第六章

几种实用的对称密码

6.1 考察如下二种加密算法

1. Triple DES

2. RC5

其他一些较实用的算法，自己看书了解，如Blowfish，CAST，以及RC2。

1 TRIPLE DES

- DES算法设计的优点是很多的。DES在世界商业界的普遍使用，使得如何加强DES的安全性成为一个十分实际的问题。Quisquater 、Toms Berson等曾建议采用长达768 bits密钥的方案。由于已经证明DES不能成为群，见

  K.W.Campbell and M.J.Wiener

  Proof that DES is not a group

  In Advances in Cryptology——Crpto'92.

  Springer——Verlag, New York,1993.

于是多重DES，尤其是三重DES还在普遍使用。

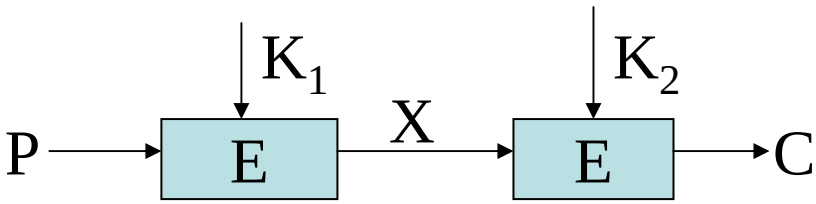
（1）二重DES(Double DES)

- 给定明文P和两个加密密钥k_1和k_2，采用二重DES对P进行加密，有

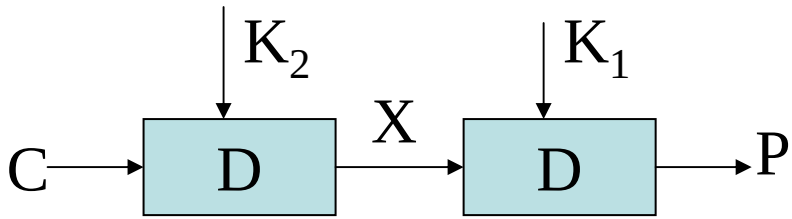
  密文 $C=E_{K2}(E_{K1}(P))$

对C进行解密D，有

明文 $P=D_{K1}(D_{K2}(C))$



加密图



解密图

对于二重DES的加密，所用密钥的长度为

$56\times2=112$ bits

这样是否真正能增强DES的强度呢？问题在于下式能否成立：

$$E_{K2}(E_{K1}(P)) = E_{K3}(P) \qquad (4.1)$$

DES是一个从集合A到集合A的一个映射。其中：

$A = \{(a_1, a_2, a_3, \ldots\ldots, a_{64}) | a_i \in Z_2 = \{0,1\}\}$, $|A| = 2^{64}$

映射DES事实上可视为对A的一个作用，作用方式为置换。所有可能的置换数为 $(2^{64})! = 10^{34738000000}$。

然而，DES对每一个不同的密钥只决定唯一的映射。而密钥数$2^{56} < 10^{17}$，(4.1)式一般是不能成立的。已知DES不能构成群！

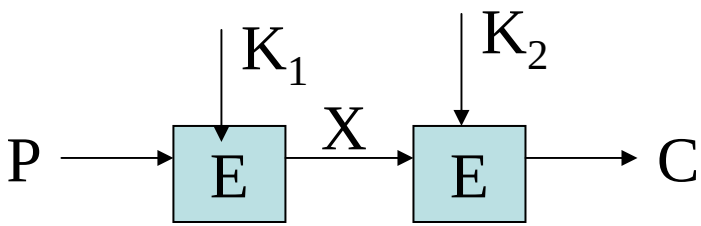
- 关于**DES**不是群的详细证明可以参见我们开始列出的文献。

- 注：二重**DES**很难抵挡住中间相遇攻击法（**Meet-in-the-Middle Attack)**

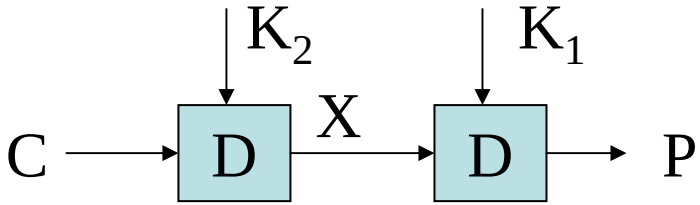
由 $C=E_{K2}(E_{K1}(P))$

从图中可见

$X=E_{K1}(P)=D_{K2}(C)$



加密



解密

若给出一个已知的明密文对（P,C)，我们对2^{56}个所有密钥K_1，分别利用DES对明文P加密，得到一张密钥对应于密文X的一张表；类似地对2^{56}个所有可能的密钥K_2，分别利用DES对密文C解密，得到相应的“明文”X。做成一张X与K_2的对应表。比较两个表就会得到真正使用的密钥对K_1, K_2。

对二重DES的中间相遇攻击的分析

- 已知，给定一个明文**P**，经二重**DES**加密有2^{64}个可能的密文。而二重**DES**所用密钥的长度应是**112 bits**，所以选择密钥有2^{112}个可能性。于是对给定明文**P**加密成密文 **C**,有$2^{112}/2^{64}=2^{48}$种可能 的密钥被选用。于是，对确定的明密文对（**P**、**C**），密文不符的大约有2^{48} 个 ，这个数字也对应于中间不相符的密文；然而中间密文的样本空间有2^{64}个样本，于是中间不相遇的概率为 $2^{48-64}=2^{-16}$。这样，对已知明文-密文对的中间相遇攻击成功的概率为$1-2^{-16}$。
- 攻击用的代价不大于$2^{56} + 2^{56}$ ，也就是数量级为2^{56} 这和单次 **DES**攻击代价2^{55} 基本差别不大。

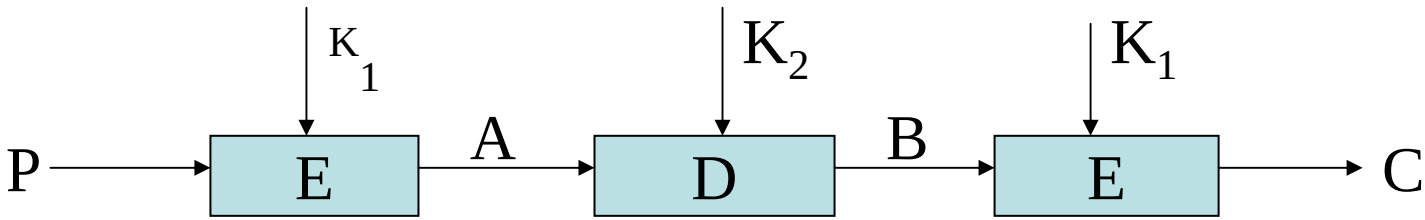
（2）带有双密钥的三重DES（Triple DES with Two Keys)

- Tuchman给出双密钥的EDE模式（加密-解密-加密）：

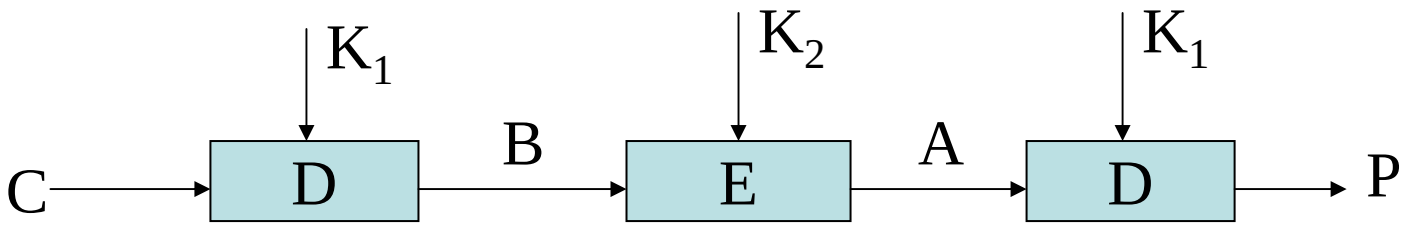
  $C=E_{K1}(D_{K2}(E_{K1}(P)))$ ……对P加密

  $P=D_{K1}(E_{K2}(D_{K1}(C)))$ ……对C解密

  这种替代DES的加密较为流行并且已被采纳用于密钥管理标准（The Key Manager Standards ANSX9.17和ISO8732).



加密图



解密图

对双密钥的三重DES的分析

- 该模式由**IBM**设计, 可与常规加密算法兼容
- 这种替代**DES**的加密较为流行并且已被采纳用于密钥管理标准（**The Key Manager Standards ANSX9.17**和**ISO8732).**
- 交替使用K_1和K_2可以抵抗中间相遇攻击.如果$C=E_{K2}(E_{K1}(E_{K1}(P)))$,只需要2^{56+2}次加密
- 到目前为止，还没有人给出攻击三重**DES**的有效方法。对其密钥空间中密钥进行蛮干搜索，那么由于空间太大为$2^{112}=5\times10^{33}$，这实际上是不可行的。若用差分攻击的方法，相对于单一**DES**来说复杂性以指数形式增长，要超过10^{52}。

- 目前还没有针对两个密钥三重DES的实用攻击方法。但对两个密钥三重DES的攻击有一些设想，以这些设想为基础将来可能设计出更成功的攻击技术。
- **Merkle R. and Hellman,M. “On the security of multiple encryption”. Communication of the ACM, July 1981**
- **Oorschot ,P and Wiener, M. “A Known-plaintext attack on two-key triple encryption” Proceedings, EUROCrypt’90,1990: published by Springer-Verlag**

虽然对上述带双密钥的三重**DES**到目前为止还没有好的实际攻击办法，但人们还是放心不下，又建议使用三密钥的三重**DES**，此时密钥总长为**168bits.**

$$C=E_{K3}(D_{K2}(E_{K1}(P)))$$

- 与**DES**的兼容性可以通过令$K_3=K_2$或$K_1=K_2$得到
- 许多基于**Internet**的应用里用到：**PGP**和**S/MIME**

2 RC5

Ron Rivest 1994设计、1995公开

RC5具有如下的特性：

1. 适用于软件或者硬件实现，运算速度快;
3. 能适应于不同字长的程序（一个字的bit数是RC5的一个参数；不同字长派生出相异的算法）;
4. 加密的轮数可变（轮数是RC5的第二个参数，这个参数用来调整加密速度和安全性的程度）;
5. 密钥长度是可变的（密钥长度是RC5的第三个参数）;
6. RC5形式简单，易于实现，加密强度可调节;
7. 对记忆度要求不高（使RC5可用于类似Smart Card这类的对记忆度有限定的器件）;
8. 高保密性（适当选择好参数）;

对RC5的系统描述：

（1）RC5的参数

RC5实际上是由三个参数决定的一族加密算法。

参数	定义	允许值
w	字的**bit**数大小。**RC5**加密的基本单位为**2**个字块	**16,32,64**
r	轮数	**0,1, …,255**
b	密钥字节的长度（**8-bit bytes)**	**0,1, …,255**

