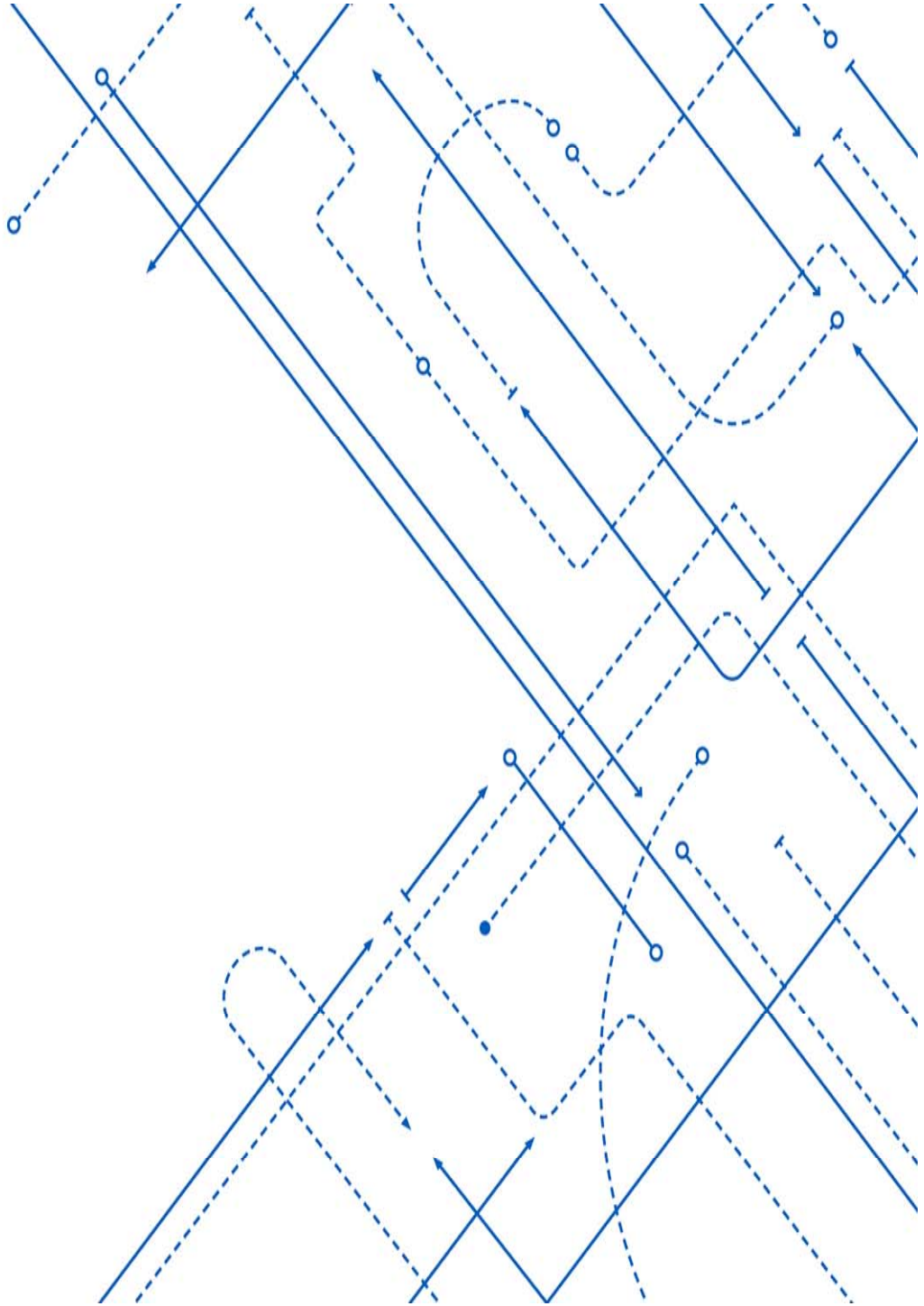
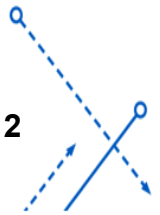
第5章 DES算法



浙江大学
ZHEJIANG UNIVERSITY

DES算法

- 1973年, NBS(National Bureau of Standards)公开征集一个希望成为国家标准的加密算法。
- 1974年, IBM公司提交了一个叫做LUCIFER的算法。NBS把该算法交给National Security Agency评估, 后者对此算法做了一些修改。
- 1975年, NBS公布了经NSA修改过的LUCIFER算法, 称为DES（Data Encryption Standard）算法。



