密码学 之
AES

范明钰

信息安全研究中心

主要内容

背景

- DES不够安全
- 3DES（或称T-DES）安全，但速度慢

NIST于1997年4月正式发出AES的征求意向

基本要求：比3-DES快，且至少和3-DES一样安全，分组长度为128位，密钥长度为128、192或256位

1998年8月20日，NIST召开第一次AES候选会议，公布满足候选要求的15个AES算法

1999年8月，NIST从15个算法中选出了5个候选：Mars、RC6、Rijndael、Serpent、Twofish

2000年10月2日，NIST正式宣布Rijndael被选为AES，同时总结了所有的算法并阐明了选择的理由

历程

- 1997年9月，NIST提出的评估标准
  - 安全性：实际安全、密文随机性、数学合理性、公众评价
  - 代价：使用许可、计算效率、存储容量
  - 执行特性：适应各种应用、软硬件适应各种环境、简单
- 2000年10月，NIST对Rijndael的最终评估
  - 整体安全性、软件执行效率、有限存储空间环境、硬件执行效率、抗攻击性、加解密算法比较、密钥灵活、其它多功能行和适应性、指令级并行操作

AES的设计要求

- 能抵抗所有已知的攻击
- 在各种平台上，执行速度快且代码紧凑
- 设计简单
- 分组长度指定为128位，密钥长度为128，192或256位，相应的迭代轮数R为10、12和14
- 可变块长、可变密钥长度

算法的设计原则：简单性和对称性

- ①各轮之间的对称性：好处是在密钥的控制下对同一个轮交换进行循环迭代；只要描述一轮变换即可将整个规范描述清楚，在软件实现中也可以仅对一轮进行编程。
- ②轮变换内部的对称性：指对状态中所有比特均用相似方法处理。
- ③S盒的对称性
- ④加密和解密的对称性

简况

克服了DES算法的弱点，保留了多轮运算的合理模式

采用*比利时*密码学家Joan Daemen和Vincent Rijmen的密码算法方案：Rijndael算法

AES沿革

- 比利时密码学家Joan Daemen和Vincent Rijmen所设计，以Rijndael为名投稿高级加密标准的甄选流程。（Rijndael的发音近于 "Rhine doll"）
- Rijndael是由Daemen和Rijmen早期所设计的Square改良而来；而Square则是由SHARK发展而来。
- 与其前任DES不同，Rijndael使用**代换-置换**网络。AES在软件及硬件上都能快速地加解密，相对来说较易于实现，且只需要很少的存储器。

AES 算法参数

- 明密文长度 (N_b) 固定：128bits
- 密钥长度(N_k)可变：128、192、256bits，分别对应AES-128、AES-192、AES-256
- 迭代次数 (N_r)随密钥改变：AES-128时Nr=10、AES-192 时Nr=12、AES-256 时Nr=14

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Number of rounds	10	12	14
Expanded key size (words/byte)	44/176	52/208	60/240

AES总体情况

- 不是Feistel结构，是一种SPN
- 明文分组大小为128 bits
- 有三种密钥长度{128, 192, 256}
- 迭代次数Nr可变：
  - |K|=128时Nr=10
  - |K|=192 时Nr=12
  - |K|=256 时Nr=14

算法结构

1.输入分组以正方形矩阵State进行描述

2.密钥扩展为矩阵

3.进行9/11/13轮迭代

①字节代换

②行移位

③列混淆

④轮密钥加

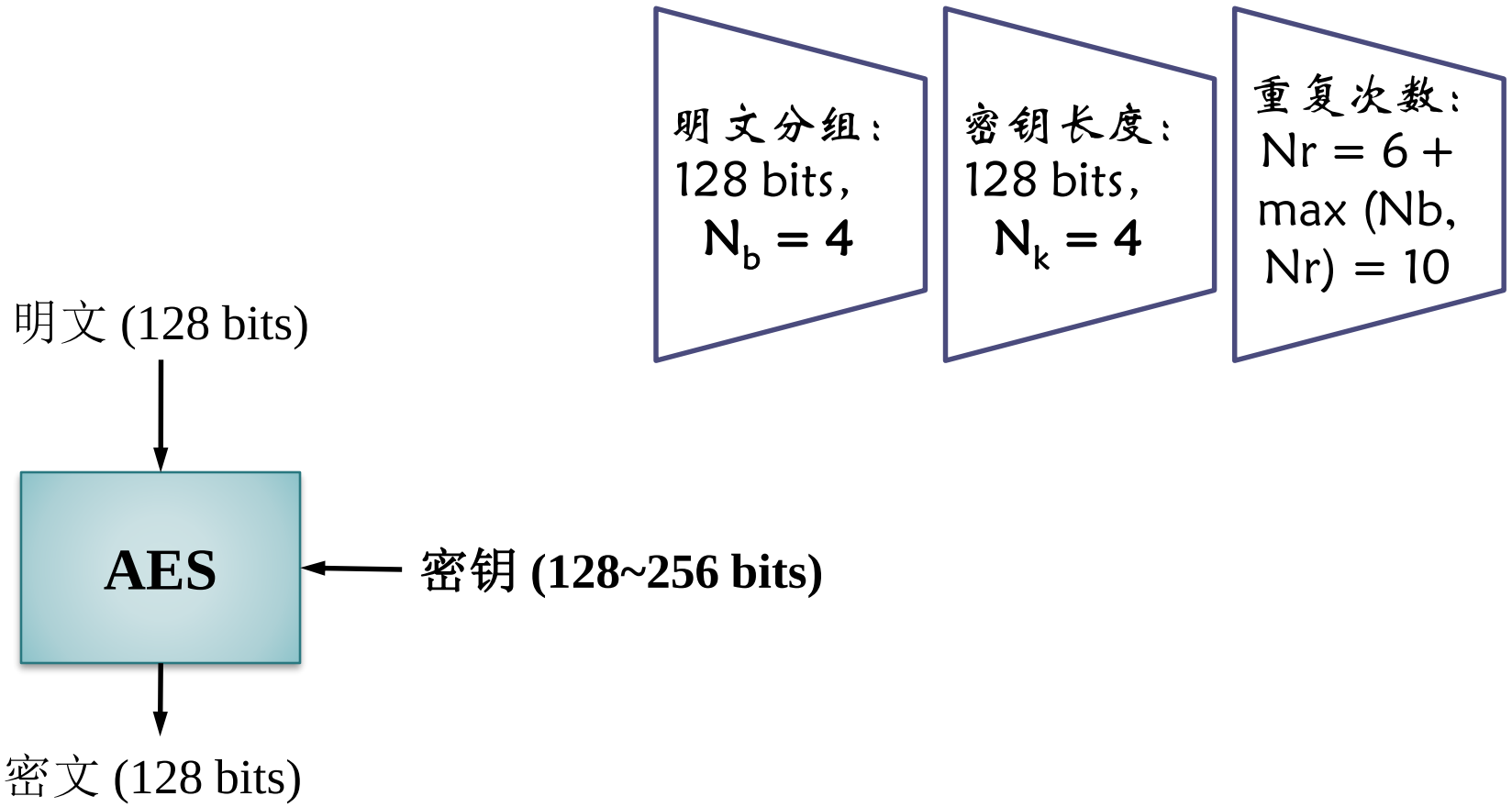
4.**最后一个不完整轮**

5.矩阵State转换为输出分组

密钥使用的分析

- 在算法的开始和结束都有轮密钥加
  - 以不需要密钥的运算用于开始和结束，不能增加安全性
- 每轮迭代中：
  - 轮密钥加实质上是一种Vernam密码，本身不难破译
  - 字节代换、行移位、列混淆三个阶段一起提供了混淆、扩散和非线性功能。这些阶段不涉及密钥，其本身并不提供安全性
  - 各阶段均可逆
  - 解密算法与加密算法不同，各阶段为逆操作；仅轮密钥加阶段算法相同

