通 信 安 全

L5—对称加密算法

- 教师：崔爱娇
- 编号：ELEC3019
- 学时：32学时

概 要

对称加密算法

- 加密算法
- DES加密算法
- AES加密算法
- 小结

概 要

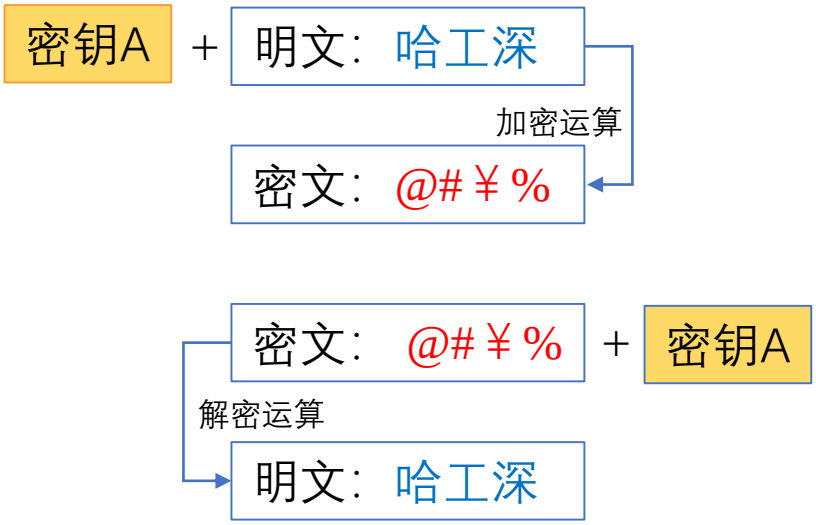
对称加密算法

- 加密算法
- DES加密算法
- AES加密算法
- 小结

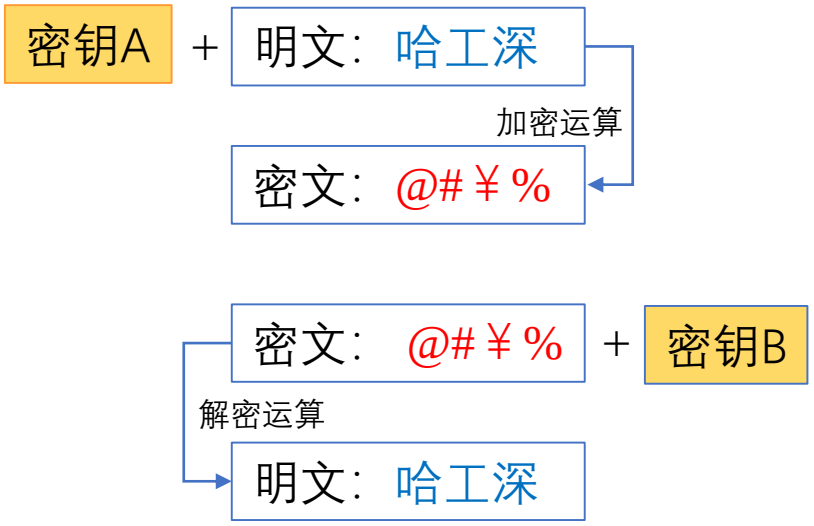
加密算法

加密：原来为明文的文件或数据按某种算法进行处理，使其成为不可读的一段代码为“密文”，使其只能在输入相应的密钥之后才能显示出原容，通过这样的途径来达到保护数据不被非法人窃取、阅读的目的。 [1]

对称加密算法



非对称加密算法



[1] https://baike.baidu.com/item/%E5%8A%A0%E5%AF%86%E7%AE%97%E6%B3%95/2816213?fr=aladdin.

对称加密

双方使用同样的密钥进行加密和解密，也称密钥加密。

- 加密方和解密方使用同一个密钥
- 加密解密的速度比较快，适合数据比较长时的使用；☺
- 密钥传输的过程不安全，且容易被破解，密钥管理也比较麻烦☹

DES（Data Encryption Standard）

数据加密标准，速度较快，适用于加密大量数据的场合；

是1972年美国IBM公司研制的对称密码体制加密算法

3DES（Triple DES）

是基于DES，对一块数据用三个不同的密钥进行三次加密，强度更高

AES（Advanced Encryption Standard）

高级加密标准，是下一代的加密算法标准，速度快，安全级别高，支持128、192、256、512位密钥的加密；

概 要

对称加密算法

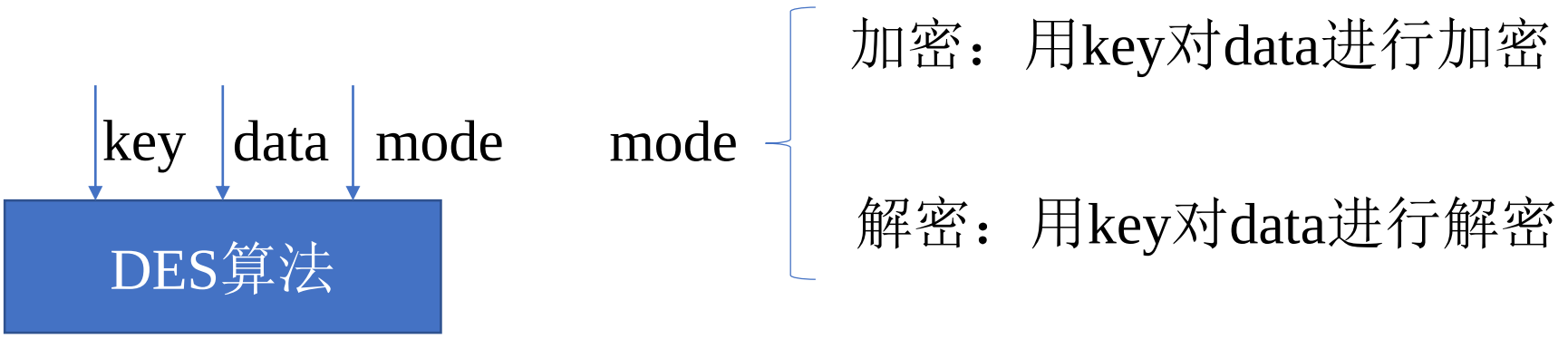
- 加密算法
- DES加密算法
- AES加密算法
- 小结

DES

DES（Data Encryption Standard）

DES算法为密码体制中的对称密码体制，又称美国数据加密标准，1972年美国IBM公司研制的对称密码体制加密算法。

明文按 64位进行分组，密钥长64位，密钥事实上是56位参与DES运算（第8、16、24、32、40、48、56、64位是校验位，使得每个密钥都有奇数个1）分组后的明文组和56位的密钥按位替代或交换的方法形成密文组的加密方法。



