DES 算法详细设计

1. 算法原理概述

数据加密标准 (DES，Data Encryption Standard) 是一种使 用密钥加密的块密码，1976 年被美国国家标准局 (NBS, National Bureau of Standards，1988 年改名为 NIST) 确定为联邦信息处理标准 (FIPS)，随后在国际上获得广泛采用。

DES 是一种典型的块加密方法：它以 64 位为分组长度，64 位一组的明文作为算法的输入，通过一系列复杂的操作，输出同样 64 位长度的密文。

DES 使用加密密钥定义变换过程，因此算法认为只有持有 加密所用的密钥的用户才能解密密文。

DES 的采用 64 位密钥，但由于每 8 位中的最后 1 位用于奇偶校验，实际有效密钥长度为 56 位。密钥可以是任意的 56 位的数，且可随时改变。其中极少量的数被认为是弱密钥，但能容易地避开它们。所有的保密性依赖于密钥。

DES 算法的基本过程是换位和置换。

2. 总体结构

2.1 基本信息

设信息空间由{0, 1}组成的字符串构成，明文信息和经过 DES 加密的密文信息是 64 位的分组，密钥也是 64 位。

明文： $M = m_1 m_2 ... m_{64}, m_i \in \{0,1\}, i = 1...64$

密文： $C = c_1 c_2 ... c_{64}, c_i \in \{0,1\}, i = 1...64$

密钥： $K = k_1 k_2 ... k_{64}, k_i \in \{0,1\}, i = 1...64$

- 除去 k8, k16, …, k64 共 8 位奇偶校验位，起作用的仅为 56 位。

加密过程

$$C = E_k(M) = IP^{-1} \cdot T_{16} \cdot T_{15} \cdot ... \cdot T_1 \cdot IP(M)$$，其中 IP 为初始置换， IP^{-1} 是 IP 的逆， $T_1, T_2, ..., T_{16}$ 是一系列的迭代变换。

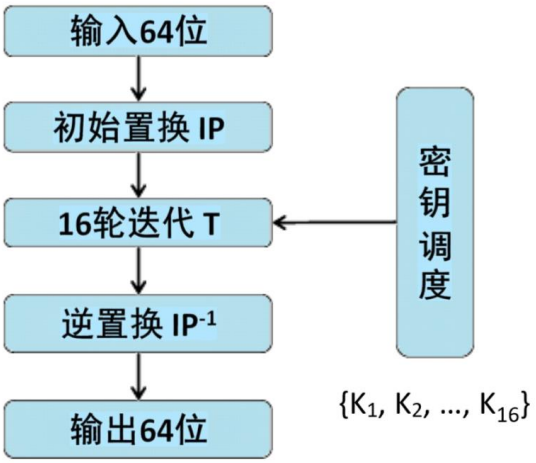
解密过程

$$M = D_k(C) = IP^{-1} \cdot T_1 \cdot T_2 \cdot ... \cdot T_{16} \cdot IP(C)$$

2.2 Feistel 结构

输入 64 位明文 M 时，密钥按 $(K_1, K_2, ..., K_{16})$ 次序调度，是加密过程。

输入 64 位密文 C 时，密钥按 $(K_{16}, K_{15}, ..., K_1)$ 次序调度，是解密过程。



1) 输入 64bit 明文

2) 初始置换 IP

给定 64 位明文块 M，通过一个固定的初始置换 IP 来重排 M 中的二进制位，得到二进制串 $M_0 = IP(M) = L_0R_0$，这里 L_0 和分别是 R_0 的前 32 位和后 32 位。下表给出 IP 置换后的下标编号序列。

