上讲主要内容

- 密码学的基本知识
- 香农理论
- 密码分析学的基本知识
- 密码系统的安全性

分组密码

主讲人：马春光

machunguang@hrbeu.edu.cn

本讲主要内容

- 分组密码的简介
- **DES**密码算法
- **AES**密码算法
- 分组密码的操作方式

分组密码的简介

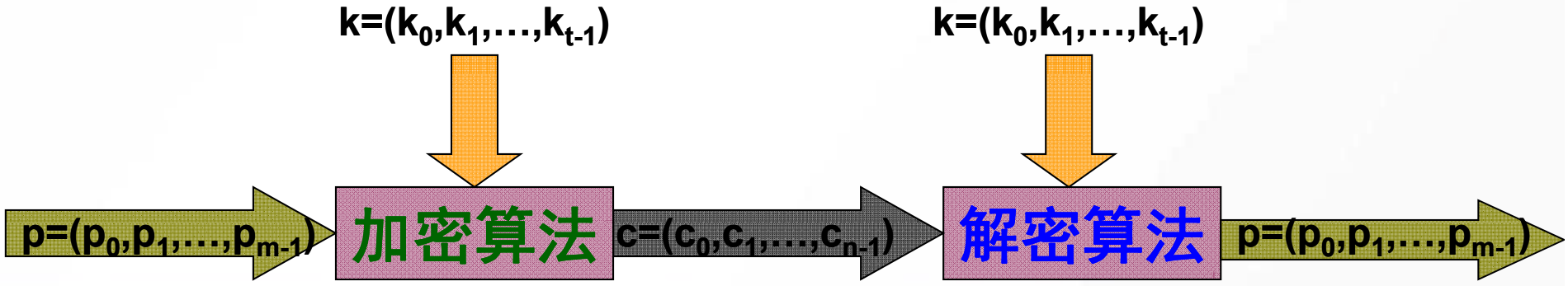
- 分组密码的含义
- 分组密码的设计思想
- 分组密码的设计准则

分组密码的概述

- 分组密码(block cipher)是现代密码学中的重要体制之一，也是应用最为广泛、影响最大的一种密码体制，其主要任务是提供数据保密性，也可以用到在许多方面，如构造伪随机数生成器、序列密码、认证码和哈希函数等。
- 分组密码又分为对称分组密码和非对称分组密码，习惯上，分组密码一词在很多场合指的是对称分组密码，简称分组密码。
- 由于分组密码加解密速度较快，安全性好，以及得到许多密码芯片的支持，现代分组密码发展非常快，在许多研究和应用领域得到了广泛的应用。

分组密码的含义

也称块密码，它是将明文消息经编码表示后的二进制序列$p_0,p_1,\ldots,p_i,\ldots$划分成若干固定长度(譬如m)的组(或块)$p=(p_0,p_1,\ldots,p_{m-1})$，各组分别在密钥$k=(k_0,k_1,\ldots,k_{t-1})$的控制下转换成长度为n的密文分组$c=(c_0,c_1,\ldots,c_{n-1})$。其本质是一个从明文空间（m长的比特串的集合）P到密文空间(n长的比特串的集合)C的一一映射。(一般而言，m=n)



分组密码的置换

- 对于一个分组长度为n的分组密码，不同的密钥对应不同的加密解密变换，即，对于给定的密钥k，加密变换E_k是$GF(2)^n$上的一个置换，解密变换D_k是E_k的逆变换。
- 如果密钥长度为t，则密钥量为2^t。因为$GF(2)^n$上共有$2^n!$个不同的置换，所以必须有$2^t \leq 2^n!$。为了便于密钥管理，通常密钥长度t不能太大。当然，密钥长度t不能太小，否则，难以抵抗对密钥的穷举搜索攻击。

分组密码的要求

- 分组长度要足够大
- 密钥量要足够大
- 密码变换足够复杂
- 加密和解密运算简单
- 无数据扩展或压缩

当分组长度较小时，攻击者通过穷举明文空间，得到密码变换规律，难于抵御选择明文攻击。

如果密钥量小，攻击者可以有效地通过穷举密钥，对密文进行解密，以得到有意义的明文，难于抵御唯密文攻击。

使攻击者除了穷举法攻击以外，找不到其他简洁的数学破译方法。

便于软件和硬件实现，性能好。

分组密码的设计思想

- 扩散
- 混乱

扩散

所谓扩散，是指要将算法设计成明文每一比特的变化尽可能多地影响到输出密文序列的变化，以便隐蔽明文的统计特性。形象地称为雪崩效应。

扩散的另一层意思是密钥每一位的影响尽可能迅速地扩展到较多的密文比特中去。即扩散的目的是希望密文中的任一比特都要尽可能与明文、密钥相关联，或者说，明文和密钥中任何一比特值发生改变，都会在某种程度上影响到密文值的变化，以防止将密钥分解成若干个孤立的小部分，然后各个击破。

扩散的举例说明

明文 密文

00000000 → xxxxxx01

00000001 → xxxxxx11

无扩散技术的加密

00000000 → 01011010

00000001 → 11101011

有扩散技术的加密

混乱

所谓混乱，是指在加解密变换过程中是明文、密钥以及密文之间的关系尽可能地复杂化，以防密码破译者采用解析法(即通过建立并求解一些方程)进行破译攻击。

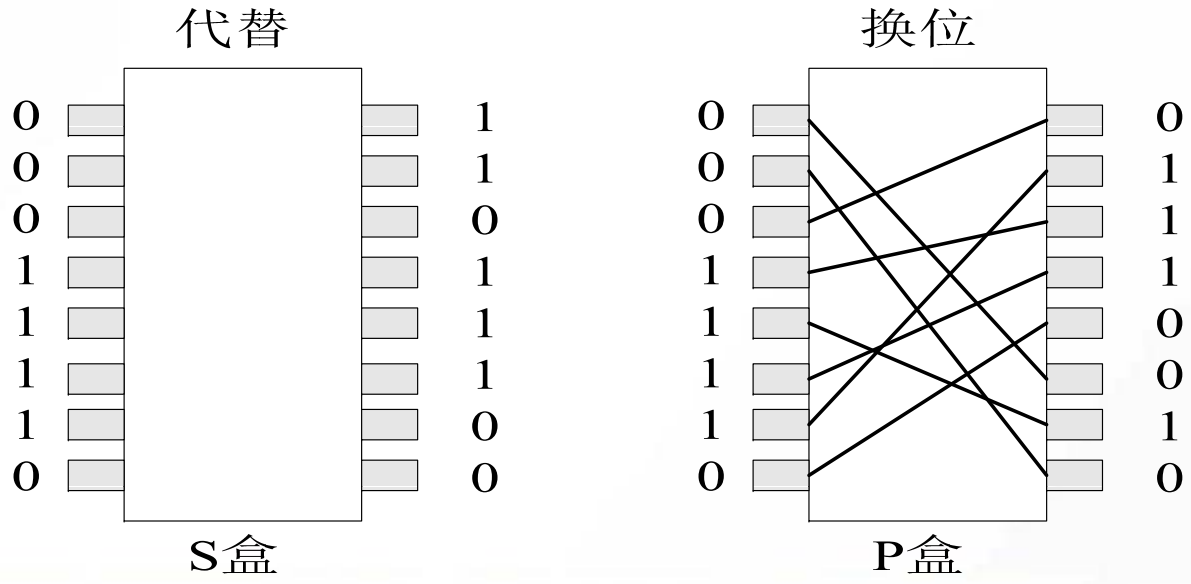
混乱可以用“搅拌机”来形象地解释，将一组明文和一组密钥输入到算法中，经过充分混合，最后变成密文。同时要求，执行这种“混乱”作业的每一步都必须是可逆的，即明文混乱以后能得到密文，反之，密文经过逆向的混乱操作以后能恢复出明文。（按照混乱原则，分组密码算法应有复杂的非线性因素）

分组密码原理---乘积密码

依次使用两个或两个以上的基本密码，所得结果的密码强度将强于所有单个密码的强度，即乘积密码是扩散和混乱两种基本密码操作的组合变换，这样能够产生比各自单独使用时更强大的密码系统。选择某个较为简单的密码变换(包含多个基本密码)，在密钥控制下以迭代方式多次利用它进行加密变换，就可以实现预期的扩散和混乱效果。

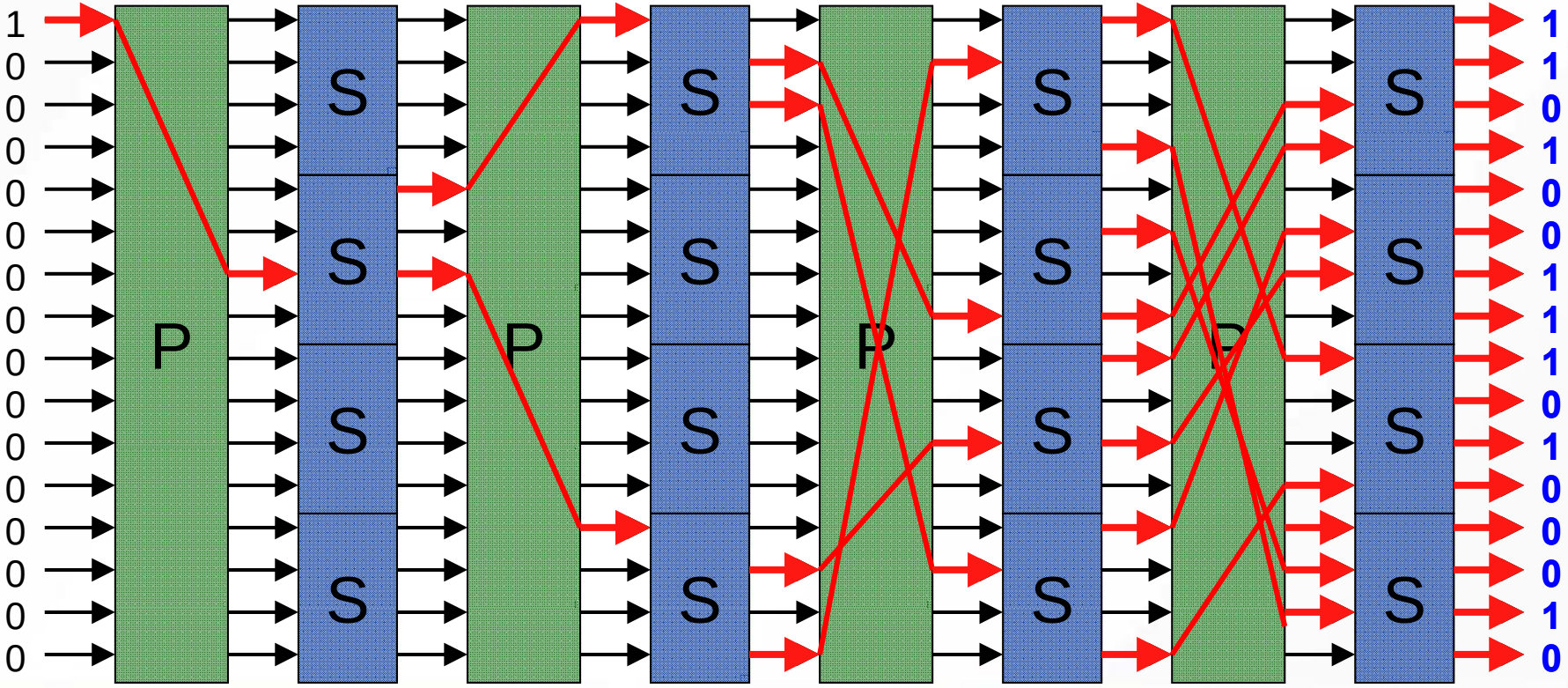
乘积密码的实现---SP网络

SP网络是由多重S变换和P变换组合成的变换网络，它是乘积密码的一种。其基本操作是S变换(代替)和P变换(换位)，前者称为S盒，后者称为P盒。S盒起到混乱作用，P盒起到扩散的作用。SP网络的构造及S盒、P盒的构造如下图所示：