- RC5加密明文块的长度为32，64，128 bits。并且对应同样长度的密文。密钥长度为从0到2040 bits。一个特定的RC5表示为

RC5-w/r/b

Rivest建议使用的标准RC5为

RC5-32/12/16

（明文分组长度64，加密轮数12，密钥长度128 bits）

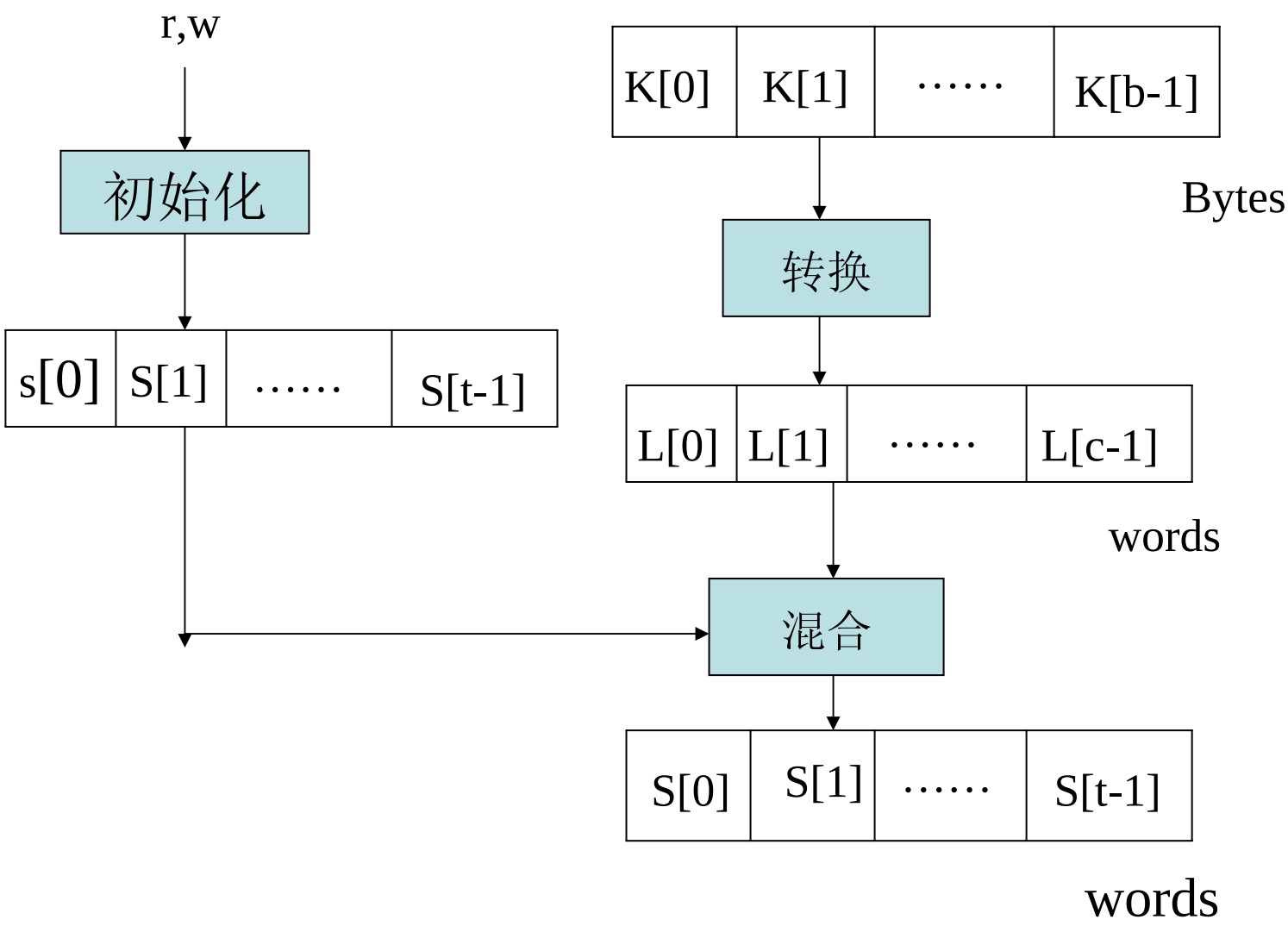
(RC5-w/r/b)

（2） RC5的密钥扩展

对给定的密钥K来说，经过一些复合运算可产生总数为t的子密钥，使得每一轮都分配一对密钥。除此之外的非轮运算部分也要分配一对密钥。总计产生 t=2r+2 个子密钥，每个密钥的长度为一个字长（w bits)。子密钥可标记在t-字阵列中:

s[0],s[1], ...,s[t-1] 它为w × t矩阵

这种阵列的产生图示为:


r,w
初始化
s[0]
S[1]
……
S[t-1]
K[0]
K[1]
……
K[b-1]
Bytes
转换
L[0]
L[1]
……
L[c-1]
words
混合
S[0]
S[1]
……
S[t-1]
words

- 将参数r，w输入，左面标出的t-字阵列是一些伪随机bit，按r,w的规格选入的。然后把b-bytes长的密钥K[0, ...,b-1]转换成c-字阵列L[0, ...,c-1]（字的bit数为w，这里c=b×8/w；注意：密钥长度为b个字节）。如果b不是w的整数倍，那么L右端的空位用0填入。

下面描述密钥生成的细节：

对于给定的参数r和w，开始初始化运算

P_w=Odd((e-2)2^w)

Q_w=Odd((Φ -1)2^w)

这里

e =2.718281828459…(自然对数的底）

Φ =1.618033988749…（黄金分割比率）

并且Odd[x]表示最接近x且可左可右的奇整数。

例： Odd[e]=3, Odd[Φ]=1

用上述两个常数，按下述方式得到初始化的阵列S:

S[0]=P_w

For i=1 to t-1 do

S[i]=S[i-1]+Q_w

其中的加法是模2^w的加法运算。

得到初始化阵列S，然后与最后产生的密钥阵列L做混合，最终得到子密钥阵列。

注1*.为了增强复杂性，可对阵列S,L做多次处理：

```
i=j=x=y=0
do 3×max(t,c) times:
{
    S[i]=(S[i]+X+Y)<<<3;
    X=S[i]；i=(i+1)(mod t);
    L[j]=(L[j]+X+Y)<<<(X+Y);
    Y=L[j]；j=(j+1)(mod c);
}
```

2*. Rivest 声称，这个扩张函数具有单向性。

（3） RC5的加密

整个加密使用了下述3个基本运算和它们的逆运算：

- 模2^w加法运算，表示为“+”；
- 逐比特异或运算，表示为“⊕”；
- 字的循环左移运算：字x循环左移y比特，表示为

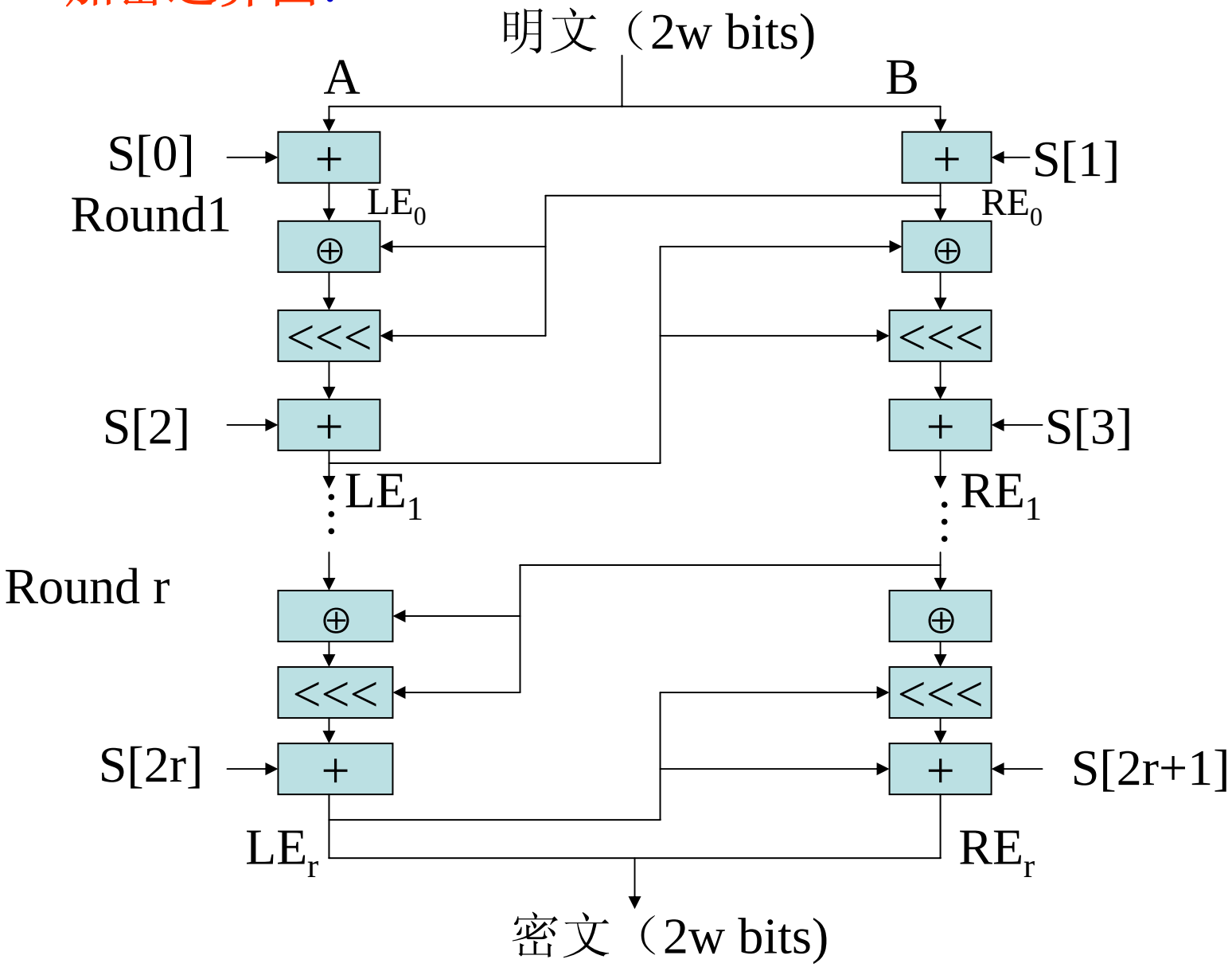
$x<<<y$

它的逆为循环右移y比特，表示为

$x>>>y$

如$(a_0,a_1,a_2,\ldots,a_{n-1})<<<3=(a_3,a_4,\ldots,a_{n-1},a_0,a_1,a_2)$

加密运算图：


明文（2w bits)
A
B
S[0]
+
S[1]
+
Round1
LE0
RE0
⊕
⊕
<<<
<<<
S[2]
+
S[3]
+
LE1
RE1
Round r
⊕
⊕
<<<
<<<
S[2r]
+
S[2r+1]
+
LEr
REr
密文（2w bits)