DES算法描述

1）输入64位明文数据，并进行初始置换IP；

2）在初始置换IP后，明文数据再被分为左右两部分，每部分32位，以L_0，R_0表示；

3）在密钥的控制下，经过16轮运算（f）；

4）16轮后，左、右两部分交换，并连接在一起，再进行逆置换；

5）输出64位密文。

分组比较短、密钥太短、密码生命周期短、运算速度较慢

计算16个子密钥

◆ 初始密钥中计算得出16个子密钥。

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

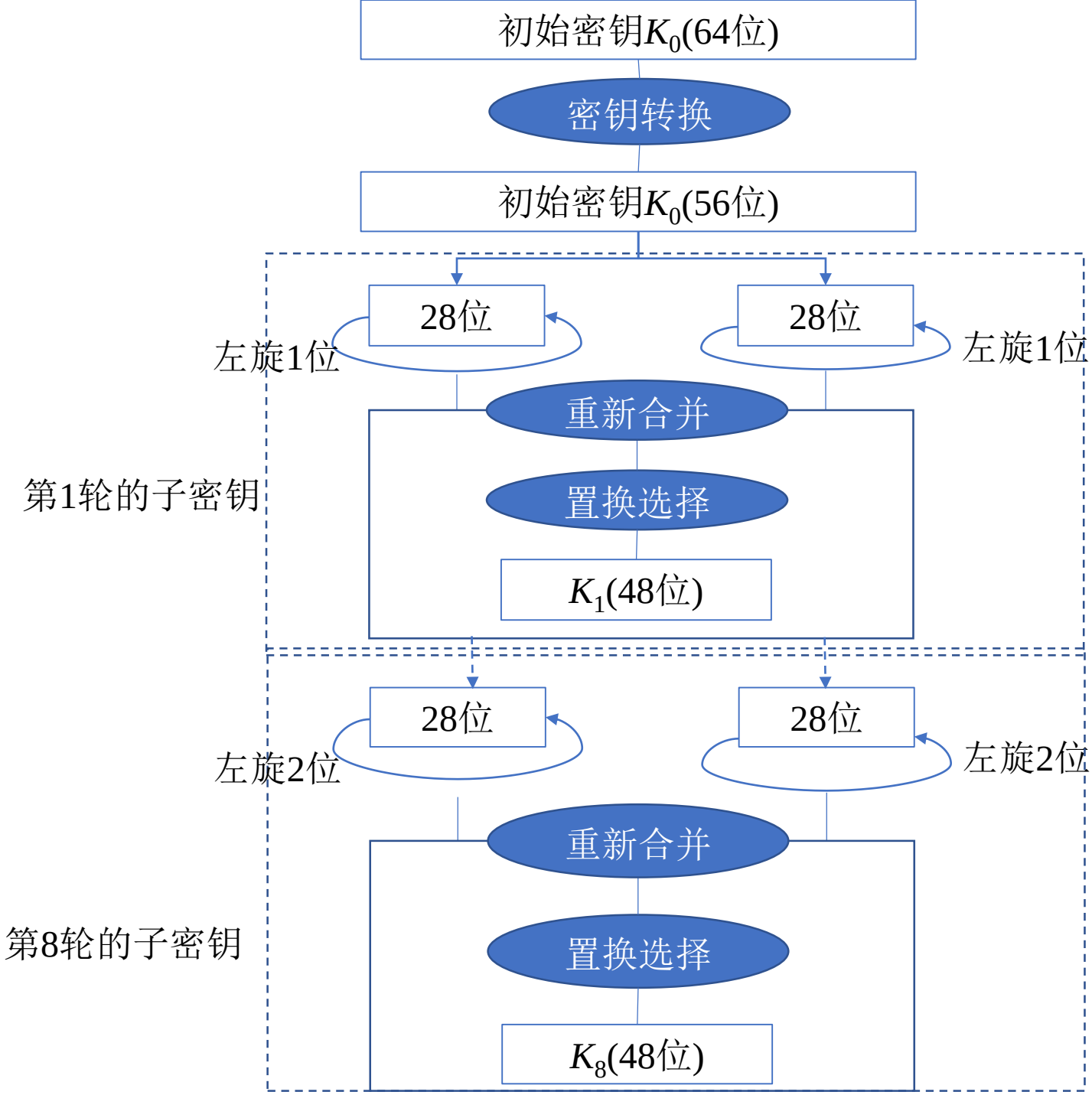
- DES使用一个56位的初始密钥，8位用于奇偶校验。
- 按照表1的方式执行密钥转换。初始密钥中的第57位就是转换后的密钥中的第1位，而初始密钥中的第49位则变成转换后的密钥中的第2位，以此类推。

轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
旋转	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

