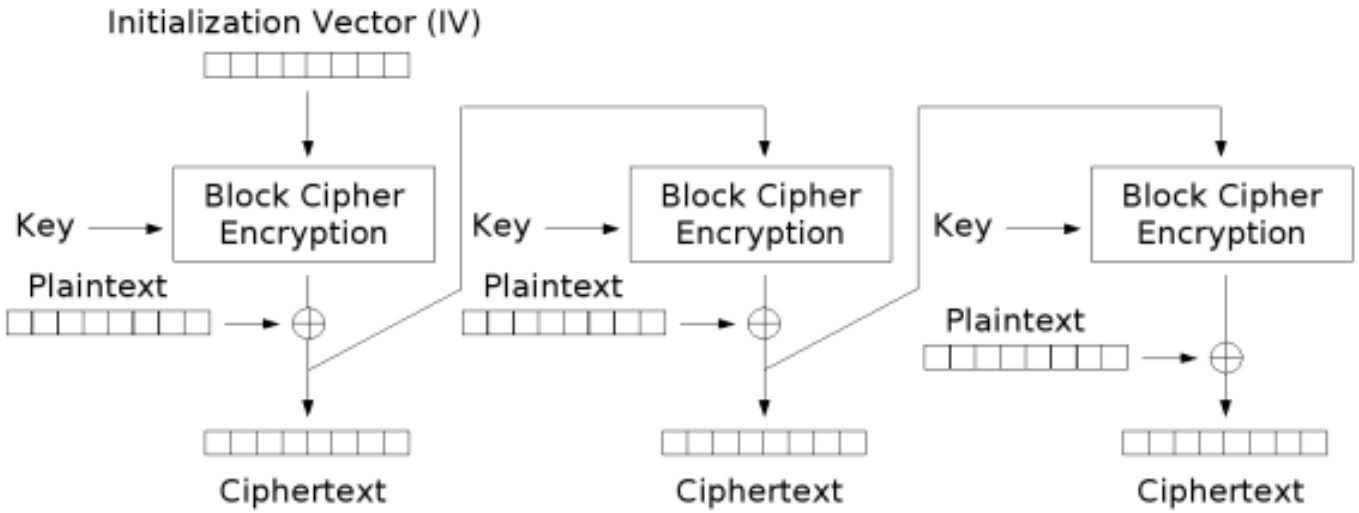
密文反馈模式（CFB）

- 密文反馈（CFB，Cipher feedback）模式类似于CBC，可以将块密码变为自同步的流密码。
- CFB拥有一些CBC所不具备的特性，这些特性与OFB和CTR的流模式相似（后面会讲到）：只需要使用块密码进行加密操作，且消息无需进行填充。
- 应用场景：
  - 流数据加密、认证
  - 数据库加密，无线通信加密等对数据格式有特殊要求的加密环境。