- 将明文分组为左右**A,B**；用变量LE_i，RE_i参与
- 运算程序为：

$LE_0=A+S[0]$

$RE_0=B+S[1]$

for i=1 to r do

$LE_i=((LE_{i-1} \oplus RE_{i-1})<<<RE_{i-1})+S[2\times i];$

$RE_i=((RE_{i-1} \oplus LE_i)<<<LE_i)+S[2\times i+1];$

对**RC5**的攻击请看：

http://grampus.jaist.ac.jp:8080/miyaji-lab/index.html

（4） RC5的解密

对两个1-字变量LD_r和RD_r。用变量LD_i和RD_i从r到1做：

for i=r down to 1 do

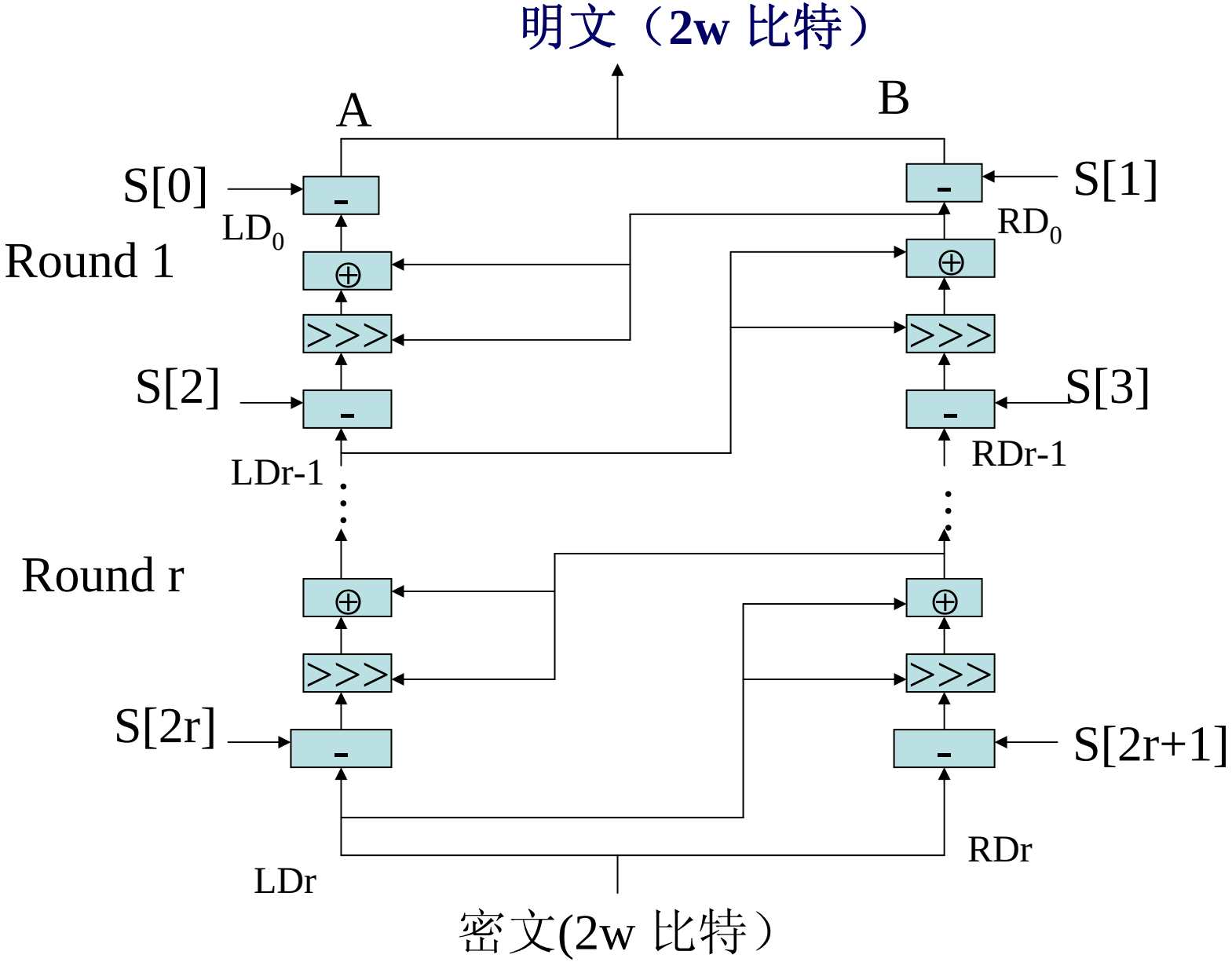
$RD_{i-1}=((RD_i-S[2*i+1])>>>LD_i) \oplus LD_i);$

$LD_{i-1}=((LD_i-S[2*i])>>>RD_{i-1}) \oplus RD_{i-1});$

$B=RD_0-S[1];$

$A=LD_0-S[0].$

(5) RC5操作模式


明文（2w 比特）
A
B
S[0]
S[1]
LD0
RD0
Round 1
S[2]
S[3]
LDr-1
RDr-1
Round r
S[2r]
S[2r+1]
LDr
RDr
密文(2w 比特）

3 RC6分组密码简介

- 企图入选为21世纪加密标准算法AES（没运气！！）。RC6是RC5的进一步改进。像RC5那样，RC6实际上是利用数据的循环移位。
- RC5自1995年公布以来，尽管至今为止还没有发现实际攻击的有效手段，然而一些理论攻击的文章先后也分析出RC5的一些弱点。
- RC6的加密程序：RC6-w/r/b

  Input: 明文存入四个w-bit寄存器A,B,C,D

  轮数r

  w-bit轮密钥S[0,1, …,2r+3]

  Output: 密文存入寄存器A,B,C,D

Procedure:

B=B+S[0]

D=D+S[1]

for i=1 to r do

{

t=(B×(2B+1))<<<$\log_2 w$

u=(D×(2D+1))<<<$\log_2 w$

A=((A⊕t)<<<u)+S[2i]

C=((C ⊕u)<<<t)+S[2i+1]

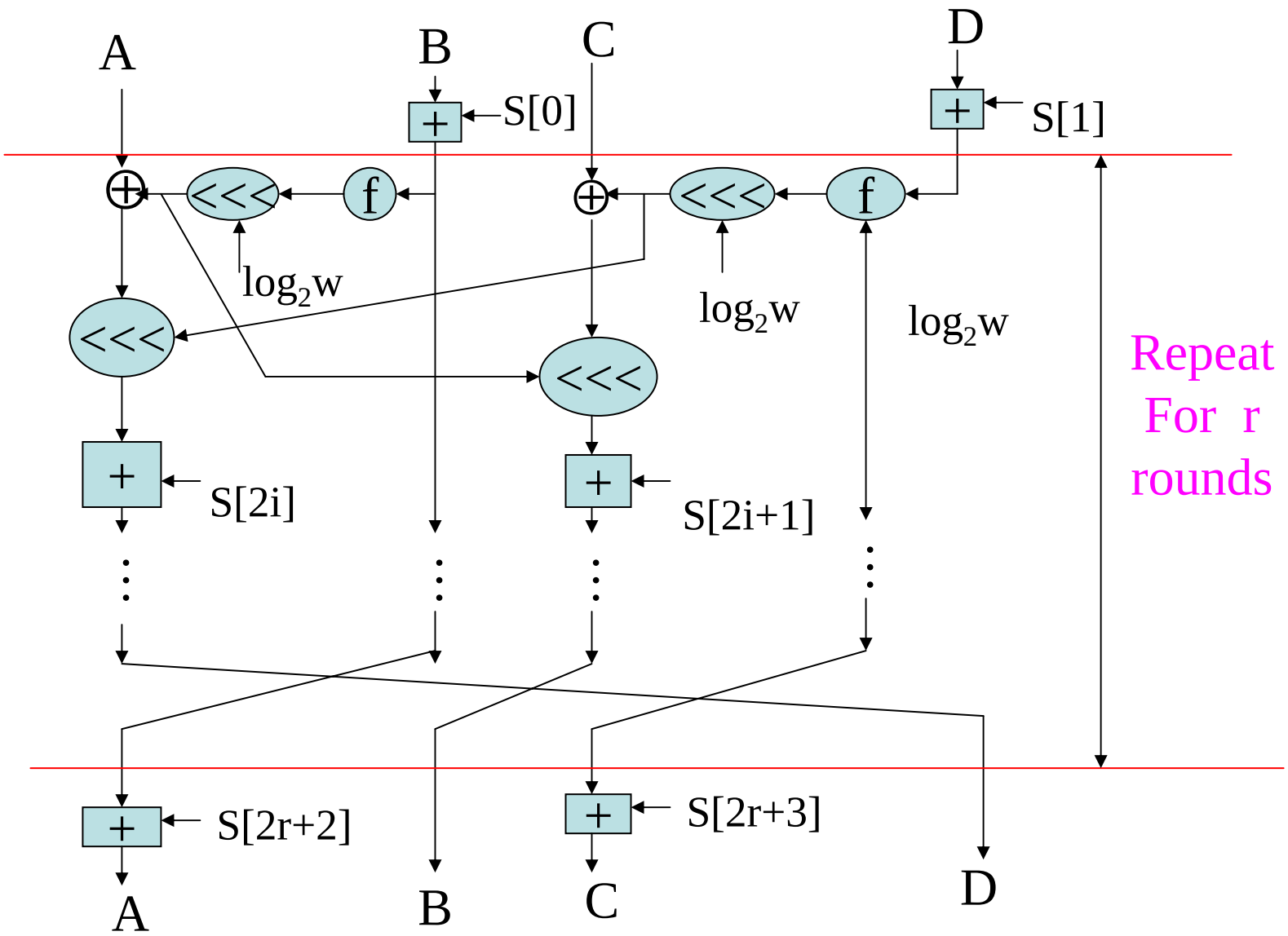
(A,B,C,D)=(B,C,D,A) ……右边寄存器到左边寄存器的并行分配

}

A=A+S[2r+2]

C=C+S[2r+3]

RC6-w/r/b加密图，其中f(x)=x×(2x+1):



Key Expansion (Same as RC5′s)

- **Input:** **array L[0 ... c-1] of input key words**
- **Output:** **array S[0 ... 43] of round key words**
- **Procedure:**

S[0] = 0xB7E15163

for **i = 1** to **43** do **S[i] = S[i-1] + 0x9E3779B9**

A = B = i = j = 0

for **s = 1** to **132** do

{ A = S[i] = (S[i] + A + B) <<< 3

B = L[j] = (L[j] + A + B) <<< (A + B)

i = (i + 1) mod 44

j = (j + 1) mod c }

RC6 Decryption

```
C = C - S[ 43 ]
A = A - S[ 42 ]
for i = 20 downto 1 do
   {
      (A, B, C, D) = (D, A, B, C)
      u = ( D x ( 2D + 1 ) ) <<< 5
      t = ( B x ( 2B + 1 ) ) <<< 5
      C = ( ( C - S[ 2i + 1 ] ) >>> t ) ⊕ u
      A = ( ( A - S[ 2i ] ) >>> u ) ⊕ t
   }
D = D - S[ 1 ]
B = B - S[ 0 ]
```

Security of Key Expansion

- **Key expansion is identical to that of RC5; no known weaknesses.**
- **No known weak keys.**
- **No known related-key attacks.**
- **Round keys appear to be a "random" function of the supplied key.**
- **Bonus: key expansion is quite "one-way"--- difficult to infer supplied key from round keys.**