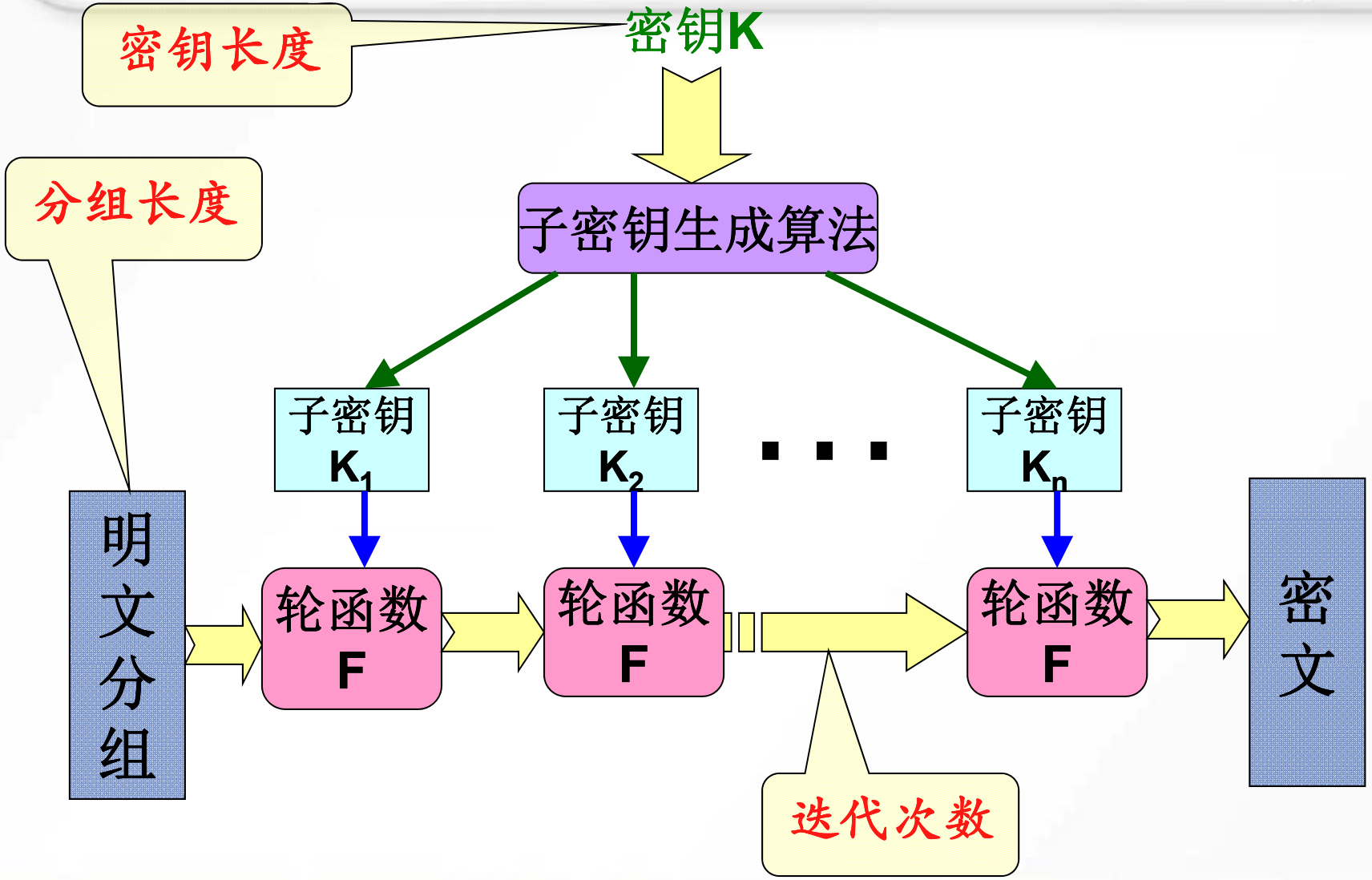


SP网络的性质

- SP网络具有雪崩效应。所谓雪崩效应是指，输入(明文或密钥)即使只有很小的变化，也会导致输出产生巨大的变化现象。



分组密码简图


密钥长度
密钥K
分组长度
子密钥生成算法
子密钥 K_1
子密钥 K_2
子密钥 K_n
明文分组
轮函数 F
轮函数 F
轮函数 F
密文
迭代次数

分组密码的设计准则

- 分组长度(能够抵御选择明文攻击)
- 密钥长度(能够抵御唯密文攻击)
- 轮函数F的设计准则
- 子密钥的生成方法
- 迭代的轮数

轮函数F的设计准则

轮函数F是分组密码的核心，是分组密码中单轮加解密函数，其基本准则：

- 非线性(主要依赖S盒)；
- 可逆性(能够实现解密)；
- 雪崩效应；

其主要性能指标是安全性、速度、灵活性。

子密钥的生成方法

子密钥的生成也是迭代分组算法的一个重要组成部分，是从初始(种子)密钥产生迭代的各轮要使用的子密钥的算法。也就是说，轮函数F的功能是在子密钥的参与和控制下实现的，子密钥的生成很重要，其评价指标：

- 实现简单、速度满足要求；
- 种子密钥的所有比特对每个子密钥比特的影响应大致相同；
- 没有弱密钥或弱密钥容易确定；

迭代的轮数

分组密码一般采用简单的、安全性弱的加密函数进行多轮迭代运算，使得安全性增强。一般来说，分组密码迭代轮数越多，密码分析越困难，但也不是追求迭代轮数越多越好，过多迭代轮数会使加解密算法的性能下降，而安全性增强不明显。 决定迭代轮数的准则：使密码分析的难度大于简单穷举搜索攻击的难度。分组密码迭代轮数一般采用**8，10，12，16，20**的居多。

DES算法

- DES算法的简介
- DES算法的实现
- DES算法的安全性
- 多重DES算法

DES简介

1973年，美国的国家标准局(National Bureau of Stand -ards,NBS)认识到建立数据加密标准的迫切性，开始征集联邦数据加密标准。有很多公司着手这项工作并提交了一些建议，最后IBM公司的Lucifer加密系统获得了胜利。经过两年多的公开讨论之后，1977年1月15日NBS决定利用这个算法，并将其更名为数据加密标准(Data Encryption Stand-ards，DES)。不久，其他组织也认可和采用DES作为加密算法，供商业和非国防性政府部分使用。当时，确定有效期为5年，随后在1983年、1988年、1993年三次再度授权该算法续用五年，1997年开始征集AES(高级加密标准)，2000年选定比利时人设计的Rijndael算法作为新标准。

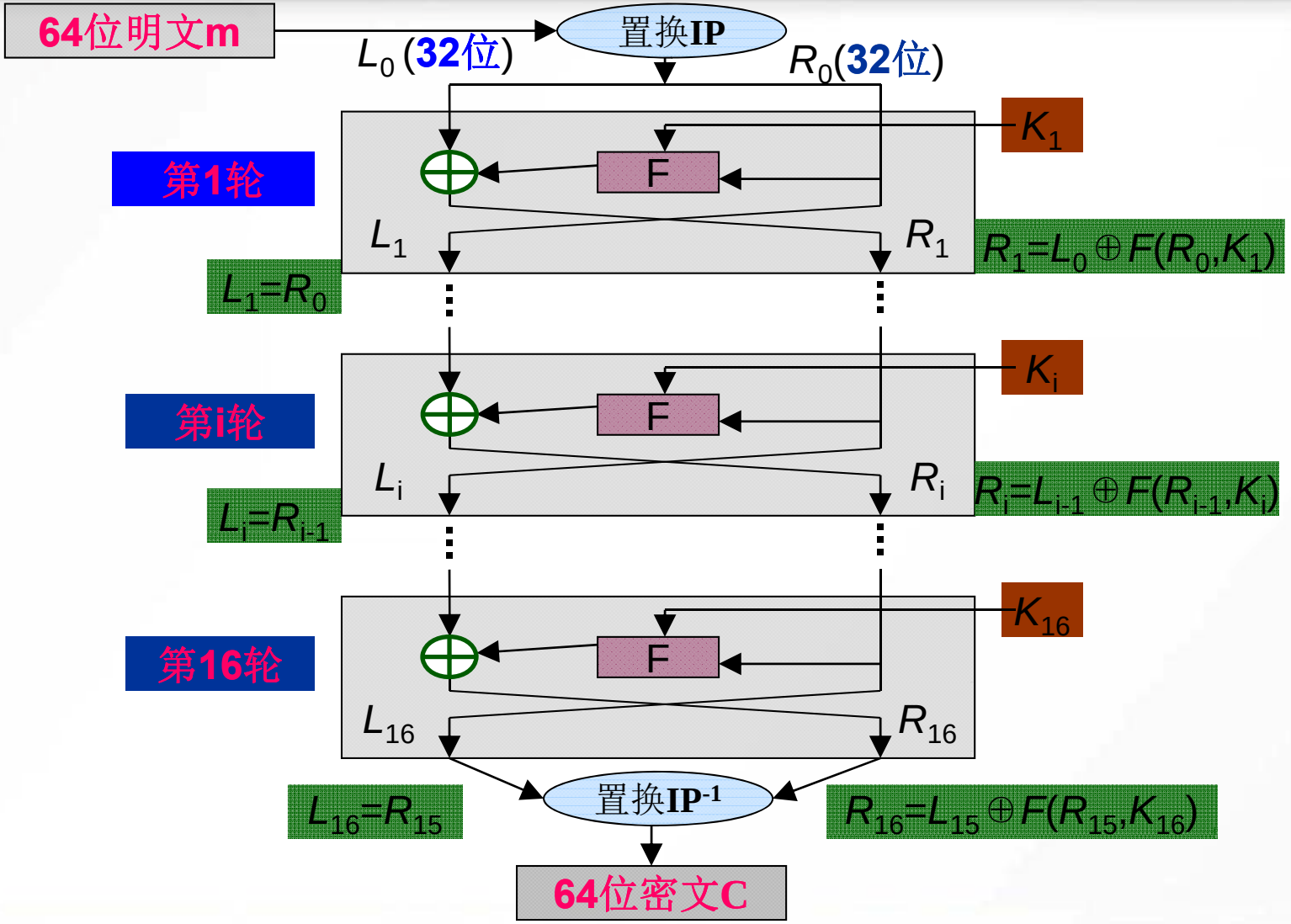
公开征集密码算法标准的主要要求

- 算法必须是安全的(具有加密保护信息安全的能力)；
- 算法必须是公开的(有完整的算法说明、容易理解、能为所有用户使用)；
- 能够经济、有效的硬件实现(性能好)；
- 能够得到批准(合法)；
- 可出口(大众化)；

DES概述

- 分组加密算法：明文和密文为64位分组长度。
- 对称算法：加密和解密除密钥编排不同外，使用同一算法。
- 密钥长度：56位，但存在弱密钥，容易避开。
- 采用混乱和扩散的组合，每个组合先替代后置换，共16轮。
- 只使用了标准的算术和逻辑运算，易于实现。
- 现代密码学诞生的标志之一，揭开了商用密码研究的序幕。

DES加密流程图


64位明文m
置换IP
L_0 (32位)
R_0(32位)
K_1
第1轮
F
L_1
R_1
$L_1=R_0$
$R_1=L_0 \oplus F(R_0,K_1)$
K_i
第i轮
F
L_i
R_i
$L_i=R_{i-1}$
$R_i=L_{i-1} \oplus F(R_{i-1},K_i)$
K_{16}
第16轮
F
L_{16}
R_{16}
$L_{16}=R_{15}$
置换IP-1
$R_{16}=L_{15} \oplus F(R_{15},K_{16})$
64位密文C

DES加密过程的公式化描述

初始置换

迭代次数

$$L_0 R_0 \leftarrow IP(<64bit\text{明文}>)$$

$$L_i \leftarrow R_{i-1} \qquad i = 1, 2, \cdots, 16$$

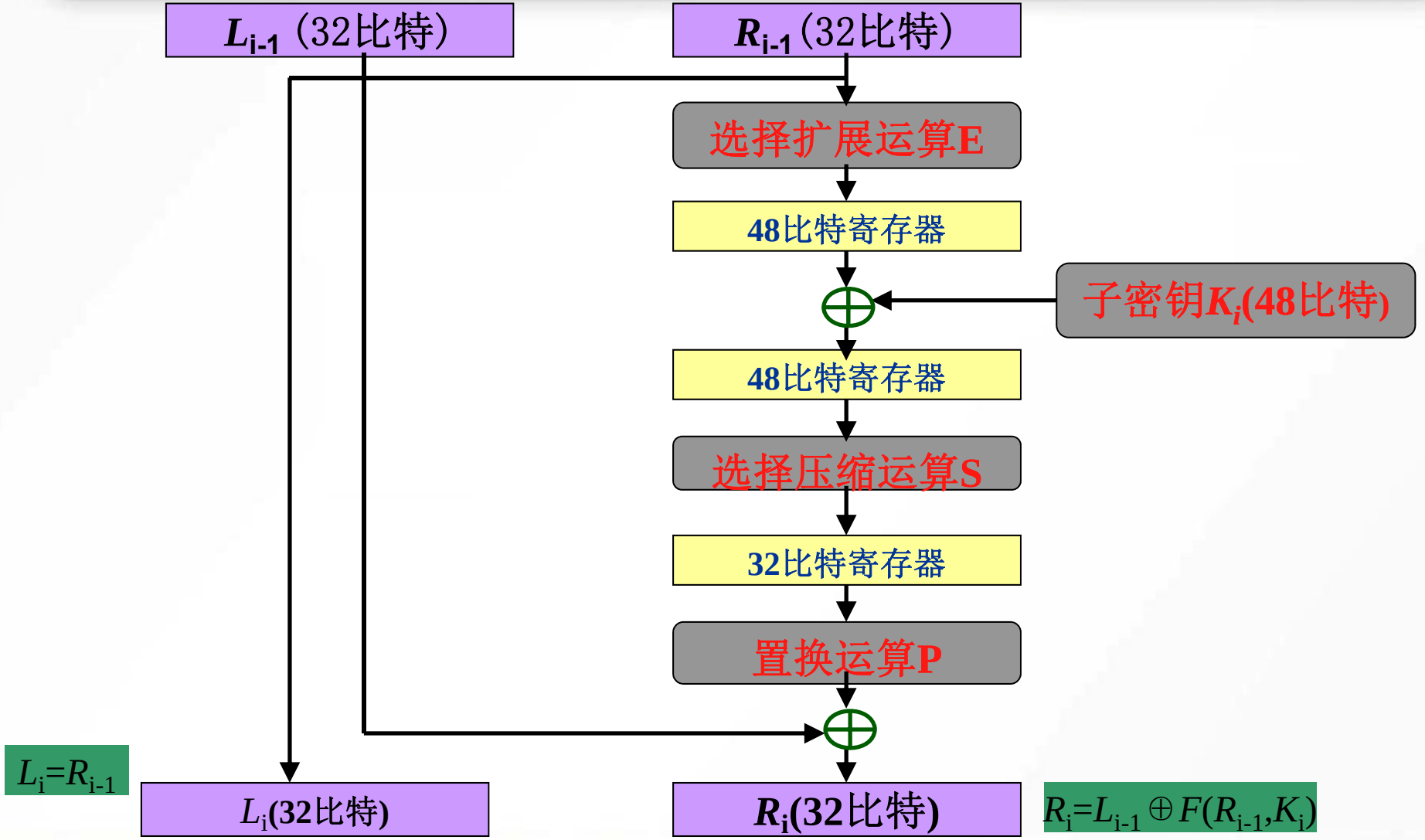
$$R_i \leftarrow L_{i-1} \oplus F(R_{i-1}, k_i) \qquad i = 1, 2, \cdots, 16$$