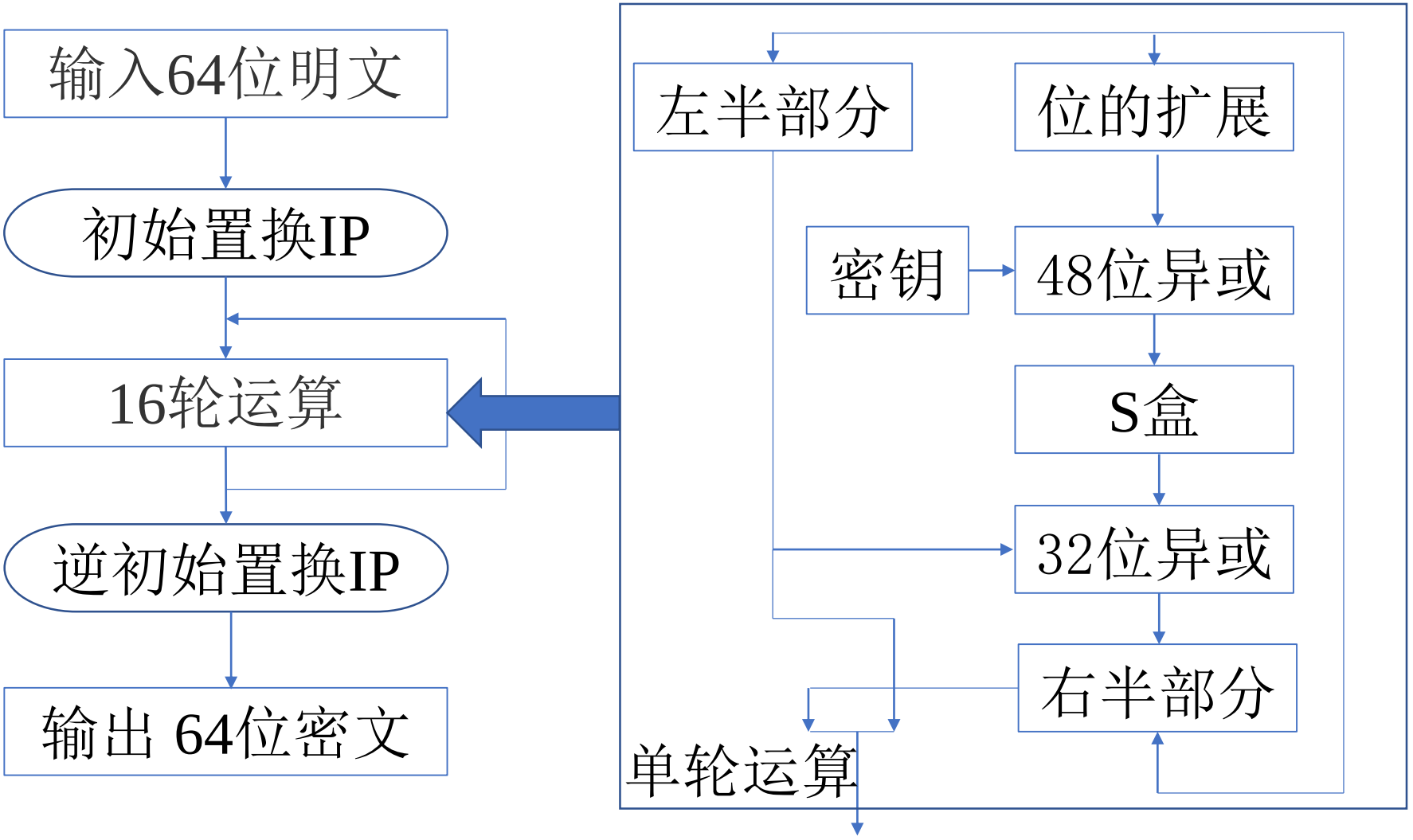
密钥每轮旋转次数

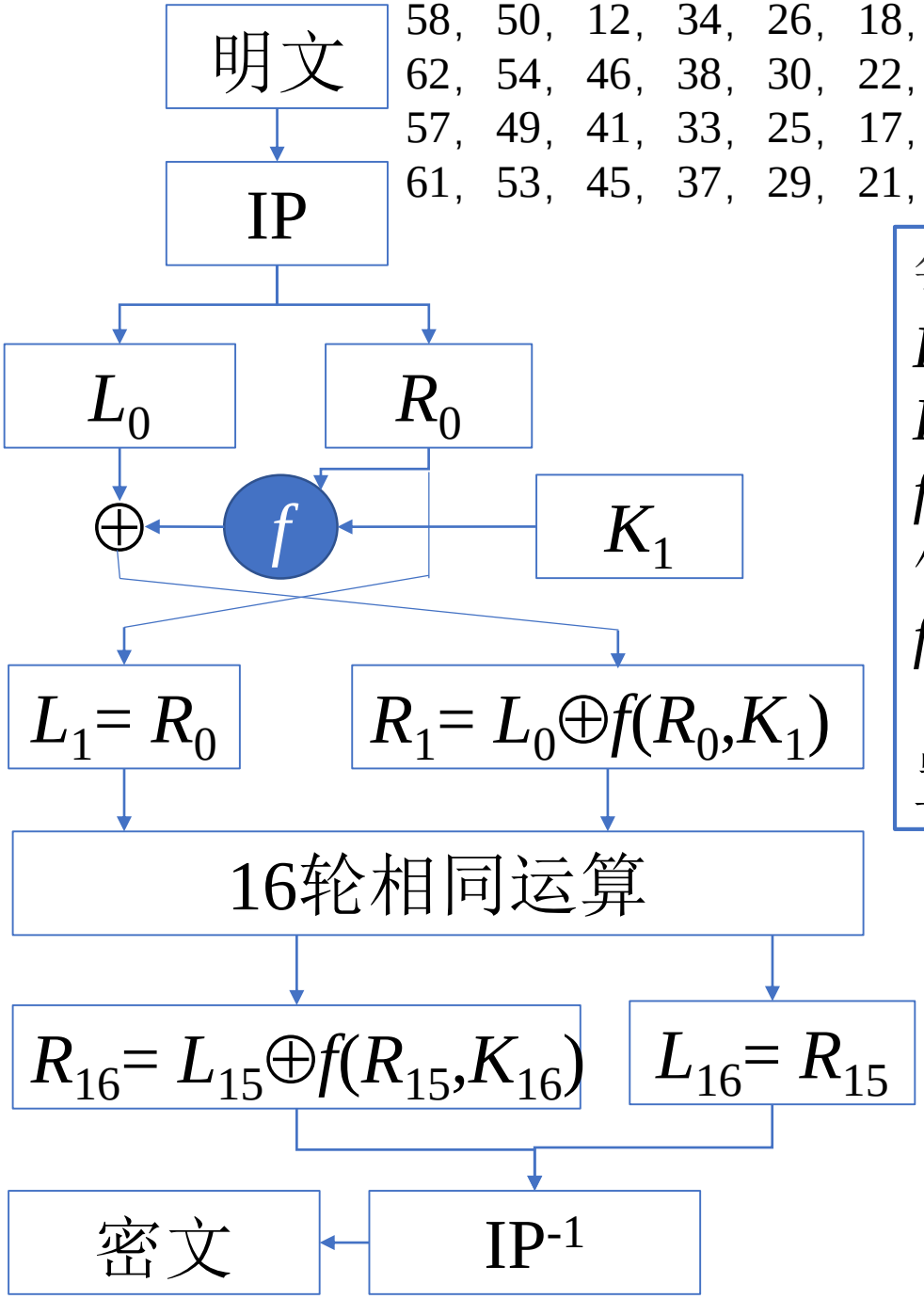
14,	17,	11,	24,	1,	5,	3,	28,	15,	6,	21,	10
23,	19,	12,	4,	26,	8,	16,	7,	27,	20,	13,	2,
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32,