密文反馈模式（CFB）- 加密

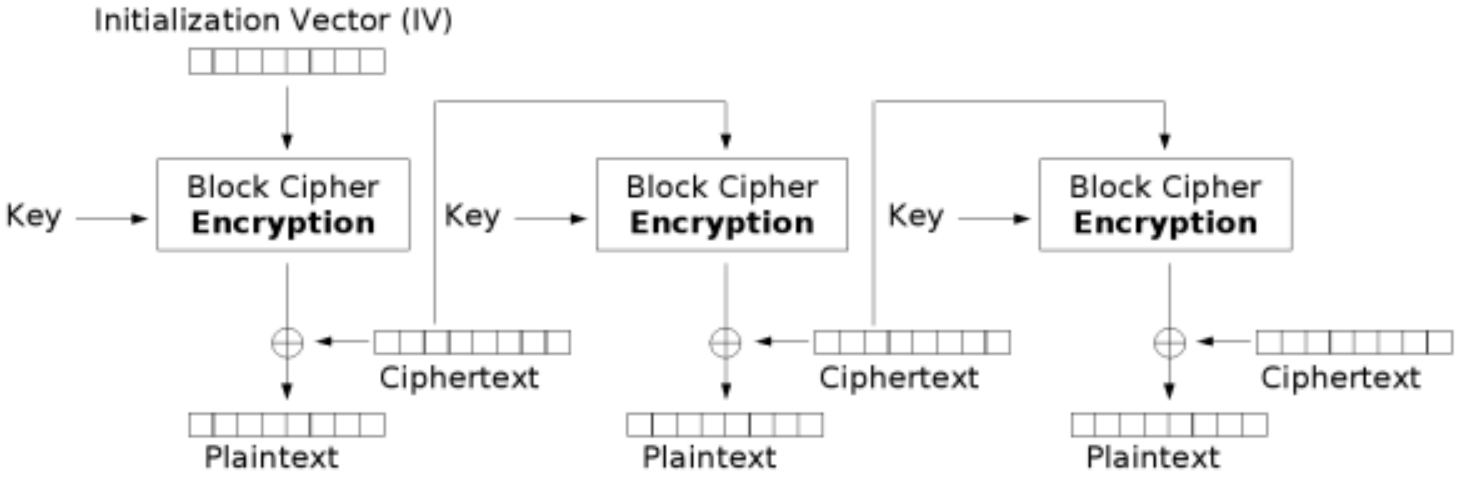
- 加密过程：$C_j = E_k(C_{j-1}) \oplus P_j$



Cipher Feedback (CFB) mode encryption

密文反馈模式（CFB）- 解密

- 解密过程： $P_j = E_k(C_{j-1}) \oplus C_j$



Cipher Feedback (CFB) mode decryption

密文反馈模式（CFB）

- 优点：
  - 适应于不同数据格式的要求，可以实时操作，立即加密并传输或交换某些纯文本或原始文本值，可以按字符或者比特处理。
  - 有限错误传播。在解密时，密文中一位数据的改变仅会影响两个明文块：对应明文块中的一位数据与下一块中全部的数据，而之后的数据将恢复正常。
  - 自同步。与CBC相同，即若密文的一整块发生错误，CBC和CFB都仍能解密大部分数据，而仅有一位数据错误。若需要在仅有了一位或一字节错误的情况下也让模式具有自同步性，必须每次只加密一位或一字节。
- 缺点：
  - 与CBC相似，明文的改变会影响接下来所有的密文，因此加密过程不能并行化；而同样的，与CBC类似，解密过程是可以并行化的。

密文反馈模式（CFB）- 错误恢复示例

- 密文反馈模式的优点是可以从密文传输的错误中恢复。比如要传输密文C_1，C_2，C_3，...，C_K，现假定C_1传输错误，把它记作C_1'，则解密还原得到的P_1有错。若X_1记作 (*, *, *, *, *, *, *, *)，则

$X_2 = (*, *, *, *, *, *, *, C_1')$

$X_3 = (*, *, *, *, *, *, C_1', C_2)$

$X_4 = (*, *, *, *, *, C_1', C_2, C_3)$ ...

$X_9 = (C_1', C_2, C_3, C_4, C_5, C_6, C_7, C_8)$

$X_{10} = (C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9)$

使用密钥$X_1, X_2, X_3, \ldots, X_9$解密$C_1'C_2C_3C_4C_5C_6C_7C_8C_9$还原得到的$P_1, P_2, P_3, \ldots, P_9$全部有错，但是从$P_{10}$开始的解密还原是正确的。

输出反馈模式（OFB）

- 输出反馈模式（Output feedback, OFB）可以将块密码变成同步的流密码。
- 它产生密钥流的块，然后将其与明文块进行异或，得到密文。
- 与其它流密码一样，密文中一个位的翻转会使明文中同样位置的位也产生翻转。这种特性使得许多错误校正码，例如奇偶校验位，即使在加密前计算，而在加密后进行校验也可以得出正确结果。
- 每个使用OFB的输出块与其前面所有的输出块相关，因此不能并行化处理。然而，由于明文和密文只在最终的异或过程中使用，因此可以事先对IV进行加密，最后并行的将明文或密文进行并行的异或处理。
- 可以利用输入全0的CBC模式产生OFB模式的密钥流。这种方法十分实用，因为可以利用快速的CBC硬件实现来加速OFB模式的加密过程