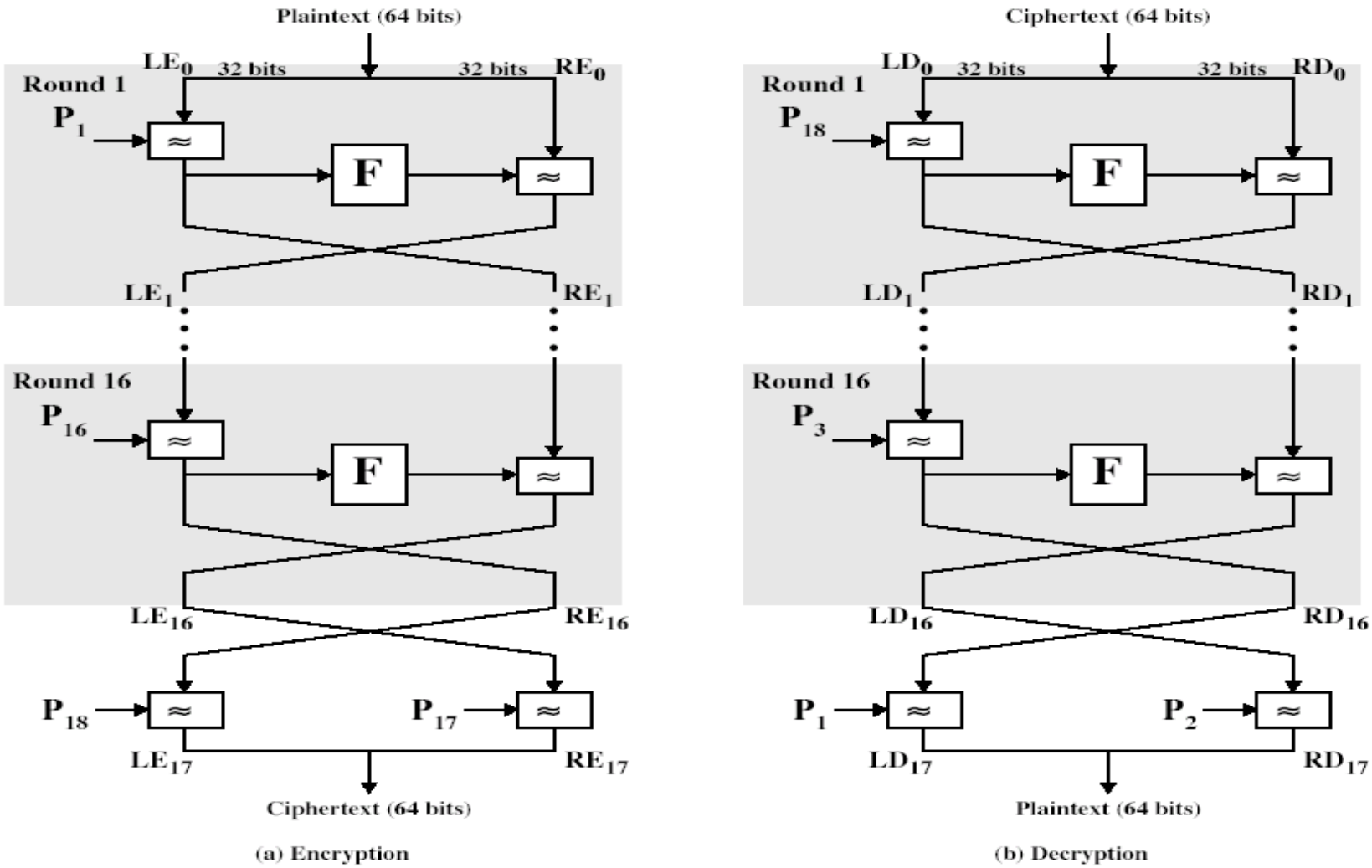
RC6小结

- **RC6 more than meets the requirements for the AES; it is**
  - **simple,**
  - **fast, and**
  - **secure.**

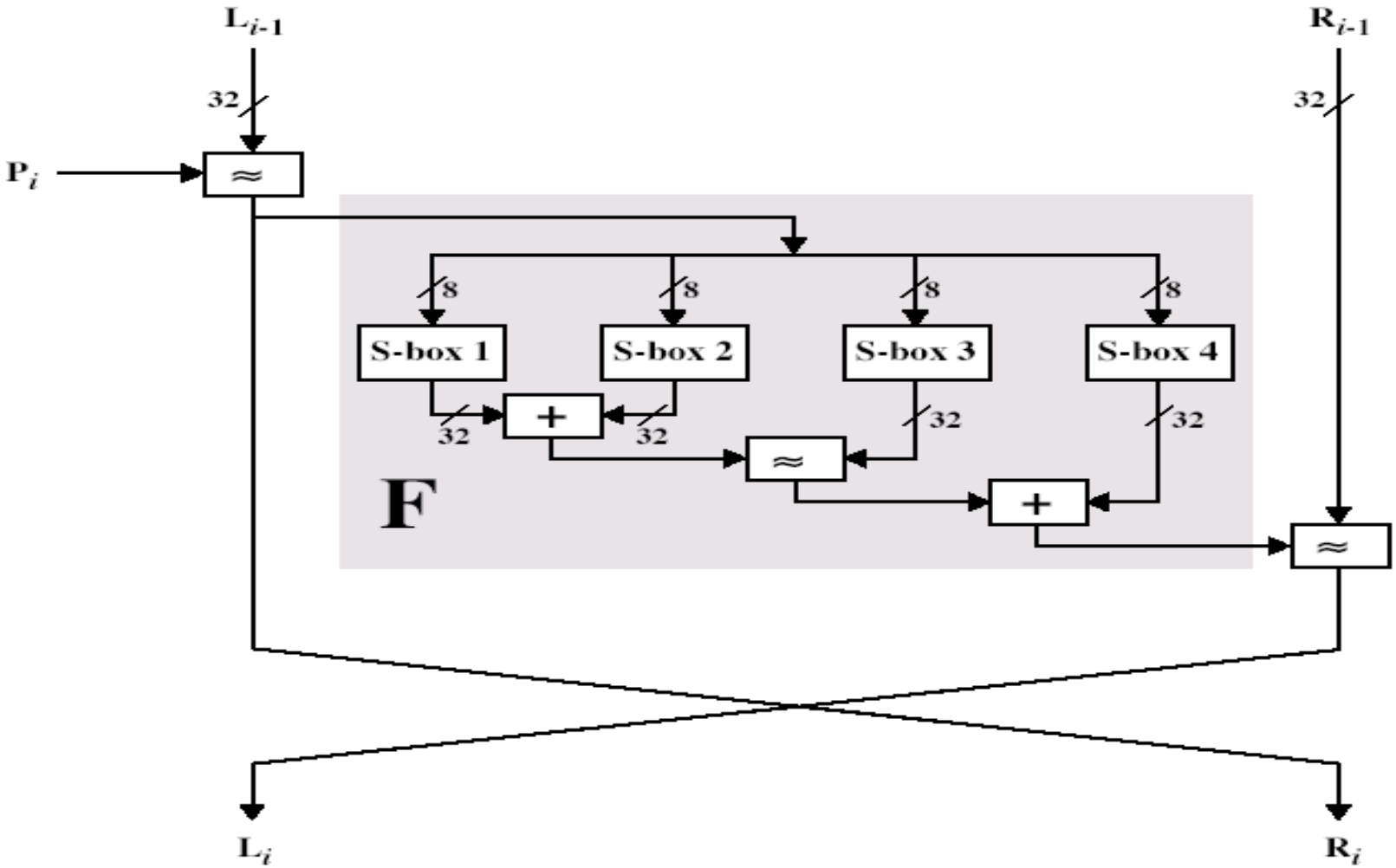
4 Blowfish算法

- 作者为**Bruce Schneier[93]**
- **BLOWFISH**算法特点
  - 采用了**Feistel**结构，**16**轮
  - 快速：**18**时钟周期一个字节
  - 紧凑：消耗不到**5k**内存
  - 简单：结构简单，易于实现和判定算法强度
  - 安全性可变：通过选择不同的密钥长度选择不同的安全级别。从**32**位到**32*14=448**位不等
  - 子密钥产生过程复杂，一次性

Blowfish算法的加密与解密



Blowfish算法的一轮



Blowfish算法讨论

- 使用两个基本运算:模2^{32}加+,按位异或⊕
- **BLOWFISH**算法可能是最难攻破的传统加密算法，因为**S-BOX**密钥相关
- 算法本身的特点
  - 由于子密钥和**S-BOX**产生需要执行**521**个**BLOWFISH**加密算法，所以不适合于密钥频繁变化的应用场合
  - 子密钥和**S-BOX**产生可以保存起来
- 与**Feistel**分组密钥算法不同，每一步的两个部分都参与运算，不是简单的传递
- 密钥变长带来灵活性
- 速度快，在同类算法中相比较是最快的

5 国际数据加密标准IDEA（International Data Encryption Algorithm)算法

- **1990**年瑞士联邦技术学院的来学嘉和**Massey**提出，**PES**，**91**年修订，**92**公布细节
- 设计目标从两个方面考虑
  - 加密强度
  - 易实现性
- 强化了抗差分分析的能力， **PGP**

IDEA算法特点

- 64位分组,128位密钥
- 运算: XOR ⊕,模2^{16}(65536)加 + ,模 (2^{16}+1)（65537）⊙乘
- 三种运算均不满足分配律与结合律
- 有大量弱密钥
- 难以直接扩展到128位块

IDEA设计思想

- 得到confusion的途径
  - 按位异或
  - 以2^{16}(65536)为模的加法
  - 以$2^{16}+1$ (65537)为模的乘法
  - 互不满足分配律、结合律
- 得到diffusion的途径
  - 乘加(MA)结构
- 实现上的考虑
  - 软件和硬件实现上的考虑

