分组密码工作模式

清华大学计算机系
于红波
2023年3月23日

分组密码的工作模式

工作模式

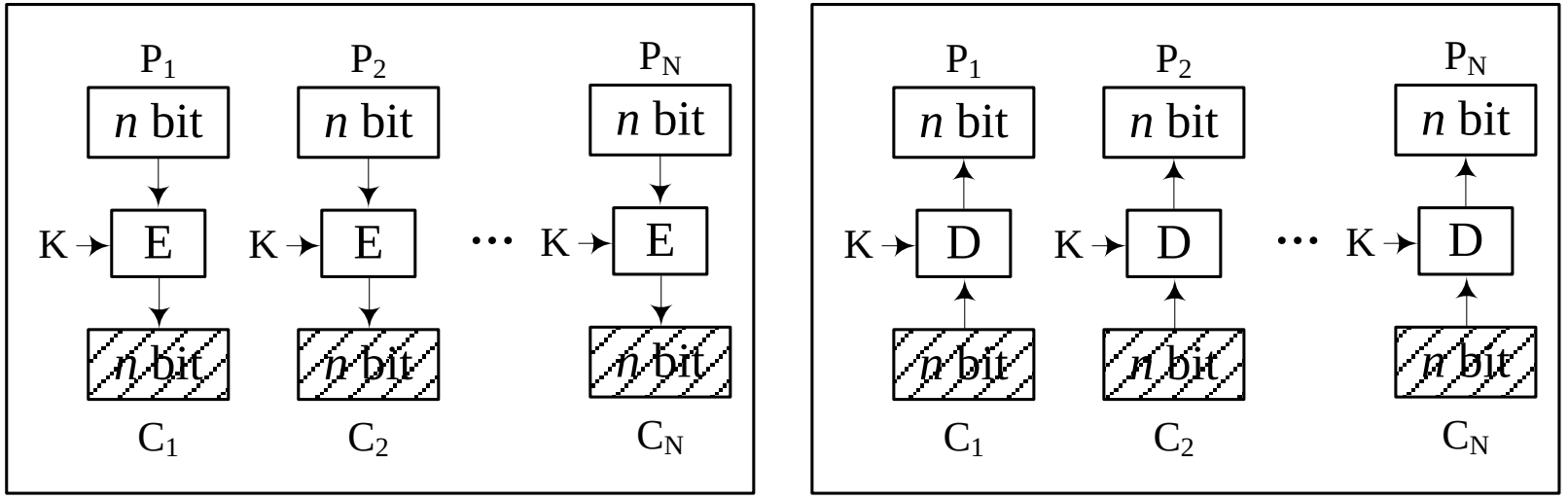
– 分组密码算法在实际中的使用方式称为工作模式。
– NIST在FIPS-PUB-81中定义了4种工作模式, 1980。
– 后又在NIST Special Publication 800－38A中增加一个，现有5个工作模式, 2001。

五种工作模式

– 电子密码本模式(Electronic Codebook Mode: ECB)
– 密码分组链接模式(Cipher Block Chaining Mode, CBC)
– 密码反馈模式(Cipher Feedback Block (Mode, CFB)
– 输出反馈模式(Output Feedback Mode, OFB)
– 计数器模式(Counter Mode, CTR)

电子密码本(ECB)模式

- ECB模式是将明文的N个分组独立地使用同一密钥K加密和解密，如下图所示。



- 可对应 2^k个密钥预先编辑 2^k个电子密码本;
- 每个密码本有 2^n个条目——（明文分组,密文分组);
- 所以称为电子密码本。不过当k和n很大时，密码本会过于庞大而无法预先编辑和保存。

ECB模式的优、缺点和应用

优点

- 实现简单；
- 不同明文分组的加密可并行实施，尤其是硬件实现时速度很快。

缺点

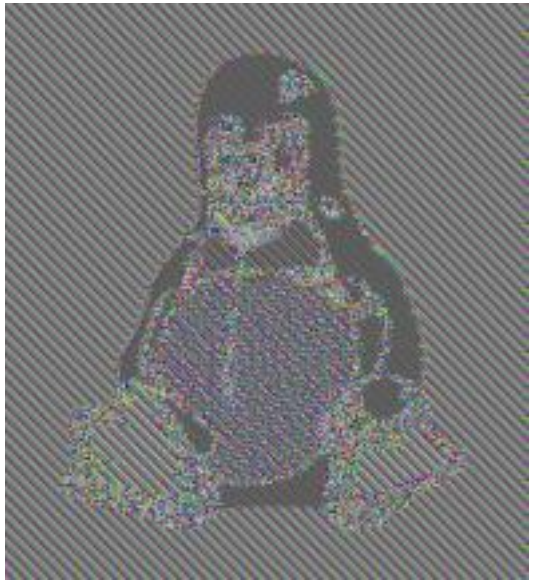
- 不同明文分组之间的加密独立进行，故保留了单表代替缺点，造成相同明文分组对应相同密文分组，因而不能隐藏明文分组的统计规律和结构规律。

