现代密码学

第二章 分组密码

信息与软件工程学院

分组密码概述

- 分组密码是许多系统安全的一个重要组成部分。可用于构造
  - 伪随机数生成器
  - 流密码
  - 消息认证码(MAC)和杂凑函数
  - 消息认证技术、数据完整性机制、实体认证协议以及单钥数字签字体制的核心组成部分。

应用中对于分组码的要求

- 安全性
- 运行速度
- 存储量(程序的长度、数据分组长度、高速缓存大小)
- 实现平台(硬、软件、芯片)
- 运行模式

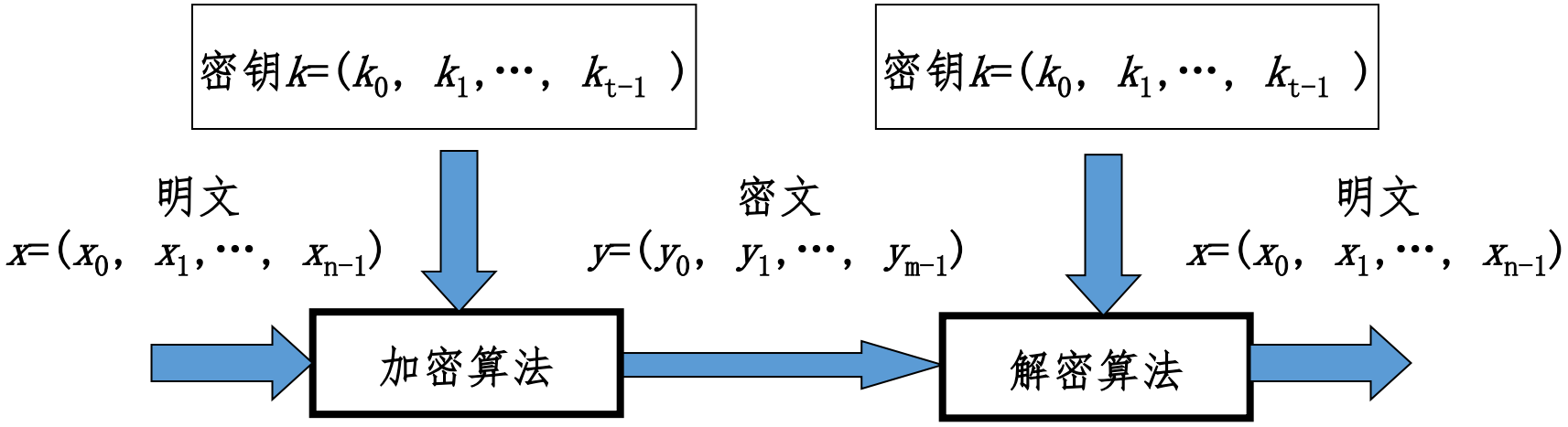
分组密码概述

明文序列 $x_0, x_1, \ldots, x_{n-1}, \ldots$， 分组长度为n

加密函数 E: $V_n \times K \rightarrow V_m$

密文分组长度为m

这种密码实质上是字长为n的数字序列的代换密码。



分组密码概述

- 通常取$n=m$。
- 若$n<m$，则为有数据扩展的分组密码。
- 若$n>m$，则为有数据压缩的分组密码。

分组密码设计问题

分组密码的设计问题在于：找到一种算法，能在密钥控制下从一个足够大且足够好的置换子集中，简单而迅速地选出一个置换，用来对当前输入的明文的数字组进行加密变换。

安全性设计原则

- **1. 混淆原则(Confusion)**
- 混淆原则就是将密文、明文、密钥三者之间的统计关系和代数关系变得尽可能复杂，使得敌手即使获得了密文和明文，也无法求出密钥的任何信息；即使获得了密文和明文的统计规律，也无法求出明文的新的信息。
- 可进一步理解为：
- （1）明文不能由已知的明文，密文及少许密钥比特代数地或统计地表示出来。
- （2）密钥不能由已知的明文，密文及少许密钥比特代数地或统计地表示出来。

安全性设计原则

- **2. 扩散原则(Diffusion)**
- 扩散原则就是应将明文的统计规律和结构规律散射到相当长的一段统计中去(Shannon的原话)。
- 也就是说让明文中的每一位影响密文中的尽可能多的位，或者说让密文中的每一位都受到明文中的尽可能多位的影响。
- 如果当明文变化一个比特时,密文有某些比特不可能发生变化,则这个明文就与那些密文无关,因而在攻击这个明文比比特时就可不利用那些密文比特.

分组密码算法应满足的要求

- 分组长度n要足够大：

  防止明文穷举攻击法奏效。

- 密钥量要足够大：

  尽可能消除弱密钥并使所有密钥同等地好，以防止密钥穷举攻击奏效。

- 由密钥确定置换的算法要足够复杂：

  充分实现明文与密钥的扩散和混淆，没有简单的关系可循，要能抗击各种已知的攻击。

分组密码算法应满足的要求

- 加密和解密运算简单：

  易于软件和硬件高速实现。

- 数据扩展：

  一般无数据扩展，在采用同态置换和随机化加密技术时可引入数据扩展。

- 差错传播尽可能地小。

  一个密文分组的错误尽可能少的影响其他密文分组的解密

分组密码的实现原则

- 软件实现的原则：
  - 使用子块和简单的运算。如将分组n化分为子段，每段长为8、16或32。在以软件实现时，应选用简单的运算，使作用于子段上的密码运算易于以标准处理器的基本运算，如加、乘、移位等实现，避免用以软件难于实现的逐比特置换。
- 硬件实现的原则：
  - 加密解密可用同样的器件来实现。

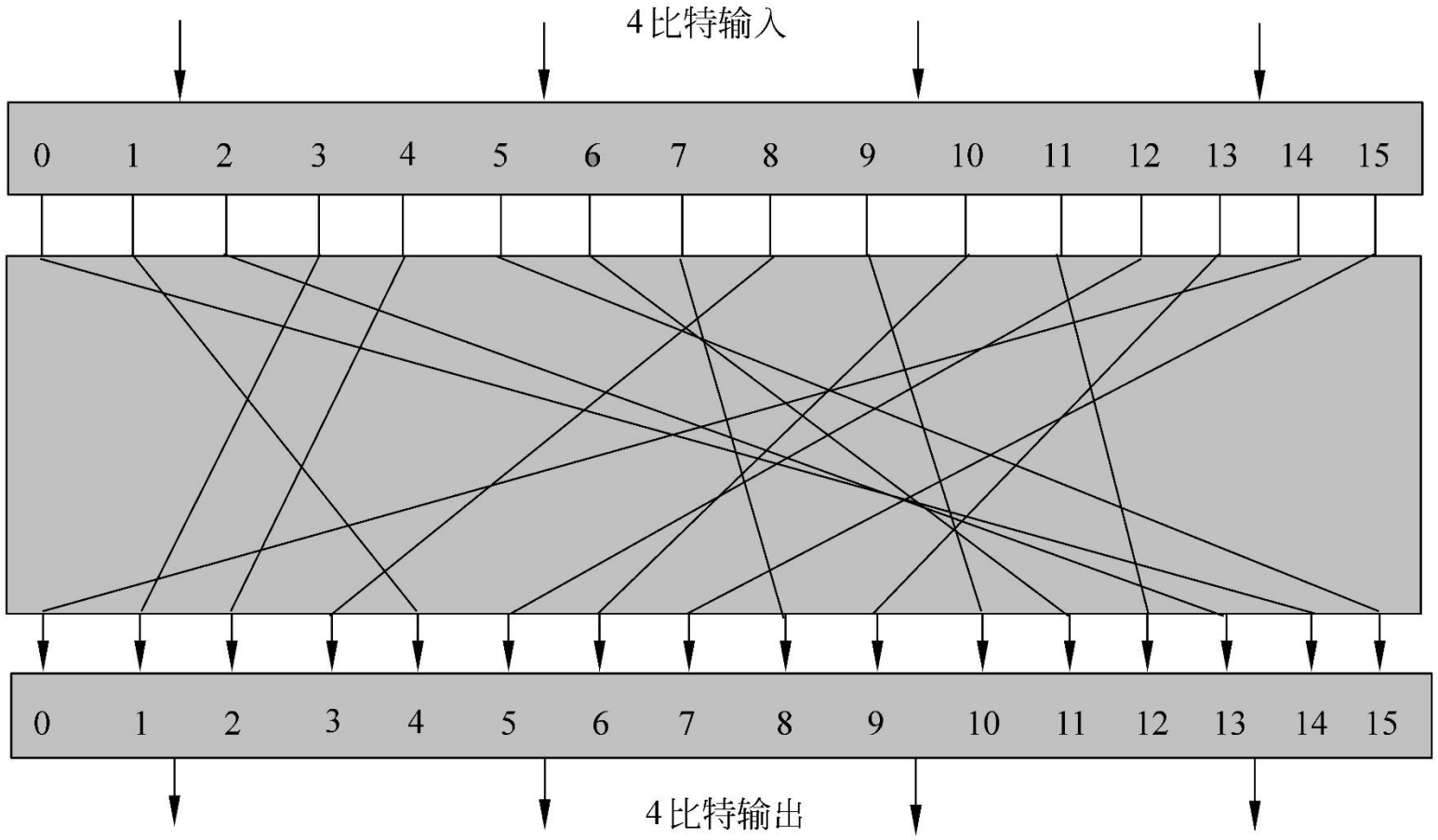
SP网络

代换的定义

•如果明文和密文的分组长都为n比特，则明文的每一个分组都有2^n个可能的取值。为使加密运算可逆（使解密运算可行），明文的每一个分组都应产生惟一的一个密文分组，这样的变换是可逆的，称明文分组到密文分组的可逆变换为代换。

•不同可逆变换的个数有$2^n!$个。

代换结构


4比特输入
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
4比特输出

代换表

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

正向代换

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

反向代换

代换的弱点

- 如果分组长度太小，如n=4，系统则等价于古典的代换密码，容易通过对明文的统计分析而被攻破
  - 这个弱点不是代换结构固有的，只是因为分组长度太小
  - 如果分组长度n足够大，而且从明文到密文可有任意可逆的代换，那么明文的统计特性将被隐藏而使以上的攻击不能奏效。

代换的弱点（续）

- 从实现的角度来看，分组长度很大的可逆代换结构是不实际的
  - 仍以4比特的代换表为例，该表定义了n=4时从明文到密文的一个可逆映射，其中第2列是每个明文分组对应的密文分组的值，可用来定义这个可逆映射。
  - 从本质上来说，第2列是从所有可能映射中决定某一特定映射的密钥
  - 密钥需要64比特
- 一般地，对n比特的代换结构，密钥的大小是$n\times 2^n$比特。如对64比特的分组，密钥大小应是$64\times 2^{64}=2^{70}\approx 10^{21}$比特，因此难以处理。

SP网络

- 在Shannon 1949 的文章中，介绍了代换-置换网络的思想即SP网络
  - 这种思想形成了现代密码的基础
  - SP网络是代换-置换乘积密码的现代形式
- SP网络是基于下列两种最基本的密码运算：
  - 代换（ Substitution ）
  - 置换（ Permutation ）

代换网络

- 代换是输入集A到输出A'上的双射变换：

$$f_k: A \rightarrow A'$$

式中，k是控制输入变量，在密码学中则为密钥。

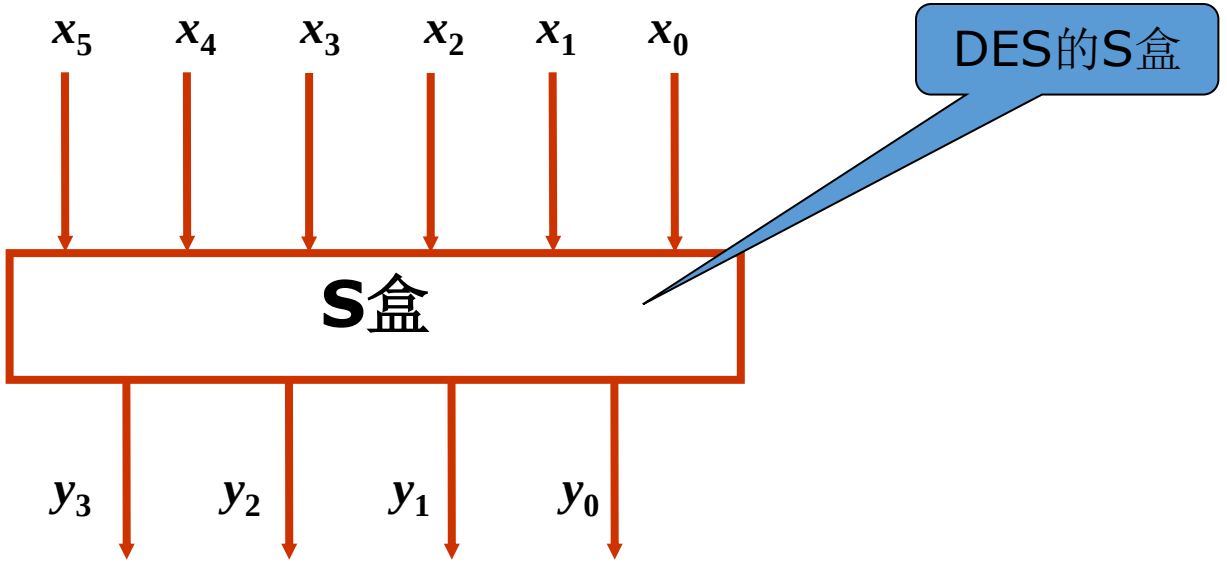
- 实现代换f_k的网络称作代换网络
  - 双射条件保证在给定k下可从密文惟一地恢复出原明文
- 代换f_k的集合：　$S=\{f_k|k\in K\}$
- 如果网络可以实现所有可能的$2^n!$个代换，则称其为全代换网络
  - 全代换网络密钥个数必须满足条件：$\#\{k\}\geq 2^n!$

代换网络

- 密码设计中需要先定义代换集S，而后还需定义解密变换集，即逆代换网络S^{-1}，它以密文y作为输入矢量，其输出为恢复的明文矢量x。
- 要实现全代换网络并不容易。因此实用中常常利用一些简单的基本代换，通过组合实现较复杂的、元素个数较多的代换集。
- 实用密码体制的集合S中的元素个数都远小于$2^n!$。

代换盒(S盒)

在密码设计中，可选$n=r\cdot n_0$，其中r和n_0都为正整数，将设计n个变量的代换网络化为设计r个较小的子代换网络，而每个子代换网络只有n_0个输入变量。称每个子代换网络为代换盒(Substitution Box)



S盒的设计准则

迄今为止，有关方面未曾完全公开有关DES的S盒的设计准则。Branstead等曾披露过下述准则：

- P1 S盒的输出都不是其输入的线性或仿射函数。
- P2 改变S盒的一个输入比特，其输出至少有两比特产生变化。
- P3 当S盒的任一输入位保持不变，其它5位输入变化时(共有2^5 =32种情况)，输出数字中的0和1的总数近于相等。

这三点使DES的S盒能够实现较好的混淆。

Feistel密码结构

Feistel密码的思想

- **乘积密码**指顺序地执行两个或多个基本密码系统，使得最后结果的密码强度高于每个基本密码系统产生的结果.
- Feistel还提出了实现代换和置换的方法。其思想实际上是Shannon提出的**利用乘积密码实现混淆和扩散思想**的具体应用。

Feistel密码实现的参数

Feistel网络的实现与以下参数和特性有关：

① 分组大小：分组越大则安全性越高，但加密速度就越慢。

② 密钥大小：密钥越长则安全性越高，但加密速度就越慢。

③ 轮数：单轮结构远不足以保证安全性，但多轮结构可提供足够的安全性。典型地，轮数取为**16**。

④ 子密钥产生算法：该算法的复杂性越大，则密码分析的困难性就越大。

⑤ 轮函数：轮函数的复杂性越大，密码分析的困难性也越大。

设计Feistel密码的两个要求

在设计**Feistel**网络时，还有以下两个方面需要考虑：

① 快速的软件实现：在很多情况中，算法是被镶嵌在应用程序中，因而无法用硬件实现。此时算法的执行速度是考虑的关键。

② 算法容易分析：如果算法能被无疑义地解释清楚，就可容易地分析算法抵抗攻击的能力，有助于设计高强度的算法。

Feistel加密结构

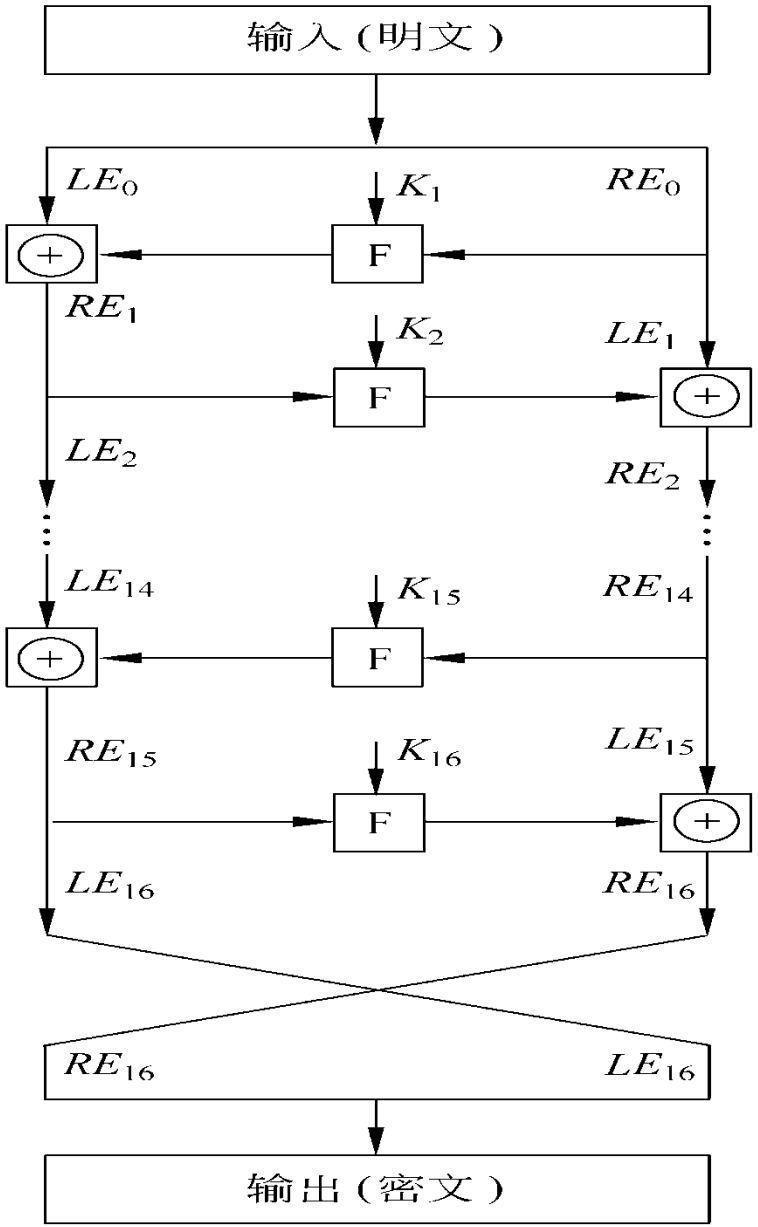
- 输入是分组长为$2w$的明文和一个密钥K。将每组明文分成左右两半L_0和R_0，在进行完n轮迭代后，左右两半再合并到一起以产生密文分组。第i轮迭代的输入为前一轮输出的函数：

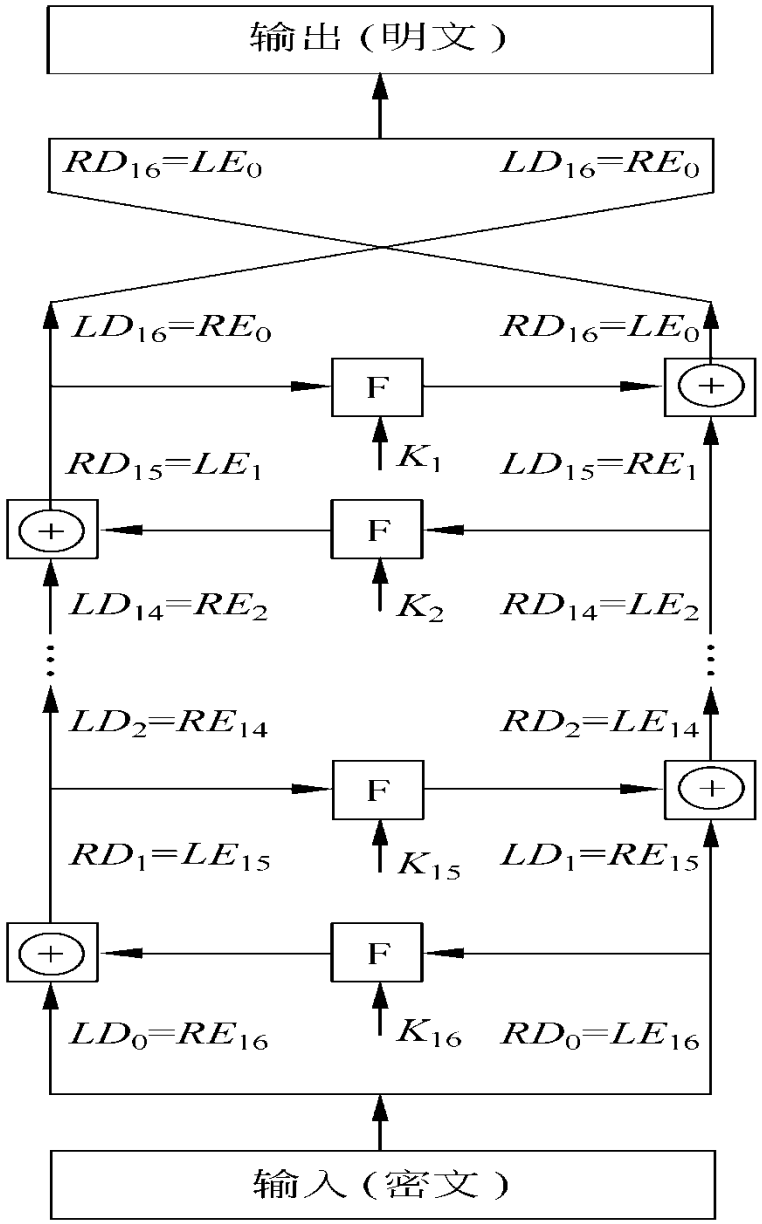
$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

