对称加密技术实验

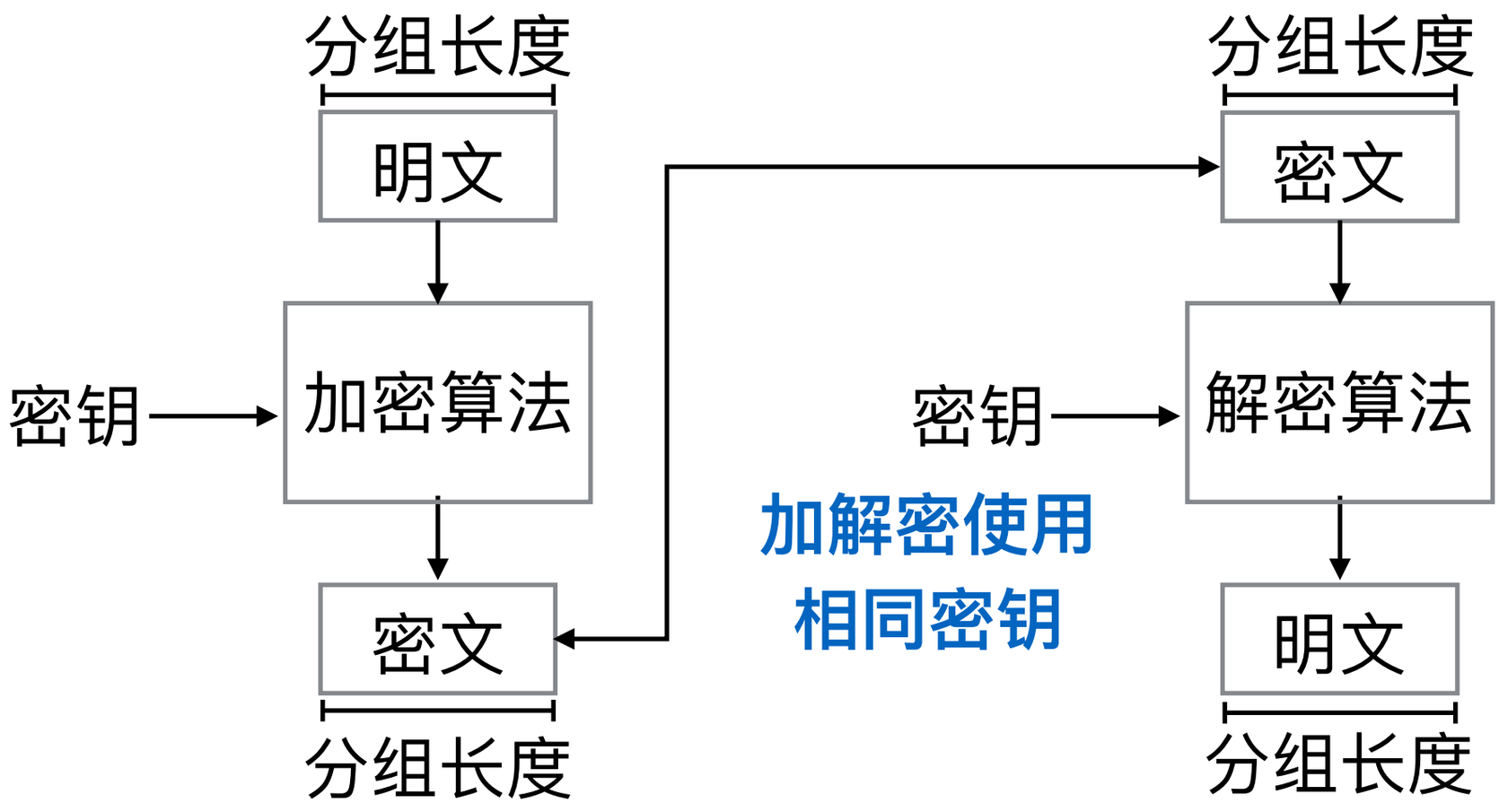
流密码: RC4

分组密码: DES

哈希函数: SHA-1

分组密码