密钥置换选择


初始密钥K_0(64位)
密钥转换
初始密钥K_0(56位)
28位
28位
左旋1位
左旋1位
重新合并
置换选择
K_1(48位)
第1轮的子密钥
28位
28位
左旋2位
左旋2位
重新合并
置换选择
K_8(48位)
第8轮的子密钥

DES算法结构


输入64位明文
初始置换IP
16轮运算
逆初始置换IP
输出 64位密文
左半部分
位的扩展
密钥
48位异或
S盒
32位异或
右半部分
单轮运算



58, 50, 12, 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,

每轮迭代的过程可以表示如下：

$L_n = R_{n-1}$，$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$

K_n：48位密钥

f：是以R_{n-1}，K_n为变量的输出32位的函数。

f由四步运算构成：密钥置换（K_n的生成，n=1-16）；扩展置换；S-盒代替；P-盒置换。

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,

位的扩展

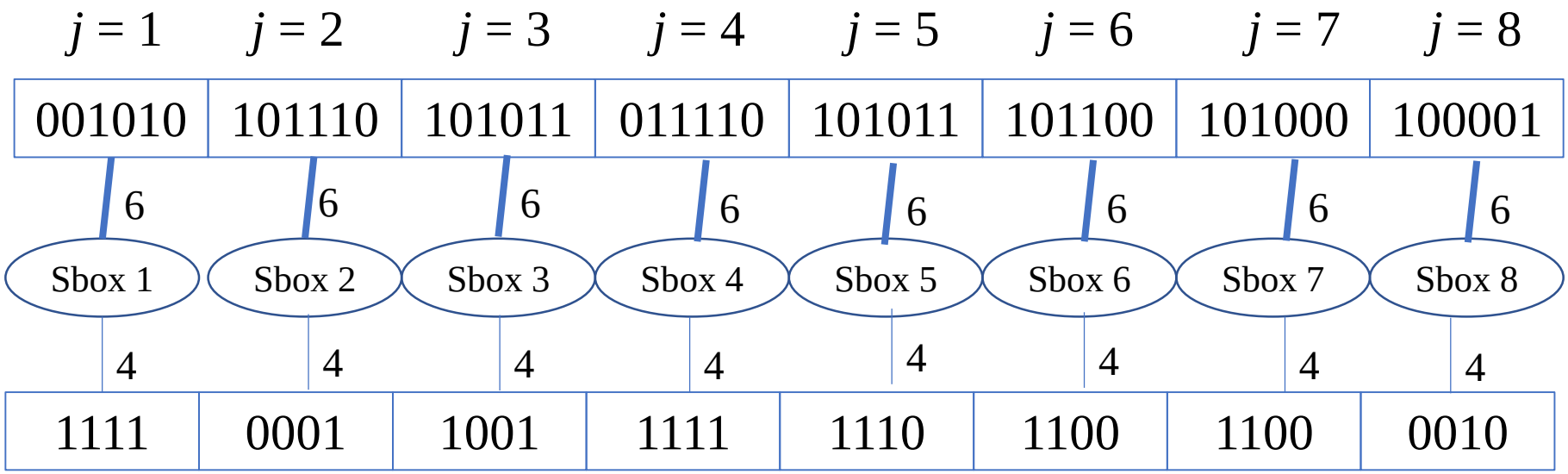
◆每一轮以L_{i-1}和R_{i-1}开始，然后根据下表所示进行扩展置换，**将R_{i-1}从32位扩展到48位**。该置换的主要目的是在加密数据的过程中制造一些**雪崩效应**，使用数据块中的1位将在下一步操作中影响更多位，从而产生**扩散效果**。

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,

◆一旦扩展置换完成，计算出48位的结果值与这一轮子密钥K_i的异或值（XOR，符号计为⊕）。这将产生48位的中间值，记为R_{int}。如果将E计为扩展置换的结果，则本轮到目前为止的操作可以表示为：

$$R_{\text{int}} = \text{E}(R_{i-1}) \oplus K_i$$

S盒替换操作



◆**R_{int} 需要通过8个单独的S盒执行8次替换操作**。每个S盒（j）从R_{int}的6（j-1） 到 6j -1的位置取出6位，并为其在表6中查出1个4位的值，将该值写到缓冲区的4j位置处

S盒替换查找表

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1	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	3	14	10	0	6	13
S2	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	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
S3	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	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
S4	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6	0	12	1	10	15	9	2	6	8	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
S7	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

- 根据第1位和最后1位组成的2位值找到表中的行号
- 而根据中间剩下的4位来确定表中的列号
- 比如，R_{int}中的第3个6位组是101011。因此，在表中查找到的第3个S盒是9。因为行号等于3，列号等于$0101_2 = 5$（查表时从索引0开始计数）。
- S盒为数据增加了不确定性，除了给DES带来安全性外，没什么特别的。