- 其中K_i是第i轮用的子密钥，由加密密钥K得到。一般地，各轮子密钥彼此不同而且与K也不同。

Feistel解密结构

- Feistel解密过程本质上和加密过程是一样的，算法使用密文作为输入
- 但使用子密钥K_i的次序与加密过程相反，即第1轮使用K_n，第2轮使用K_{n-1}，……，最后一轮使用K_1。这一特性保证了解密和加密可采用同一算法。





Feistel加解密过程

Feistel密码解密的正确性

在加密过程中：

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

在解密过程中

$$LD_1 = RD_0 = LE_{16} = RE_{15}$$
$$RD_1 = LD_0 \oplus F(RD_0, K_{16}) = RE_{16} \oplus F(RE_{15}, K_{16})$$
$$= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16})$$
$$= LE_{15}$$

所以解密过程第1轮的输出为$\mathbf{LE_{15} \| RE_{15}}$，等于加密过程第16轮输入左右两半交换后的结果。

Feistel密码解密的正确性（续）

- 容易证明这种对应关系在16轮中每轮都成立。一般地，加密过程的第i轮有

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

因此

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

DES算法简介

美国制定数据加密标准简况

- 目的

通信与计算机相结合是人类步入信息社会的一个阶梯，它始于六十年代末，完成于90年代初。计算机通信网的形成与发展，要求信息作业标准化，安全保密亦不例外。只有标准化，才能真正实现网的安全，才能推广使用加密手段，以便于训练、生产和降低成本。

美国制定数据加密标准简况

- 美国NBS在1973年5月15公布了征求建议。1974年8月27日NBS再次出公告征求建议，对建议方案提出如下要求：

(1)算法必须提供高度的安全性

(2)算法必须有详细的说明,并易于理解

(3)算法的安全性取决于密钥,不依赖于算法

(4)算法适用于所有用户

(5)算法适用于不同应用场合

(6)算法必须高效、经济

(7)算法必须能被证实有效

(8)算法必须是可出口的

美国制定数据加密标准简况

- **IBM公司在1971年完成的LUCIFER密码 (64 bit分组，代换-置换，128 bit密钥)的基础上，改进成为建议的DES体制**
- **1975年3月17日NBS公布了这个算法，并说明要以它作为联邦信息处理标准，征求各方意见。**
- **1977年1月15日建议被批准为联邦标准[FIPS PUB 46]，并设计推出DES芯片。**
- **1981年美国国家标准学会ANSI 将其作为标准，称之为DEA[ANSI X3.92]**
- **1983年国际标准化组织(ISO)采用它作为标准，称作DEA-1**

美国制定数据加密标准简况

- NSA宣布每隔5年重新审议DES是否继续作为联邦标准，1988年（FIPS46-1）、1993年（FIPS46-2），1998年不再重新批准DES为联邦标准。
- 虽然DES已有替代的数据加密标准算法，但它仍是迄今为止得到最广泛应用的一种算法，也是一种最有代表性的分组加密体制。
- 1993年4月，Clinton政府公布了一项建议的加密技术标准，称作密钥托管加密技术标准EES(Escrowed Encryption Standard)。算法属美国政府SECRET密级。

美国制定数据加密标准简况

- DES发展史确定了发展公用标准算法模式，而EES的制定路线与DES的背道而驰。人们怀疑有陷门和政府部门肆意侵犯公民权利。此举遭到广为反对。
- 1995年5月AT&T Bell Lab的M. Blaze博士在PC机上用45分钟时间使SKIPJACK（EES标准中的加密算法）的LEAF协议失败，伪造ID码获得成功。1995年7月美国政府宣布放弃用EES来加密数据，只将它用于语音通信。
- 1997年1月美国国家标准与技术研究所NIST着手进行AES（Advanced Encryption Standard）的研究，成立了标准工作室。2001年Rijndael被批准为AES标准。

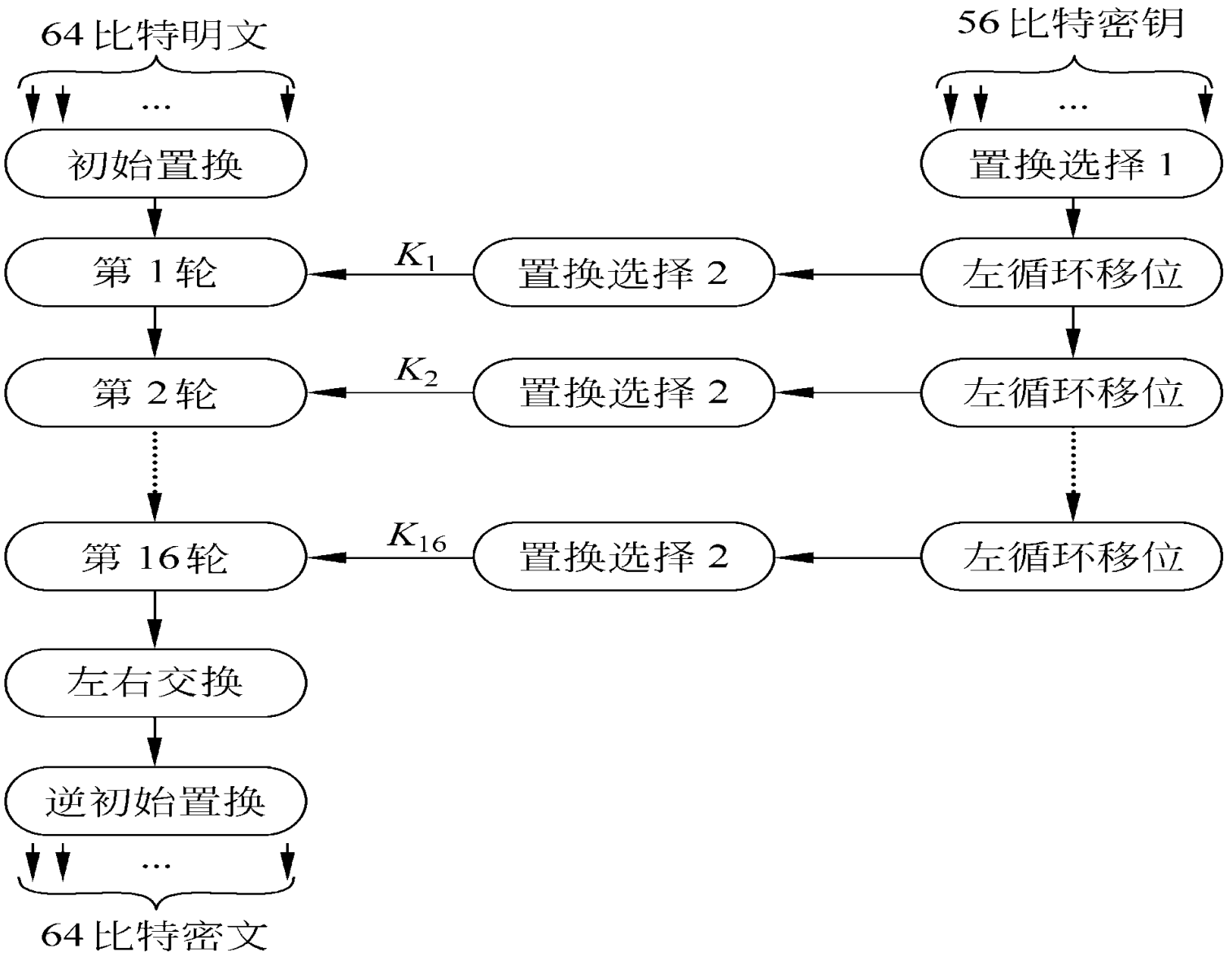
美国制定数据加密标准简况

- **DES**（**Data Encryption Standard**）算法于1977年得到美国政府的正式许可，是一种用56位密钥来加密64位数据的方法。这是IBM的研究成果。
- DES是第一代公开的、完全说明细节的商业级现代算法，并被世界公认。

DES 算法

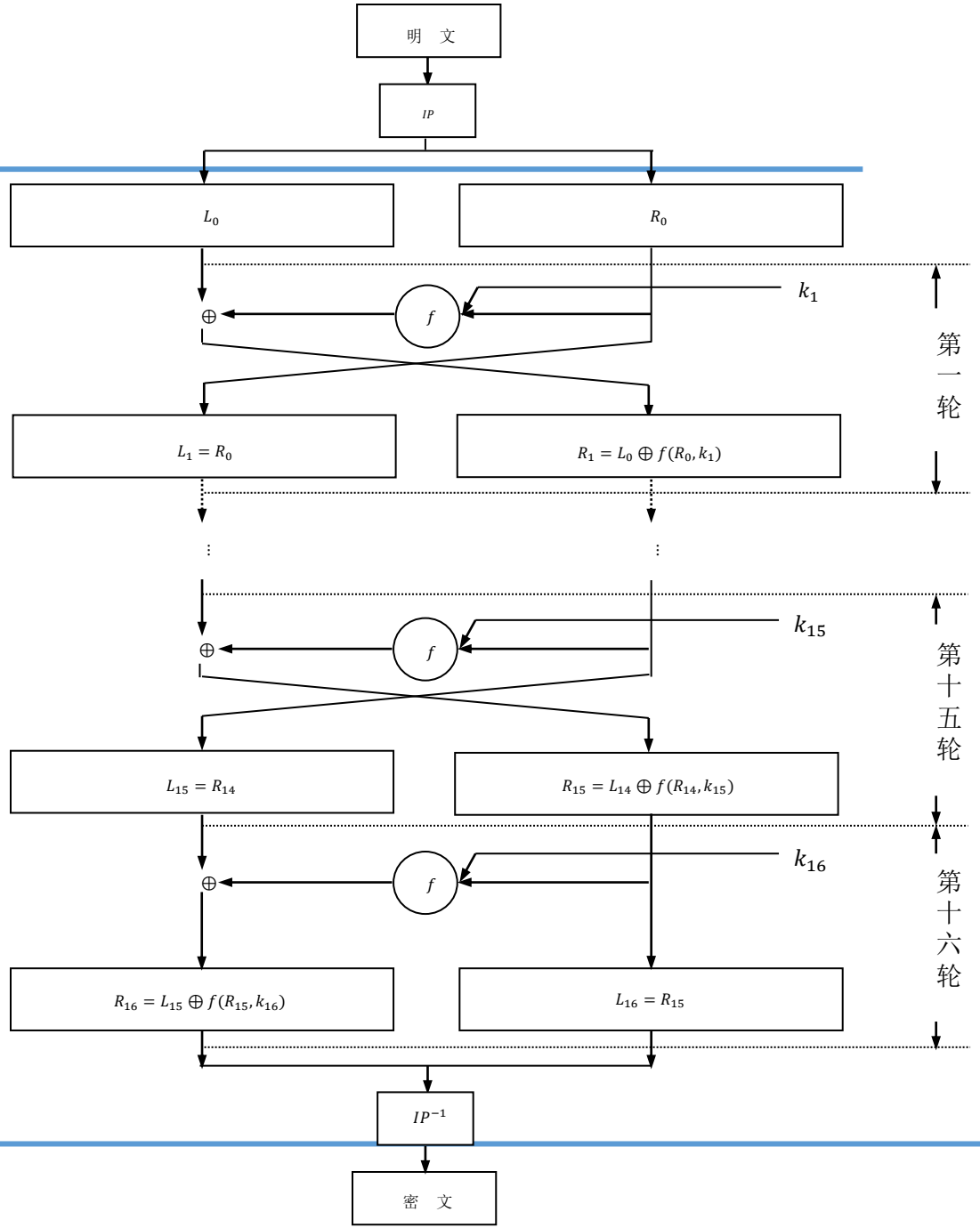
- 分组长度为**64 bits (8 bytes)**
- 密文分组长度也是**64 bits**。
- 密钥长度为**64 bits**，有**8 bits**奇偶校验，有效密钥长度为**56 bits**。
- 算法主要包括：初始置换***IP***、**16**轮迭代的乘积变换、逆初始置换IP^{-1}以及**16**个子密钥产生器。

DES算法框图


64 比特明文
...
初始置换
第 1 轮
第 2 轮
第 16 轮
左右交换
逆初始置换
...
64 比特密文
K_1
K_2
K_{16}
置换选择 2
置换选择 2
置换选择 2
56 比特密钥
...
置换选择 1
左循环移位
左循环移位
左循环移位

DES算法流程

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



初始置换*IP*与逆初始置换

- 初始置换是将**64 bit**明文的位置进行置换，得到一个乱序的**64 bit**明文组。
- 逆初始置换IP^{-1}。将**16**轮迭代后给出的**64 bit**组进行置换，得到输出的密文组。输出为阵中元素按行读得的结果。
- IP和IP^{-1}在密码意义上作用不大，它们的作用在于打乱原来输入x的**ASCII**码字划分的关系。

初值置换IP

(a) 初始置换 IP

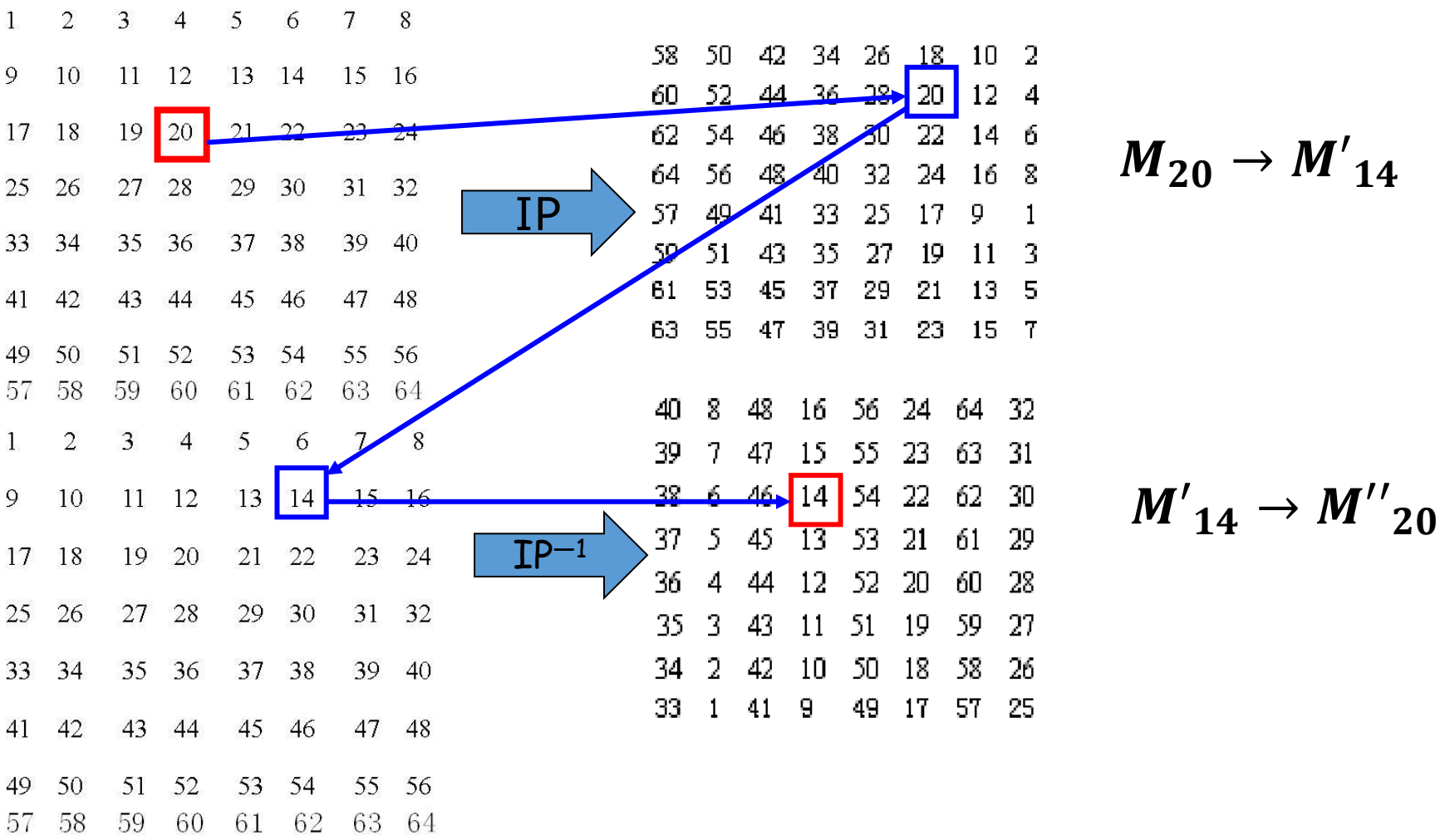
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

逆初值置换IP^{-1}

（b） 逆初始置换 IP^{-1}

	40	8	48	16	56	24	64	32
	39	7	47	15	55	23	63	31
	38	6	46	14	54	22	62	30
	37	5	45	13	53	21	61	29
	36	4	44	12	52	20	60	28
	35	3	43	11	51	19	59	27
	34	2	42	10	50	18	58	26
	33	1	41	9	49	17	57	25

IP与IP^{-1}


1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
IP
58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17 9 1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7
$M_{20} \rightarrow M'_{14}$
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
IP^{-1}
40 8 48 16 56 24 64 32
39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30
37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28
35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26
33 1 41 9 49 17 57 25
$M'_{14} \rightarrow M''_{20}$