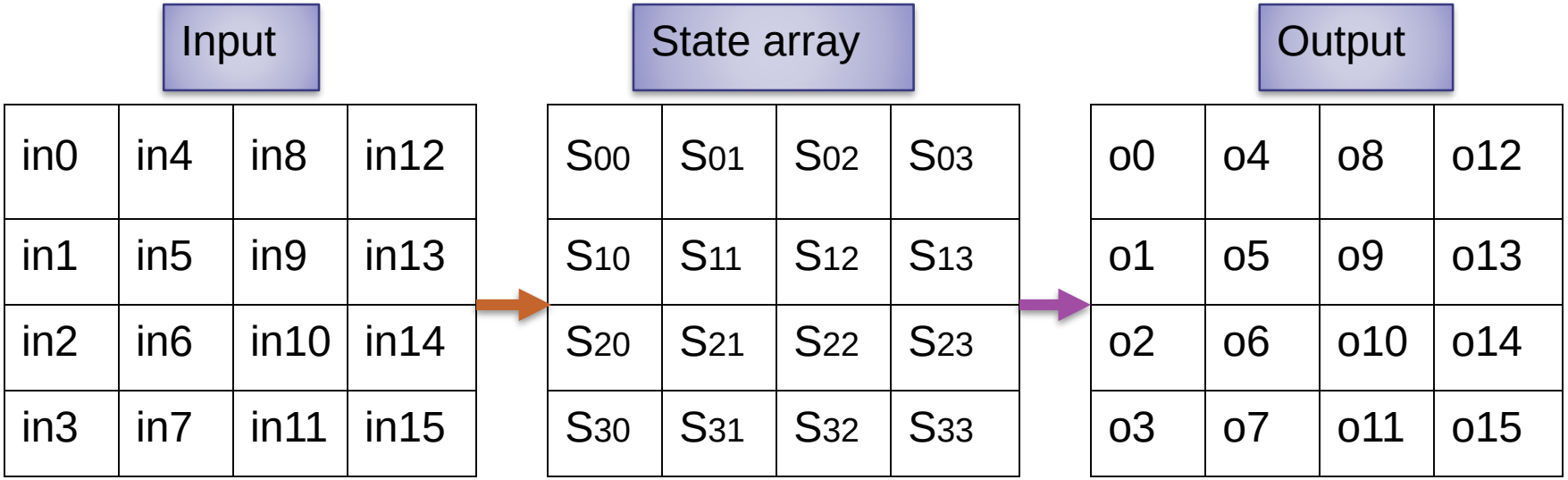
AES分组密码范例（128BIT）



算法中的明文分组

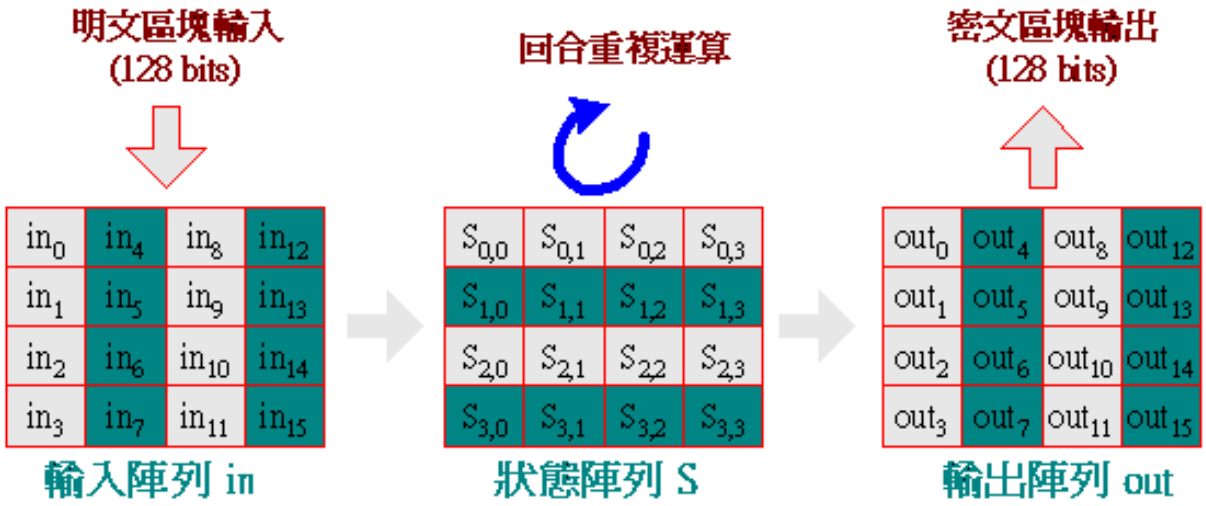
- 每个128 bit分组输入
- 拷进N_b列的状态阵列 (N_b=4)
- 按照四种运算进行处理（一种置换，三种代替）：

  Substitute bytes, Shift rows, Mix columns, Add round key

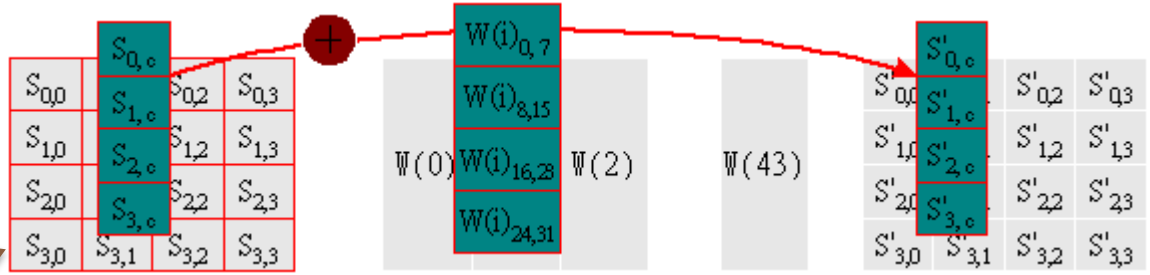


AES加密过程

每区段为 8 bits → 输入数组 in → 状态数组 (State) S → 输出数组 out

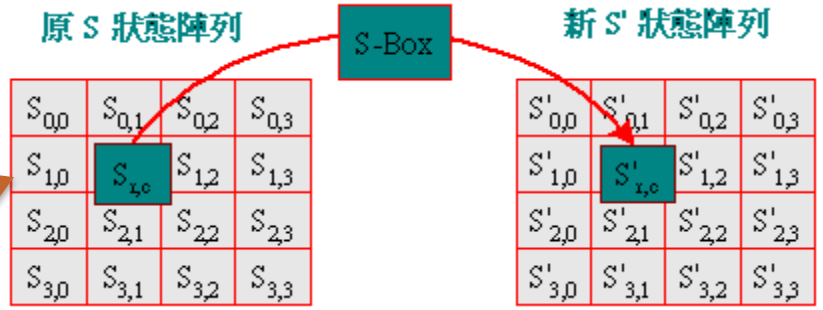


单轮AES

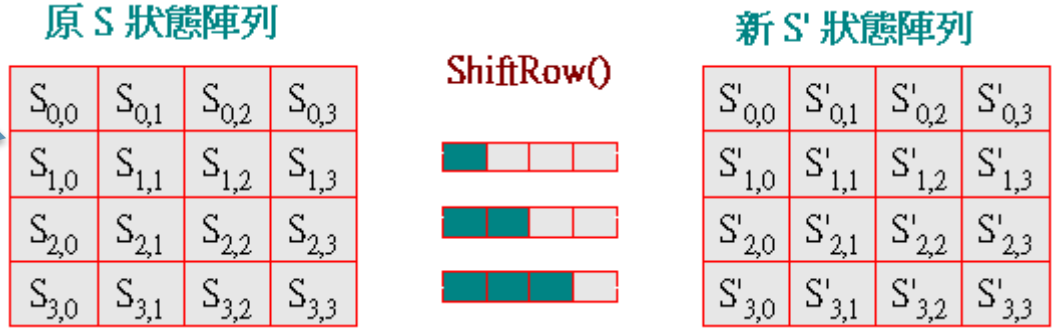


Add key：密钥加

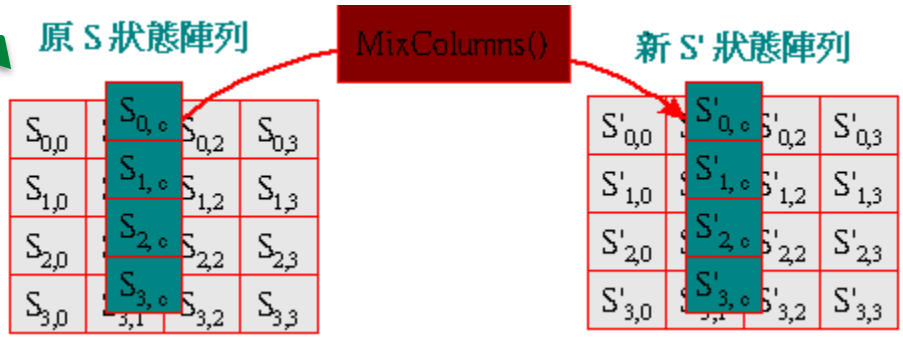
S-box



Shift row：行位移



Mix column：列混合



S-Box

		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

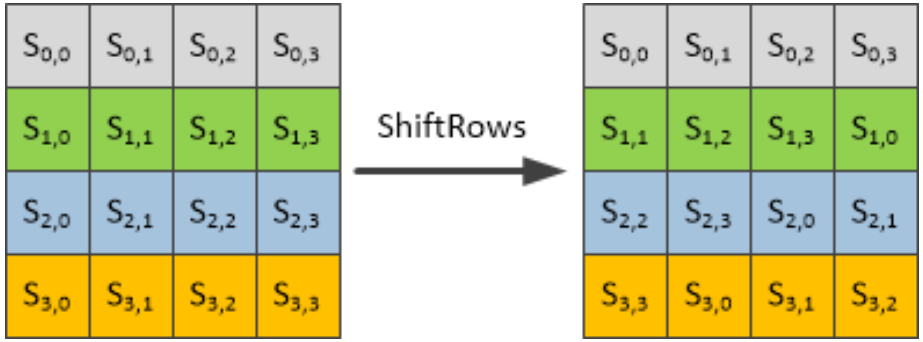
(a) S-box

S-Box逆

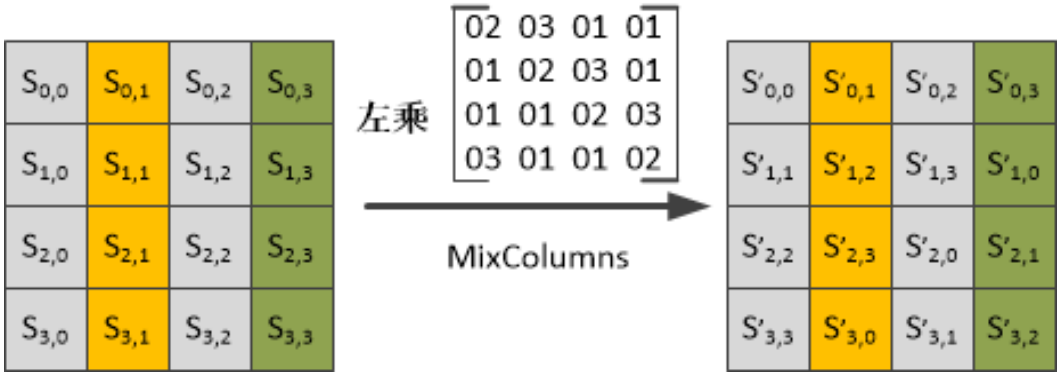
x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	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

(b) Inverse S-box

行位移原理


S0,0 S0,1 S0,2 S0,3
S1,0 S1,1 S1,2 S1,3
S2,0 S2,1 S2,2 S2,3
S3,0 S3,1 S3,2 S3,3
ShiftRows
S0,0 S0,1 S0,2 S0,3
S1,1 S1,2 S1,3 S1,0
S2,2 S2,3 S2,0 S2,1
S3,3 S3,0 S3,1 S3,2

列混淆原理



- 每个字节对应的值只与该列的4个值有关系。此处的乘法和加法都是定义在GF(2^8)上的，需要注意：
  - 1) 将某个字节所对应的值乘以2，其结果就是将该值的二进制位左移一位，如果原始值的最高位为1，则还需要将移位后的结果异或00011011
  - 2) 乘法对加法满足分配率，例如：
    $07 \cdot S_{0,0} = (01 \oplus 02 \oplus 04) \cdot S_{0,0} = S_{0,0} \oplus (02 \cdot S_{0,0})(04 \cdot S_{0,0})$
  - 3) 此处的矩阵乘法与一般意义上矩阵的乘法有所不同，各个值在相加时使用的是模2^8加法（异或运算）