输出反馈模式（OFB）- 加/解密

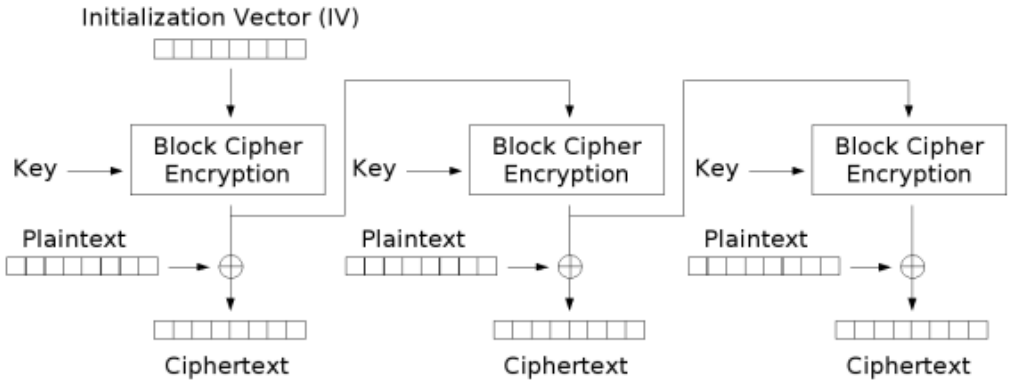
- 由于XOR操作的对称性，加密和解密操作是完全相同的：

$$C_j = P_j \oplus O_j$$

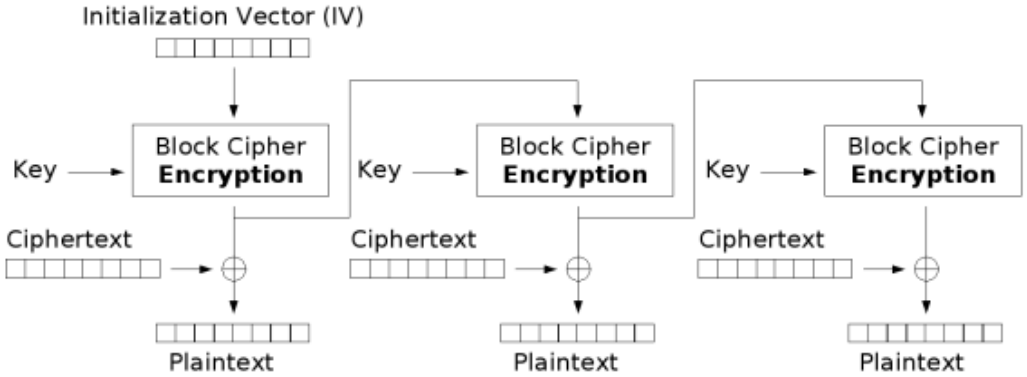
$$P_j = C_j \oplus O_j$$

$$O_j = E_k(P_j \oplus O_{j-1})$$

$$O_0 = IV$$



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

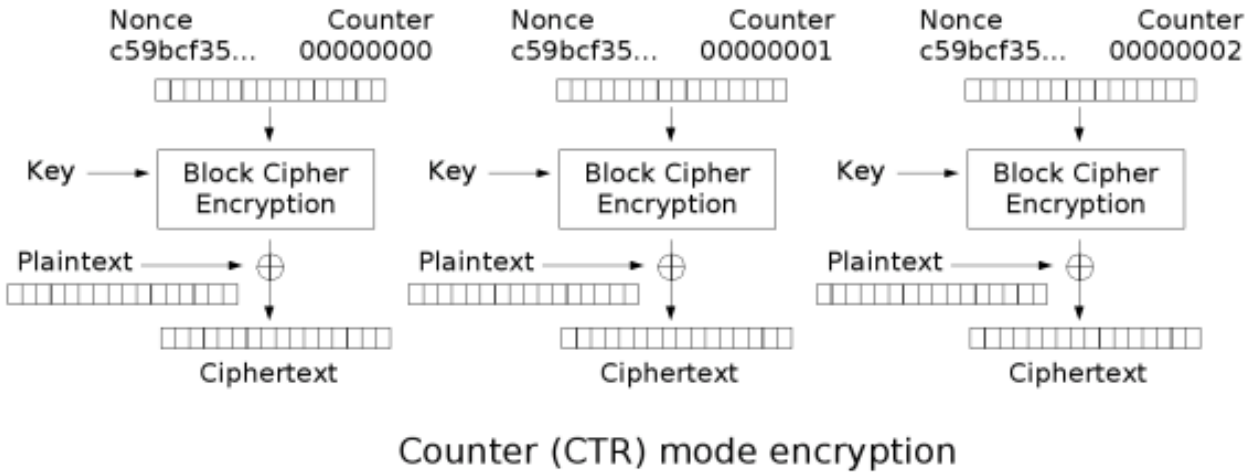
输出反馈模式（OFB）

- 优点：
  - 不具有错误传播特性。
- 缺点：
  - 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。
  - 每个使用OFB的输出块与其前面所有的输出块相关，因此不能并行化处理。
  - 比 CFB 更容易受到消息流修改攻击。
  - 发送方和接收方需要保持同步，否则所有数据都会丢失。
- 适用场景：
  - 适用于一些明文冗余度比较大的场景，如图像加密和语音加密。