$$<64bit\text{密文}> \leftarrow IP^{-1}(R_{16} L_{16})$$

逐位模2求和

轮函数

DES一轮的实现流程图

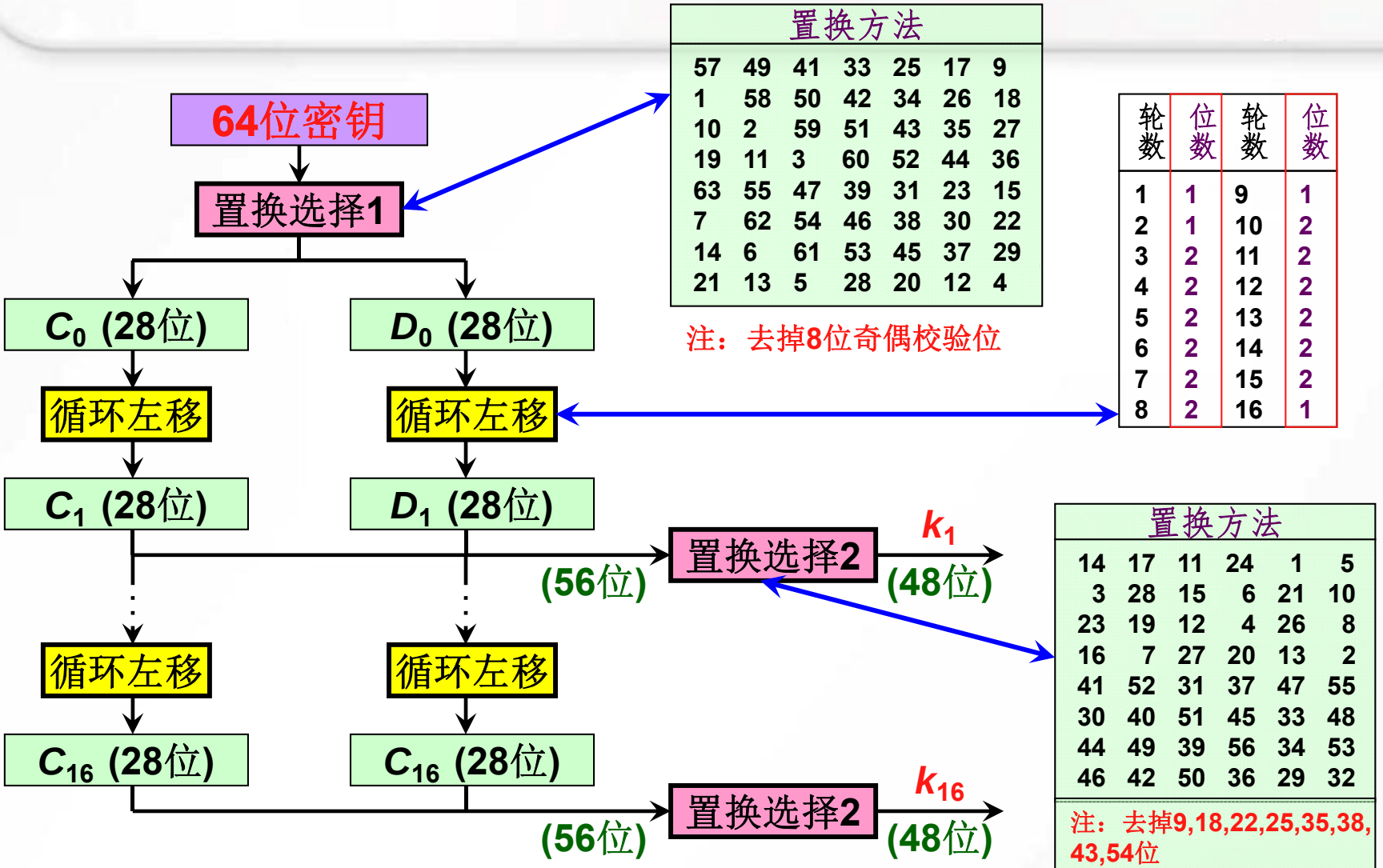


选择扩展置换E (32位扩展到48位)

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

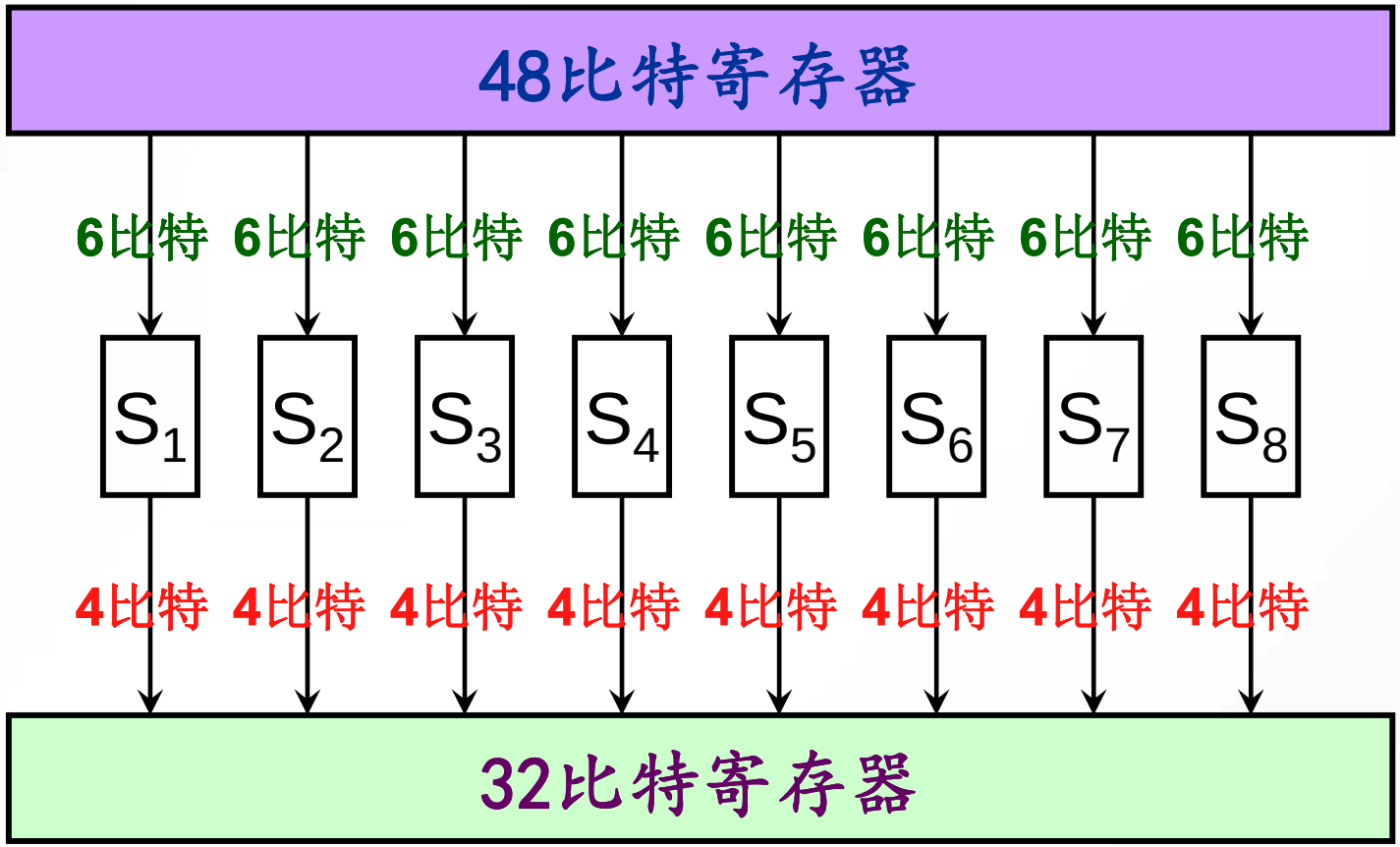
32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

DES子密钥的生成



注：密钥各位在子密钥出现次数基本相同(12次至15次)

压缩替代S-盒(48位压缩到32位)



S盒的规则

S-盒1

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

S-盒2

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

S-盒3

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

S-盒4

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

S-盒5

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

S-盒6

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

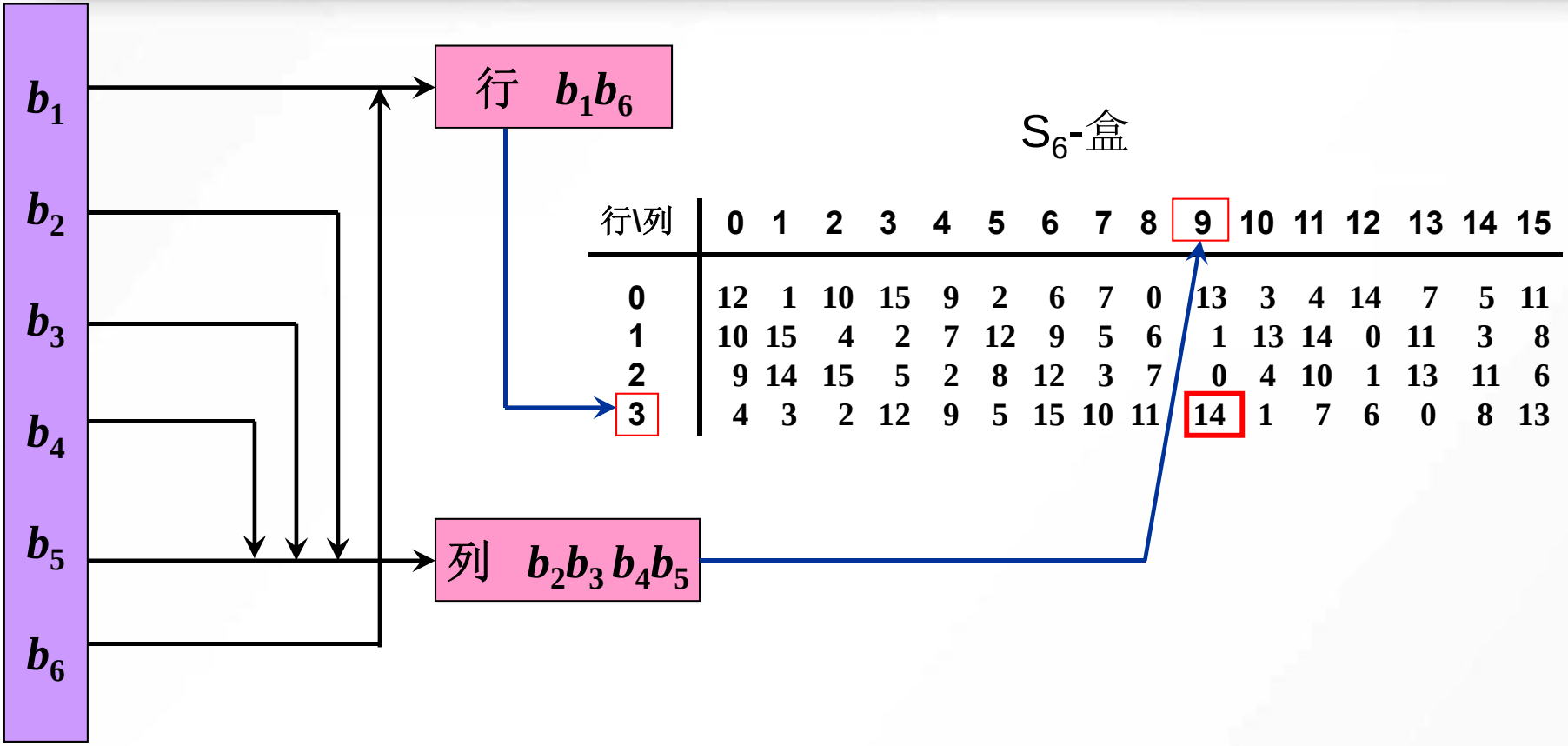
S-盒7

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

S-盒8

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

S-盒的构造方法(举例)



S_6-盒

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

$b_1b_2b_3b_4b_5b_6$ → 110011

→ 行:$b_1b_6=11_2=3$；列:$b_2b_3b_4b_5=1001=9$

→ S_6-盒 3行9列，值：14=1100

置换p-盒的构造方法

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的核心技术

- S盒设计准则
- P盒设计准则

S盒设计准则

- 具有良好的非线性；（输出的每一个比特与全部输入比特有关）
- 每一行包括所有16种4位二进制；
- 两个输入相差1bit时，输出相差2bit；
- 如果两个输入刚好在中间两个比特上不同，则输出至少有两个比特不同；
- 如果两个输入前两位不同而最后两位相同，则输出一定不同；
- 相差6bit的输入共有32对，在这32对中有不超过8对的输出相同；

P盒设计准则

- 每个S盒的4位输出影响下一轮6个不同的S盒，但是没有两位影响同一S盒;
- 在第i轮S盒的4位输出中，2位将影响i+1轮中间位，其余2位将影响两端位;
- 如果一个S盒的4位输出影响另一个S盒的中间的1位，则后一个的输出位不会影响前面一个S盒的中间位;