逆列混淆原理

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix}$$

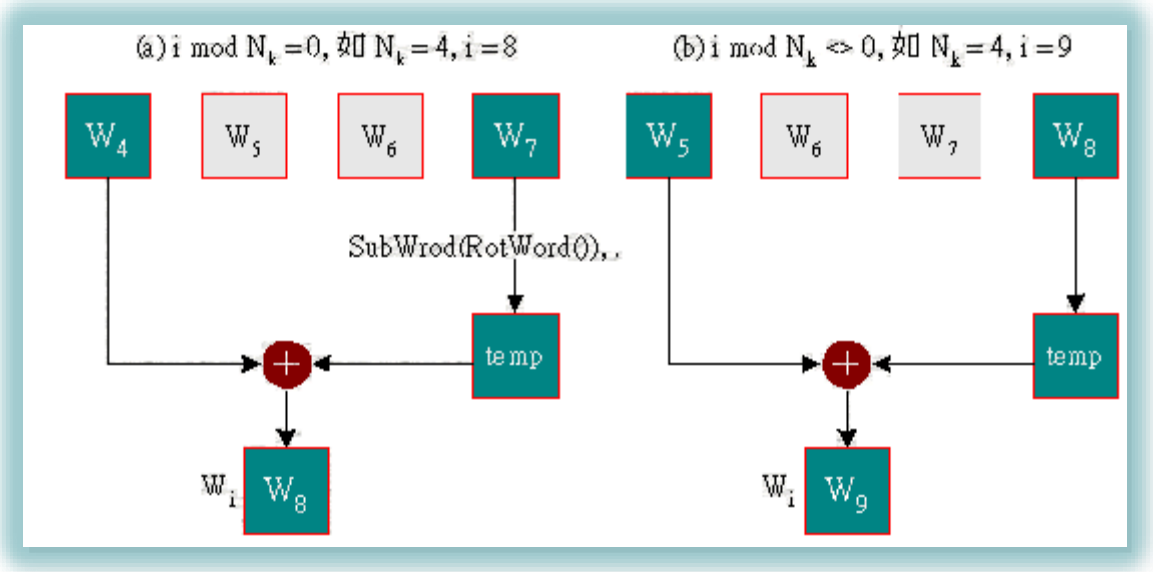
密钥扩展

密钥字符单位：W[i]，32 bits

密钥字符数量：$N_b * (N_r + 1)$

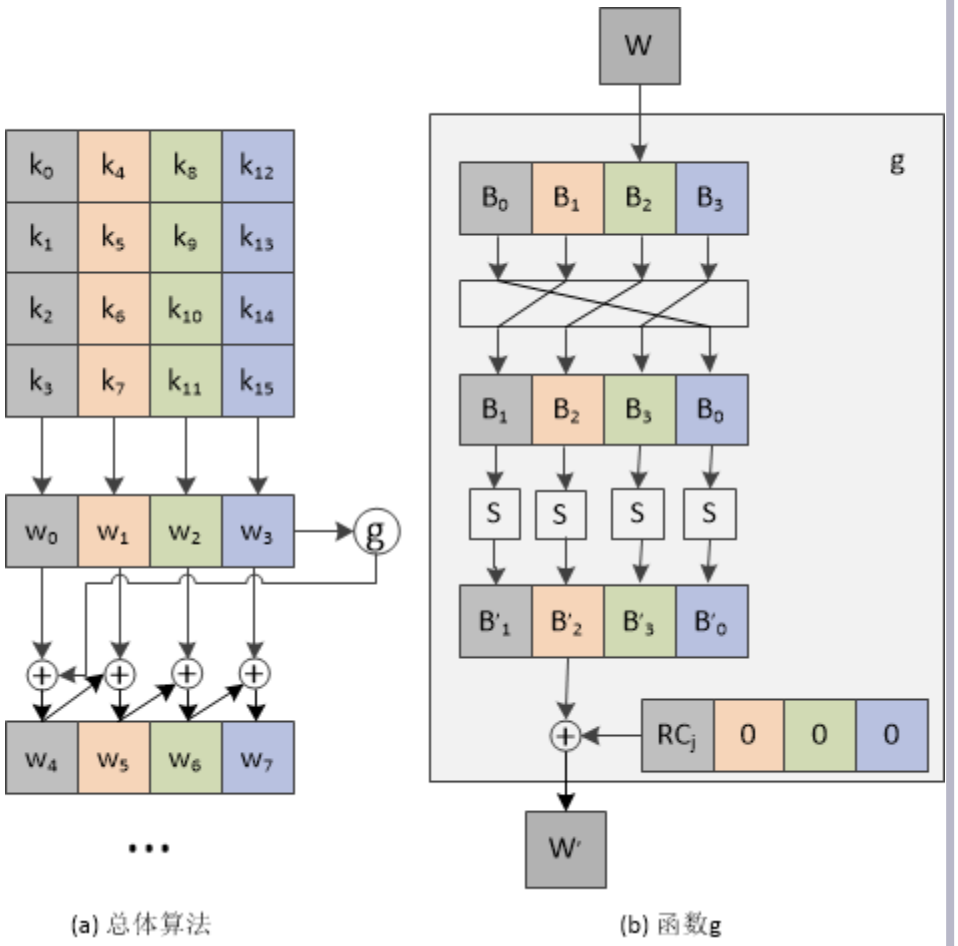
AES-128 需要：44 个密钥字符

每一轮使用 4 个字符



算法中的密钥

- 密钥写成4行，N_k列（N_k = 4,6, or 8）
- 输入密钥扩展成32bits的44/52/60 字的阵列
- 每4个不同的字作用于每一轮的密钥



(a) 总体算法　　(b) 函数g

- 播放动画：AES加密过程

- 课堂作业：
- AES明文输入：$a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_9\ a_{10}\ a_{11}\ a_{12}\ a_{13}\ a_{14}\ a_{15}\ a_{16}$……,在一轮运算中的阵列如何排,写出结果阵列

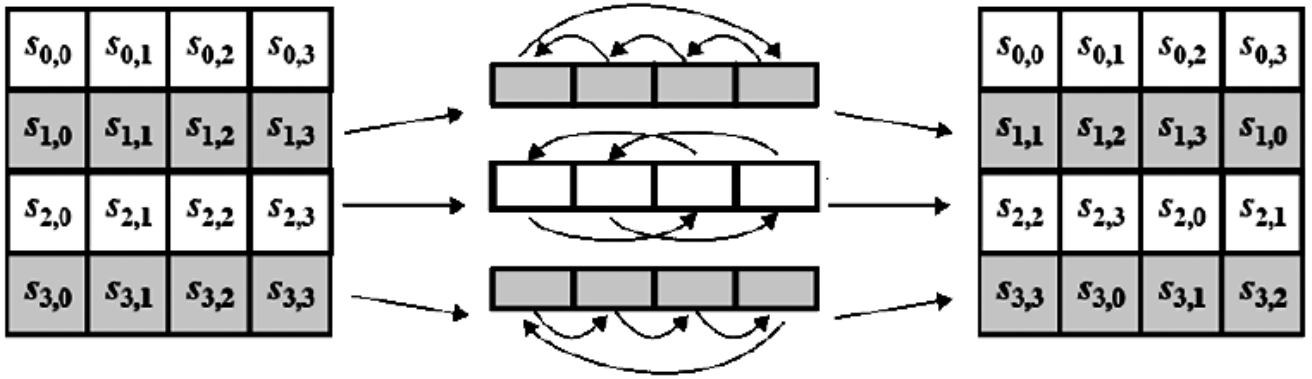
- 作业：采用自己的动画的方式，给出AES操作过程原理
- 下次课提交：发邮箱

对S盒的评价

- S盒设计成能够防止各种已有的密码分析攻击
- 输入位和输出位之间几乎没有相关性
- 输出值不能利用一个简单数学函数变换输入值得到
- 变换中常数的选择使得S盒中没有不动点［S盒(a)=a］，也没有反不动点［S盒(a)= $\bar{a}$ ］。这里 $\bar{a}$ 指a的反。
- S盒是可逆的：逆S盒(S盒(a))=a
- S盒是不自逆的：S盒(a)≠逆S盒(a)

行移位评价

- 正向行移位变换
  - State的第一行不变，第二行循环左移一个字节，第三行循环左移两个字节，第四行循环左移三个字节



- 逆向行移位变换
- 将左移改为右移

- 评价
- 将某个字节从一列移到另一列中，确保某列中的4字节扩展到了4个不同的列

对列混淆的评价

- 变换矩阵的系数是基于在码字间有最大距离的线性编码
- 正向变换的系数$(01)_{16}(02)_{16}(03)_{16}$考虑到算法执行效率
  - 系数的乘法至多涉及一次移位、一次异或
  - 各项和需要三次异或、无需GF(2^8)中的模运算
  - 不考虑逆向变换的算法执行效率。加密被视为比解密重要
- 在每列的所有字节中有良好的混淆性
- 列混淆和行移位变换使得经过几轮迭代后，所有输出位均与所有输入位相关