DES算法

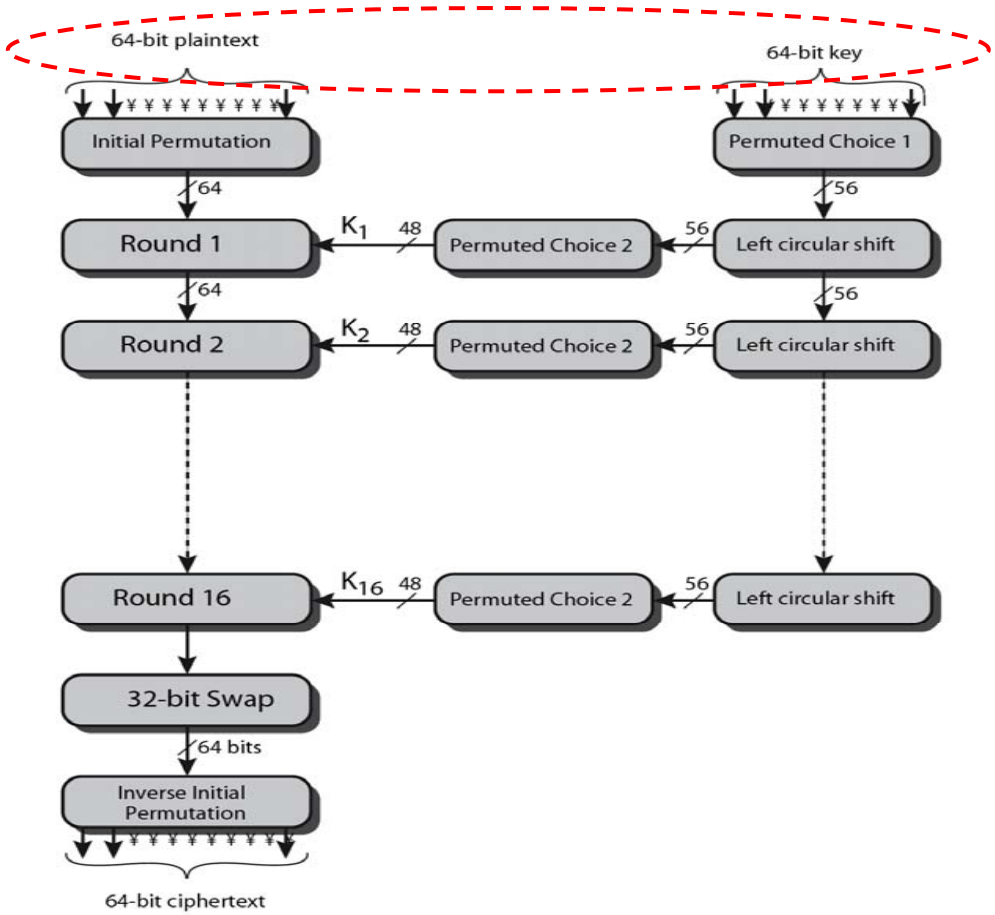
特点：

- 明文是64位=8字节；
- 密钥是64位=8字节；
- 加密与解密的密钥相同；

缺点：

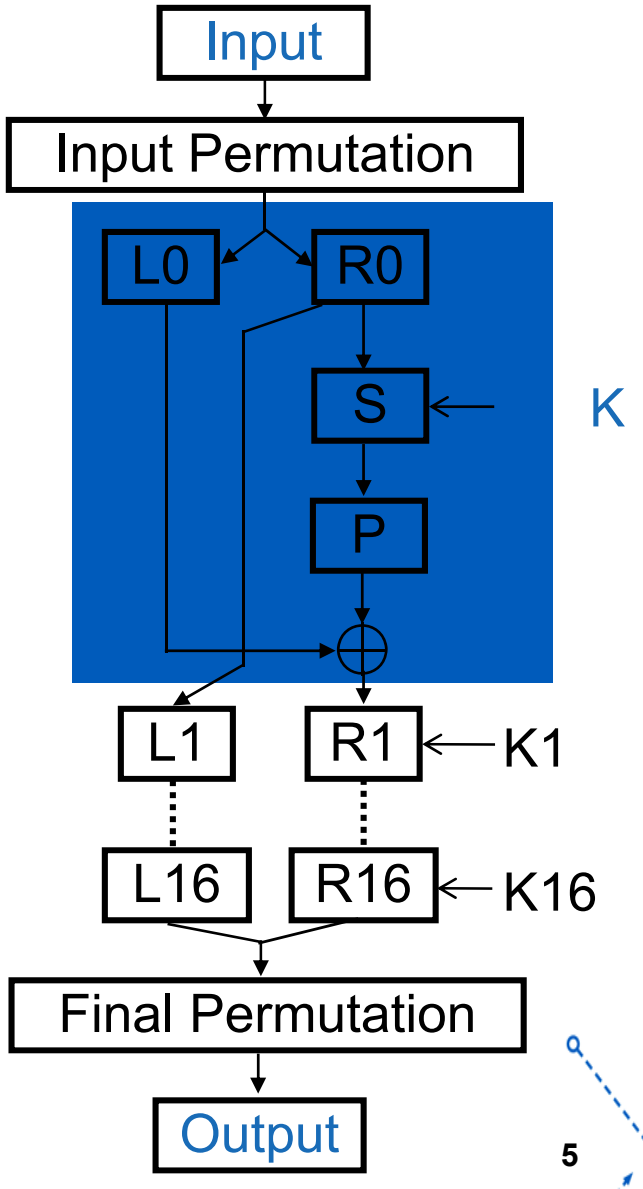
- 1. 密钥太短（Lucifer 128位）
- 2. 差分分析 可以攻击 DES 算法
- 3. Sbox的设计没有公开，怀疑有后门

DES算法流程


64-bit plaintext
64-bit key
Initial Permutation
Permuted Choice 1
64
56
Round 1
K1
48
Permuted Choice 2
56
Left circular shift
64
56
Round 2
K2
48
Permuted Choice 2
56
Left circular shift
Round 16
K16
48
Permuted Choice 2
56
Left circular shift
32-bit Swap
64 bits
Inverse Initial Permutation
64-bit ciphertext

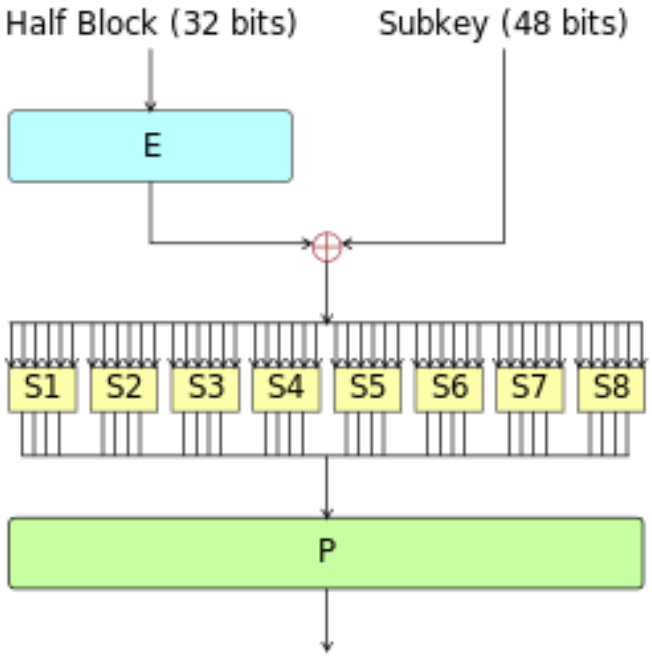
DES算法流程

- DES算法有16个相同的处理阶段，称为轮（round）。还有一种初始和最终的排列，称为IP和FP，它们是相反的（IP“撤销”FP的作用，反之亦然）。IP和FP没有密码学意义，它们的加入是为了方便在20世纪70年代中期基于8位的硬件中加载块。
- 在主循环之前，块（block）被分成两个32位的两半L&R，并交替处理；这种交叉被称为Feistel方案。Feistel结构确保解密和加密是非常相似的过程。唯一的区别是，在解密时，子密钥以相反的顺序应用。算法的其余部分是相同的。这大大简化了实现，尤其是在硬件中，不需要单独的加密和解密。