典型应用

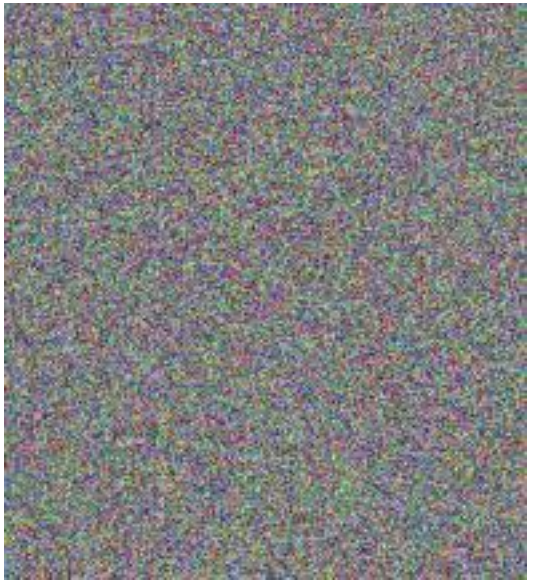
- 用于随机数的加密保护；
- 用于单分组明文的加密。

ECB模式缺点

原始图片
bmp格式



使用ECB
模式加密

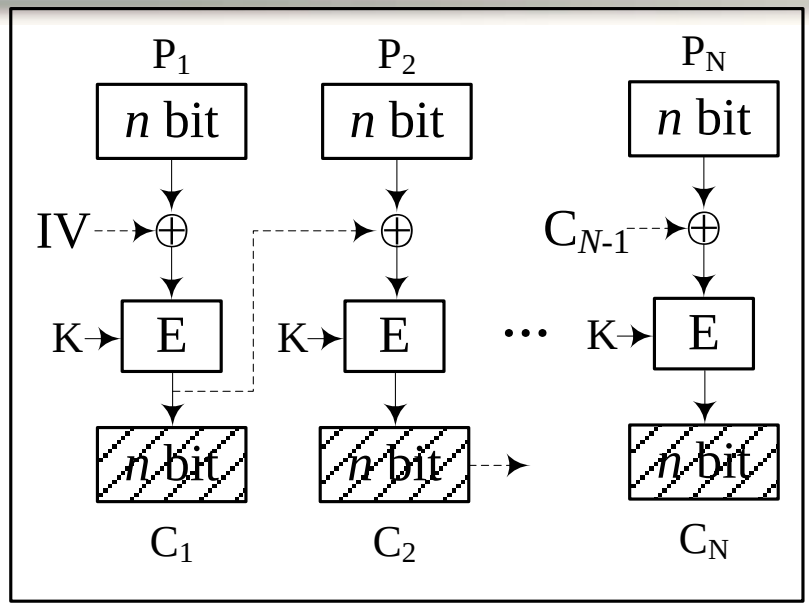


使用其他四
种模式加密

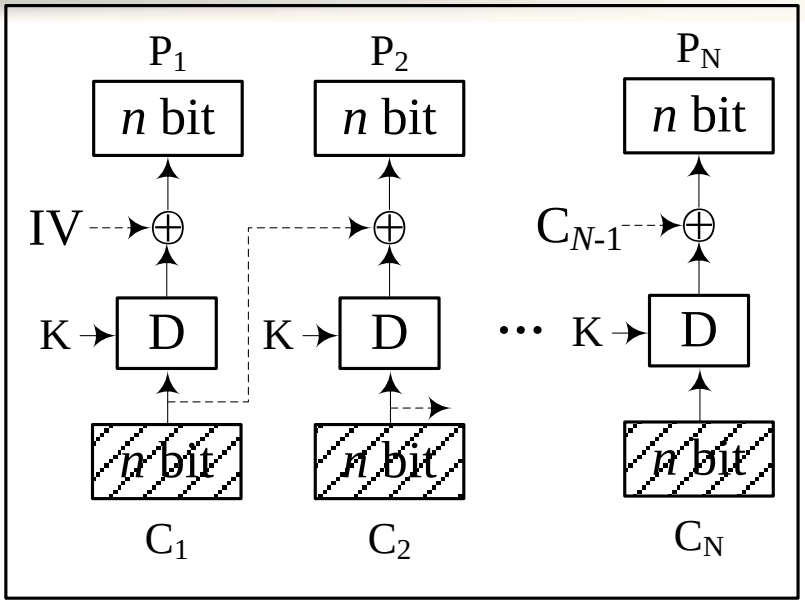
密码分组链接(CBC)模式

- 克服ECB的安全性缺陷：同一个明文分组重复出现时产生不同的密文分组。
- 简单的想法：使输出不仅与当前输入有关，而且与以前输入和输出有关——密码分组链接模式。
- CBC模式中每个明文分组在加密之前都要与以前的密文分组进行异或。第一个分组之前没有密文，故要用到一个伪分组IV。
  - IV不要求保密
  - IV必须是不可预测的，而且要保证完整性
- 在发送方，异或要在加密之前完成；在接收方，解密要在异或之前完成。

密码分组链接(CBC)模式



加密



解密

加密：

$$C_0 = IV$$
$$C_i = E_k(P_i \oplus C_{i\text{-}1})$$

解密：

$$C_0 = IV$$
$$P_i = D_k(C_i) \oplus C_{i\text{-}1}$$

CBC模式的特点

- 优点
  - 明文块的统计特性得到了隐蔽
    - CBC模式中各密文块不仅与当前明文块有关，而且还与以前的明文块及初始化向量有关，从而使明文的统计规律在密文中得到了较好的隐蔽。
  - 具有有限的(两步)错误传播特性；
  - 具有自同步功能
    - 密文出现丢块和错块不影响后续密文块的解密。若从第t块起密文块正确，则第$t+1$个明文块就能正确求出。
- 缺点
  - 加密不能并行处理（解密可以）
  - 消息是分组长度的整数倍

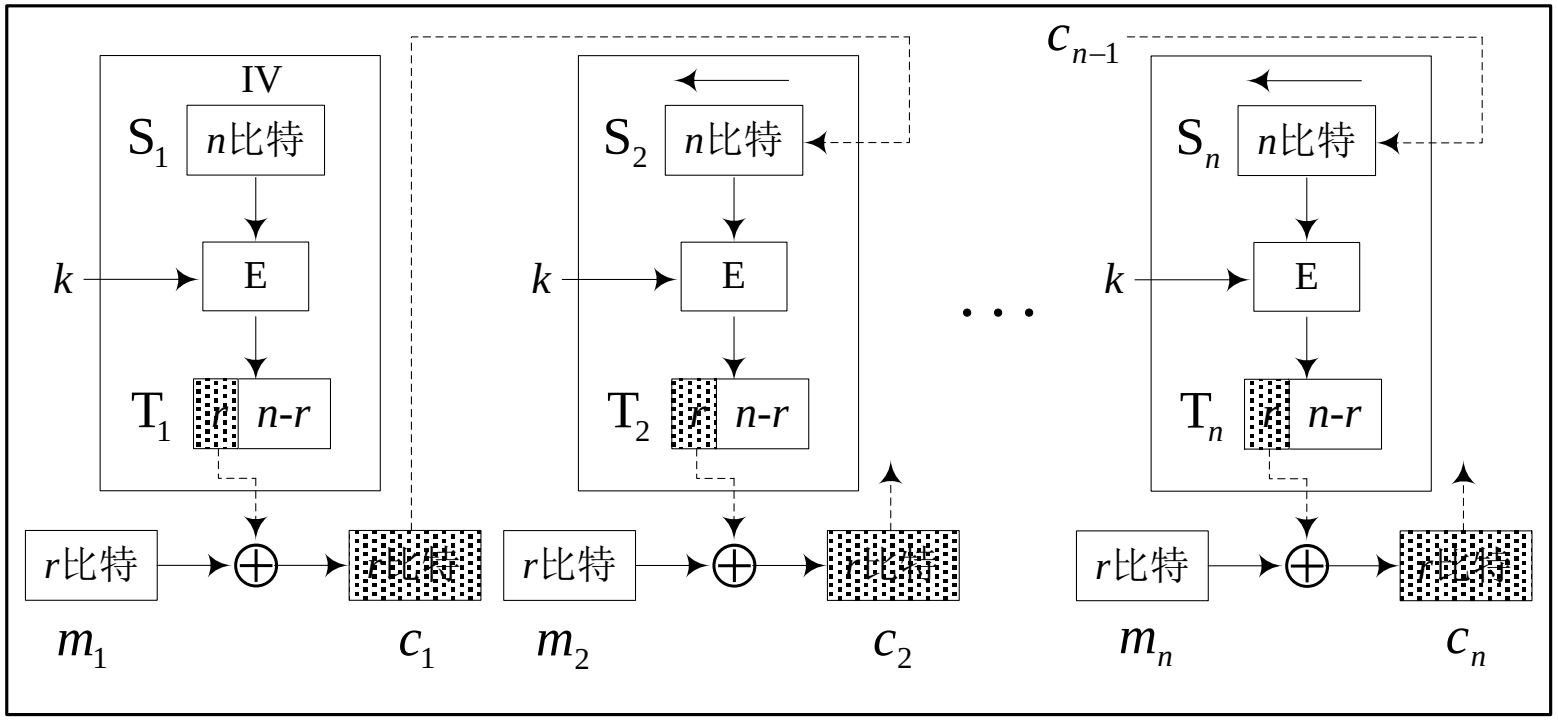
密码反馈(CFB)模式

- EBC和CBC加密，分组大小n由基本密码确定，如若用DES则n=64，用AES则n=128。
- 若待加密消息需按字符、字节或比特处理时，可采用CFB模式。其中基本密码分组大小为n，但是明文或密文分组大小为r，且$1 \le r \le n$，称为 r 比特CFB模式。
  - 如：1比特 CFB模式，8比特CFB模式，64比特CFB模式，128比特CFB模式
- 需要IV作为第一个输入分组
  - IV：不需要保密，但要不可预测
- 适用于每次处理 r比特明文块的特定需求的加密情形，能灵活适应数据各格式的需要。如数据库加密要求加密时不能改变明文的字节长度，这时就要以明文字节为单位进行加密。