DES的解密算法

密文 $C = L_{16} R_{16}$

$L_{16} = R_{15}$

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

$\therefore L_{15} = R_{16} \oplus F(R_{15}, K_{16})$

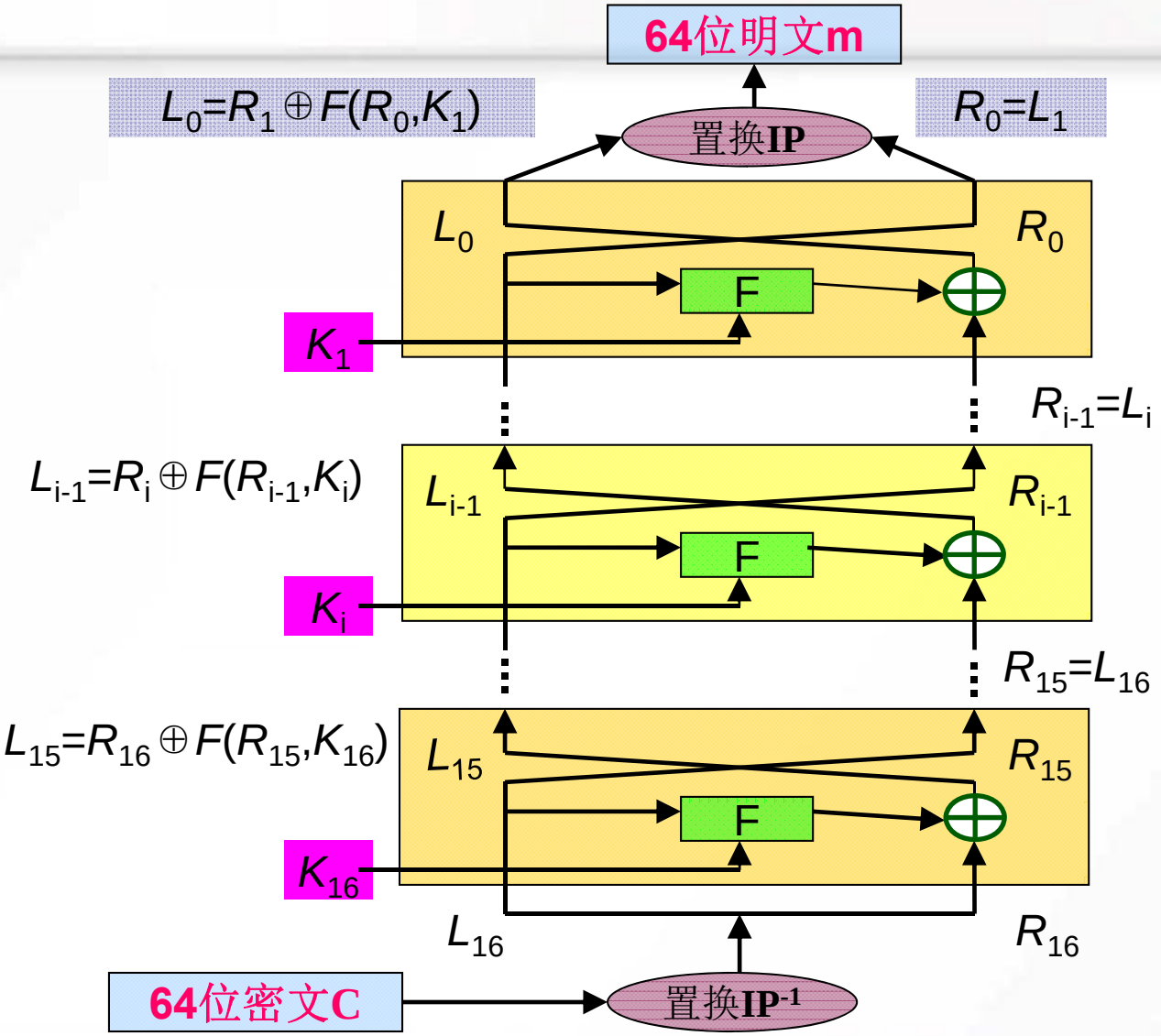
$R_{15} = L_{16}$

可得解密一般式 ：

$R_{i-1} = L_i$

$L_{i-1} = R_i \oplus F(L_i, K_i) \quad (i = 16, 15, ..., 1)$

DES解密流程图


64位明文m
$L_0=R_1 \oplus F(R_0,K_1)$
置换IP
$R_0=L_1$
L_0
R_0
F
K_1
$R_{i-1}=L_i$
$L_{i-1}=R_i \oplus F(R_{i-1},K_i)$
L_{i-1}
R_{i-1}
F
K_i
$R_{15}=L_{16}$
$L_{15}=R_{16} \oplus F(R_{15},K_{16})$
L_{15}
R_{15}
F
K_{16}
L_{16}
R_{16}
64位密文C
置换IP^{-1}

DES的安全性

- 互补性
- 弱密钥和半弱密钥
- 密钥搜索
- 差分分析和线性分析

互补性

设$F = R \oplus K, F' = \overline{R} \oplus \overline{K}$,证明$F' = F$.

证:

$F = R \oplus K$

$F' = \overline{R} \oplus \overline{K}$

等式两边相加:

$F \oplus F' = 0$

$\therefore F' = F$

设$S = L \oplus F, S' = \overline{L} \oplus F$,证明$S' = \overline{S}$.

证:

$S = L \oplus F$

$S' = \overline{L} \oplus F$

等式两边相加:

$S \oplus S' = 1$

$\therefore S' = \overline{S}$

对明文 m逐位取补 ,记为 $\overline{m}$, 密钥 k逐位取补 ,记为 $\overline{k}$, 若 $c = E_k(m)$, 则有 $\overline{c} = E_{\overline{k}}(\overline{m})$.

这种特性被称为算法上的互补性，是由算法中的两次异或运算的配置所决定的。两次异或运算一次在**S**盒之前，一次在**P**盒置换之后。若对**DES**输入的明文和密钥同时取补，则选择扩展运算**E**的输出和子密钥产生器的输出也都取补，因而经异或运算后的输出和明文及密钥未取补时的输出一样，这使得到达**S**盒的输入数据未变，其输出自然也不会变，但经第二个异或运算时，由于左边的数据已取补，因而输出也就取补了。

互补性会使DES在选择明文攻击下所需的工作量减半

在选择明文攻击下，可得：

$$c_1 = E_k(m) \qquad ①$$

$$c_2 = E_k(\overline{m}) \qquad ②$$

根据互补性由②得：$\overline{c_2} = E_k(m)$ ③

根据①式穷举搜索密钥 k 时，若输出密文是 c_1，则加密密钥就是所应用的密钥；若输出密文是 $\overline{c_2}$，根据③可知加密密钥是所应用的密钥的补；这样，利用一个密钥的加密尝试，能够检测两个密钥是否为真正的加密密钥。因此，DES 的互补性会使 DES 在选择明文攻击下所需的工作量减半。

弱密钥

- 如果给定初始密钥k，经子密钥产生器产生的各个子密钥都相同，即有$k_1=k_2=\ldots=k_{16}$，则称给定的初始密钥k为**弱密钥**。
- 若k为弱密钥，则对任意的64bit信息有：$E_k(E_k(m))=m$和$D_k(D_k(m))=m$。
- 弱密钥的构造由子密钥产生器中寄存器C和D中的存数在循环移位下出现重复图样决定的（C和D中的存数为全0或全1）。共有4个（十六进制）：

01 01 01 01 01 01 01 01

1F 1F 1F 1F 0E 0E 0E 0E

E0 E0 E0 E0 1F 1F 1F 1F

FE FE FE FE FE FE FE FE

半弱密钥

- 若给定初始密钥k，产生的16个子密钥只有两种，且每种都出现8次，则称k为半弱密钥。
- 半弱密钥的特点是成对出现，且具有下述性质：

  若k_1和k_2为一对弱密钥，m为明文组，则有：

  $E_{k2}(E_{k1}(m))=E_{k1}(E_{k2}(m))=m$。
- 半弱密钥表见下页；
- 此外，还有四分之一弱密钥和八分之一弱密钥等等；
- 在DES的2^{56}(72057594037927936)个密钥中，弱密钥(256)所占的比例是非常小的，而且极易避开，因此，弱密钥的存在对DES的安全性威胁不大。

半弱密钥表

C,D存数编号	初始密钥（十六进制表示）	
（10，10） （5，5）	01 FE 01 FE 01 FE 01 FE FE 01 FE 01 FE 01 FE 01	互逆对
（10，5） （5，10）	1F E0 1F E0 0E F1 0E F1 E0 1F E0 1F F1 0E F1 0E	互逆对
（10，0） （5，0）	01 E0 01 E0 01 F1 01 F1 E0 01 E0 01 F1 01 F1 01	互逆对
（10，15） （5，15）	1F FE 1F FE 0E FE 0E FE FE 1F FE 1F FE 0E FE 0E	互逆对
（0，10） （0，5）	01 1F 01 1F 01 0E 01 0E 1F 01 1F 01 0E 01 0E 01	互逆对
（15，10） （15，5）	E0 FE E0 FE F1 FE F1 FE FE E0 FE E0 FE F1 FE F1	互逆对

密钥搜索

2^56=7.205*10^16.86

- **DES的强度：56比特的密钥长度**
  - 如果每秒搜索10^6个密钥，大约需要100年。
  - 如果采用并行技术，97年10万美金的机器（5760个搜索$5X10^7$芯片）可以在1.5天用穷举法找到DES密钥。
  - 密钥分割的方法，利用Internet的分布式计算能力，组织志愿军连接了100000台计算机系统，在22小时15分钟完成DES算法的攻击。
- **最近的一次评估是在1994年1月，当时决定1998年12月以后，DES不再作为联邦加密标准。**
- **AES (128位)取代DES。**

差分分析

选择明文攻击：加密算法，截获的部分密文，自己选择的明文消息及由密钥产生的相应密文

差分分析是一种攻击迭代密码体制的选择明文攻击方法，与一般统计分析法的不同之处是，它不是直接分析密文或密钥和明文的统计相关性，而是分析一对给定明文的异或（称为差分）与对应密文对的异或之间的统计相关性。差分分析的基本思想是在要攻击的迭代密码系统中找出某些高概率的明文差分和密文差分对来推算密钥。利用此法攻击**DES**，需要用2^{47}个选择明文和2^{47}次加密运算，比穷举搜索的工作量大大减少。然而找到2^{47}个选择明文的要求使这种攻击只有理论上的意义。

线性分析

已知明文攻击：加密算法，截获的部分密文和相应的明文。

该方法是**Mitsuru Matsui**于**1993**年公开的另一种对分组密码进行分析攻击的方法，这种方法试图通过大量的“明---密文对”找出分组密码算法中与密钥有关的线性方程，然后试着得到大量的这类关系从而确定密钥。其基本思想是以最佳的线性函数逼近**DES**的非线性变换**S**盒，这是一种已知明文攻击方法，可以在有2^{43}个已知明文的情况下破译**DES**。虽然获得已知明文比选择明文更容易，但线性分析作为一种攻击手段在实际上仍然不可行。

多重DES

- 为了提高DES的安全性能，并充分利用有关DES的现有软件和硬件资源，可以使用多重DES。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，从而大大提高抵抗对密钥的穷举搜索攻击的能力。
- 已经证明多重DES并不等价于使用一个56位密钥的单重DES。
- 二重DES
- 三重DES

所有可能的64位明文分组映射到所有可能的64位密文分组共有$2^{64}!(>10^{1020})$种不同的方法，56位密钥的DES算法，提供了$2^{56}(<10^{17})$个这种映射关系，所以，多重DES所对应的映射不同于单DES所定义的映射。以双重DES为例，对于消息m_1，能够找到一个k_3满足：$E_{k2}(E_{k1}(m_1))=E_{k3}(m_1)$；对于消息$m_2$，能够找到一个k4满足：$E_{k2}(E_{k1}(m_2))=E_{k4}(m_2)$；但$k_3$和$k_4$相同的概率只有$1/2^{64}$。

二重DES

二重DES指的是取独立的两个密 钥k_1和k_2

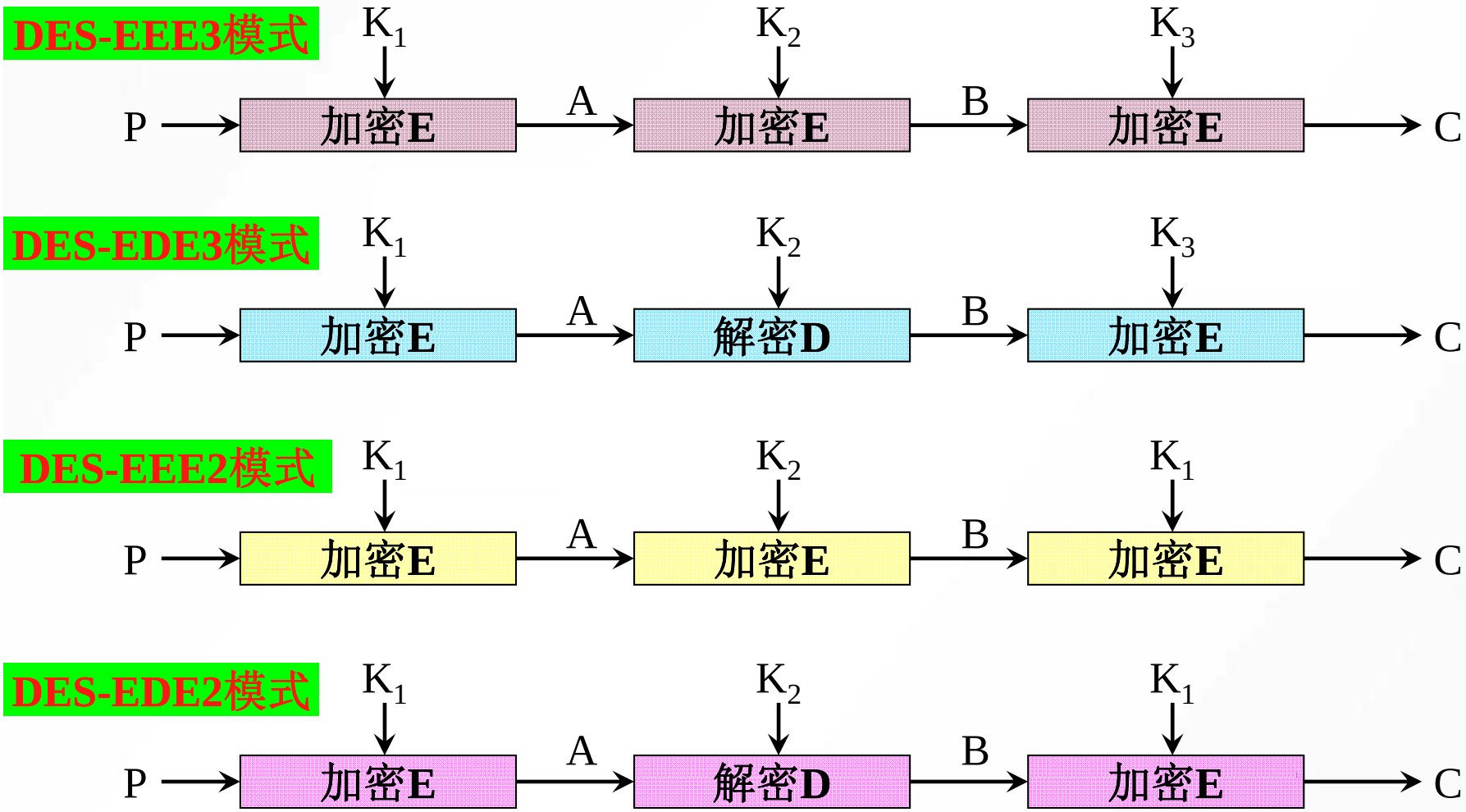
加密：$C = DES_{k_2}(DES_{k_1}(m))$

解密：$m = DES^{-1}_{k_1}(DES^{-1}_{k_2}(C))$

则$DES_{k_1}(m) = DES^{-1}_{k_2}(C)$

可对**m**采取一切可能的k_1加密并存储，在对**C**取一切可能的k_2解密，并与存储的$DES_{k1}(m)$比较，若有相等的，则k_1和k_2便可获得。二重**DES**并不像人们相像那样可提高密钥长度到**112**比特，而相当**57**比特。 （中途相遇攻击）

多重DES（四种模式）


DES-EEE3模式
K1
K2
K3
P
加密E
A
加密E
B
加密E
C
DES-EDE3模式
K1
K2
K3
P
加密E
A
解密D
B
加密E
C
DES-EEE2模式
K1
K2
K1
P
加密E
A
加密E
B
加密E
C
DES-EDE2模式
K1
K2
K1
P
加密E
A
解密D
B
加密E
C