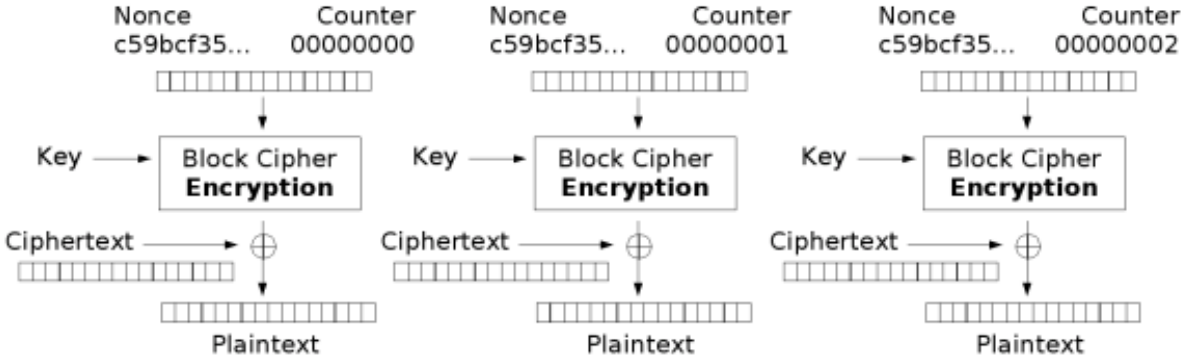
计数器模式（CTR）

- 与OFB相似，CTR将块密码变为流密码。
- 它通过递增一个加密计数器以产生连续的密钥流，其中，计数器可以是任意保证长时间不产生重复输出的函数，但使用一个普通的计数器是最简单和最常见的做法。
- CTR模式的特征类似于OFB，但它允许在解密时进行随机存取。由于加密和解密过程均可以进行并行处理，CTR适合运用于多处理器的硬件上。
- 每个明文块必须有不同的键和计数器值（从不重复使用）。
- 用途：高速网络加密

计数器模式（CTR）- 加/解密



Counter (CTR) mode encryption



Counter (CTR) mode decryption

计数器模式（CTR）

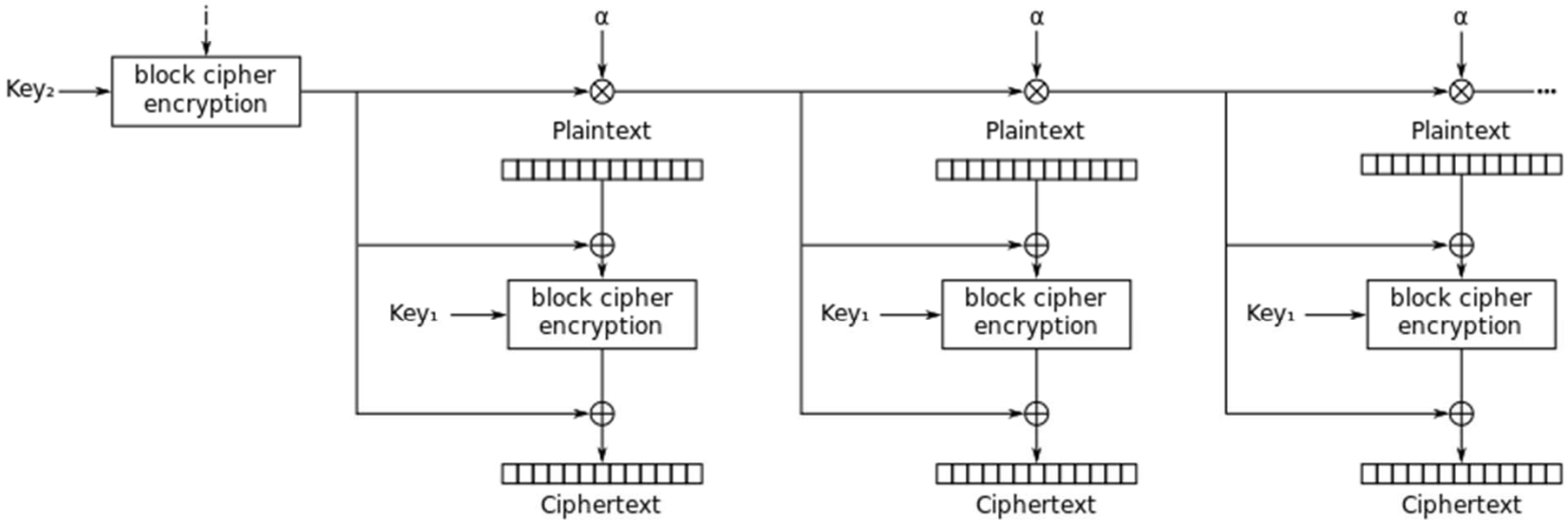
- 优点：
  - 高效
  - 可以在 h/w 或 s/w 中进行并行加密
  - 可以在需要之前进行预处理
  - 适用于高速网络
  - 随机访问加密数据块
  - 可证明的安全性
- 缺点：
  - 必须确保永远不能重复使用密钥/计数器值