DES算法流程

- ⊕ 表示异或操作。F函数将半个块与密钥一起置乱。然后，F函数的输出与块的另一半相结合，在下一轮之前对这两个部分进行交换。在最后一轮后，两半进行交换；这是Feistel结构的一个特点，使得加密和解密过程类似。



DES算法流程

(1)64位明文在进入加密前有个打乱的过程，要用到一张打乱表(static char ip[64])，把64位的顺序打乱,例如:

ip[0]=58表示源数据中的第58位(实际是第58-1=57位)要转化成目标数据中的第0+1=1位，其中下标代表的是以0为基数的目标位，而数组的元素值代表的是以1为基数的源位;

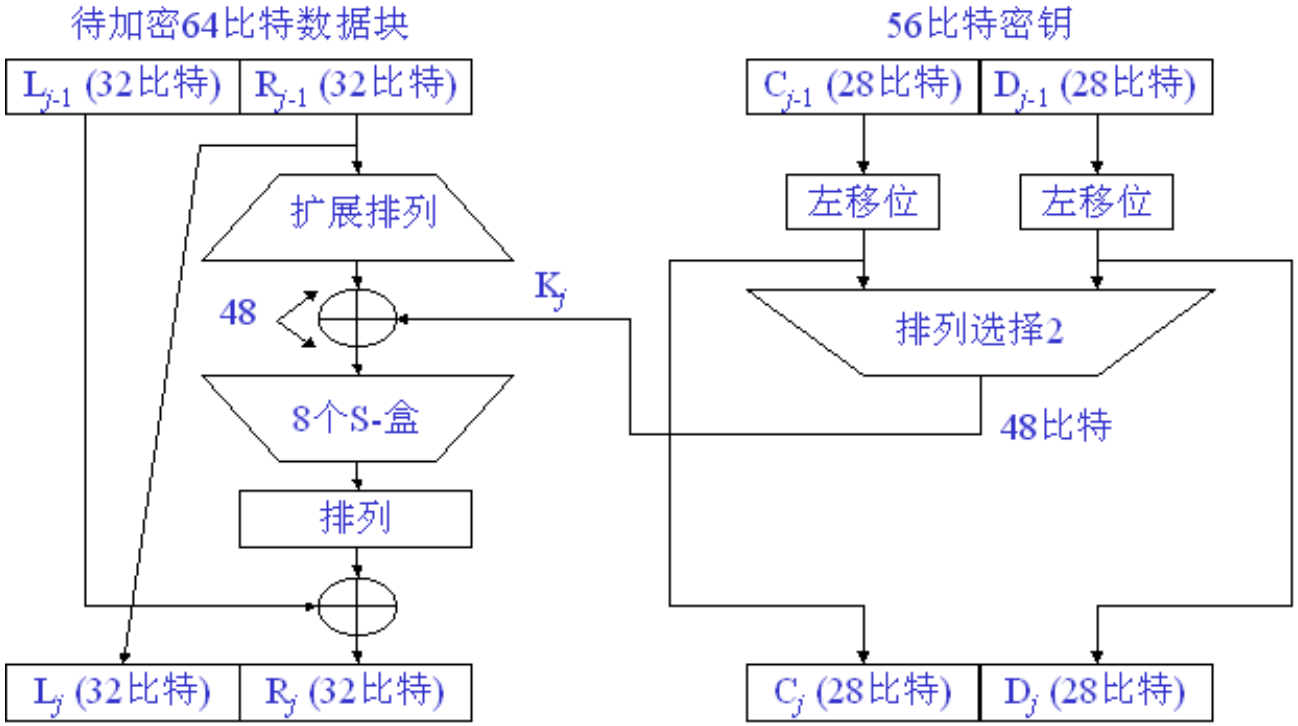


图2.6 DES算法每轮处理过程

DES算法流程

(2)64位明文经过加密变成密文后还要有一个打乱的过程，此时要用到的打乱表为static char fp[64];

(3)8个sbox转化出来32位结果也需要打乱，这张打乱表为static char sbox_perm_table[32];

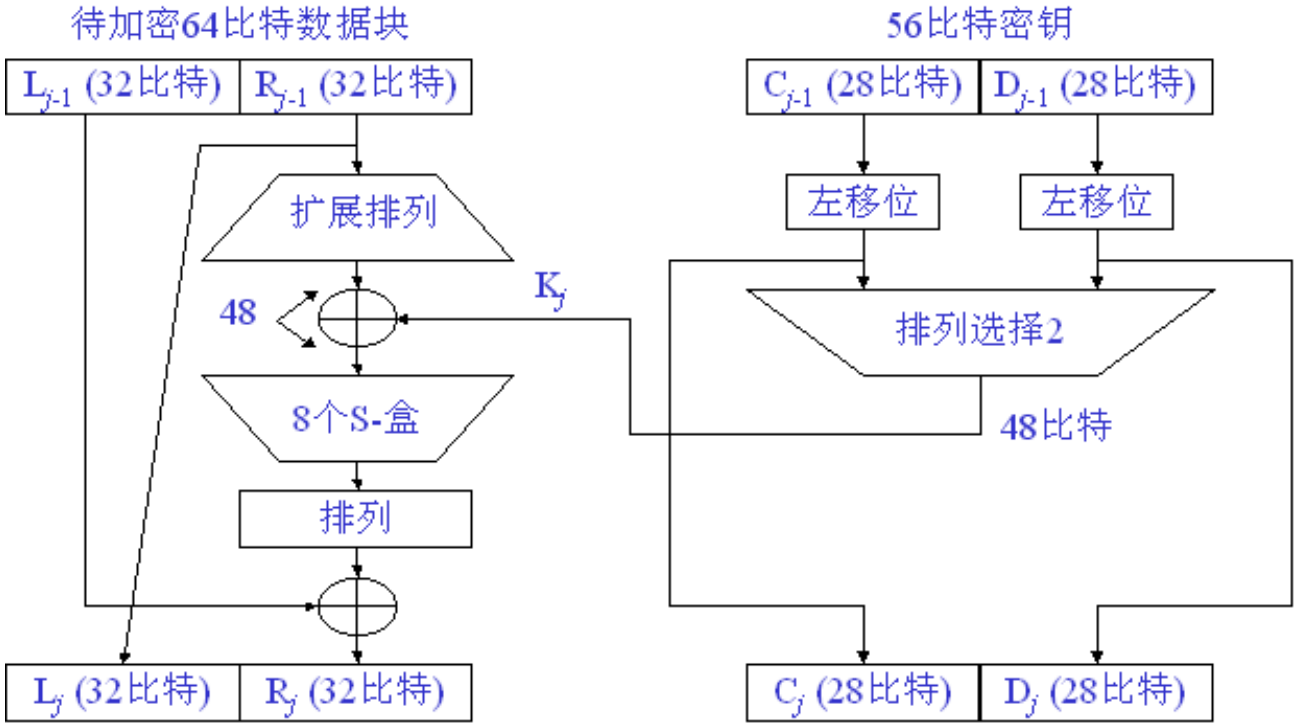


图2.6 DES算法每轮处理过程

DES算法流程

(4)64位密钥要砍掉8位变成56位, 此时要用到一张表:

static char key_perm_table[56];