数据的P盒置换

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

- 如果 b_j 代表R_{int}中的第j个6位组，S_j 代表第j个S盒，而P代表P盒置换，则该函数可以定义为：

$$f = P(S_1(b_1), S_2(b_2), ..., S_8(b_8))$$

- 每一轮的最后一个操作是计算 f 的32位结果值与传入本轮操作的原始数据的左分组L_{i-1}之间的异或值。

概 要

对称加密算法

- 加密算法
- DES加密算法
- AES加密算法
- 小结

AES

DES的密钥长度是56比特，因此算法的理论安全强度是2^{56}

DES不能提供足够的安全性

元器件制造工艺的进步使得计算机的处理能力越来越强

1997年1月2号，NIST宣布希望征集高级加密标准（Advanced Encryption Standard: AES）

Rijindael， Serpent，Twofish， RC6 和MARS。最终经过安全性分析、软硬件性能评估等严格的步骤，Rijndael算法获胜。

AES的基本结构

分组密码：明文分组，每组长度相等，每次加密一组数据，直到加密完整个明文。分组长度128位。密钥长度：128位，192位或256位。

AES	密钥长度（32比特字）	分组长度（32比特字）	加密轮数
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

- 一个明文分组会被加密10轮，前9次一样，第10次有所不同。
- 处理单位是字节，明文和密钥均被分成16个字节；
- $P = P_0 P_1 \ldots P_{15}$，$K = K_0 K_1 \ldots K_{15}$

状态矩阵

明文矩阵（4×4）

P_0	P_4	P_8	P_{12}
P_1	P_5	P_9	P_{13}
P_2	P_6	P_{10}	P_{14}
P_3	P_7	P_{11}	P_{15}

状态矩阵1

S_0	S_4	S_8	S_{12}
S_1	S_5	S_9	S_{13}
S_2	S_6	S_{10}	S_{14}
S_3	S_7	S_{11}	S_{15}

AES的10轮加密

状态矩阵N

S_0'	S_4'	S_8'	S_{12}'
S_1'	S_5'	S_9'	S_{13}'
S_2'	S_6'	S_{10}'	S_{14}'
S_3'	S_7'	S_{11}'	S_{15}'

密文矩阵N （4×4）

C_0	C_4	C_8	C_{12}
C_1	C_5	C_9	C_{13}
C_2	C_6	C_{10}	C_{14}
C_3	C_7	C_{11}	C_{15}

状态矩阵

明文矩阵

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

状态矩阵1

*0x*61	*0x*65	*0x*69	*0x*6D
*0x*62	*0x*66	*0x*6A	*0x*6E
*0x*63	*0x*67	*0x*6B	*0x*6F
*0x*64	*0x*68	*0x*6C	*0x*70

AES的10轮加密

状态矩阵*N*

*0x*A9	*0x*99	*0x*AE	*0x*77
*0x*13	*0x*A7	*0x*C1	*0x*57
*0x*29	*0x*8D	*0x*7C	*0x*AA
*0x*AF	*0x*02	*0x*50	*0x*EF

密文矩阵*N*

C_0	C_4	C_8	C_{12}
C_1	C_5	C_9	C_{13}
C_2	C_6	C_{10}	C_{14}
C_3	C_7	C_{11}	C_{15}

密钥

密钥矩阵（4×4）

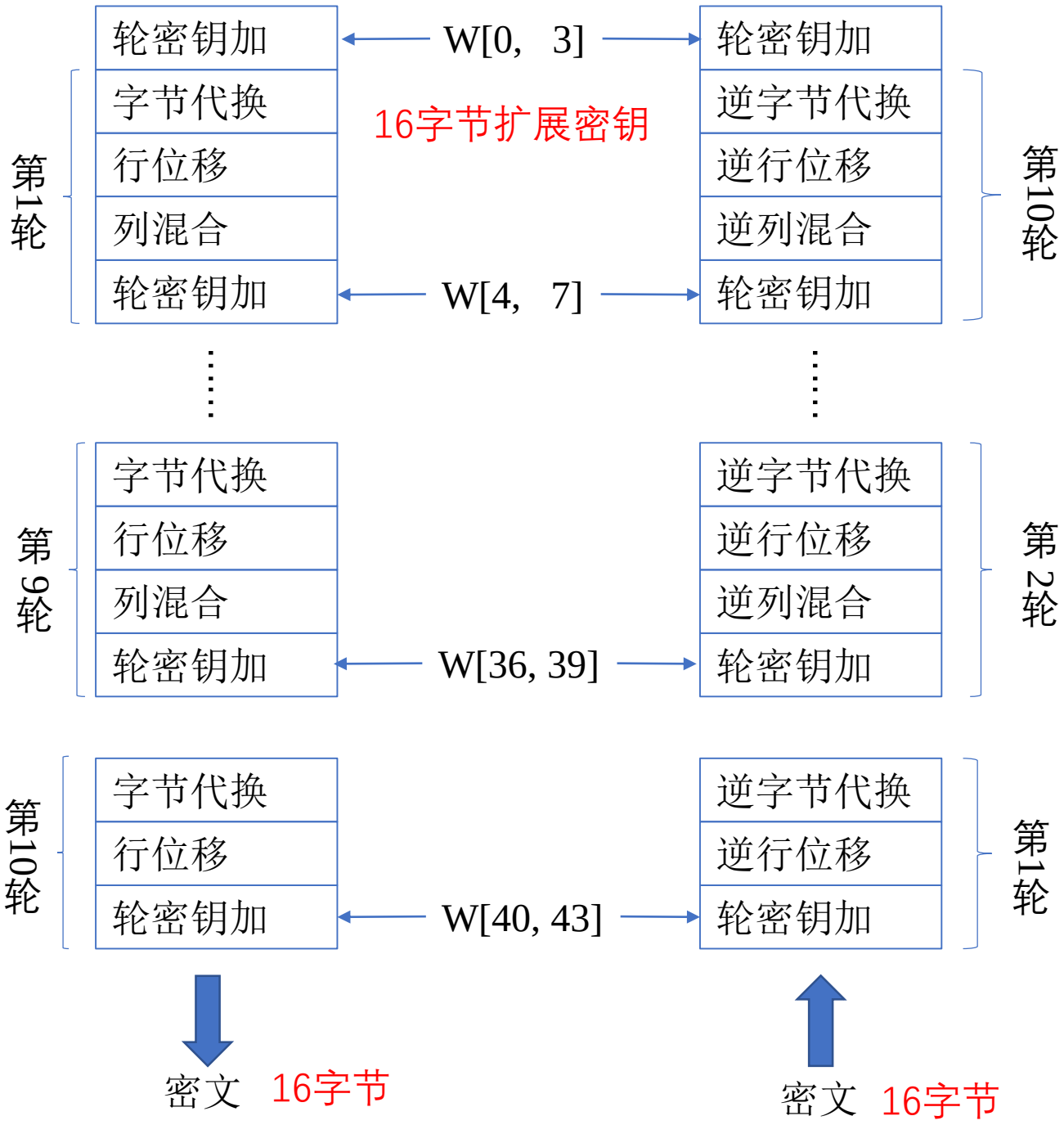
K_0	K_4	K_8	K_{12}
K_1	K_5	K_9	K_{13}
K_2	K_6	K_{10}	K_{14}
K_3	K_7	K_{11}	K_{15}