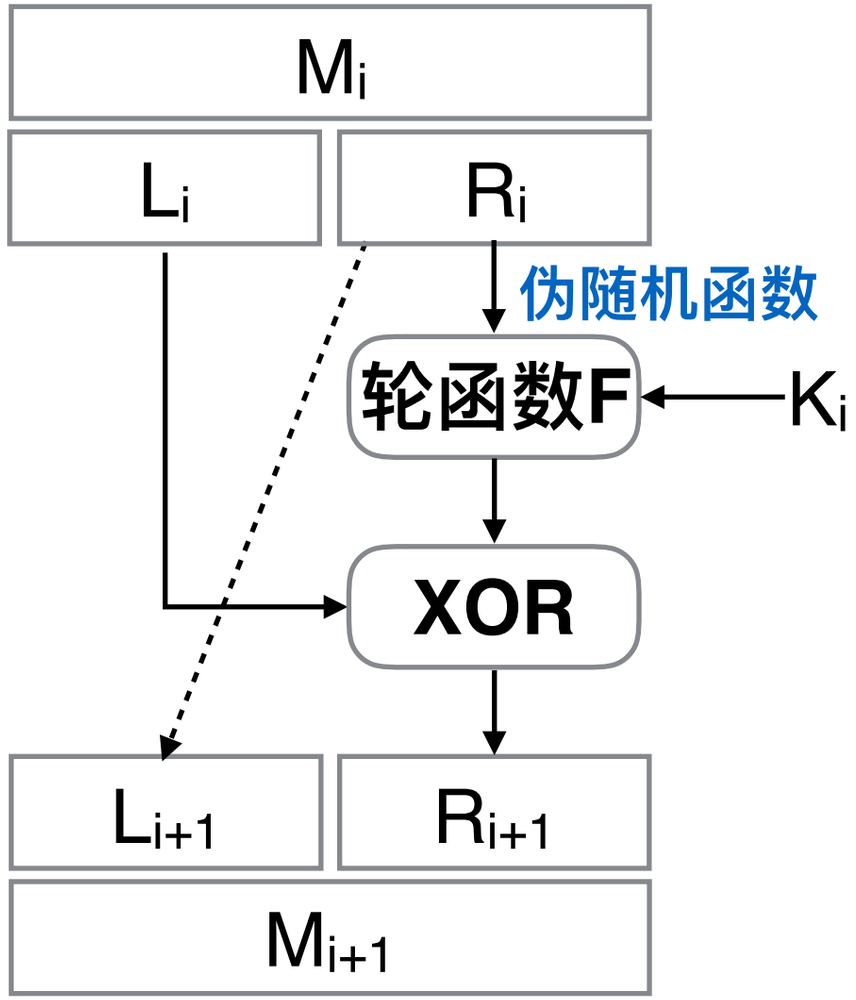
- 分组密码: 一种对称加密算法, 将明文进行**分组**, 将**每个明文分组作为整体进行加密**



