网络与信息安全
密码学基础(一)

潘爱民，北京大学计算机研究所

panaimin@icst.pku.edu.cn

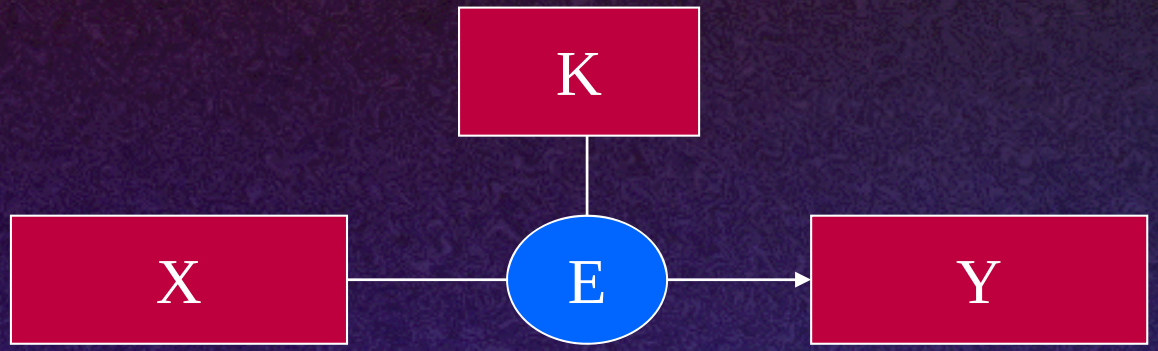
http://www.icst.pku.edu.cn/InfoSecCourse

内容

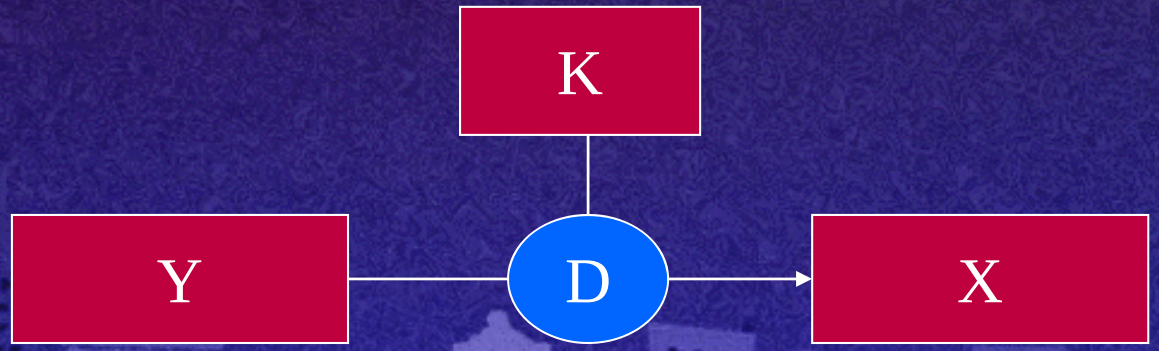
- 对称加密算法
  - 经典密码算法
  - 现代密码算法
  - **AES**
- 随机数发生器

对称加密算法的基本模型

- 加密: E: (X,K) → Y, y = E(x,k)



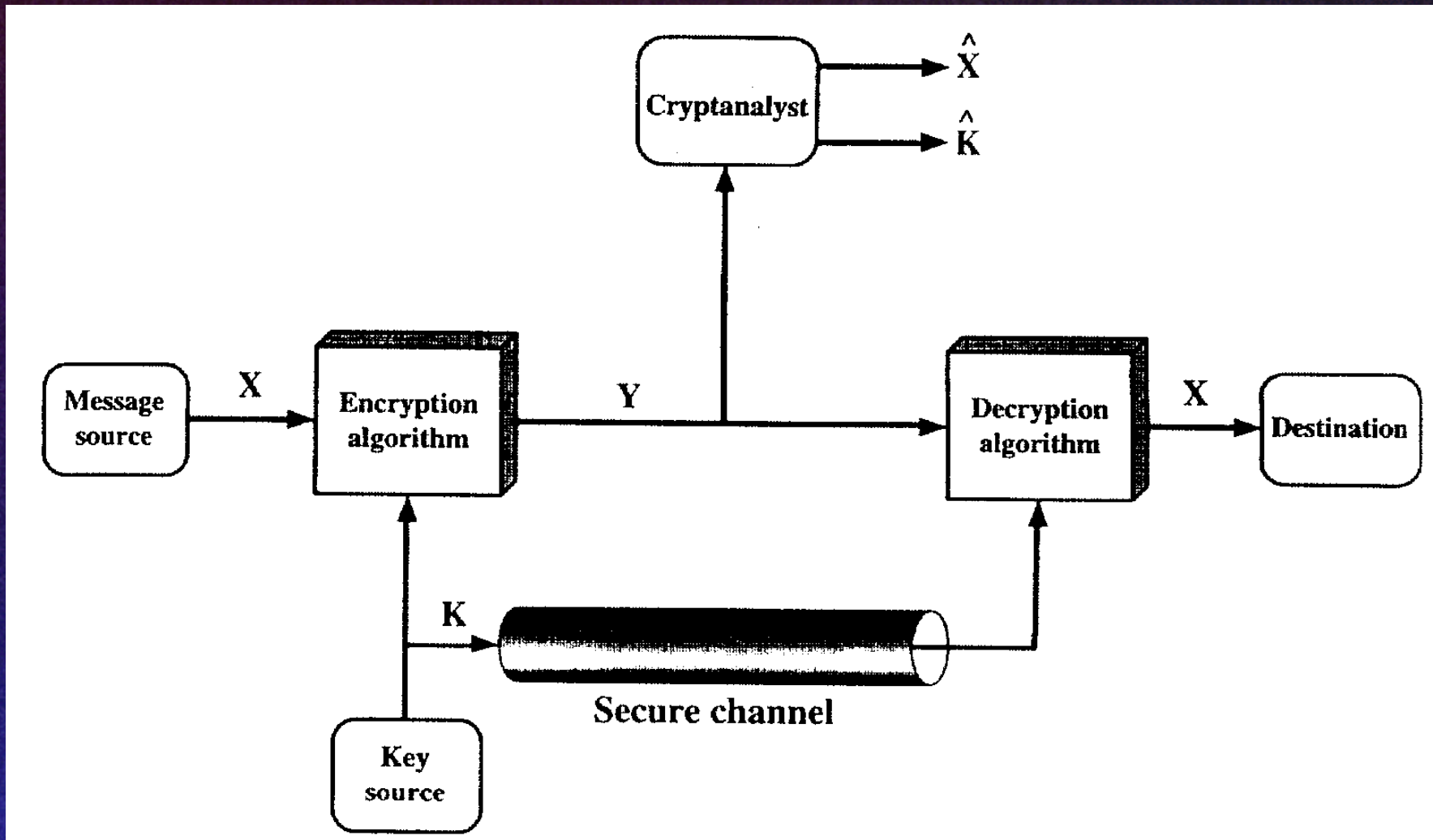
- 解密: D: (Y,K) → X, x = D(y,k)



对称加密算法研究

- 对称加密算法的特点
  - 算法强度足够
  - 安全性依赖于密钥，不是算法
  - 速度快
- 两门学科
  - 密码学
  - 密码分析

用对称加密算法建立起来的安全通讯



密码学

- 三种考虑角度
  - （1）从明文到密文的变换
    - 替换(substitution)
    - 置换(transposition)
    - ……
  - （2）钥匙的数目
    - 对称、单钥加密法
    - 双钥、公钥加密
  - （3）明文的处理方式
    - 分组加密（块加密算法）
    - 流方式加密

密码分析

- 发现X和K的过程被称为密码分析
  - 分析的策略取决于加密的技术以及可利用的信息，在加密算法设计和攻击时都需要用到的技术
- 根据可利用信息的不同，可分为5类：
  - （1）只有密文
  - （2）已知部分明文-密文对
  - （3）选择明文
  - （4）选择密文

*（1）（2）（3）常见、（4）不常见

加密算法的有效性

- **Unconditionally secure，绝对安全？**
  - 永不可破，是理想情况，理论上不可破，密钥空间无限，在已知密文条件下，方程无解。但是我们可以考虑：
    - 破解的代价超过了加密信息本身的价值
    - 破解的时间超过了加密信息本身的有效期
- **Computationally secure，**
  - 满足上述两个条件

直觉：什么是一个好的加密算法

- 假设密码(password)k是固定的
- 明文和密文是一个映射关系：单射，即
  $E_k(x_1) != E_k(x_2)$ if $x_1 != x_2$
- 通常情况是：明文非常有序
- 好的密码条件下，我们期望得到什么样的密文
  - 随机性
- 如何理解随机性
  - 静态：特殊的点
  - 动态：小的扰动带来的变化不可知

考虑设计一个加密算法

- 打破明文本身的规律性
  - 随机性(可望不可及)
  - 非线性(一定要)
  - 统计意义上的规律
- 多次迭代
  - 迭代是否会增加变换的复杂性
  - 是否存在通用的框架，用于迭代
- 复杂性带来密码分析的困难和不可知性
  - 实践的检验和考验

已有密码算法的讨论

- 经典密码算法
  - 替换技术
  - 置换技术
- 现代密码算法
  - **DES**
  - 其他密码算法
- **AES密码算法**
  - **Rijndael**

经典密码算法

- 替换技术
  - **Caesar**加密制
  - 单表替换加密制
  - **Playfair**加密制
  - **Hill**加密制
  - 多表加密制
- 置换技术
  - 改变字母的排列顺序，比如
    - 用对角线方式写明文，然后按行重新排序
    - 写成一个矩阵，然后按照新的列序重新排列
- 转轮加密体制
- 多步结合

经典密码算法特点

- 要求的计算强度小
- DES之前
- 以字母表为主要加密对象
- 替换和置换技术
- 数据安全基于算法的保密
- 密码分析方法基于明文的可读性以及字母和字母组合的频率特性

现代密码算法

- DES(Data Encryption Standard)
- IDEA
- Blowfish
- RC5
- CAST-128
- ……

分组密码算法设计指导原则

- **Diffusion(发散)**
  - 小扰动的影响波及到全局
  - 密文没有统计特征，明文一位影响密文的多位，增加密文与明文之间关系的复杂性
- **Confusion(混淆)**
  - 强调密钥的作用
  - 增加密钥与密文之间关系的复杂性
- 结构简单、易于分析

Feistel分组加密算法结构之动机

- 分组加密算法，一一映射
- 当n较小时，等价于替换变换

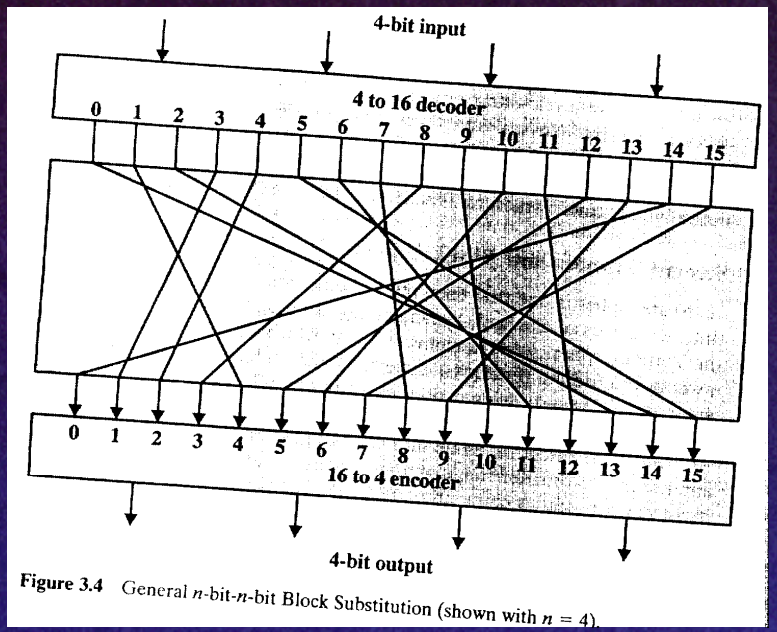


Figure 3.4 General *n*-bit-*n*-bit Block Substitution (shown with $n = 4$).