Xor-encrypt-xor（XEX）

- 允许在一个数据单元（例如磁盘扇区）内高效处理连续块。

$$X = E_K(I) \otimes \alpha^j,$$
$$C = E_K(P \oplus X) \oplus X,$$

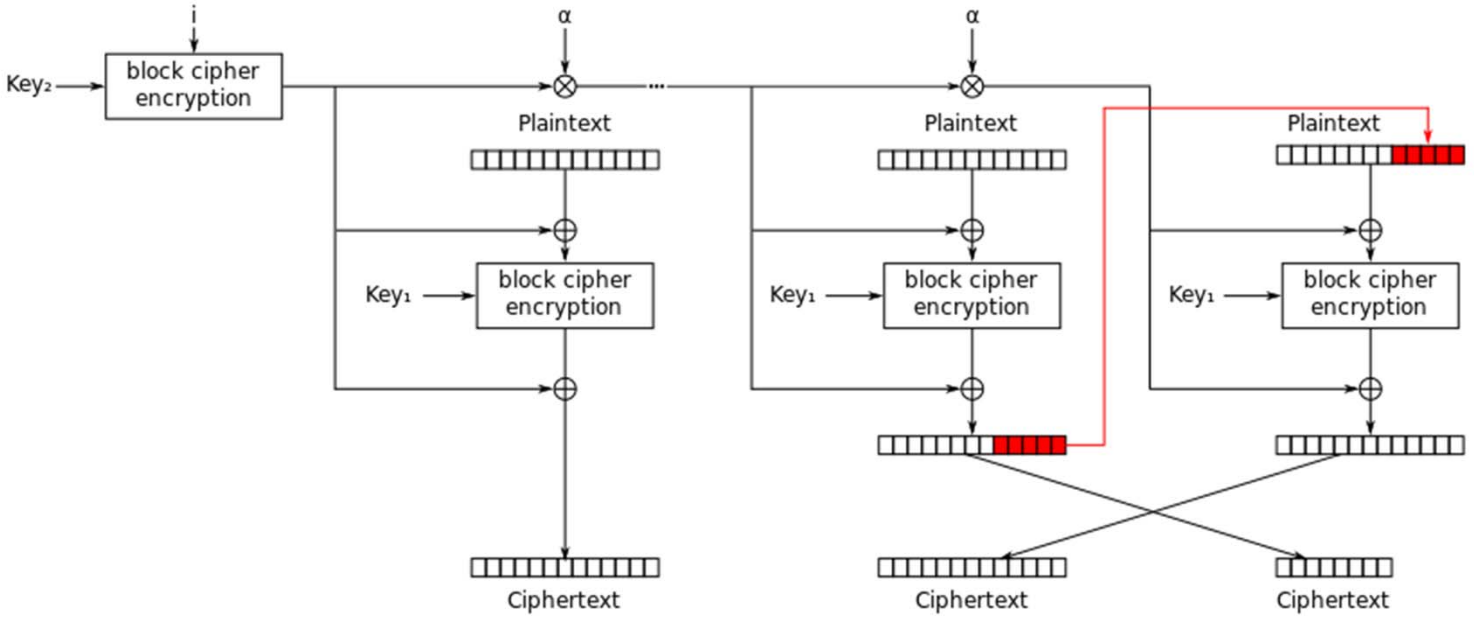
P is the plaintext,
I is the number of the sector,
α is the primitive element of $\mathrm{GF}(2^{128})$ defined by polynomial x; i.e., the number *2*,
j is the number of the block within the sector.



XEX mode encryption

XTS

- XEX-based tweaked-codebook mode with ciphertext stealing，简称XTS，基于XEX进行改进，更流行的磁盘加密操作模式之一。描述了一种对基于扇区的设备上的数据进行加密的方法。



XEX with tweak and ciphertext stealing (XTS) mode encryption

XTS

- 优点：
  - 密文窃取支持大小不能被块大小整除的扇区，例如520字节扇区和16字节块。

- XTS缺点：
  - XTS模式易受数据操纵和篡改的影响，如果存在操纵和篡改问题，应用程序必须采取措施来检测数据的修改。
  - 该模式易受对扇区和16字节块的流量分析、重播和随机攻击的影响。在重写给定扇区时，攻击者可以收集细粒度（16字节）密文，用于分析或重放攻击（16字节粒度）。