密码反馈(CFB)模式

- *r*比特CFB模式中的加密如下图

IV：初始向量（S_1）　　S_i：移位寄存器（左移*r*比特）　　T_i：临时寄存器

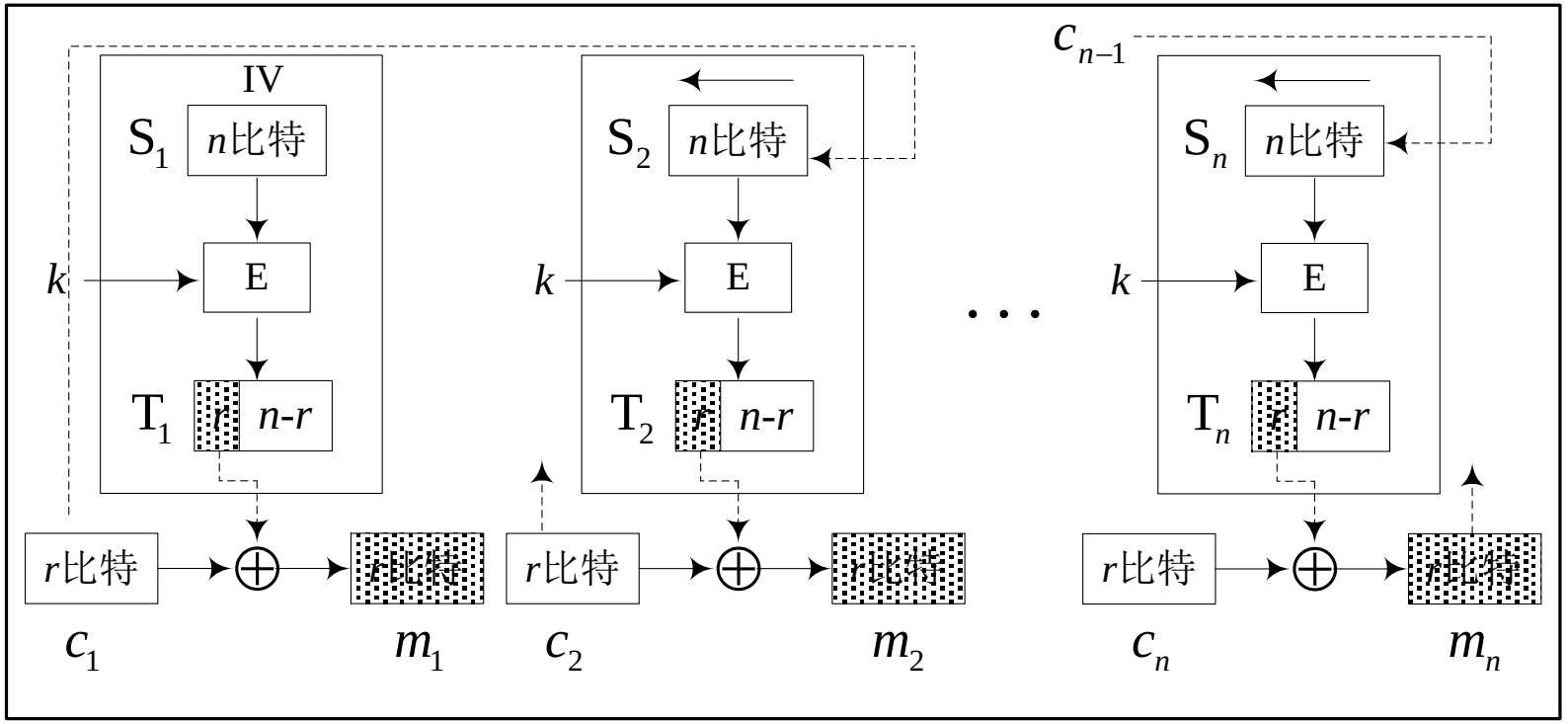


加密

密码反馈(CFB)模式

r比特CFB模式中的解密如下图

IV：初始向量（S_1）　　S_i：移位寄存器（左移r比特）　　T_i：临时寄存器



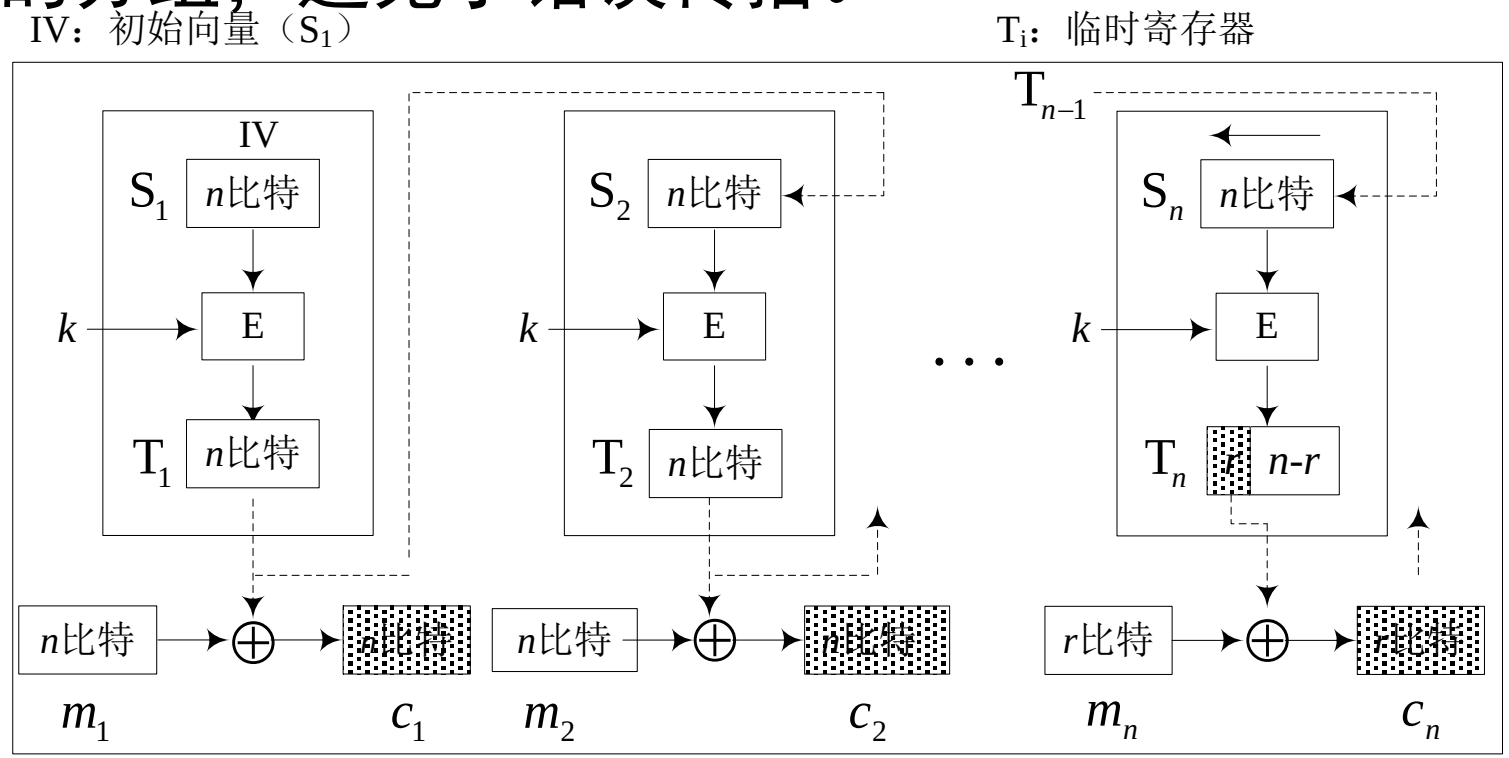
解密

CFB模式的优、缺点及应用

- 优点：
  - 适应用户不同的数据格式的需求；
  - 具有有限步的错误传播(n/r步)，可用于认证；
  - 具有自同步功能
- 缺点：
  - 加密效率低
  - 加密不能并行(解密可以并行）
  - 明文或IV的一个比特的变化影响密文所有比特
- 应用：
  - 该模式适应于数据库加密、无线通信加密等对数据格式有特殊要求