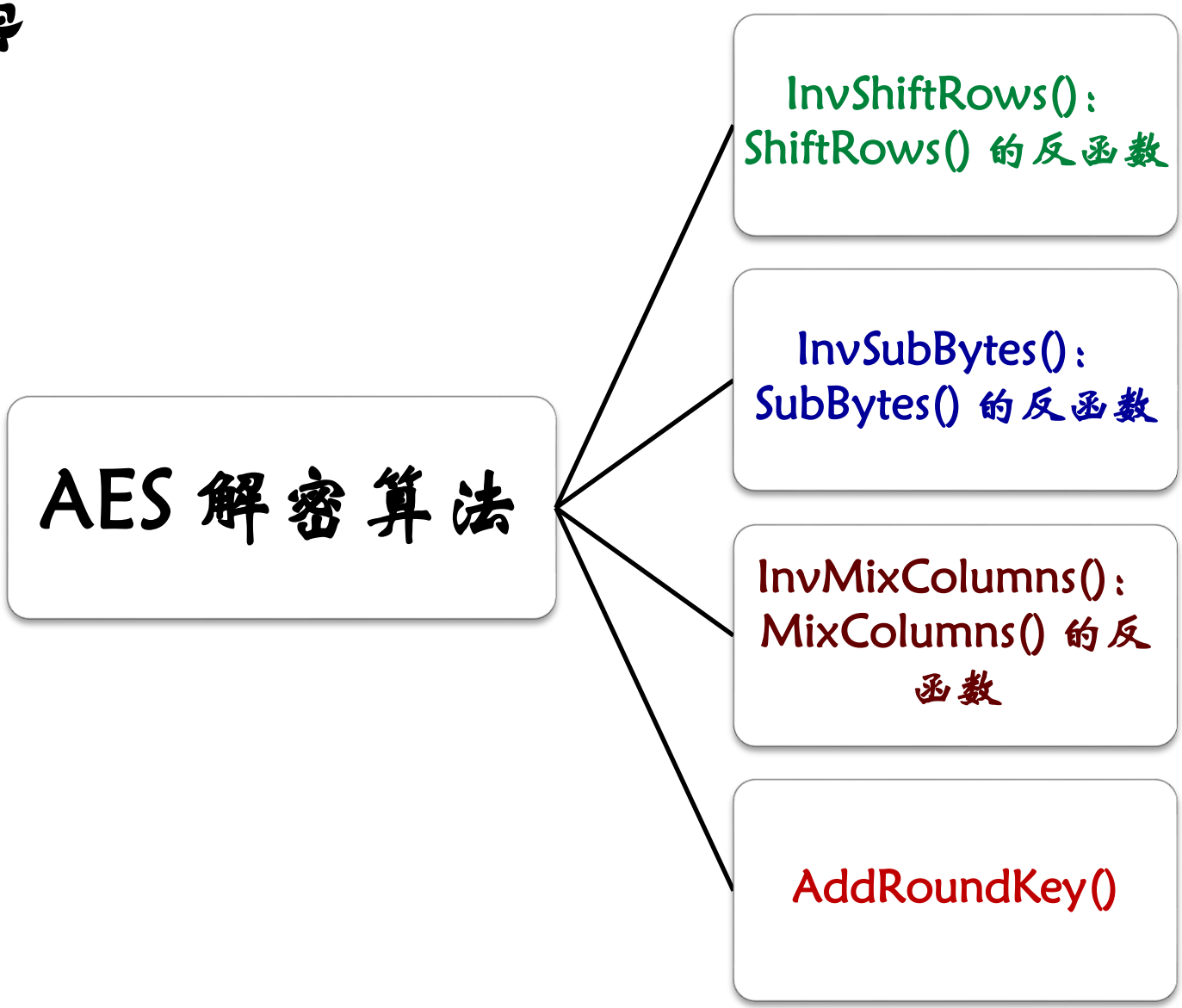
密钥扩展的评价

- 已知密钥或轮密钥的部分，不能计算出轮密钥的其他位
- 是一个可逆变换：知道扩展密钥中任何连续的N_k个字能够重建整个扩展密钥，N_k是构成密钥所需的字数
- 能够在各种处理器上有效地执行
- 使用轮常量排除对称性
- 将密钥的差异性扩散到轮密钥中：密钥的每个位能影响到轮密钥的一些位
- 足够的非线性：轮密钥的差异与密钥的差异关系复杂
- 易于描述

解密运算

- 解密过程中各变换的顺序与加密中变换的顺序不同
- 解密的一个等效版本与加密算法有相同的结构，变换顺序相同（但用逆向变换取代正向变换）

解密


AES 解密算法
InvShiftRows()：
ShiftRows() 的反函数
InvSubBytes()：
SubBytes() 的反函数
InvMixColumns()：
MixColumns() 的反函数
AddRoundKey()

AES的安全性分析—国际层面

- 美国NSA审核了所有的参与竞选AES的最终入围者（包括Rijndael），认为均能满足美国政府传递非机密文件的安全需要。2003.6，美国政府宣布AES可以用于加密机密文件：AES加密算法（使用128，192，和256比特密钥的版本）的安全性，在设计结构及密钥的长度上俱已到达保护机密信息的标准。最高机密信息的传递，则至少需要192或256比特的密钥长度。用以传递国家安全信息的AES实现产品，必须先由国家安全局审核认证，方能被发放使用。
- 这标志着，由NSA批准在最高机密信息上使用的加密系统首次可以公开使用。许多大众化产品只用128位密钥当作默认值；由于最高机密文件的加密系统必须保证数十年以上的安全性，故推测NSA可能认为128位太短，才以更长的密钥长度为最高机密的加密保留了安全空间。
- 截至2006年，针对AES唯一成功的攻击是旁道攻击。

AES的安全性分析 —技术层面

轮数	数据量	时间复杂度	空间复杂度	方法	时间
7	2^{36}	2^{172}	2^{32}	Square	2000
7	$2^{127.997}$	2^{120}	2^{64}	Square	2000
7	2^{32}	2^{200}	2^{32}	Square	2000
7	2^{32}	2^{184}	2^{140}	Square-functional	2000
7	$2^{92.5}$	$2^{250.5}$	2^{153}	Impossible	2004
7	$2^{115.5}$	2^{119}	2^{45}	impossible	2007
7	$2^{113.8}$	$2^{118.8}$ MA	$2^{89.2}$	impossible	2008
7	2^{92}	2^{163} MA	2^{61}	impossible	2008
7	2^{34+n}	$2^{74+n}+2^{208-n}$ precomp.	2^{206-n}	MitM	2008
7	2^{80}	$2^{113}+2^{123}$ precomp.	2^{122}	MitM	2009
8	$2^{127.997}$	2^{204}	2^{1044}	Square	2000
8	$2^{116.5}$	$2^{247.5}$	2^{45}	impossible	2007
8	$2^{89.1}$	$2^{229.7}$ MA	2^{97}	impossible	2008
8	$2^{111.1}$	$2^{227.8}$ MA	$2^{112.1}$	impossible	2008
8	2^{34+n}	$2^{202+n}+2^{208-n}$ precomp.	2^{206-n}	MitM	2008
8	2^{80}	2^{241}	2^{123}	MitM	2009
8	2^{113}	2^{196}	2^{129}	Square-multiset	2010

- 破解一个分组加密系统最常见的方式，是先对其较弱版本（加密循环次数较少）尝试各种攻击
- AES中，128位Key：10个加密循环；192位Key：12个加密循环；256位Key：14个加密循环
- 左图：对AES-256的分析

AES的安全性争议

- 由于已破解的弱版AES的加密轮数和原来的加密轮数近，有密码学家担心AES的安全性：要是能将该攻击加以改进，AES就会被破解。密码学意义上，只要存在一种方法，比穷举法更有效率，就能被视为一种“破解”。所以针对AES-128的攻击若需要2^{120}计算复杂度（少于穷举法2^{128}），AES-128就算被破解了；即便该方法在目前还不实用。从应用的角度来看，这种程度的破解依然太不切实际。最著名的暴力攻击法是distributed.net针对64位密钥RC5所作的攻击。该攻击在2002年完成。根据摩尔定律，到2005年12月，同样的攻击应该可以破解66比特密钥的RC5。
- 其他的争议着重于AES的数学结构。AES具有相当的代数结构。虽然相关的代数攻击尚未出现，但许多学者认为，把安全性创建于未经透彻研究过的结构上是有风险的。Ferguson，Schroeppel 和 Whiting 因此写道：“...我们很担心 Rijndael [AES] 算法应用在机密系统上的安全性。”
- 2002年，Nicolas Courtois 和 Josef Pieprzyk发表名为XSL 攻击的理论性攻击，试图展示AES潜在的弱点。但几位密码学专家[谁？]发现该攻击的数学分析有问题，推测应是作者的计算有误。因此，这种攻击法是否对AES奏效，仍是未解之谜。就现阶段而言，XSL攻击AES的效果不十分显著，故将之应用于实际的可能性并不高。

旁道攻击

- 旁道攻击，又称旁路攻击、侧信道攻击，不攻击密码本身，而是攻击那些基于不安全系统（会在不经意间泄漏信息）上的加密系统。
- 2005年4月，D.J. Bernstein公布了一种缓存时序攻击法，以此破解了一个装载OpenSSL AES加密系统的客户服务器。为了设计使该服务器公布所有的时序信息，攻击算法使用了2亿多条筛选过的明码。对于需要多跳的互联网而言，这样的攻击方法并不实用。 Bruce Schneier称此攻击为“好的时序攻击法”。
- 2005年10月，Eran Tromer和另外两个研究员发表了一篇论文，展示了数种针对AES的缓存时序攻击法。其中一种攻击法只需要800个写入动作，费时65毫秒，就能得到完整的AES密钥。但攻击者必须在运行加密的系统上拥有运行程序的权限，方能以此法破解该密码系统。

分组密码工作模式

- 为将分组密码应用于各种实际场合，NIST在800-38A中推荐5种工作模式

模式	描述	典型应用
电码本(ECB)	用相同的密钥分别对明文组加密	• 单个数据的安全传输
密文分组链接(CBC)	加密算法的输入是上一个密文组和本次明文组的异或	• 普通目的的面向分组的传输 • 认证
密文反馈(CFB)	上一块密文作为加密算法的输入，产生j位伪随机数与明文异或	• 普通目的的面向分组的传输 • 认证
输出反馈(OFB)	与CFB基本相同，只是加密算法的输入是上一次DES的输出	• 噪声频道上的数据流的传输（如卫星通信）
计数器(CTR)	每个明文分组都与一个加密计数器相异或。对每个后续分组计数器递增	• 普通目的的面向分组的传输 • 用于高速需求

分组密码的操作模式

- 分组预处理：待加密的明文长度是随机的，当按要求进行分组时，最后一组消息的长度可能不足要求的长度，这时需要填充明文。
- 通常要留出最后一字节来说明所填充的字节数。

分组预处理：明文填充

- 明文末尾可能有不足一个分组的数据，需要填充
- 填充随机数据
  - 需要在消息开头明确消息实际长度
  - 或约定消息结束标志
- 填充固定格式数字
  - 例如：[b1 b2 b3 0 0005]表示有3字节有效数据，填充5个字节
  - 这可能导致需要一个新的分组；
  - 当恰好不需要填充时，仍需补充一个完成分组，以避免歧义
- 其他填充方式