初始化向量

- 初始化向量（IV，Initialization Vector）是许多模式中用于将加密随机化的一个位块，由此即使同样的明文被多次加密也会产生不同的密文，避免了较慢的重新产生密钥的过程。
- 初始化向量与密钥相比有不同的安全性需求，因此IV通常无须保密，然而在大多数情况中，不应当在使用同一密钥的情况下两次使用同一个IV。
- 对于CBC和CFB，重用IV会导致泄露明文首个块的某些信息，亦包括两个不同消息中相同的前缀。对于OFB和CTR而言，重用IV会导致完全失去安全性。另外，在CBC模式中，IV在加密时必须是无法预测的（随机或伪随机）；特别的，在许多实现中使用的产生IV的方法，例如SSL2.0使用的，即采用上一个消息的最后一块密文作为下一个消息的IV，是不安全的。如果攻击者在指定下一个明文之前知道IV（或上一个密文块），他们可以检查自己对之前使用相同密钥加密的某个块的明文的猜测（TLS CBC IV攻击）。

填充

- 块密码只能对确定长度的数据块进行处理，而消息的长度通常是可变的。因此部分模式（如ECB和CBC）需要对最后一块在加密前进行填充。
- 填充方法：
- Pad with bytes all of the same value as the number of padding bytes (PKCS5 padding)

```
DES INPUT BLOCK  = f  o  r  _  _  _  _  _
(IN HEX)           66 6F 72 05 05 05 05 05
KEY              = 01 23 45 67 89 AB CD EF
DES OUTPUT BLOCK = FD 29 85 C9 E8 DF 41 40
```

- Pad with 0x80 followed by zero bytes (OneAndZeroes Padding)

```
DES INPUT BLOCK  = f  o  r  _  _  _  _  _
(IN HEX)           66 6F 72 80 00 00 00 00
KEY              = 01 23 45 67 89 AB CD EF
DES OUTPUT BLOCK = BE 62 5D 9F F3 C6 C8 40
```

填充

- Pad with zeroes except make the last byte equal to the number of padding bytes

```
DES INPUT BLOCK  = f  o  r  _  _  _  _  _
(IN HEX)           66 6f 72 00 00 00 00 05
KEY              = 01 23 45 67 89 AB CD EF
DES OUTPUT BLOCK = 91 19 2C 64 B5 5C 5D B8
```

- Pad with zero (null) characters

```
DES INPUT BLOCK  = f  o  r  _  _  _  _  _
(IN HEX)           66 6f 72 00 00 00 00 00
KEY              = 01 23 45 67 89 AB CD EF
DES OUTPUT BLOCK = 9E 14 FB 96 C5 FE EB 75
```

- Pad with spaces

```
DES INPUT BLOCK  = f  o  r  _  _  _  _  _
(IN HEX)           66 6f 72 20 20 20 20 20
KEY              = 01 23 45 67 89 AB CD EF
DES OUTPUT BLOCK = E3 FF EC E5 21 1F 35 25
```

填充

- CFB，OFB和CTR模式不需要对长度不为密码块大小整数倍的消息进行特别的处理。
- 因为这些模式是通过对块密码的输出与明文进行异或工作的。
- 最后一个明文块（可能是不完整的）与密钥流块的前几个字节异或后，产生了与该明文块大小相同的密文块。

流密码

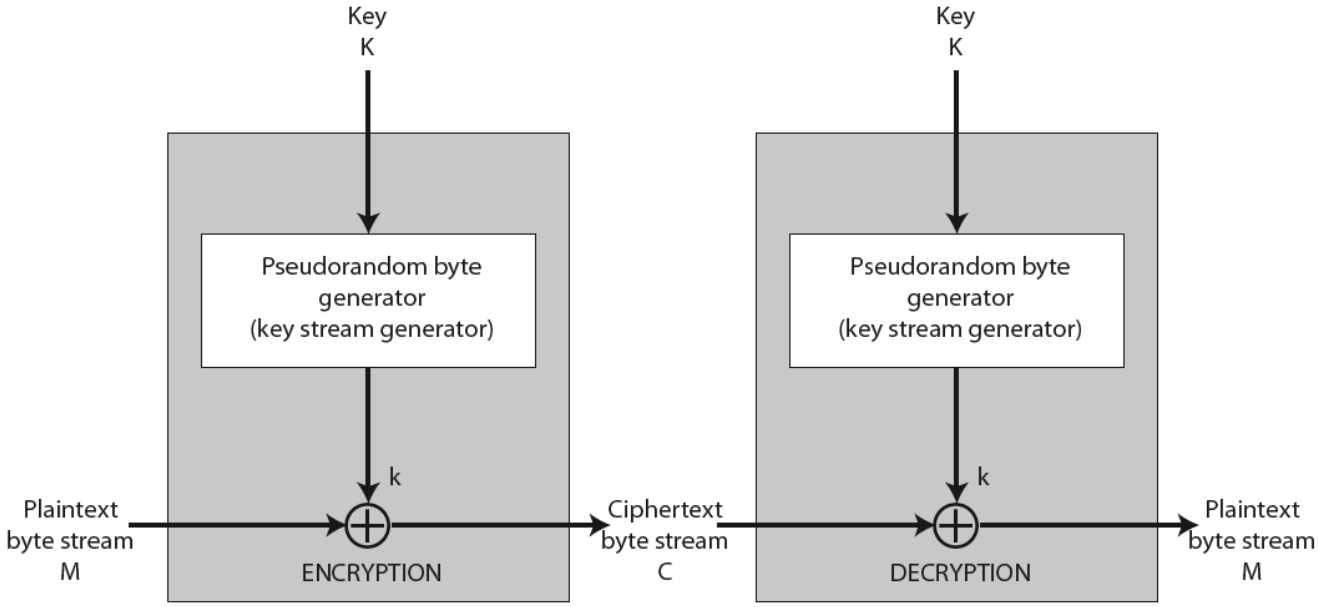
- 流密码（Stream cipher），是一种对称加密算法，加密和解密双方使用相同伪随机加密数据流（pseudo-random stream）作为密钥，明文数据每次与密钥数据流顺次对应加密，得到密文数据流。
- 实践中数据通常是一个位（bit）并用异或（xor）操作加密。
- 该算法解决了对称加密完善保密性（perfect secrecy）的实际操作困难。“完善保密性”由克劳德·香农于1949年提出。由于完善保密性要求密钥长度不短于明文长度，故而实际操作存在困难，改由较短数据流通过特定算法得到密钥流。