输出反馈(OFB)模式

- OFB模式在结构上类似于CFB模式，但反馈内容是分组密码输出的密钥流而不是密文。
- 因此，OFB模式的密文每一个分组都独立于先前的分组，避免了错误传播。



加密

OFB模式的优、缺点及应用

优点：

- 这是将分组密码当作同步序列密码使用的一种方式
- 不具有错误传播特性。密文的1比特错误只导致明文的1比特错误。只要密文在传输过程中不丢信号，即使信道不好，也能将明文的大部分信号正常恢复。

缺点：

- 不能实现报文的完整性认证
- IV无需保密，但是对每个消息必须选择不同的IV
- 不具有自同步能力
- 密钥序列的周期可能有短周期现象

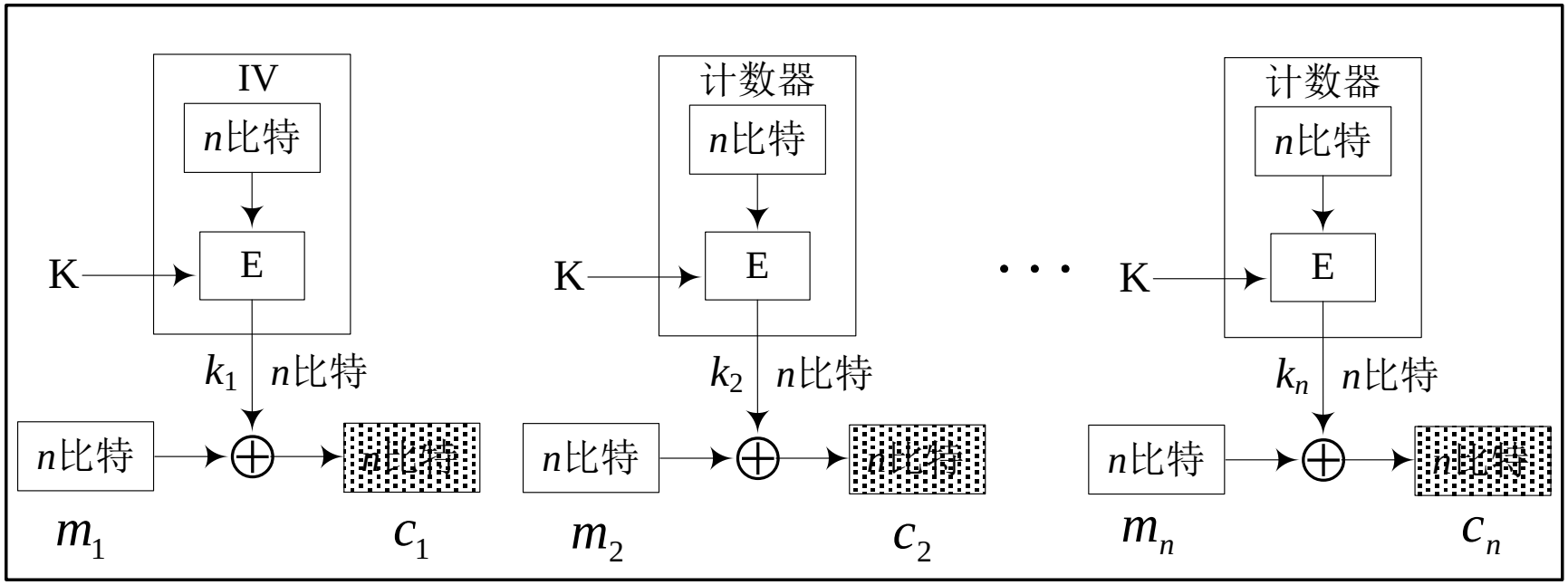
适用范围：

- 明文的冗余度特别大，信道不好但不易丢信号，明文信号出些错误也不影响效果的情形。如图象加密、语音加密等。

计数器（CTR）模式

- Diffe和Hellman设计：明密文分组与基本密码大小相同，密钥流中的伪随机数运用计数器获得，CTR模式中的加密如下图：

对每一个分组计数器都要增加



加密

计数器（CTR）模式

- 明文分组和密文分组的关系：

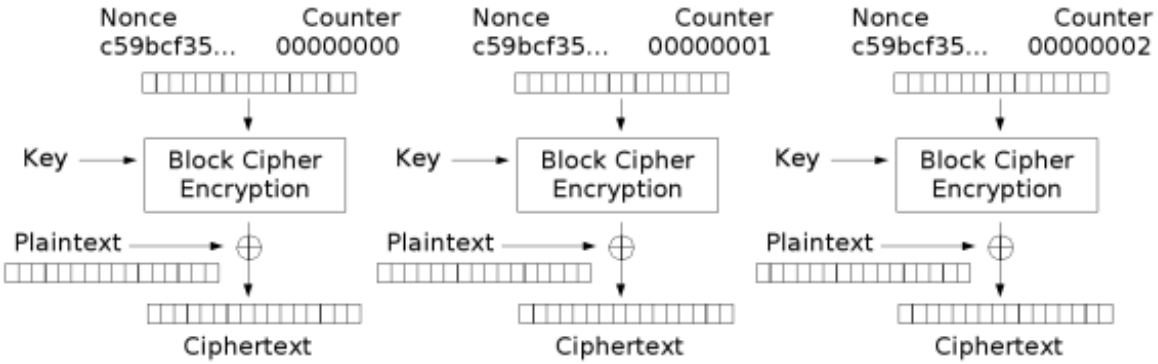
加密：$C_i = P_i \oplus E_{k_i}$（计数器）　　解密：$P_i = C_i \oplus E_{k_i}$（计数器）

- CTR与OFB、ECB比较：
  - CTR模式与OFB模式一样，创建了独立于以前密文分组的密钥流，但是不运用反馈；
  - CTR模式与ECB模式一样，创建了彼此相互独立的n比特的密文分组，他们只依赖于计数器的值。

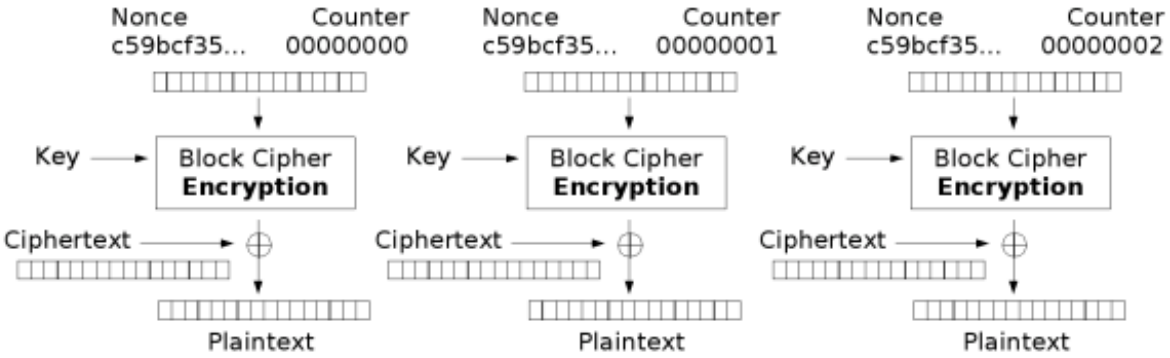
计数器分组的产生

- 方法一：所有的明文依次加密
  - 假设明文分组长度为n比特，则任意选择一个长度为n比特的计数器ctr, 则第i个分组所用的计数器为 $T_i=ctr+i-1 \bmod 2^n$
- 方法二：不同的明文间可以并行加密
  - 由两部分够成：m比特的IV和m比特计数器ctr
  - IV：不同的明文IV不同，相同的明文IV相同
  - ctr：从0计数，每个明文分组增加1