IP 置换表							
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

64bit 看作线性表，用下标位置表示排列结果

3) 迭代 T

根据L_0R_0按下述规则进行16次迭代，即

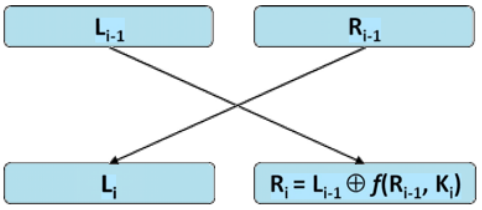
$$L_i = R_{i-1},\quad R_i = L_{i-1} \oplus f(R_{i-1}, K_i),\ i = 1 .. 16.$$

这里⊕ 是32位二进制串按位异或运算，f 是Feistel轮函数

16 个长度为 48bit 的子密钥 K_i (i = 1 .. 16) 由密钥 K 生成

16 次迭代后得到 $L_{16}R_{16}$

左右交换输出 $R_{16}L_{16}$



4) 逆置换 IP^{-1}

对迭代T输出的二进制串$R_{16}L_{16}$ 使用初始置换的逆置换IP^{-1} 得到密文C，

即：$C = IP^{-1}(R_{16}L_{16})$.

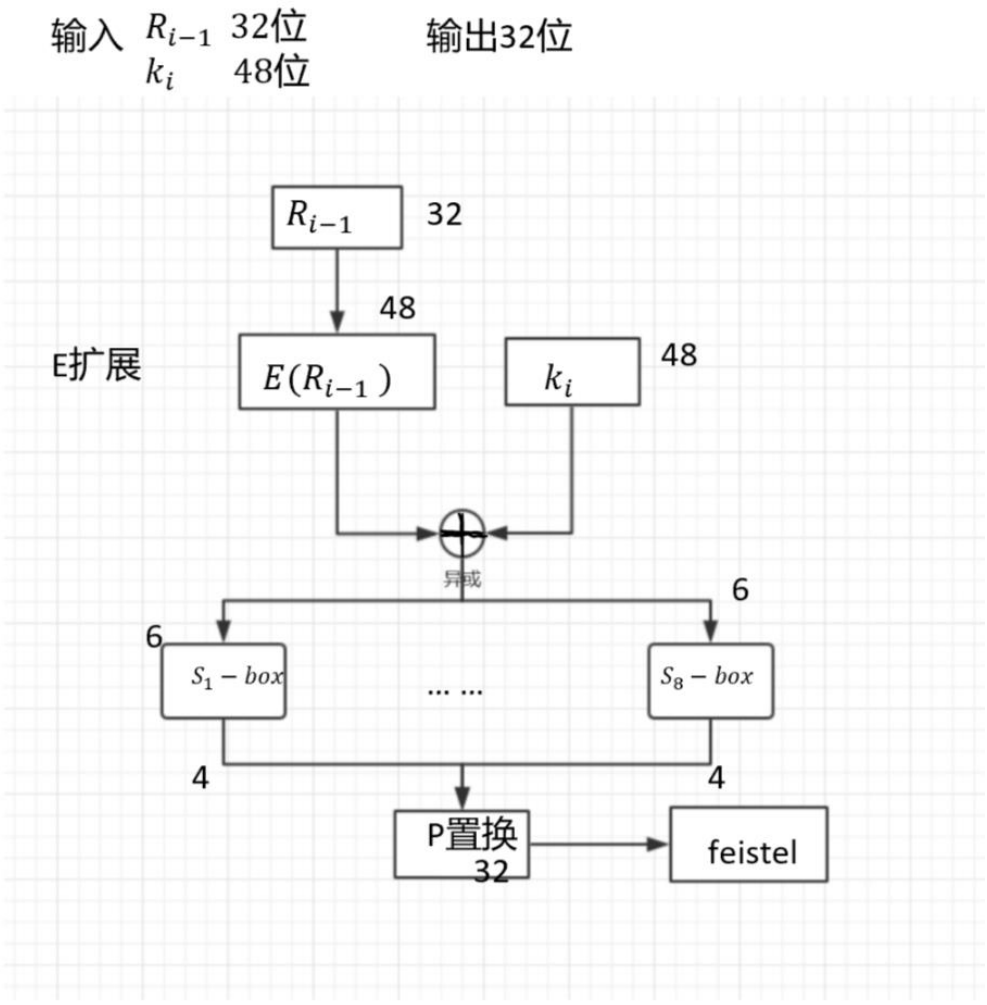
IP 置换表							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1} 置换表							
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