(5)56位密钥在循环左移后, 要提取其中的48位,此时也要用到一张表:

static char key_56bit_to_48bit_table[48];

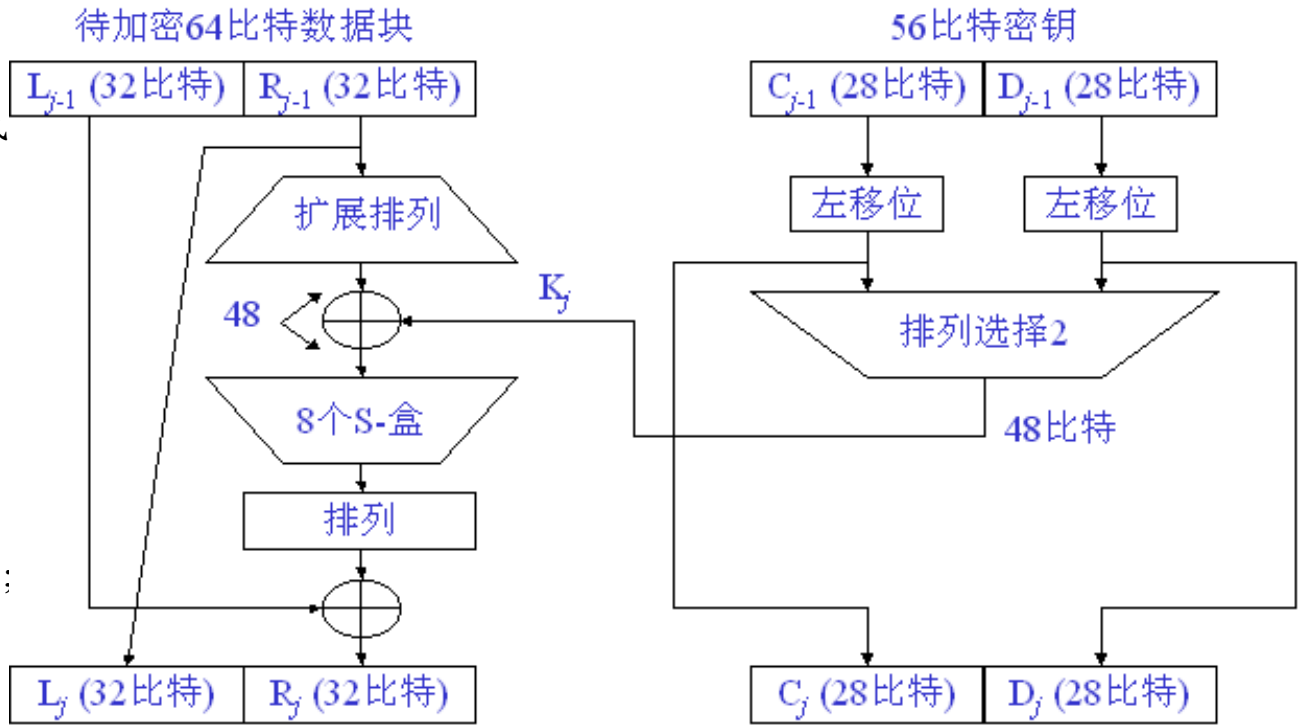


图2.6 DES算法每轮处理过程

初始排列 Initial Permutation IP

- 数据计算的第一步
- IP重新排列输入数据位
- 偶数位到左半部LH，奇数位到右半部RH
- 结构非常规则（h/w结构简单）
- 例子:
- IP（675a6967 5e5a6b5a）=（ffb2194d 004df6fb）