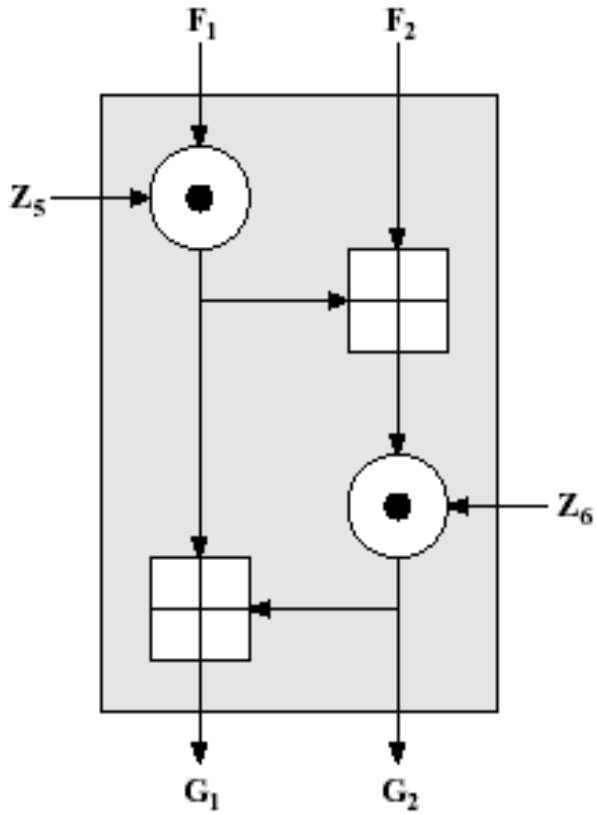


Figure 4.3 Multiplication/addition (MA) Structure

IDEA加密算法

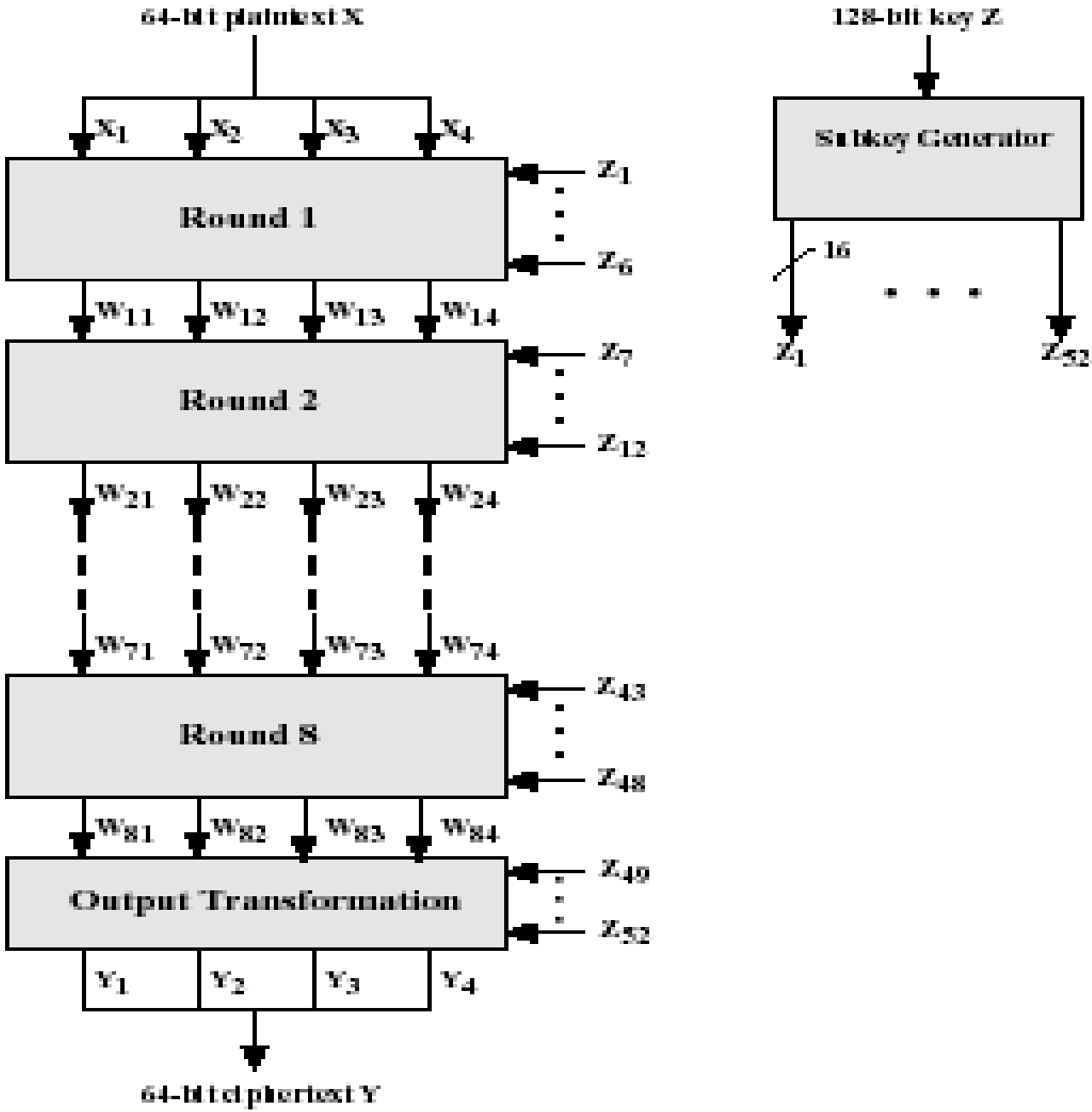


Figure 4.4 Overall IDEA Structure

IDEA
每一轮

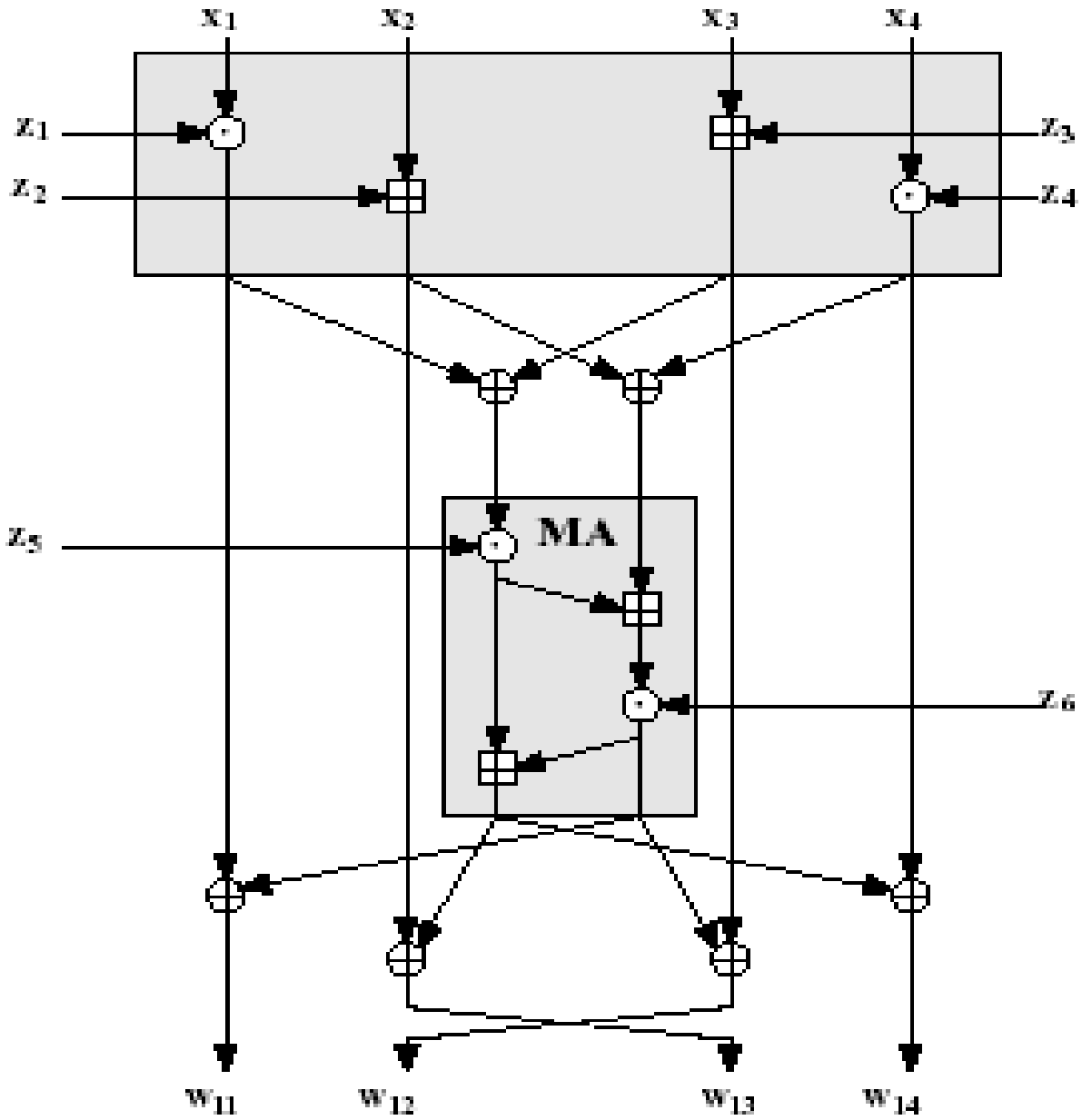
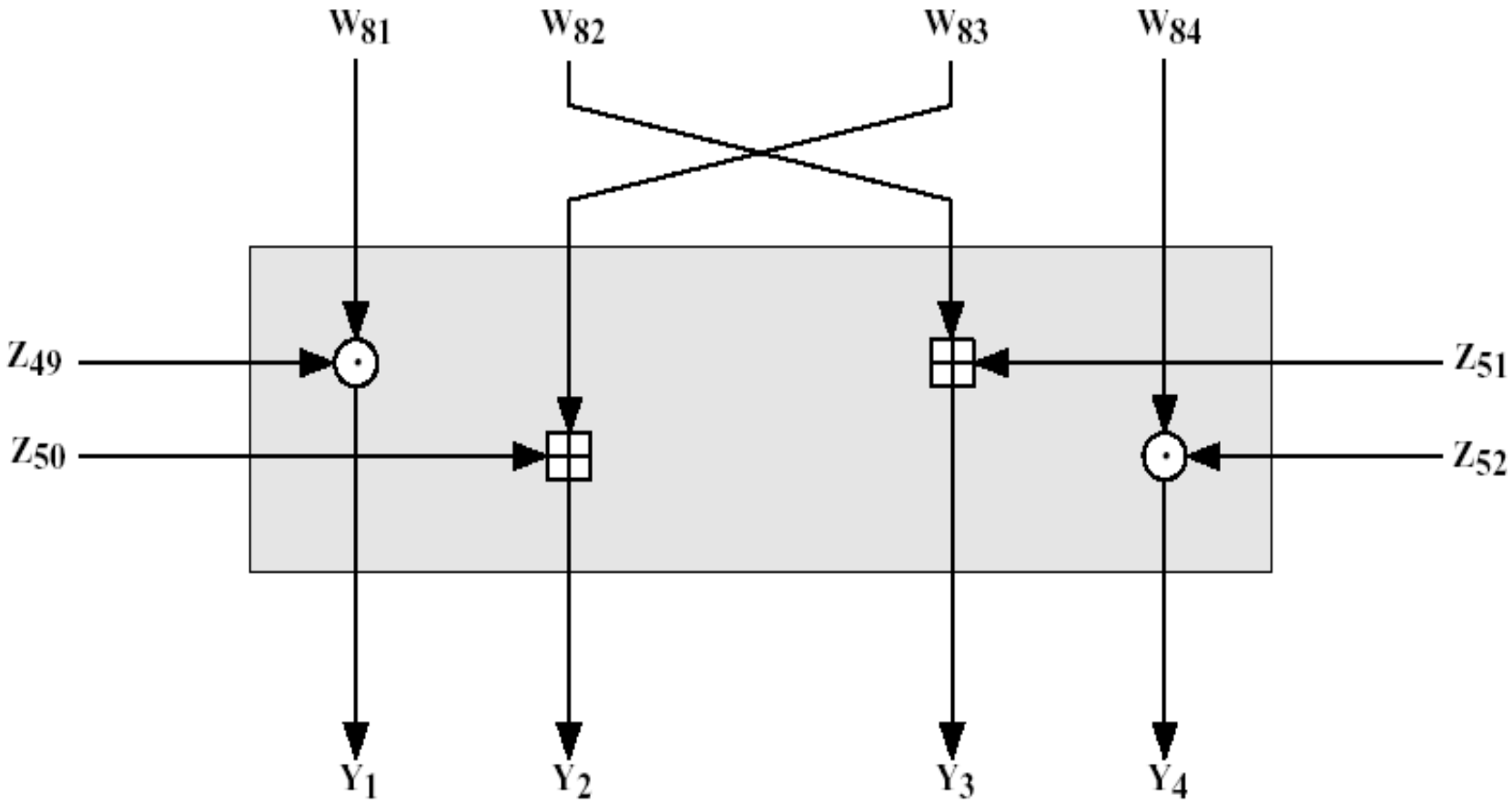
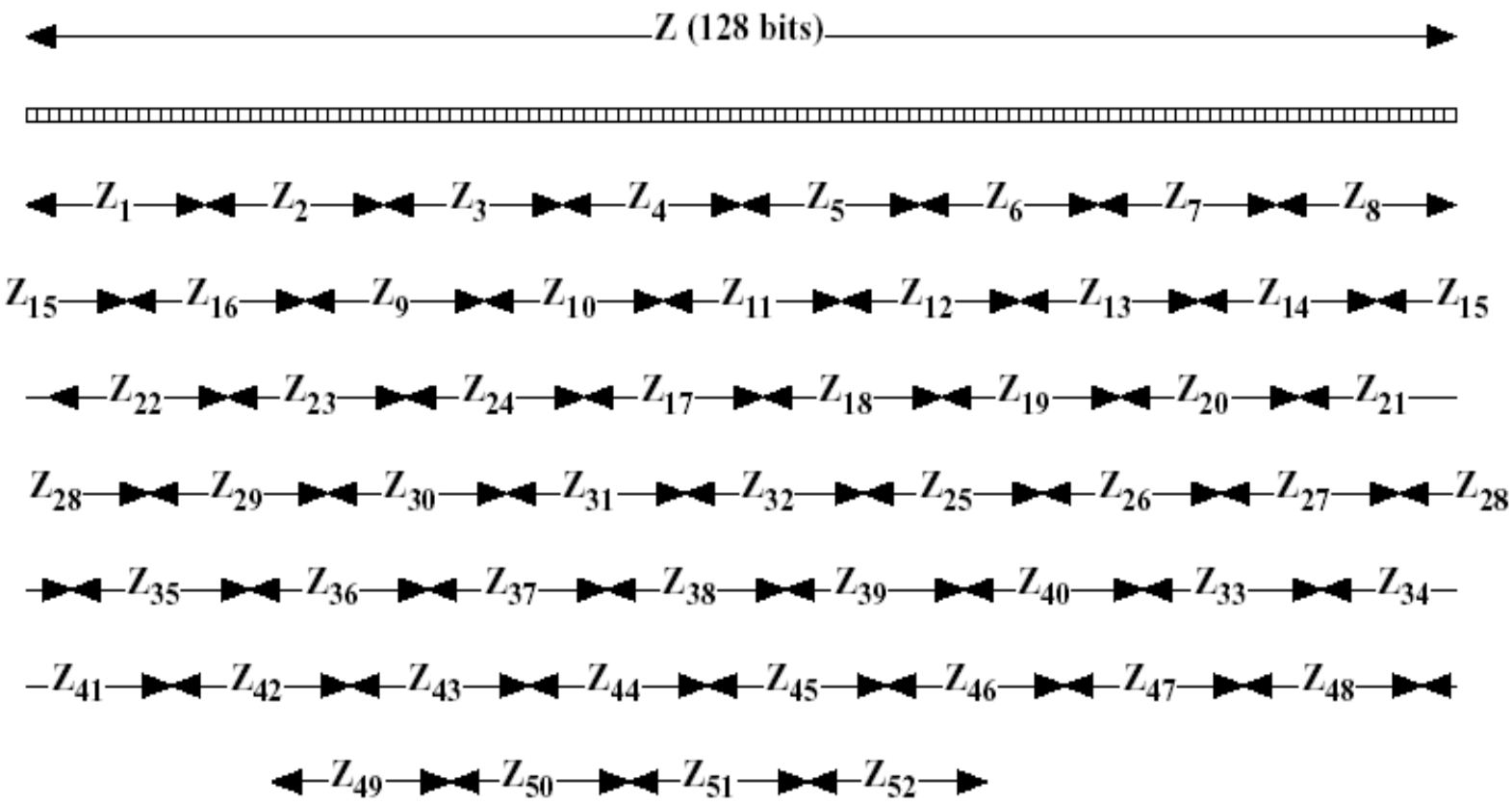