5) 输出64位密文C

注1： 迭代T中使用Feistel轮函数 *f*(Ri-1, Ki)详解

Feistel轮函数



(1) 将长度为32位的串 Ri-1作 E-扩展，成为48位的串 E(Ri-1)

E-扩展规则：

E-扩展规则 (比特-选择表)					
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

(2) 将 E(Ri-1) 和长度为*48*位的子密钥 *Ki* 作48位二进制串按位异或运算，Ki 由密钥 K 生成

(3) 将 (2) 得到的结果平均分成8个分组 *(*每个分组长度6位*)*，分别经过8个不同的 *S*-盒进行 6-4 转换，得到8个长度分别为4位的分组

二进制6-4 转换机制：S-盒

S-盒是一类选择函数，用于二进制6-4转换。Feistel 轮函数使用8个S-盒 $S_1, \ldots, S_8$，每个S-盒是一个4行(编号0-3)、16 列(编号0-15)的表，表中元素是一个4位二进制数的十进制表示，取值在0-15之间。

设S_i 的6位输入为$b_1b_2b_3b_4b_5b_6$，则:

$n = (b_1b_6)_{10}$ 确定行号

$m = (b_2b_3b_4b_5)_{10}$ 确定列号

$[S_i]_{n,m}$ 元素的值的二进制形式即为所要的S_i 的输出

- S-盒
  - 例1：设 S_1 的输入 $b_1b_2b_3b_4b_5b_6 = 101100$，则
    $n = (b_1b_6)_{10} = (10)_{10} = 2$，
    $m = (b_2b_3b_4b_5)_{10} = (0110)_{10} = 6$
    查表得到 $[S_1]_{2,6} = 2 = (0010)_2$ 即为所要的输出。

S_1-BOX															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	15	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_1-BOX															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	15	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2-BOX															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3-BOX															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4-BOX															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
12	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5-BOX															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6-BOX															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7-BOX															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8-BOX															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(4) 将 (3) 得到的分组结果合并得到长度为32位的串

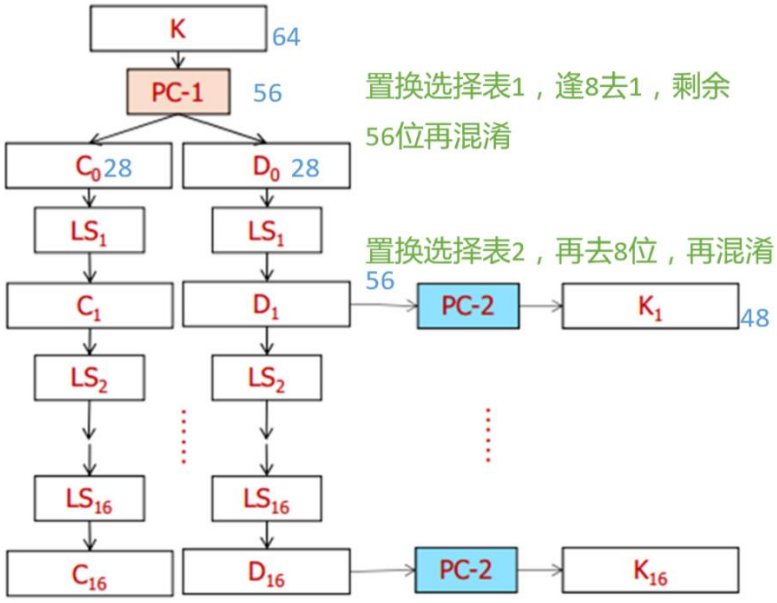
(5) 将 (4) 的结果经过 *P*-置换，得到轮函数 *f(Ri-1, Ki)* 的最终结果

