第5讲 高级加密标准

(The Advanced Encryption Standard)

高级加密标准是由美国标准和技术协会（**National Institute of Standards and Technology)NIST**于**2001**年正式公布的，简称为**AES**。**AES**是一个对称的分组密码，目的是取代**DES**使其成为加密性能更好，应用更广泛的新标准。这个新标准采用了两位比利时密码学家**Joan Daemen** 和 **Vincent Rijmen** 的密码算法方案，称之为 **Rijndael** 算法。

AES背景

- **1997年4月15日，（美国）国家标准技术研究所（NIST）发起征集高级加密标准（Advanced Encryption Standard）AES的活动，活动目的是确定一个非保密的、可以公开技术细节的、全球免费使用的分组密码算法，作为新的数据加密标准。**
- **1997年9月12日，美国联邦登记处公布了正式征集AES候选算法的通告。作为进入AES候选过程的一个条件，开发者承诺放弃被选中算法的知识产权。**

  对AES的基本要求是：比三重DES快、至少与三重DES一样安全、数据分组长度为128比特、密钥长度为128/192/256比特。

- **1998年8月12日，在首届AES会议上指定了15个候选算法。**
- **1999年3月22日第二次AES会议上，将候选名单减少为5个，这5个算法是RC6，Rijndael，SERPENT，Twofish和MARS。**
- **2000年4月13日，第三次AES会议上，对这5个候选算法的各种分析结果进行了讨论。**
- **2000年10月2日，NIST宣布了获胜者—Rijndael算法，2001年11月出版了最终标准FIPS PUB197。**

下面讨论AES—Rijndael算法

Rijndael为AES所定义的迭代式的分组密码算法，它的分组长度（block length）和密钥长度（key length）是可以各自独立的，当然它们都可以是128bit，192bit或256bit。

AES参数取决于密钥长度的选择。下面给出AES的参数配置表：

AES参数配置表

密钥长度 (words/bytes/bits)	4/16/128	6/24/192	8/32/256
明文分组长度 (words/bytes/bits)	4/16/128	4/16/128	4/16/128
轮数	10	12	14
（轮回）密钥长 (words/bytes/bits)	4/16/128	4/16/128	4/16/128
扩展密钥总长 (words/bytes)	44/176	52/208	60/240

我们通常限定AES密钥长度为128bit。

Rijndael的数据结构：

（1）不论是明文，还是密文，密钥都是若干字节（1byte=8bits）构成的串（strings）：

$$b_1 b_2 \ldots\ldots b_s$$

易见：

128bits $\longleftrightarrow$ S=16 bytes；

192bits $\longleftrightarrow$ S=24 bytes；

256bits $\longleftrightarrow$ S=32 bytes

（2）将字节串排列成4行的矩阵：

b_1	b_5	b_9	b_{13}	b_{17}	b_{21}	b_{25}	b_{29}
b_2	b_6	b_{10}	b_{14}	b_{18}	b_{22}	b_{26}	b_{30}
b_3	b_7	b_{11}	b_{15}	b_{19}	b_{23}	b_{27}	b_{31}
b_4	b_8	b_{12}	b_{16}	b_{20}	b_{24}	b_{28}	b_{32}

注：1）数据加密过程中产生的中间结果都称为State（状态）。一个状态可以表成4行N_b列的一个矩阵，其中N_b=分组长度/32；矩阵中的每一个系数是一个字节（1byte=8bits）。

注：2）

密钥（cipher key）也表示成一个4行N_k列的矩阵，其中N_k=密钥长度/32。

下面给出，当明文分组为128比特，密钥长也为128比特时的各自状态矩阵：

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} ; \begin{pmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{pmatrix}$$

(3) 加密流程：

```
Rijndael(State,CipherKey){
    KeyExpansion(CipherKey,ExpandedKey);
    AddRoundKey(State,ExpandedKey);
for(i=1;i<Nr;i++){
     ByteSub(State);
     Shift Row(State);
     MixColumn(State);
     AddRoundKey(State,ExpandedKey);
}
ByteSub(State);
Shift Row(State);
AddRoundKey(State,ExpandedKey);
}
```