AES-CTR



Counter (CTR) mode encryption



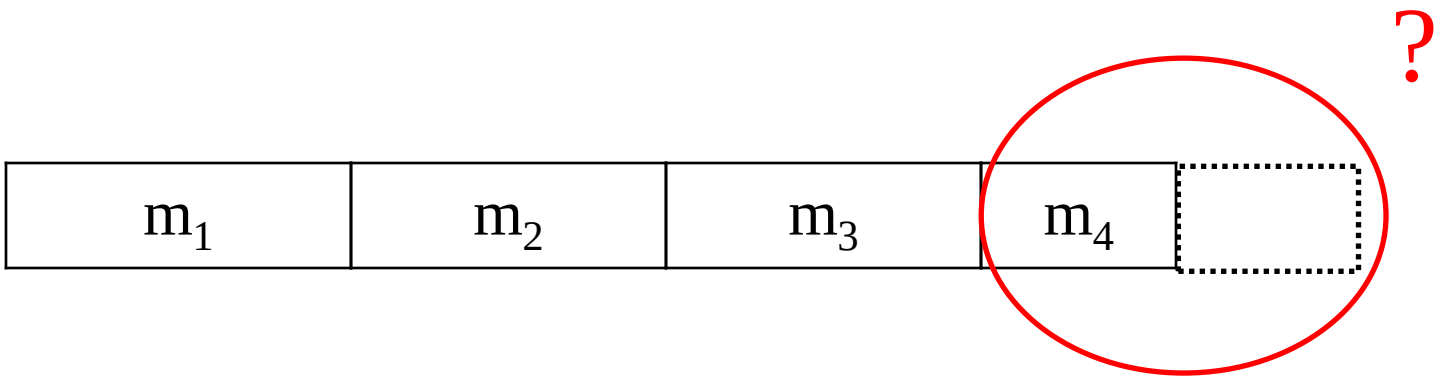
Counter (CTR) mode decryption

五种操作模式对比

操作模式	加密	结果类型	数据单位大小
ECB	每一个n比特的分组都要用相同的密码密钥独立进行加密	分组密码	n
CBC	与EBC相同，不过每一个分组都要用前面的密文进行异或	分组密码	n
CFB	每一个r比特的分组都要用一个r比特的密钥进行异或，这个密钥是前面密文的一部分	序列密码	$r \leq n$
OFB	与CFB相同，不过移位寄存器要用前面的r比特密钥更新	序列密码	$r \leq n$
CTR	与OFB相同，不过用的是计数器而不是移位寄存器	序列密码	n

如何加密一个部分分组？

- 若一个明文长度不是一个分组密码分组长度的倍数
  - 最后一个分组称为partial block



明文填充（padding）

- ECB，CBC模式需要填充
  - 填充后密文长度大于明文长度
- CFB，OFB，CTR模式不需要填充
- 填充方法
  - 比特填充
    - 在消息后面先填充一比特的“1”，然后填充若干比特的“0”，使其成为一个完整的消息分组
    - 为了使填充没有二义性，针对于这种填充方法，即使明文是一个完整的分组，也要填充；或者是使用一个明文长度指示器

... | 1011 1001 1101 0100 0010 011**1 0000 0000** |

明文填充(padding)

- 字节填充
  - PKCS7: 填充“需要填充部分”的字节数，如

... | DD DD DD DD DD DD DD DD | DD DD DD DD **04 04 04 04**

  - ANSI X.923：填充“0”字节，最后一个字节填充padding的长度,如

... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04 |

  - ISO 10126：填充若干个随机字节，最后一个字节填充padding的长度

... | DD DD DD DD DD DD DD DD | DD DD DD DD 81 A6 23 04 |

  - ISO/IEC 7816-4：与比特填充相同

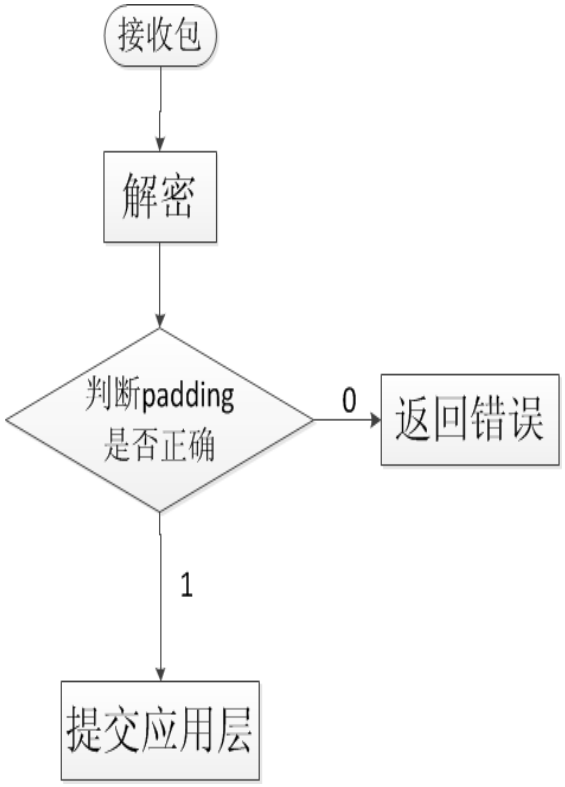
... | DD DD DD DD DD DD DD DD | DD DD DD DD **80 00 00 00** |

  - Zero填充，空格填充

Padding Oracle Attack

- Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS
- Practical Padding Oracle Attacks

基于填充格式的攻击



服务器解包过程

攻击者可以构造密文发送给服务器，通过服务器返回的信息判断构造的密文所对应的的明文是否构成一个合法padding。利用padding格式和CBC加密的特性，攻击者可以借助服务器解密任何一个block的密文。