Feistel密码结构

- Feistel结构: 一种典型的分组密码结构
  - ➡ **扩散**: 明文每一位影响密文许多位
  - ➡ **混淆**: 隐藏密文与密钥统计关系

基于迭代实现混淆和扩散

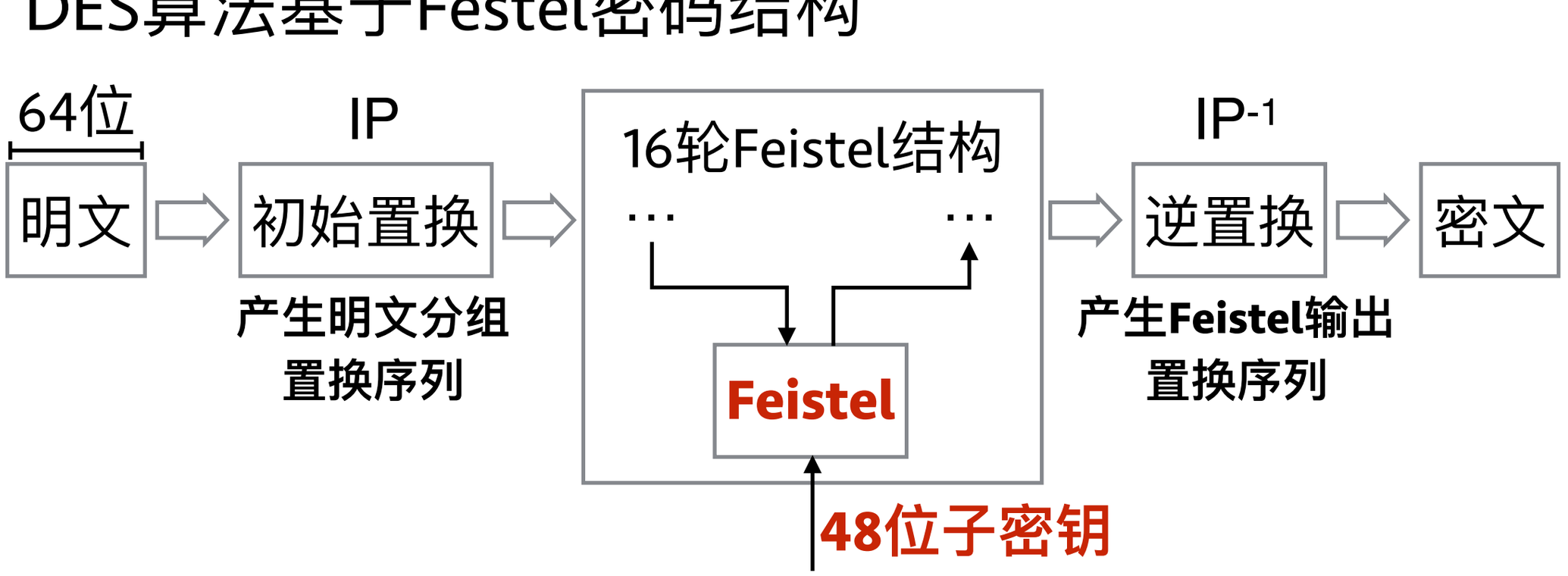


- ➡ 经过**两轮**Feistel运算, 可以将明文/密钥中**一个比特位**改变, 扩散/混淆到密文中**至少一半比特位**
- ➡ Feistel密码结构具有**加解密过程的相似性**

数据加密标准DES

- DES参数: 分组长度**64**位; 有效密钥长度**56**位 (通过奇偶校验位扩展为64位)