流密码

- 流加密目前来说都是对称加密。
- 一般来说，流密码的密钥长度会与明文的长度相同。
- 流密码的密钥派生自一个较短的密钥，派生算法通常为一个伪随机数生成算法。伪随机数生成算法生成的序列的随机性越强，明文中的统计特征被覆盖的更好。
- 流密码的关键在于设计好的伪随机数生成器。一般来说，伪随机数生成器的基本构造模块为反馈移位寄存器。也有一些特殊设计的流密码，比如 RC4

流密码结构

- 一个密钥被输入到一个伪随机比特生成器，该生成器生成一个随机的比特密钥流，并与消息数据进行异或来对其进行加密。由接收器再次进行异或来解密。



流密码攻击

- 多次使用同一密码本
- 一种严重的错误即反复使用同一密码本对不同明文进行加密。攻击者可利用这种方式对密文进行解密。
- 用p表示明文，C表示密文，k表示种子，PRG表示密钥流生成算法，则：

  $C_1 = p_1 \text{ xor } PRG(k)$

  $C_2 = p_2 \text{ xor } PRG(k)$
- 攻击者监听到此段消息（包含两段相同密钥流加密的密文）后，即可利用：

  $C_1 \text{ xor } C_2$得到$p_1 \text{ xor } p_2$
- 足量的冗余（此处表示p_1，p_2）则可破解明文。

RC4

- RC4（Rivest Cipher 4）是一种面向字节的串流加密法，密钥长度可变。它加解密使用相同的密钥，因此也属于对称加密算法。
- RC4由伪随机数生成器和异或运算组成。RC4的密钥长度可变，范围是[1,255]。RC4一个字节一个字节地加解密。给定一个密钥，伪随机数生成器接受密钥并产生一个S盒。S盒用来加密数据，而且在加密过程中S盒会变化。
- 由于异或运算的对合性，RC4加密解密使用同一套算法。
- RC4是有线等效加密（WEP）中采用的加密算法，也曾经是TLS可采用的算法之一。 2015年2月发布的RFC 7465禁止在TLS中使用RC4。

RC4-流程

- RC4 主要包含三个流程：

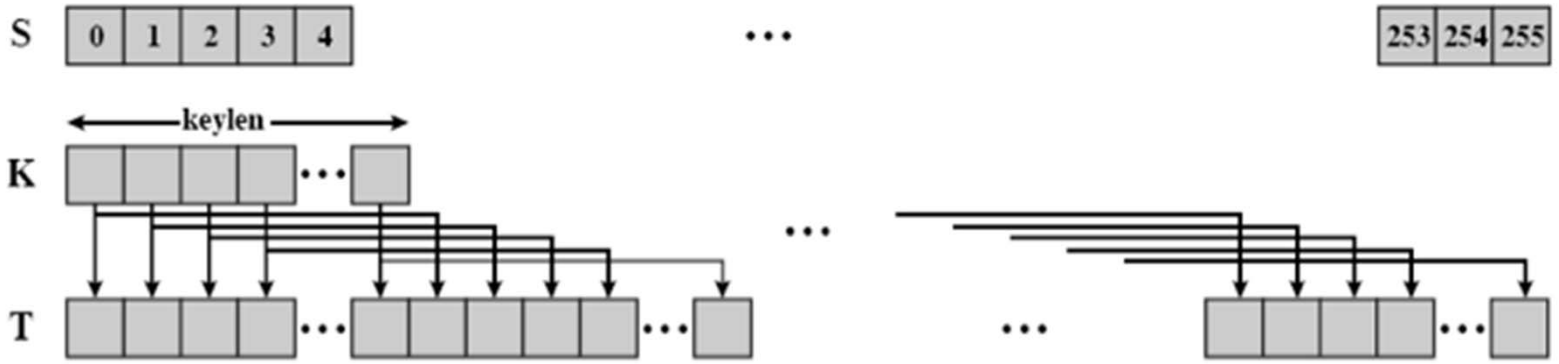
- 1. 初始化 S 和 T 数组。
- 2. 初始化置换 S。
- 3. 生成密钥流并加密。