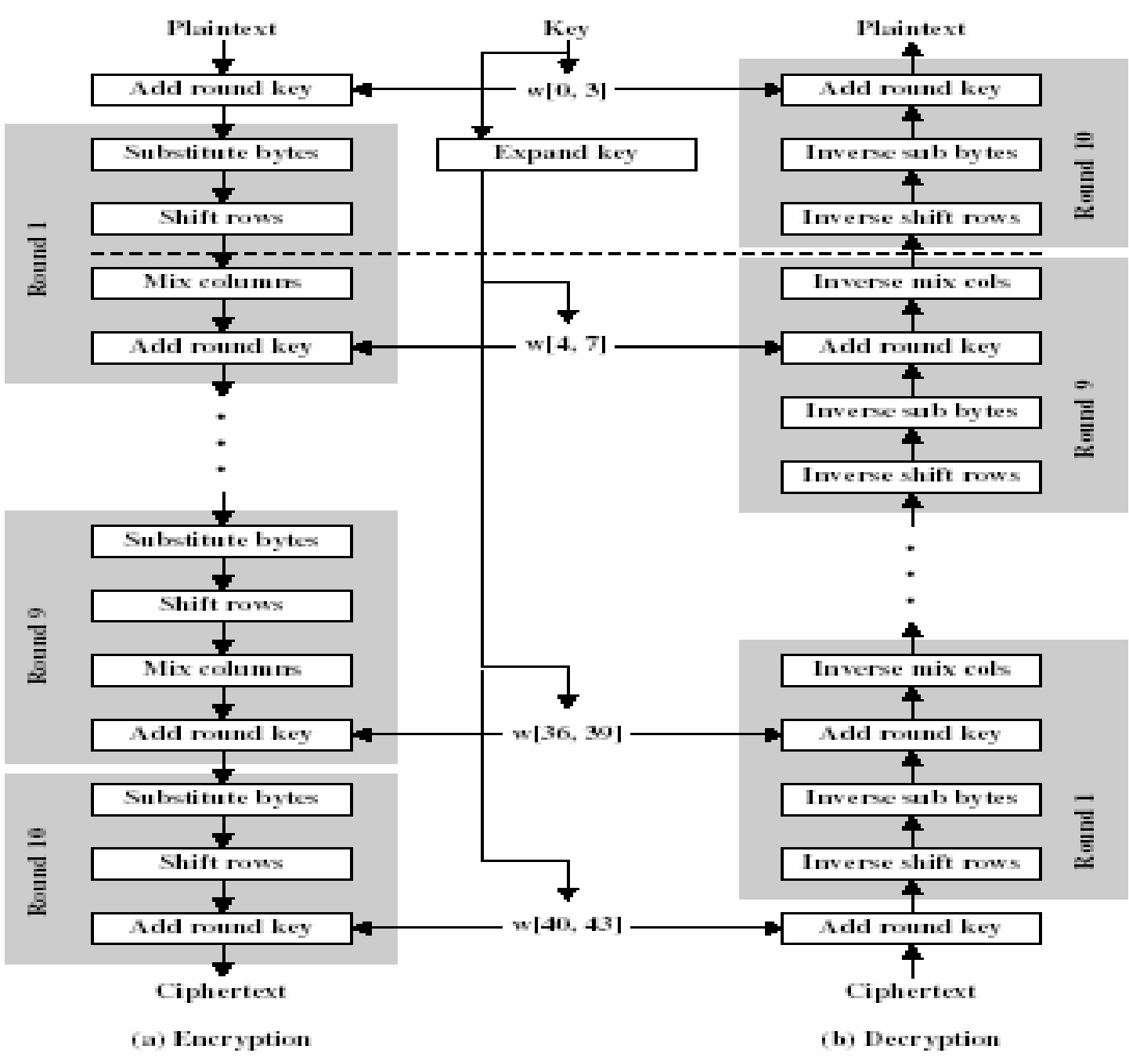
初始化

中间轮

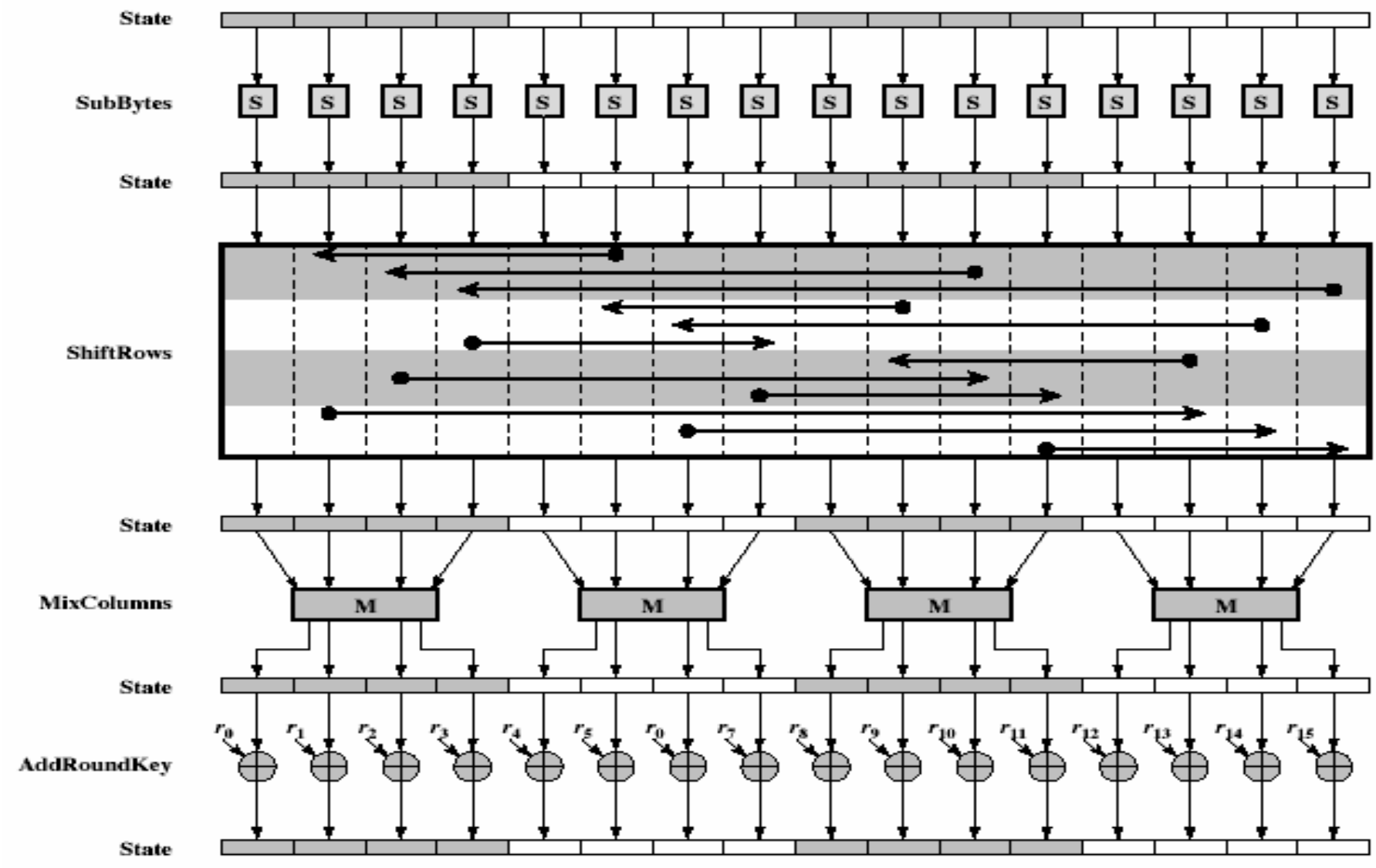
末轮

参见AES加密和解密的图表：

AES-128 加解密过程



AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
State

AES算法结构:

AES算法中没有**Feistel**结构.Feistel结构是对信息数据分组的每一组都对分成两部分进行变换。另一方面，AES中的子变换不是对合变换。 该结构由五个子程序组成:

1 轮密钥加 （**Add Round Key);**

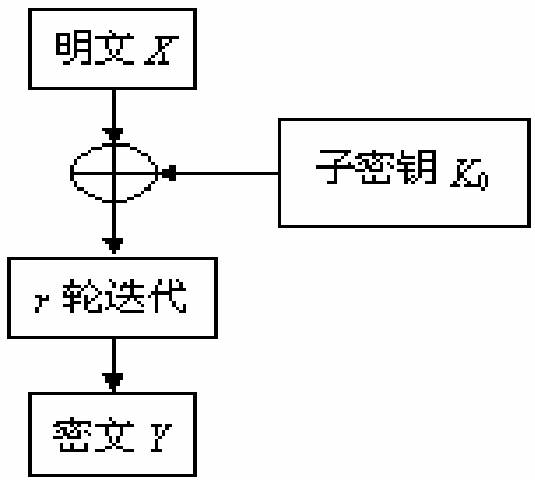
2 Byte替换 **(Byte Sub.) ;**

3 行移位 **(Shift Row) ;**

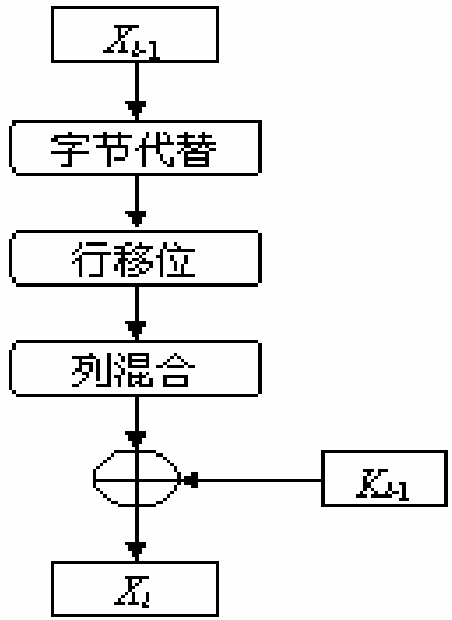
4 列混合 **(Mix Column) ;**

5 密钥扩展**(Key Expansion):**

输入**Cipher Key,**输出子密钥**Expanded Key.**



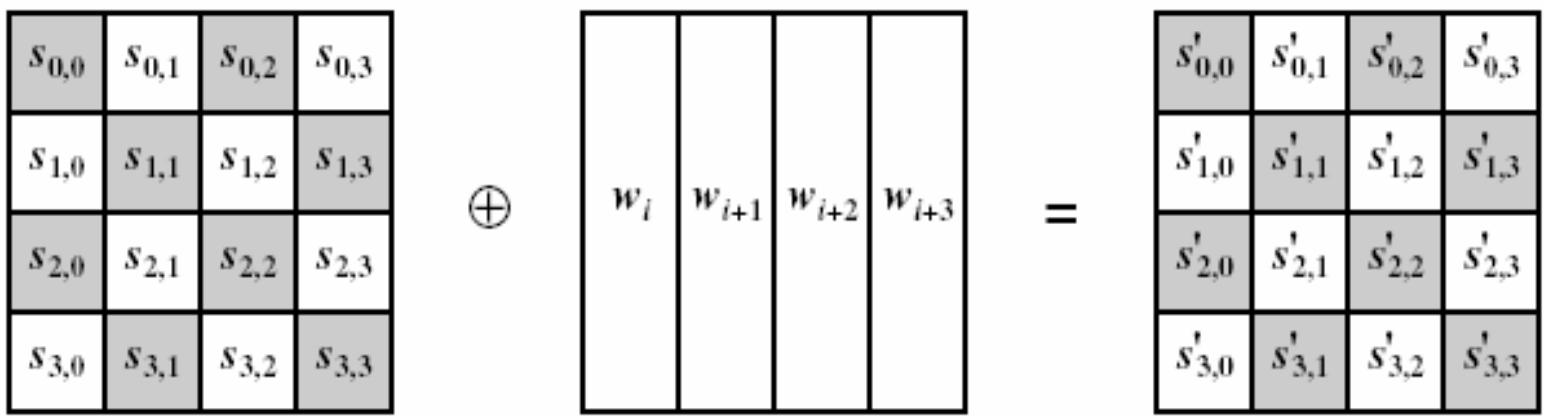
（a）AES 算法框图



（b） 一轮 AES 结构

轮密钥加（Add Round Key)