Table 3.1 Encryption and Decryption Tables for Substition Cipher of Figure 3.4

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

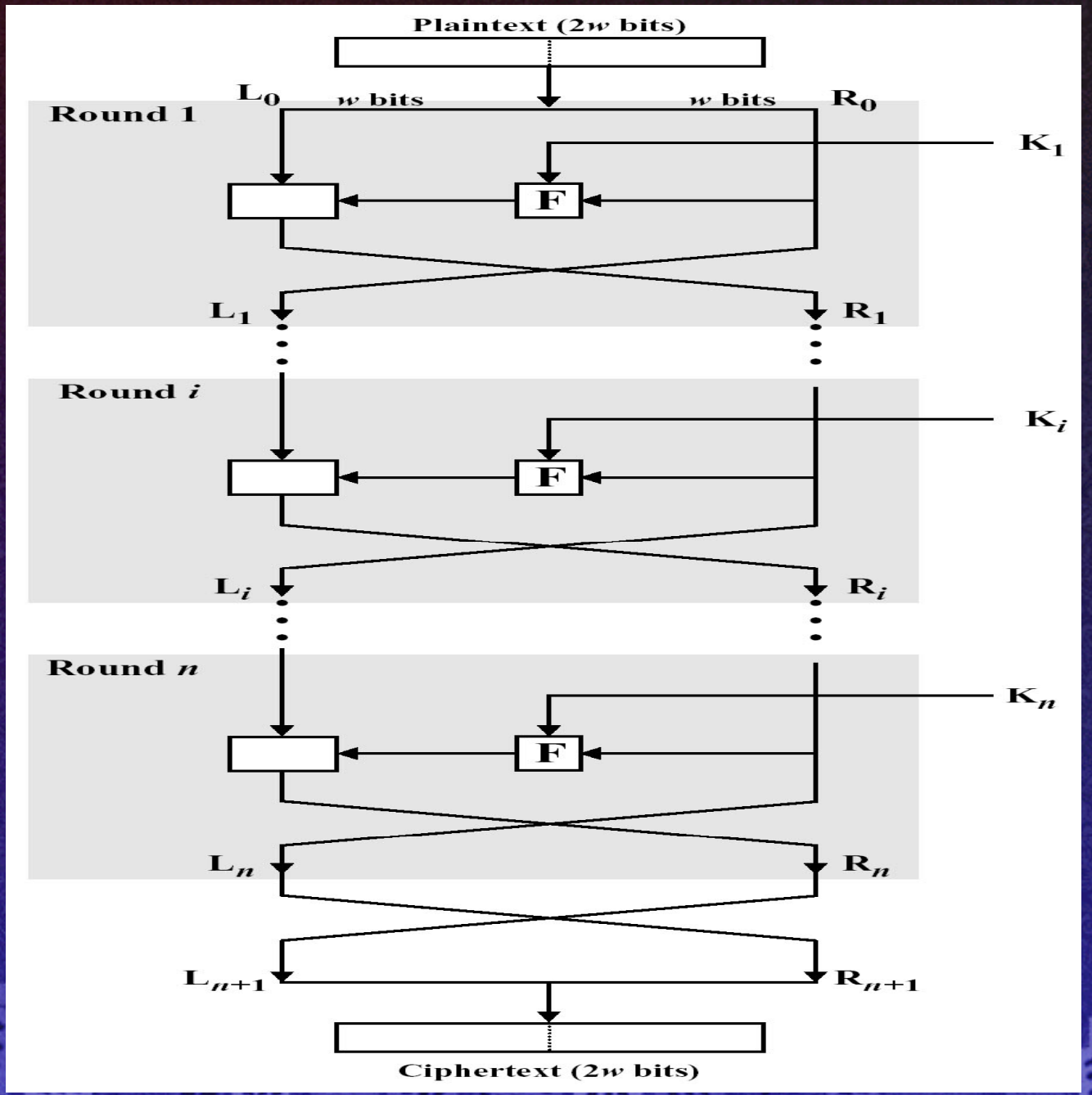
Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

- 当n较大时，比如n=64，无法表达这样的任意变换。
- Feistel结构很好地解决了二者之间的矛盾

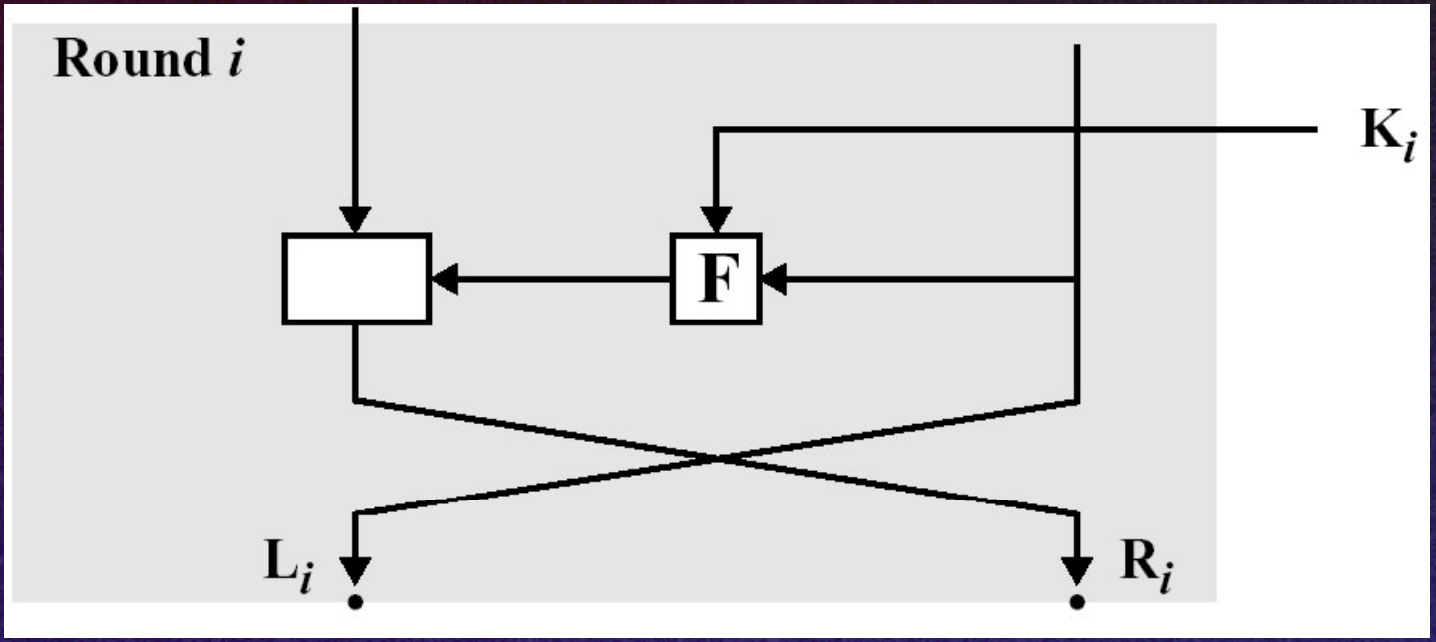
Feistel分组加密算法结构之思想

- 基本思想：用简单算法的乘积来近似表达大尺寸的替换变换
- 多个简单算法的结合得到的加密算法比任何一个部分算法都要强
- 交替使用替换变换和排列(permutation)
- 混淆(confusion)和发散(diffusion)概念的应用

Feistel 结构图


Plaintext (2w bits)
L0
w bits
w bits
R0
Round 1
K1
F
L1
R1
Round i
Ki
F
Li
Ri
Round n
Kn
F
Ln
Rn
Ln+1
Rn+1
Ciphertext (2w bits)

Feistel结构定义



- 加密: $L_i = R_{i-1}$; $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$
- 解密: $R_{i-1} = L_i$

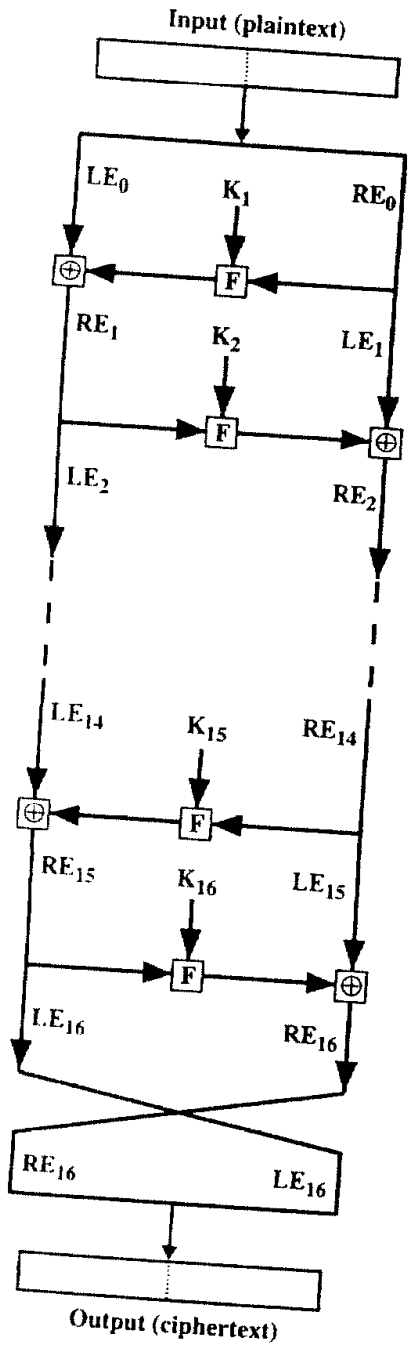
$$L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$$

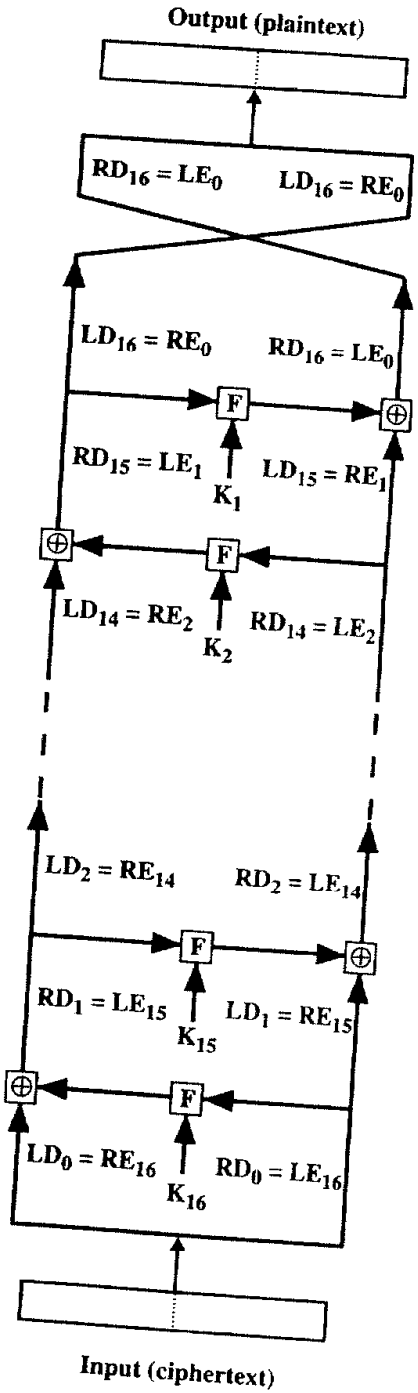
$$= R_i \oplus F(L_i, K_i)$$

Feistel分组加密算法特点

- 分组大小。越大安全性越高，但速度下降，64比较合理
- 密钥位数。越大安全性越高，但速度下降，64广泛使用，但现在已经不够用—〉128
- 步数，典型16步
- 子钥产生算法。算法越复杂，就增加密码分析的难度
- 每一步的子函数。函数越复杂，就增加密码分析的难度
- 快速软件实现，包括加密和解密算法
- 易于分析。便于掌握算法的保密强度以及扩展办法。

Feistel分组加密算法之解密算法





Feistel分组加密算法之解密算法推导

加密：

$LE_{16} = RE_{15}$

$RE_{16} = LE_{15} \oplus F(RE_{15}, K_{16})$

解密：

$LD_1 = RD_0 = LE_{16} = RE_{15}$

$RD_1 = LD_0 \oplus F(RD_0, K_{16})$

$= RE_{16} \oplus F(RE_{15}, K_{16})$

$= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16})$

$= LE_{15}$

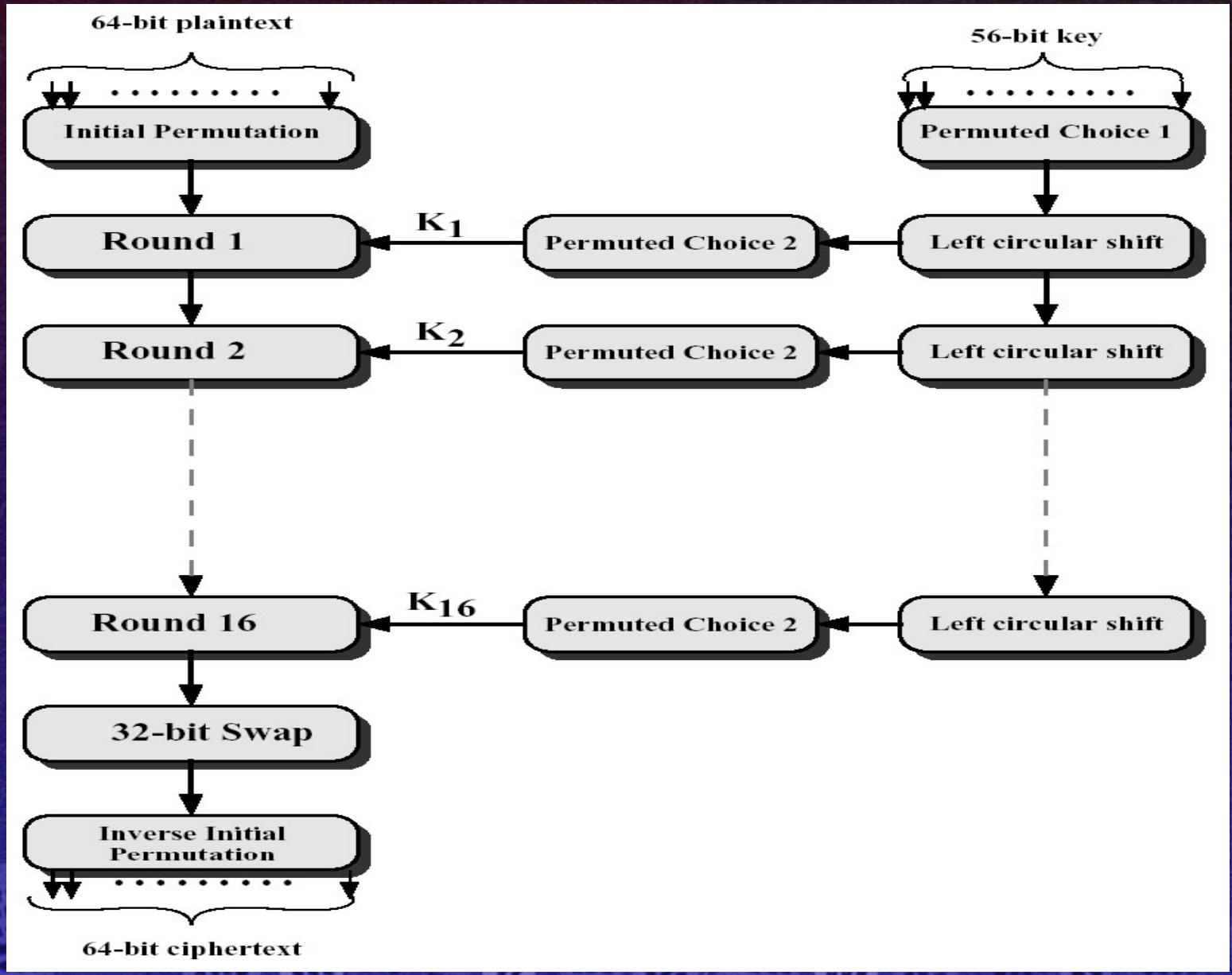
$$\begin{cases} LE_i = RE_{i-1} \\ RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i) \end{cases}$$

$$\Rightarrow \begin{cases} RE_{i-1} = LE_i \\ LE_{i-1} = RE_i \oplus F(RE_{i-1}, K_i) = RE_i \oplus F(LE_i, K_i) \end{cases}$$