AES算法

- AES算法的简介
- AES算法的实现
- AES算法与DES算法的对比

AES的简介

稳定的数学基础、没有算法弱点、算法抗密码分析的强度等。

不能占用大量的存储空间和内存。

高级加密标准(AES，Advanced Encryption Standard)作为传统对称加密标准DES的替代者，由美国国家标准与技术研究所(NIST)于1997年提出征集该算法的公告。1999年3月22日，NIST从15个候选算法中选出了5算法进入下一轮：MARS、RC6、Rijndael、SERPRNT和Twofish。2000年10月2日，以安全性、性能、大小、易实现等标准而最终选定由比利时的Joan Daemen与Vincent Rijmen开发的Rijndael算法，并于2001年正式发布了AES标准。

能在多个平台上以较快的速度实现。

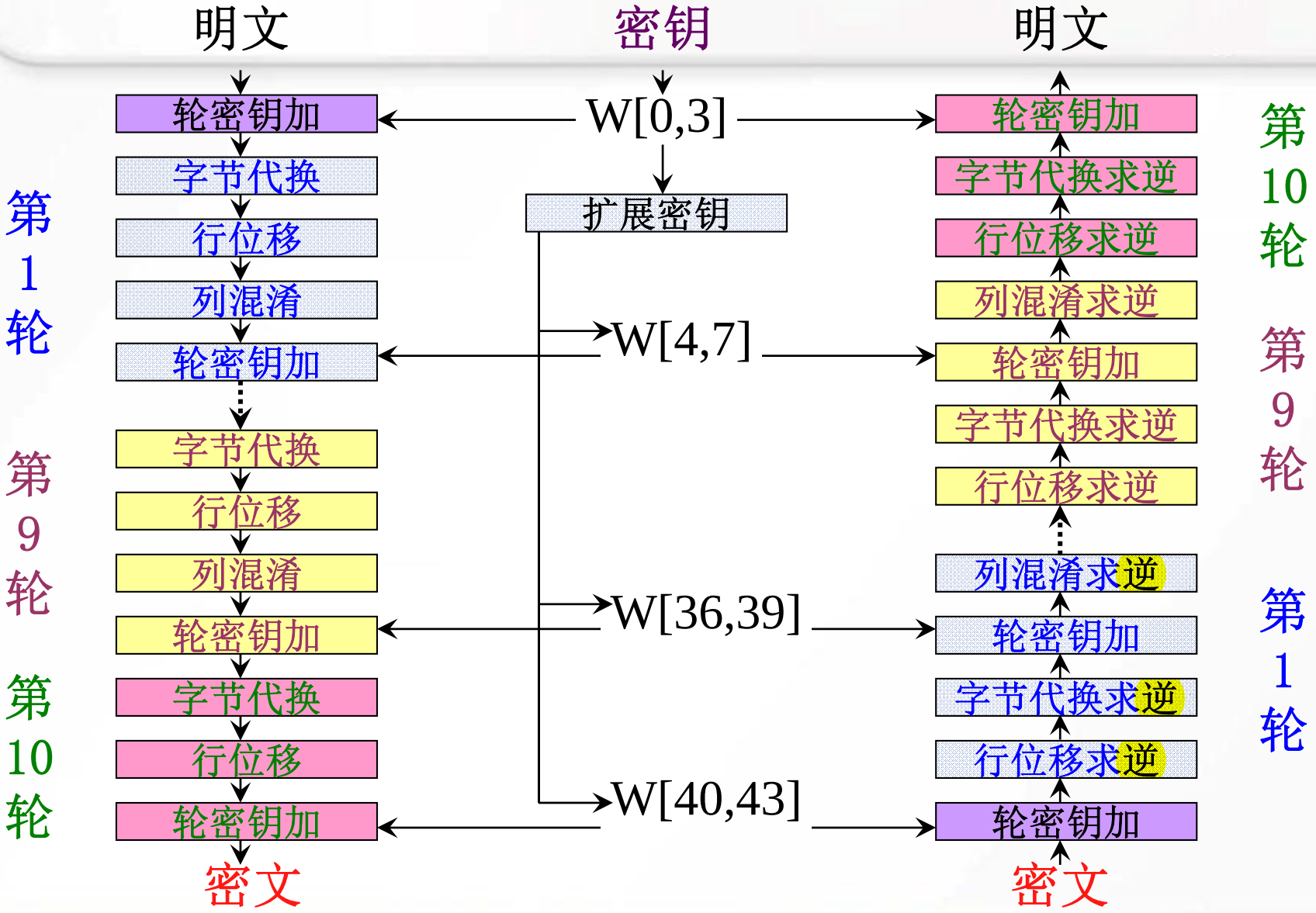
灵活性、硬件和软件适应性、算法的简单性等。

AES分组长度、密钥长度、轮数的关系

N_r	N_b=4(128位)
N_k=4(128位)	10
N_k=6(192位)	12
N_k=8(256位)	14

N_b: 分组长度； N_k: 密钥长度； N_r: 轮数

AES加解密的流程图

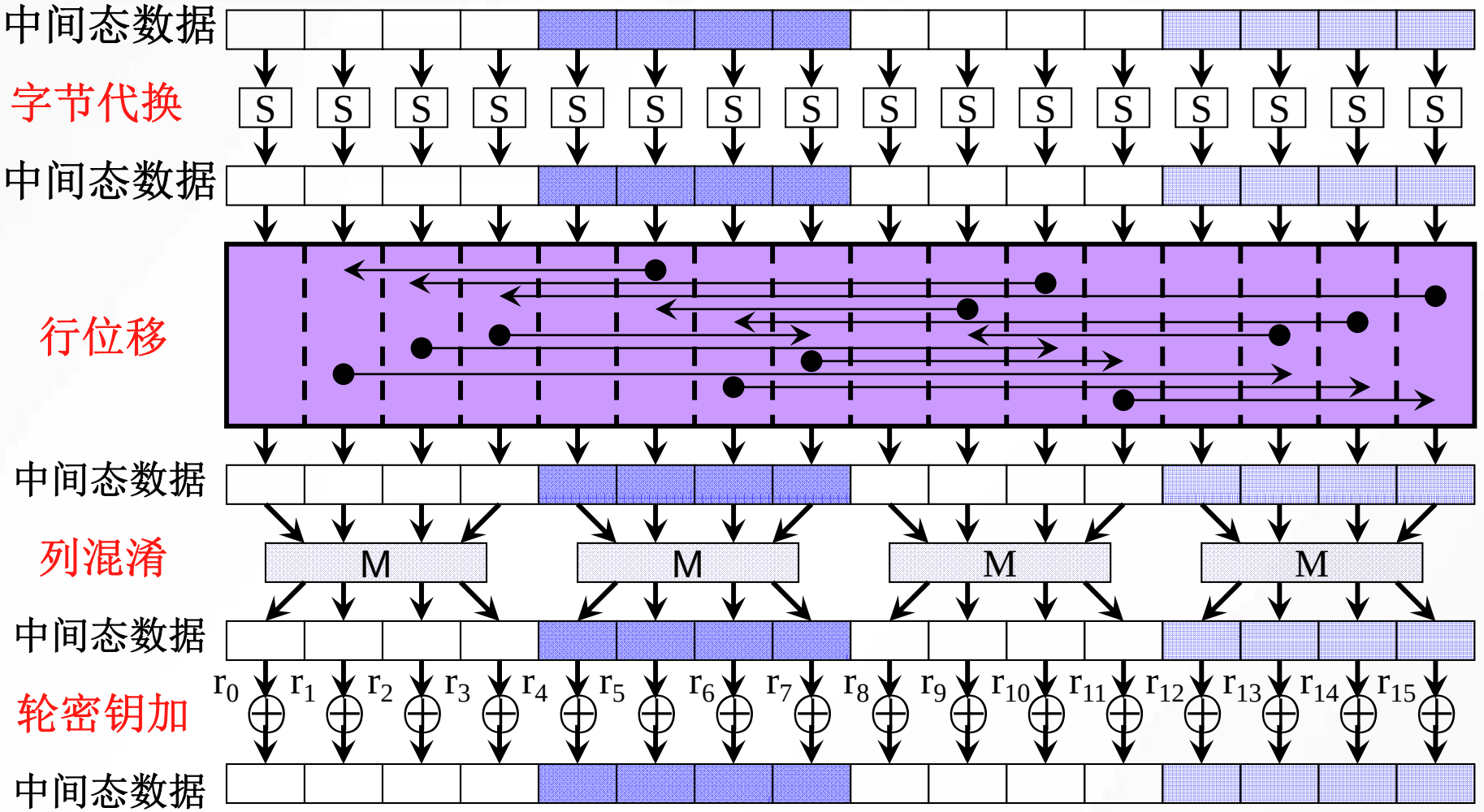

明文
密钥
明文
轮密钥加
字节代换
行位移
列混淆
轮密钥加
字节代换
行位移
列混淆
轮密钥加
字节代换
行位移
轮密钥加
密文
W[0,3]
扩展密钥
W[4,7]
W[36,39]
W[40,43]
轮密钥加
字节代换求逆
行位移求逆
列混淆求逆
轮密钥加
字节代换求逆
行位移求逆
列混淆求逆
轮密钥加
字节代换求逆
行位移求逆
轮密钥加
密文
第1轮
第9轮
第10轮
第10轮
第9轮
第1轮

AES的结构

w[0],w[1],...w[3]
w[4],...w[43]

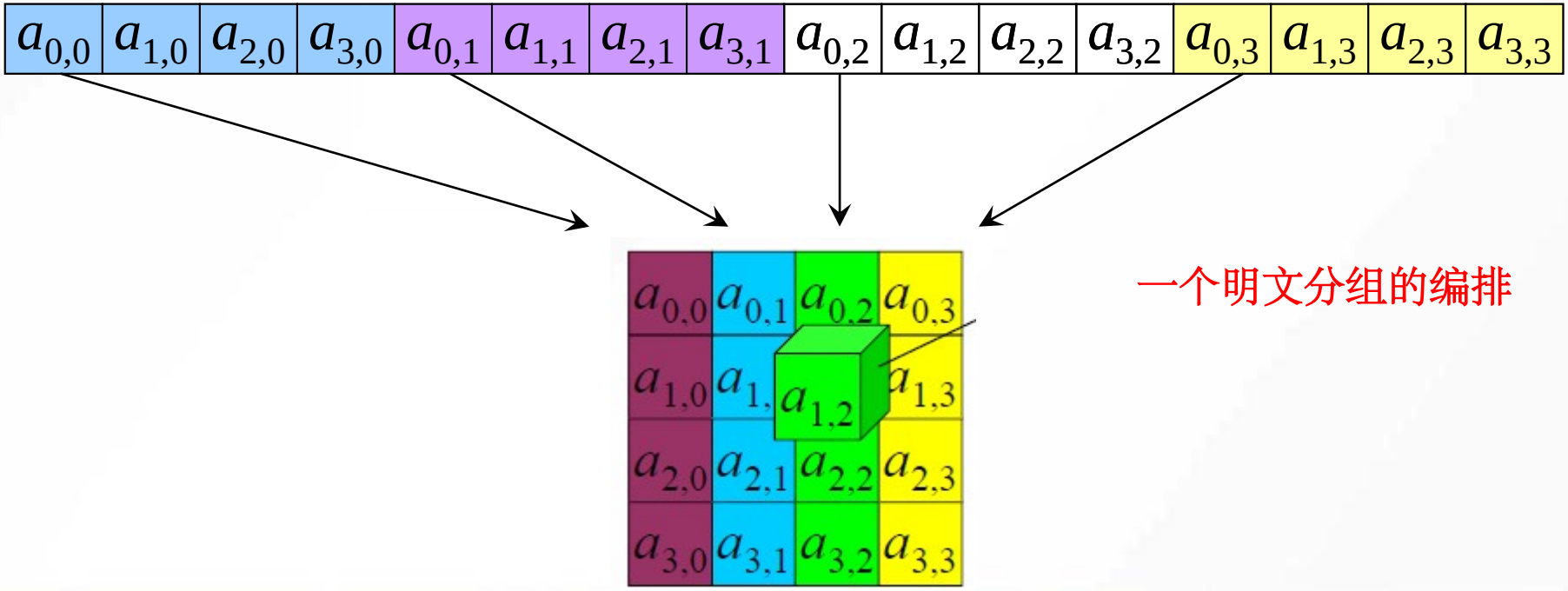
- AES的结构的一个显著特征是不同于DES的结构;
- 输入的密钥被扩展成由44个32位字所组成的数组w[i]。
- AES结构由四个不同的阶段组成：字节代换、行位移、列混淆、轮密钥加。
- 仅仅在轮密钥加阶段中使用密钥，并在算法的开始和结束都都使用轮密钥加阶段。
- 每个阶段均可逆，解密算法和加密算法并不一样。
- 加密和解密过程的最后一轮均只包含3个阶段，这是由AES的特定结构所决定的，而且也是密码算法可逆性所要求的。

AES的一轮加密过程


中间态数据
字节代换
S
中间态数据
行位移
中间态数据
列混淆
M
中间态数据
轮密钥加
r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 r15
中间态数据