一个简单地按位异或的操作



(b) Add Round Key Transformation

▲ Add Round Key：

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

输入 **state**

+

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

子密钥

=

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

输出 **state**

$b_{ij} = a_{ij} + k_{ij}$

▲ Byte Sub：

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

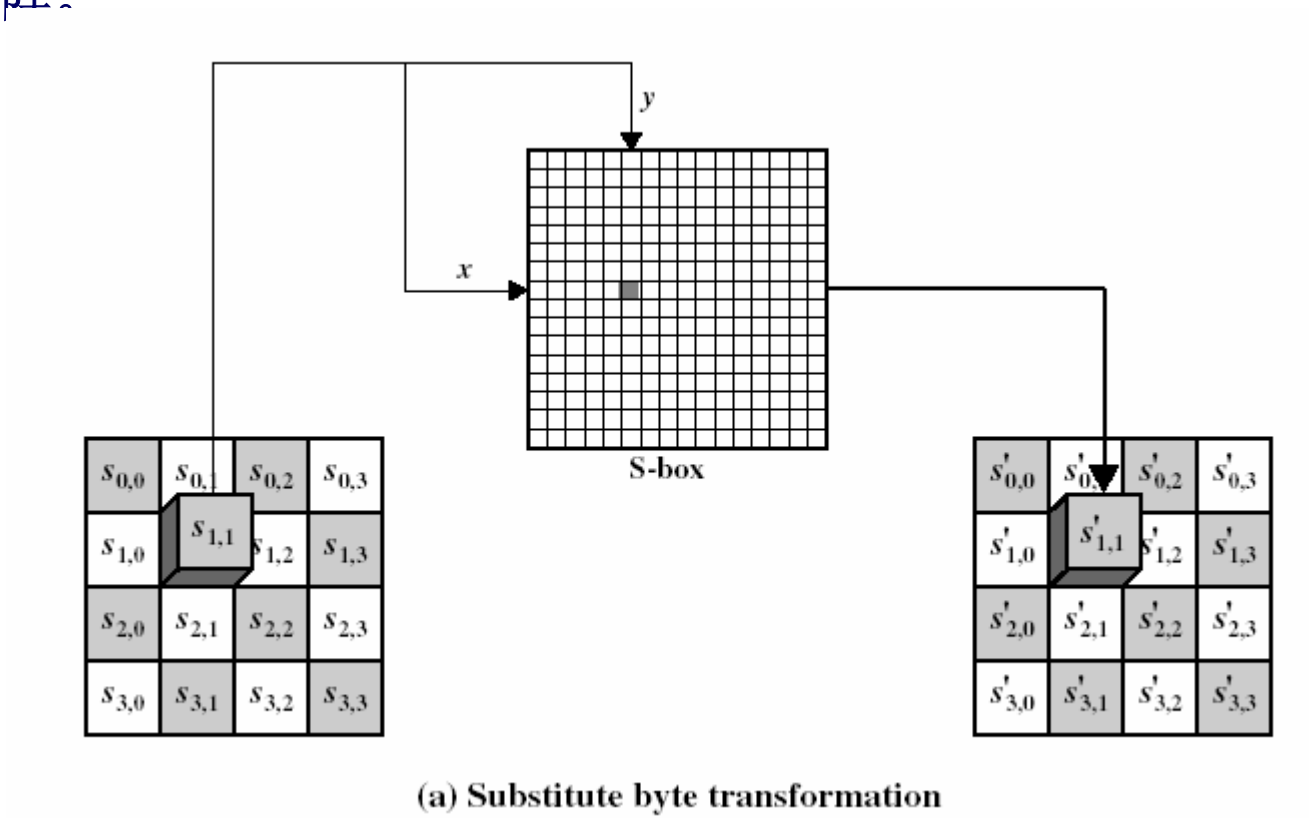
→ S →

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

整个算法仅用一个S—盒！简单，明了。

Substitute Bytes :

- 字节替换是一个非线性的字节代替，独立地在每个状态字节上进行运算。代替表（S-盒）是可逆的，是一个16×16的矩阵。



(a) Substitute byte transformation

Table 5.4 AES S-Boxes

(a) 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

example

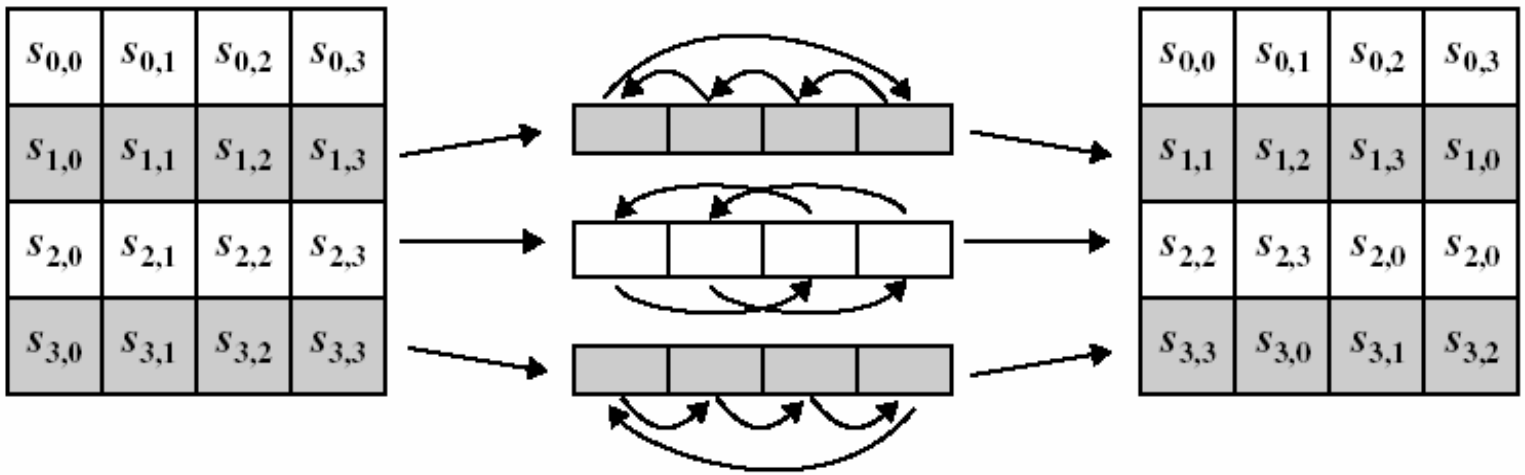
EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