- DES算法基于Festel密码结构

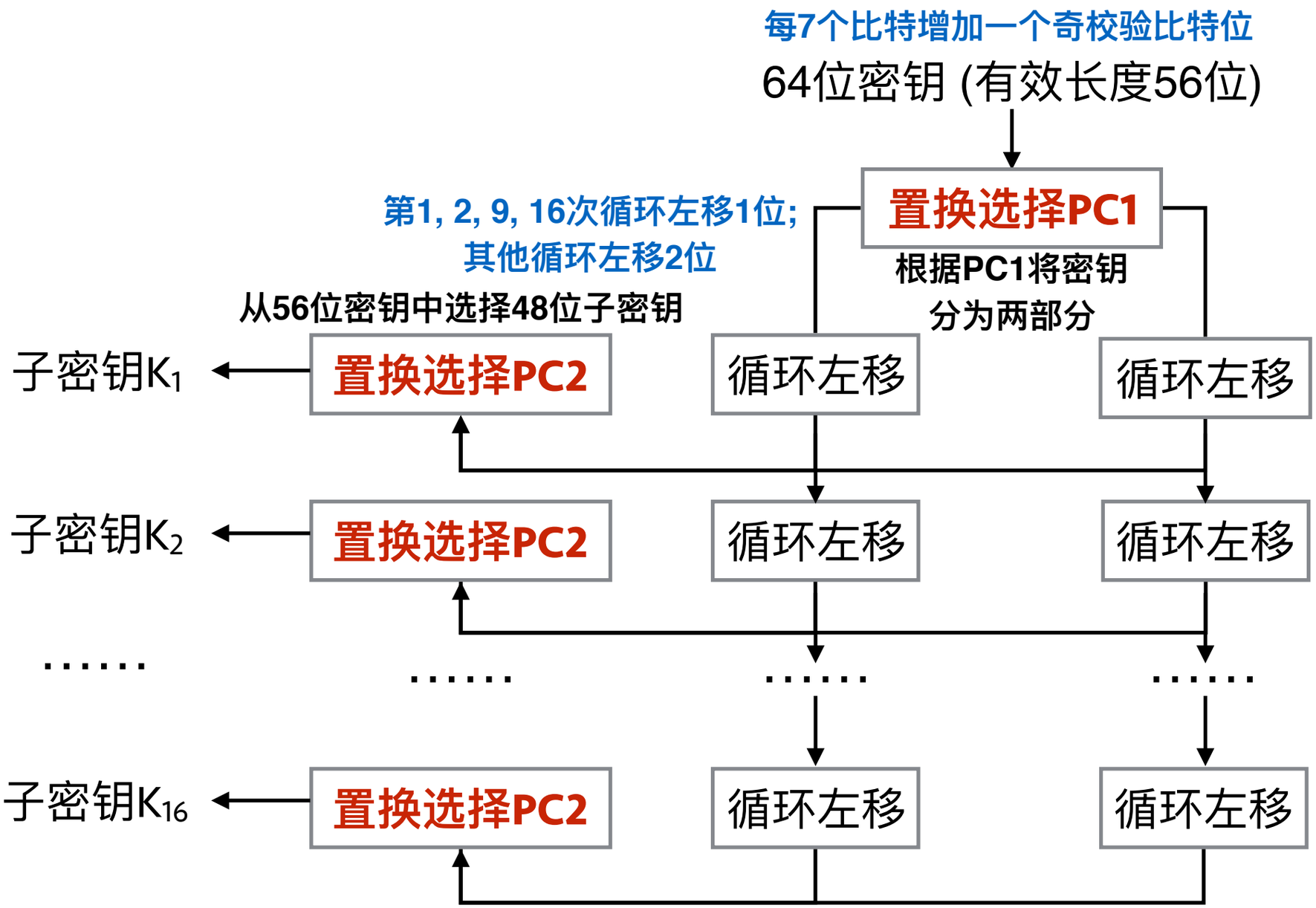


初始置换IP

- 根据IP置换表产生明文分组的置换
  - 例子: 置换后明文分组第一个比特源于输入明文第58个比特; 置换后明文分组第二个比特源于输入明文第50个比特...

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

子密钥生成


每7个比特增加一个奇校验比特位
64位密钥 (有效长度56位)
置换选择PC1
根据PC1将密钥
分为两部分
第1, 2, 9, 16次循环左移1位;
其他循环左移2位
从56位密钥中选择48位子密钥
子密钥K1
置换选择PC2
循环左移
循环左移
子密钥K2
置换选择PC2
循环左移
循环左移
.....
.....
.....
.....
子密钥K16
置换选择PC2
循环左移
循环左移