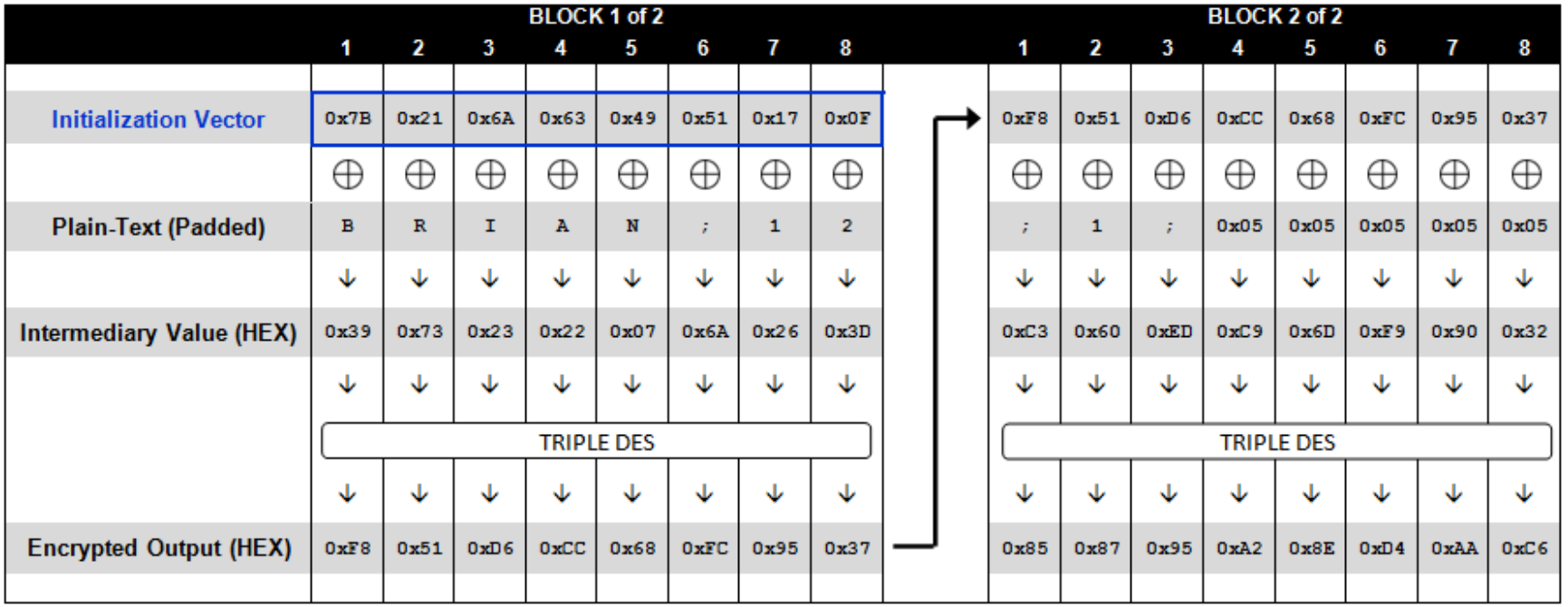
Padding Oracle

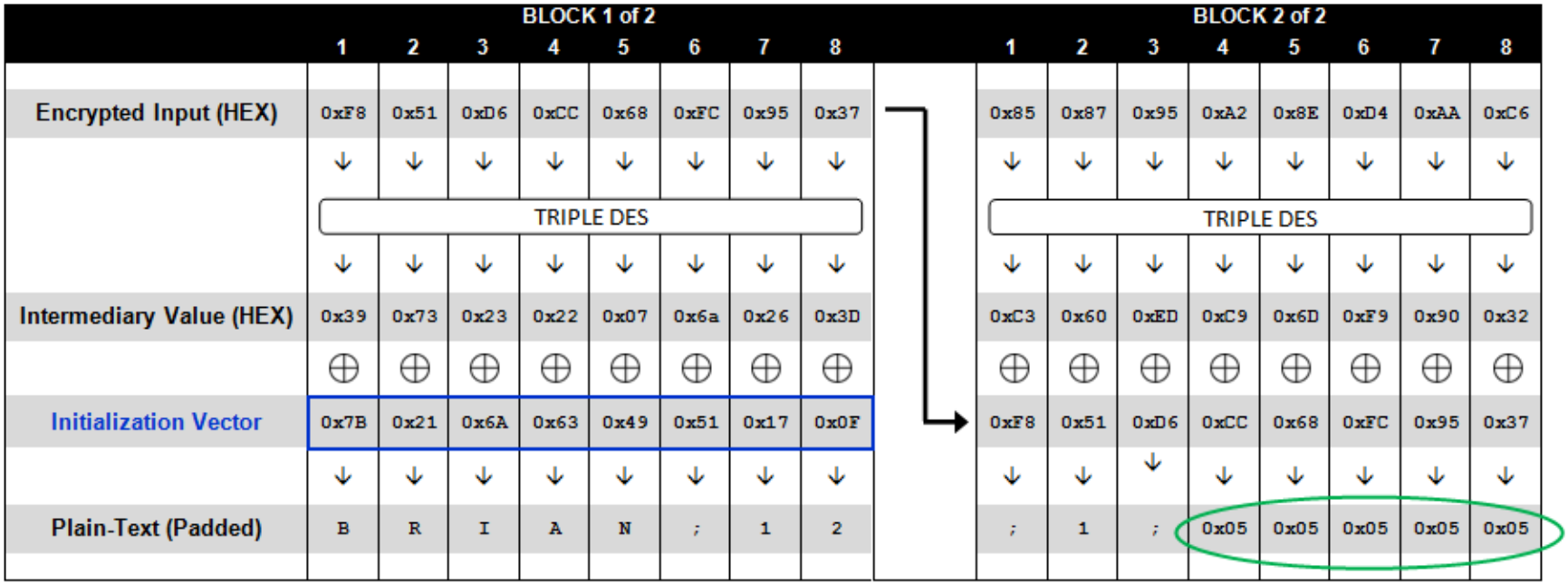
考虑以下场景：

某个应用程序使用Query String参数来传递一个用户加密后的用户名，公司ID及角色ID。参数使用CBC模式加密，每次都使用不同的初始化向量（IV，Initialization Vector）并添加在密文前段。

当应用程序接受到加密后的值以后，它将返回三种情况：

- 接受到正确的密文之后（填充正确且包含合法的值），应用程序正常返回（200 - OK）。
- 接受到非法的密文之后（解密后发现填充不正确），应用程序抛出一个解密异常（500 - Internal Server Error）。
- 接受到合法的密文（填充正确）但解密后得到一个非法的值，应用程序显示自定义错误消息（200 - OK）。





VALID PADDING

利用Padding Oricle 恢复明文

- 每次操作一个单独的加密块，独立出第一块密文（IV后的那块），在前面加上全为NULL的IV值，并发送至应用程序。以下是URL极其相关回复：
- Request:http://sampleapp/home.jsp?UID=0000000000000000F851D6CC68FC9537
  Response: 500 - Internal Server Error

	BLOCK 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D

INVALID PADDING

- 将IV加1，并发送同样的密文，看看会发生什么：
- Request:http://sampleapp/home.jsp?UID=00000000000001F851D6CC68FC9537
  Response: 500 - Internal Server Error

	BLOCK 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3C

INVALID PADDING

- 重复发送这样的请求，每次将IV的最后一个字节加1（直至0xFF），最终将会产生一个合法的单字节填充序列（0x01）。对于可能的256个值中，只有一个值会产生正确的填充字节0x01。这时得到回复结果：
- Request: http://sampleapp/home.jsp?UID=00000000000003CF851D6CC68FC9537
  Response: 200 OK

	Block 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x01

VALID PADDING

0×3D ^ 0×0F= 0x32，这表示数字“2”（明文中第一个数据块的最后一个字节）

恢复明文第7个字节

- 要求第7个字节与第8个字节都为0x02；
- 已知中间值的最后一个字节是0x3D，可以将IV中的第8个字节设为0x3F（这会产生0x02）并暴力枚举IV的第七个字节（从0x00开始，直至0xFF）

Block 1 of 1	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3F
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x02

INVALID PADDING

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3F
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x26	0x02	0x02

VALID PADDING

0×26 ^ 0×17= 0x31，这表示数字“1”（明文中第一个数据块的倒数第二个字节）

- 从后往前破解中间值里的每个字节，最终得到解密后的值

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x31	0x7B	0x2B	0x2A	0x0F	0x62	0x2E	0x35
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08

VALID PADDING

Ciphertext stealing

- Ciphertext stealing技术
  - 取得
    - 密文长度=明文长度
  - ECB ciphertext stealing
    - 明文长度必须大于1个分组
    - 否则，只能使用填充方法
  - CBC ciphertext stealing
    - 明文分组不必要大于1
    - 当明文分组小于1时，从C_0(IV) stealing