AES的分组

AES首先将明文按字节分成列组。前4个字节组成第一列，接下来的4个字节组成第二列，以此类推，如下图所示。如果块为128位，那么就可组成一个4×4的矩阵。



一个明文分组的编排

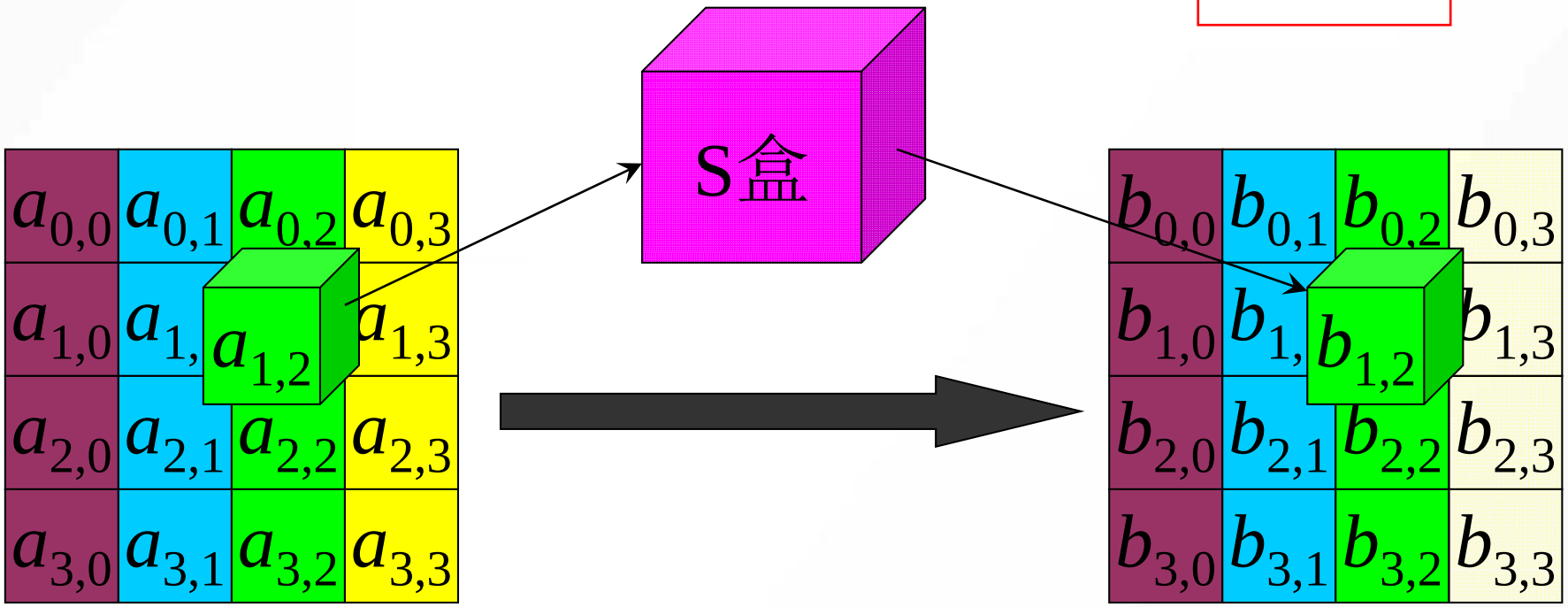
AES的S盒(字节代换ByteSub)

- S盒的替换操作
- S盒的替换表
- S盒的代数转换
- S盒替换举例

S盒的替换操作

AES算法的操作对象是字节(byte)!
DES算法的操作对象是比特(byte)!

列的每个元素作为输入用来指定S盒的地址：前4位指定S盒的行，后4位指定S盒的列。由行和列所确定的S盒位置的元素取代了明文矩阵中相应位置的元素。



S盒的替换表

（1）求95的逆元，95*x mod 11B=1
（2）对x进行仿射变换，求得是2A

x/y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

S盒的代数转换

首先，将字节看作$GF(2^8)$上的元素，映射到自己的乘法逆元，”00”映射到自己。即$a*a^{-1} \equiv 1 \bmod (x^8+x^4+x^2+x+1)$

其次，对字节（逆元）作如下的仿射变换：

$$\begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{vmatrix} \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{vmatrix}^{-1} + \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{vmatrix}$$

逆字节代换

- 先进行仿射变换的逆变换；
- 再计算乘法逆；

$$\begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{vmatrix} \begin{vmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{vmatrix} + \begin{vmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{vmatrix}$$

S盒替换举例

输人矩阵（用十六进制表示）与相应的输出如下所示：

输入				输出			
12	2a	21	0b	c9	e5	fd	2b
45	bd	04	c1	6e	7a	f2	78
23	0a	00	l c	26	67	63	9c
89	11	2a	fc	a7	82	e5	b0

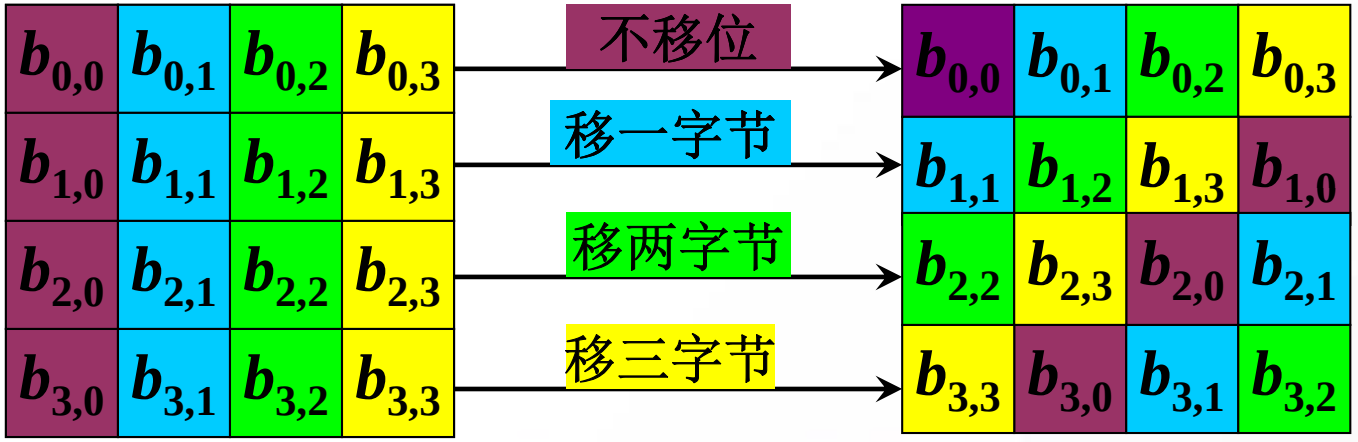
S盒评价

S盒被设计成能防止已有的各种密码分析攻击。Rijndael的开发者特别寻求在输入位和输出位之间几乎没有相关性的设计，且输出值不能通过利用一个简单的数学函数变换输入值所得到。另外，在代码变换中所选择的常量使得在S盒中没有不动点[S盒(a)=a]，也没有“反不动点”[S盒(a)=逆a]。

当然，S盒必须是可逆的，即逆S盒[S盒(a)]=a。然而，S盒(a)=逆S盒(a)不成立，在这个意义上S盒不是自逆的。例如，S盒({95})={2A}，但逆S盒({95})={AD}。

AES的行移位(ShiftRow)

行移位操作是作用于S盒的输出的，其中，列的4个行螺旋地左移，即第0行左移0字节，第1行左移1字节，第2行左移2字节，第3行左移3字节，如下图所示。从该图中可以看出，这使得列完全进行了重排，即在移位后的每列中，都包含有未移位前每个列的一个字节。接下来就可以进行列内混合了。(逆向行移位变换将中间态数据的后三行执行相反方向的移位操作)



行移位评价

由于中间态数据和算法的输入输出数据一样，也是由4列所组成的数组，在加密过程中，明文逐列被复制到中间态数据上，且后面的轮密钥也是逐列应用到中间态数据上，因此，行位移将某个字节从一列移到另一列中，这个变换确保了某列中的4字节被扩展到了4个不同的列。

AES的列混淆(MixColumn)

在列混合变换中，将状态阵列的每个列视为GF(2^8)上的多项式，再与一个固定的多项式c(x)进行模[ ]乘法，要求c(x)是模[ ]可逆的多项式。

m(x)

$c(x)=\text{'03'}x^3+\text{'01'}x^2+\text{'01'}x+\text{'02'};$

设：b(x)=c(x) *a(x) mod [ ]

$m(x)=x^8+x^4+x^3+x+1\ (100011011)$

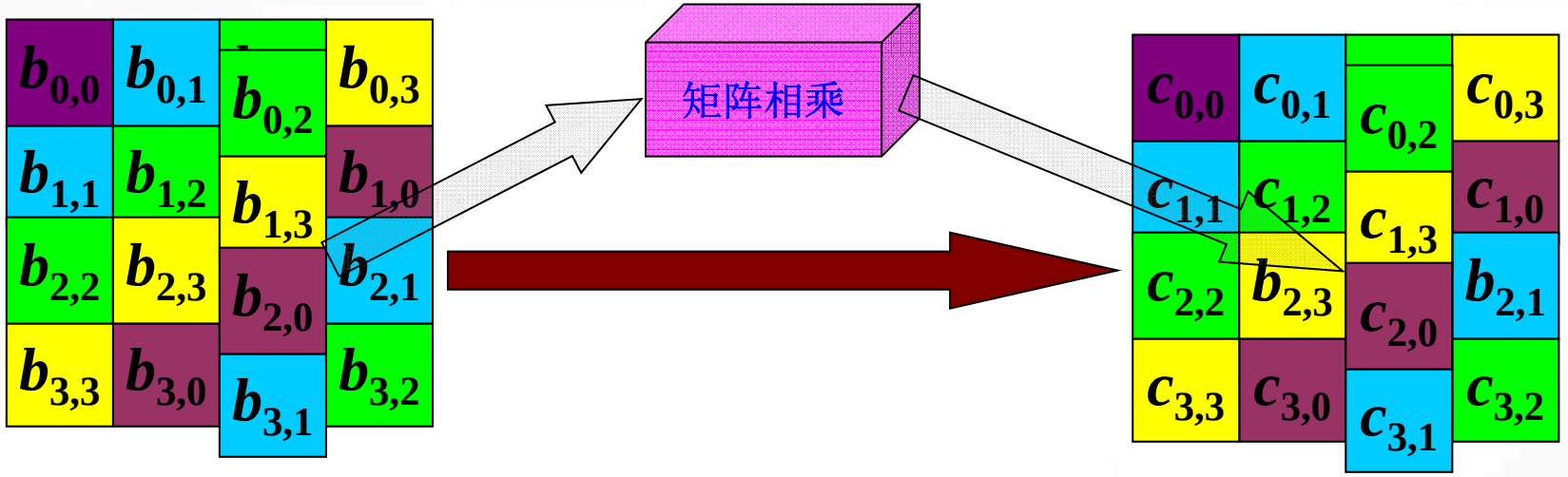
则：

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

逆变换：

$$\begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{vmatrix} = \begin{vmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{vmatrix} \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{vmatrix}$$

AES的列淆(MixColumn)



$$\begin{vmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{vmatrix} = \begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 01 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{vmatrix}$$

举例

$$\begin{vmatrix} 73 \\ 6b \\ ba \\ a7 \end{vmatrix} = \begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 01 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \begin{vmatrix} 11 \\ 09 \\ 01 \\ 35 \end{vmatrix}$$

列混淆的数学基础

$$b(x) \equiv (b_3x^3 + b_2x^2 + b_1x + b_0) \bmod (x^4 + 1)$$

$$a(x) \equiv (a_3x^3 + a_2x^2 + a_1x + a_0) \bmod (x^4 + 1)$$

$$b(x) \equiv c(x)a(x) \bmod (x^4 + 1)$$

$$\because\ x^i \bmod (x^4 + 1) \equiv x^{i(\bmod 4)}$$

$$\therefore\ b_0 = a_0c_0 \oplus a_1c_3 \oplus a_2c_2 \oplus a_3c_1$$

$$b_1 = a_0c_1 \oplus a_1c_0 \oplus a_2c_3 \oplus a_3c_2$$

$$b_2 = a_0c_2 \oplus a_1c_1 \oplus a_2c_0 \oplus a_3c_3$$

$$b_3 = a_0c_3 \oplus a_1c_3 \oplus a_2c_1 \oplus a_3c_0$$

在AES中，取

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

则 $c^{-1}(x) = 0Bx^3 + 0Dx^2 + 09x + 0E$

列混淆评价

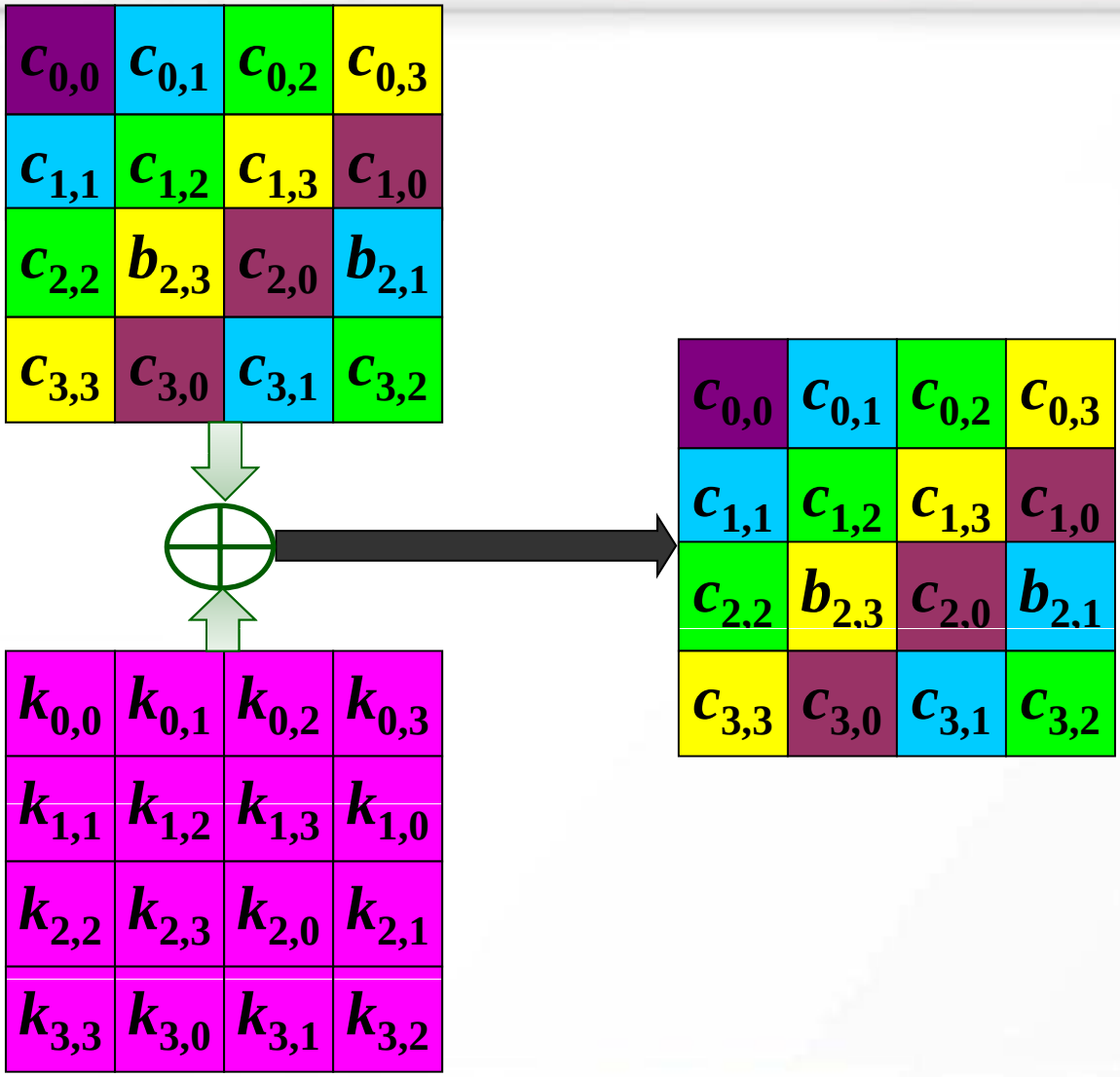
列混淆变换的矩阵系数是基于在码字间有最大距离的线性编码，这使得在每列的所有字节中有良好的混淆性。列混淆变换和行移位变换使得在经过几轮变换后，所有的输出位均与所有的输入位相关。