Figure 4.5 Single Round of IDEA (first round)

IDEA输出变换阶段



IDEA的子密钥


Z (128 bits)
Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7 Z_8
Z_{15} Z_{16} Z_9 Z_{10} Z_{11} Z_{12} Z_{13} Z_{14} Z_{15}
Z_{22} Z_{23} Z_{24} Z_{17} Z_{18} Z_{19} Z_{20} Z_{21}
Z_{28} Z_{29} Z_{30} Z_{31} Z_{32} Z_{25} Z_{26} Z_{27} Z_{28}
Z_{35} Z_{36} Z_{37} Z_{38} Z_{39} Z_{40} Z_{33} Z_{34}
Z_{41} Z_{42} Z_{43} Z_{44} Z_{45} Z_{46} Z_{47} Z_{48}
Z_{49} Z_{50} Z_{51} Z_{52}

6.2 分组密码的工作模式

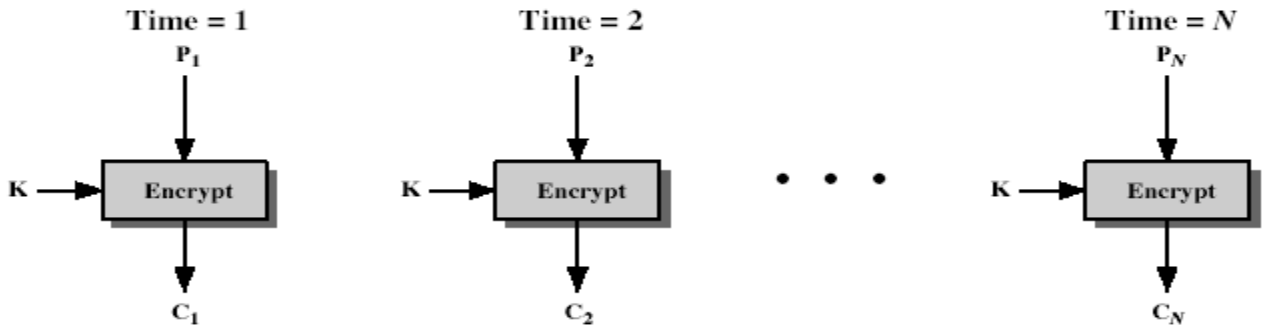
为了将分组密码更好地应用于实际,NIST正式公布了五个工作模式(ANSI X3.106-1983 Modes of Use **(now FIPS 81) defines 4 possible modes;subsequently 5 defined for AES & DES)**

电码本模式(ECB) Electronic Codebook Book

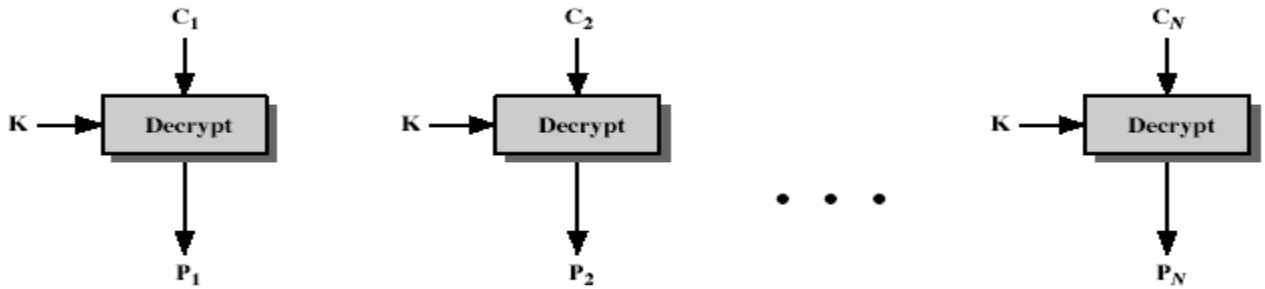
- 将信息分划为等长的相互独立的子块,每一块视为一个分组,对每一组分别实施加密;
- 任何t-比特位的明文组都有唯一的一个密文组与其对应,可以想象为有一个密码本

$$C_i = DES_{K1}(P_i)$$

见下面的图：



(a) Encryption



(b) Decryption

ECB模式的特征

1　一段消息中若有几个相同的明文组,则相对应的密文组也有几个相同的,可使破译者利用结构特性破译密文.所以该模式对很大的消息文本不适用 .

2 对于短消息,采取该模式是简单易行的.

密文分组链接(CBC)模式

Cipher Block Chaining (CBC)

为了克服ECB的弱点,给出如下解决方案:

收发方共享一初始向量IV(双方要保密IV):

应用初始向量 (IV) 来开始我们的加密过程

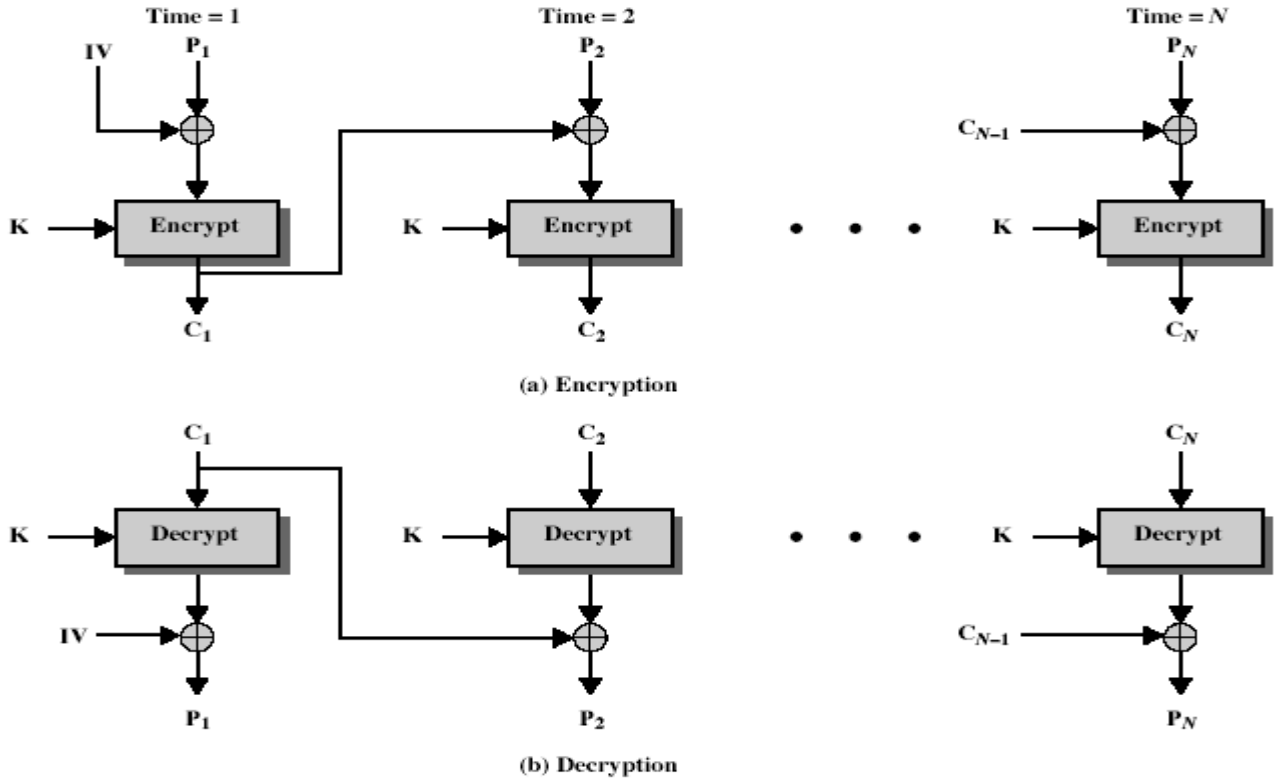
$$C_i = E_K(P_i \oplus C_{i-1})$$

$$C_{-1} = IV$$

解密: $D_K(C_i) = D_K E_K(P_i \oplus C_{i-1}) = P_i \oplus C_{i-1}$

明文P_i= 密文$C_{i-1} \oplus D_K(C_i)$

密文分组链接模式


Time = 1
Time = 2
Time = N
IV
P_1
P_2
P_N
C_{N-1}
K
Encrypt
C_1
C_2
C_N
(a) Encryption
Decrypt
(b) Decryption

Message Padding

- at end of message must handle a possible last short block
  - which is not as large as blocksize of cipher
  - pad either with known non-data value (eg nulls)
  - or pad last block along with count of pad size
    - eg. [b1 b2 b3 0 0 0 0 5]
    - means have 3 data bytes, then 5 bytes pad+count
  - this may require an extra entire block over those in message
- there are other, more esoteric modes, which avoid the need for an extra block

密文反馈模式(CFB)

Cipher FeedBack

对分组密码算法来说,可以利用CFB模式,将其转化成流密码形式.DES为例来说明.