密钥编排函数

W[i]：32位比特字；

40个字，分10组，每组4个字128比特，
用于10轮加密运算中的轮密钥加；

1-9轮的轮函数：字节代换，行位移，列混合，轮密钥加
第10轮轮函数：字节代换，行位移，轮密钥加


轮密钥加
字节代换
行位移
列混合
轮密钥加
W[0, 3]
W[4, 7]
16字节扩展密钥
第1轮
轮密钥加
逆字节代换
逆行位移
逆列混合
轮密钥加
第10轮
字节代换
行位移
列混合
轮密钥加
W[36, 39]
第 9轮
逆字节代换
逆行位移
逆列混合
轮密钥加
第 2轮
字节代换
行位移
轮密钥加
W[40, 43]
第10轮
逆字节代换
逆行位移
轮密钥加
第1轮
密文 16字节
密文 16字节

字节替代

字节代替的主要功能是通过S盒完成一个字节到另外一个字节的映射，S盒用于提供密码算法的**混淆性 [1]**

		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[D][D]=??

[1] Joan Daemen and Vincent Rijmen, The Design of Rijndael, AES - The Advanced Encryption Standard, Springer-Verlag 2002 (238 pp.).

字节替代

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

$S^{-1}[C][1]=??$

S和S^{-1}分别为16x16的矩阵，完成一个8比特输入到8比特输出的映射，输入的高4-bit对应的值作为行标，低4-bit对应的值作为列标。

行移位

行移位是一个4x4的矩阵内部字节之间的置换，用于提供算法的**扩散性**。

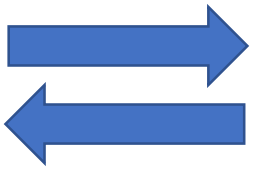
正向行移位：正向行移位用于加密

- 第一行保持不变
- 第二行循环左移8比特
- 第三行循环左移16比特
- 第四行循环左移24比特。

逆向行移位：**逆**向行移位用于解密

- 第一行保持不变
- 第二行循环右移8比特
- 第三行循环右移16比特
- 第四行循环右移24比特。

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

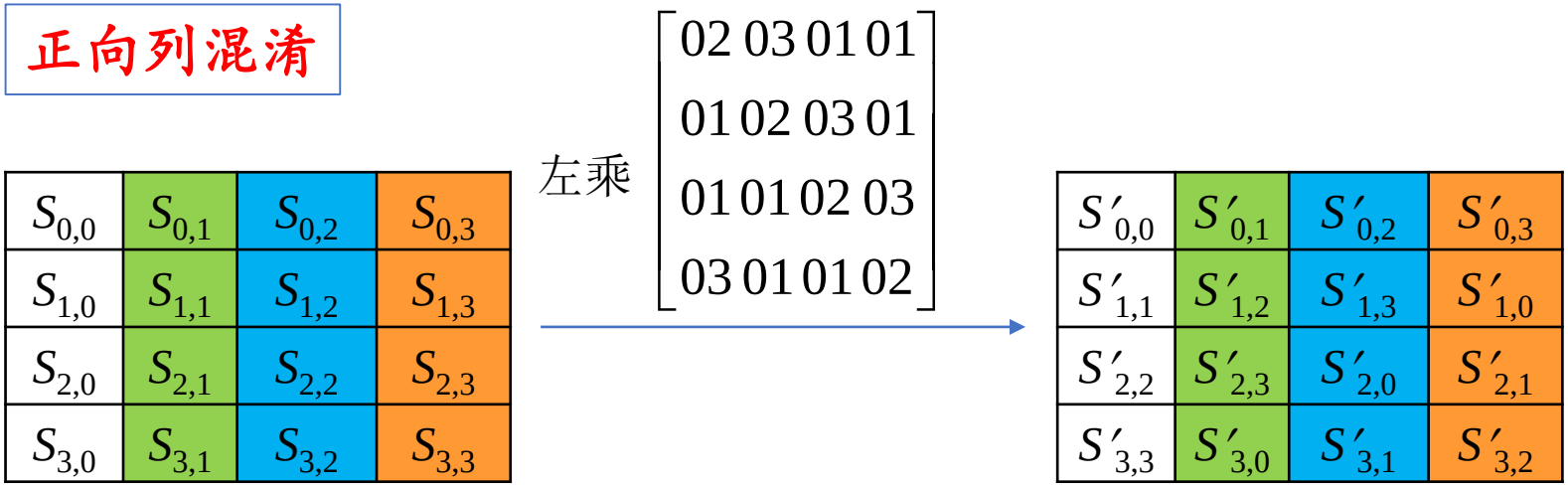


$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

列混淆

列混淆：利用GF(2^8)域上算术特性的一个代替，用于提供算法的**扩散性**。

正向列混淆



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：$07 \cdot S_{0,0} = (01 \oplus 02 \oplus 04) \cdot S_{0,0} = S_{0,0} \oplus (02 \cdot S_{0,0}) \oplus (04 \cdot S_{0,0})$