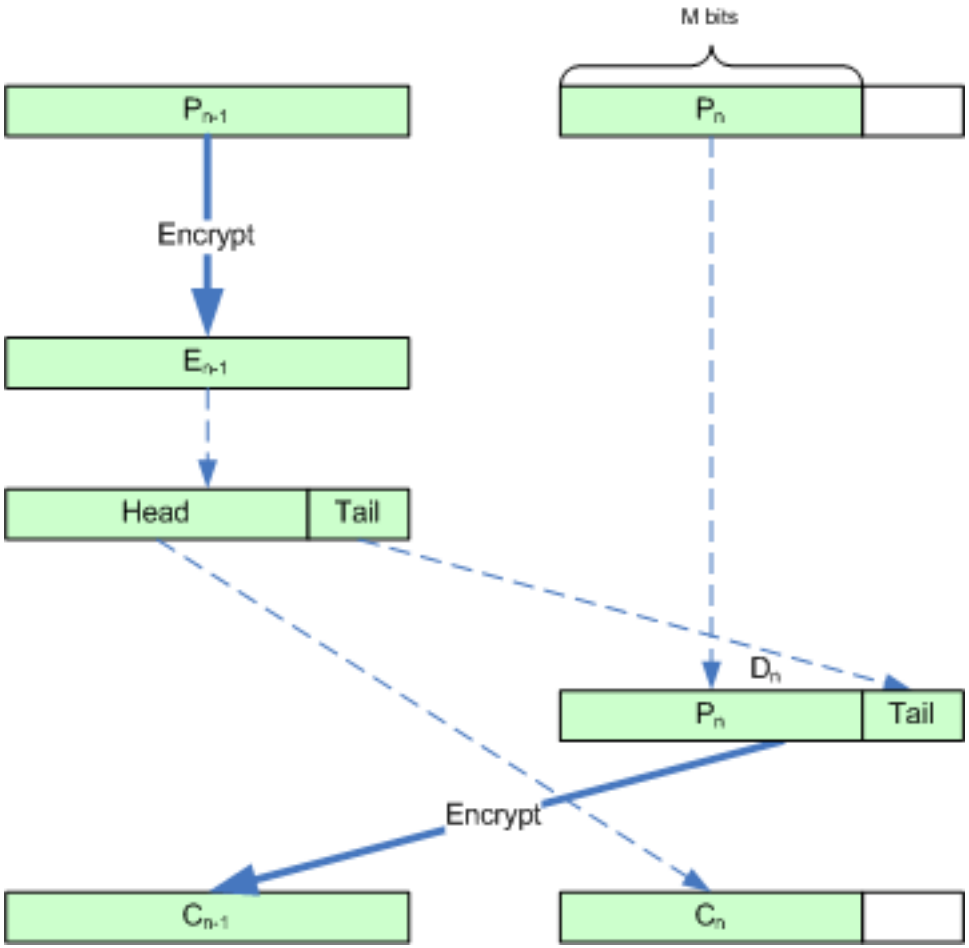
ECB ciphertext stealing

加密

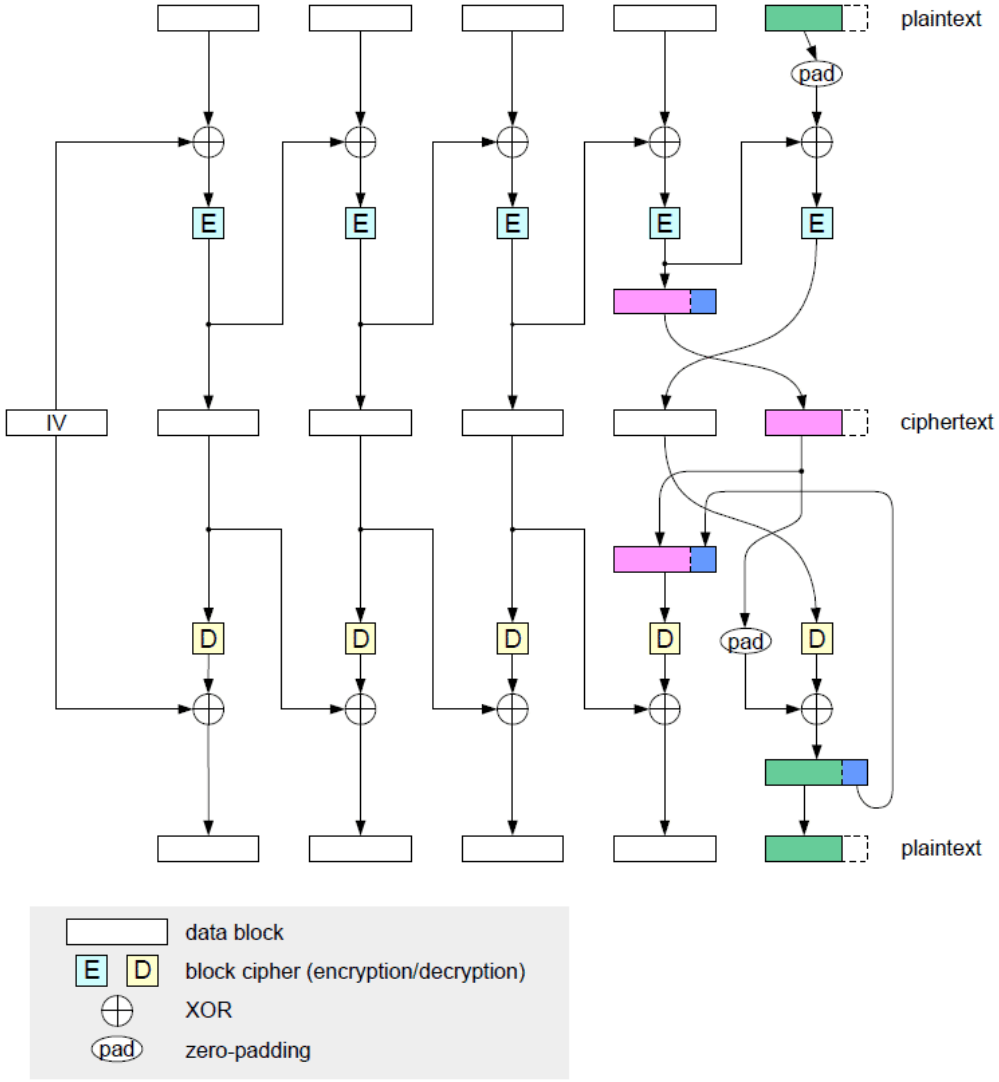
1. $E_{n-1} = Encrypt(k, P_{n-1})$
2. $C_n = head(E_{n-1}, M)$
3. $D_n = P_n \| Tail(E_{n-1}, B-M)$
4. $C_{n-1} = Encrypt(k, D_n)$

解密

1. $D_n = Decrypt(k, C_{n-1})$
2. $\mathrm{E}_{n-1} = C_n \| Tail(D_n, B-M)$
3. $P_n = Head(D_n, M)$
4. $P_{n\text{-}1} = Decrypt(k, E_{n-1})$