P-置换

P-置换表			
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

注2：迭代T中Feistel轮函数 $f(Ri-1, Ki)$子密钥Ki生成

子密钥生成过程根据给定的64位密钥K 生成Feistel 轮函数的每轮中使用的子密钥K_i



(1) 对K 的56个非校验位实行置换PC-1，得到C_0D_0，其中C_0 和D_0 分别由PC-1置换后的前28位和后28位组成。i = 1

PC-1 置换表

C_0	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
D_0	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

(2) 计算 $C_i = LS_i(C_{i-1})$和 $D_i = LS_i(D_{i-1})$

当i =1, 2, 9, 16时，$LS_i(A)$ 表示将二进制串A循环左移一个位置；否则循环左移两个位置。

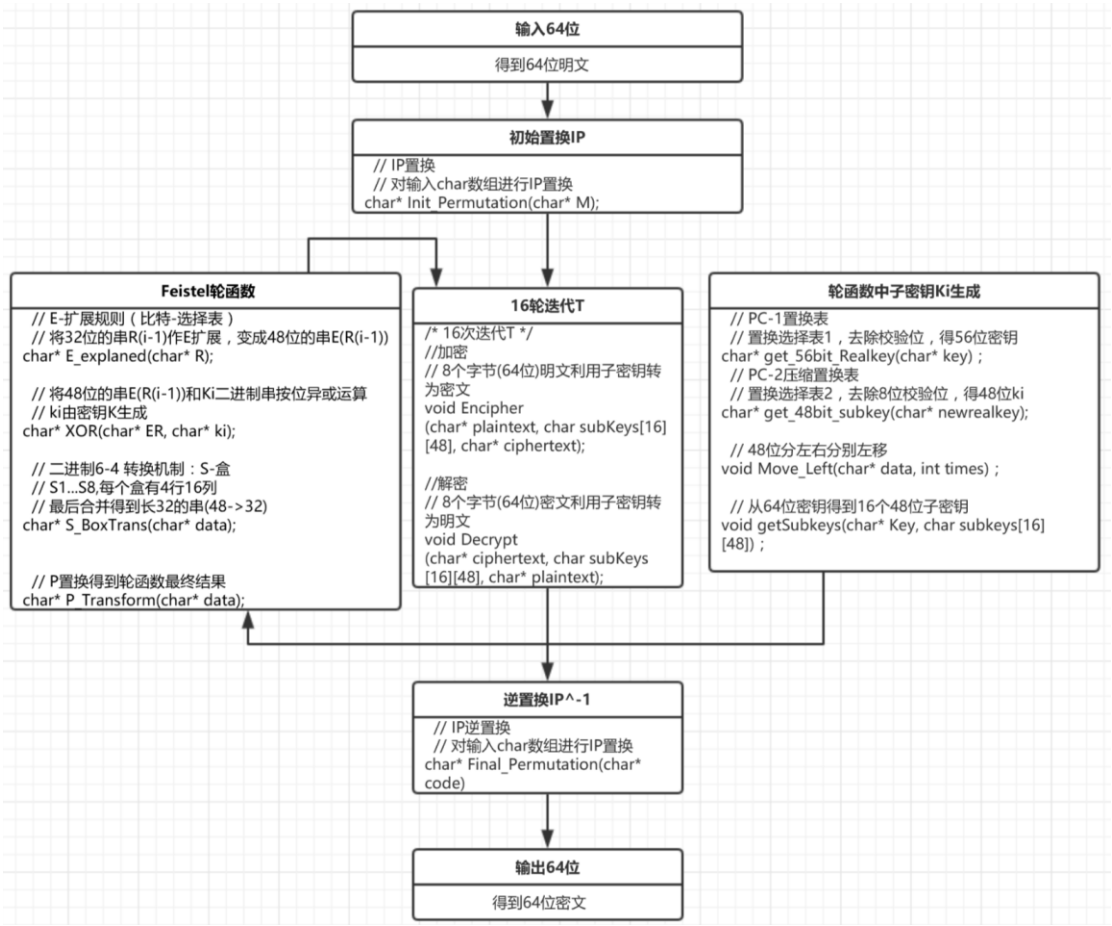
(3) 对 56 位的 C_iD_i 实行 PC-2 压缩置换，得到 48 位的 K_i 。i = i+1。