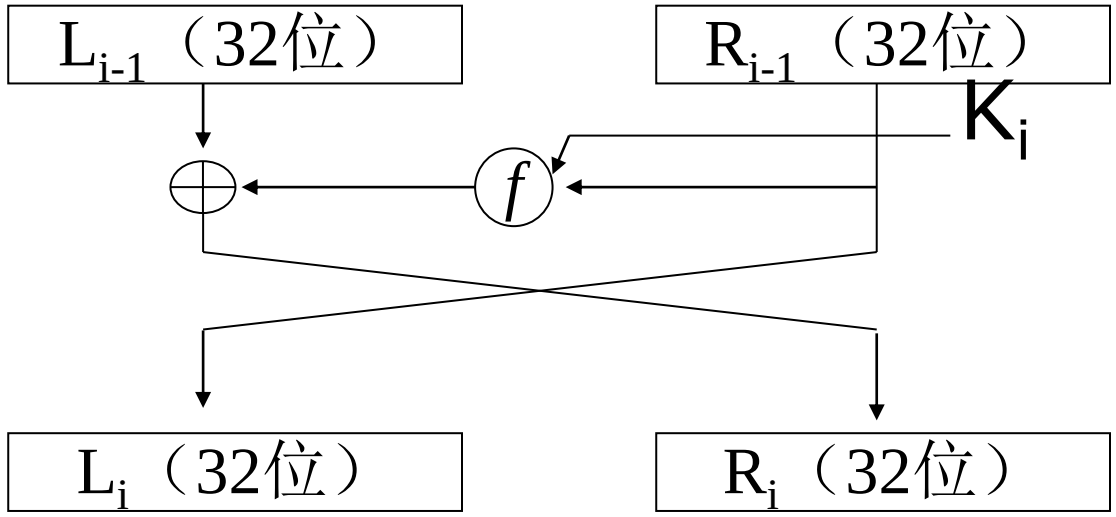
DES的轮函数及密钥编排

DES的轮函数的结构

设输入为（x, y）

则DES的轮函数输出为：

$$(y, x \oplus f_k(y))$$



它等价于两个对合变换的复合：

$$(x, y) \mapsto (x \oplus f(k, y), y) \mapsto (y, x \oplus f(k, y))$$

$$(a, b) \mapsto (b, a)$$

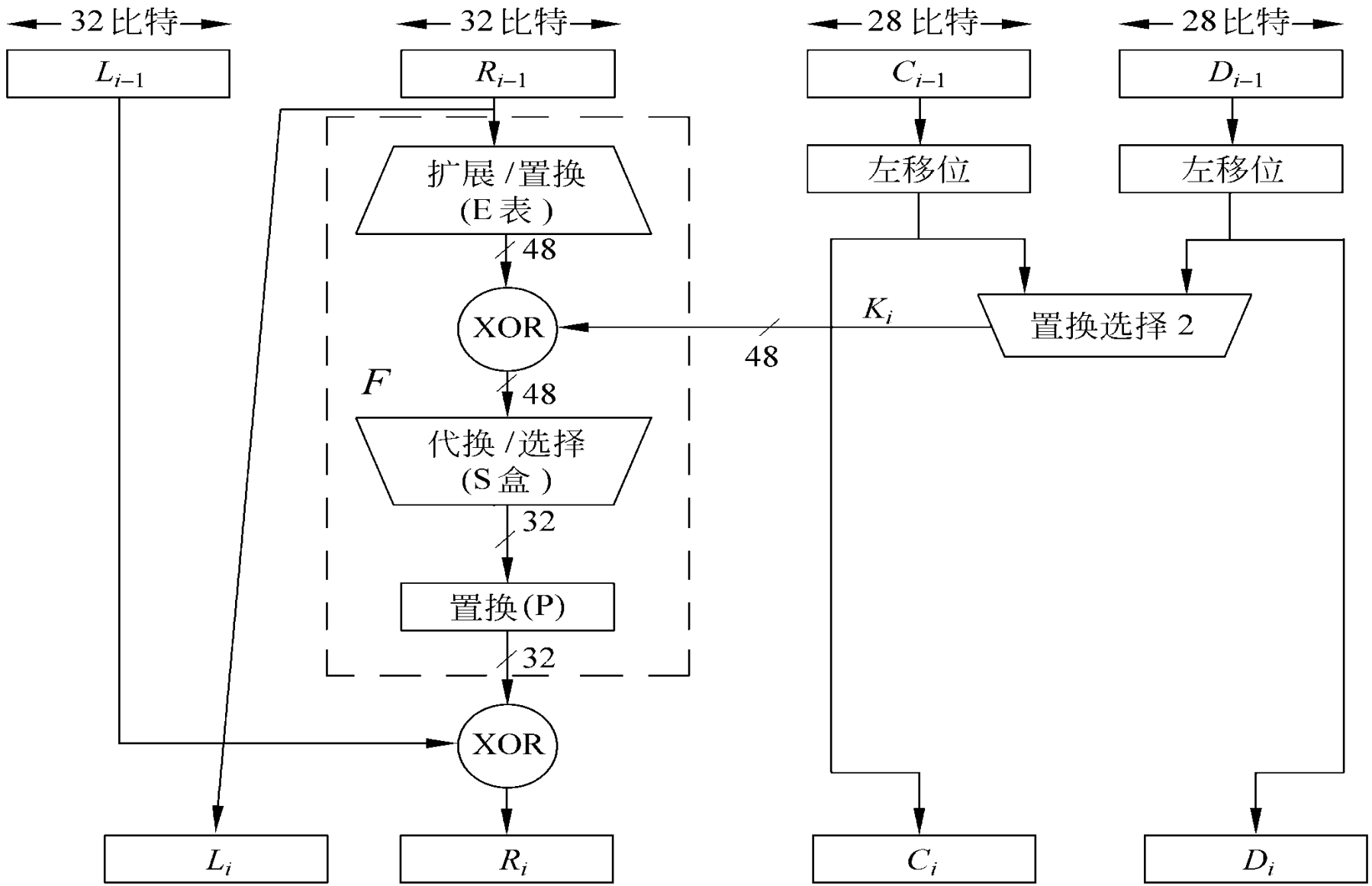
注意

- 无论f函数如何选取，DES的轮函数是对合变换的复合。

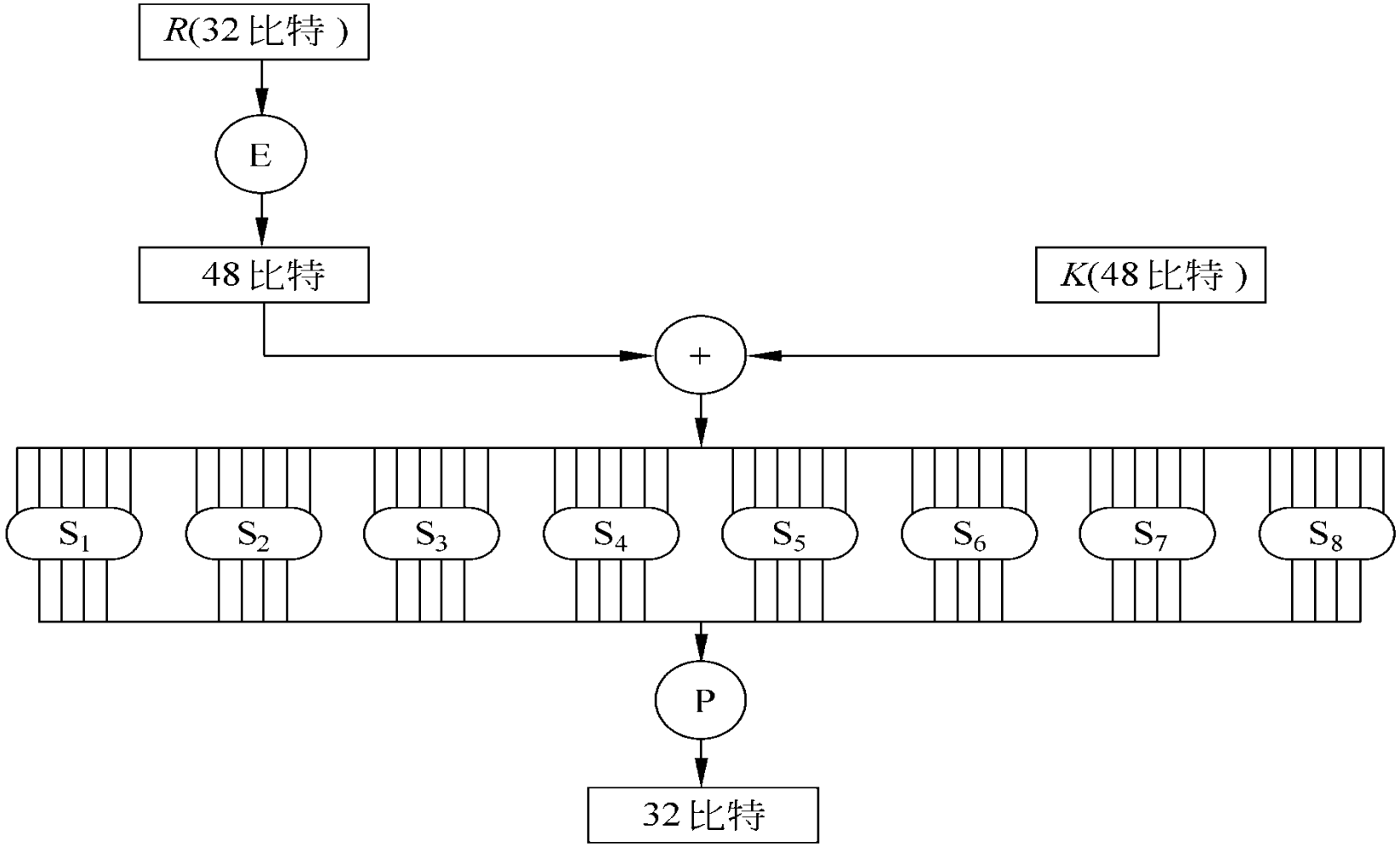
$$F(x,y) = (x \oplus f(k,y), y)$$

$$F(F(x,y)) = F(x \oplus f(k,y), y) = ((x \oplus f(k,y)) \oplus f(k,y), y) = (x,y)$$

DES算法轮结构


32 比特
32 比特
28 比特
28 比特
L_{i-1}
R_{i-1}
C_{i-1}
D_{i-1}
扩展 /置换
(E 表)
左移位
左移位
48
XOR
K_i
48
置换选择 2
F
48
代换 /选择
(S 盒)
32
置换(P)
32
XOR
L_i
R_i
C_i
D_i

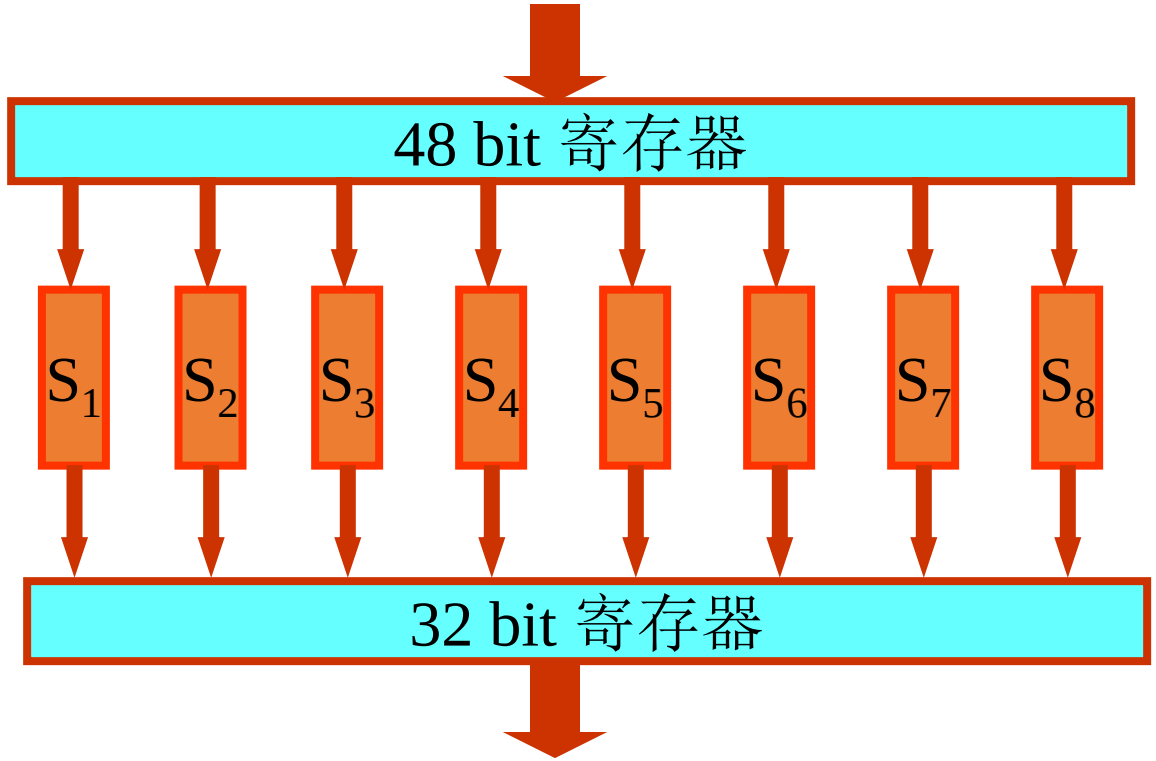
函数f(R,K)的计算过程



选择扩展运算E

32	1	2	3	4	**5**
4	5	6	7	8	**9**
8	9	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	**1**

选择压缩运算S



DES的S盒

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	

DES的S-盒的输入和输出关系

$x_5\ x_0$

1 0

$x_5\ x_4\ x_3\ x_2\ x_1\ x_0$

1 0 1 1 0 0

$(y_3, y_2, y_1, y_0)=(0,0,1,0)$

行号 \ 列号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	2	14	10	0	6	13

P盒置换

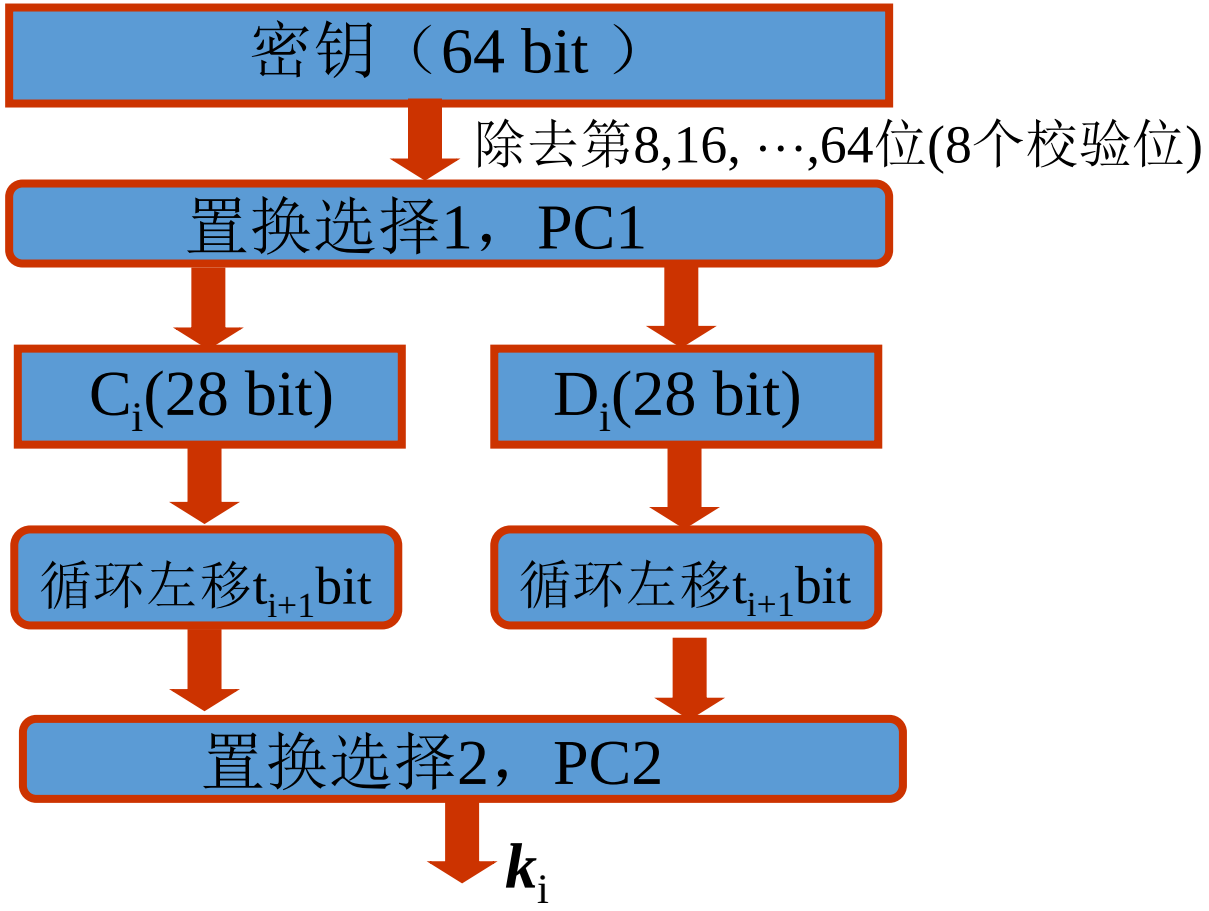
将S-盒变换后的32比特数据再进行P盒置换，置换后得到的32比特即为 f 函数的输出。

- 基本特点：
- （1）P盒的各输出块的4个比特都来自不同的输入块；
- （2） P盒的各输入块的4个比特都分配到不同的输出块之中；
- （3） P盒的第t输出块的4个比特都不来自第t输入块。

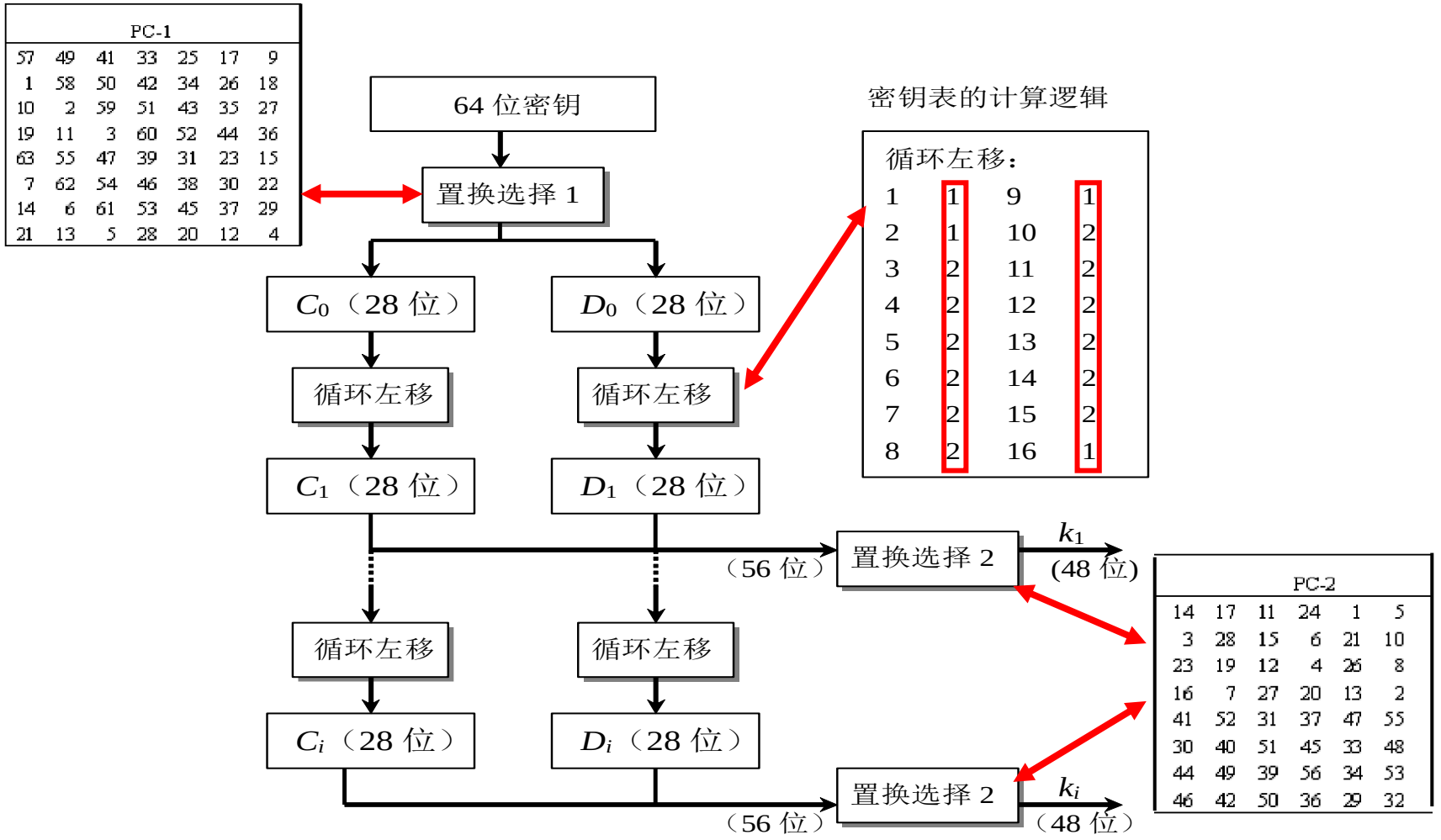
P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

含义:P盒输出的第1个元是输入的第16个元。

DES密钥编排


密钥（64 bit）
除去第8,16, …,64位(8个校验位)
置换选择1，PC1
C_i(28 bit)
D_i(28 bit)
循环左移t_{i+1}bit
循环左移t_{i+1}bit
置换选择2，PC2
k_i

DES中的子密钥的生成



DES算法密钥编排中使用的表

PC-1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

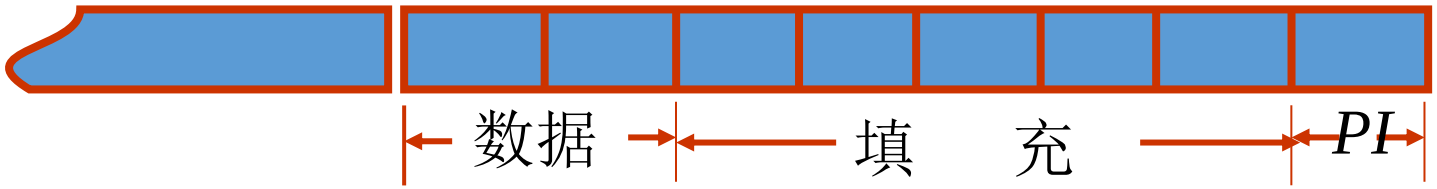
PC-2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

移位次数表

第*i*次迭代	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
循环左移次数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

填充(Padding)

给定加密消息的长度是随机的，按**64 bit**分组时，最后一组消息长度可能不足**64 bit**。可以填充一些数字，通常用最后**1**字节作为填充指示符（***PI***）。它所表示的十进制数字就是填充占有的字节数。数据尾部、填充字符和填充指示符一起作为一组进行加密。



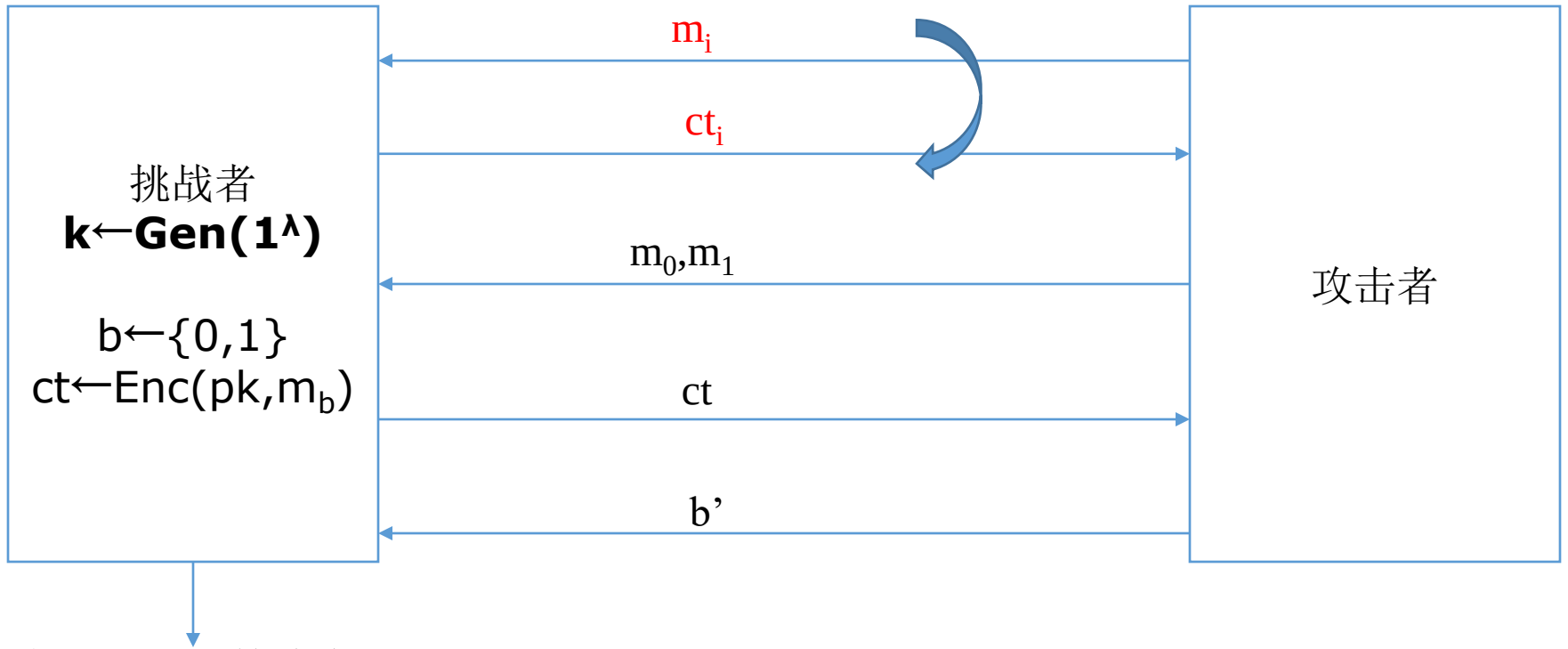
选择题

DES的加密算法是概率性算法还是确定性算法

A. 概率性算法

B. 确定性算法

针对私钥加密的选择明文攻击下的不可区分性（IND-CPA）



DES的安全性

关于DES密钥

- **互补性**。DES算法具有下述性质。若明文组x逐位取补，密钥k逐位取补，即$y=\mathrm{DES}_k(x)$, 则有$\bar{y} = DES_{\bar{K}}(\bar{x})$