分组码的操作模式

2001年，NIST发布AES的五种操作模式 文件

四种最常用的模式：ECB, CBC, CFB, OFB

最常见的分组密码操作模式

一、分组模式：

Electronic Code Book (ECB)：电码本模式

Cipher-Clock Chaining(CBC)：密码分组连接模式

二、序列模式：典型的用分组密码实现序列密码的应用模式

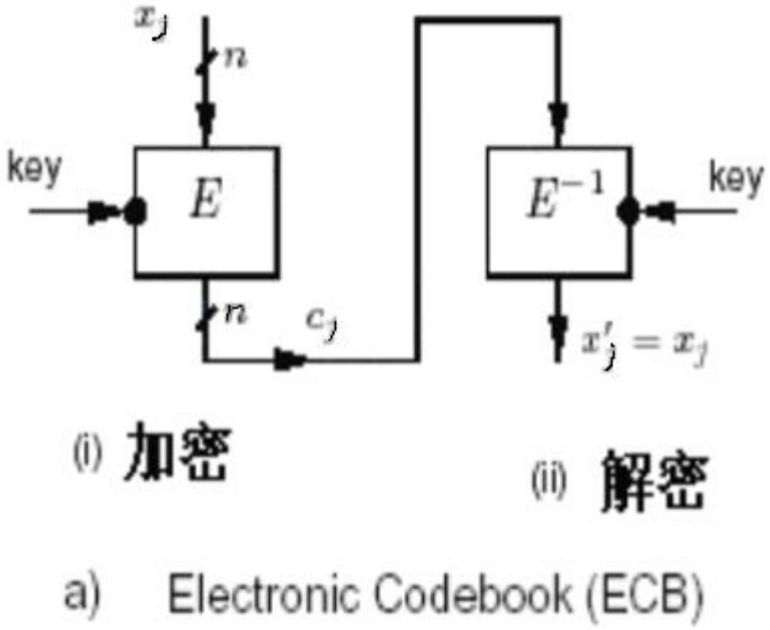
Cipher Feed Back(CFB)：密码反馈模式

Output Feed Back(OFB)：输出反馈模式

三、计数模式：

Counter (CTR)(CM)：也称整数计数模式、分段块模式

ECB：电码本模式



相同的明文块在相同的密钥作用下产生相同的密文

(明文相同性)

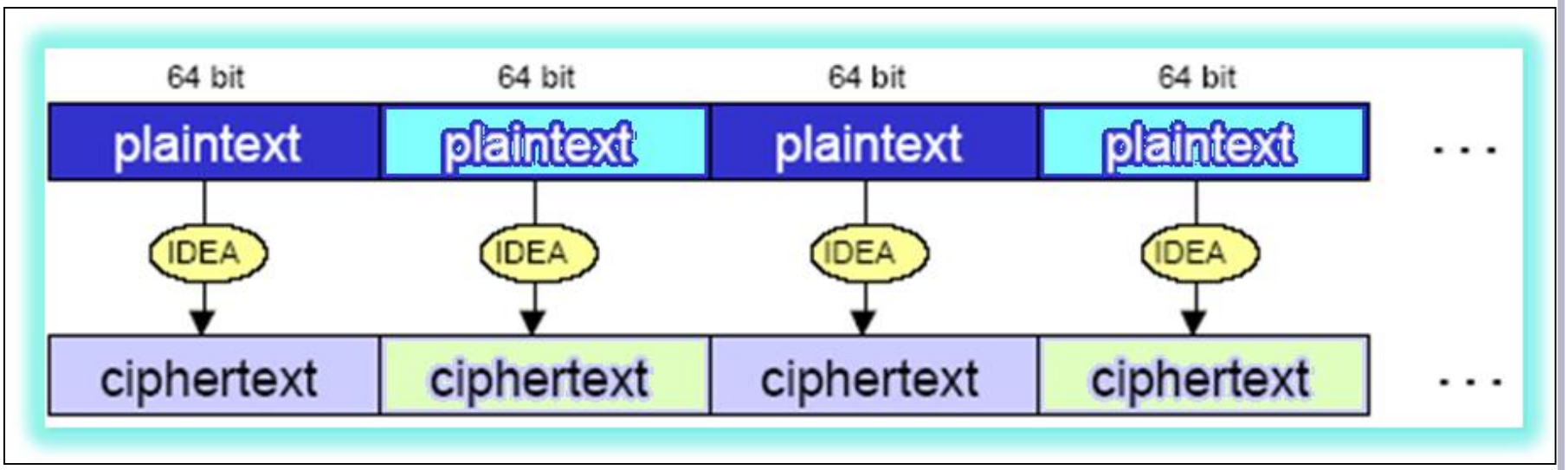
明文块相互独立的加密，密文块重新排序→相应明文重新排序

(前后关联性)

一个密文块的bits错误→本块解密错，约50%的明文比特错误

(错误增殖)

ECB举例


64 bit
64 bit
64 bit
64 bit
plaintext
plaintext
plaintext
plaintext
IDEA
IDEA
IDEA
IDEA
ciphertext
ciphertext
ciphertext
ciphertext

ECB模式的缺陷分析

第一、若$M_{i+t}=M_i$，则必有$C_{i+t}=C_i$，由此在密文中就暴露了明文的数据格式和特征。因为一般况下：

❶明文数据一般具有固定的格式、或大量的重复和较长的零串，重要的数据常常在同一位置上出现；

❷明文数据开头通常是格式化的报头，内含发送者、接收者、通信地址、作业号、发报时间等信息。所有这些数据格式和统计特征均会在ECB模式的密文中暴露出来。

ECB模式的缺陷

第二、密码分析者在未知密钥的情况下：

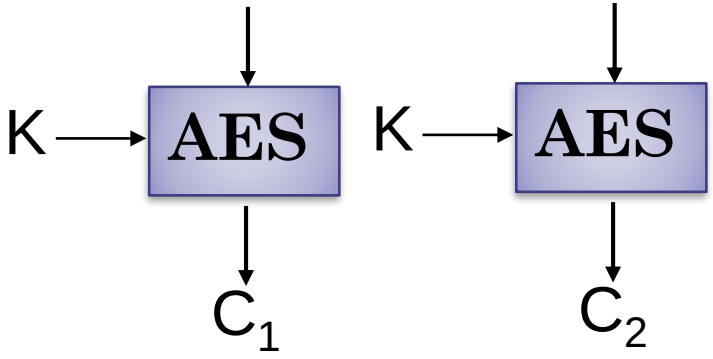
❶利用大量积累的明密文对(M,C)，自行编制译码本，以便对今后截取的密文直接解密；

❷利用已知的明密文对进行替换、嵌入或删除等，从而进行一系列的主动攻击。

ECB模式的替换攻击举例

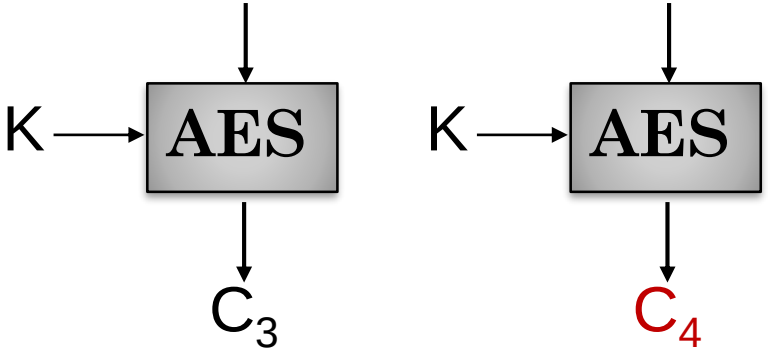
原文

M_1=Pay to J.Jones　　M_2=$1000

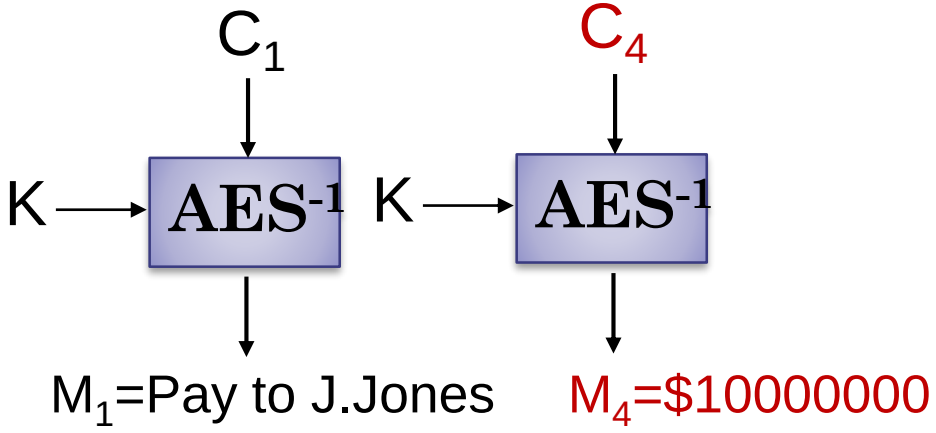


篡改后

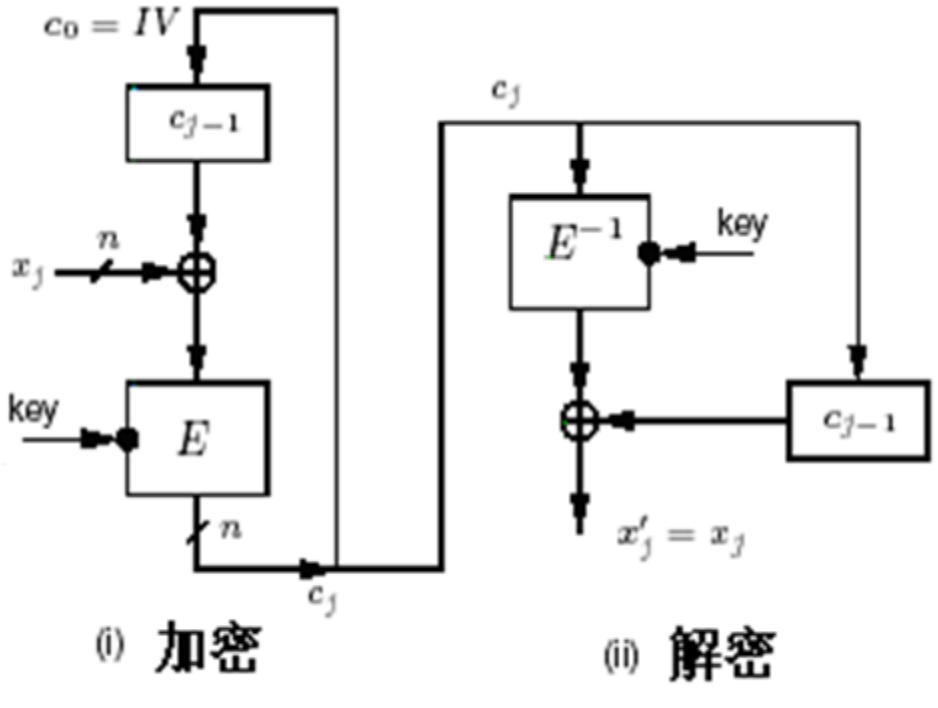
M_3=Pay to S.Smith　　M_4=$10000000



接收解密后



CBC：密码分组连接模式



b) Cipher-block Chaining (CBC)