初始排列 Initial Permutation IP

IP

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

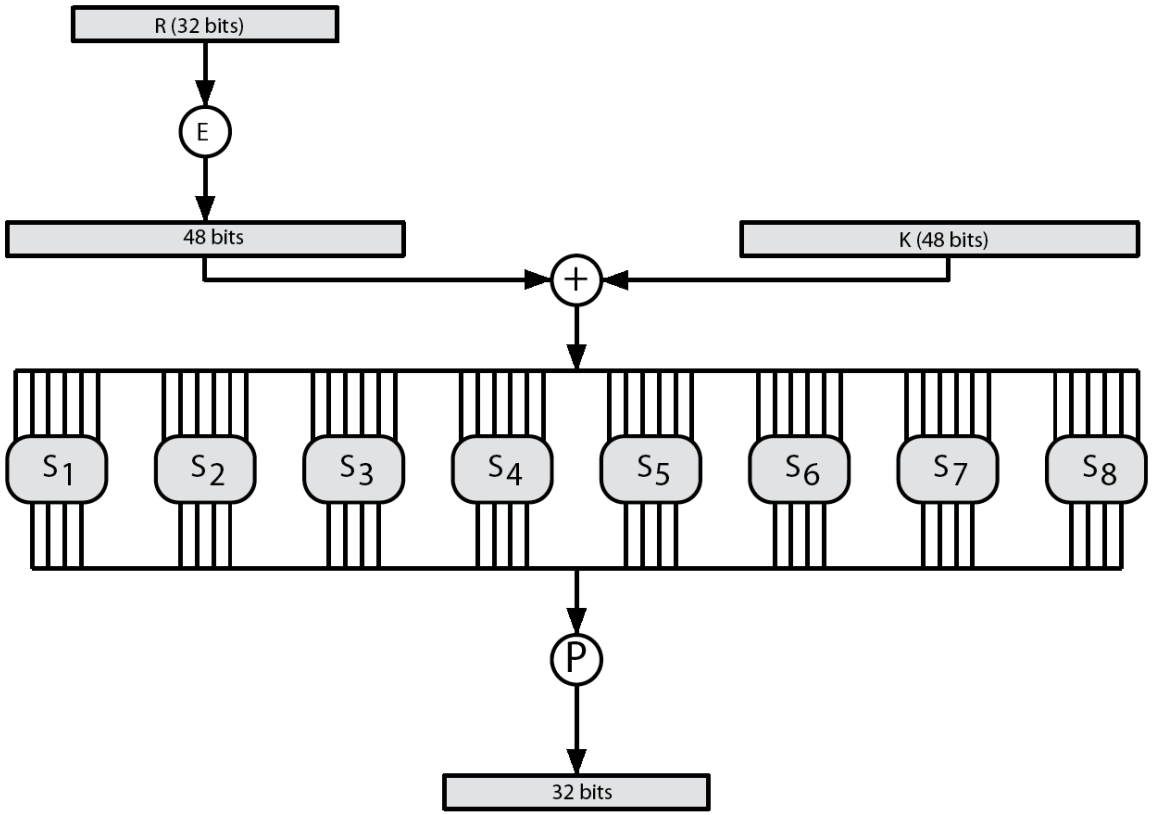
DES Round Structure - Feistel（F）函数

- 使用两个32位数据 L&R （块的两部分）
- 任何Feistel密码，可以描述为：

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus \mathrm{F}(R_{i-1}, K_i)$$

- 1. 扩展（Expansion）：32位半块通过复制一半位，使用扩展置换（perm E）扩展到48位。
- 2. 密钥混合（Key mixing）：使用异或操作将结果与子密钥（subkey）组合。
- 3. 替换（Substitution）：在子密钥中混合后，块被分成八个6位块，然后由替换盒（Substitution boxes，S-boxes）进行处理。根据以查找表形式提供的非线性变换，八个S盒中的每一个都将其六个输入位替换为四个输出位。S盒是DES安全性的核心。如果没有S盒，密码是线性的，容易被破解。
- 4. 排列（Permutation）：最后，S盒的32个输出按照固定排列P盒重新排列。这样设计的目的是，在排列之后，来自这一轮中每个S盒的输出的位在下一轮中分布在四个不同的S盒上。

DES Round Structure



替换盒 Substitution Boxes S

- 有8个S盒，每个S盒接受6位作为输入，产生4位作为输出。
- 每个S盒实际上是4个小的4位盒
- 外部位第1位和第6位（作为行位）选择一行
- 内部位第2-5位（作为列位）选择一列
- 外部位和内部位共同选择下，确定输出
- 结果是8个4位，即32位
- 行选择取决于数据和密钥
- 例子:
  - S（18 09 12 3d 11 17 38 39）=5fd25e03

替换盒 Substitution Boxes S

***S*₁**

Column Number

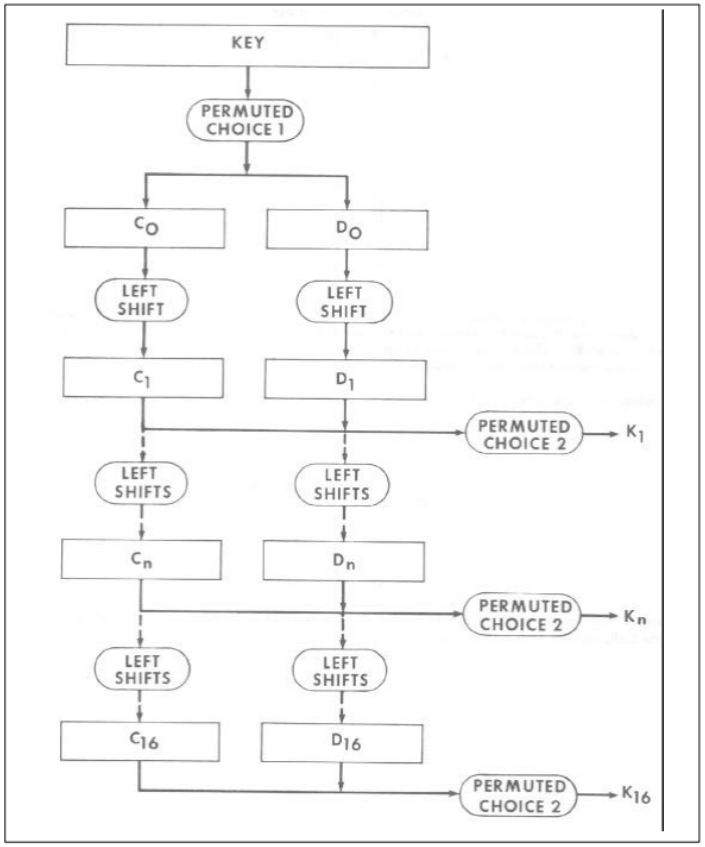
Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the range 0 to 3. Let that number be i. The middle 4 bits of B represent in base 2 a number in the range 0 to 15. Let that number be j. Look up in the table the number in the i'th row and j'th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input 011011 the row is 01, that is row 1, and the column is determined by 1101, that is column 13. In row 1 column 13 appears 5 so that the output is 0101. Selection functions $S_1,S_2,\ldots,S_8$ of the algorithm appear in Appendix 1.

DES Key Schedule

- 形成每轮中使用的子密钥（subkey）
- 通过置换选择1（PC-1）从初始64位中选择密钥的56位——剩余的8位要么被丢弃，要么被用作奇偶校验位。
- 然后将56位分成两个28位的两半；此后，每一半都要单独处理。
- 16个阶段包括：
  - 两半都向左旋转一或两位（每轮指定）
  - 然后通过置换选择2（PC-2）选择48个子密钥——左半部分24位，右半部分24位。
  - 解密的key schedule与加密类似，子密钥的顺序与加密相反。除此之外，解密过程与加密过程相同。

DES Key Schedule



DES 解密

- 使用Feistel 设计，再次使用子密钥以相反顺序执行加密步骤（SK16…SK1）
- IP撤销加密的最后一步FP
- 第1轮SK16撤销第16轮加密
- …
- 第16轮SK1撤销第1轮加密
- 然后最终FP撤销初始加密IP
- 从而恢复原始数据值

DES －源代码分析

- void des_set_key(char *key)

- (1) 根据key_perm_table从8字节的key中选择56位存放到pc1m，每一位保存为一个字节。
- (2) 根据key_rol_steps对pc1m左边28个元素及右边28个元素分别进行循环左移，移位以后的结果保存到pcr中。
- (3) 根据key_56bit_to_48bit_table从pcr中选出48个元素，每6个元素靠右对齐合并成1个字节，保存到kn中。