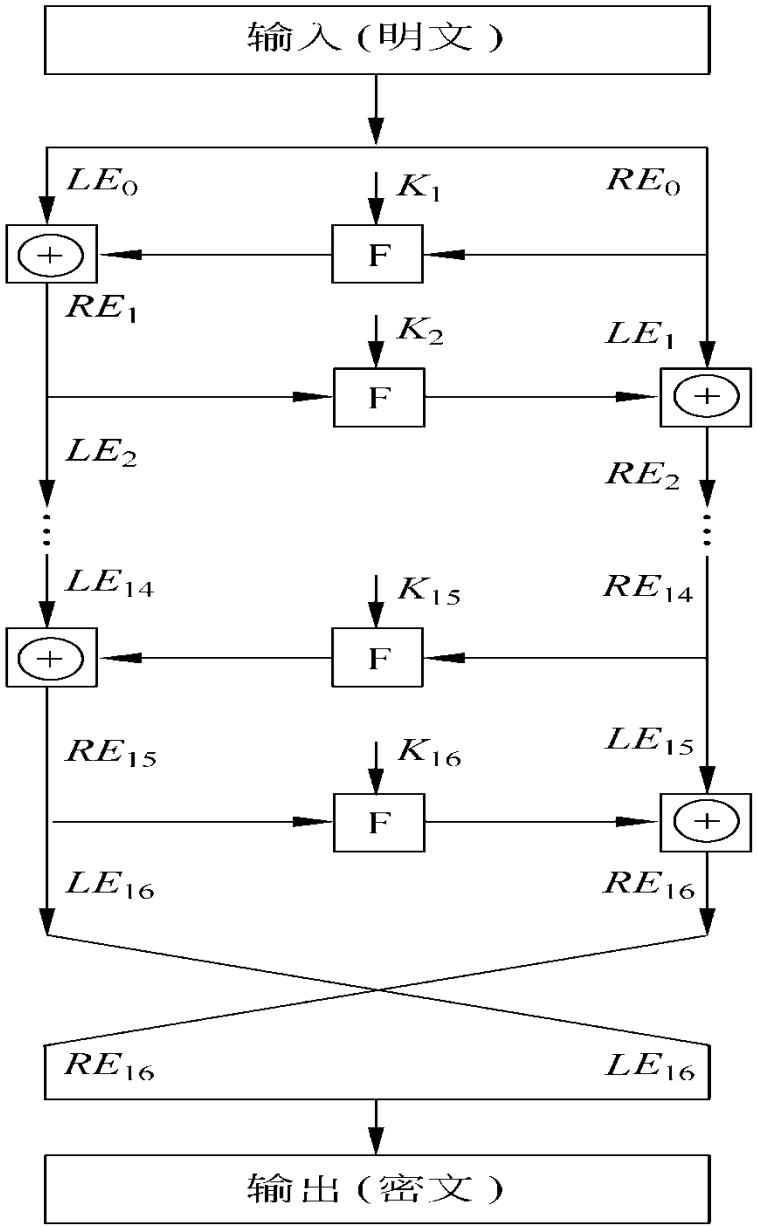
  这种互补性会使DES在选择明文破译下所需的工作量减半。

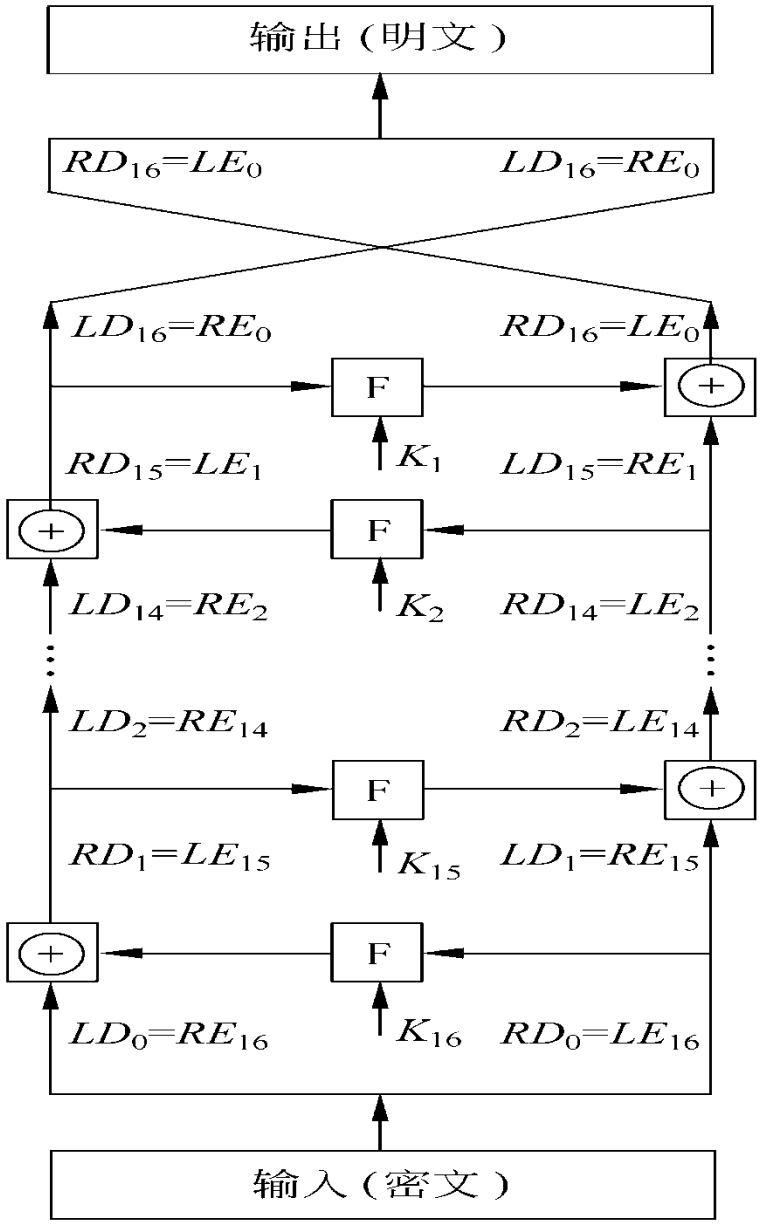
- **弱密钥和半弱密钥**。
  - 弱密钥：$E_K \bullet E_K = I$，DES存在4个弱密钥

    $$\mathrm{DES}_k(\mathrm{DES}_k(x))=x$$

  - 半弱密钥: $E_{K1} = E_{K2}$，至少有12个半弱密钥

    $$y = E_{k_1}(x) = E_{k_2}(x)$$





Feistel加解密过程

DES的弱密钥

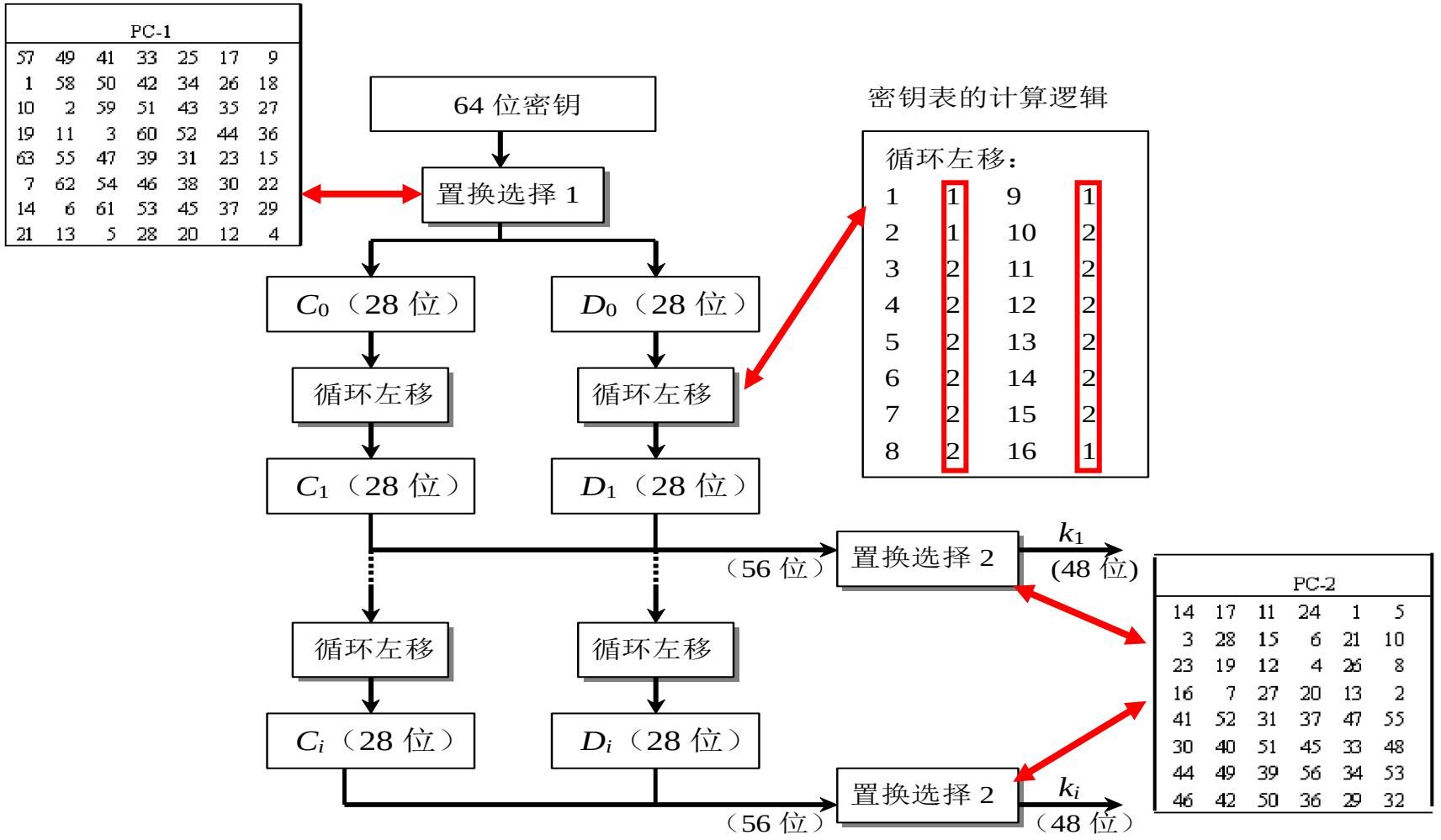
- **DES算法在每次迭代时都有一个子密钥供加密用。如果给定初始密钥k，各轮的子密钥都相同，即有$k_1=k_2=\ \ldots\ =k_{16}$，就称给定密钥$k$为弱密钥(Weak key)。**
- 原始密钥

$$(0,\ \ 0) \rightleftharpoons 01\ 01\ 01\ \ 01\ \ 01\ \ 01\ \ 01\ \ 01$$
$$(0,15) \rightleftharpoons 1F\ \ 1F\ \ 1F\ \ 1F\ 0E\ 0E\ 0E0E$$
$$(0,15) \rightleftharpoons E0\ E0\ E0\ E0\ F1\ \ F1\ \ F1\ F1$$
$$(0,15) \rightleftharpoons FE\ FE\ FE\ FE\ FE\ FE\ FE\ FE$$

置换选择1后的密钥

$$C\quad D$$
$$(0,\ \ 0) \rightleftharpoons 00\ \ \ 00\ \ 00\ \ 00\ \ 00\ \ 00\ \ 00$$
$$(0,15) \rightleftharpoons 00\ \ \ 00\ \ 00\ \ 0F\ FF\ FF\ FF$$
$$(0,15) \rightleftharpoons FF\ FF\ FF\ F0\ \ 00\ \ 00\ \ 00$$
$$(0,15) \rightleftharpoons FF\ FF\ FF\ FF\ FF\ FF\ FF$$

DES中的子密钥的生成



密钥长度的争论

- DES算法正式公开发表以后，引起了一场激烈的争论
- 对DES安全性批评意见中，较为一致的看法是DES的密钥短了些。IBM最初向NBS提交的建议方案采用112 bits密钥，但公布的DES标准采用64 bits密钥。有人认为NSA故意限制DES的密钥长度。
- 采用穷搜索已经对DES构成了威胁。
- 1977年Diffie和Hellman提出了制造一个每秒能测试106个密钥的大规模芯片，这种芯片的机器大约一天就可以搜索DES算法的整个密钥空间，制造这样的机器需要两千万美元。

密钥搜索

- 1993年，R.Session和M.Wiener给出了一个非常详细的密钥搜索机器的设计方案
- 基于并行的密钥搜索芯片，此芯片每秒测试5×107个密钥
- 当时这种芯片的造价是10.5美元，5760个这样的芯片组成的系统需要10万美元，这一系统平均1.5天即可找到密钥
- 如果利用10个这样的系统，费用是100万美元，但搜索时间可以降到2.5小时。

密钥搜索与超级计算

- DES的56位短密钥面临的另外一个严峻而现实的问题是：国际互联网Internet的超级计算能力。
- 1997年1月28日，美国的RSA数据安全公司在互联网上开展了一项名为“密钥挑战”的竞赛，悬赏一万美元，破解一段用56位密钥加密的DES密文。

密钥挑战

- 一位名叫Rocke Verser的程序员设计了一个可以通过互联网分段运行的密钥穷举搜索程序，组织实施了一个称为DESHALL的搜索行动，成千上万的志愿者加入到计划中
- 计划实施的第96天，即挑战赛计划公布的第140天，1997年6月17日晚上10点39分，美国盐湖城Inetz公司的职员Michael Sanders成功地找到了密钥
- 在计算机上显示了明文：“The unknown message is: Strong cryptography makes the world a safer place”

DES的破解

- 1998年7月电子前沿基金会（EFF）使用一台25万美元的电脑在56小时内破译了56比特密钥的DES。
- 1999年1月RSA数据安全会议期间，电子前沿基金会用22小时15分钟就宣告破解了一个DES的密钥。

DES的安全性的其他方面

密文与明文、密文与密钥的相关性

Meyer[1978]详细研究了**DES**的输入明文与密文及密钥与密文之间的相关性。表明每个密文比特都是所有明文比特和所有密钥比特的复合函数，并且指出达到这一要求所需的迭代次数至少为**5**。**Konheim[1981]**用χ^2检验证明，迭代**8**次后输出和输入就可认为是不相关的了。

DES的其他攻击方法

目前攻击**DES**的主要方法有时间-空间权衡攻击、差分攻击、线性攻击和相关密钥攻击等方法，在这些攻击方法中，线性攻击方法是最有效的一种方法。

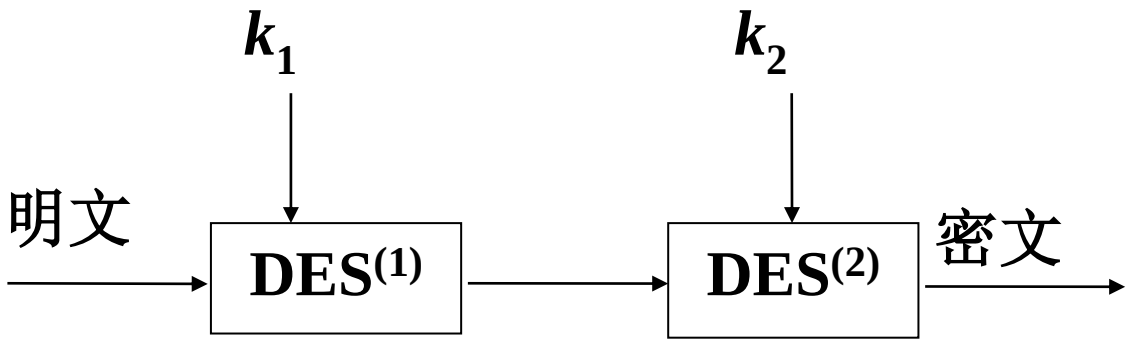
3DES

多重DES

- 如果一个分组密码易受到穷举密钥搜索攻击，那么对同一消息加密多次就有可能增强安全性
- 多重DES就是使用多个密钥利用DES对明文进行多次加密。使用多重DES可以增加密钥量，从而大大提高抵抗穷举密钥搜索攻击的能力
- 多重加密类似于一个有着多个相同密码的级联，但各级密码无需独立，且每级密码既可以是一个分组密码加密函数，也可是相应的解密函数

双重DES

- 简单的对消息x_i利用两个不同的密钥进行两次加密
- 目的是为了抵抗穷搜索攻击，期望密钥长度扩展为112比特



中间相遇攻击

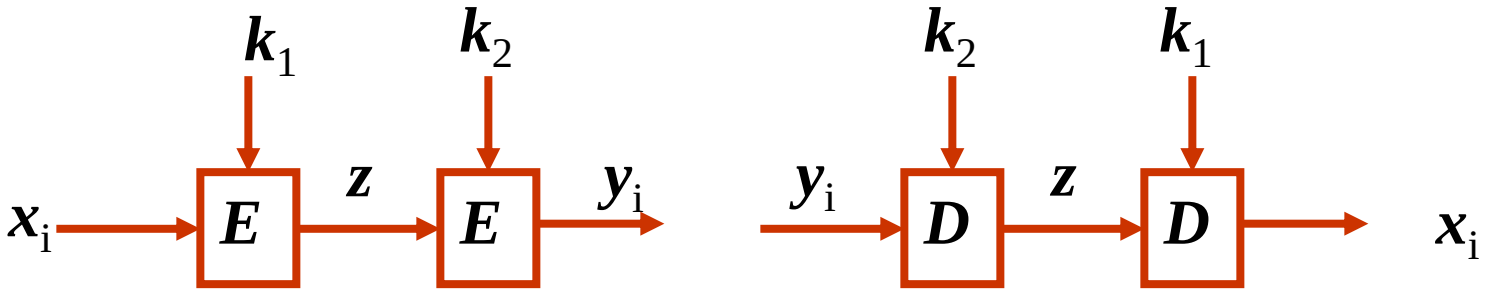
由Diffie和Hellman[1977]最早提出，可以降低搜索量，基本想法如下。

若有明文/密文对(x_i,y_i)满足

$$y_i=E_{k2}[E_{k1}[x_i]]$$

则可得

$$z=E_{k1}[x_i]=D_{k2}[y_i]$$



中间相遇攻击的步骤

给定一已知明密文对(x_1,y_1)，可按下述方法攻击。

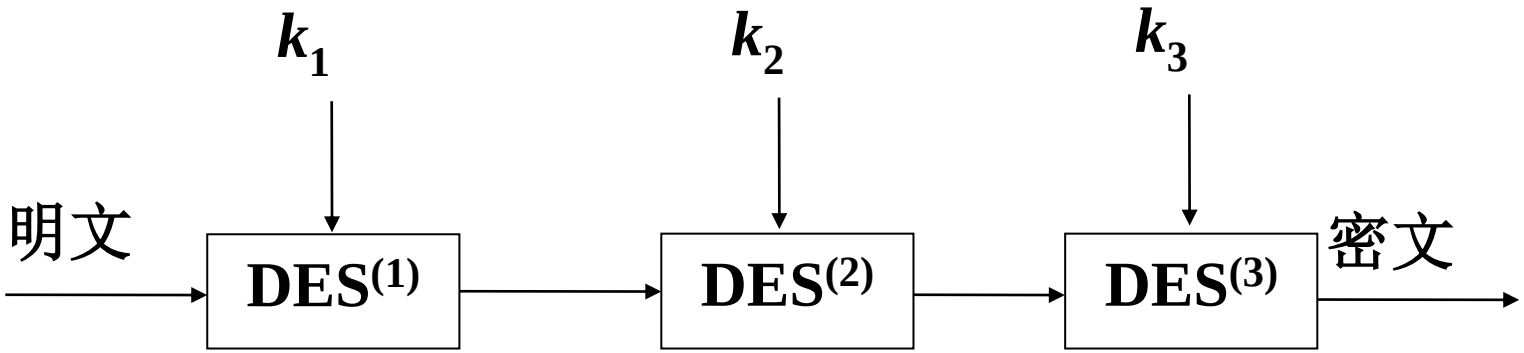
- 以密钥k_1的所有2^{56}个可能的取值对此明文x_1加密，并将密文z存储在一个表中；
- 从所有可能的2^{56}个密钥k_2中依任意次序选出一个对给定的密文y_1解密，并将每次解密结果z在上述表中查找相匹配的值。一旦找到，则可确定出两个密钥k_1和k_2；
- 以此对密钥k_1和k_2对另一已知明文密文对(x_2, y_2)中的明文x_2进行加密，如果能得出相应的密文y_2就可确定k_1和k_2是所要找的密钥。

中间相遇攻击的复杂度

- 对于给定明文x，以两重DES加密将有2^{64}个可能的密文。
- 可能的密钥数为2^{112}个。所以，在给定明文下，将有$2^{112}/2^{64}=2^{48}$个密钥能产生给定的密文。
- 用另一对64比特明文/密文对进行检验，就使虚报率降为$2^{48-64}=2^{-16}$。
- 这一攻击法所需的存储量为$2^{56}\times 8$ Byte，最大试验的加密次数$2\times 2^{56}=2^{57}$。这说明破译双重DES的难度为2^{57}量级。

三重DES算法

- 三重DES中三个密码组件既可以是一个加密函数，也可以是一个解密函数。
- 当$k_1=k_3$时，则称为双密钥三重DES

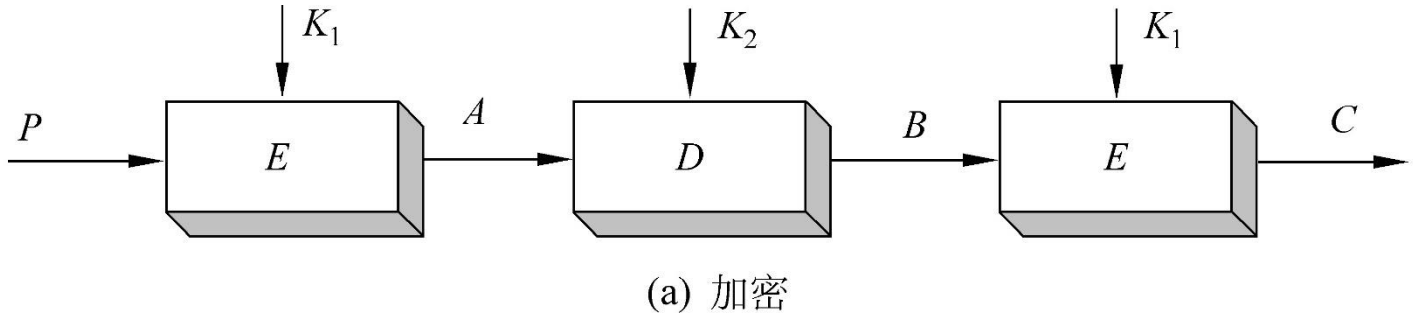


双密钥三重DES算法

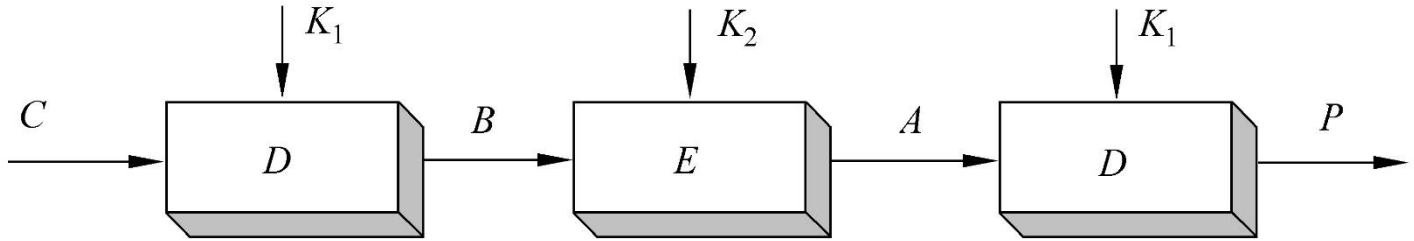
加密：$y=E_{k1}[D_{k2}[E_{k1}[x]]]$

解密：$x=D_{k1}[E_{k2}[D_{k1}[y]]]$

- 称其为加密-解密-加密方案，简记为EDE(encrypt-decrypt-encrypt)。
- 此方案已在ANSI X9.17和ISO 8732标准中采用，并在保密增强邮件(PEM)系统中得到利用。