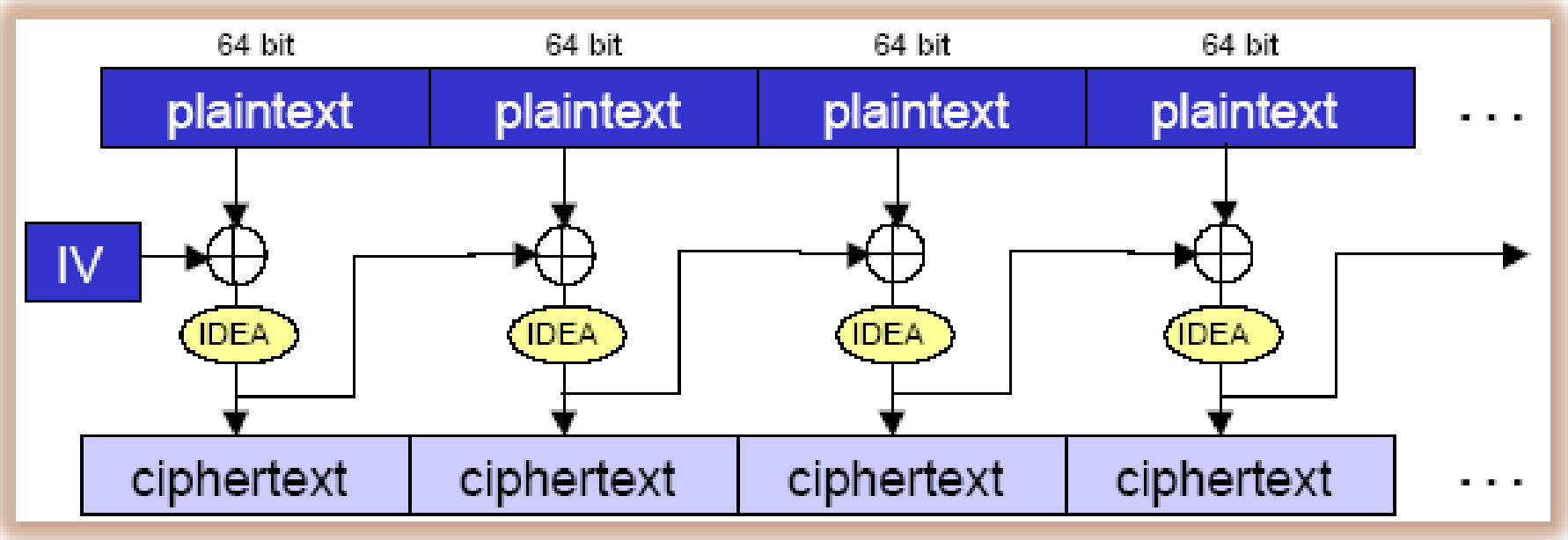
相同的明文在相同的**密钥和IV**作用下，才产生相同的密文

链式机制使得密文c_j与明文x_j以及前面所有的明文块都有关系

密文块中的一比特位的错误→本身和随后的整个密文块的解密

错误恢复。CBC模式是**自同步**

CBC举例


64 bit
64 bit
64 bit
64 bit
plaintext
plaintext
plaintext
plaintext
IV
IDEA
IDEA
IDEA
IDEA
ciphertext
ciphertext
ciphertext
ciphertext

实际应用CBC模式时的注意点

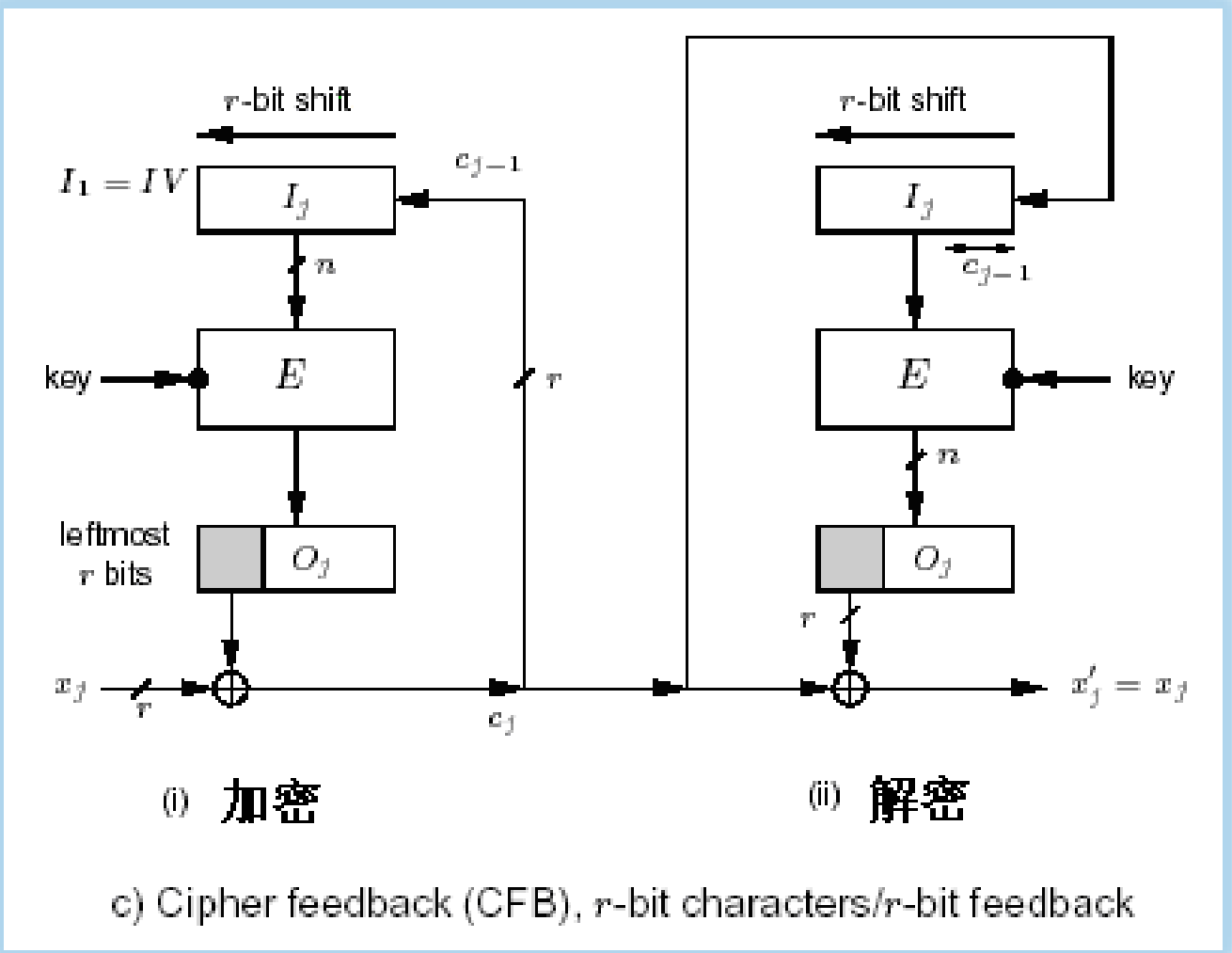
❶不同的明文信息应使用不同的初始向量IV。因为，若不然，即使是两份不同的明文信息,它们开始的若干组也可能是相同的（如相同的报头格式），从而在相应的两份密文中，开始的若干组也是相同的，由此会给密码分析者提供一些有用的线索；