DES算法

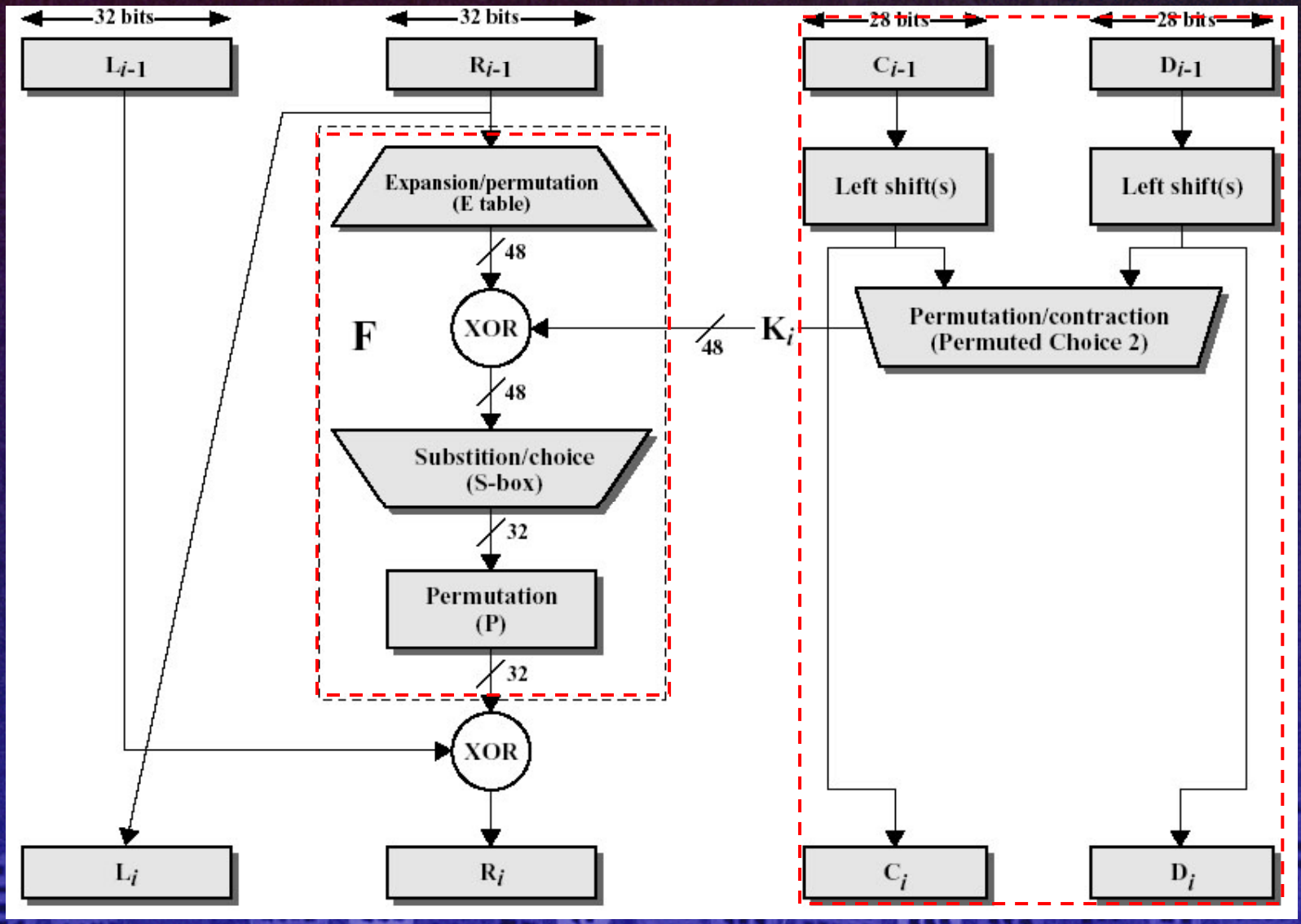
- 1977年由美国的标准化局(NBS，现为NIST)采纳
- 64位分组、56位密钥
- 历史：
  - IBM在60年代启动了LUCIFER项目，当时的算法采用128位密钥
  - 改进算法，降低为56位密钥，IBM提交给NBS(NIST)，于是产生DES

DES算法基本结构

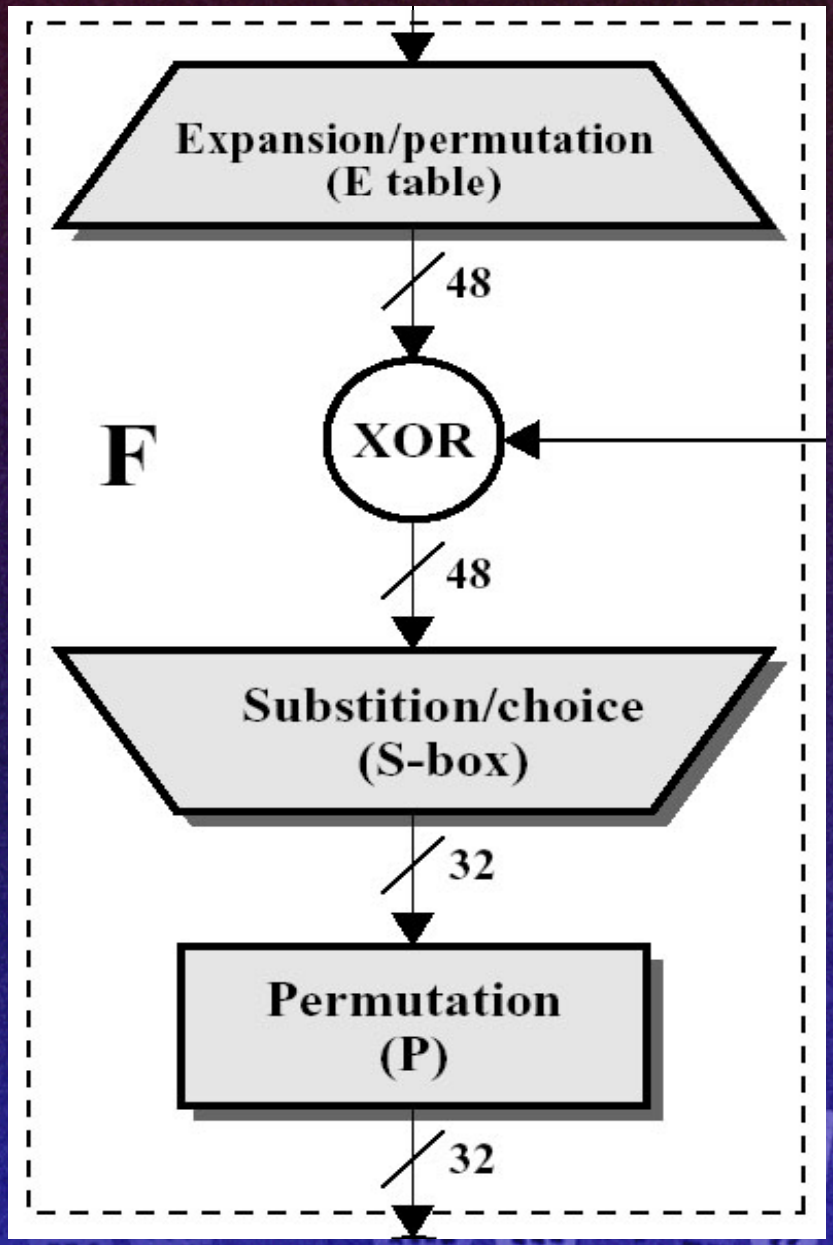

64-bit plaintext
56-bit key
Initial Permutation
Permuted Choice 1
Round 1
K1
Permuted Choice 2
Left circular shift
Round 2
K2
Permuted Choice 2
Left circular shift
Round 16
K16
Permuted Choice 2
Left circular shift
32-bit Swap
Inverse Initial Permutation
64-bit ciphertext

DES: 每一轮

$$L_i = R_{i-1} \quad R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$



DES: Function F



Expansion: 32 → 48

S-box: 6 → 4

Permutation

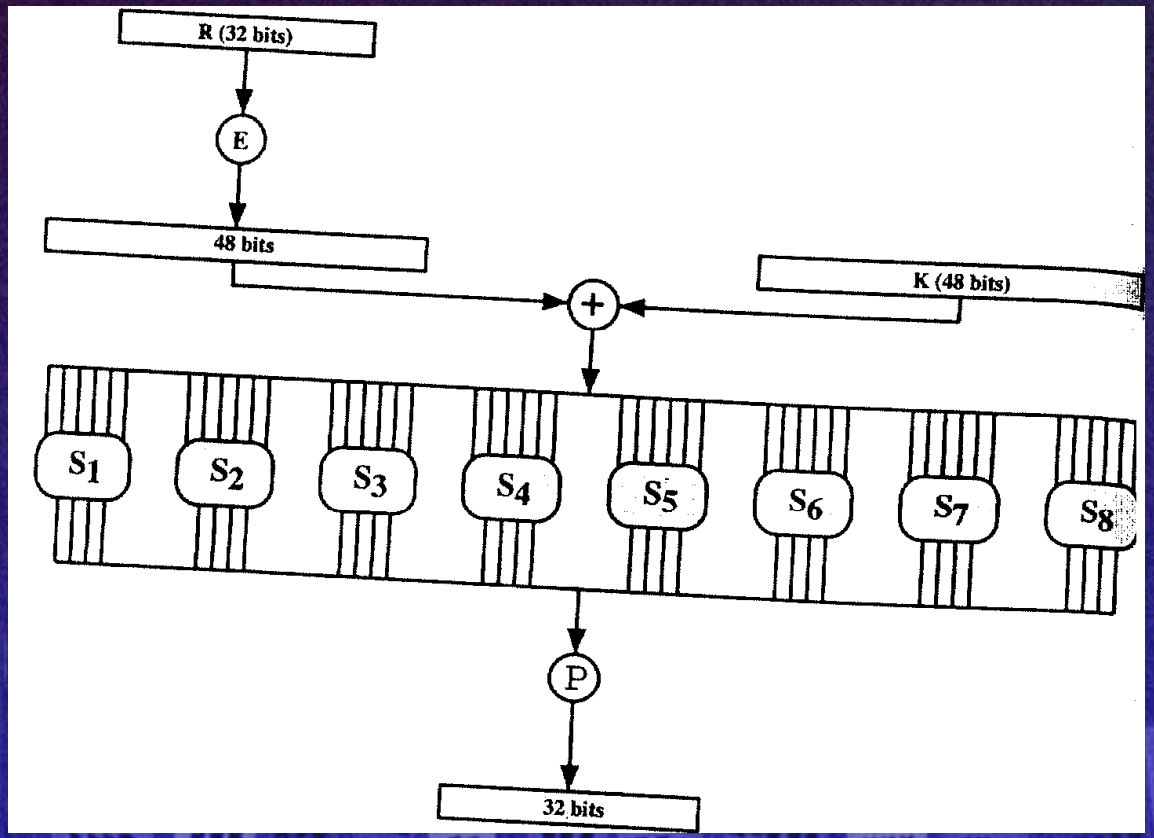
DES: 32位到48位的扩展表

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

DES: S-box

S(1):

0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

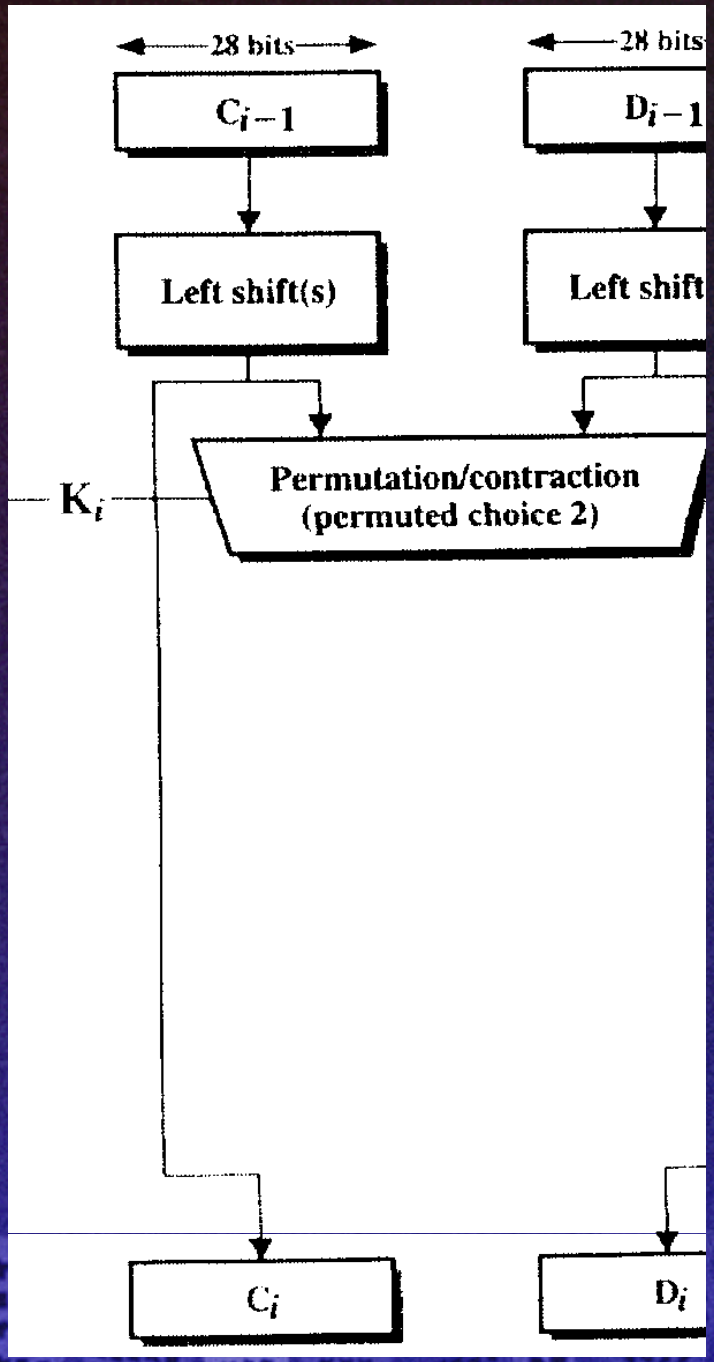


DES: Permutation

```
16 07 20 21 29 12 28 17
01 15 23 26 05 18 31 10
02 08 24 14 32 27 03 09
19 13 30 06 22 11 04 25
```