行移位(Shift Row)变换

简单的置换



(a) Shift row transformation

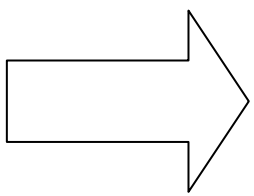
▲shift Row

对由字节组成的4行状态矩阵的每一行进行左循环移位，移位量(off sets)根据分组长度N_b和所在行数r而定：

N_b \ r	0	1	2	3
4	0	1	2	3
6	0	1	2	3
8	0	1	3	4

还是以N=4 (128 bit)为例：

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p



a	e	i	m
f	j	n	b
k	o	c	g
p	d	h	l

Offsets

0

1

2

3

example

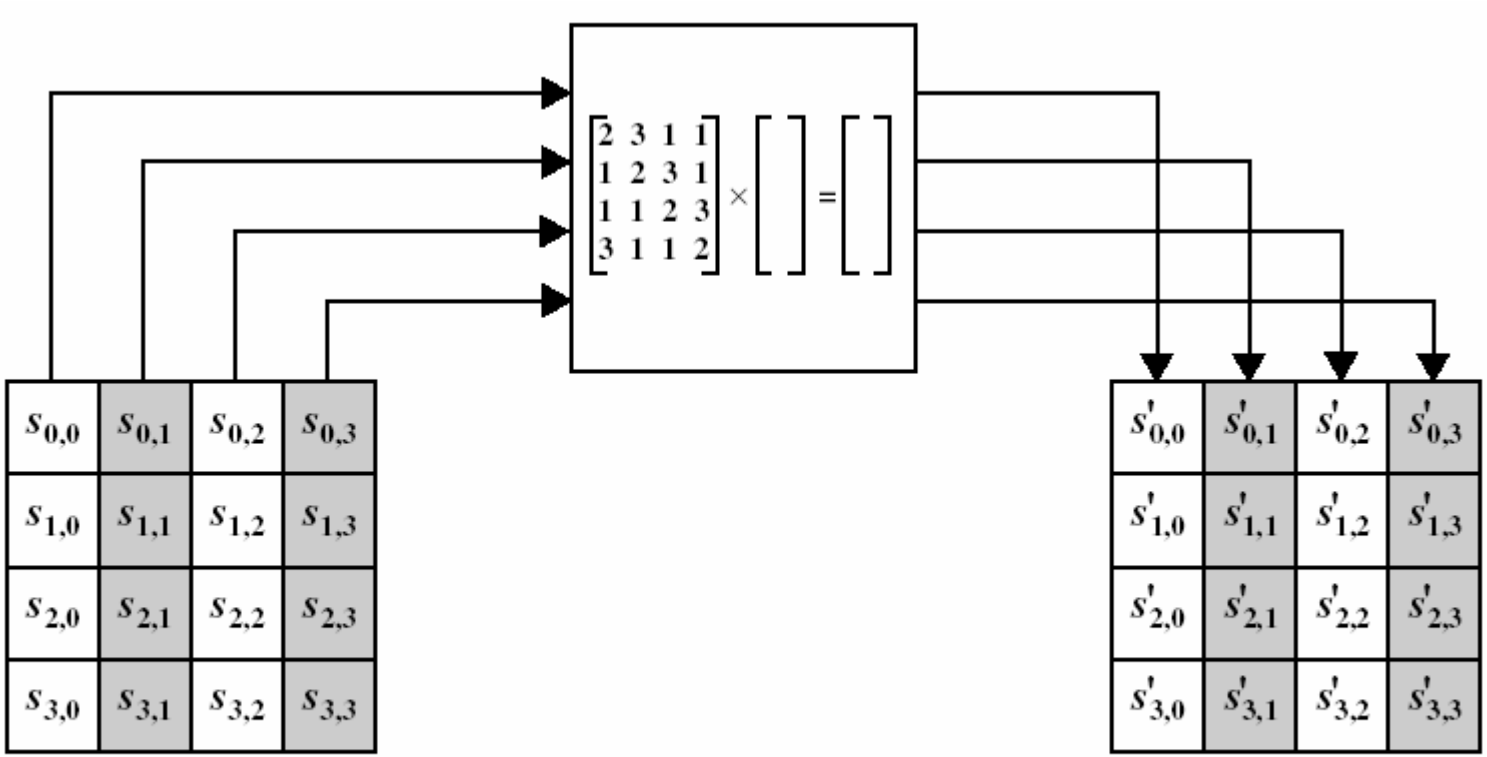
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

列混合Mix Column变换

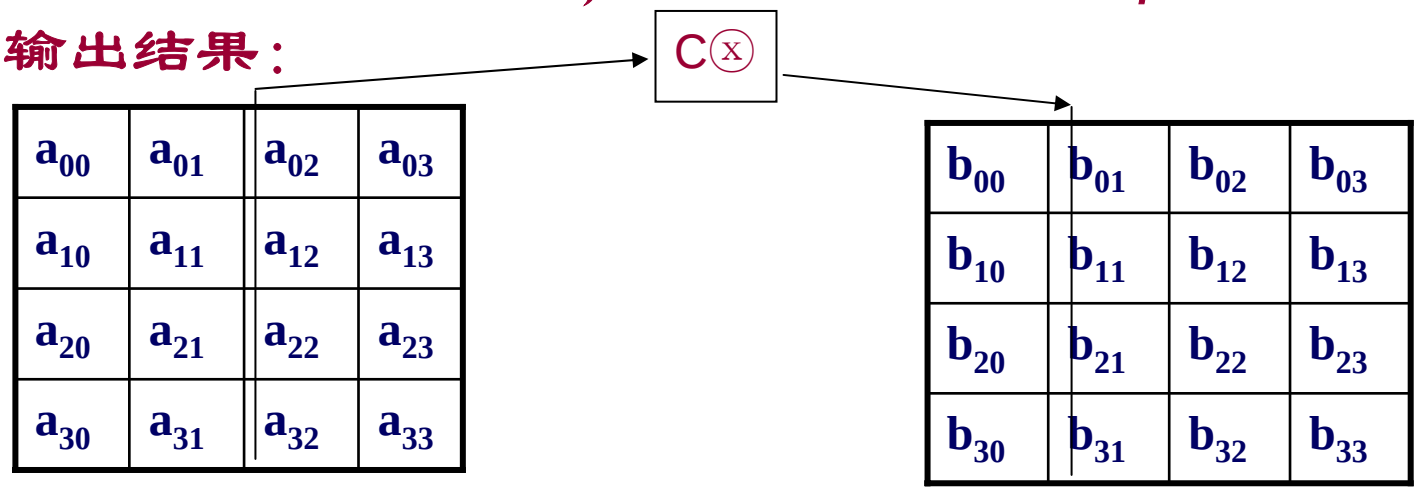
- 代替操作，将状态的列看作有限域GF（2^8）上的4维向量并被有限域GF（2^8）上的一个固定可逆方阵A乘