❷初始向量IV的作用主要是为了填充存储的初值，从而实现密文的反馈，而保密的作用不大，所以可在信道上明传。如果不明传，效果是加大密钥

CBC模式优缺点

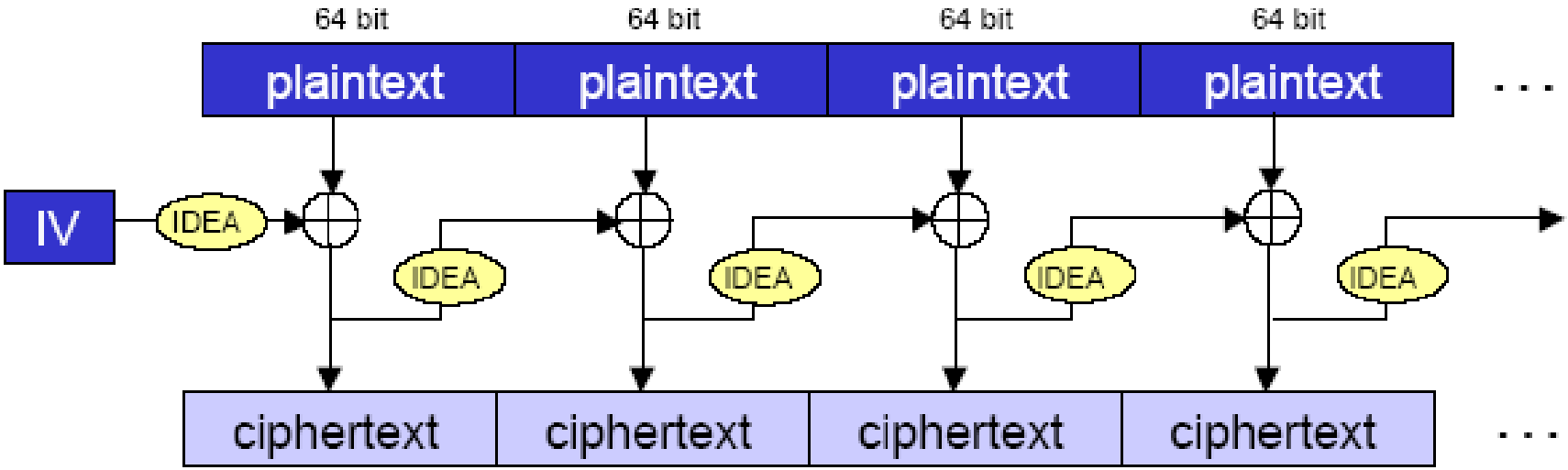
- **优点：** 克服了ECB模式存在的两个缺陷。
- **不足：** **存在错误扩散**，若在信道上传送的第i组密文C_i出现1bit错误，则在解密时，将引起M_i全错及M_{i+1}出现错误；此外，$M_j(j>i+1)$将不再受此错误影响，系统自动恢复正常。
- CBC模式的错误扩散不大(至多影响两个明文组的（分组长+1）比特)，但对**传输中的同步差错(增加或丢失若干比特)很敏感**。因而要求系统具有良好的帧同步，为防止这类错误酿成恶果，**系统还应采取纠错技术。**

CFB密码反馈模式



c) Cipher feedback (CFB), r-bit characters/r-bit feedback

CFB举例


64 bit
64 bit
64 bit
64 bit
plaintext
plaintext
plaintext
plaintext
IV
IDEA
IDEA
IDEA
IDEA
IDEA
ciphertext
ciphertext
ciphertext
ciphertext

CFB模式的优点

- 须预置移位寄存器的初态(称为初始向量IV)，可在信道上明传给接收方。

CFB模式的优点：

❶ 适合用户的数据格式。

❷ 防篡改。由于密文反馈的作用，象CBC模式一样，该模式能隐蔽明文的数据特征，也可以检测出对密文的篡改

❸ 自同步。CFB是典型的自同步序列密码：只要接收方连续收到[分组长/n]组正确的密文，收发双方的分组长级移位寄存器存储的状态就完全一样，从而双方可重新建立起同步。

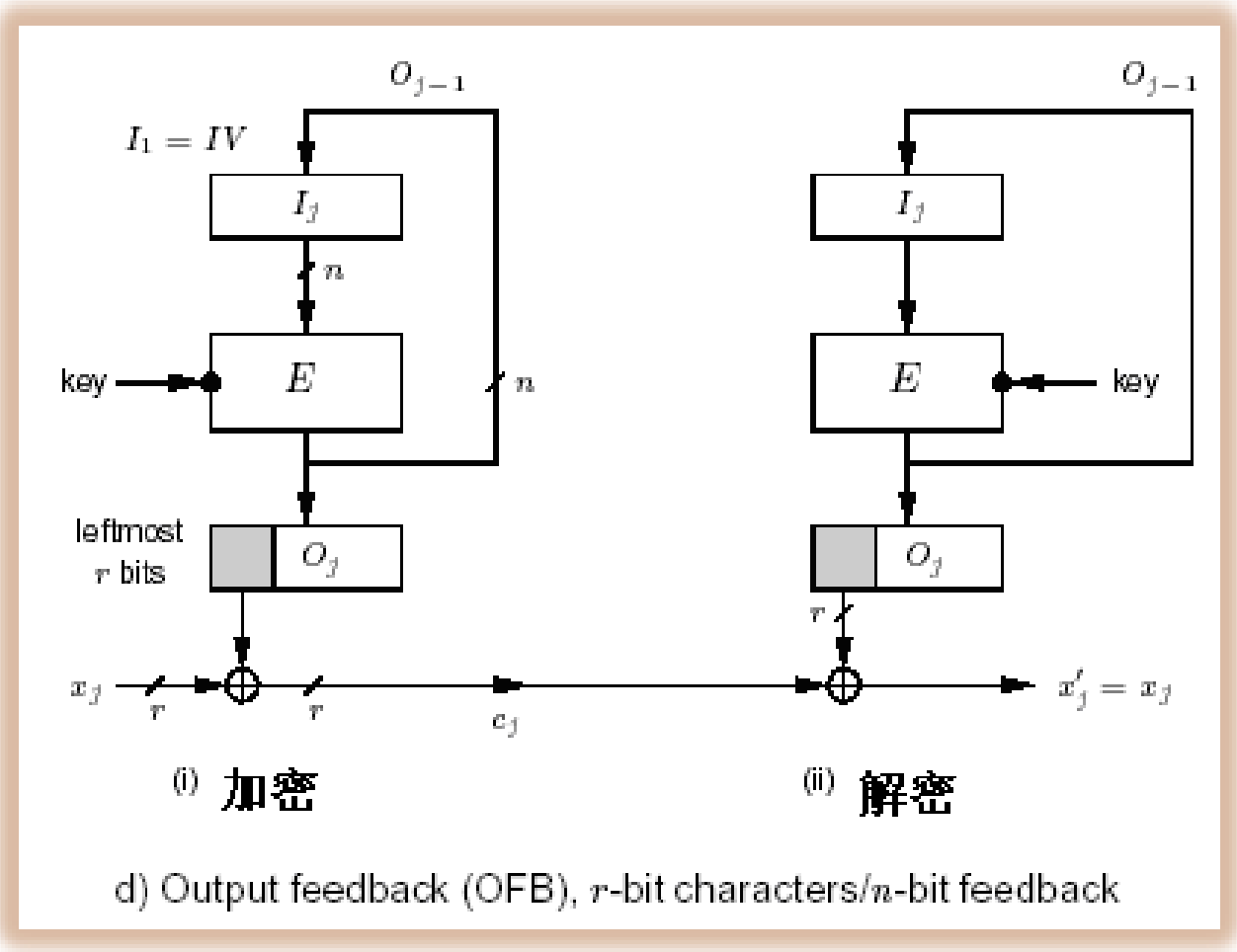
CFB模式的不足

存在有限的(其实是[分组长/n]+1组)错误扩散：当传输的密文组C_i出现1bit错误时，解密的明文组M_i也有1bit错误，而且随后解密出来的[分组长/n]组明文

$$M_{i+1}, M_{i+2}, \cdots, M_{i+[\text{分组长}/n]}$$

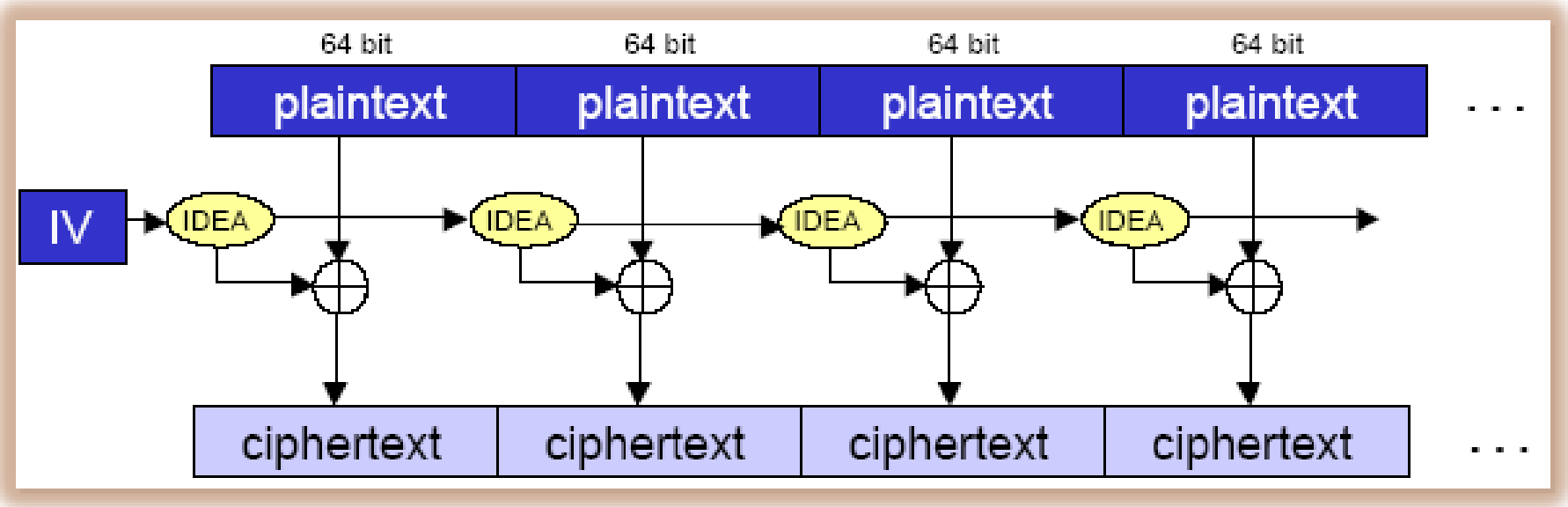
全错，直至此后原C_i的1bit错误刚好移出分组长级移位寄存器，系统可自动恢复正常。

OFB输出反馈模式



d) Output feedback (OFB), r-bit characters/n-bit feedback

OFB举例


64 bit
64 bit
64 bit
64 bit
plaintext
plaintext
plaintext
plaintext
IV
IDEA
IDEA
IDEA
IDEA
ciphertext
ciphertext
ciphertext
ciphertext