(a) 加密



(b) 解密

双密钥三重DES算法的安全性

- 破译它的穷举密钥搜索量为$2^{112} \approx 5 \times 10^{35}$量级
- 差分分析破译也要超过10^{52}量级
- 此方案仍有足够的安全性

分组密码的工作模式

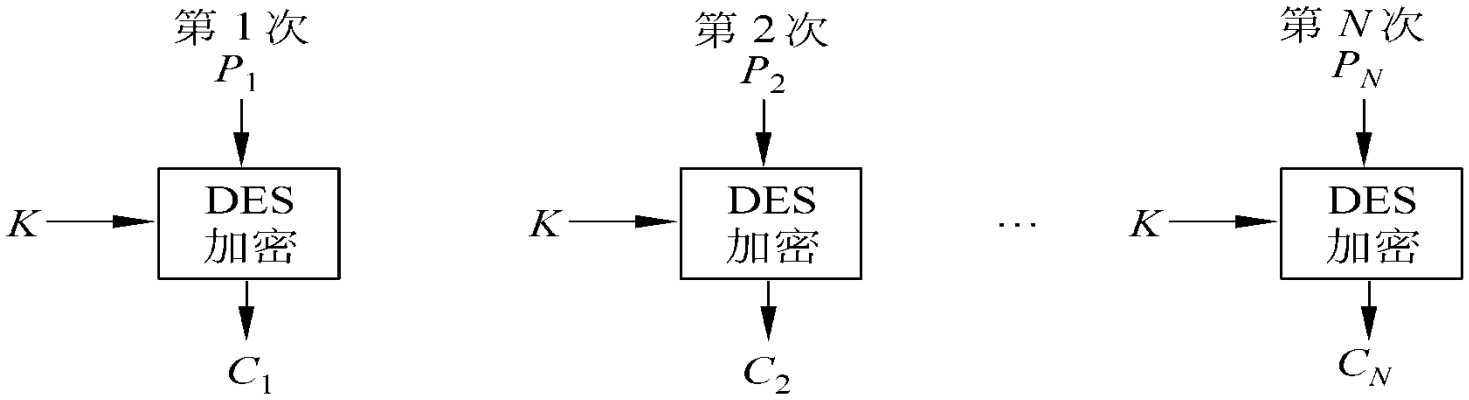
为什么需要工作模式？

- 分组密码的工作模式是：根据不同的数据格式和安全性要求，以一个具体的分组密码算法为基础构造一个分组密码系统的方法
- 分组密码的工作模式应当力求简单，有效和易于实现
- 需要采用适当的工作模式来隐蔽明文的统计特性、数据的格式等
- 降低删除、重放、插入和伪造成功的机会

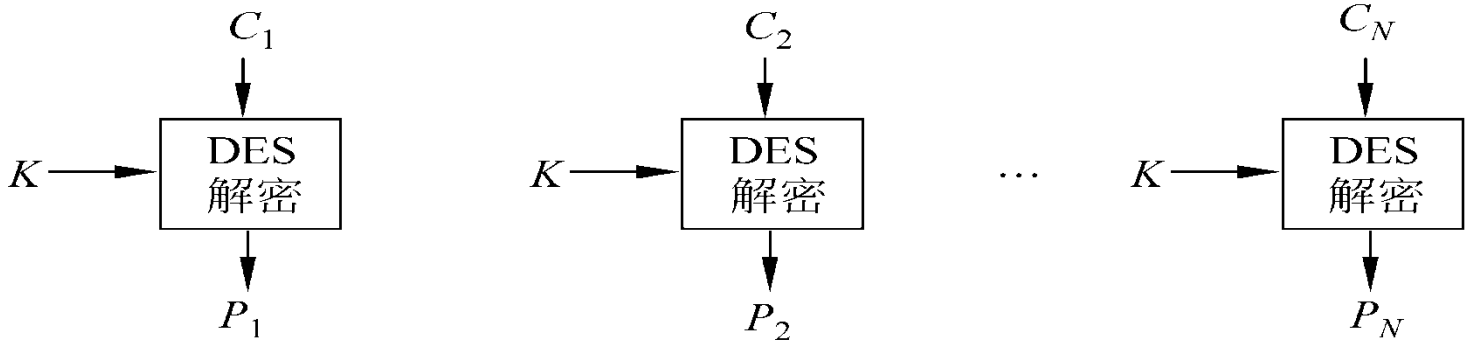
分组密码的主要工作模式

1. 电码本(ECB)模式
2. 密码分组链接(CBC)模式
3. 密码反馈(CFB)模式
4. 输出反馈(OFB)模式
5. 计数器模式

电码本ECB（Electronic Code Book）模式



(a) 加密



(b) 解密

ECB模式的优、缺点

- 优点：

（1）实现简单;

（2）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：

（1）相同明文分组对应相同密文分组：不能隐蔽明文分组的统计规律和结构规律,不能抵抗替换攻击

- 应用：

（1）用于随机数的加密保护

（2）用于会话密钥的加密

电码本模式缺陷的例子

- 例: 假设银行A和银行B之间的资金转帐系统所使用报文模式如下：

1	2	3	4	5	6	7	8	9	10	11	12	13
时间标记	发送银行		接收银行	储户姓名 1						储户帐号 1		存款金额

1	2	3	4	5	6	7	8	9	10	11	12	13
时间标记	发送银行		接收银行	储户姓名 2						储户帐号 2		存款金额

- 敌手C通过截收从A到B的加密消息，只要将第5至第12分组替换为自己的姓名和帐号相对应的密文，即可将别人的存款存入自己的帐号。

密码分组链接CBC（Cipher Block Chaining）模式

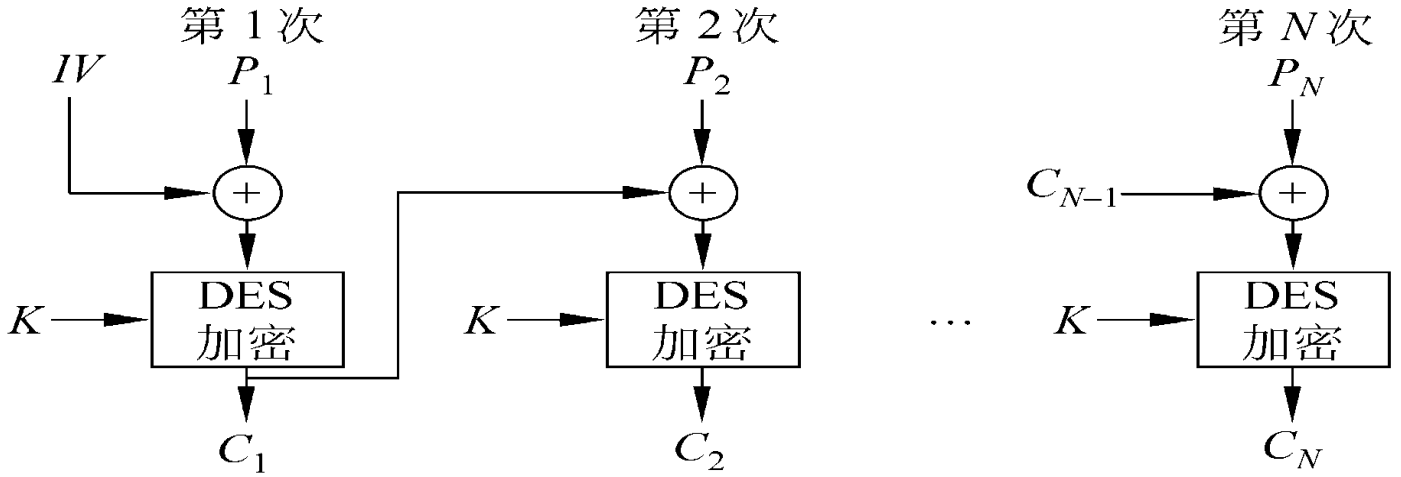
- 这种模式先将明文分组与上一次的密文块进行按比特异或，然后再进行加密处理。这种模式必须选择一个初始向量c_0=IV，用于加密第一块明文。
- 加密过程为

$$c_i = E_k(m_i \oplus c_{i-1})$$

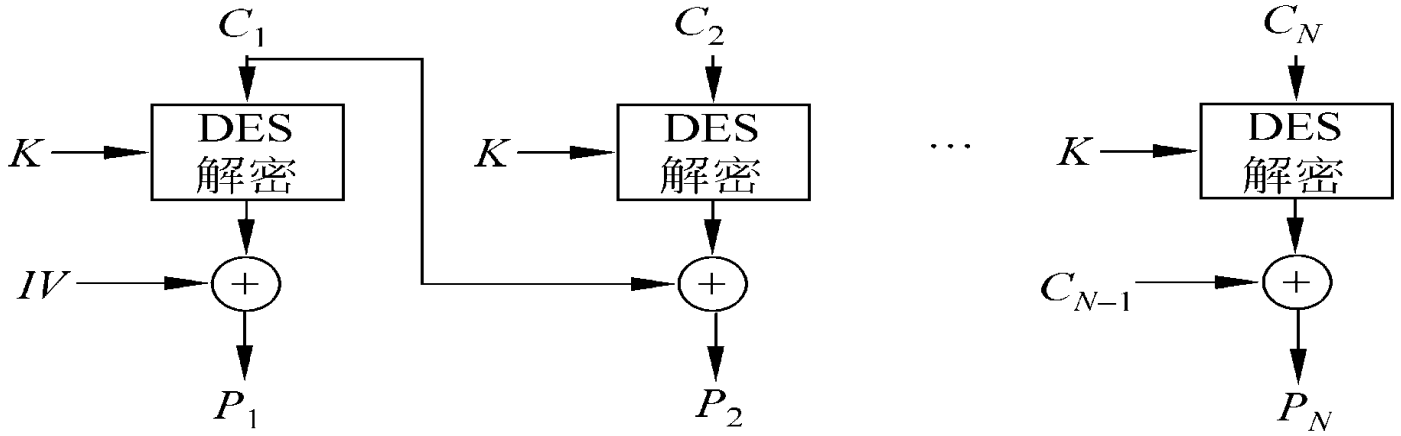
- 解密过程为

$$m_i = D_k(c_i) \oplus c_{i-1}$$

密码分组链接CBC（Cipher Block Chaining）模式



(a) 加密



(b) 解密

CBC模式的特点

1. 明文块的统计特性得到了隐蔽
   - 由于在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藏
2. 具有有限的(两步)错误传播特性
   - 一个密文块的错误将导致两个密文块不能正确解密
3. 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

利用CBC模式实现报文的完整性认证

- 目的:检查文件在(直接或加密)传输和存储中是否遭到有意或无意的篡改.
- 关键技术:

  (1) 文件的制造者和检验者共享一个密钥

  (2) 文件的明文必须具有检验者预先知道的冗余度

  (3) 文件的制造者用共享密钥对具有约定冗余度的明文用**CBC**模式加密

  (4) 文件的检验者用共享密钥对密文解密，并检验约定冗余度是否正确

报文完整性认证的具体实现技术

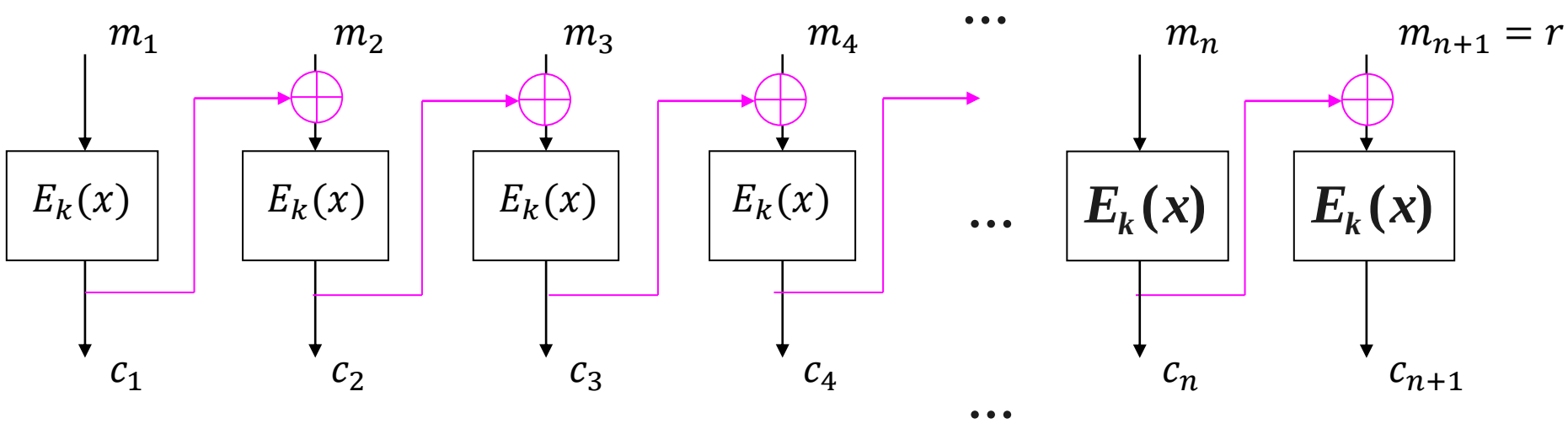
- (1) 文件的制造者和检验者共享一个密钥；
- (2) 利用文件的明文**m**产生一个奇偶校验码**r**的分组；
- (3) 采用分组密码的**CBC**模式，对附带校验码的已扩充的明文**(m, r)**进行加密，得到的最后一个密文分组就是认证码

认证码生成

n个分组明文 $m=(m_1,\cdots,m_n)$ ，校验码为

$$r=m_{n+1}=m_1\oplus\cdots\oplus m_n$$

C_{n+1}为认证码。

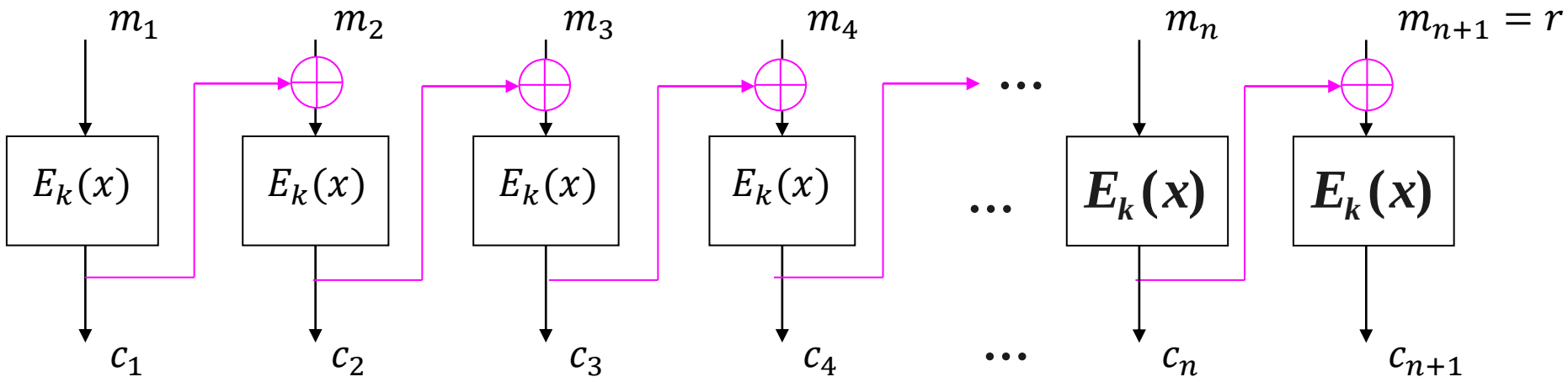


(1) 仅需对明文认证,而不需加密时,传送明文**m**和认证码C_{n+1}, 此时也可仅保留C_{n+1}的 **t** 个比特作为认证码;

(2) 既需对明文认证,又需要加密时,传送密文**C**和认证码C_{n+1}

认证码检验

- （1）仅需对明文认证而不需加密时,此时验证者仅收到明文m和认证码C_{n+1}，他需要：
- **Step1** 产生明文m的校验码 $r = m_{n+1} = m_1 \oplus \cdots \oplus m_n$
- **Step2** 利用共享密钥使用CBC模式对(m,r)加密，将得到的最后一个密文分组与接收到的认证码C_{n+1}比较，二者一致时判定接收的明文无错；二者不一致时判定明文出错。

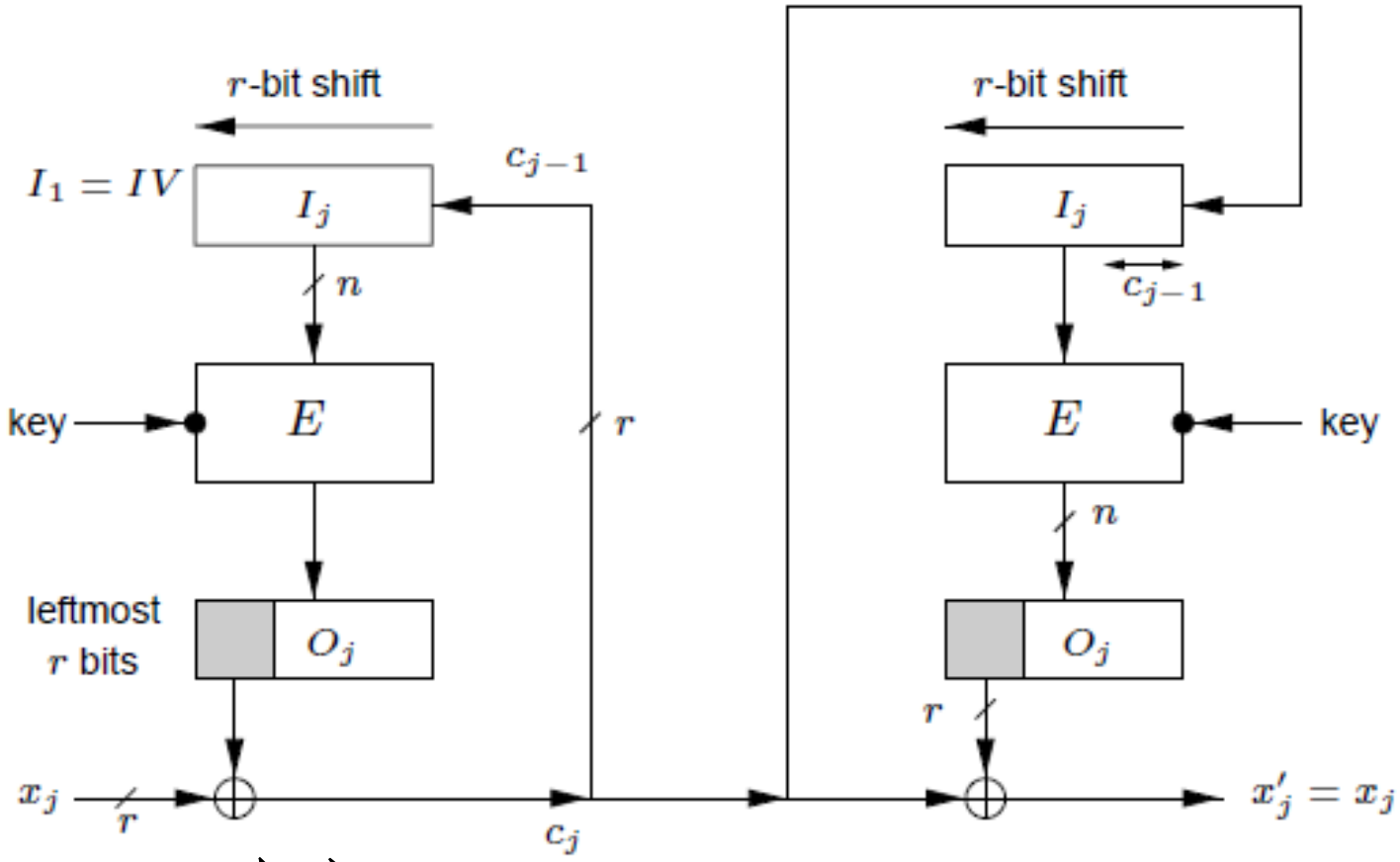


密码反馈CFB（Cipher Feedback）模式

- 若待加密消息需按字符、字节或比特处理时，可采用**CFB**模式。并称待加密消息按 ***r*** 比特处理的**CFB**模式为 ***r*** 比特**CFB**模式。
- 适用范围:
  - 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
  - 例如,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

CFB的加密解密

- 若记$IV=c_{-l+1}\ldots c_{-1}\,c_0$，$|c_i|=r$，则加密过程可表示为：$c_i = x_i \oplus left_r(E_k(c_{i-l}\cdots c_{i-2}c_{i-1}))$