DES
之每步密钥产生过程

- PC-1
  - 在第一步之前
- PC2
- 左移位数目表
  - 1或者2位



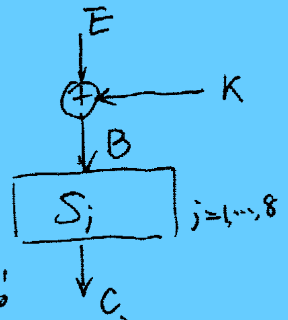
DES的强度

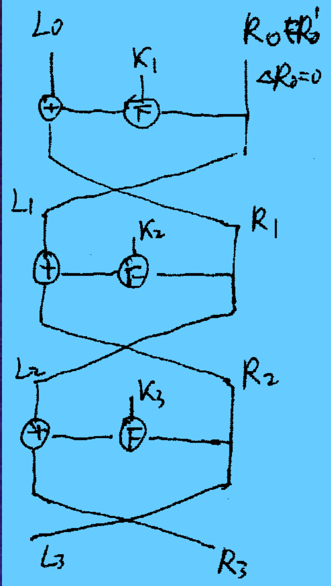
- 56位密钥的使用
  - 理论上的强度，97年$100000的机器可以在6小时内用穷举法攻破DES
  - 实际攻破的例子，97年1月提出挑战，有人利用Internet的分布式计算能力，组织志愿军连接了70000多个系统在96天后攻破
- DES算法的本质
  - 关键在于8个S-BOX
- 针对DES的密码分析
  - 差分分析法
  - 线性分析法

差分分析法

- 属于选择明文攻击
- 基本思想：通过分析特定明文差对结果密文差的影响来获得可能性最大的密钥
- 差分在DES的16步加密过程中的传递，每一个S-BOX对于差分传递的概率特性
- 子密钥不影响每一步的输入差分，但是影响输出的差分
- 针对每一个DES步骤，用差分法可以推导出该步的密钥

每一步的差分分析法





输入 $E, E' \in Z_2^6$, $K \in Z_2^6$, $B, B' \in Z_2^6$

输出 $C, C' \in Z_2^4$.

$$\Delta B = B \oplus B' = (E \oplus K) \oplus (E' \oplus K) = E \oplus E' = \Delta E$$

定义1：$IN_j(B_j', C_j') = \{B \in Z_2^6 \mid S_j(B) \oplus S_j(B \oplus B_j') = C_j'\}$

即：$IN_j(B_j', C_j')$ 表示输入异或为 B_j'，输出异或为 C_j' 的所有可能输入的集合.

对于每一个S-BOX，我们可以构造出这样的 IN_j 集合.

定义2：$test_j(E, E_j', C_j') = \{B \oplus E_j \mid B \in IN_j(E_j', C_j')\}$

即：$test_j(E, E_j', C_j')$ 为 E 和 $IN_j(E_j', C_j')$ 中的每个元素取异或后所得异或值作成的集合.

结论：E 和 E' 为两个输入 (S_j)，输出异或为 ΔC，设 $\Delta E = E \oplus E'$，则 $K \in test_j(E, E', \Delta C)$

提示：$E' = \Delta E \oplus E$，只须证明：$K \oplus E \in IN(E', \Delta C)$

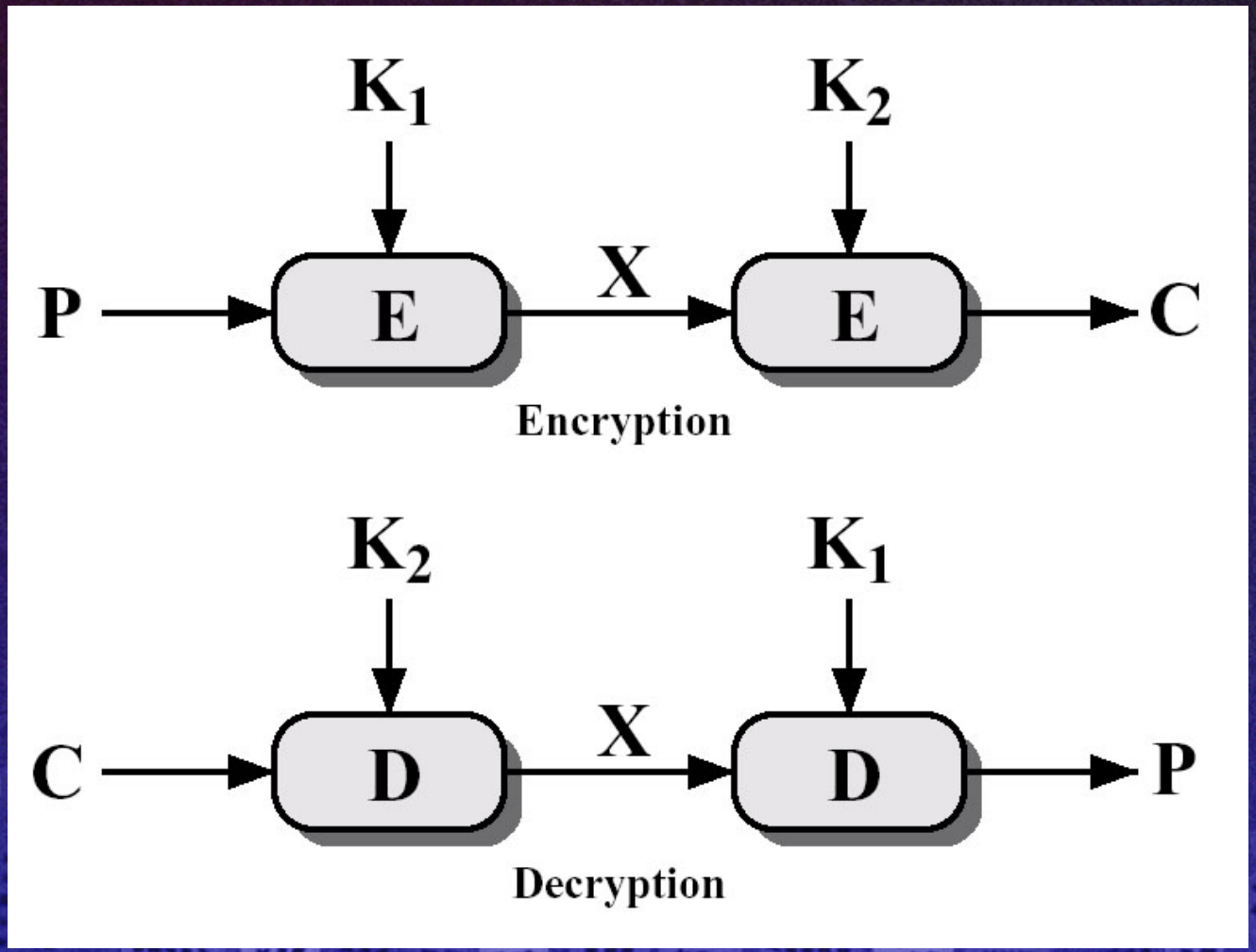
3-轮DES可以直接推导出密钥，不涉及概率传递分析. 选择明文

针对DES的密码分析

- 差分分析法
  - 2^{47}对选择明文，经过2^{47}量级的计算可攻破
- 线性分析法
  - 思想：用线性近似描述DES变换
  - 根据2^{47}已知明文，可以找到DES的密钥

二重DES

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



针对两重分组加密算法的中间相会攻击

中间相会攻击：

$$C=E_{K_2}[E_{K_1}[P]] \Rightarrow X=E_K[P]=D_{K_2}[C]$$

对于一对已知的(P, C)

① $\forall K_1 \in K_{56}$，计算$E_{K_1}[P]$，将2^{56}个结果排序

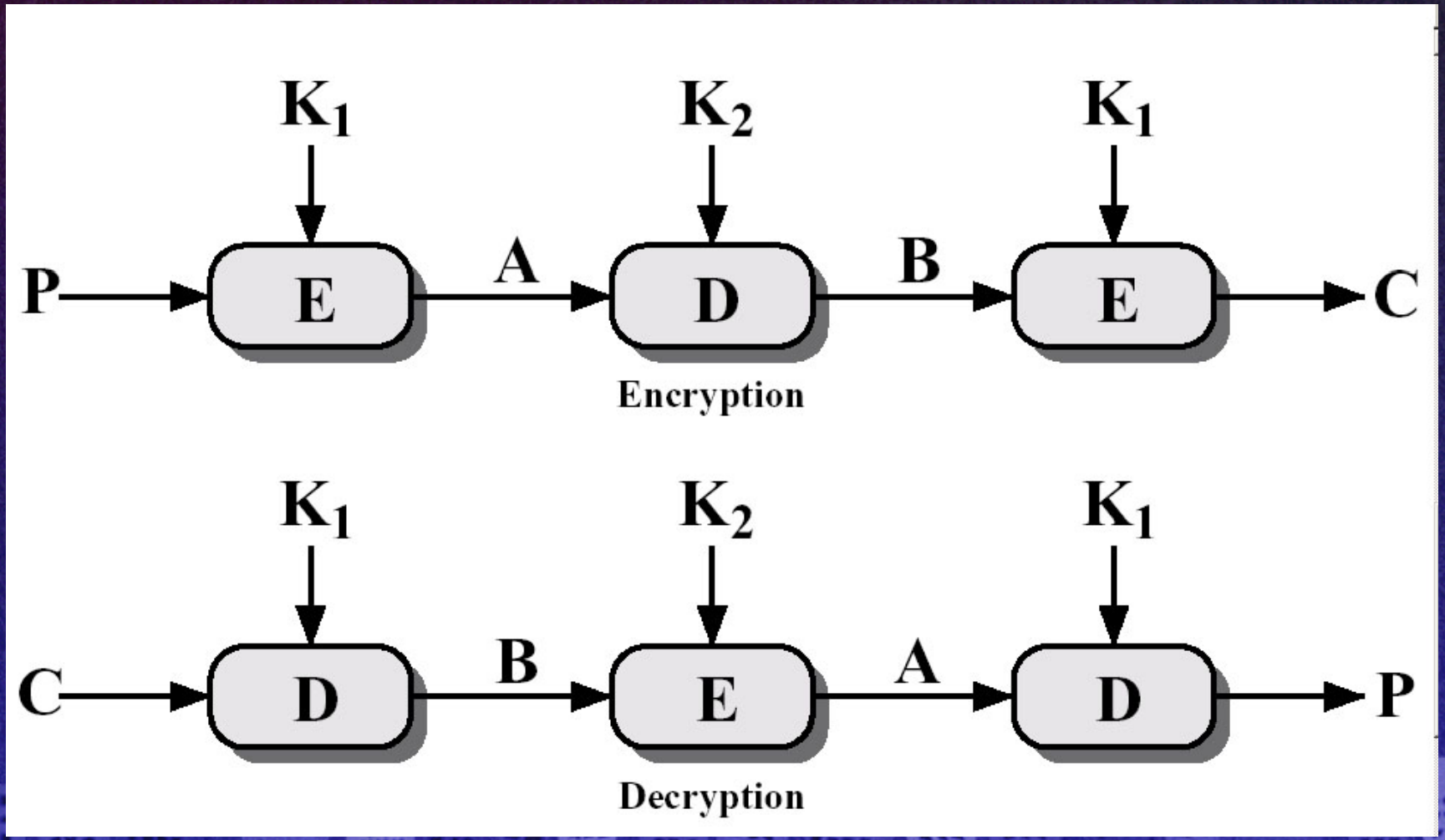
② $\forall K_2 \in K_{56}$，计算$D_{K_2}[C]$，将2^{56}个结果排序

③ 进行匹配搜索，每找到一对再用其它(P, C)对进行试验，直到找到为止。

密钥空间为2^{112}所，但2^{57}个DES操作。

三重DES

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

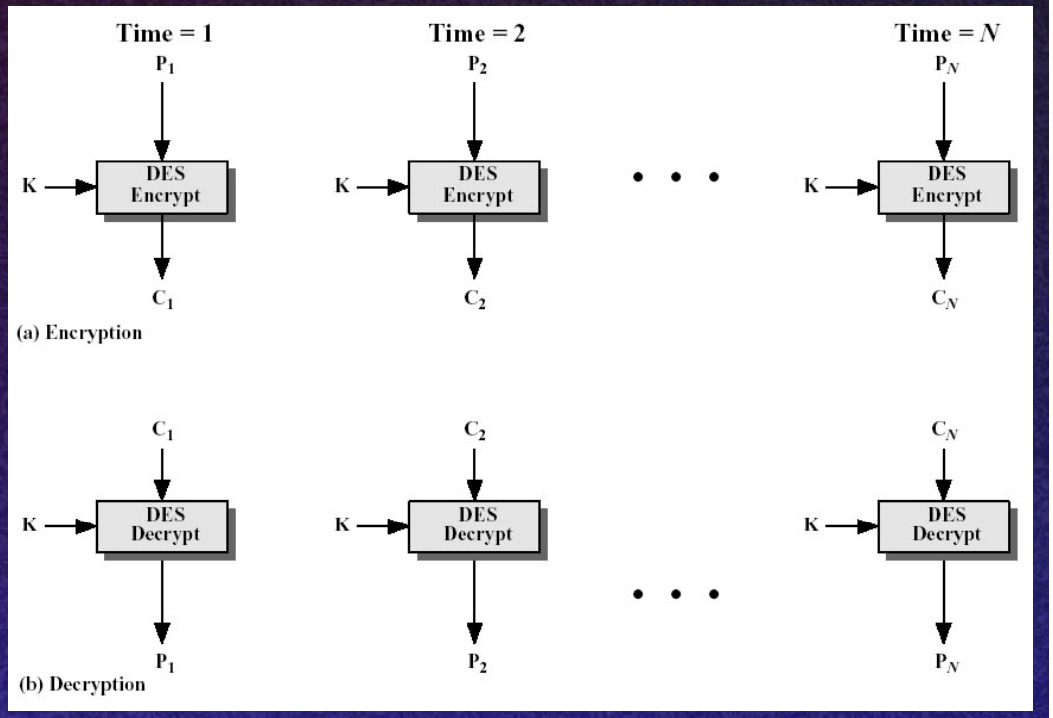


分组密码的用法

- 电子簿模式(electronic codebook mode)ECB
- 密码块链接(cipher block chaining)CBC
- 密码反馈方式(cipher feedback)CFB
- 输出反馈方式(output feedback)OFB