PC1和PC2

- PC1基于64位密钥选择产生**2 * 28**位部分密钥
  - ➡ 64位密钥的第8, 16, 24,…,64位为奇校验位

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

PC1-左

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

PC1-右

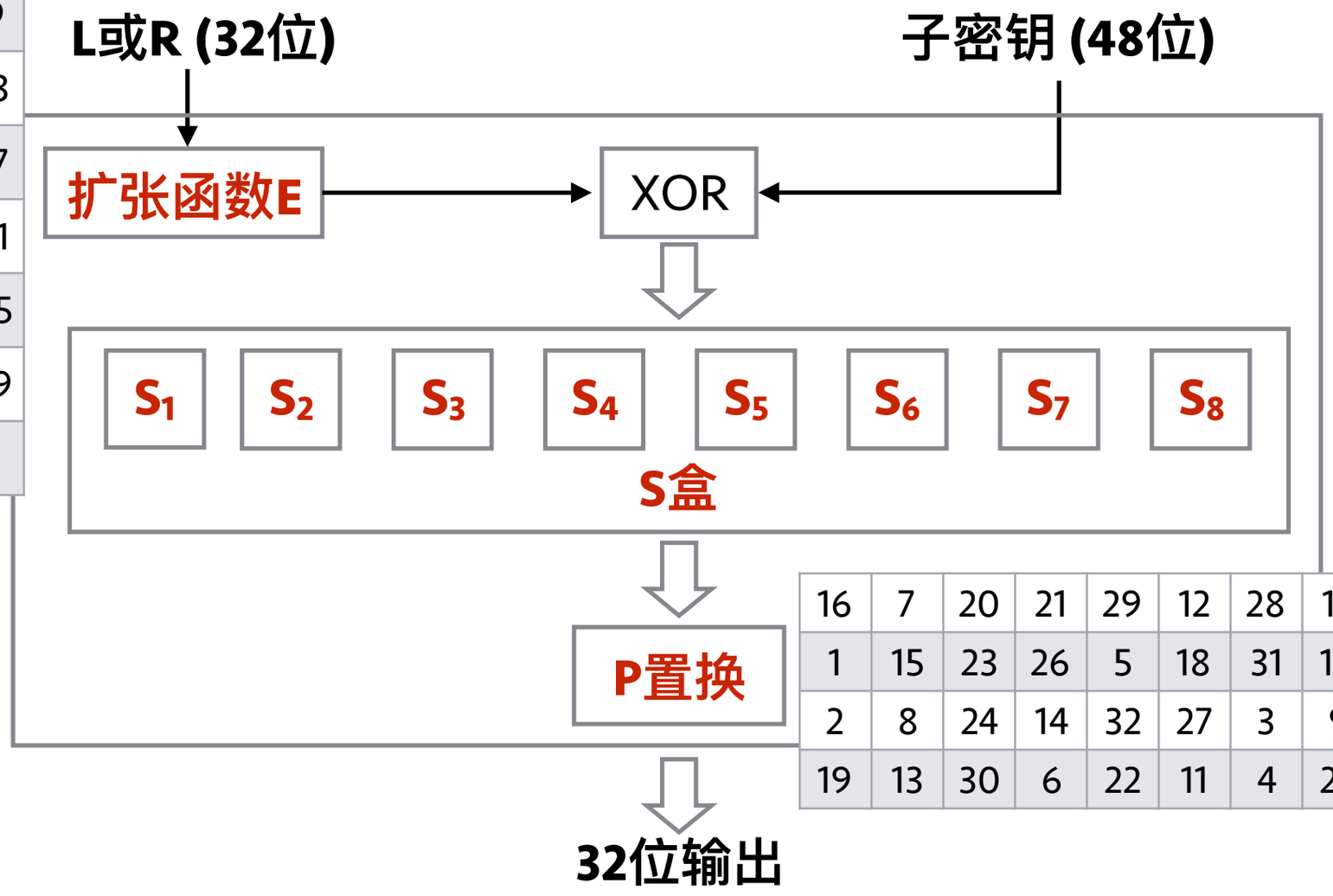
- PC2从移位后的左||右部分密钥选出**48位**子密钥

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

PC2

Feistel轮函数

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



S盒

- DES加密中**唯一非线性变换部分**
  - 输入**6比特**$b_0b_1b_2b_3b_4b_5$: b_0b_5决定输出的**4比特**所在的行; $b_1b_2b_3b_4$决定输出的**4比特**所在的列

S_1	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

  - 例子: **011011**的输出结果在S_1盒替换表的第3行 (01), 第15列 (1101), 最终输出5 (0101)

更多S盒参数参考**DES补充材料**

逆置换IP^{-1}

- 根据16轮Feistel网络运算输出**置换**产生最终密文
  - 输入和输出均为64比特位

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^{-1}置换表

OpenSSL中的DES

- 头文件: **#include<openssl/des.h>**

```
DES_cblock key;
DES_random_key(&key);                                          // 随机产生密钥

DES_key_schedule keys;
DES_set_key_checked(&key, &keys);                              // 生成子密钥

DES_cblock plaintext, ciphertext;
DES_ecb_encrypt(&plaintext, &ciphertext, &keys, DES_ENCRYPT);  // 加密
DES_ecb_encrypt(&ciphertext, &plaintext, &keys, DES_DECRYPT);  // 解密
```

- DES_cblock结构: typedef unsigned char des_cblock[8];

- 参考**更多DES函数接口**

扩展阅读

- **分组密码算法的工作模式**: 加密超过分组长度数据
- **Padding**: 扩充数据达到分组长度的整数倍
- **Ciphertext stealing**: 保证明文密文具有相同的(任意)长度