DES －源代码分析

- void sbox_output_perm_table_init(void)

- (1) 根据sbox_perm_table生成一张反查表sbox_perm_table_inverse

- (2) 根据sbox生成sbox_output_perm_table:进入sbox的6位数据出来后变成4位，再打散并保存到sbox_output_perm_table中的一个32位元素内。

DES －源代码分析

- perm_init(char perm[16][16][8], char p[64])

- (1) p定义的是一张64位打乱表，下标为目标位，元素值为源位。
- (2) 假定X是一个任意的64位数，现把它的64位按从左到右的顺序划分成16组，每组4位。
- (3) 设j是第i组中的4位，显然j的值一共有16种变化，现通过查表p得到j中每个位分别落在64位中的哪一位，于是就把j中的4位打散并保存到perm[i][j]的8个字节内。

DES －源代码分析

- permute(char *inblock, char perm[16][16][8], char *outblock)

- (1) 把inblock中的8字节划分成16组，每组4位。
- (2) 设j是第i组中的4位，查perm[i][j]得8字节即64位。
- (3) 把每组查perm所得的64位求或，结果保存到outblock中。

DES －源代码分析

- long f(unsigned long r, unsigned char subkey[8])

- (1) 根据plaintext_32bit_expanded_to_48bit_table把r扩展成48位，并把这48位划分成8组，每组6位。
- (2) 把这8组数按顺序分别与subkey中包含的8组数做异或运算
- (3) 异或后仍旧得到8组数，每组6位。
- 在sbox_output_perm_table中按顺序查这8组数，每组数6位进去，出来一个包含有打散4位的32位数，把8个32位数求或即得f()的返回值。

DES －源代码分析

- long f(unsigned long r, unsigned char subkey[8])

64位明文分成L32、R32

64位key选56位,砍8位

56位key分成L28、R28

L28及R28循环左移n次(1<=n<=2)

56位key中选取48位

- R32展开成48位 xor 48位key

进入8个sbox(每6位进入一个sbox并出来4位)

sbox共输出32位,打乱后生成一个新的32位数N32

最后L32' = L32 xor N32

DES －作业1

- http://10.71.45.100/bhh/mydes.c
- 重写DES算法中的核心函数f

```
static long32 f(ulong32 r, unsigned char subkey[8]) {
   unsigned char s[4], plaintext[8], m[4], t[4];
   unsigned long int rval;
   (1) 复制r到s, 即把r当成4个字节看待(注意大小端)。
   (2) 根据plaintext_32bit_expanded_to_48bit_table这张表, 把s中的4字节共32位扩展成48位并
       保存到数组plaintext中; /* plaintext每个元素的左边2位恒为0, 右边6位用来保存数据 */
       32位转48位的过程要求使用双重循环来做,外循环8次,内循环6次,其中内循环每次只提取1位;
       注意提取某1位的时候只能从数组s中取(要计算该位是第几个字节中的第几位), 不得从参数
       r中获取; 判断某位是否为1的时候可以使用表bytebit。
   (3) 把plaintext中的8个字节与subkey中8个字节异或, 异或后的值保存到plaintext内;
        该过程使用8次循环来做。
   (4) 取出plaintext[i]查表sbox[i]得4位数, 循环8次可得8个4位数, 按从左到右顺序合并
        得32位数保存到数组m中。
   (5) 根据sbox_perm_table把m中的32位数打乱保存到数组t中, 该过程需要使用32次循环来做,
        每次循环提取1位。
   (6) 复制t到rval(注意大小端), 并返回rval。
}
```

DES - 作业2

➢ 改写perm_init()函数及permute()函数

把以下两个指针定义

```
static char (*iperm)[16][8];
static char (*fperm)[16][8];
```

改成:

```
static char (*iperm)[256][8];
static char (*fperm)[256][8];
```

目的是把原先64位划分成16组(每组4位)改成64位划分成8组(每组8位)。注意原先每组4个位仅有16种变化，现在改成每组8个位有256种变化。

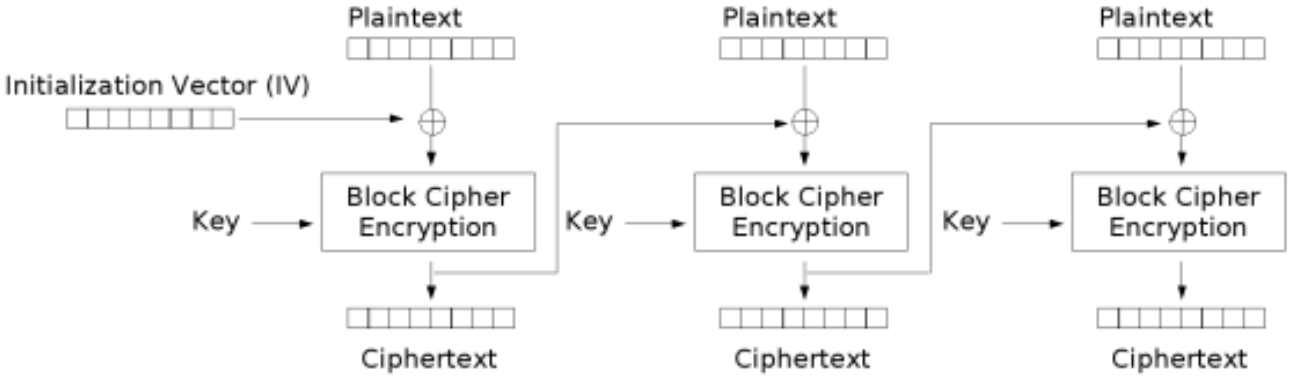
经过这样修改后, perm_init()函数会把每组8位按打乱表打乱后变成分散在8个字节中的8个位,最后把8组8字节进行或运算合并成完整的64位。

对应地，我们需要修改以下函数使它们符合上述要求:

```
static void perm_init(char perm[8][256][8], char p[64]);
static void permute(char *inblock, char perm[8][256][8], char *outblock);
int des_init(int mode);
```