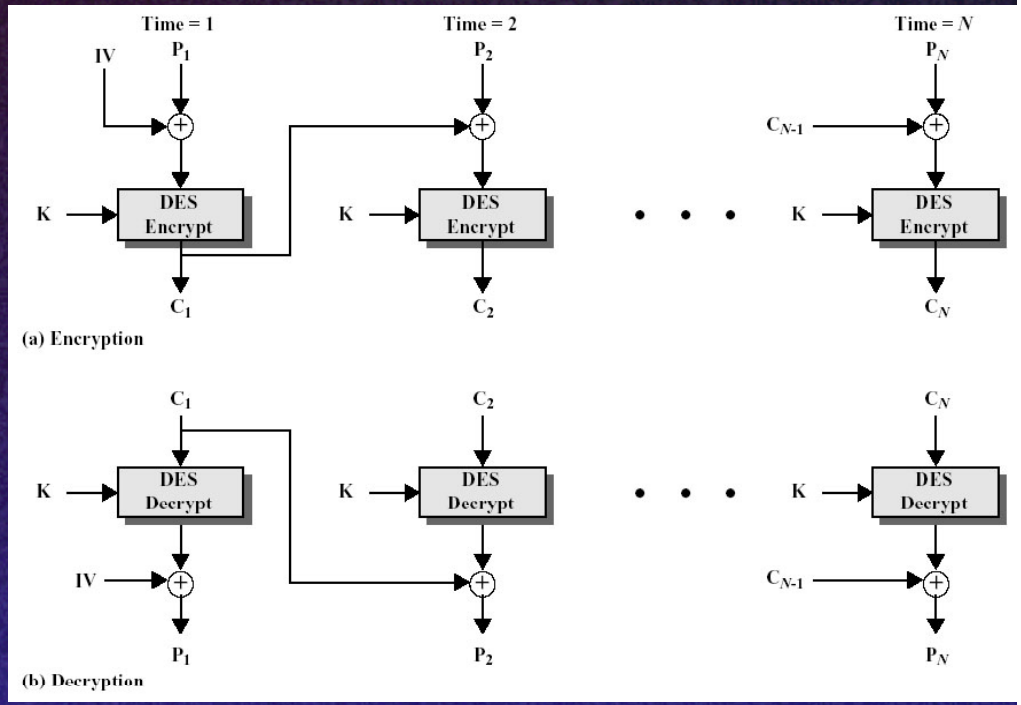
电子簿模式ECB

- 相同明文⇨相同密文
- 同样信息多次出现造成泄漏
- 信息块可被替换
- 信息块可被重排
- 密文块损坏⇨仅对应明文块损坏
- 适合于传输短信息



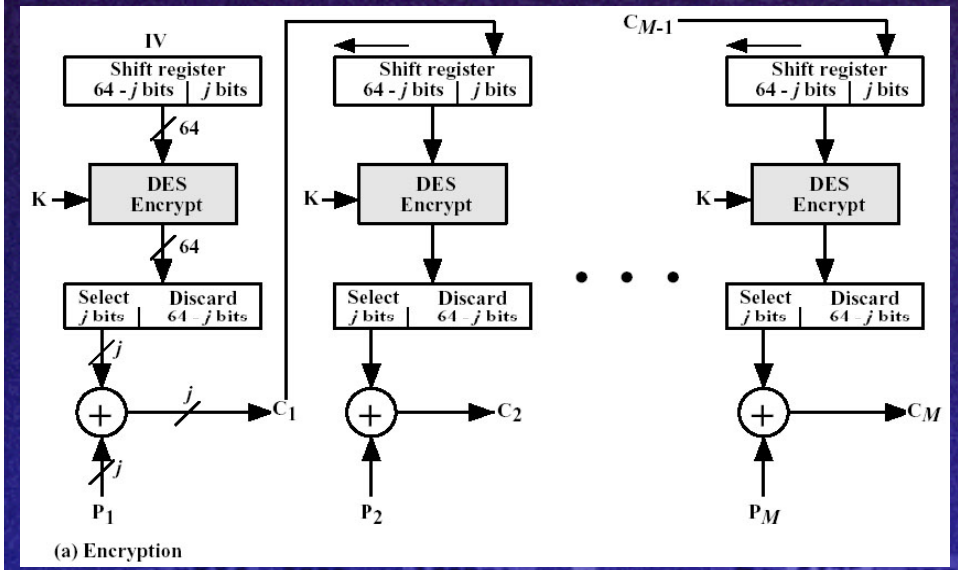
密码块链接CBC

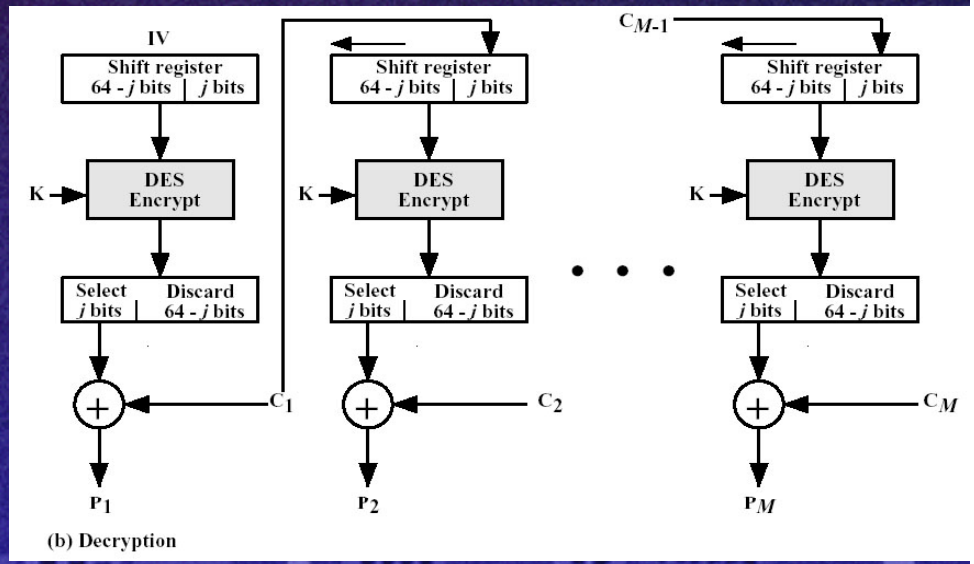
- 需要共同的初始化向量IV
- 相同明文⇨不同密文
- 初始化向量IV可以用来改变第一块
- 密文块损坏⇨两明文块损坏
- 安全性好于ECB



密码反馈方式CFB

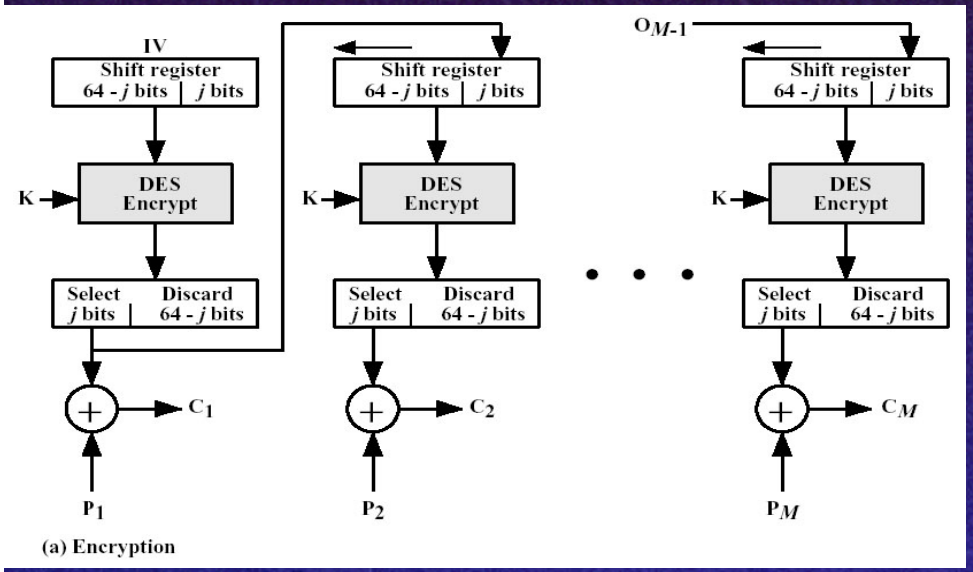
- CFB:分组密码⇨流密码
- 需要共同的移位寄存器初始值IV
- 对于不同的消息，IV必须唯一
- 一个单元损坏影响多个单元: (W+j-1)/j
  W为分组加密块大小，j为流单元位数



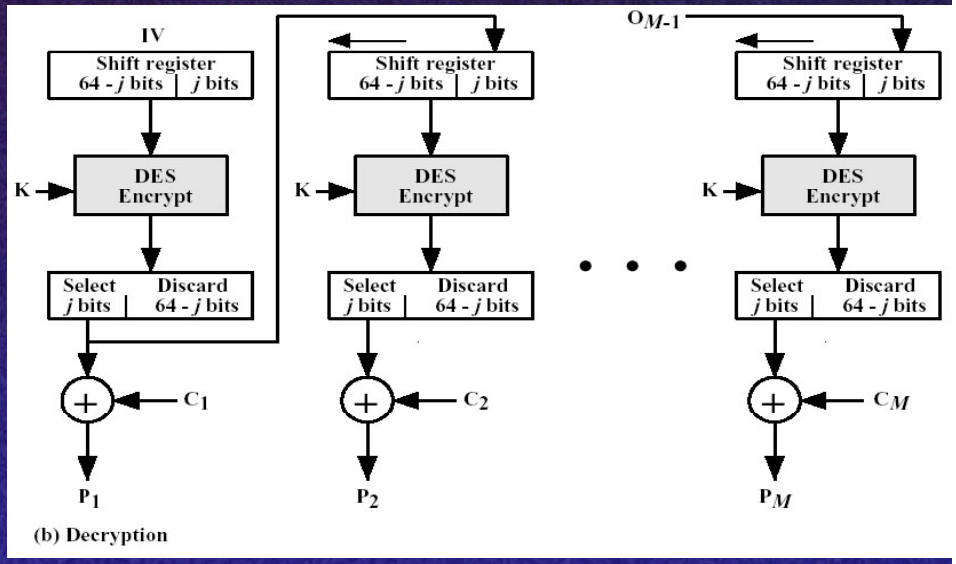


输出反馈方式OFB

- OFB:分组密码⇨流密码
- 需要共同的移位寄存器初始值IV
- 一个单元损坏只影响对应单元



(a) Encryption



(b) Decryption

分组密码与序列(流)密码

- 定义：
  - 分组密码(block cipher)是对一个大的明文块(block)进行固定变换的操作
  - 序列密码(stream cipher)也叫流密码，是对单个明文位(组)的随时间变换的操作
- 相互转换
- 序列密码
  - 异或One-time pad

其他的密码算法

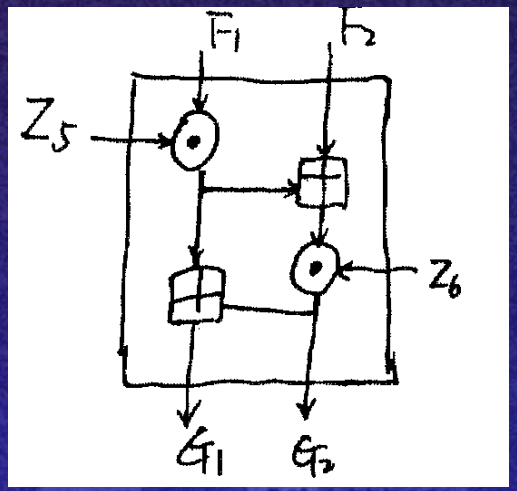
- IDEA
- Blowfish
- RC5
- CAST-128
- RC2
- RC4