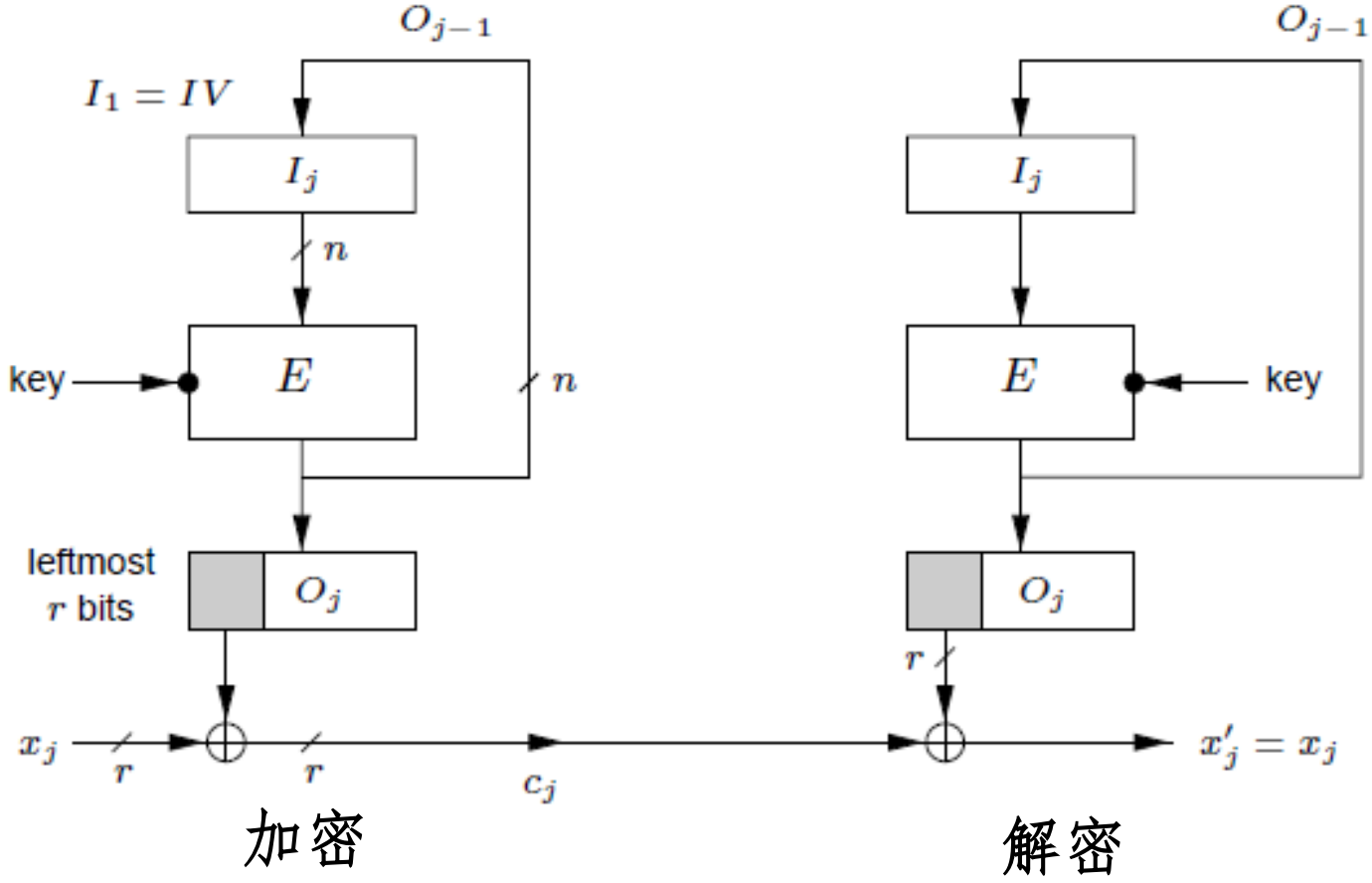
加密　　解密

CFB模式的特点

- 相同明文：和按**CBC**模式加密一样，改变***IV***同样会导致相同的明文输入得到不同的加密输出。***IV***无需保密（虽在某些应用中**IV**须是不可预测的）。
- 链接依赖性：类似**CBC**加密，链接机制致使密文组依赖于当前明文组和其前面的明文组；因此，重排密文组会影响解密。
- 错误的传播：一个或多个比特错误出现在任一个**r**比特的密文组中会影响这个组和后继 $\lceil n/r \rceil$ 个密文组的解密。
- 错误恢复：**CFB**和**CBC**相似，也是自同步的，但它需有 $\lceil n/r \rceil$ 个密文组才能还原。

输出反馈OFB(Output Feedback)模式

- **OFB模式在结构上类似于CFB模式，但反馈的内容是DES的输出而不是密文！**



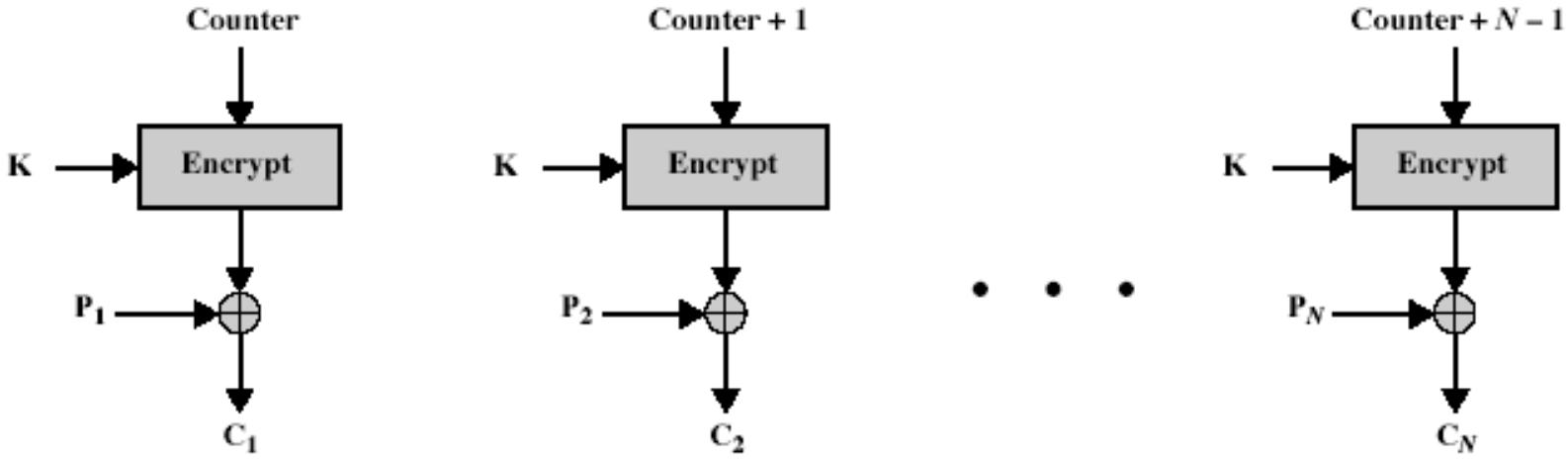
OFB工作模式的特点

- 相同明文：和**CBC**及**CFB**一样，改变***IV***同样会导致相同的明文输入得到不同的加密输出。
- 链接依赖性：密钥流是独立于明文的。
- 错误传播：有一个或多个比特错误的任一密文字符仅会影响该字符的解密， 密文字符的某比特位置出错将致使还原明文的相应位置也出错。
- 错误恢复：**OFB**模式能从密文比特错误中得以恢复，但在丢失密文比特后就无法实现自同步了，这是因为丢失密文比特会破坏密钥流的编排。**(**可以出错，不能丢失**)**

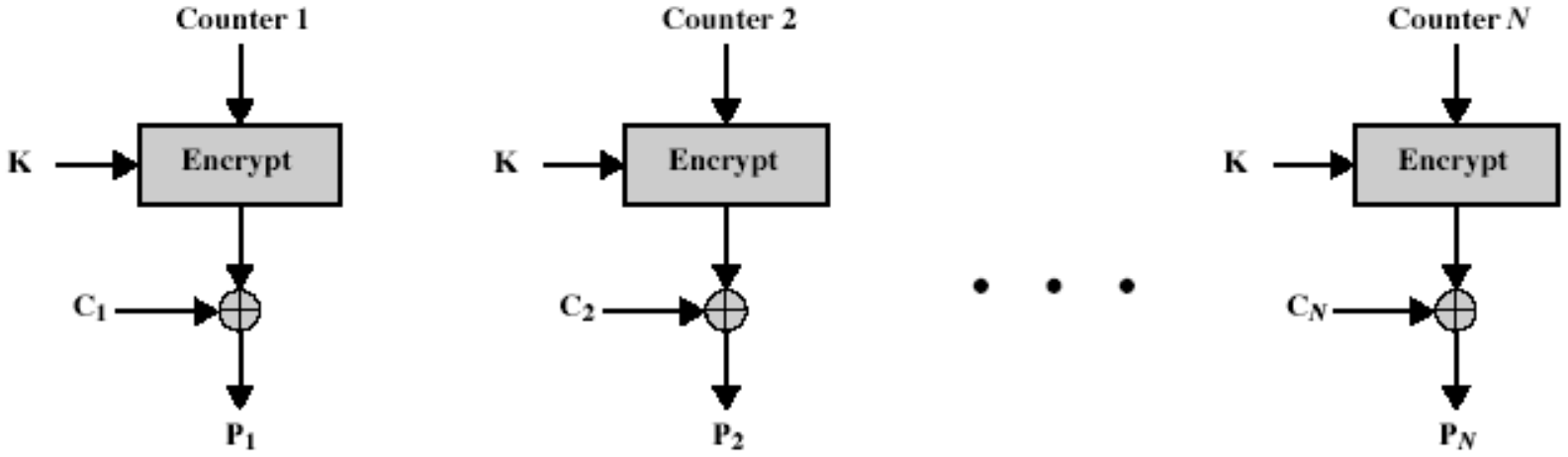
计数器模式

- 利用固定密钥**k**对自然数序列**1,2,3,…,n, …**加密，将得到的密文分组序列看作密钥流序列，按加法密码的方式与明文分组逐位异或的一种方式
- 利用这种方式可以产生伪随机数序列,其伪随机特性远比计算机产生的随机数的性质好

计数器模式的结构



(a) Encryption



(b) Decryption

CTR的优点

- 效率
  - 可并行加密
  - 预处理
  - 吞吐量仅受可使用并行数量的限制
- 加密数据块的随机访问
- 简单性（只要求实现加密算法）

有限域基础

什么是域

- F是一个非空集合，定义了加法、乘法两个二元运算，对这两个运算封闭
- 加法满足：对于任意a,b,c∈F
  - a+b=b+a；交换律
  - (a+b)+c=a+(b+c)；结合律
  - 存在0 ∈F，使得a+0=a；有零元
  - 存在-a ∈F，使得a+(-a)=0；有负元
- 乘法满足：对于任意a,b,c∈F
  - a·b=b·a；交换律
  - (a·b) ·c=a·(b·c)；结合律
  - 存在e ∈F，使得a·e=a；有单位元
  - 存在a^{-1} ∈F，使得$a \cdot a^{-1} = e$；有逆元
- 乘法对加法满足分配率
  - a·(b+c)=a·b+a·c

域的例子

- $Z_n=\{0,1,2,\ldots, n-1\}_{mod n}$，加法和乘法都是模n的运算，运算封闭
- 加法满足结合律和交换律，有零元0，有负元
- 乘法满足结合律和交换律，有单位元1，不一定有逆元
- Z_n中的数什么时候才有乘法逆元呢？
- 引理：整数a在模n乘法下有逆元，当且仅当a与n互素。
- 所有与n互素的元素在模n乘法下构成乘法交换群
- 1…n-1都与n互素，则n为素数
- 对于任一素数p，Z_p为域，其元素个数为p个

域的例子（续）

- $F[x]/(f(x))=\{r(x)=r_{n-1}x^{n-1}+r_{n-2}x^{n-2}+\cdots+r_1x+r_0 | r_i \in F, 0\le i\le n-1\}$，加法和乘法都是模$f(x)$的运算，运算封闭
- 加法满足结合律和交换律，有零元0，有负元
- 乘法满足结合律和交换律，有单位元1，不一定有逆元
- $F[x]/(f(x))$中的多项式什么时候才有乘法逆元呢？

域的例子（续）

- 引理：r(x)在模f(x)乘法下有逆元，当且仅当r(x)与f(x)互素。
- 所有与f(x)互素的元素在模f(x)乘法下构成乘法交换群
- 次数比f(x)的次数低的多项式都与f(x)互素，则f(x)为不可约多项式
- 对于任一不可约多项式，F[x]/(f(x))为域
- 若$F=Z_p$，则F[x]/(f(x))中元素个数为p^n个
- p^n域的构造方法是首先选取Z_p中的一个n次不可约多项式f(x)，然后构造集合$F[x]/(f(x))=\{r(x)=r_{n-1}x^{n-1}+r_{n-2}x^{n-2}+\ldots+r_1x+r_0 | r_i \in F, 0\le i\le n-1\}$

集合中的加法和乘法运算为模多项式f(x)的运算

AES中的处理单元

- **AES**加密标准算法中是以字节为处理单元
- 可以将每一字节看作是有限域$GF(2^8)$上的一个元素，分别对应于一个次数不超过7的多项式。如$b_7b_6b_5b_4b_3b_2b_1b_0$可表示为多项式

$$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0$$

- 还可以将每个字节表示为一个十六进制数，即每4比特表示为一个十六进制数，代表较高位的4比特的符号仍在左边。例如，**01101011**可表示为**6B**
- 它们之间的运算为$GF(2^8)$中的运算

GF(2⁸)中的运算

定义：在*GF*(2⁸)上的加法定义为二进制多项式的加法，且其系数模2。

例如：‘57’+‘83’=‘D4’，用多项式表示为

$$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2 \pmod{m(x)}$$

用二进制表示为

$$01010111+10000011=11010100$$

定义：在*GF*(2⁸)上的乘法（用符号·表示）定义为二进制多项式的乘积模一个次数为8的不可约二进制多项式

$$m(x)= x^8+x^4+x^3+x+1$$

它的十六进制表示为‘11B’。

例如：‘57’·‘83’=‘C1’可表示为以下的多项式乘法：

$$(x^6+x^4+x^2+x+1)\cdot(x^7+x+1)=x^7+x^6+1 \pmod{m(x)}$$

GF(2^8)中的逆元和x乘法

定义：对任何次数小于8的多项式b(x)，可用推广的欧几里得算法得

$$b(x)a(x)+m(x)c(x)=1$$

即$a(x)\cdot b(x)=1 \bmod m(x)$。因此a(x)是b(x)的乘法逆元。

定义：函数xtime(x)定义为GF（2）上的

$x\cdot b(x)=b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x \bmod m(x)$。

注：$m(x)= x^8+x^4+x^3+x+1$

其运算如下：若$b_7=0$，则$x\cdot b(x)$的结果就是把字节b左移一位，且在最右边补上上0；若$b_7=1$，则先对b(x)在字节内左移一位（最后一位补0），则再与‘1B’（00011011）做逐比特异或。

xtime(x)的例子

- 例如，‘57’·‘13’可按如下方式实现：
- ‘57’·‘02’=xtime(57)=‘AE’；
- ‘57’·‘04’=xtime(AE)=‘47’；
- ‘57’·‘08’=xtime(47)=‘8E’；
- ‘57’·‘10’=xtime(8E)=‘07’；
- ‘57’·‘13’=‘57’·(‘01’ ⊕ ‘02’ ⊕ ‘10’)
- =‘57’ ⊕ ‘AE’ ⊕ ‘07’=‘FE’

GF(2^8)上的模多项式运算

- 4个字节构成的向量可以表示为系数在GF(2^8)上的次数小于4的多项式
- 多项式的加法就是对应系数相加；换句话说，多项式的加法就是4字节向量的逐比特异或。
- 规定多项式的乘法运算必须要取模$M(x)=x^4+1$，这样使得次数小于4的多项式的乘积仍然是一个次数小于4的多项式，将多项式的模乘运算记为 $\otimes$，设$a(x)=a_3x^3+a_2x^2+a_1x+a_0$，$b(x)=b_3x^3+b_2x^2+b_1x+b_0$，$c(x)=a(x)\otimes b(x)=c_3x^3+c_2x^2+c_1x+c_0$。由于$x^j \bmod (x^4+1)= x^{j \bmod 4}$，所以

$$c_0=a_0b_0\oplus a_3b_1 \oplus a_2b_2 \oplus a_1b_3;$$
$$c_1= a_1b_0 \oplus a_0b_1 \oplus a_3b_2 \oplus a_2b_3;$$
$$c_2= a_2b_0 \oplus a_1b_1 \oplus a_0b_2 \oplus a_3b_3;$$
$$c_3= a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus a_0b_3。$$

多项式乘法的矩阵表示

可将上述计算表示为

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

其中元素的加法和乘法为**GF（2^8）**域上的运算

模x^4+1逆元

定理：系数在GF(2^8)上的多项式$a_3x^3+a_2x^2+a_1x+a_0$是模x^4+1可逆的，当且仅当矩阵

$$\begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix}$$

在GF(2^8)上可逆。

证明

证明：$a_3x^3+a_2x^2+a_1x+a_0$是模x^4+1可逆的，当且仅当存在多项式$h_3x^3+h_2x^2+h_1x+h_0$满足

$$(a_3x^3+a_2x^2+a_1x+a_0)(h_3x^3+h_2x^2+h_1x+h_0)=1 \bmod (x^4+1)$$

因此有

$$(a_3x^3+a_2x^2+a_1x+a_0)(h_2x^3+h_1x^2+h_0x+h_3)=x \bmod (x^4+1)$$

$$(a_3x^3+a_2x^2+a_1x+a_0)(h_1x^3+h_0x^2+h_3x+h_2)=x^2 \bmod (x^4+1)$$

$$(a_3x^3+a_2x^2+a_1x+a_0)(h_0x^3+h_3x^2+h_2x+h_1)=x^3 \bmod (x^4+1)$$

证明（续）

将以上关系写成矩阵形式即得

$$\begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} h_0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_3 \\ h_3 & h_2 & h_1 & h_0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

（证毕）

AES算法简介

AES标准算法征集

- DES算法由于其密钥较短，难以抵抗现有的攻击，因此不再作为加密标准
- 1997年1月，美国NIST向全世界密码学界发出征集21世纪高级加密标准（AES——Advanced Encryption Standard）算法的公告，并成立了AES标准工作研究室，1997年4月15日的例会制定了对AES的评估标准。

AES算法征集的要求

（1）AES是公开的；

（2）AES为对称密钥分组密码体制；

（3）AES的密钥长度可变，可按需要增大；

（4）AES适于用软件和硬件实现；

（5）AES可以自由地使用，或按符合美国国家标准（ANST）策略的条件使用。

算法衡量条件

- 满足以上要求的AES算法，需按下述条件判断优劣
  - 安全性
  - 计算效率
  - 内存要求
  - 使用简便性
  - 灵活性

AES的评审

- 1998年4月15日全面征集AES算法的工作结束。1998年8月20日举行了首届AES讨论会，对涉及14个国家的密码学家所提出的候选AES算法进行了评估和测试，初选并公布了15个被选方案，供大家公开讨论。

  CAST-256，RC-6，CRYPTON-128，DEAL-128，

  FROG，DFC，LOKI-97，MAGENTA，

  MARS，HPC，RIJNDAEL，SAFER+，

  SERPENT，E-2，TWOFISH。

- 这些算法设计思想新颖，技术水平先进，算法的强度都超过3-DES，实现速度快于3-DES。

AES的评审（续）

- 1999年8月9日NIST宣布第二轮筛选出的5个候选算法为：

  MARS(C.Burwick等, IBM），

  RC6™（R. Rivest等, RSA Lab.),

  RIJNDEAL(J. Daemen, V. Rijmen, 比利时),

  SERPENT(R. Anderson等，英国、以利时、挪威),

  TWOFISH(B. Schneier，美国)。

- 2000年10月2日，NIST宣布Rijndael作为新的AES

AES算法设计思想

- 设计简单
- 在多个平台上速度快，编码紧凑
- 抵抗所有已知的攻击
- Rijndael没有采用Feistel结构，轮函数由3个不同的可逆均匀变换构成的，称为3个层
  - 均匀变换是指状态的每个bit都用类似的方法处理

轮函数的3层

- 线性混合层
  - 确保多轮之上的高度扩散；
- 非线性层
  - 将具有最优的“最坏情况非线性特性”的S盒并行使用，确保混淆特性；
- 密钥加层
  - 单轮子密钥简单的异或到中间状态上，实现一次性掩盖。

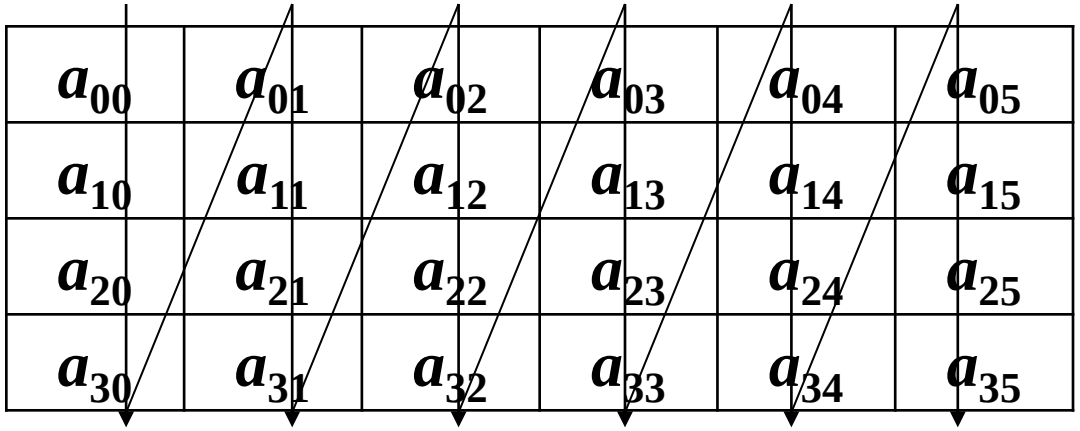
算法说明

- **Rijndael明文分组可变，128、192、256比特**
- **AES明文分组固定为128比特**
- **密钥长度可变，各自可独立指定为128、192、256比特。**
- 状态
  - 算法中间的结果也需要分组，称之为状态，状态可以用以字节为元素的矩阵阵列表示，该阵列有4行，列数N_b为分组长度除32
- 种子密钥
  - 以字节为元素的矩阵阵列描述，阵列为4行，列数N_k为密钥长度除32

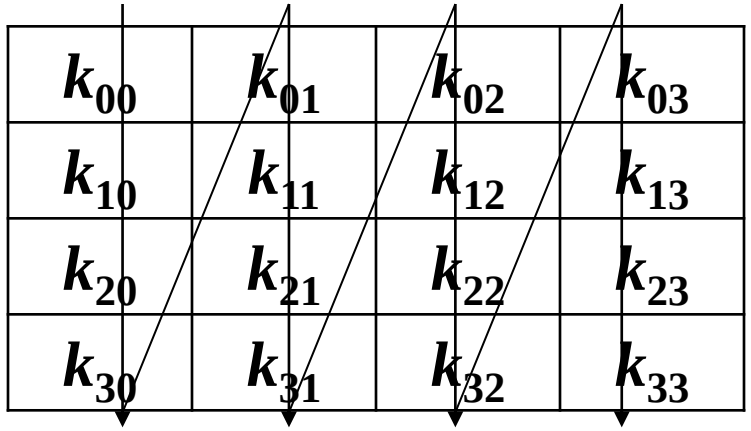
算法说明

- 算法的输入、输出和种子密钥可看成字节组成的一维数组。
- 下标范围
  - 输入输出：**0—$4N_b$-1**
  - 种子密钥：**0—$4N_k$-1**