OFB模式的优点

❶ 特别适于用户数据格式的需要（密码体制有一个重要设计原则：应尽量避免更改现有系统的数据格式和一些规定）

❷ 具有序列密码的优点：无错误扩散。这对于信息冗余度较大的语音或图象等数据加密处理来讲，比较合适，可以容忍传输和存储过程中产生少量错误

OFB模式的不足

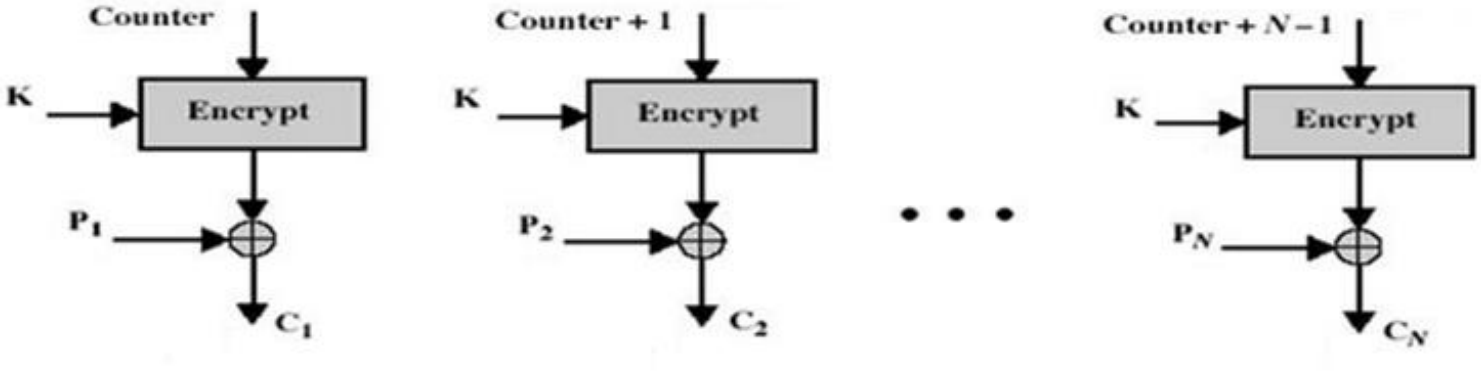
❶ 不具备自同步能力，要求系统保持严格的同步，否则难以解密

❷ 具有序列密码的缺点：无法检测和识别攻击者对密文的篡改。

但由于OFB模式多在同步信道中运行，对手难于知道消息的起止点。由此，这类攻击很难奏效。

CM：计数模式

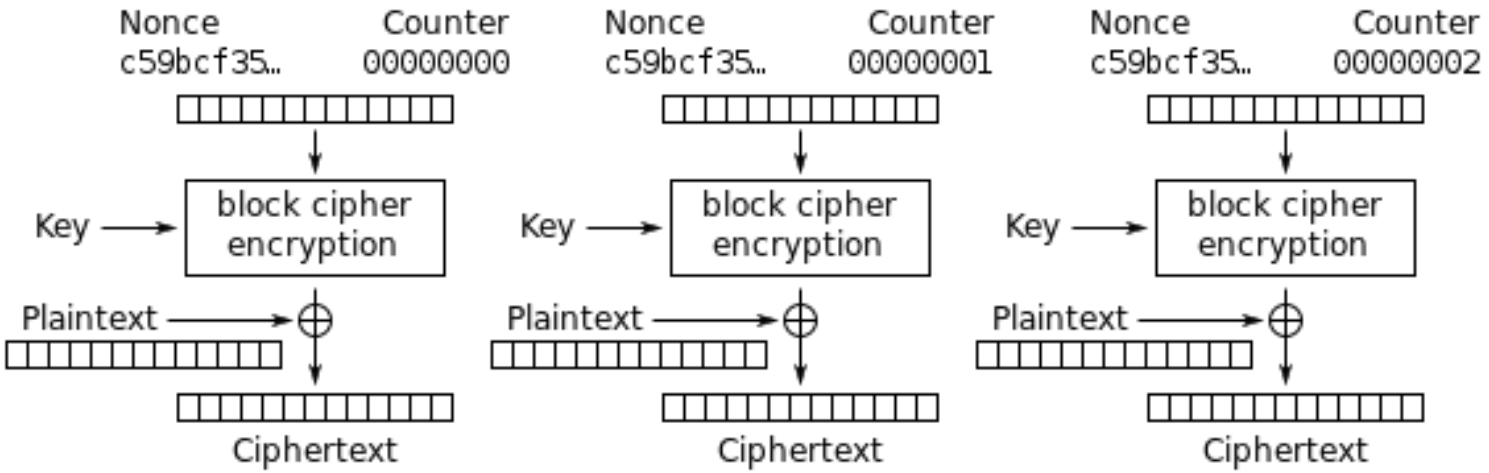
- 与OFB相似。
- 对一系列计数进行加密→一系列的输出块。
- 输出块与明文异或→密文。对于最后的u-bit数据块，将用于异或操作，而剩下的b-u位将被丢弃（b表示分组长度）。
- CTR解密类似。
- 上述过程中计数必须互不相同。计数可以用任意函数产生，为保证它在相当长的一段时间内不重复，**通常使用加1的方式**（即操作一个block后counter加1）



(a) Encryption

举例

- 图中nonce=IV
- 用key将counter加密→ ecounter，将明文与ecounter做异或运算
- counter ← counter+1，再用key将counter加密后得到ecounter，再将明文与ecounter做异或运算
- 重复以上操作

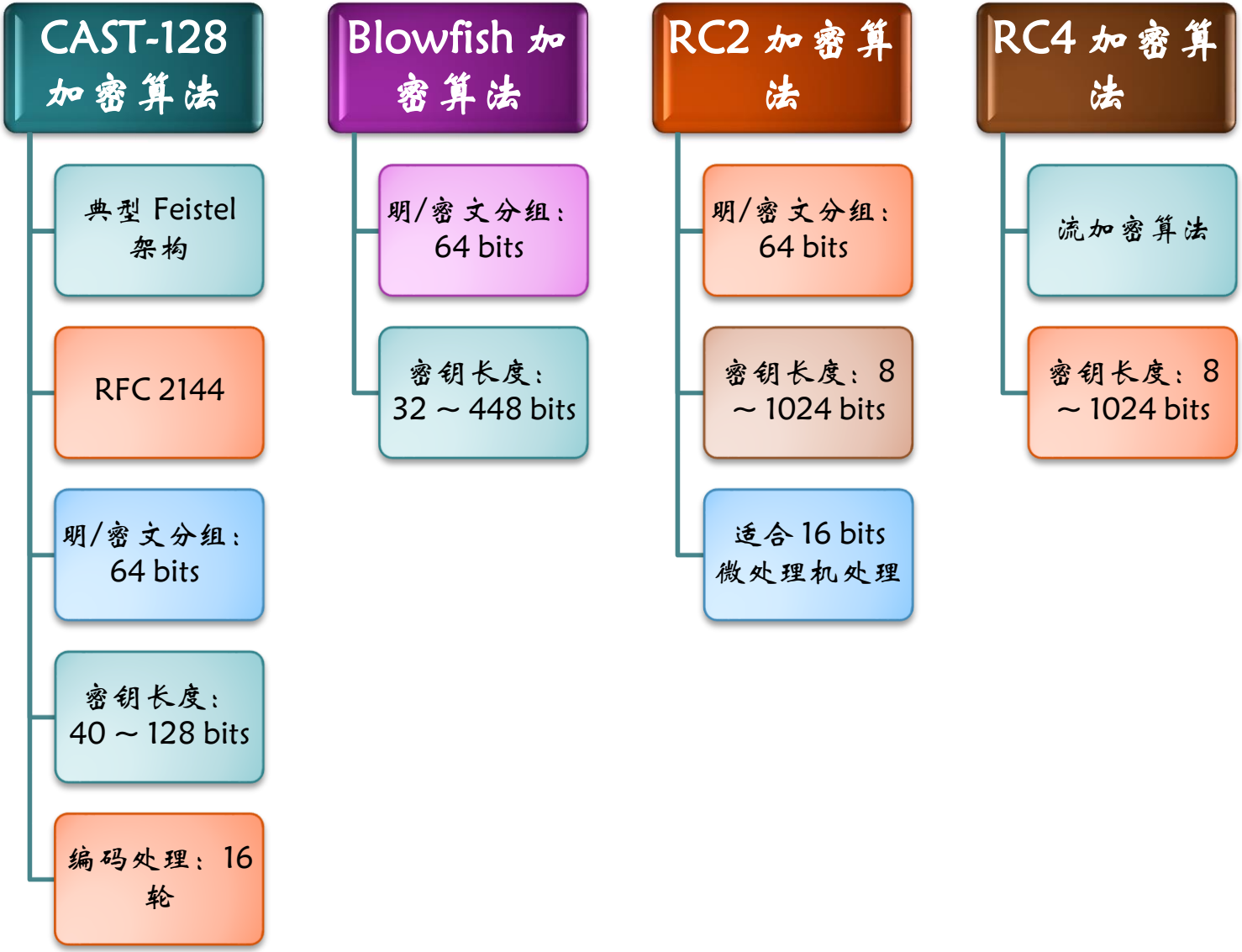


Counter (CTR) mode encryption

特点

- 有自增的counter，对其加密之后和明文异或得到密文，相当于一次一密。
- 广泛用于 ATM 、IPSec应用中
- 硬件效率：可同时处理多块明文/密文。
- 软件效率：可并行计算，可很好地利用 CPU 流水等并行技术。
- 预处理：算法和加密盒的输出不依赖明文和密文，因此如果有足够的保证安全的存储器，加密算法仅仅是一系列异或运算，这将极大地提高吞吐量。
- 随机访问：第 i 块密文的解密不依赖于第 i-1 块密文，提供很高的随机访问能力
- 可证明的安全性：可证明 CTR 至少和其他模式一样安全（CBC, CFB, OFB, ...）
- 简单性：与其它模式不同，CTR模式仅要求实现加密算法，但不要求实现解密算法。对于 AES 等加/解密本质上不同的算法来说，这种简化是巨大的。
- 无填充，可以高效地作为流式加密使用。

其他密码算法


CAST-128 加密算法
典型 Feistel 架构
RFC 2144
明/密文分组：64 bits
密钥长度：40 ~ 128 bits
编码处理：16 轮
Blowfish 加密算法
明/密文分组：64 bits
密钥长度：32 ~ 448 bits
RC2 加密算法
明/密文分组：64 bits
密钥长度：8 ~ 1024 bits
适合 16 bits 微处理机处理
RC4 加密算法
流加密算法
密钥长度：8 ~ 1024 bits

下次课程

- 序列密码