IDEA算法

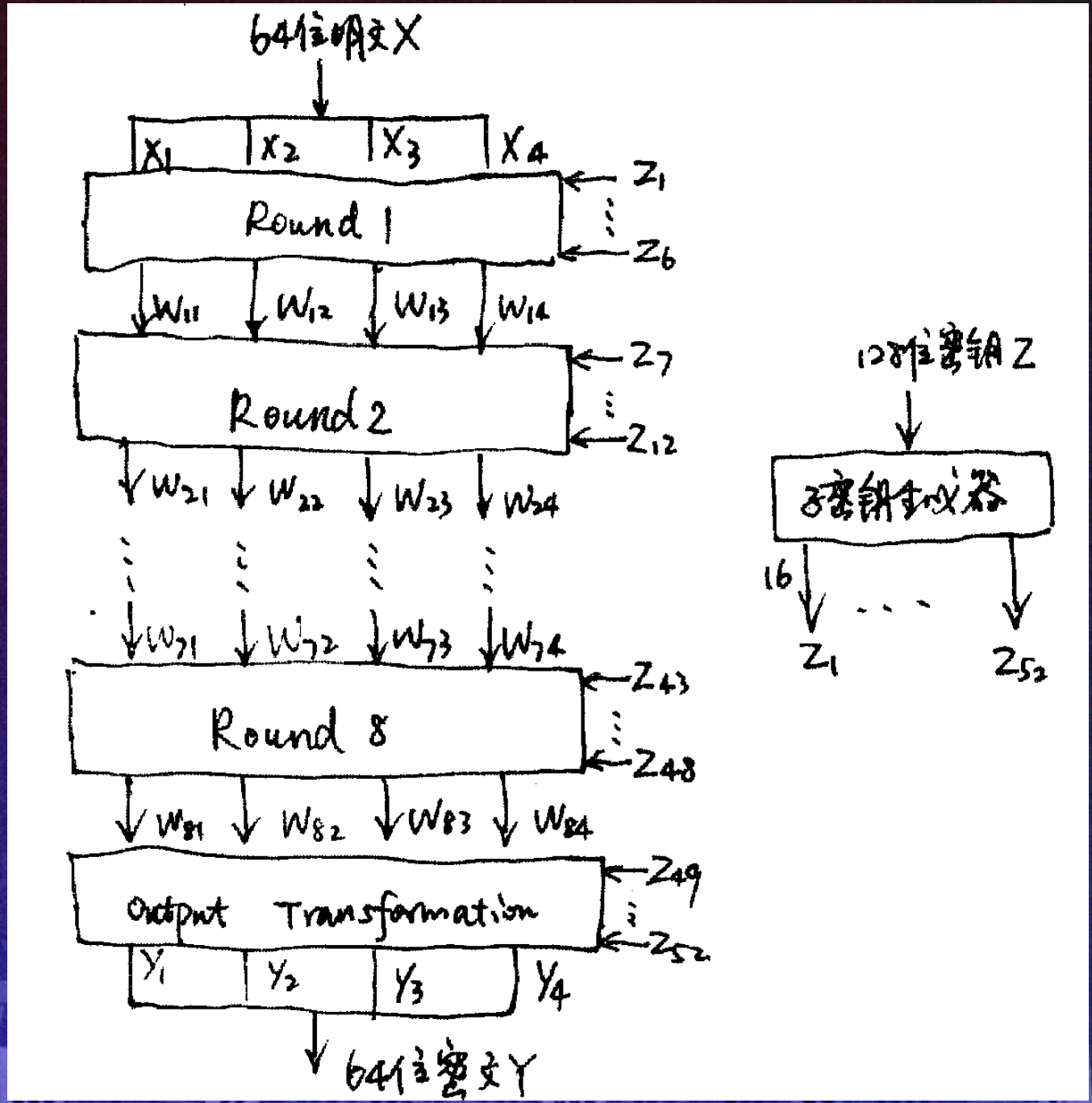
- 90年首次出现，91年修订，92公布细节
- 有望取代DES
- 128位密钥，64位分组
- 设计目标可从两个方面考虑
  - 加密强度
  - 易实现性

IDEA设计思想

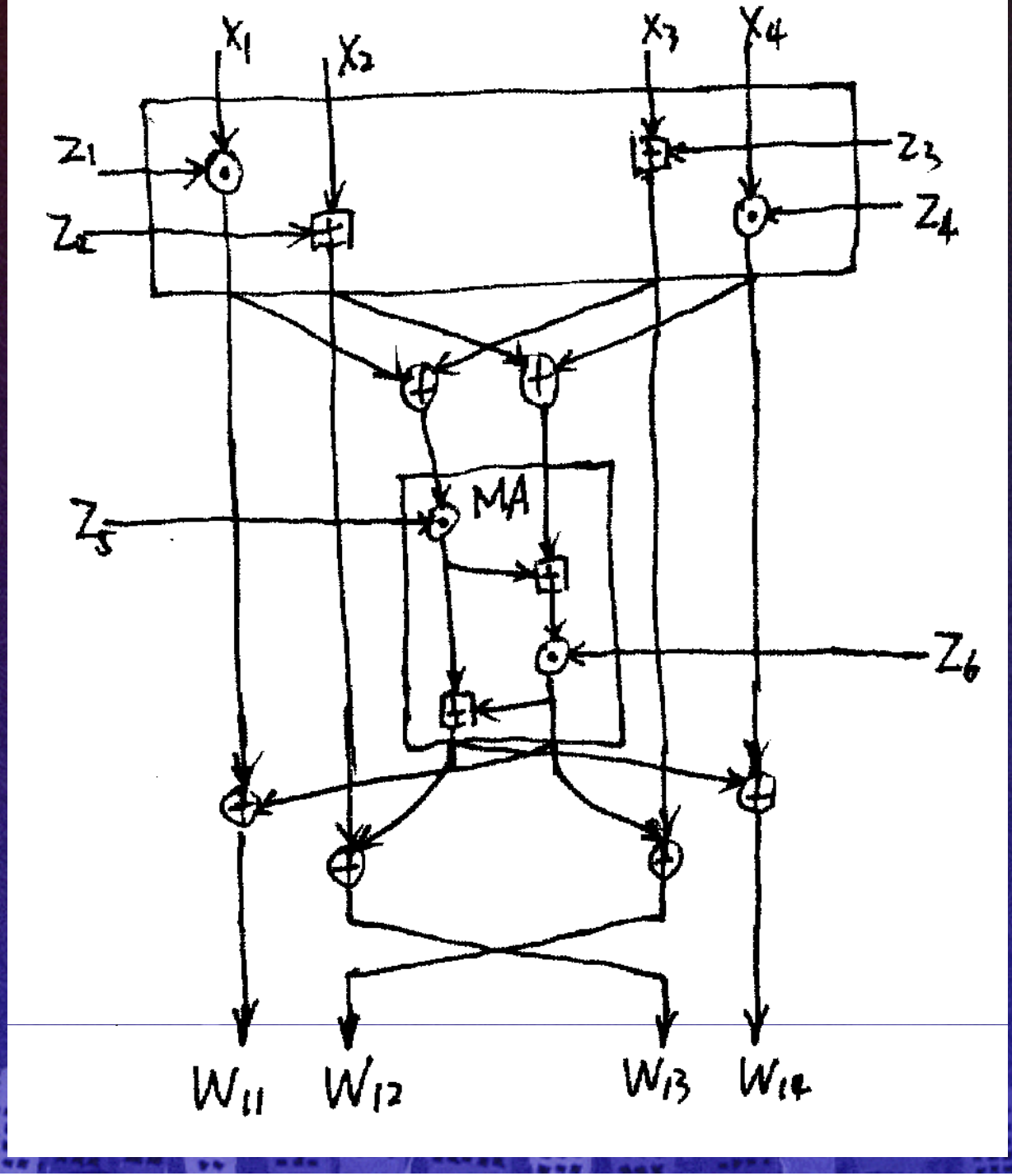
- 得到confusion的途径
  - 按位异或
  - 以2^{16}(65536)为模的加法
  - 以$2^{16}+1$ (65537)为模的乘法
  - 互不满足分配律、结合律
- 得到diffusion的途径
  - 乘加(MA)结构
- 实现上的考虑
  - 软件和硬件实现上的考虑



IDEA
加密算法


64位明文X
X1
X2
X3
X4
Round 1
Z1
Z6
W11
W12
W13
W14
Round 2
Z7
Z12
W21
W22
W23
W24
W71
W72
W73
W74
Round 8
Z43
Z48
W81
W82
W83
W84
Output Transformation
Z49
Z52
Y1
Y2
Y3
Y4
64位密文Y
128位密钥Z
子密钥生成器
16
Z1
Z52

IDEA
每一轮



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算法讨论

- 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

RC5加密算法

- 作者为Ron Rivest
- 算法特点
  - 三个参数
    - 参数w：表示字长，RC5加密两字长分组，可用值为16、32、64
    - 参数r：表示轮数，可用值0,1,…,255
    - 参数b：表示密钥K的字节数，可用值0,1,…,255
  - RC5版本：RC5-w/r/b
  - 算法作者建议标定版本为RC5-32/12/16

RC5加密算法

- 三个基本运算
  - 字的加法，模2^w $+$
  - 按位异或 $\oplus$
  - 左循环移位 $<<<$

- 算法：

$LE_0 = A + S[0]$

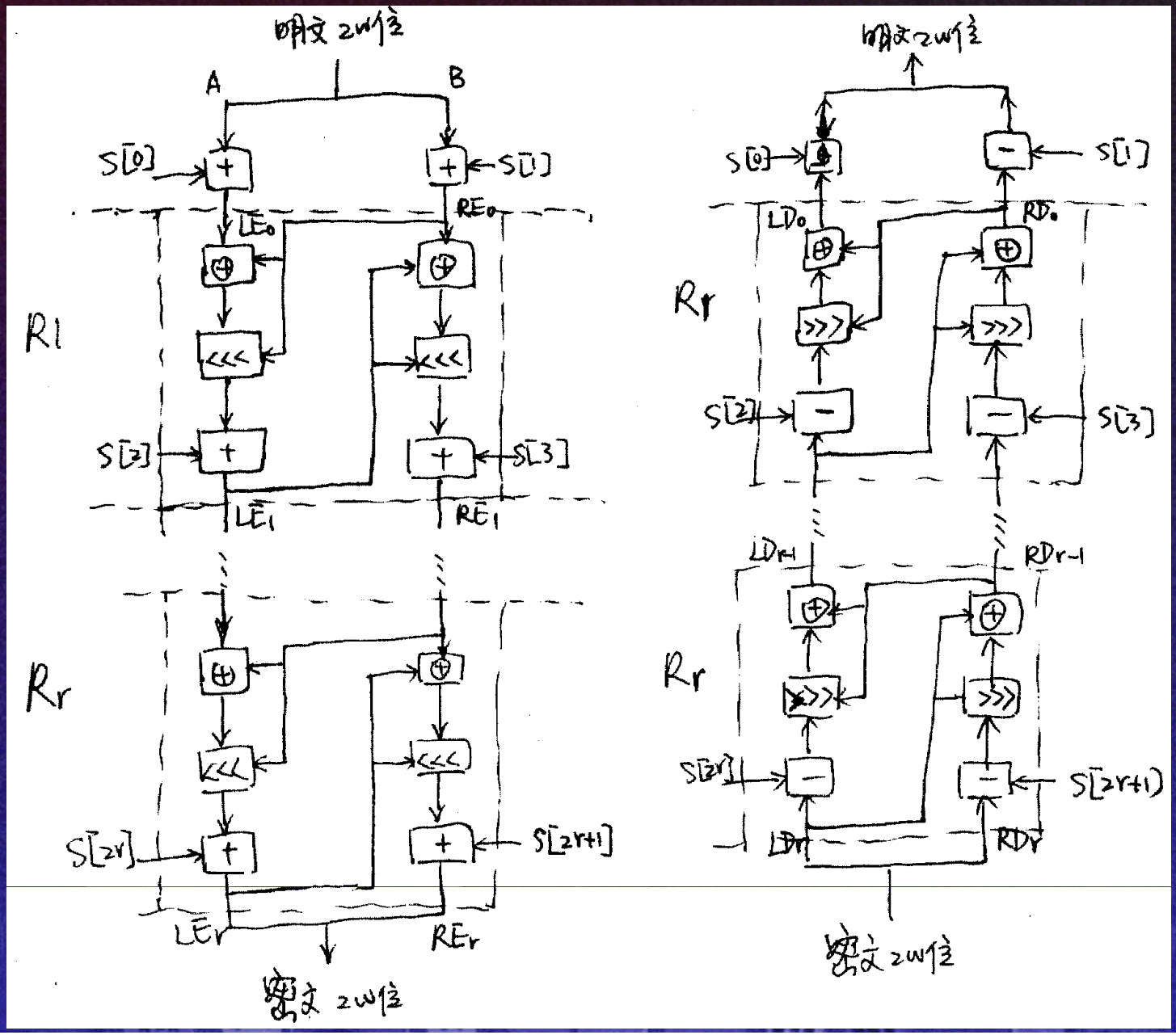
$RE_0 = B + S[1]$

for i = 1 to r do

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

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

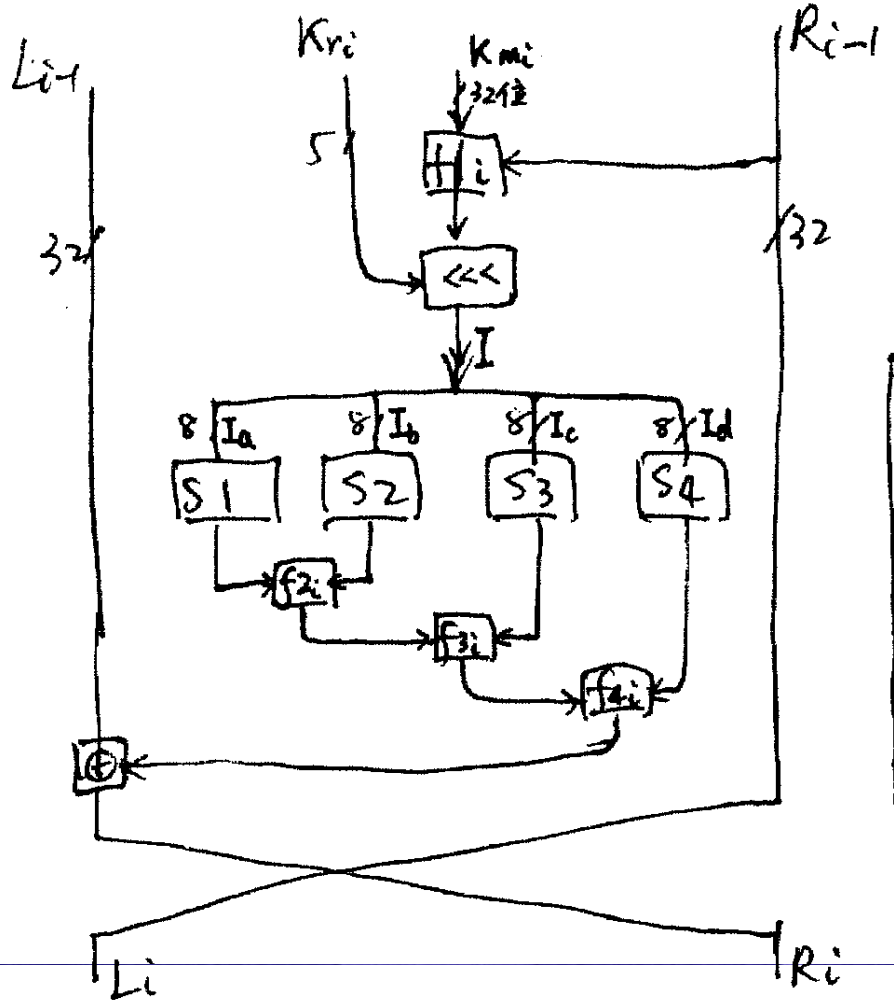
RC5加、解密算法结构


明文 2w位
A
B
S[0]
S[1]
LE0
RE0
R1
S[2]
S[3]
LE1
RE1
Rr
S[2r]
S[2r+1]
LEr
REr
密文 2w位
明文 2w位
S[0]
S[1]
LD0
RD0
Rr
S[2]
S[3]
LDr+1
RDr-1
S[2r]
S[2r+1]
LDr
RDr
密文 2w位