(b) Mix column transformation

▲Mix Column

状态矩阵中的每一列视为$GF(2^8)$上的一个4维列向量α，用一个固定的4 x 4矩阵C左乘，得到的4维列向量$\beta = C\alpha$作为变换的输出结果：



矩阵$C=(C_{ij})$中的系数取自$GF(2^8)$，

$$C = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

其中：

1= (0 0 0 0 0 0 0 1)，

2= (0 0 0 0 0 0 1 0)，

3= (0 0 0 0 0 0 1 1)。

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

$$\begin{aligned} s'_{0,j} &= (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j}) \\ s'_{3,j} &= (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j}) \end{aligned}$$

AES的密钥扩展和轮密钥选取

所有轮密钥比特的总数等于分组长度乘轮数加**1**。（如**128**比特的分组长度和**10**轮迭代，共需要**1408**比特的密钥）。

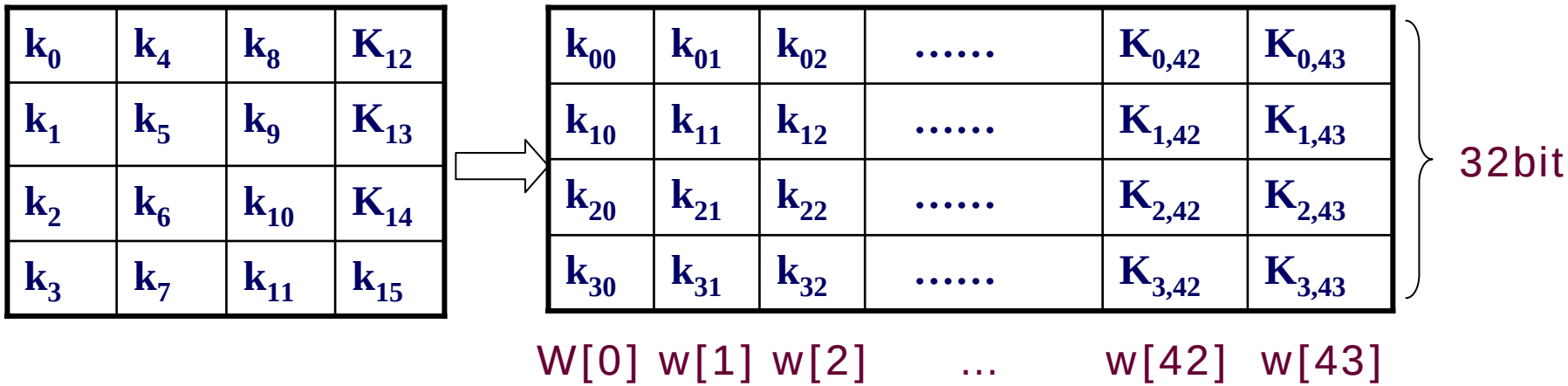
▲Key Expansion

▲子密钥发生器Key Expansion的输入是初始密钥CipherKey，长度为32•N_k比特，其中N_k=4，6或者8。其输出为各轮的子密钥Expanded Key。

▲子密钥只用在Add RoundKey中，长度等于明文分组，为32•N_b比特。进行N_r轮变换，需要N_r+1个子密钥，所以总共扩展的密钥长为32•N_b•(N_r+1)比特。

▲将子密钥的序列表示成4行的矩阵，其中每个元素都是一个字节(8bit)，则矩阵需要有N_b•(N_r+1)列。

▲若设N_b=4，N_r=10，且将第i列记作一个32bit的字(word) W[i]，我们有



这个图表示，密钥和它扩展出的子密钥w[i]。

(当$N_b=4$, $N_r=10$).