通过反馈方式可以按每个节拍输出1,8, 64 or 128) 比特;分别表示为CFB-1, CFB-8, CFB-64, CFB-128 .

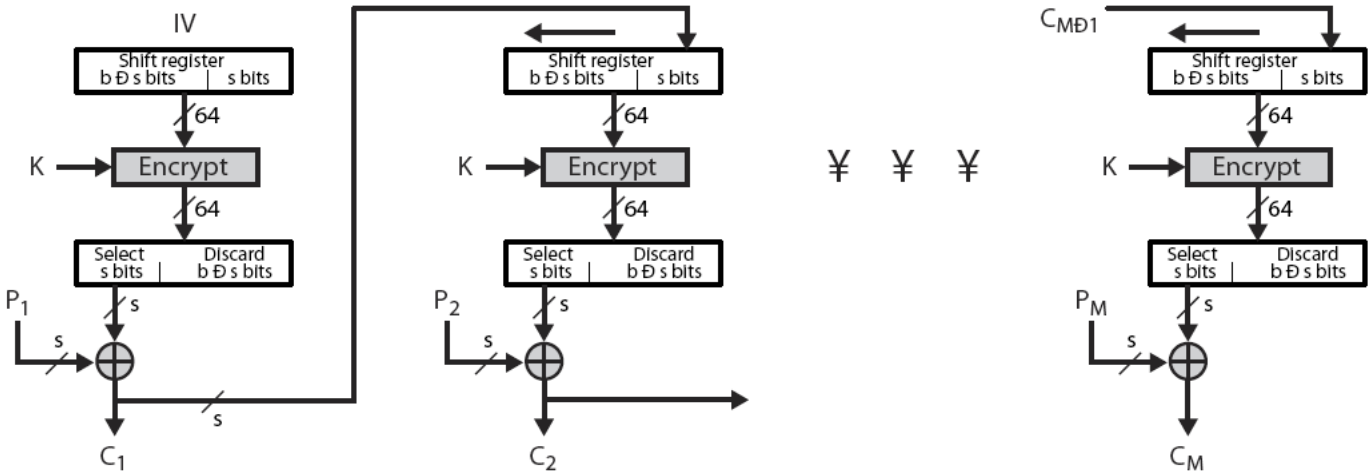
c_i为分组的片段 ,有

$$C_i = P_i \text{ XOR } DES_{K1}(C_{i-1})$$

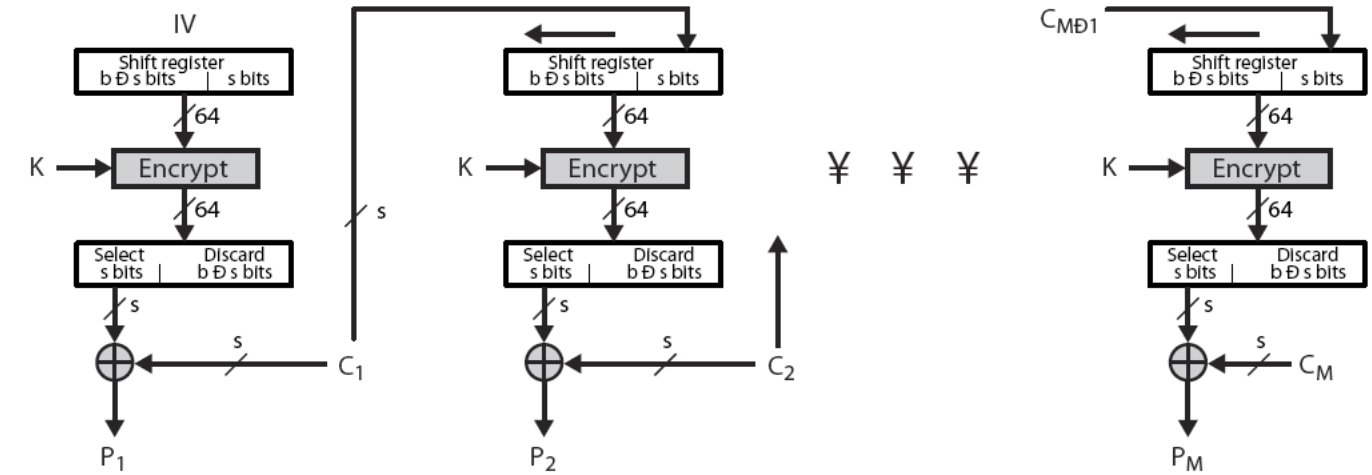
$$C_{-1} = IV$$

见下图 :

Cipher FeedBack (CFB)



(a) Encryption



(b) Decryption

输出反馈模式(OFB)

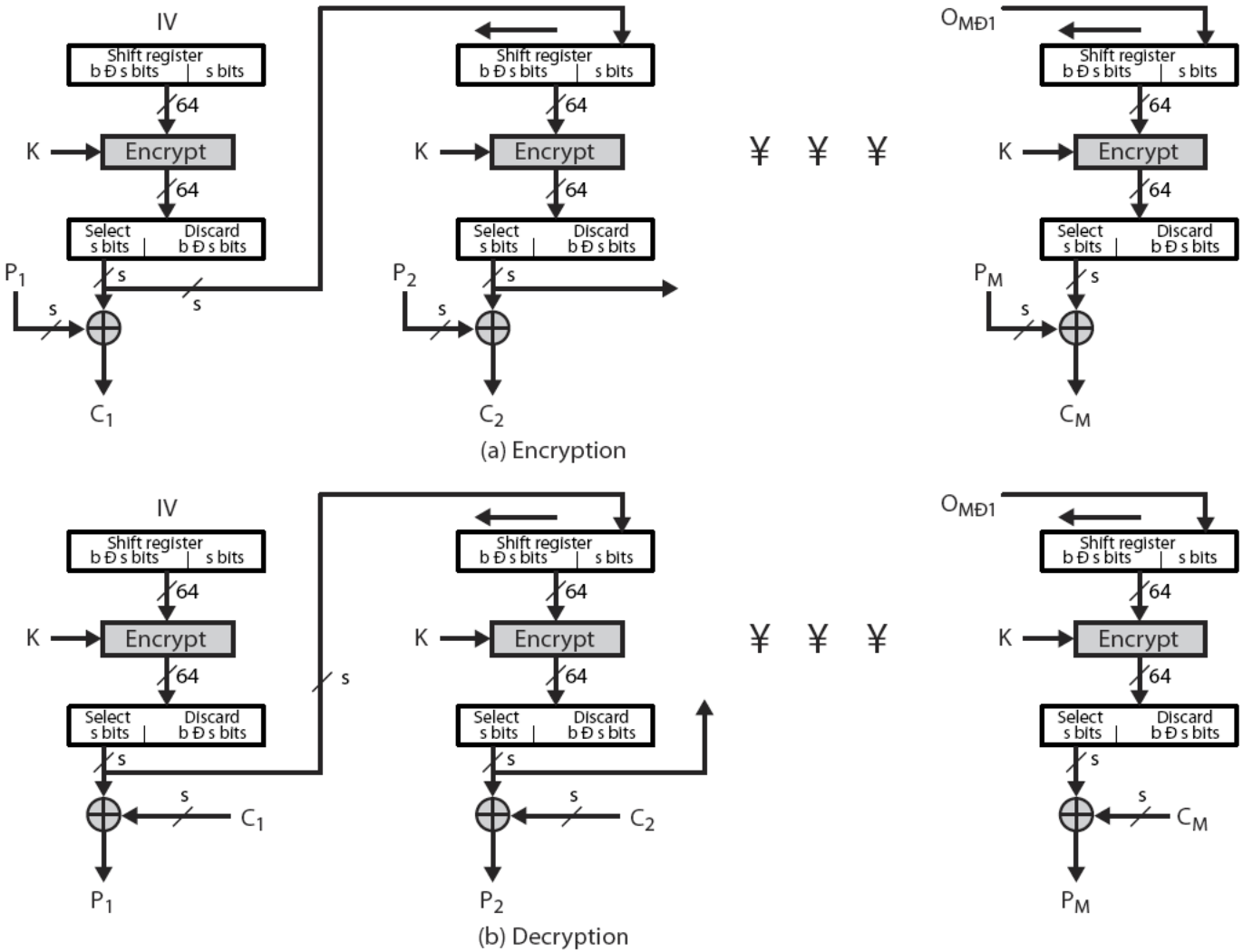
Output Feed Back

- **message is treated as a stream of bits**
- **output of cipher is added to message**
- **output is then feed back (hence name)**
- **feedback is independent of message**
- **can be computed in advance**

$$C_i = P_i \text{ XOR } O_i$$
$$O_i = DES_{K1}(O_{i-1})$$
$$O_{-1} = IV$$

- **uses: stream encryption on noisy channels**

Output FeedBack (OFB)



计数器模式(CTR)

Counter

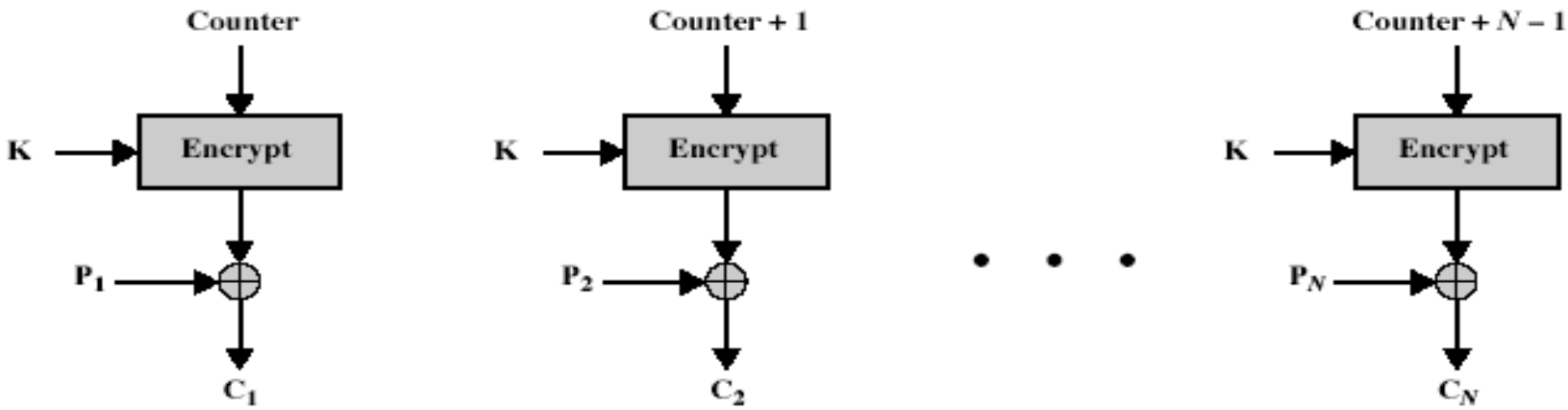
- a “new” mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