N_b=6和N_k=4的状态密钥阵列



按此顺序放入和读出



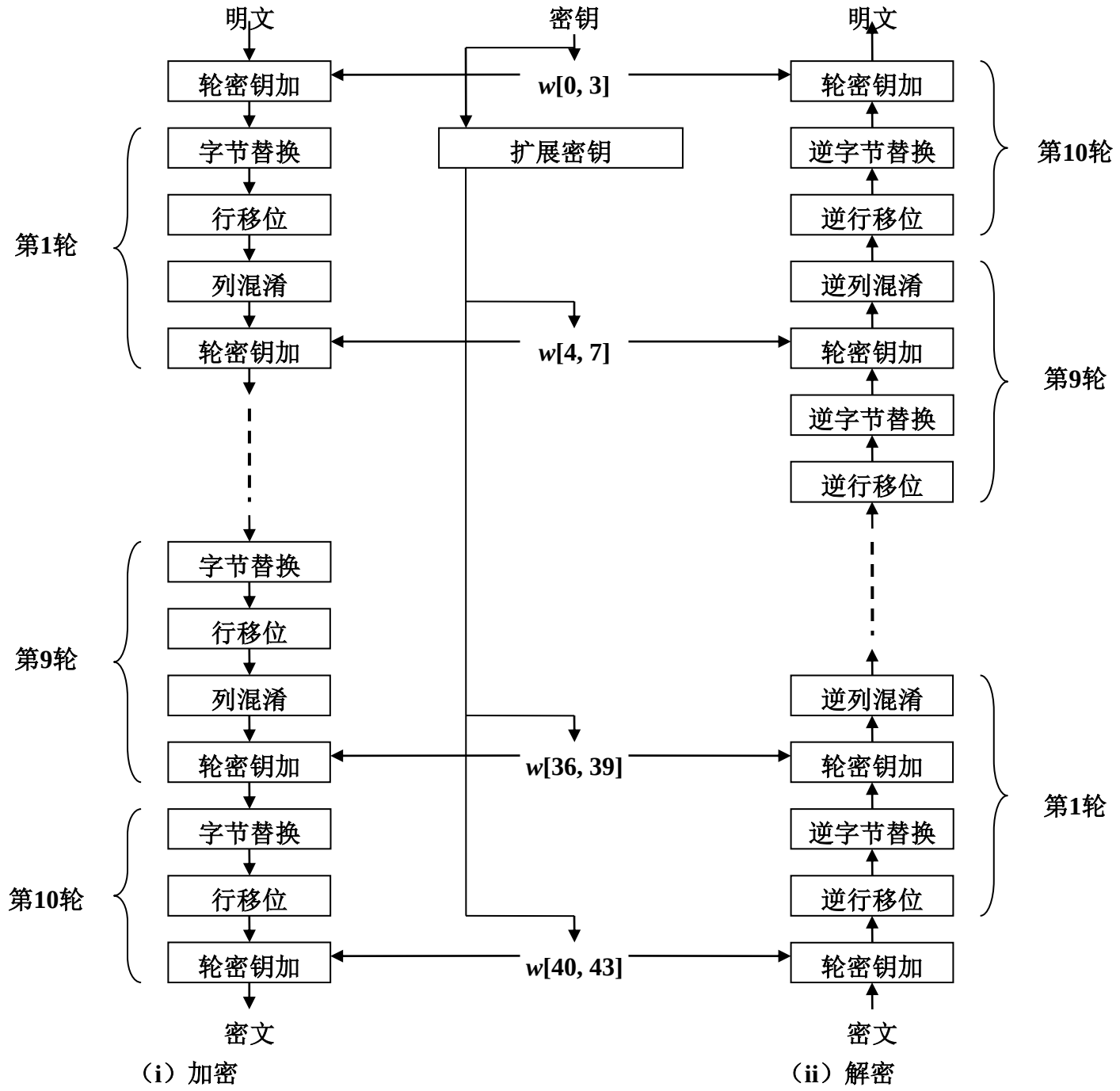
按此顺序放入

分组下标n
阵列位置(i,j)
i=n mod 4, j=[n/4]; n=i+4j

轮数对应关系

- 轮数N_r与N_b和N_k对应关系

	$N_b=4$	$N_b=6$	$N_b=8$
$N_k=4$	10	12	14
$N_k=6$	12	12	14
$N_k=8$	14	14	14


明文
密钥
明文
轮密钥加
w[0, 3]
轮密钥加
字节替换
扩展密钥
逆字节替换
第10轮
行移位
逆行移位
第1轮
列混淆
逆列混淆
轮密钥加
w[4, 7]
轮密钥加
第9轮
逆字节替换
逆行移位
字节替换
行移位
第9轮
列混淆
逆列混淆
轮密钥加
w[36, 39]
轮密钥加
第1轮
字节替换
逆字节替换
第10轮
行移位
逆行移位
轮密钥加
w[40, 43]
轮密钥加
密文
密文
（i）加密
（ii）解密

AES的轮函数

字节代换

- 非线性代换，独立地对状态的每个字节进行，并且代换表(S盒)可逆，记为**ByteSub(State),**分两步

(1) 将字节作为**GF(2^8)**上的元素映射到自己的逆元

(2) 将字节做**GF(2)**上的仿射变换

即$\boldsymbol{y = Ax^{-1} + B}$

其中**A**是一个**GF**（2）上**8X8**的可逆矩阵，**B**是**GF**（2）上一个**8**位列向量

字节代换

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

AES的S盒

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	B5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	B0	54	bb	16

AES的S盒的使用：输入8a，输出7e，即7e=S（8a）

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	B5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	B0	54	bb	16

逆字节代换InvSubBytes()

- 逆字节替代变换是字节替代变换的逆变换，在状态的每个字节上应用逆S盒
  - 这是通过应用字节替代变换中的仿射变换的逆变换，再对所得结果应用有限域的乘法逆运算得到的

即

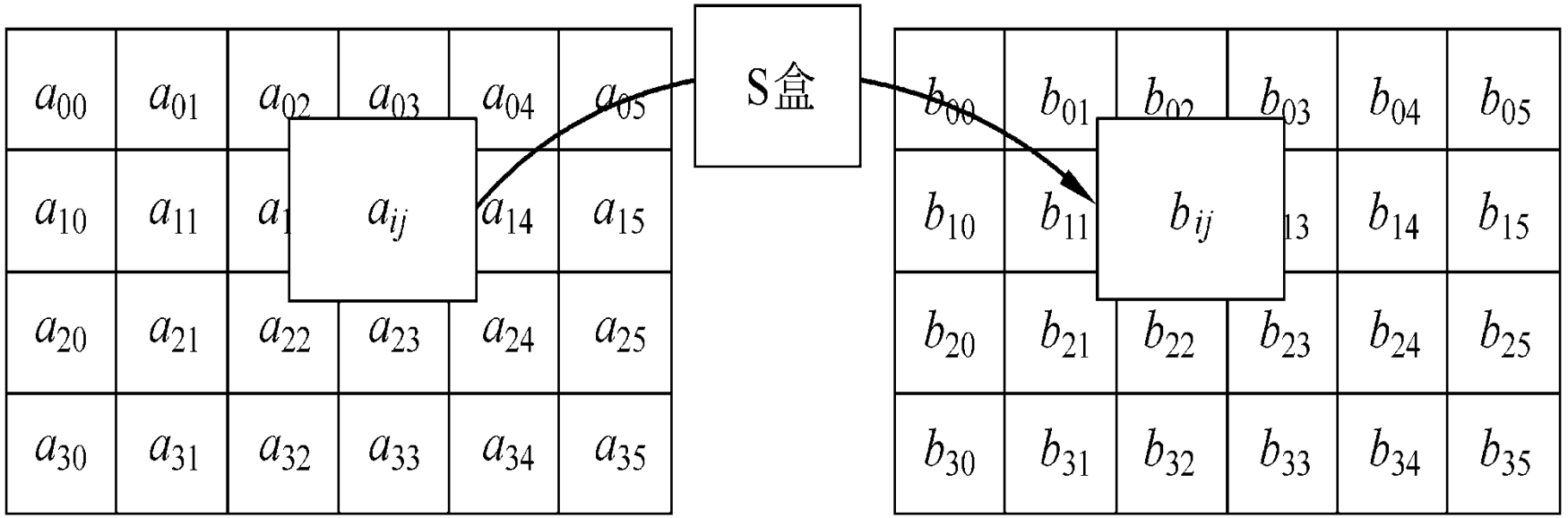
$$y = (A^{-1}(x - B))^{-1}$$

AES的逆S盒

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	A1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	B6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

字节代换示意图

- 上述S-盒对状态的所有字节所做的变换记为ByteSub（State）

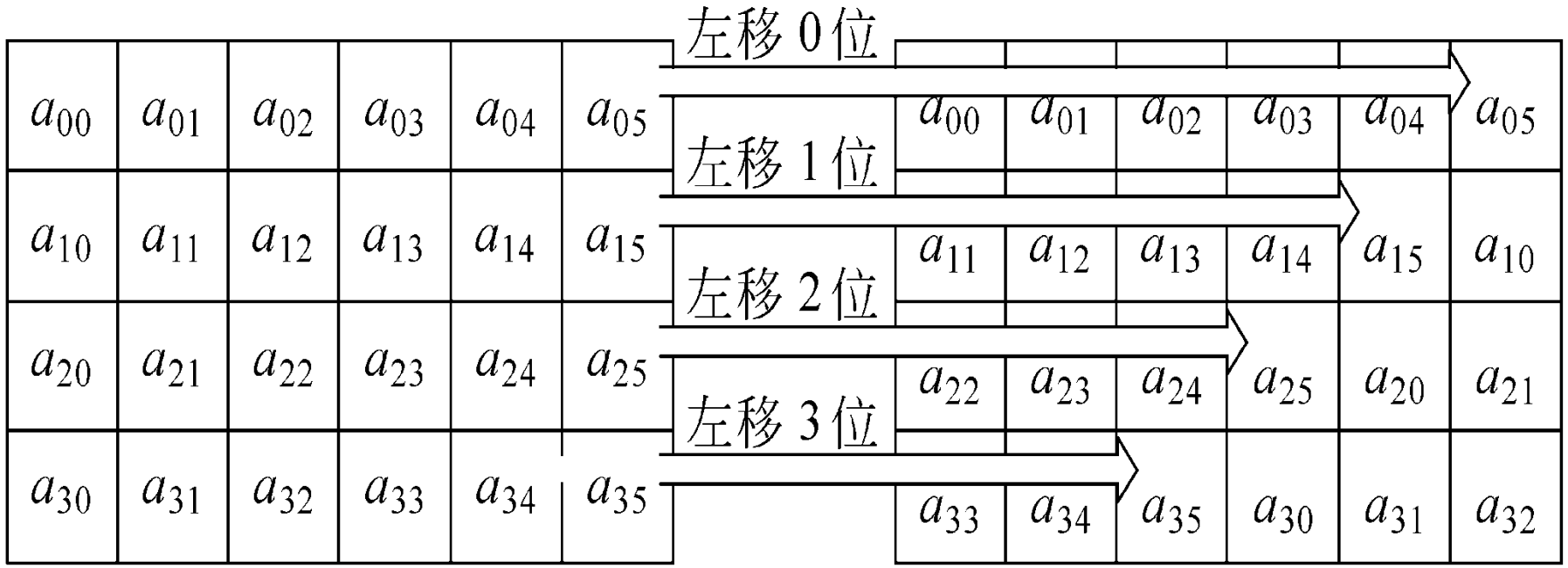


行移位

- 将状态阵列的各行进行循环移位，不同行的移位量不同
- 0行：不动
- 1行：循环左移C1字节
- 2行：循环左移C2字节
- 3行：循环左移C3字节
- 记为：ShiftRow(State)

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

行移位示意图


左移 0 位
左移 1 位
左移 2 位
左移 3 位
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}
a_{00} a_{01} a_{02} a_{03} a_{04} a_{05}
a_{11} a_{12} a_{13} a_{14} a_{15} a_{10}
a_{22} a_{23} a_{24} a_{25} a_{20} a_{21}
a_{33} a_{34} a_{35} a_{30} a_{31} a_{32}

逆行移位InvShiftRows()

- 逆行移位变换是行移位变换的逆变换
  - 它对状态的每一行进行循环右移，
  - 第0行保持不变
  - 第1行循环右移C_1个字节
  - 第2行循环右移C_2个字节
  - 第3行循环右移C_3个字节

列混淆

- 将每列视为**GF(2^8)**上多项式，与固定的多项式**c(x)**进行模**x^4+1**乘法，记为 $\otimes$,要求**c(x)**模**x^4+1**可逆。
- 表示为**MixColumn(State)**

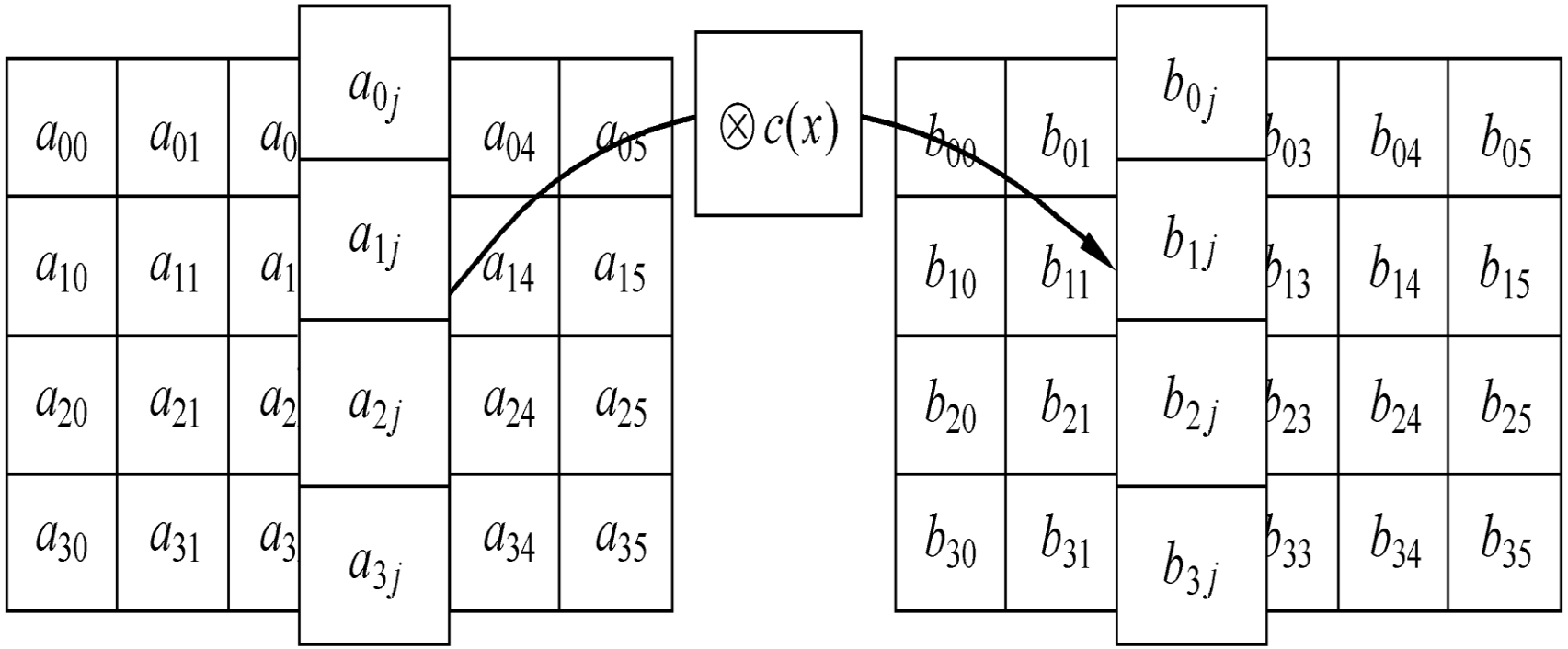
$$c(x) = '03'x^3 + '01'x^2 + '01'x + '02'$$

列混淆的矩阵表示

列混淆运算也可写为矩阵乘法。设 $b(x) = c(x) \otimes a(x)$

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

列混淆运算示意图


a_{00} a_{01} a_{0} a_{0j} a_{04} a_{05}
a_{10} a_{11} a_{1} a_{1j} a_{14} a_{15}
a_{20} a_{21} a_{2} a_{2j} a_{24} a_{25}
a_{30} a_{31} a_{3} a_{3j} a_{34} a_{35}
$\otimes c(x)$
b_{00} b_{01} b_{0j} b_{03} b_{04} b_{05}
b_{10} b_{11} b_{1j} b_{13} b_{14} b_{15}
b_{20} b_{21} b_{2j} b_{23} b_{24} b_{25}
b_{30} b_{31} b_{3j} b_{33} b_{34} b_{35}

逆列混淆InvMixColumns()

- 逆列混淆变换是列混淆变换的逆
- 它将状态矩阵中的每一列视为系数在 $GF(2^8)$ 上的次数小于4的多项式与同一个固定的多项式 $d(x)$相乘。 $d(x)$满足

$$(\text{‘03’}x^3+\text{‘01’}x^2+\text{‘01’}x+\text{‘02’}) \otimes d(x)=\text{‘01’}$$

由此可得

$$d(x)=\text{‘0B’}x^3+\text{‘0D’}x^2+\text{‘09’}x+\text{‘0E’}$$

逆列混淆的矩阵形式

- 同样，逆列混淆可以写成矩阵乘法形式

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_2 \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

轮密钥加

- 轮密钥与状态进行逐比特异或。
- 轮密钥由种子密钥通过密钥编排算法得到
- 轮密钥长度与分组长度相同
- 表示为AddRoundKey(State,RoundKey)

AES的密钥编排及伪代码

密钥编排

•密钥编排指从种子密钥得到轮密钥的过程，它由密钥扩展和轮密钥选取两部分组成。其基本原则如下：

• （1）轮密钥的总比特数等于分组长度乘以(轮数加1)；例如要将128比特的明文经过10轮的加密，则总共需要（10+1）*128=1408比特的密钥。

• （2）扩展：种子密钥被扩展成为扩展密钥；

• （3）选取：轮密钥从扩展密钥中取，其中第1轮轮密钥取扩展密钥的前N_b个字，第2轮轮密钥取接下来的N_b个字，如此下去。

密钥扩展

•扩展密钥是以4字节字为元素的一维阵列，表示为W[Nb*(N_r+1)]，其中前N_k个字取为种子密钥，以后每个字按递归方式定义。扩展算法根据$N_k \leq 6$和$N_k > 6$有所不同。

w_0	w_1	w_2	w_3	w_4	w_5	……	
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}		

……

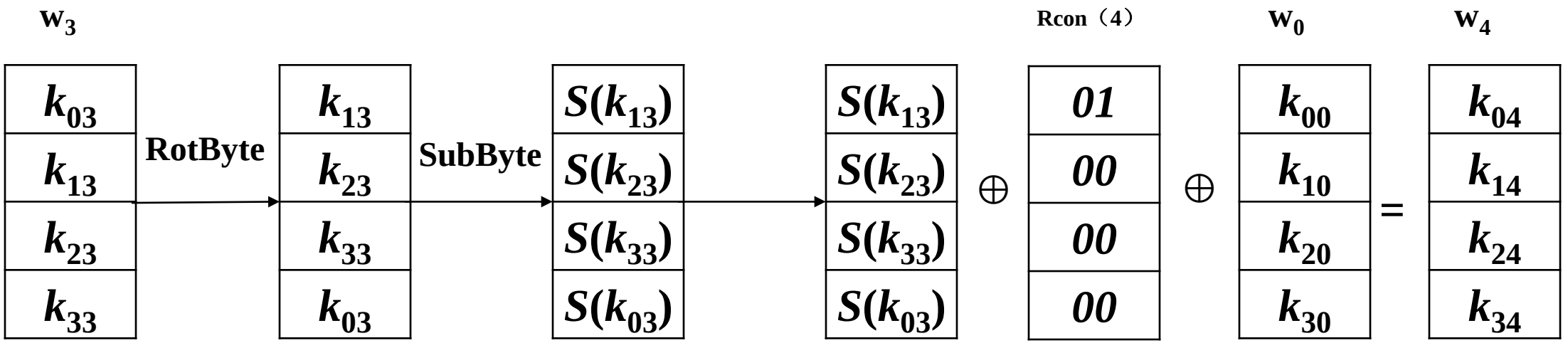
扩展算法（$N_k \leqslant 6$）

```
KeyExpansion (byteKey[4*Nk] , 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] );\\前4*Nk个个字节为种子密钥
   for (i =Nk; i <Nb*(Nr+1); i ++)
   {
        temp=W[i-1];
        if (i % Nk= =0)
                temp=SubByte (RotByte (temp)) ^ Rcon[i /Nk];\\W[i-1]与W[i-Nk]的异或
        W[i]=W[i-Nk] ^ temp;
   }
}
```

扩展算法（$N_k \leqslant 6$）（续）

w_0	w_1	w_2	w_3	w_4	w_5	……	
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}		

……



扩展算法（$N_k \leqslant 6$）（续）

w_0	w_1	w_2	w_3	w_4	w_5	……	
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}		

……

w_1		w_4		w_5
k_{01}		k_{04}		k_{05}
k_{11}	$\oplus$	k_{14}	$=$	k_{15}
k_{21}		k_{24}		k_{25}
k_{31}		k_{34}		k_{35}

扩展算法（N_k>6）

```
KeyExpansion (byte  Key[4*Nk] , 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=SubByte (RotByte (temp))^Rcon[i /Nk];
      else if (i % Nk==4)
                temp=SubByte (temp);
W[i]=W[i - Nk]^ temp;
   }
}
```

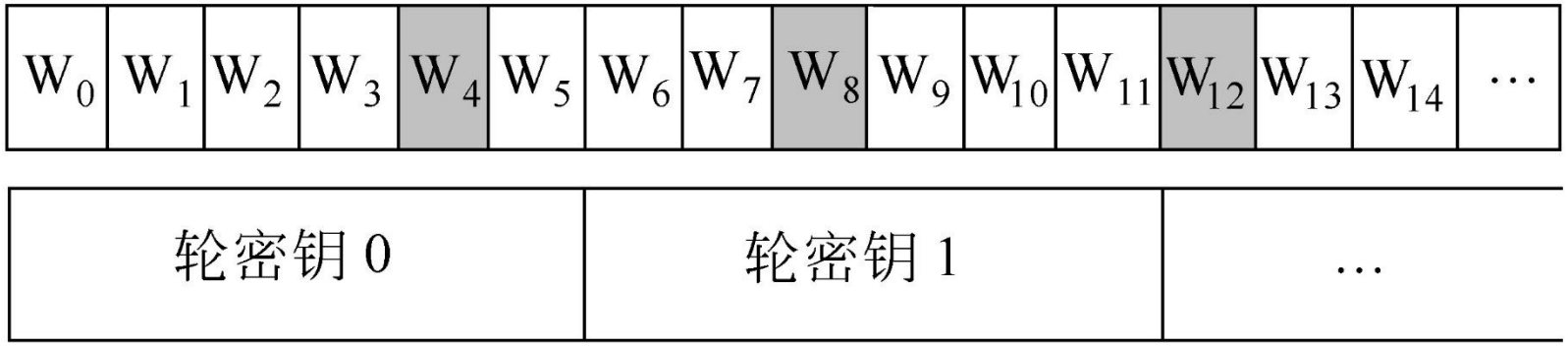
N_k>6与N_k≤6的密钥扩展算法的区别在于：当i-4为N_k的整数倍时，须先将前一个字W[i-1]经过SubByte变换（S盒）