PC-2压缩置换：从56位的C_iD_i 中去掉第9, 18, 22, 25, 35, 38,43, 54位，将剩下的48位按照PC-2置换表作置换，得到K_i 。

PC-2 压缩置换表					
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

(4) 如果已经得到K_{16}，密钥调度过程结束；否则转(2)。

3. 模块分解



划分为如图几个主要模块

模块函数定义如下：

1) 初始置换 IP

```
/*---------初始置换IP---------*/
// IP置换
// 对输入char数组进行IP置换
char* Init_Permutation(char* M);
```

2) Feistel 轮函数

```
/*---------迭代T---------*/
/* ----------------------------------------1. Feistel轮函数 */
// E-扩展规则（比特-选择表）
// 将32位的串R(i-1)作E扩展，变成48位的串E(R(i-1))
char* E_explaned(char* R);
// 将48位的串E(R(i-1))和Ki二进制串按位异或运算
// ki由密钥K生成
char* XOR(char* ER, char* ki);

// 二进制6-4 转换机制：S-盒
// S1...S8,每个盒有4行16列
// 最后合并得到长32的串(48->32)
char* S_BoxTrans(char* data);


// P置换得到轮函数最终结果
char* P_Transform(char* data);
```

3) 轮函数中子密钥 Ki 生成

```
 /* ---------------------------------2. 轮函数中子密钥Ki生成 */
// PC-1置换表
// 置换选择表1，去除校验位，得56位密钥
char* get_56bit_Realkey(char* key);
// PC-2压缩置换表
// 置换选择表2，去除8位校验位，得48位ki
char* get_48bit_subkey(char* newrealkey);

// 48位分左右分别左移
void Move_Left(char* data, int times);

// 从64位密钥得到16个48位子密钥
void getSubkeys(char* Key, char subkeys[16][48]);
```

4) 逆置换 IP^-1

```
/*---------逆置换IP^-1---------*/
// IP逆置换
// 对输入char数组进行IP置换
char* Final_Permutation(char* code)

// 8位char数组转64bit
char* charTobit(char* text);
// 64bit转8位char数组
char* bitTochar(char* bits);

```

4. 数据结构

线性表是最常用且最简单的一种数据结构，它是 n 个数据元素的有限序列。使用数组存储线性表的元素，即用一组连续的存储单元依次存储线性表的数据元素。

用简单的数组来存储所用到的置换表和明文密文及中间密钥等基本数据。

1) 置换表基本存储结构

```
/* 置换表基本数据结构 */

/*---------初始置换IP---------*/
// IP置换表
int IP_table[64];


/*---------迭代T---------*/
/* ----------------------------------------1. Feistel轮函数 */
// E-扩展规则（比特-选择表）
// 将32位的串R(i-1)作E扩展，变成48位的串E(R(i-1))
int E_Table[48];

// 二进制6-4 转换机制：S-盒
// S1...S8,每个盒有4行16列
int S_box[8][4][16];

// P置换得到轮函数最终结果
int P_table[32];

 /* ---------------------------------2. 轮函数中子密钥Ki生成 */
// PC-1置换表
int PC1_table[56];

// PC-2压缩置换表
int PC2_table[56];

/*---------逆置换IP^-1---------*/
// 逆置换表IP^-1
int FP_table[64];
```