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）。

列混淆

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{matrix} C9 \\ 6E \\ 46 \\ A6 \end{matrix} \rightarrow \begin{matrix} DB \\ 37 \\ 94 \\ ED \end{matrix}$$

$$S'_{0,0} = (02 \bullet C9) \oplus (03 \bullet 6E) \oplus (01 \bullet 46) \oplus (01 \bullet A6)$$

$$02 \bullet C9 = 02 \bullet 11001001_B = 10010010_B \oplus 00011011_B = 10001001_B$$

$$03 \bullet 6E = (01 \oplus 02) \bullet 6E = 01101110_B \oplus 11011100_B = 10110010_B$$

$$01 \bullet 46 = 01000110_B$$

$$01 \bullet A6 = 10100110_B$$

$$S'_{0,0} = 10001001_B \oplus 10110010_B \oplus 01000110_B \oplus 10100110_B$$
$$= 11011011_B = DB$$

逆向列混淆

逆向列混淆

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$	$S'_{1,0}$
$S'_{2,2}$	$S'_{2,3}$	$S'_{2,0}$	$S'_{2,1}$
$S'_{3,3}$	$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$

左乘 $\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$

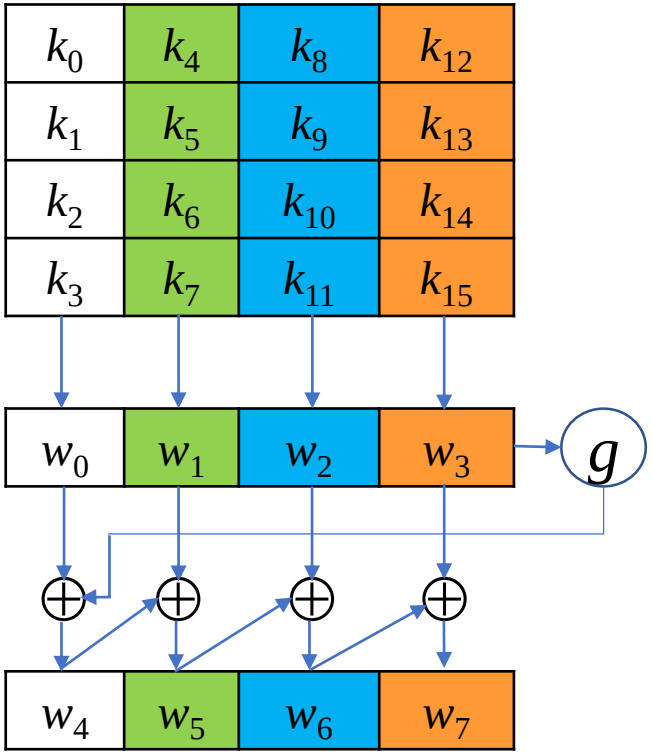
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

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

两个矩阵互逆，经过一次逆向列混淆，即可恢复原文。

轮密钥加

原理：“任何数和自身的异或结果为0”。加密过程中，每轮的输入与轮子密钥异或一次；因此，解密时再异或上该轮的轮子密钥即可恢复



（a）总体算法

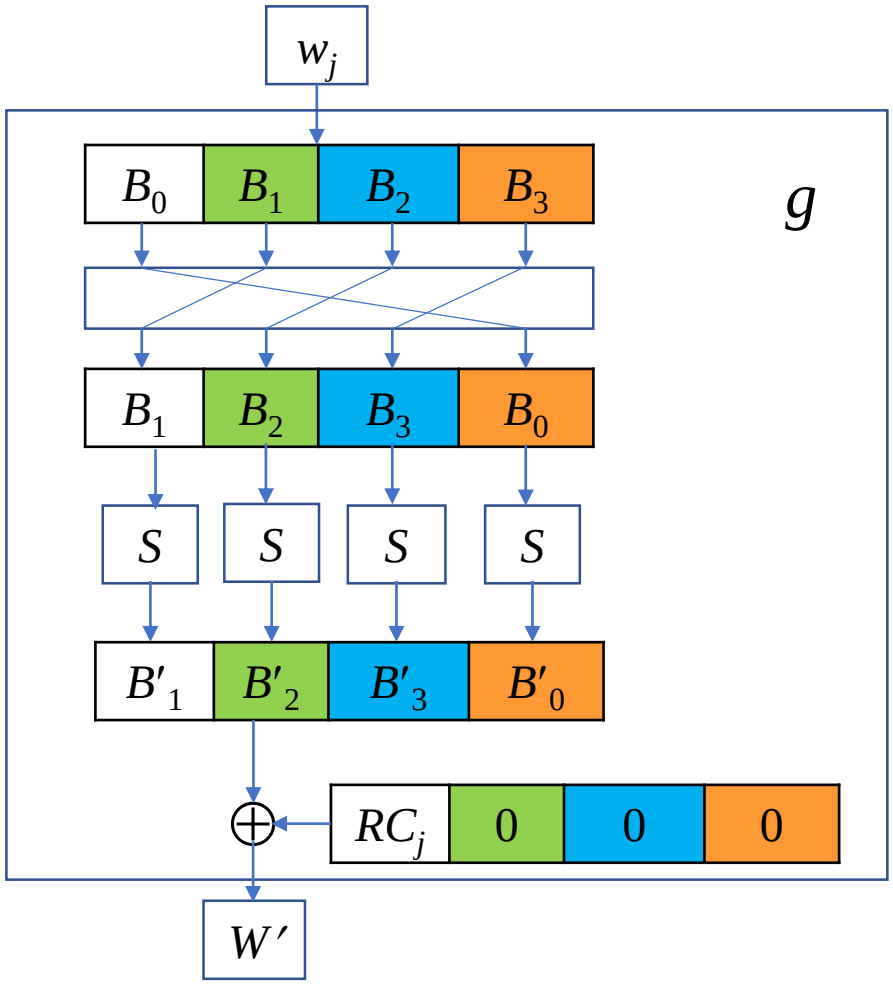
密钥扩展过程：

1) 将种子密钥按图(a)的格式排列，其中k_0、k_1、……、k_{15}依次表示种子密钥的一个字节；排列后用4个32比特的字表示，分别记为$w[0]$、$w[1]$、$w[2]$、$w[3]$；