混乱与扩散

此外，列混淆变换的系数，即**{01}**，**{02}**，**{03}**是基于算法执行效率考虑的，而逆向列混淆变换中的系数并不是出于效率的考虑，加密被视为比解密更重要。

AES的轮密钥加变换

上结果与子密钥进行异或逻辑运算



AES的密钥生成

AES密钥生成算法的操作对象是字（word）=4bytes

密钥是按矩阵的列进行分组的，然后添加40个新列来进行扩充。如果前4列(即由密钥给定的那些列）为W(0) ,W(1) ,W(2)和W(3)，那么新列以递归方式产生:

如果i不是4的倍数，那么第i列由如下等式确定：

$$W(i)=W(i-4)\ \text{XOR}\ W(i-1)$$

如果i是4的倍数，那么第i列由如下等式确定：

$$W(i)=W(i-4)\ \text{XOR}\ T[W(i-1)]$$

T[W(i-1)]的实现方式

前2步的结果与轮常量进行异或

- 循环地将W(i-1)的元素移位，每次一个字节，也就是说，abcd变成了bcda。 循环左移8bits(=1bytes)
- 将这4个字节作为S盒的输入，输出新的4个字节efgh。
- 轮常量Rcon[i]见下表。 轮常量是4字节的字，但后面3个字节都为0，所以和efgh进行异或，只相当于对第一个自己进行了异或
- 这样生成转换后的列：［e XOR Rcon[i], f, g, h］。

i	1	2	3	4	5
Rcon[i]	01000000	02000000	04000000	08000000	10000000
i	6	7	8	9	10
Rcon[i]	20000000	40000000	80000000	1b0000000	36000000

$$R_{con}[i] = (RC[i], '00', '00', '00')$$

$$RC[i] = x \cdot RC[i-1] = x^{i-1} \bmod (x^8 + x^4 + x^3 + x + 1) \quad (i \geq 1)$$

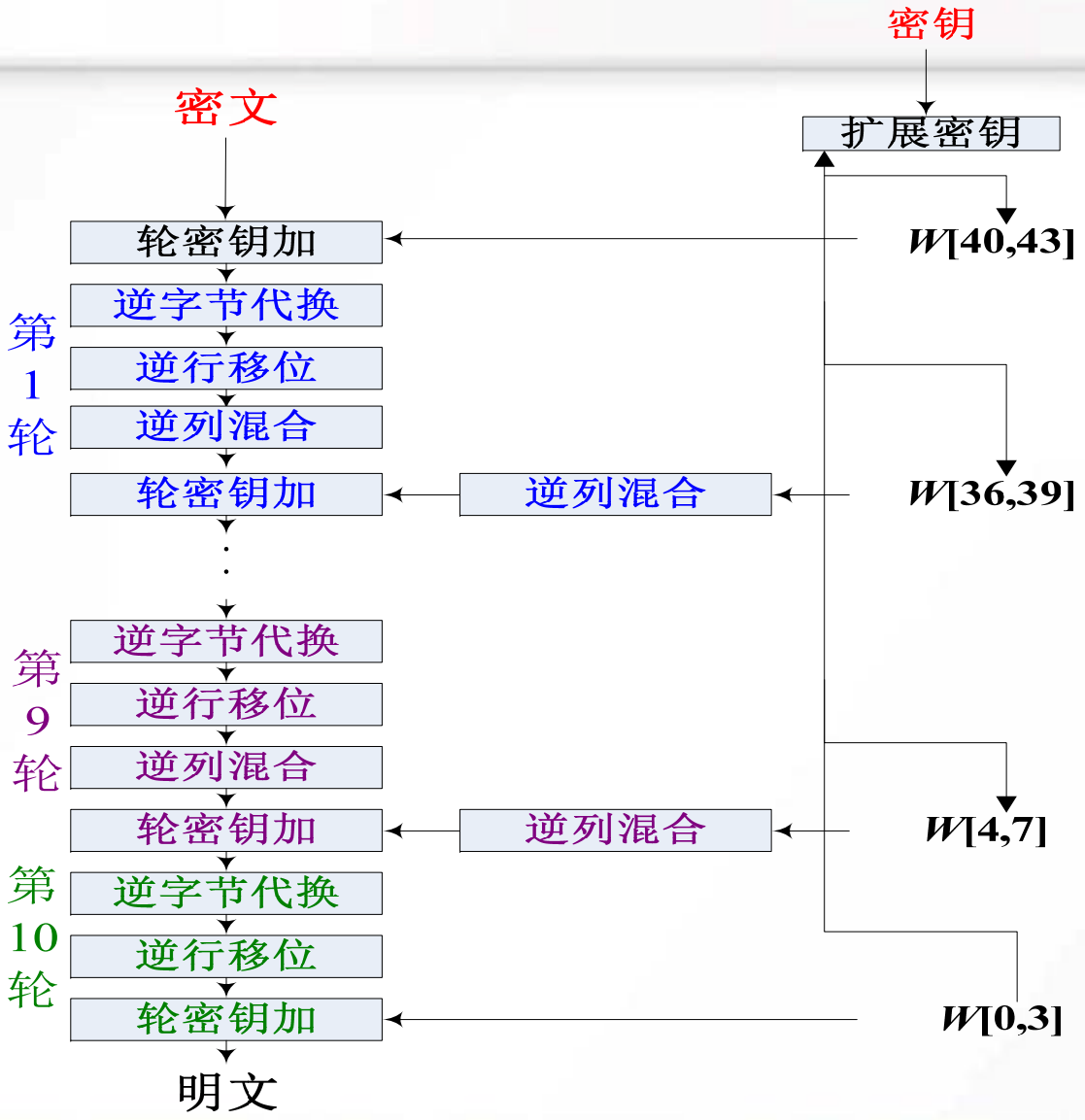
=2*RC[i-1],即对RC[i-1]左移1位，右边填充0，再与m(x)异或

密钥扩展评价

- 知道密钥或轮密钥的部分位不能计算出轮密钥的其它位，但知道扩展密钥中任何连续的N_k个字能够重新产生整个扩展密钥。
- 使用轮常量来排除对称性。
- 密钥的每一位能影响到轮密钥的一些位。
- 足够的非线性以防止轮密钥的差异完全由密钥的差异所决定。

AES的解密过程

四种操作顺序与加密正好相反，key的使用也是从w[43]到w[0]



AES设计上的考虑

- 与DES相比，扩散的效果更快，即两轮可达到完全扩散。
- S盒使用清晰而简单的代数方法构造，避免任何对算法安有后门的怀疑。
- 密钥扩展方案实现对密钥位的非线性混合，既实现了雪崩效应，也实现了非对称性。
- 比穷举攻击更好的攻击进行到6轮，多出4轮可以提供足够多的安全性。

AES的安全性

- 弱密钥

  AES在设计上不是对称的，其加密和解密过程不一致，这也避免弱密钥的存在。

- 差分分析和线性分析

  由于在设计时考虑了这两种攻击的方法，因此AES具有较好的抗击其攻击的能力。

- 密钥穷举攻击

  平均需要2^{127}次AES运算，计算量是非常大。

AES和DES相似之处

- 二者的圈(轮)函数都是由三层构成，非线性层、线性混合层、子密钥异或，只是顺序不同。
- AES的子密钥异或对应于DES中S盒之前的子密钥异或。
- AES的列混合运算的目的是让不同的字节相互影响，而DES中F函数的输出与左边一半数据相加也有类似的效果。
- AES的非线性运算是字节代替(ByteSub)，对应于DES中惟一的非线性运算S盒。
- 行移位运算保证了每一行的字节不仅仅影响其它行对应的字节，而且影响其它行所有的字节，这与DES中置换P相似。

AES和DES不同之处

- AES的密钥长度(128位、192位、256位)是可变的，而DES的密钥长度固定为56位。
- DES是面向比特的运算，AES是面向字节的运算。
- AES的加密运算和解密运算不一致，因而加密器不能同时用作解密器，DES则无此限制。

前面讲的DES/AES算法都是针对一个明文分组的！！

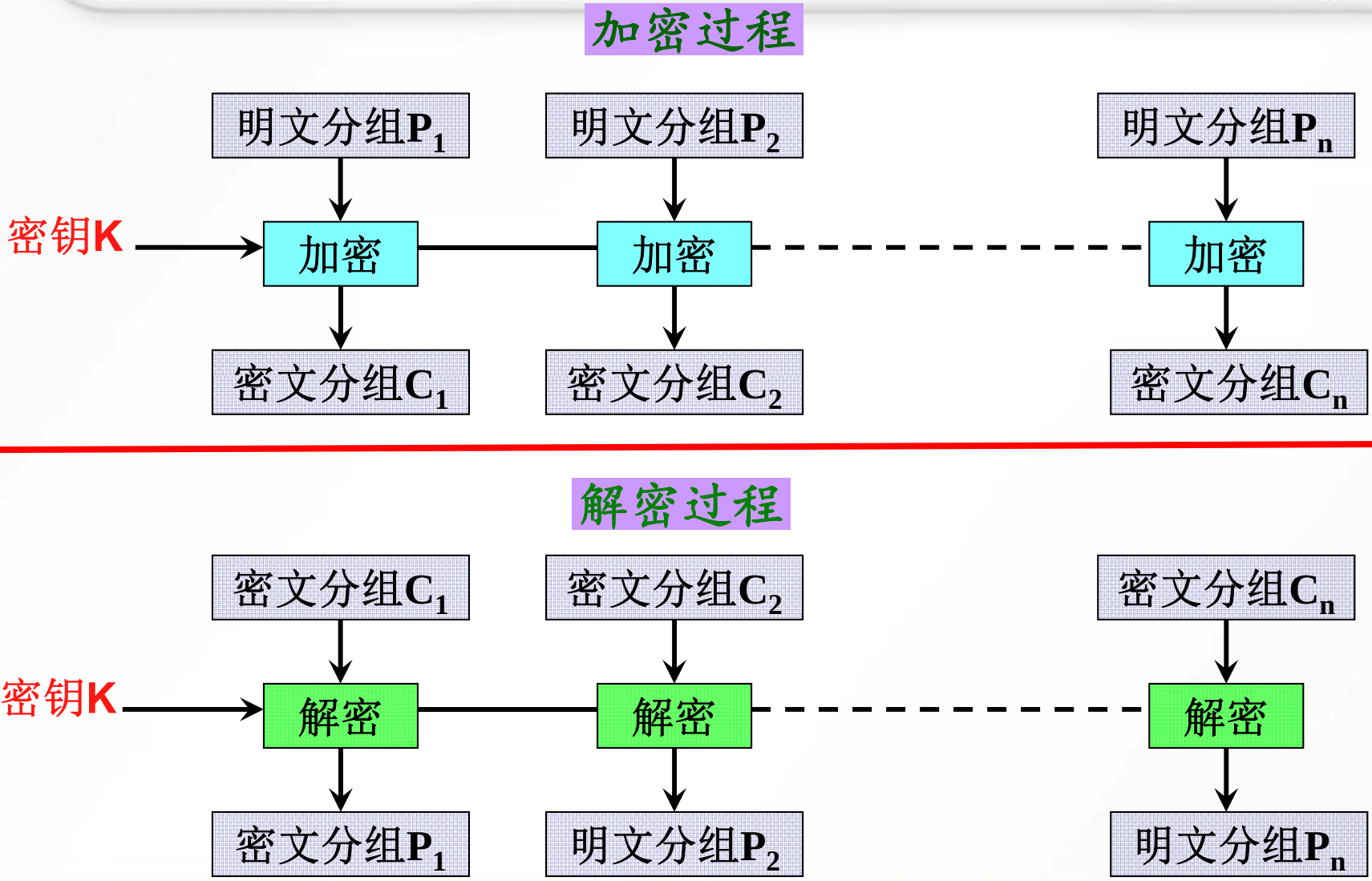
分组密码操作模式的前言

在实际运用中，需要加密的消息的数据量是不定的，数据格式可能是多种多样的，因此需要做一些变通，灵活地运用这些基本密码。而且，也需要采用适当的工作模式来隐藏明文的统计特性、数据格式等，以提高整体的安全性，降低删除、重放、插入和伪造成功的机会。这种基本密码算法的适当的工作模式就是密码模式，也称为分组密码算法的运行模式。密码模式通常是基本密码、一些反馈、和一些简单运算的组合。

分组密码的操作模式

- 电子密码本模式(ECB，Electronic Code Book)
- 密码分组链接模式(CBC，Clipher Block Chaining)
- 密码反馈模式(CFB，Cipher Feedback)
- 输出反馈模式(OFB，Output Feedback)