2) 其他数据存储

均采用 char 数组存储。

5. 类 C 语言算法过程（为便于观看转为 markdown 显示再截图如下）

```
/*---------初始置换IP---------*/
// IP置换
// 对输入char数组进行IP置换
char* Init_Permutation(char* M) {
    char* IP_result = new char[64];
    for (int i = 0; i < 64; i++) {
        IP_result[i] = M[IP_table[i]];
    }
    return IP_result;
}


/*---------迭代T---------*/
/* ----------------------------------------1. Feistel轮函数 */
// E-扩展规则（比特-选择表）
// 将32位的串R(i-1)作E扩展，变成48位的串E(R(i-1))
char* E_explaned(char* R) {
    char* E_result = new char[48];
    for (int i = 0; i < 48;i++) {
        E_result[i] = R[E_table[i]];
    }
    return E_result;
}
// 将48位的串E(R(i-1))和Ki二进制串按位异或运算
// ki由密钥K生成
char* XOR(char* ER, char* ki) {
    char* XOR_result = new char[48];
    for (int i = 0; i < 48; i++) {
        XOR_result[i] = ER[i]^ki[i];
    }
    return XOR_result;
}


// 二进制6-4 转换机制：S-盒
// S1...S8,每个盒有4行16列
// 最后合并得到长32的串(48->32)
char* S_BoxTrans(char* data) {
    char* result = new char[32];
    for (int i = 0; i < 8; i++) {
        int nindex = i * 6;
        int mindex = i << 2;
        // n行m列
        // n=(b1b6)10
        // m=(b2b3b4b5)10
        int n = (data[nindex] << 1) + data[nindex+5];
        int m = (data[nindex+1] << 3) + (data[nindex+2] << 2)
                + (data[nindex+3] << 1) + data[nindex+4];
        // 从sbox中选取对应值
        int num = S_box[i][n][m];

        // 转成4位二进制
        result[mindex] = (num&0x08) >> 3;
        result[mindex+1] = (num&0x04) >> 2;
        result[mindex+2] = (num&0x02) >> 1;
        result[mindex+3] = num&0x01;
    }
    return result;
}
```

```
60.
61.  // P置换得到轮函数最终结果
62.  char* P_Transform(char* data) {
63.      char* result = new char[32];
64.      for (int i = 0; i < 32; i++) {
65.          result[i] = data[P_table[i]];
66.      }
67.      return result;
68.  }
69.
70.
71.
74.  // 置换选择表1，去除8位校验位，得56位密钥
75.  char* get_56bit_Realkey(char* key) {
76.      char* realkey = new char[56];
77.      for (int i = 0; i < 64; i++) {
78.           // 逢8去1
79.          if ((i+1)%8 != 0) {
80.              realkey = key[PC1_table[i]];
81.          }
82.      }
83.      return realkey;
84.  }
85.
86.  // 置换选择表2，去除8位校验位，得48位ki
87.  char* get_48bit_subkey(char* newrealkey) {
88.      char* subkey = new char[48];
89.      for (int i = 0; i < 56; i++) {
90.          // 去除9，18，22，25，35，38，43，54位
91.          if (i != 9 && i != 18 && i != 22 && i != 25
92.           && i != 35 && i != 38 && i != 43 && i != 54) {
93.              subkey = newrealkey[PC2_table[i]];
94.          }
95.      }
96.      return subkey;
97.  }
98.
99.
100. // i=1,2,9,16时，循环左移一个位置，否则循环左移两个位置
101. int movetoleft[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};
102. // 48位分左右分别左移
103. void Move_Left(char* data, int times) {
104.     char* savedata = new char[56];
105.     // 保存移位数据
106.     memcpy(savedata, data, times);
107.     memcpy(savedata+times, data+28, times);
108.
109.     // 前28位
110.     memcpy(data, data+times, 28-times);
111.     memcpy(data+28-times, savedata, times);
112.     // 后28位
113.     memcpy(data+28, data+28+times, 28-times);
114.     memcpy(data+56-times, savedata+times, times);
115.
116.
117. }
118.
```

```
119.
120.   // 从64位密钥得到16个子密钥
121.   void getSubkeys(char* Key, char subkeys[16][48]) {
