式**对于输入消息长度的要求**。

（5）比较不同的工作模式的**执行效率**。

4. 短块处理技术的比较

设明文实际长度不是分组长度的整倍数，试使用填充法、序列密码加密法、密文挪用技术进行处理。总结这三种方法的特点，并完成表格 2：

（1）是否造成短块数据扩张；

（2）试分析三种方案的安全性（提示：假设攻击者进行选择明文攻击）

四、实验结果与数据处理

表 1：各工作模式的特点

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	×	√	√	√	√	√	√
加密错误传播无界	×	√	√	×	√	√	×
解密错误传播无界	×	√	×	×	√	×	×
是否改变消息长度	×	×	×	×	×	×	×
能否处理消息短块	×	×	×	√	√	√	√
执行速度	**快**	**还行**	**还行**	**快**	**快**	**慢**	**快**

短块处理方式	填充	序列密码加密	密文挪用
短块数据扩张	√	√	×
实现难度	**易**	**中**	**难**
安全性	×	×	√

五、分析与讨论

1. DES（Data Encryption Standard）本身已经是一个完整的加密算法，可以独

立用于加密数据，那么为什么还需要像 ECB（Electronic Codebook）这样的加密模式呢？

数据加密标准（DES）本身是一个对称密钥加密算法，而不是加密模式。在实际应用中，DES 可以和不同的加密模式结合使用。这些加密模式定义了如何重复应用 DES 算法来加密较长的消息或数据流。所以需要区分两个概念，加密算法和加密模式。

加密算法（如 DES，AES）是用于加密和解密数据的具体算法。它定义了如何使用密钥对数据进行加密和解密。例如，DES 是一个经典的对称加密算法，它使用固定长度（64 位，实际有效密钥为 56 位）的密钥。

加密模式（如 ECB）定义了如何应用加密算法来加密数据流或大块数据。即使使用相同的加密算法，不同的模式可以使加密过程大不相同。ECB 是最简单的模式之一，它将数据分割为算法所需的块大小并独立加密每个块。

2. 为什么密文挪用技术不会造成短块数据扩张？

用密文挪用技术加密的短块，在加密前，首先从上一轮的密文中挪出刚好够填充的位数，填充进去后，得到了一个标准的明文块。但此时上一轮的密文成了短块。然后在对填充后的明文加密，得到密文。这是虽然上一轮的密文成了短块，但这一轮的密文确实标准块，**两者的总位数为这两轮对应明文的总位数**，并没有数据扩张。解密时，先对这一轮的密文解密，还原出这一轮的明文以及从上一轮的密文中挪用的数据，再把这部分数据挪回上一轮的密文，这时再对上一轮的密文进行解密就正常了。当明文本身就是短块的时候，用初始向量 Z 代替上一轮的密文。

六、教师评语 签名： 日期：	成绩