CAST-128加密算法

- RFC 2144[97]定义
- 密钥48-128位，8位增量
- 16轮Feistel分组结构
- 64位分组
- 特殊处：
  - 每一步两个子密钥
  - 每一步的F不同

CAST-128每步细节图



这里f1i, f2i, f3i, f4i 4个函数与轮数有关，分别为 +, −, ⊕ 各种类型，F函数如下表所示：

轮数	I 和 F
1,4,7,10,13,16	$I=((Km_i+R_{i-1})\lll Kr_i)$ $F=((S_1[I_a]\oplus S_2[I_b])-S_3[I_c])+S_4[I_d]$
2,5,8,11,14	$I=((Km_i\oplus R_{i-1})\lll Kr_i)$ $F=((S_1[I_a]-S_2[I_b])+S_3[I_c])\oplus S_4[I_d]$
3,6,9,12,15	$I=((Km_i-R_{i-1})\lll Kr_i)$ $F=((S_1[I_a]+S_2[I_b])\oplus S_3[I_c])-S_4[I_d]$

CAST-128算法之讨论

- S-Box是固定的，但设计时尽量保证了非线性。设计者认为，选择一个好的非线性S-BOX比随机的S-BOX更可取
- 子密钥的产生过程采用了与其他算法不同的产生法来加强其强度。目标是对抗已知明文攻击。Blowfish和RC5算法使用了不同的技术来保证这一点。
- F函数具有好的confusion、diffusion等特性。子密钥相关、轮数相关增加了强度。

RC2加密算法

- 设计者Ron Rivest
- 分组长度64位，密钥长度8到1024位
- 适合于在16位微处理器上实现
- RC2在S/MIME中用到的密钥为40、64、128位不等

RC4流加密算法

- 设计者Ron Rivest
- 工作方式OFB
- 算法特点：
  - 简单、快速
  - 随机序列的产生，用到8*8的S盒

AES介绍

- 1997年NIST宣布征集AES算法
  - 要求: 与三重DES比，要快且至少一样安全, 分组128位,密钥128/192/256位
- 1998年确定第一轮15个候选者
- 1999年确定第二轮五个候选者: MARS, RC6，Rijndael, Serpent, Twofish
- 2000年底Rijndael胜出

Rijndael简介

- 不属于Feistel结构
- 加密、解密相似但不对称
- 支持128/192/256(/32=Nb)数据块大小
- 支持128/192/256(/32=Nk)密钥长度
- 有较好的数学理论作为基础
- 结构简单、速度快

Rijndael简介(续)

- 数据/密钥的矩阵表示

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

- 轮数

Nr	Nb=4	Nb=6	Nb=8
Nk=4	10	12	14
Nk=6	12	12	14
Nk=8	14	14	14

Rijndael算法结构

- 假设：State表示数据，以及每一轮的中间结果
  RoundKey表示每一轮对应的子密钥
- 算法如下：
  - 第一轮之前执行AddRoundKey(State,RoundKey)
  - Round(State,RoundKey) {
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State,RoundKey);
    }
  - FinalRound(State,RoundKey) {
    ByteSub(State);
    ShiftRow(State);
    AddRoundKey(State,RoundKey);
    }

Rijndael: AddRoundKey操作

- 按字节在GF(2^8)上相加(XOR)

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

$\oplus$

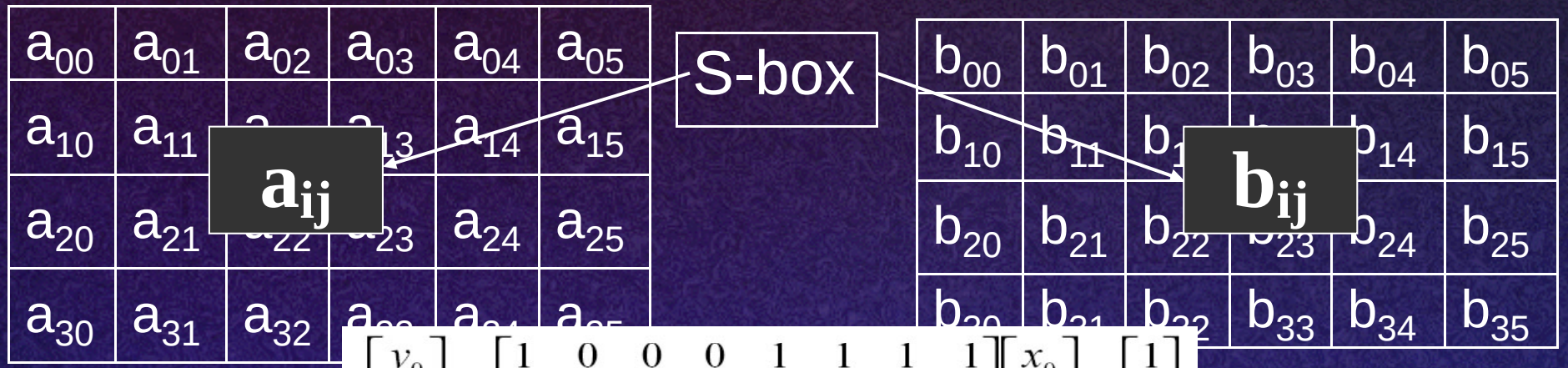
k_{00}	k_{01}	k_{02}	k_{03}	k_{04}	k_{05}
k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
k_{20}	k_{21}	k_{22}	k_{23}	k_{24}	k_{25}
k_{30}	k_{31}	k_{32}	k_{33}	k_{34}	k_{35}

$=$

b_{00}	b_{01}	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}

Rijndael: ByteSub操作

- **ByteSub(S-box)**非线性、可逆
- 独立作用在每个字节上
- 先取**GF(2^8)**上乘法的逆,再经过**GF(2)**上一个仿射变换

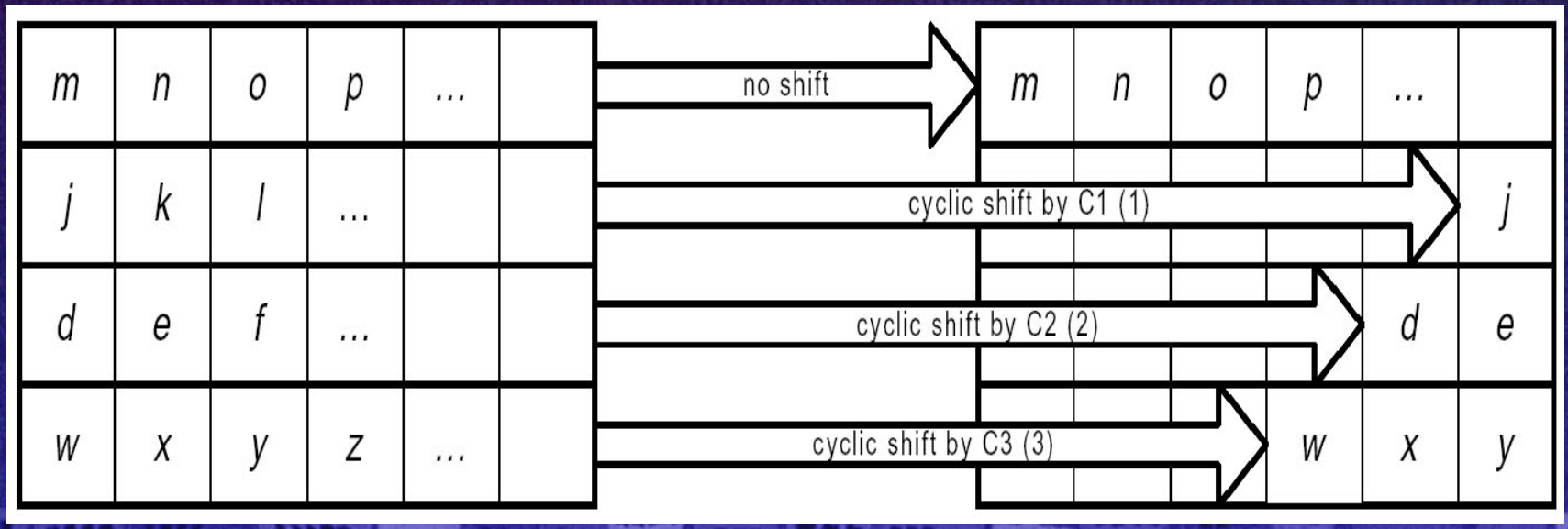


$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Rijndael: ShiftRow操作

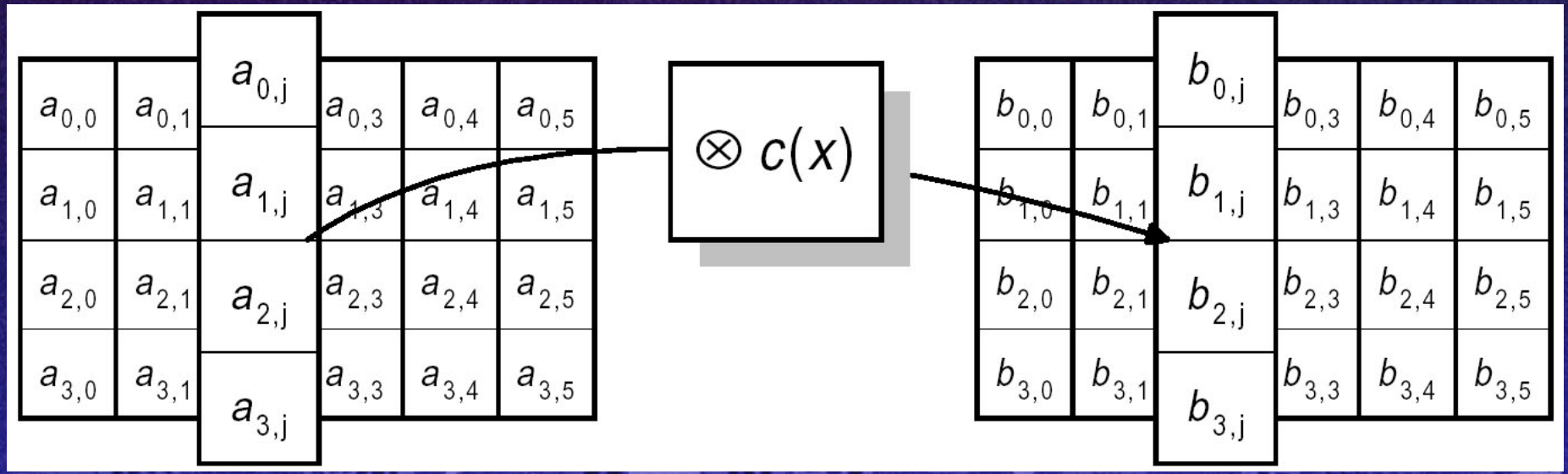
- 第一行保持不变,其他行内的字节循环移位

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4



Rijndael: MixColumn操作

- 列作为GF(2^8)上多项式乘以多项式c(x)
  多项式c(x) = ‘03’x^3+ ‘01’x^2+ ‘01’x+ ‘02’
  $c^{-1}(x)$ = ‘0B’x^3+ ‘0D’x^2+ ‘09’x+ ‘0E’
- 模M(x) = x^4+1



每一轮操作

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

```
Round(State,RoundKey) {
        ByteSub(State);
        ShiftRow(State);
        MixColumn(State);
        AddRoundKey(State,RoundKey);
    }
```

Rijndael: Key schedule(1)

- 主密钥生成子密钥数组, (Nr+1)*Nb个字
- Nk<=6

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)])
{
  for(i=0;i<Nk;i++)
    W[i]=(Key[4*i], Key[4*i|+1], Key[4*i+2], Key[4*i+3]);
  for(i=Nk;i<Nb*(Nr+1);i++) {
    temp=W[i-1];
    if(i%Nk == 0)
      temp=ByteSub(temp<<<8)^Rcon[i/Nk];
    W[i]=W[i-Nk]^temp;
  };
};
```

- Rcon[i]=(x^{i-1}, ‘00’, ‘00’, ‘00’); x^{i-1}为GF(2^8)上的数.

Rijndael: Key schedule(2)

- **Nk>6**

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)])
{
  for(i=0;i<Nk;i++)
    W[i]=(Key[4*i], Key[4*i|+1], Key[4*i+2], Key[4*i+3]);
  for(i=Nk;i<Nb*(Nr+1);i++)
  {
    temp=W[i-1];
    if(i%Nk == 0)
      temp=ByteSub(temp<<<8)^Rcon[i/Nk];
    else if(i%Nk == 4)
      temp=ByteSub(temp<<<8);
    W[i]=W[i-Nk]^temp;
  };
};
```

Rijndael: 加密结构

```
Rijndael(State,CipherKey)
{
  KeyExpansion(CipherKey, ExpandedKey);
  AddRoundKey(State, ExpandedKey)
  For(i=1;i<Nr;++i)
  {
      ByteSub(State);
      ShiftRow(State);
      MixColumn(State);
      AddRoundKey(State,ExpandedKey+Nb*i);
  }
  ByteSub(State);
  ShiftRow(State);
  AddRoundKey(State, ExpandedKey+Nb*i);
}
```

Rijndael: 解密结构

```
AddRoundKey()
For(i=1;i<Nr;++i)
{
ByteSub();
ShiftRow();
MixColumn();
AddRoundKey()
}
ByteSub();
ShiftRow();
AddRoundKey()
```

```
I_AddRoundKey()
I_ShiftRow();
I_ByteSub();
For(i=1;i<Nr;++i)
{
I_AddRoundKey()
I_MixColumn();
I_ShiftRow();
I_ByteSub();
}
I_AddRoundKey()
```

```
I_AddRoundKey()
For(i=1;i<Nr;++i)
{
I_ShiftRow();
I_ByteSub();
I_AddRoundKey()
I_MixColumn();
}
I_ShiftRow();
I_ByteSub();
I_AddRoundKey()
```

Rijndael算法的抵抗攻击能力

- 消除了DES中出现的弱密钥的可能
- 也消除了IDEA中发现的弱密钥
- 能有效抵抗目前已知的攻击算法
  - 线性攻击
  - 差分攻击

随机数产生

- 随机数用途，重要的角色，例如
  - 认证过程中，避免重放攻击
  - 会话密钥
  - **RSA**公钥算法
- 随机数的基本特点
  - 随机性
    - 均匀分布，有大量的测试方案
    - 独立性，难以测试，只能测试足够独立
  - 不可预测性
- 随机选择**(Randomization)**在算法设计中的意义

伪随机数产生器

线性一致法(linear congruential method)

$\{X_n\}$: $X_{n+1} = (aX_n + c) \bmod m$

$m > 0$，通常尽可能大。选：2^{31}.