CBC ciphertext stealing

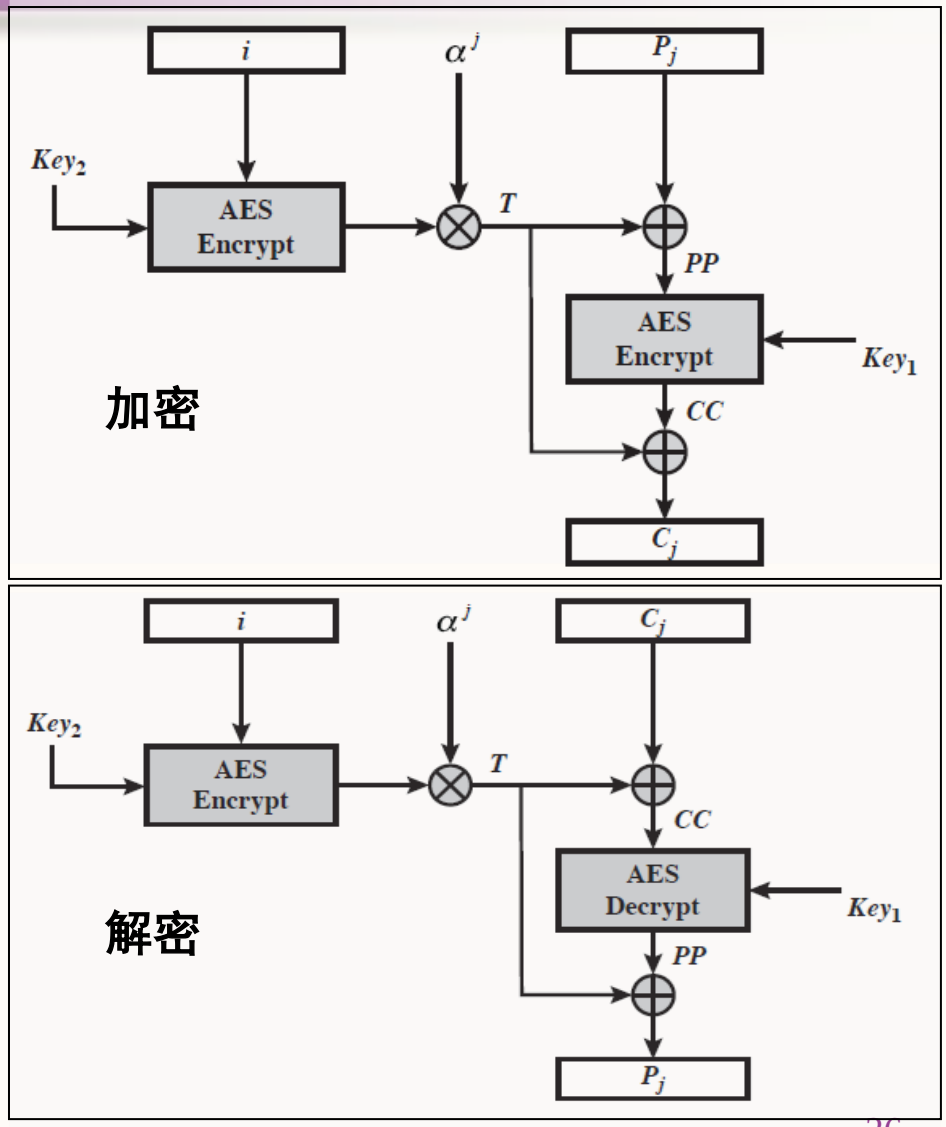


存储加密

- 对基于扇区的磁盘数据进行加密要求
  - 加密后密文长度与明文长度相同
  - 数据访问和加密为独立的、固定长度的数据块(扇区)
  - 明文分组的位置是仅有的可用的元数据（描述数据的数据）
  - 不同位置的相同的数据被加密成不同的密文
- XTS-AES被推荐为存储加密标准
  - IEEE std 1619-2007 ； NIST SP 800-38E, 2010
- 该标准被广泛应用
  - BestCrypt, dm-crypt, FreeOTFE, TrueCrypt, DiskCryptor, FreeBSD’s geli, OpenBSD softraid disk encryption software, Mac OS X Lion’s FileVault, in hardware-based media encryption devices by the SPYRUS Hydra PC Digital Attache and The Kingston Data Traveler 5000

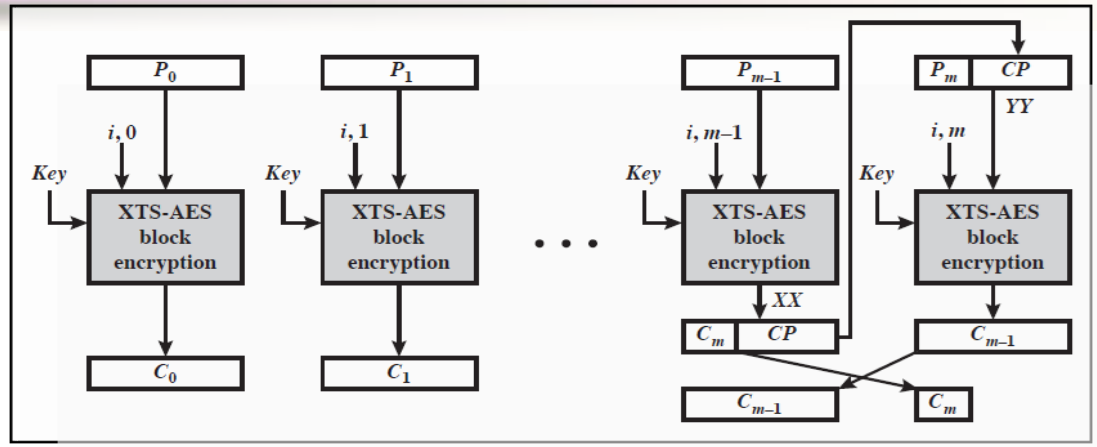
XTS-AES磁盘加密模式

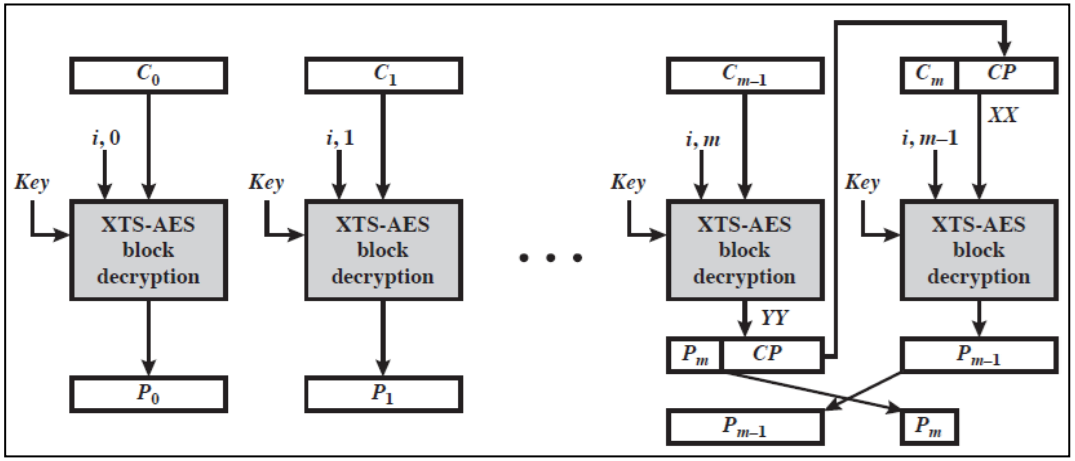
- 单一明文分组的加密
- 参数
  - key=(key_1||key_2)，256比特或512比特
  - P_j：明文的第j个分组
  - j：在一个数据单元中明文分组的序号
  - i：128比特的tweak值（数据块的磁盘地址）
  - α：GF(2128)中的一个生成元，对应多项式 x



XTS-AES磁盘加密模式

- 对一个扇区数据的加密
  - 每个明文分组单独加密
  - 若最后一个明文分组小于128比特，则使用ciphertext stealing 技术





XTS-AES磁盘加密

- $T \cdot \alpha$的计算： GF(2^{128})中模多项式$x^{128}+x^7+x^2+x+1$乘法
- 128比特的T表示为T[15],T[14]….,T[0],则 $T \cdot \alpha$ 计算如下

```
cin = 0;
for( j = 0; j <16; j++)
{
Cout = (T[ j] >> 7) &1; //取一个字节的最高比特
T[ j] = ((T[ j] <<1) + cin) & 0xFF; //循环左移1比特
Cin = Cout; //低位字节的最高位
}
if (Cout) //若最高比特是1
T[0] = T[0] ^ 0x87
```

谢谢！