对称加密技术实验

流密码: RC4

分组密码: DES

哈希函数: SHA-1

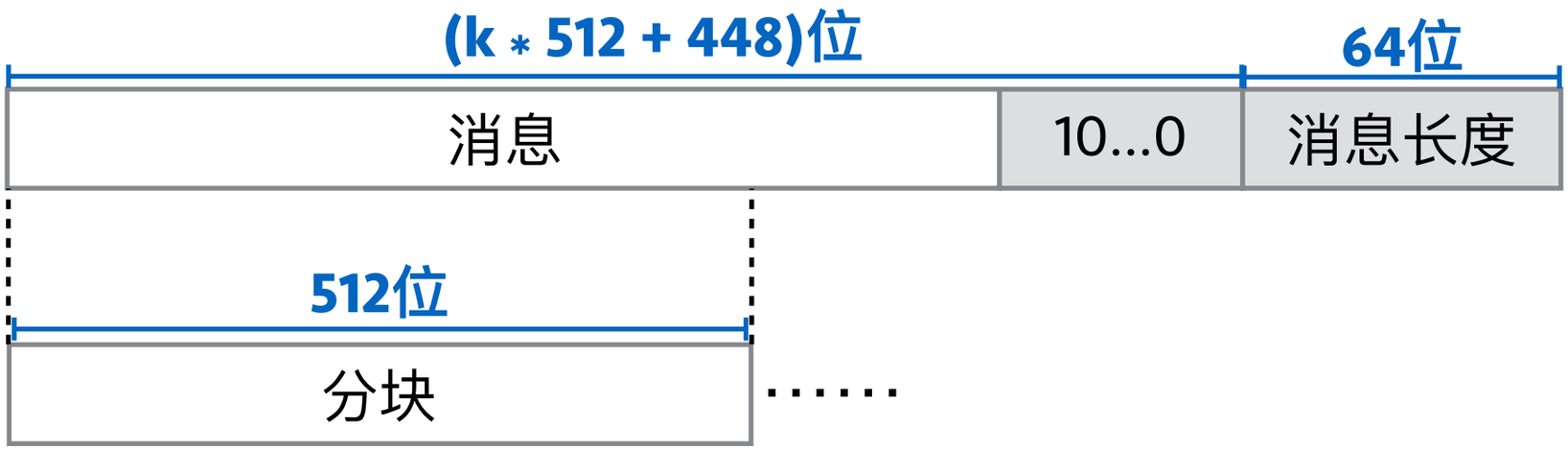
哈希函数

- 哈希函数: 将**任意长度**数据块映射为**固定长度**哈希值
  - 特性: **单向性**, **抗碰撞**

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
消息摘要长度	**160**	224	256	384	512
消息长度	**$< 2^{64}$**	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
分组长度	**512**	512	512	1024	1024
字长度	**32**	32	32	64	64
步骤数	**80**	64	64	80	80

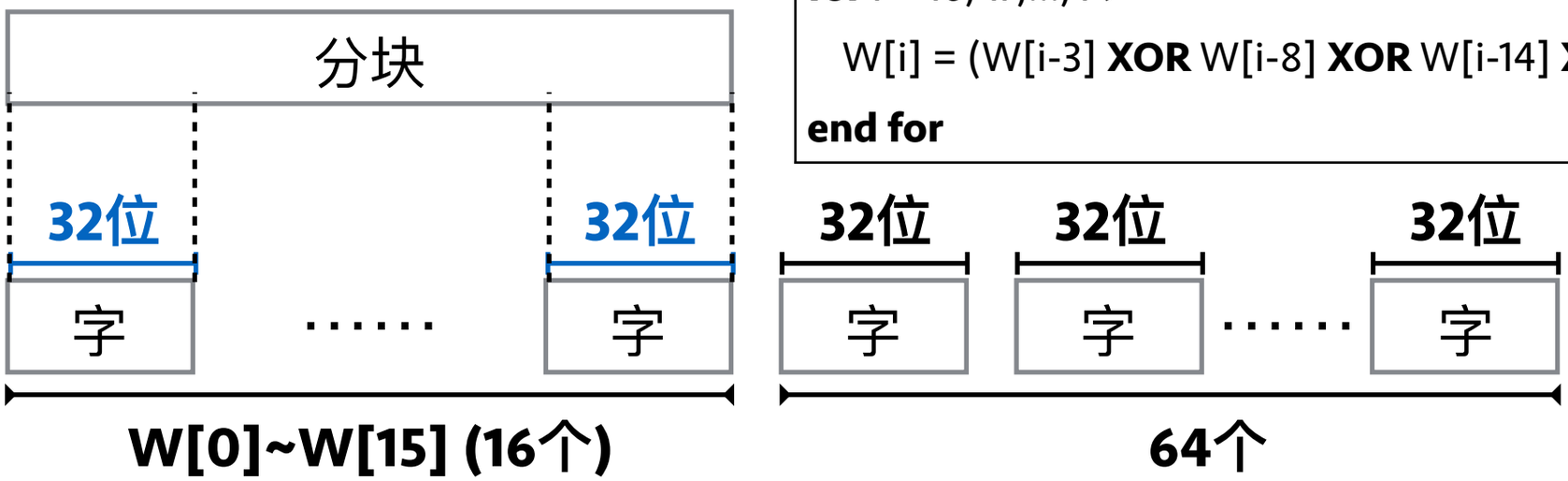
- 应用: 数据**完整性**检测 (防篡改)