- 一般称前两部分为 KSA（Key-scheduling algorithm），最后一部分是 PRGA（Pseudo-random generation algorithm）。

RC4-流程

➢ 1. 初始化 S 和 T 数组。

for i = 0 to 255 do
  S[i] = i
  T[i] = K[i mod keylen])

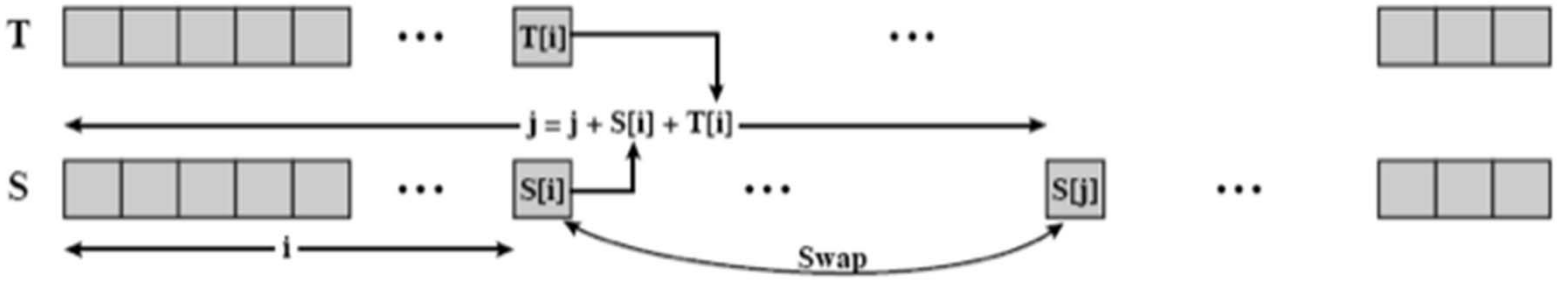


(a) Initial state of S and T

RC4-流程

➢ 2. 初始化置换 S。

```
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])
```



(b) Initial permutation of S

RC4-流程

➢ 3. 生成密钥流并加密。

$i = j = 0$

for each message byte M_i

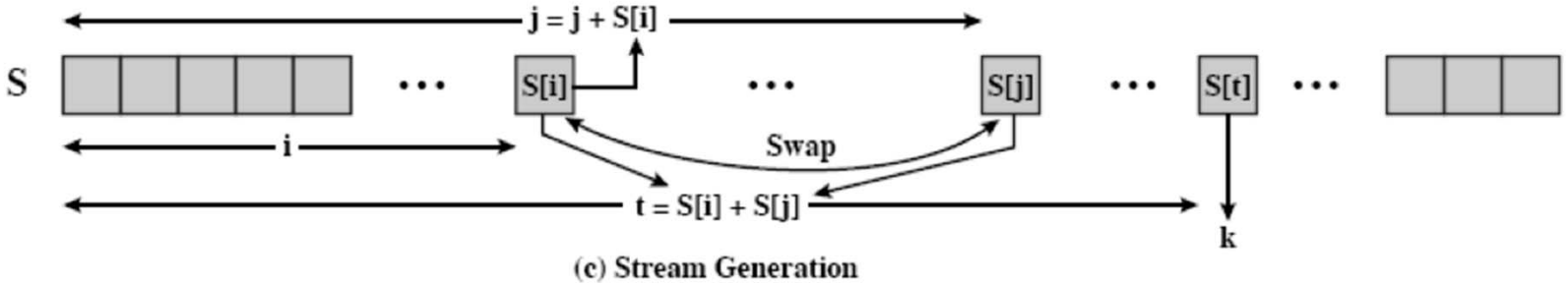
$i = (i + 1) \pmod{256}$

$j = (j + S[i]) \pmod{256}$

swap($S[i]$, $S[j]$)

$t = (S[i] + S[j]) \pmod{256}$

$C_i = M_i$ XOR $S[t]$



(c) Stream Generation

Thank you!