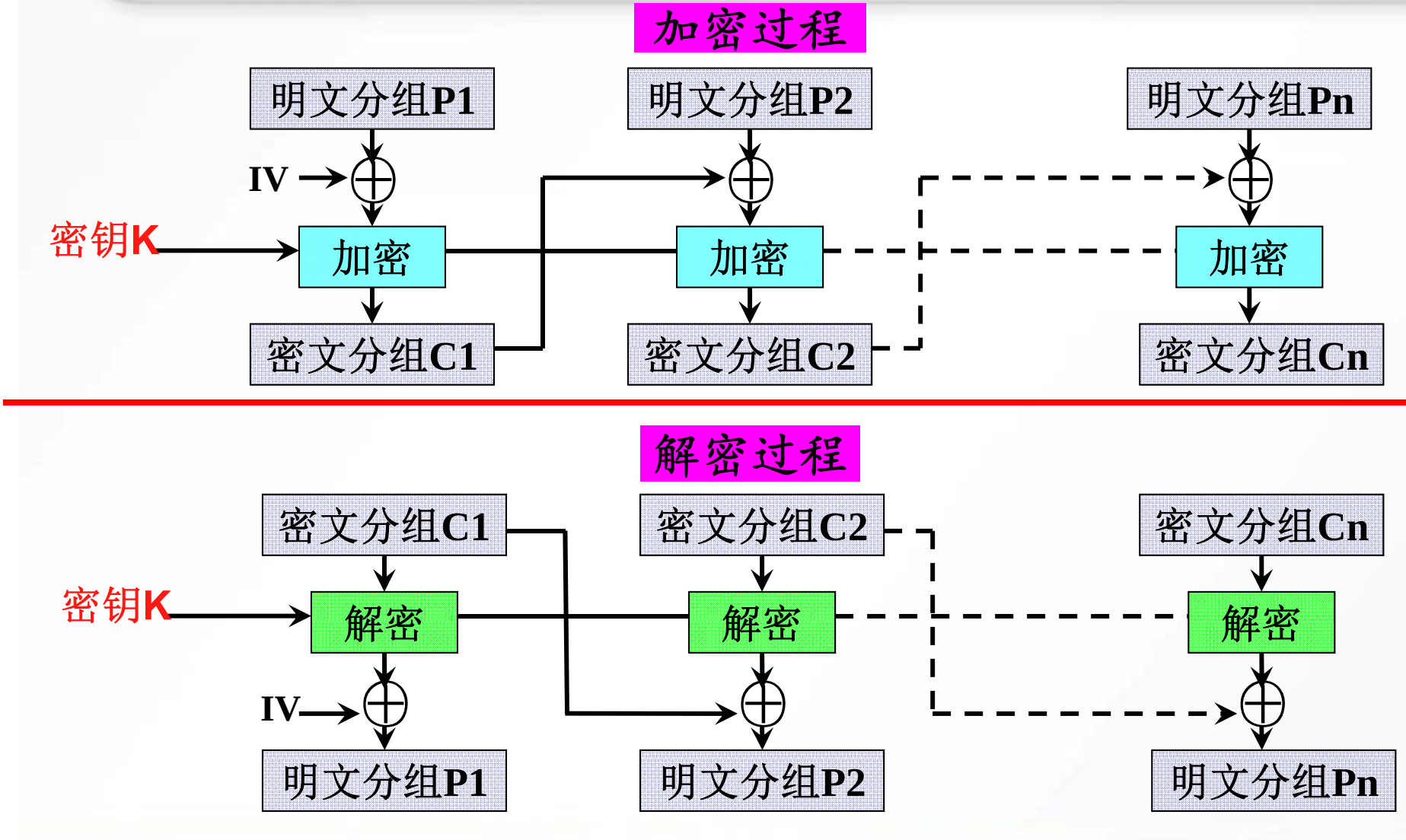
电子密码本模式（ECB）


加密过程
明文分组P_1
明文分组P_2
明文分组P_n
密钥K
加密
加密
加密
密文分组C_1
密文分组C_2
密文分组C_n
解密过程
密文分组C_1
密文分组C_2
密文分组C_n
密钥K
解密
解密
解密
密文分组P_1
明文分组P_2
明文分组P_n

ECB模式特点

- 模式操作简单，主要用于内容较短且随机的报文的加密传递；
- 相同明文（在相同密钥下）得出相同的密文，即明文中的重复内容将在密文中表现出来，容易实现统计分析攻击、分组重放攻击和代换攻击；
- 链接依赖性：各组的加密都独立于其它分组，可实现并行处理；
- 错误传播：单个密文分组中有一个或多个比特错误只会影响该分组的解密结果；

密码分组链接模式（CBC）

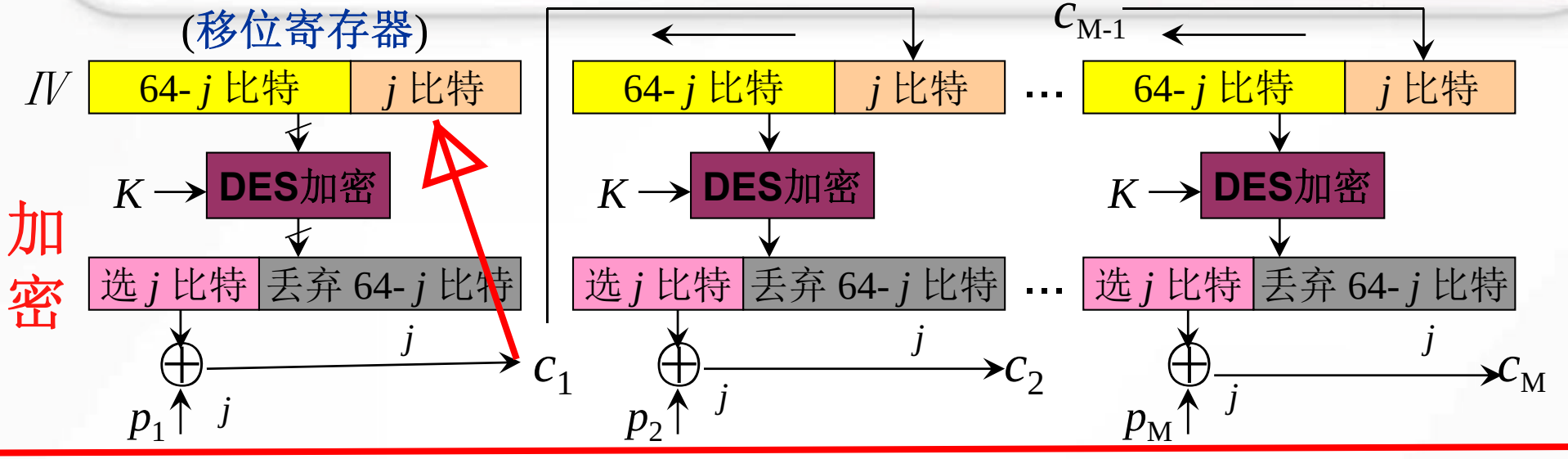


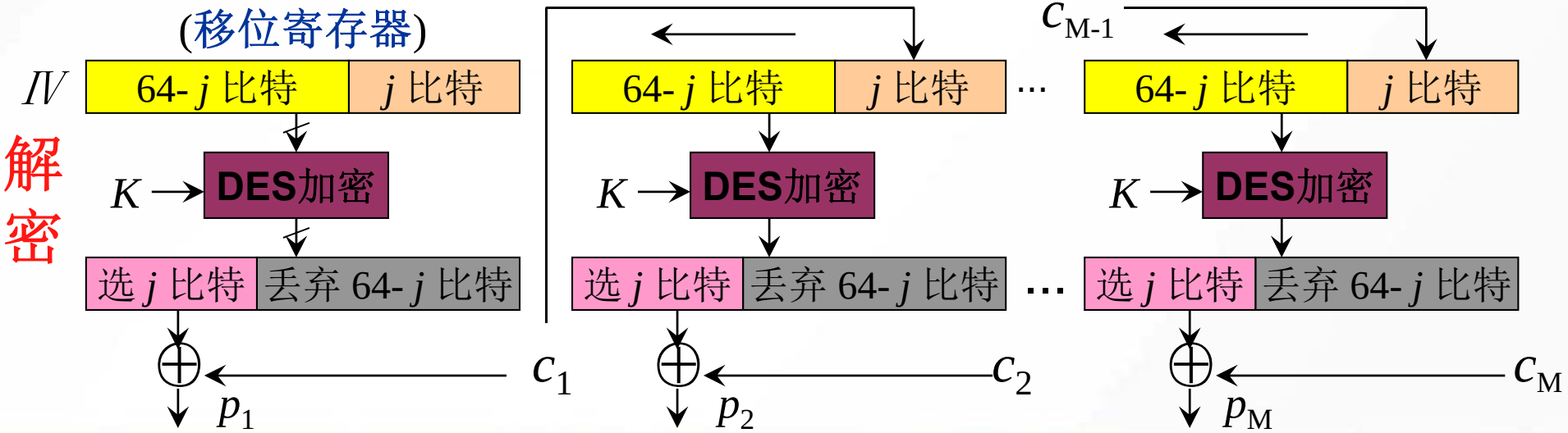
CBC特点

- 一种反馈机制在分组密码中的应用，每个密文分组不仅依赖于产生它的明文分组，还依赖于它前面的所有分组；
- 相同的明文，即使相同的密钥下也会得到不同的密文分组，隐藏了明文的统计特性；
- 链接依赖性：对于一个正确密文分组的正确解密要求它之前的那个密文分组也正确，不能实现并行处理；
- 错误传播：密文分组中的一个单比特错误会影响到本组和其后分组的解密，错误传播为两组；
- 初始化向量IV不需要保密，它可以明文形式与密文一起传送 。

密码反馈模式（CFB）

(1)明文不再是64bits,而是j个bits
(2)CFB实际上是同步序列密码（流密码）

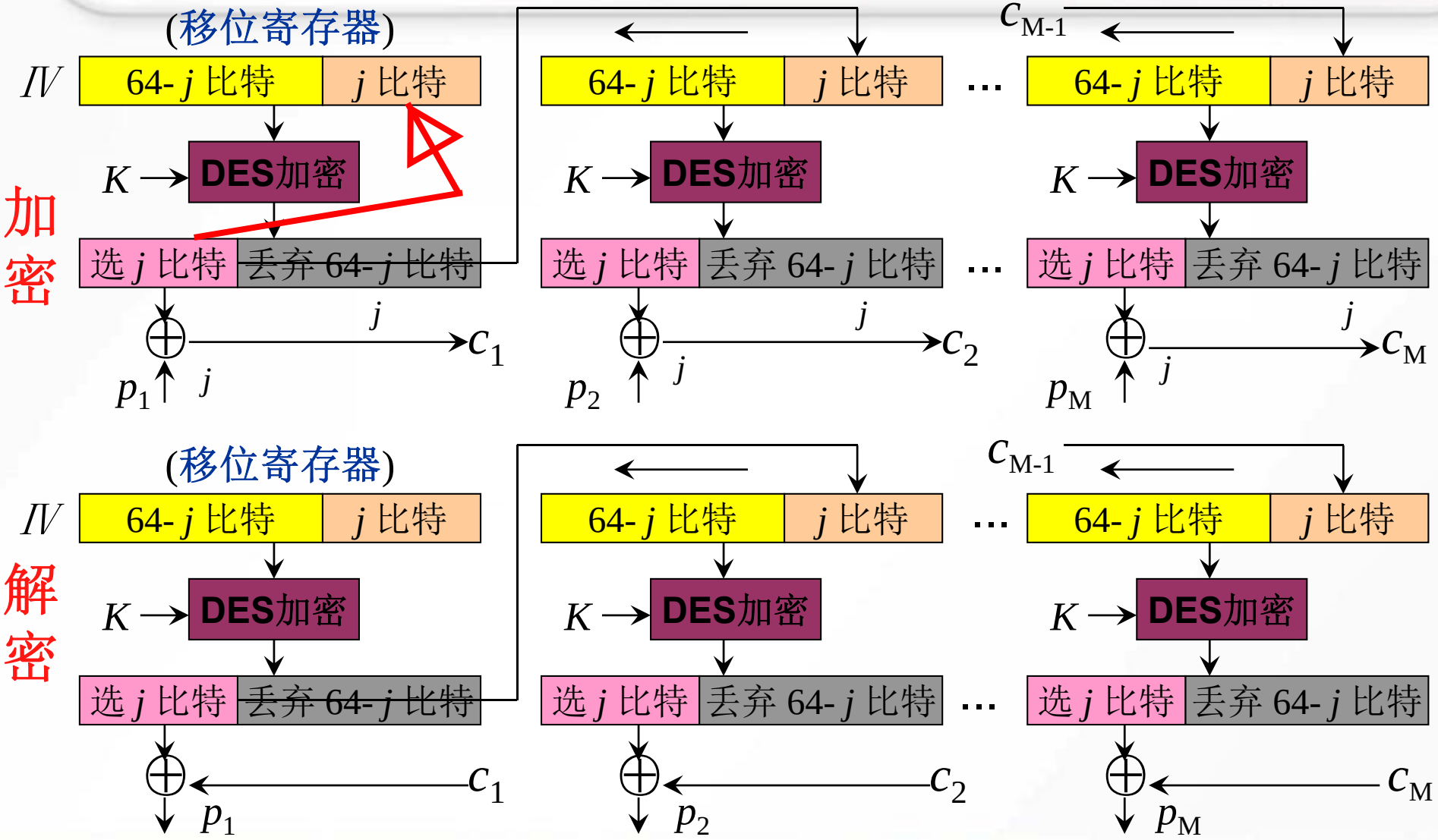




CFB模式特点

- 消息被看作bit流，不需要整个数据分组在接受完后才能进行加解密。
- 可用于同步序列密码。
- 具有CBC模式的优点。
- 对信道错误较敏感且会造成错误传播。
- 数据加解密的速率降低，其数据率不会太高。

输出反馈模式（OFB）


加密
解密
(移位寄存器)
IV
64- j 比特
j 比特
K
DES加密
选 j 比特
丢弃 64- j 比特
j
c_1
c_2
c_M
c_{M-1}
p_1
p_2
p_M

OFB特点

- OFB模式是CFB模式的一种改进，克服由错误传播带来的问题，但对密文被篡改难于进行检测;
- OFB模式不具有自同步能力，要求系统保持严格的同步，否则难于解密;
- 初始向量IV无需保密，但各条消息必须选用不同的IV;

分组密码的操作模式小结

- ECB是最快、最简单的分组密码模式，但它的安全性最弱，一般不推荐使用ECB加密消息，但如果是加密随机数据，如密钥，ECB则是最好的选择。
- CBC适合文件加密，而且有少量错误时不会造成同步失败，是软件加密的最好选择。
- CFB通常是加密字符序列所选择的模式，它也能容忍少量错误扩展，且具有同步恢复功能。
- OFB是在极易出错的环境中选用的模式，但需有高速同步机制。

本讲主要内容

- 分组密码的简介
- **DES密码算法**
- **AES密码算法**
- 分组密码的操作方式

其它分组加密算法：
IDEA
RC6
TWOFISH
SKIPJACK

谢　谢！