DES －分组加密模式 － CBC

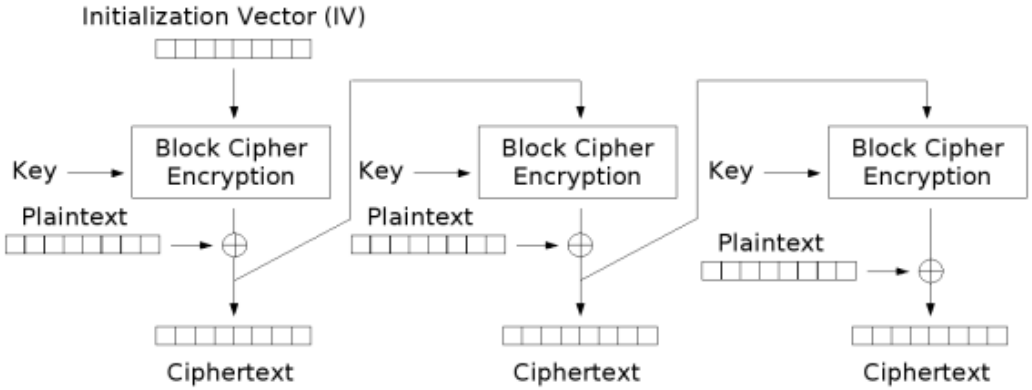
- 需要一个8字节的初始向量(initialization vector)或初始变量(starting value)。令iv=这8个字节。现要加密block[0], block[1]：
- c[0] = des_encrypt(block[0] ^ iv);
- c[1] = des_encrypt(block[1] ^ c[0]);
- c[0]和c[1]就是密文。



Cipher Block Chaining (CBC) mode encryption

DES - 分组加密模式 - CFB

- 同样需要一个8字节的初始向量。令iv=这8个字节。现在要加密b(共8字节):
- v = des_encrypt(iv); /* v包含8字节 */
- c[0] = b[0] ^ v[0]; /* c[0]是1字节密文 */
- iv = iv << 8 | c[0]; /*iv左移1字节,末尾补c[0]*/
- v = des_encrypt(iv);
- c[1] = b[1] ^ v[0]; /* c[1]是1字节密文 */

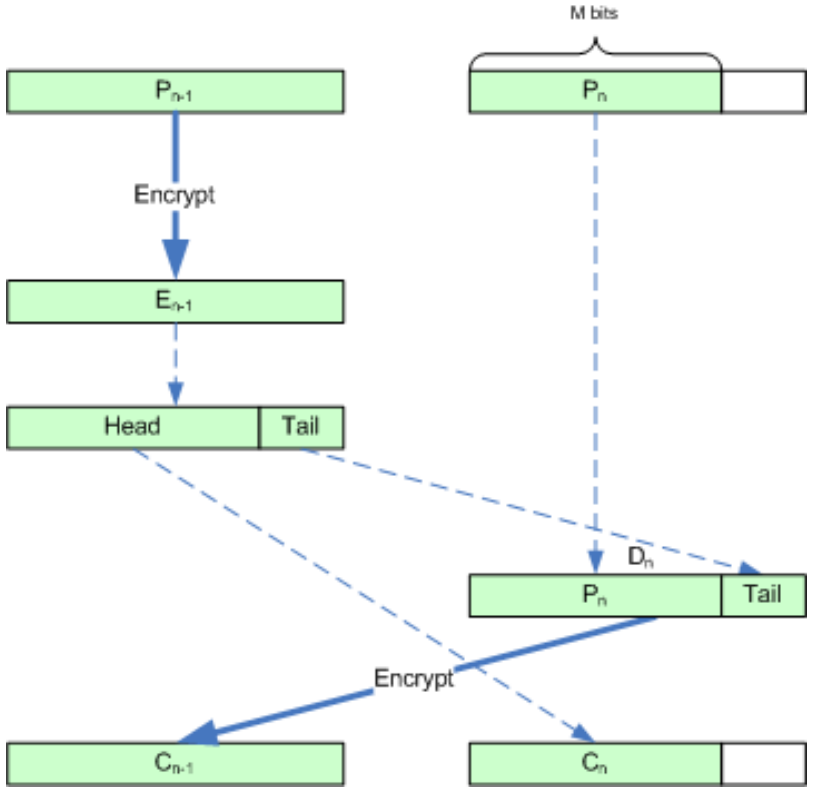


Cipher Feedback (CFB) mode encryption

DES －ciphertext stealing － ECB

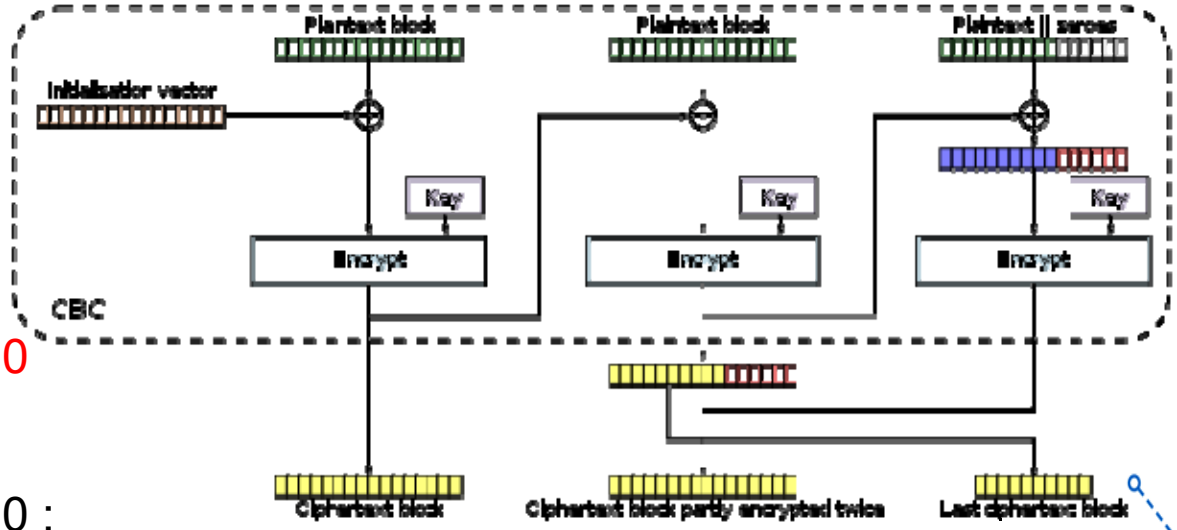
- E_{n-1}= Encrypt （K， P_{n-1}）
- C_n = Head (E_{n-1}, M)
- D_n = P_n || Tail (E_{n-1}, B−M)
- C_{n-1}= Encrypt (K, D_n)