密钥扩展(Key Expansion)对于$N_k \leqslant 6$和$N_k > 6$的情形是不同的。

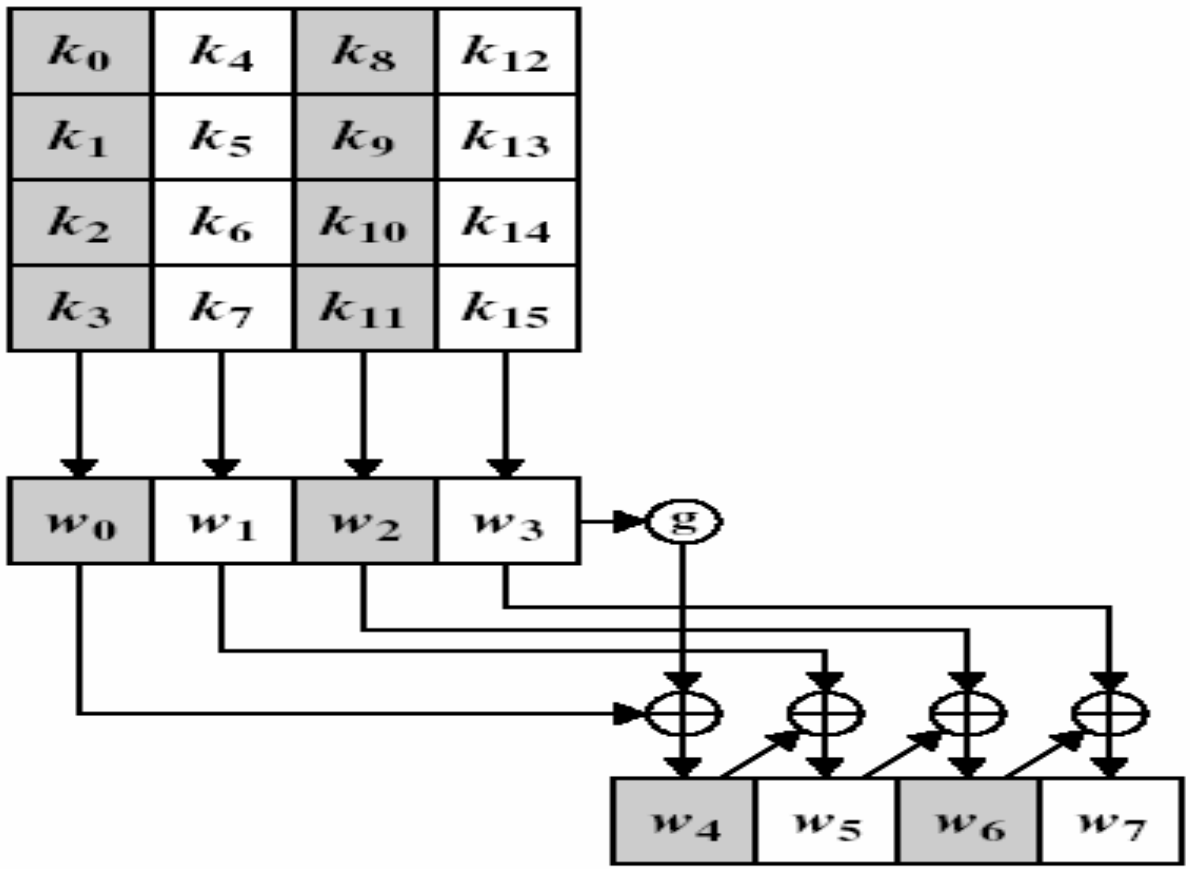


Figure 5.6 AES Key Expansion

- 轮密钥按下述方式从扩展密钥中选取：第一个轮密钥由开始**Nb**个字组成，第二个轮密钥由接下来的**Nb**个字组成，如此继续下去。

Rijndael安全性

- 没有发现弱密钥或补密钥
- 能有效抵抗目前已知的攻击算法
  - 线性攻击
  - 差分攻击

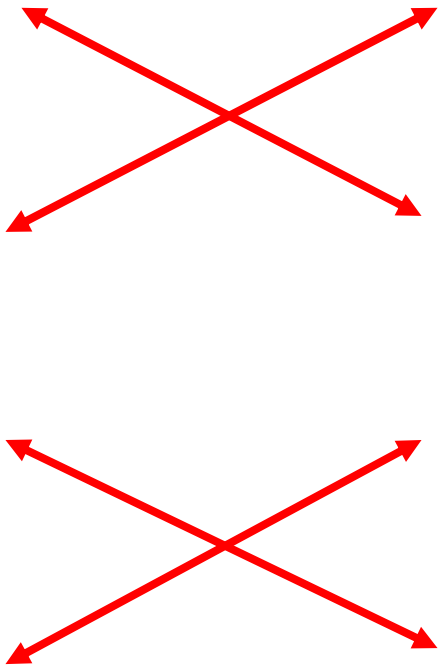
AES的解密

- AES的解密算法和加密算法不同
- 因此对于加密和解密而言，需要两个不同的软件或固件模块 —— 这是AES目前已知的最大缺点。
- 弥补措施：

可以构造一个解密算法的等价版本使其与加密算法有同样的结构

AES加密　　AES解密等价版本

1. 字节代换
2. 行移位
3. 列混淆
4. 轮密钥加



1. （逆向）行移位
2. （逆向）字节代换
3. 轮密钥加
4. （逆向）列混淆