轮常数

- 以上两个算法中，**Rcon[i/Nk]** 为轮常数，其值与**Nk**无关，定义为（字节用十六进制表示，同时理解为**GF(2^8)**上的元素）：
- **Rcon [i]=(RC[i], ‘00’, ‘00’, ‘00’)**
- 其中**RC[i]** 是**GF(2^8)** 中值为x^{i-1}的元素，因此
- **RC[1] =1(即 ‘01’)**
- **RC[i] = x(即 ‘02’)·RC[i-1]= x^{i-1}**

轮密钥选取

•轮密钥i（即第i 个轮密钥）由轮密钥缓冲字W[Nb* i]到W[Nb*(i+1)]给出，如图所示。



N_b=6且N_k=4时的密钥扩展与轮密钥选取

AES加密过程的伪代码

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*Nr+1])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[0, Nb-1])
  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  Out = state
end
```

AES解密过程的伪代码

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*Nr+1])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)
  end for
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[0, Nb-1])
  Out = state
end
```

AES密钥编排算法的伪代码

```
KeyExpansion (byte key[4*Nk] , word w[Nb*(Nr+1)], Nk)
begin
      word temp
      i=0
      while (i<Nk)
            w[i]=word(Key[4* i], Key[4* i +1], Key[4* i +2], Key[4* i +3] )
            i=i+1
       end while
       i=Nk
       while(i<Nb*(Nr+1))
            temp=W[i-1]
            if (I mod Nk= =0)
               temp=SubByte (RotByte (temp)) xor Rcon[i /Nk]
            else if (Nk>6 and i mod Nk = 4)
               temp=SubWord(temp)
             end if
               w[i]=w[i-Nk ] xor temp
         end while
end
```

SM4算法

SM4概况

- **SM4**分组密码算法是国家密码管理局于**2006**年**1**月**6**日公布的无线局域网产品使用的密码算法，是国内官方公布的第一个商用密码算法。
- **SM4**是一个分组密码算法，分组长度和密钥长度均为**128**比特。加密算法与密钥扩展算法都采用**32**轮非线性迭代结构。
- 它的解密算法与加密算法的结构相同，只是轮密钥的使用顺序相反，解密轮密钥是加密轮密钥的逆序。

SM4算法的术语说明

- Z_2^e 表示**e-**比特的向量集，Z_2^8 中的元素称为字节，Z_2^{32}中的元素称为字
- **S**盒是一个固定的**8**比特输入**8**比特输出的置换，记为$Sbox(.)$
- **SM4**中的采用了两个基本运算：$\oplus$，**32**比特异或；$<<<i$，**32**比特循环左移 ***i*** 位。

SM4算法的术语说明（续）

- SM4算法的加密密钥长度为128比特，表示为$\boldsymbol{MK} = (\boldsymbol{MK_0}, \boldsymbol{MK_1}, \boldsymbol{MK_2}, \boldsymbol{MK_3})$，其中，$\boldsymbol{MK_i}$为字。
- 轮密钥为$(\boldsymbol{rk_0}, \boldsymbol{rk_1}, \cdots, \boldsymbol{rk_3})$，$\boldsymbol{rk_i}$为字。轮密钥由加密密钥通过密钥扩展算法生成。
- $\boldsymbol{FK} = (\boldsymbol{FK_0}, \boldsymbol{FK_1}, \boldsymbol{FK_2}, \boldsymbol{FK_3})$为系统参数。
- $\boldsymbol{CK} = (\boldsymbol{CK}_0, \boldsymbol{CK_1}, \boldsymbol{CK_4}, \boldsymbol{CK_3})$为固定参数，用于密钥扩展算法。

SM4加密算法整体结构

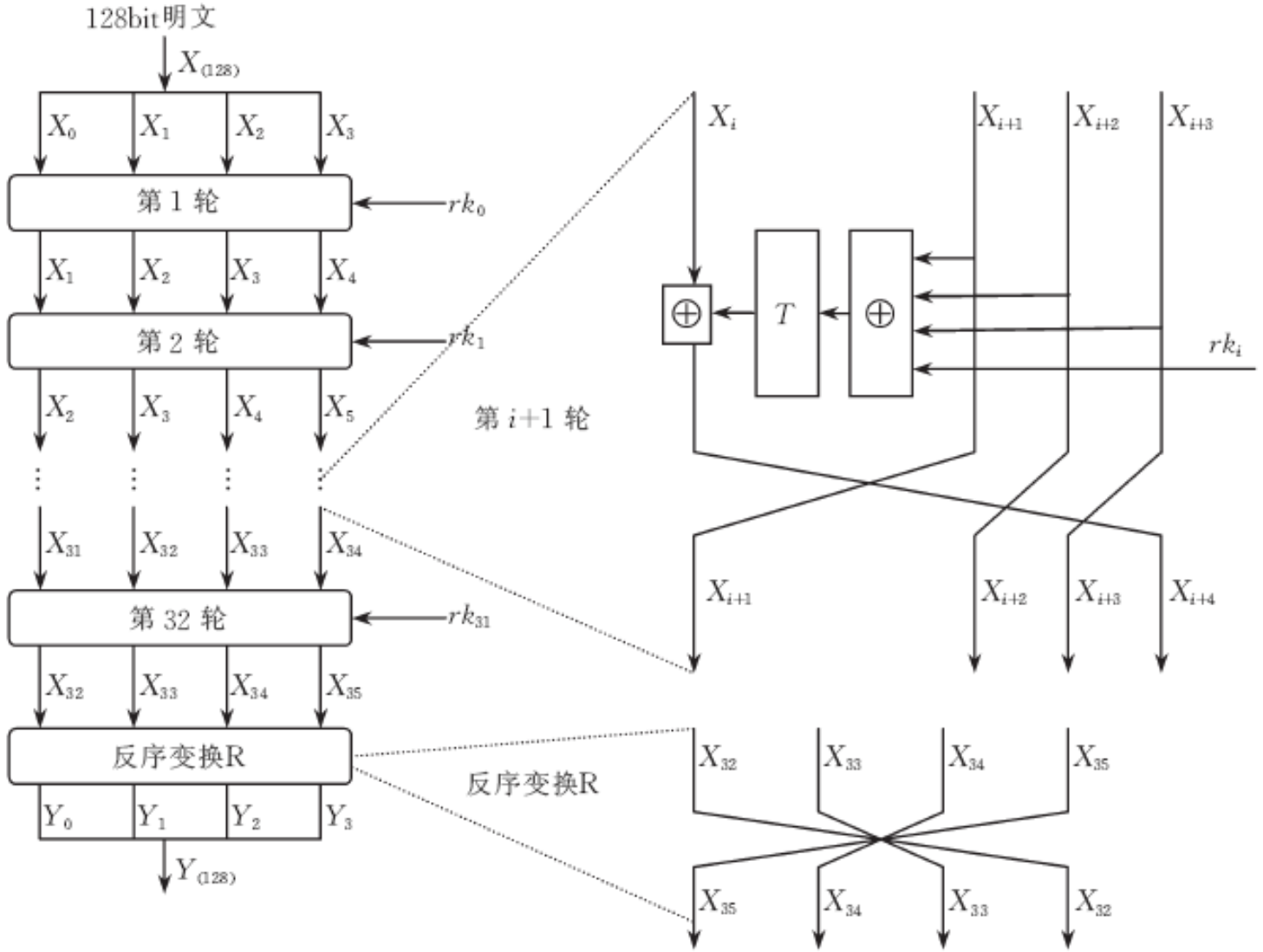


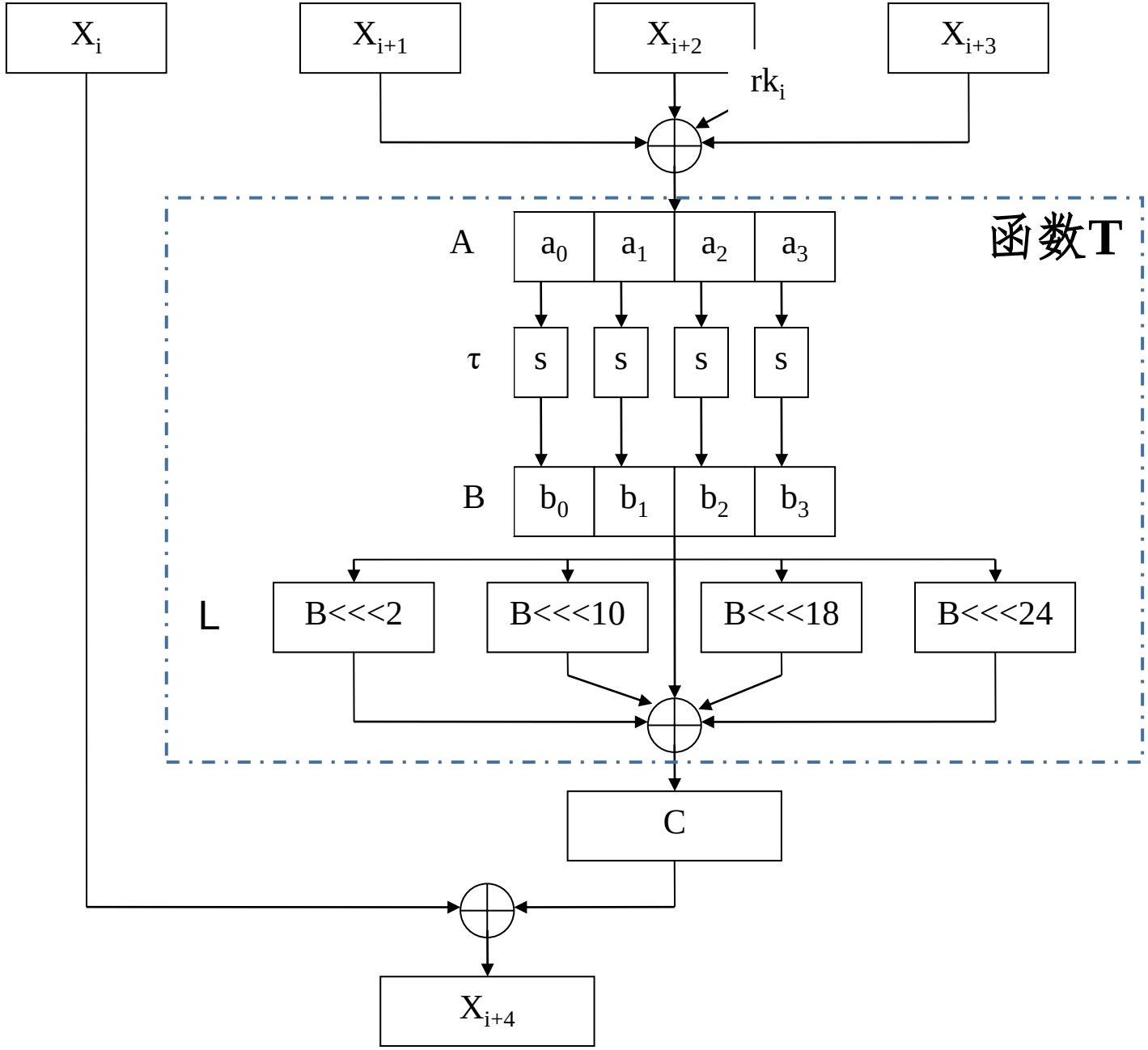
图 1　SMS4 加密算法整体结构

SM4 的轮函数

- 设输入为$(X_i, X_{i+1}, X_{i+2}, X_{i+3}) \in (Z_2^{32})^4$，轮密钥为$rk_i \in Z_2^{32}$，则轮函数为：

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)$$
$$= X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i), \quad i = 0, 1, \cdots, 31$$

- 其中$T: Z_2^{32} \to Z_2^{32}$称为合成置换，是一个由非线性变换和一个线性变换复合而成的可逆变换，即$T(.) = L(\tau(.))$。


X_i
X_{i+1}
X_{i+2}
X_{i+3}
rk_i
函数T
A
a_0 a_1 a_2 a_3
τ
s s s s
B
b_0 b_1 b_2 b_3
L
B<<<2
B<<<10
B<<<18
B<<<24
C
X_{i+4}

SM4的S盒

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
	1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
	2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
	3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
	4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
	5	68	6b	81	b2	71	64	da	8b	F8	eb	0f	4b	70	56	9d	35
	6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
	7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
	8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
	9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
	a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
	b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
	c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
	d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
	e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
	f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	D7	cb	39	48

SM4的S盒说明

- 非线性变换 τ 中所使用的S盒是一个具有很好密码学特性的、由8比特输入产生8比特输出的置换
- 在设计原理上，SMS4比AES的S盒设计多了一个仿射变换
- 即

$$\boldsymbol{y} = \boldsymbol{A}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B})^{-1} + \boldsymbol{B}$$

- SMS4有很高的灵活性，所采用的S盒可以灵活地被替换，以应对突发性的安全威胁。

- Analysis of the SMS4 Block Cipher. Fen Liu, Wen Ji, Lei Hu, Jintai Ding. ACISP 2007.

SM4的加密算法和解密算法

- 设明文输入为 $(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$
- 密文为 $rk_i \in Z_2^{32}$，轮密钥为 $(Y_0, Y_1, Y_2, Y_3) \in (Z_2^{32})^4$。加密变换为：

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)$$

$$= X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i), \quad i = 0, 1, \cdots, 31$$

$$(Y_0, Y_1, Y_2, Y_3) = (X_{35}, X_{34}, X_{33}, X_{32})$$

- SM4算法的解密变换和加密变换结构相同，不同的仅是轮密钥的使用顺序。
  - 加密时轮密钥的使用顺序为 $(rk_0, rk_1, \cdots, rk_{31})$
  - 解密时轮密钥的使用顺序为 $(rk_{31}, rk_{30}, \cdots, rk_0)$

SM4解密的合理性

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i) = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i), \quad i = 0, 1, \cdots, 31$$

$$(Y_0, Y_1, Y_2, Y_3) = (X_{35}, X_{34}, X_{33}, X_{32})$$

$$X_{35} = X_{31} \oplus T(X_{34} \oplus X_{33} \oplus X_{32} \oplus rk_{31})$$

$$\begin{aligned} Y_4 &= F(Y_0, Y_1, Y_2, Y_3, rk_{31}) \\ &= Y_0 \oplus T(Y_1 \oplus Y_2 \oplus Y_3 \oplus rk_{31}) \\ &= X_{35} \oplus T(X_{34} \oplus X_{33} \oplus X_{32} \oplus rk_{31}) \\ &= X_{31} \oplus T(X_{34} \oplus X_{33} \oplus X_{32} \oplus rk_{31}) \oplus T(X_{34} \oplus X_{33} \oplus X_{32} \\ &= X_{31} \end{aligned}$$

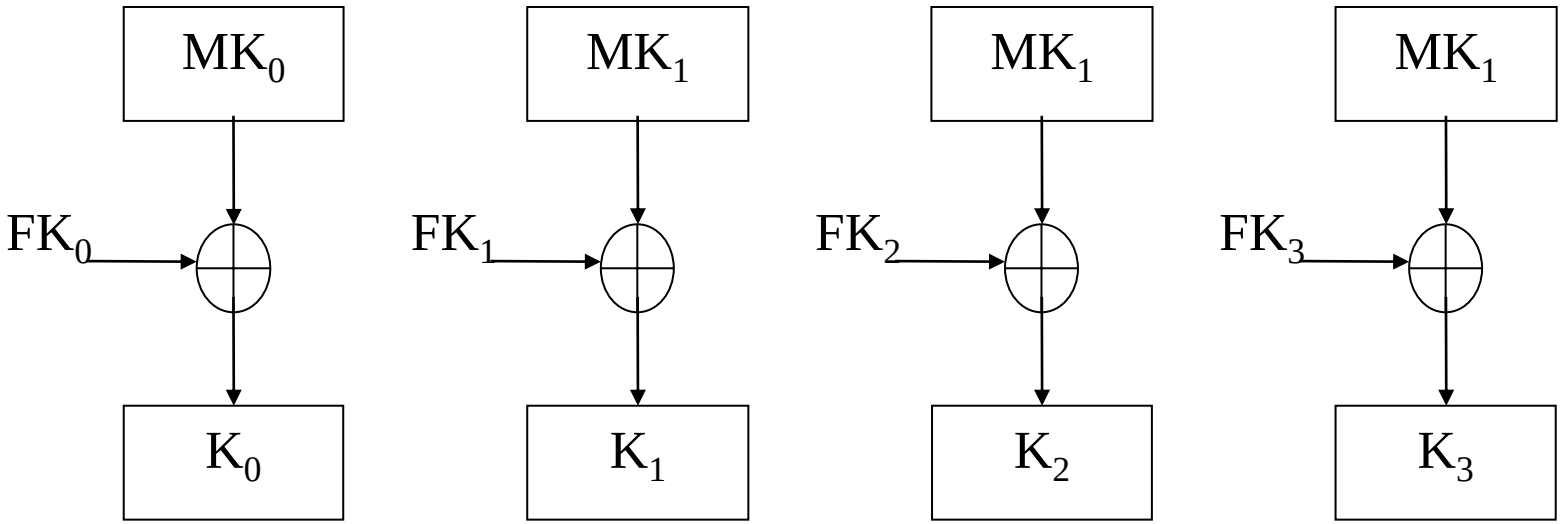
SM4的密钥扩展算法

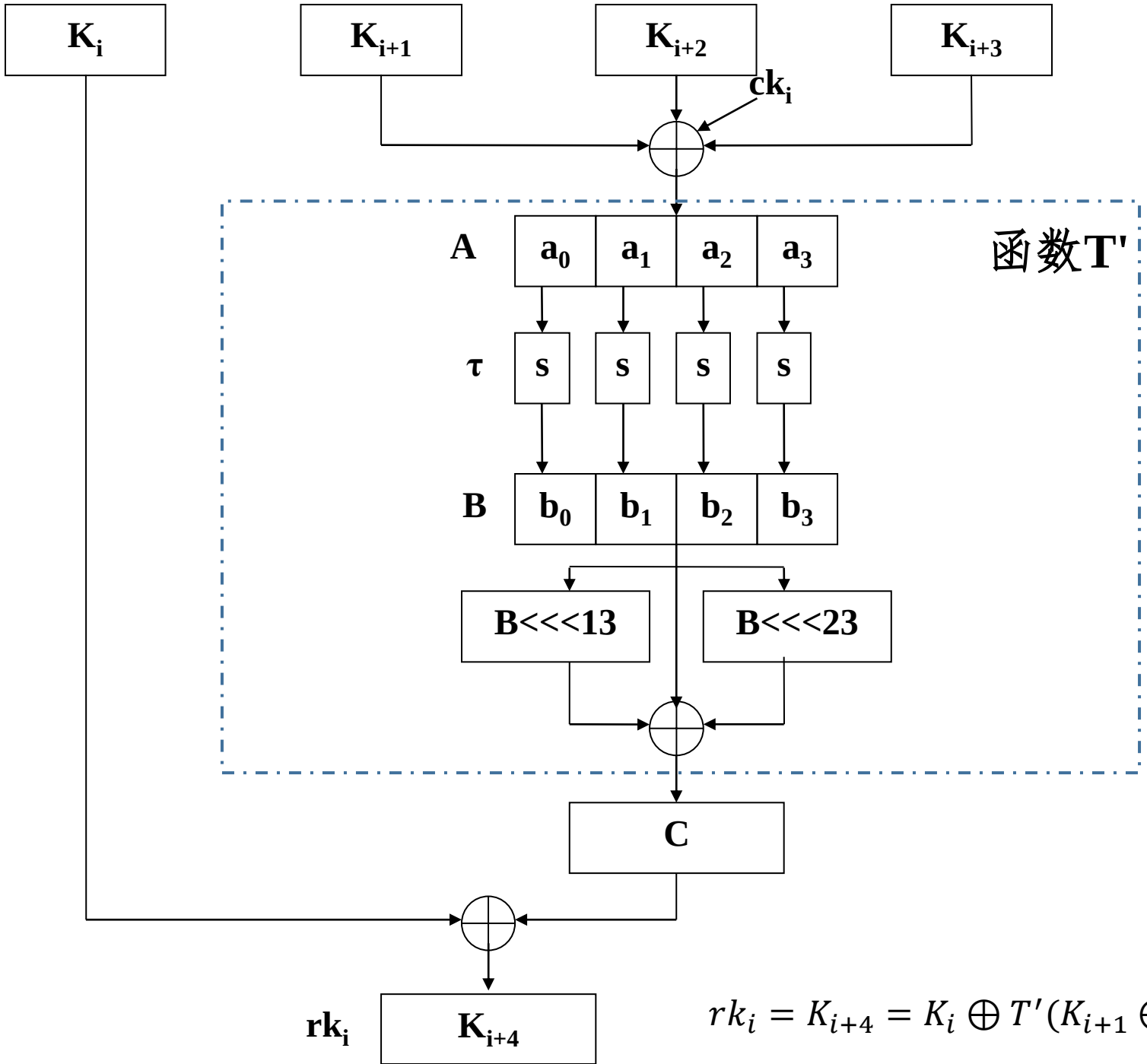
- 设加密密钥$MK=(MK_0,MK_1,MK_2,MK_3)$，其中MK_i为字。
- 轮密钥为$(rk_0,rk_1,\cdots,rk_{31})$
- 轮密钥的生成方法具体为：

$$(K_0,K_1,K_2,K_3)$$
$$=(MK_0\oplus FK_0,MK_1\oplus FK_1,MK_2\oplus FK_2,MK_3\oplus FK_3)$$
$$rk_i=K_{i+4}=K_i\oplus T'(K_{i+1}\oplus K_{i+2}\oplus K_{i+3}\oplus CK_i)$$

SM4的密钥扩展算法（续）

$$(K_0, K_1, K_2, K_3) = (MK_0 \oplus FK_0, MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3)$$





$$rk_i = K_{i+4} = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i)$$