2) 按照如下方式，依次求解w_j，其中j是整数并且属于[4,43]；

3) 若$j\%4=0$,则$w_j=w_{j-4}\oplus g(w_{j-1})$,否则$w_j=w_{j-4}\oplus w_{j-1}$；

密钥扩展算法



函数g

函数g的流程说明：

a) 将w_j循环左移8比特；

b) 分别对每个字节做S盒置换；

c) 与32比特的常量（$RC_{j/4}$,0,0,0）进行异或，RC是一个一维数组，其值如下。（RC的值只需要有10个，而此处用了11个，实际上RC_0在运算中没有用到，增加RC_0是为了便于程序中用数组表示。由于j的最小取值是4，j/4的最小取值则是1，因此不会产生错误。）

RC = {0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36}

子密钥计算举例

① 设初始的128位密钥为：
3C A1 0B 21 57 F0 19 16 90 2E 13 80 AC C1 07 BD

② 那么4个初始值为：
w_0 = 3C A1 0B 21 , w_1 = 57 F0 19 16 , w_2 = 90 2E 13 80 , w_3 = AC C1 07 BD

③ 下面求扩展的第1轮的子密钥(w_4, w_5, w_6, w_7)。
由于4是4的倍数，所以：
$w_4 = w_0 \oplus g(w_3)$

④ g(w_3)的计算步骤如下：
1. 循环地将w_3的元素移位：AC C1 07 BD变成C1 07 BD AC;
2. 将 C1 07 BD AC 作为S盒的输入，输出为78 C5 7A 91;
3. 将78 C5 7A 91与第一轮轮常量RC_1进行异或运算，将得到79 C5 7A 91，因此， $g(w_3)$ =79 C5 7A 91，故
w_4 = 3C A1 0B 21 ⊕ 79 C5 7A 91 = 45 64 71 B0

⑤ 其余的3个子密钥段的计算如下：
$w_5 = w_1 \oplus w_4$ = 57 F0 19 16 ⊕ 45 64 71 B0 = 12 94 68 A6
$w_6 = w_2 \oplus w_5$ =90 2E 13 80 ⊕ 12 94 68 A6 = 82 BA 7B 26
$w_7 = w_3 \oplus w_6$ = AC C1 07 BD ⊕ 82 BA 7B 26 = 2E 7B 7C 9B

第一轮的密钥为 45 64 71 B0 12 94 68 A6 82 BA 7B 26 2E 7B 7C 9B

软硬件实现

◆ 查找表提高效率

AES中列混淆部分设计有限域乘法操作，在运行中需要消耗较多的时间。现在的计算平台都拥有丰富的软件资源(RAM、Flash等)，因此AES的软件实现一般都会采用**查表**的方式，将字节替代、行移位、列混淆合在一起查表，大概消耗8-10K字节的存储空间，但效率非常之高。

◆ 复用减少开销

在硬件上，为了减少实现面积，可以通过对解密的操作进行适当调换，这样解密操作可以复用加密的电路

实际中，一般是通过RSA加密AES的密钥，传输到接收方，接收方解密得到AES密钥，然后发送方和接收方用AES密钥来通信。

概 要

对称加密算法

- 加密算法
- DES加密算法
- AES加密算法
- 小结

DES与AES的比较

一、数据加密标准不同

1、DES算法的入口参数有三个：Key、Data、Mode。其中Key为7个字节共56位，是DES算法的工作密钥；Data为8个字节64位，是要被加密或被解密的数据；Mode为DES的工作方式,有两种:加密或解密。

2、AES的基本要求是，采用对称分组密码体制，密钥的长度最少支持为128、192、256，分组长度128位，算法应易于各种硬件和软件实现。因此AES的密钥长度比DES大，它也可设定为32比特的任意倍数，最小值为128比特，最大值为256比特，所以用穷举法是不可能破解的。

DES与AES的比较

二、数据加密标准不同

1、二、运行速度不同1、作为分组密码，DES 的加密单位仅有64 位二进制，这对于数据传输来说太小，因为每个分组仅含8 个字符，而且其中某些位还要用于奇偶校验或其他通讯开销。处理速度慢、加密耗时

2、AES对内存的需求非常低，运算速度快，在有反馈模式、无反馈模式的软硬件中，Rijndael都表现出非常好的性能。

三、适用范围不同

1、数据加密标准，速度较快，适用于加密大量数据的场合。DES在安全上是脆弱的，但由于快速DES芯片的大量生产，使得DES仍能暂时继续使用，为提高安全强度，通常使用独立密钥的三级DES

2、AES 适用于8位的小型单片机或者普通的32位微处理器,并且适合用专门的硬件实现，硬件实现能够使其吞吐量（每秒可以到达的加密/解密bit数）达到十亿量级。同样，其也适用于RFID系统。

。

AES的由来

AES是采用公开竞标的，NIST的要求是：
1.候选算法使用128bit的区块；
2. 候选算法必须使用最小128bit长度的密钥，并且可以向上扩大，且可以使用128、192、256bit密钥；
3.候选算法要足够快，最好和DES算法差不多快；
4.候选算法必须免费，没有知识产权约束。可以提名专利算法，但是如果被选，必须成为免费使用许可。

- 最初有约15种算法被提交作为AES的候选算法，经过长时间筛选，NIST测试的算法减少到5个，分别为Rijndael、Serpent、 Twofish、MARS 和RC6。

- 再次筛选后还剩Rijndael、Serpent、Twofish，这三种算法各有长短：Serpent被认为最安全，而Rijndael速度最快，Twofish则居中。

- 最后NIST选择了Rijndael作为AES的算法。

作业

AES算法的程序实现：
1.实现AES加解密算法；
2.可演示；