$$C_i = P_i \text{ XOR } O_i$$

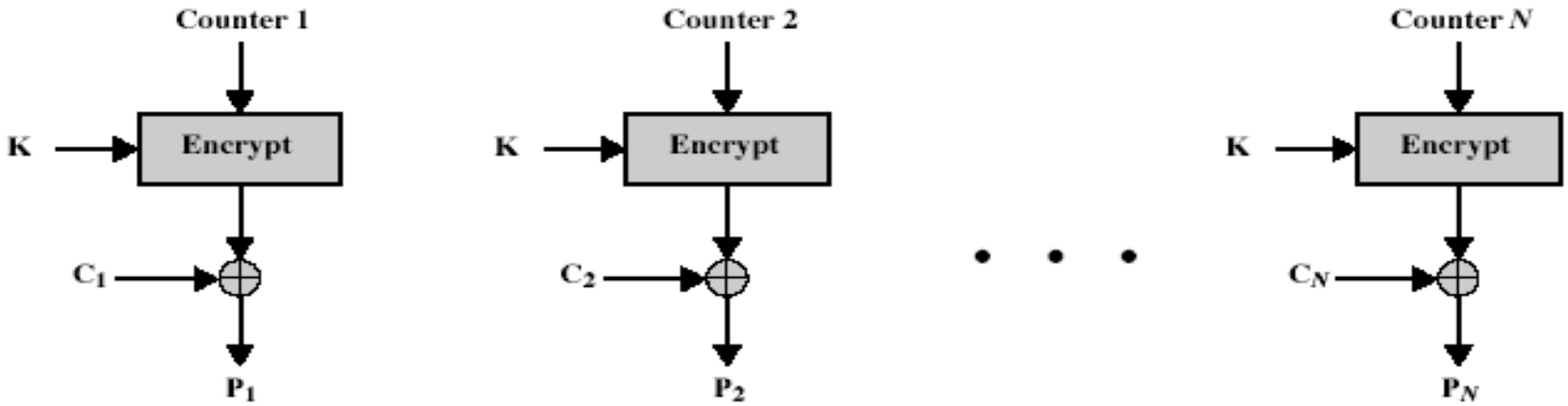
$$O_i = DES_{K1}(i)$$

- uses: high-speed network encryptions

Counter (CTR)



(a) Encryption

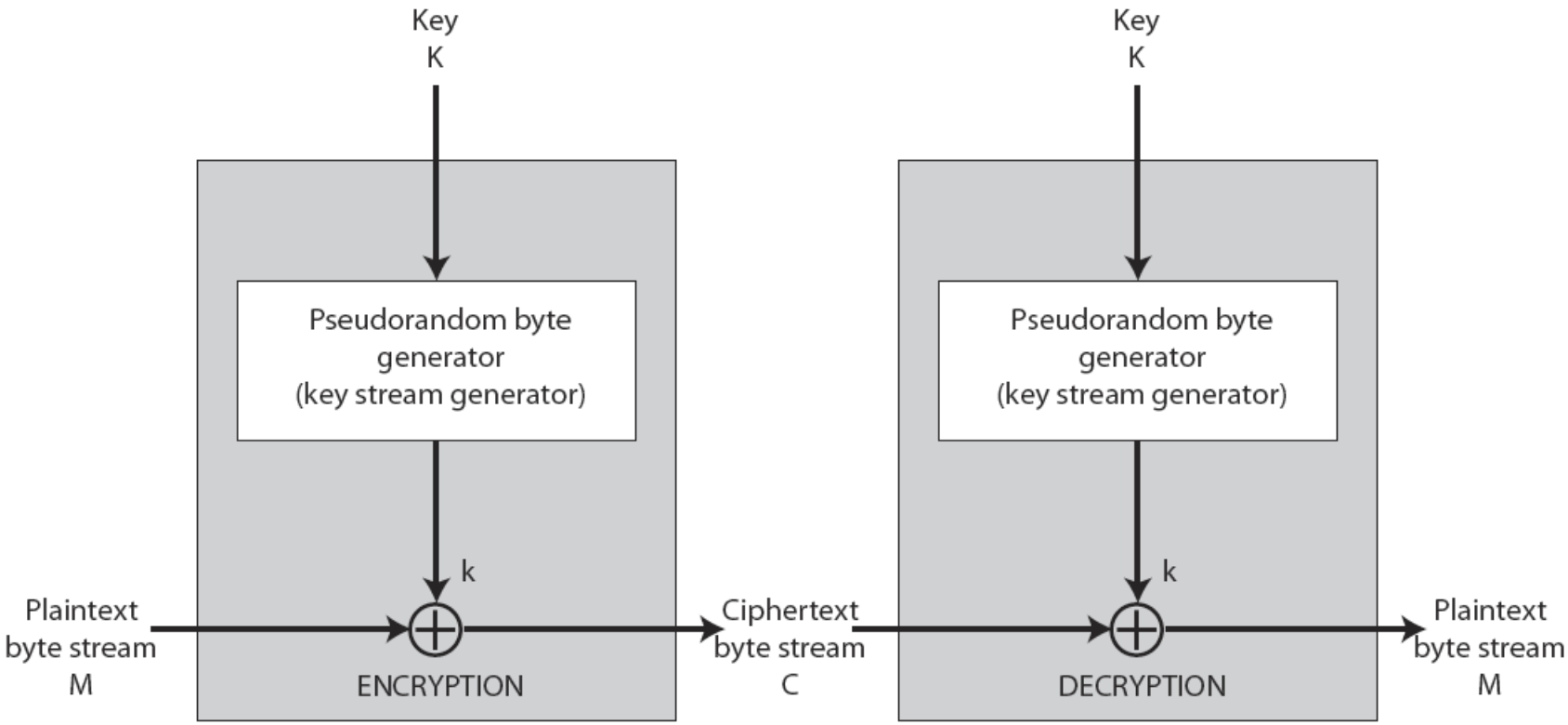


(b) Decryption

流密码 (Stream Ciphers)

- process message bit by bit (as a stream)
- have a pseudo random keystream
- combined (XOR) with plaintext bit by bit
- randomness of stream key completely destroys statistically properties in message
  - $C_i = M_i$ XOR $StreamKey_i$
- but must never reuse stream key
  - otherwise can recover messages (cf book cipher)

Stream Cipher Structure



Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - large linear complexity
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

RC4流密码

- RC4是一个密钥大小可变的序列密码，于1987年由Ron Rivest 为RSA公司开发;是一个面向字节的流密码.
- RC4的密码序列独立于明文. RSA公司声称RC4对线性和差分分析具有免疫力.
- RC4是流密码，必须避免重复使用密钥;应用于SSL/TLS协议，WEP（IEEE802.11)
- 1994年匿名公开于Internet上.

RC4 算法的初始化

初始化一个256个字节的状态矢量S, Fill S[1] to S[255] linearly (i.e. S[0] = 0; S[1] = 1 ... S[255] = 255);将密钥K的值按如下程序赋给临时变量T

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

事实上,对S的操作就是通过对换来达到置换的目的.

RC4 密钥流的生成(S初始化后,输入密钥就不再使用.密钥流的生成,当 S[255]完成后,操作从S[0]再次开始.)

i,j=0

While(true)

i = (i + 1) mod 256

j = (j + S[i]) mod 256

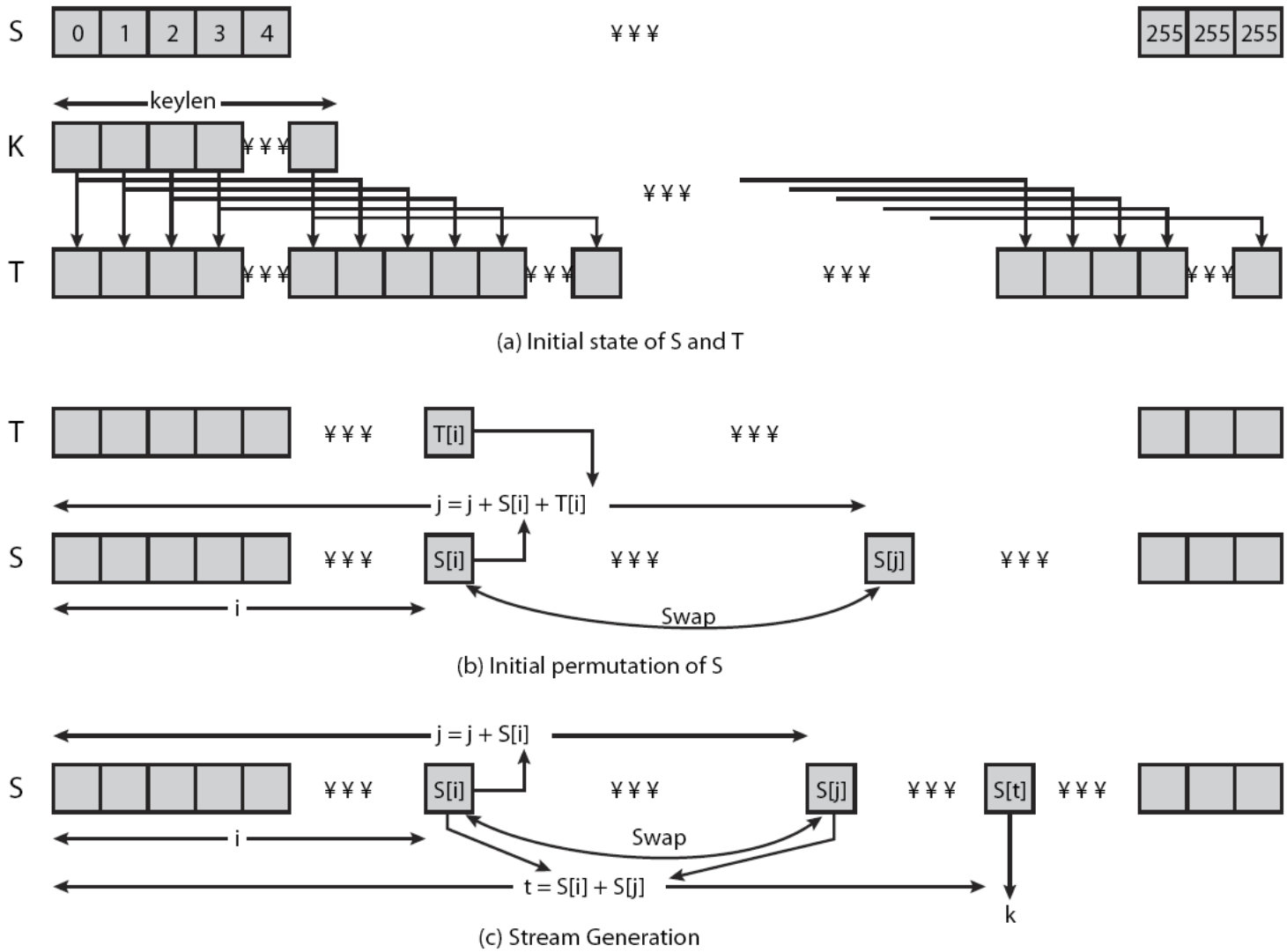
Swap S[i] and S[j]

t = (S[i] + S[j]) mod 256

k = S[t]

k 与明文异或产生密文，与密文异或产生明文.

RC4 图示


S
0
1
2
3
4
255
255
255
keylen
K
T
(a) Initial state of S and T
T
T[i]
j = j + S[i] + T[i]
S
S[i]
S[j]
i
Swap
(b) Initial permutation of S
j = j + S[i]
S
S[i]
S[j]
S[t]
i
Swap
t = S[i] + S[j]
k
(c) Stream Generation

RC4 安全性(见书137页)

- **claimed secure against known attacks**
  - **– have some analyses, none practical**
- **result is very non-linear**
- **since RC4 is a stream cipher, must** never reuse a key
- **have a concern with WEP, but due to key handling rather than RC4 itself**