- 举例：
- 12345678 12345
- ~~12345678~~
- 12345~~678 12345~~
- 12345~~678 12345~~



DES －ciphertext stealing － CBC

- $X_{n-1}=P_{n-1}$ XOR C_{n-2}
- E_{n-1}= Encrypt (K, X_{n-1})
- C_n = Head (E_{n-1}, M)
- P = P_n || 0^{B-M} 在Pn的末端加上零，以产生长度为B的P。
- D_n = E_{n-1} XOR P
- C_{n-1} = Encrypt (K, D_n)
- 举例：
- IV
- 12345678 12345,0,0,0
- ~~12345678~~
- 12345~~678~~ ~~12345~~,0,0,0 ;
- 12345~~678~~ ~~12345~~



DES －作业3

- http://10.71.45.100/bhh/MyDesCbc.c
- 实现函数des_cfb_encrypt：调用DES算法以CFB模式对明文进行加密

本题要求实现一个函数`des_cfb_encrypt()`，该函数原型如下:

```
void des_cfb_encrypt(unsigned char p[], int n, unsigned char des_seed_iv[],
/*==================*/ unsigned char des_iv[], unsigned char des_seed_key[]);
```

其中p为明文，n为p的长度，des_iv为调用DES算法以CFB模式加密时需要的8字节初始向量。

你可以在本函数内调用以下函数实现DES加密:

```
void des_encrypt(unsigned char *block, unsigned char *des_key);
```

其中block在调用前指向明文，调用后指向密文，des_key指向8字节DES密钥。

雪崩效应

- 加密算法（尤其是块密码和加密散列函数）的一种理想属性
- 其中一个输入或关键位的变化导致约一半的输出位发生变化
- 试图猜密钥是不可能的
- DES表现出强烈的雪崩效应

DES - key size

- 密钥长度为56位时，有2^{56}个可能的密钥，即7.2×10^{16}种
- 暴力搜索看起来非常困难
- 但最近的进展表明这是可能的:
  - 1997年，大型计算机网络上，在几个月内
  - 1998年，专门的h/w（EFF），在几天内
  - 1999年，结合上述两者，在22小时内
- 密钥搜索攻击不仅仅是遍历所有可能的密钥。除非提供已知的明文，否则分析员必须能够识别明文。
- 必须考虑DES的替代方案，其中最重要的是AES和三重DES。

DES - Analytic Attacks

- 利用密码的深层结构
  - 通过收集有关加密的信息
  - 最终可以恢复部分/所有子密钥位
  - 在必要时彻底搜索其余的子密钥位
- 一般来说，这些都是统计攻击
  - 差分密码分析（differential cryptanalysis）
  - 线性密码分析（linear cryptanalysis）
  - 相关密钥攻击（related key attacks）

DES – 差分密码分析

- 密码分析最重要的最新进展之一
- 分析分组密码的有力方法
- 用于分析大多数当前的分组密码，取得了不同程度的成功
- 比起Lucifer，DES对它有更合理的抵抗力

DES – 差分密码分析

- 针对Feistel密码的统计攻击
- 使用以前未使用过的密码结构
- S-P网络的设计使函数f的输出同时受到输入和key的影响
- 因此，在不知道密钥的情况下，无法通过密文追溯明文
- 差分密码分析比较了两对相关的加密，如果有足够多的合适对，它们可能会泄露有关密钥的信息

DES – 差分密码分析比较加密对

- 当使用相同的子密钥时，使用输入中的已知差分搜索输出中的已知差分
- 下式显示了如何消除密钥K的影响，从而实现差分分析。

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus \mathrm{f}(m_i, K_i)] \oplus [m'_{i-1} \oplus \mathrm{f}(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [\mathrm{f}(m_i, K_i) \oplus \mathrm{f}(m'_i, K_i)]\end{aligned}$$

DES - 差分密码分析比较加密对

下面讨论的 DES 算法中符号有所改变。设原始明文分组 m 被分成两部分 m_0、m_1。DES 的每一轮迭代都将其右侧的输入映射为左侧的输出，而将左侧的输入和该轮子密钥经运算后作为右手的输出。所以在每一轮迭代中，仅产生 32 位新数据。如果我们将其记为 $m_i(2 \leqslant i \leqslant 17)$，可得以下关系式：

$$m_{i+1} = m_{i-1} \oplus \mathrm{f}(m_i, K_i), \qquad i = 1, 2, \cdots, 16$$

在差分密码分析中，我们从两条已知异或 XOR 差分 $\Delta m = m \oplus m'$ 的明文 m 和 m' 开始，考虑中间数据的差分 $\Delta m_i = m_i \oplus m_i'$。我们有

$$\begin{aligned} \Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus \mathrm{f}(m_i, K_i)] \oplus [m'_{i-1} \oplus \mathrm{f}(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [\mathrm{f}(m_i, K_i) \oplus \mathrm{f}(m'_i, K_i)] \end{aligned}$$

现在，假设函数 f 有许多对具有相同差分的输入，当使用相同的子密钥时产生相同差分的输出。更精确地说，若差分为 X 的所有输入，产生差分为 Y 的输出占所有输出的百分比为 p，我们就称由差分 X 导致差分 Y 的概率为 p。假设有许多 X 都会以很高的概率产生某些特定的差分。因此，若以很高的概率知道 Δm_{i-1} 和 Δm_i，就能以很高的概率知道 Δm_{i+1}。而且，如果知道很多那样的差分数据，那么确定函数 f 所使用的子密钥就是可行的。

密码编码学与网络安全（中文版），第五版，p63

S-Box是非差分均匀的

$S_1(x_1 \ldots x_6)$：6位输入，4位输出

$S1$

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

考虑一个S-box的差分现象：

对于输入S盒的6比特的(x, x^*)值对，一共有多少种可能？

$64^2 = 4096$

输入值对的差分为$x' = x \oplus x^*$
差分值可能有多少种？

$2^6 = 64$

对于其4比特输出值，$y=S(x), y^*=S(x^*)$，以及$y' = y \oplus y^* = S(x) \oplus S(x^*)$
输出差分值有多少种可能？

$2^4 = 16$

8

S-Box是非差分均匀的

$S_1(x_1 \ldots x_6)$：6位输入，4位输出

输入差分f'=1111

$S1$

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

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x ⊕ f'	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
S(x)	e	4	d	1	2	f	b	8	3	a	6	c	5	9	0	7
S(x ⊕ f')	7	0	9	5	c	6	a	3	8	b	f	2	1	d	4	E
S(x) ⊕ S(x ⊕ f')	9	4	4	4	e	9	1	b	b	1	9	e	4	4	4	9

Thank you!