$0 \leq a, c, X_0 < m$

三条准则：

(1) 全周期。每一个周期中尽可能出现0~m中的所有数.

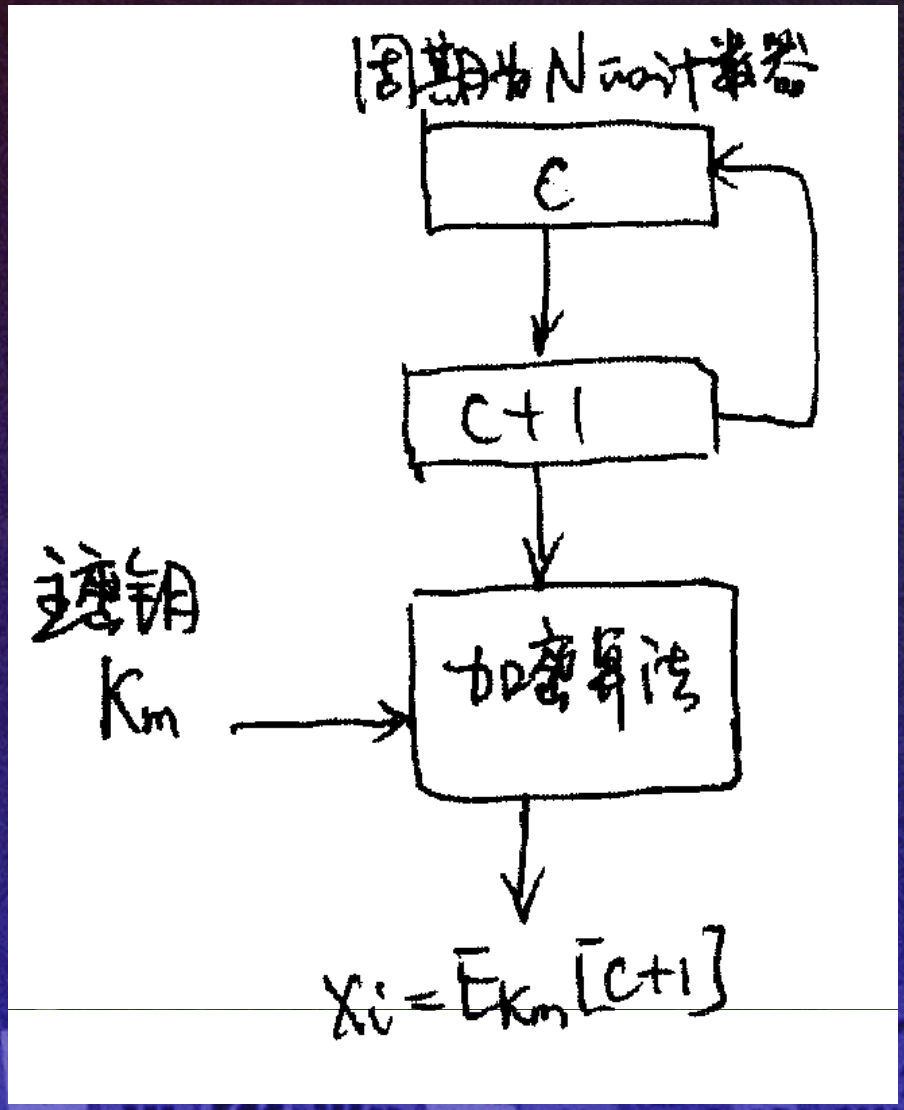
(2) 尽可能表现出随机性。能够通过一些随机性统计测试

(3) 函数能够用32位算术有效地实现出来。

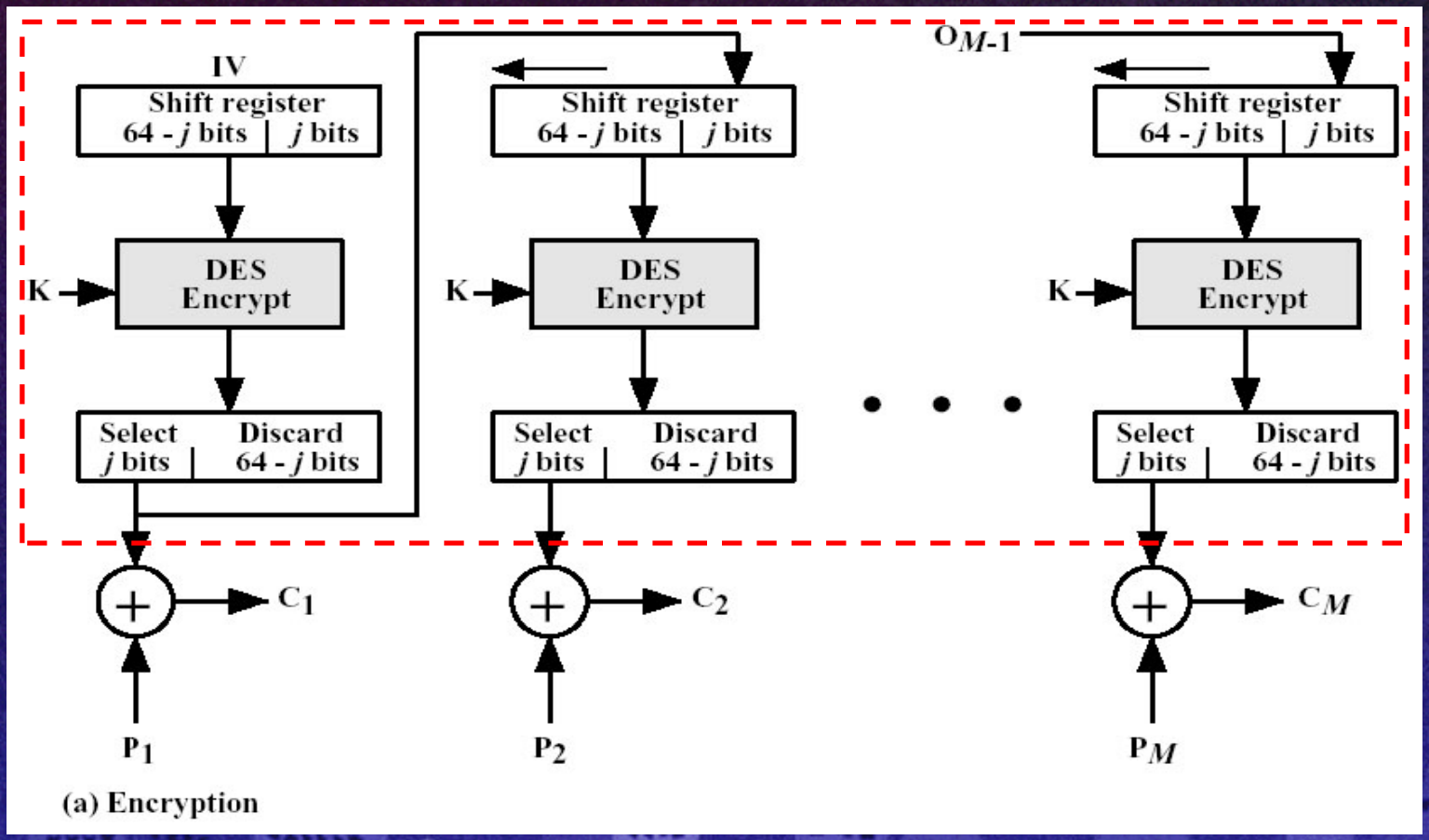
例：$X_{n+1} = (aX_n) \bmod (2^{31}-1)$

$a = 7^5 = 16807$.

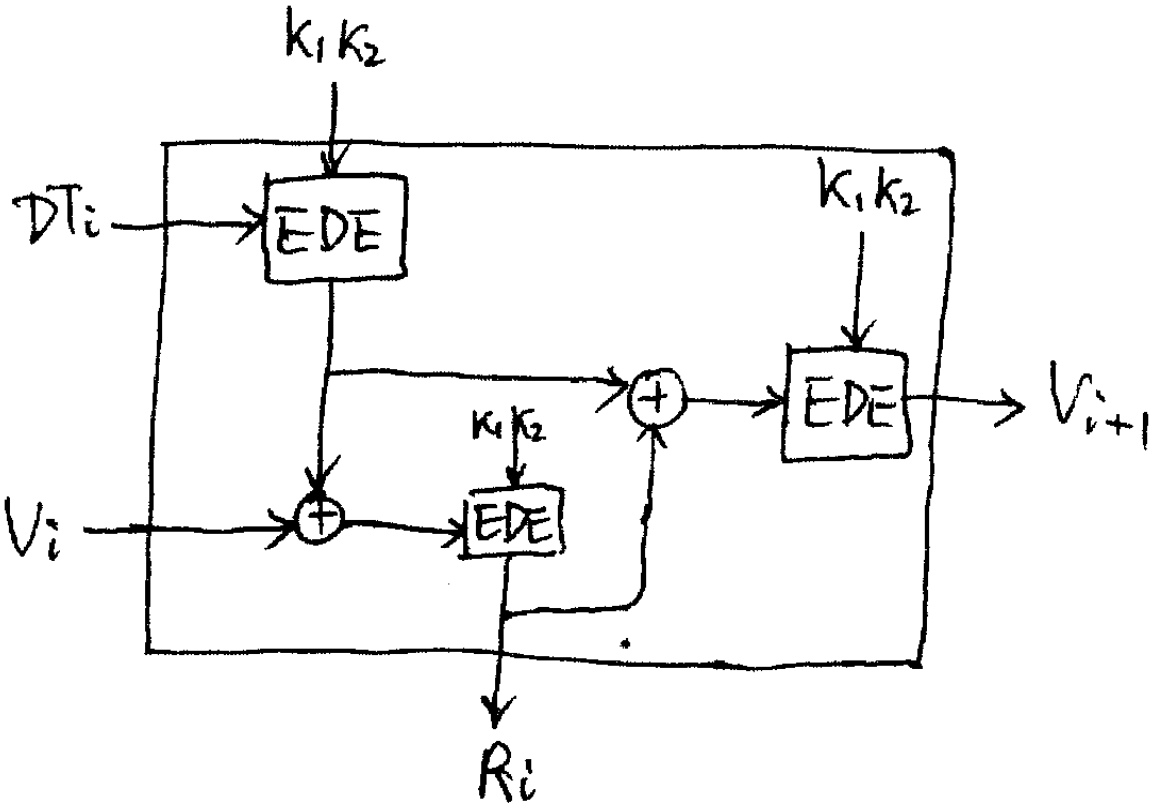
基于密码学产生的随机数(一)
之循环加密法



基于密码学产生的随机数(二)之DES输出反馈模型



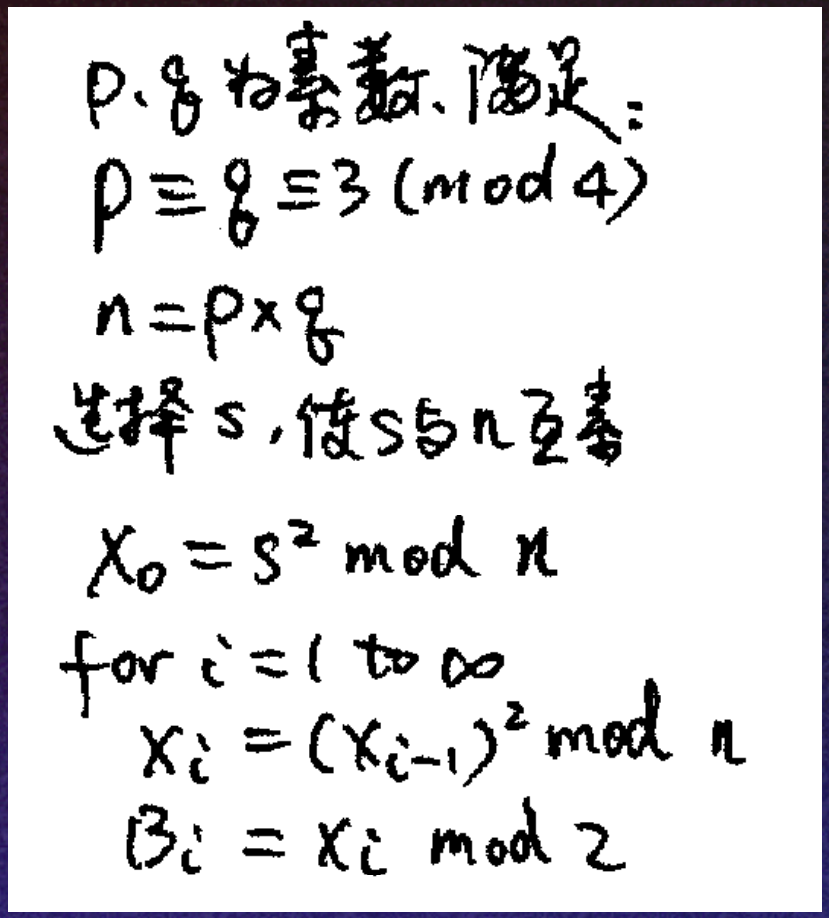
基于密码学产生的随机数(三)之ANSI X9.17



$$R_i = EDE_{K_1,K_2}[V_i \oplus EDE_{K_1,K_2}[DT_i]]$$

$$V_{i+1} = EDE_{K_1,K_2}[R_i \oplus EDE_{K_1,K_2}[DT_i]]$$

BBS伪随机数产生器



- 通过了“下一位”测试(next-bit test)
  - 不存在多项式时间的算法使得在已知前k位的情况下预测出第k+1位的概率大于0.5
- BBS的安全性同样基于分解n的难度