SHA-1算法: 预处理 (padding)



- 附加填充**一个"1"和若干个"0"**, 填充后长度**模512余448**
- 附加填充**64位消息长度**, 填充后长度为512的整数倍
- 产生消息分块, 分块长**512位**

SHA-1算法: 产生字 (word)




```
for i = 16, 17,..., 79
   W[i] = (W[i-3] XOR W[i-8] XOR W[i-14] XOR W[i-16]) leftrotate 1
end for
```

- 分割分块: 产生**16个字**, 字长**32位**
- 扩展字: 基于分块的16个字, 产生额外**64个扩展字**

SHA-1算法: 计算哈希值

f(t; B, C, D) = (B **AND** C) **OR** ((**NOT** B) **AND** D)	(0 <= t <= 19)
f(t; B, C, D) = B **XOR** C **XOR** D	(20 <= t <= 39)
f(t; B, C, D) = (B **AND** C) **OR** (B **AND** D) **OR** (C **AND** D)	(40 <= t <= 59)
f(t; B, C, D) = B **XOR** C **XOR** D	(60 <= t <= 79)
K(t) = 5A827999	(0 <= t <= 19)
K(t) = 6ED9EBA1	(20 <= t <= 39)
K(t) = 8F1BBCDC	(40 <= t <= 59)
K(t) = CA62C1D6	(60 <= t <= 79)

预定义函数

H_0 = 67452301
H_1 = EFCDAB89
H_2 = 98BADCFE
H_3 = 10325476
H_4 = 10325476

寄存器
H_0, H_1, H_2, H_3, H_4
初始化

$A = H_0, B = H_1, C = H_2, D = H_3, E = H_4$

for t = 0, 1,..., 79

  tmp = A **leftrotate** 5 + **f**(t; B, C, D) + E + W[t] + K(t)

  E = D, D = C, C = B **leftrotate** 30, B = A, A = tmp;

end for

$H_0 = H_0 + A, H_1 = H_1 + B, H_2 = H_2 + C, H_3 = H_3 + D, H_4 = H_4 + E;$

分块处理 (+为2^{32}的模加运算)

- 更新后的H_0, H_1, H_2, H_3, H_4用于处理下一个分块
- 所有分块处理完毕后形成的**$H_0||H_1||H_2||H_3||H_4$**即为SHA-1的最终输出
- 参考**SHA-1伪代码**

OpenSSL中的SHA-1

- 头文件: **#include<openssl/sha.h>**

```
/*定义msg为消息, msg_len为消息长度, hash为至少20字节缓冲区*/
SHA1(msg, msg_len, hash)                                    // SHA-1函数

/*处理消息难以在内存中存放的情况*/
SHA_CTX ctx;
/*将消息分块, 每个块放入chunk缓冲区, chunk_len为块长度*/
SHA1_Update(&ctx, chunk, chunk_len)
SHA1_Final(hash, &ctx);
```

➡ 参考**更多SHA函数族**