122.       char* realkey = new char[56];
123.       // PC-1置换
124.       realkey = get_56bit_Realkey(key);
125.       // 16次循环迭代
126.       for (int i = 0; i < 16; i++) {
127.           // 左移movetoleft[i]位
128.           Move_Left(realkey, movetoleft[i]);
129.           // // PC-2置换，得到子密钥
130.           subkeys[i] = get_48bit_subkey(realkey);
131.       }
132.   }
133.
134.
135.   // IP逆置换
136.   // 对输入char数组进行IP置换
137.   char* Final_Permutation(char* code) {
138.       char* FP_result = new char[64];
139.       for (int i = 0; i < 64; i++) {
140.           FP_result[i] = code[FP_table[i]];
141.       }
142.       return FP_result;
143.   }
144.
```

```
146.   char* charTobit(char* text);
147.   char* bitTochar(char* bits);
148.   //加密
149.   // 8个字节(64位)明文利用子密钥转为密文
150.   void Encipher(char* plaintext, char subKeys[16][48], char* ciphertext) {
151.       char* plain = new char[64];
152.       char* afterIP = new char[64];
153.       char* Left = new char[48];
154.       char* Right = new char[48];
155.       char* eRight = new char[48];
156.       char* xor_result = new char[48];
157.       char* s_result = new char[32];
158.       char* f_result = new char[32];
159.       char* Ri = new char[32];
160.       char* afterFP = new char[64];
161.
162.       plain = charTobit(plaintext);
163.       //初始IP
164.       afterIP = Init_Permutation(plain);
165.
```

```
    //16轮迭代
    for(int i = 0; i < 16; i++){
        // 左半部分
        memcpy(Left, afterIP, 32);
        // 右半部分
        memcpy(Right,afterIP+32,32);
        //右半部分E扩展置换，32位->48位
        eRight = E_explaned(Right);
        //将右半部分与子密钥进行异或操作
        xor_result = XOR(eRight, subkeys[i]);
        //异或结果进入Sbox，输出32位结果
        s_result = S_BoxTrans(xor_result);
        //P置换得到轮函数最终结果
        f_result = P_Transform()
        //明文左半部分与轮函数结果进行异或
        Ri = XOR(Left,f_result);
        // Li和Ri交换
        if(i < 15){
            Swap(Right,Ri);
        }
        memcpy(afterIP, Right, 32);
        memcpy(afterIP+32, Ri, 32);
    }
    //逆初始置换（IP^-1置换）
    afterFP = Final_Permutation(afterIP);
    ciphertext = bitTochar(afterFP);
}
//解密
// 8个字节(64位)密文利用子密钥转为明文
void Decrypt(char* ciphertext, char subKeys[16][48], char* plaintext) {
    char* cipher = new char[64];
    char* afterIP = new char[64];
    char* Left = new char[48];
    char* Right = new char[48];
    char* eRight = new char[48];
    char* xor_result = new char[48];
    char* s_result = new char[32];
    char* f_result = new char[32];
    char* Ri = new char[32];
    char* afterFP = new char[64];

    cipher = charTobit(ciphertext);
    //初始IP
    afterIP = Init_Permutation(cipher);

    //16轮迭代
    for(int i = 15; i >= 0; i--){
        // 左半部分
        memcpy(Left, afterIP, 32);
        // 右半部分
        memcpy(Right,afterIP+32,32);
        //右半部分E扩展置换，32位->48位
        eRight = E_explaned(Right);


        //将右半部分与子密钥进行异或操作
        xor_result = XOR(eRight, subkeys[i]);
        //异或结果进入Sbox，输出32位结果
        s_result = S_BoxTrans(xor_result);
        //P置换得到轮函数最终结果
        f_result = P_Transform()
        //明文左半部分与轮函数结果进行异或
        Ri = XOR(Left,f_result);
        // Li和Ri交换
        if(i < 15){
            Swap(Right,Ri);
        }
        memcpy(afterIP, Right, 32);
        memcpy(afterIP+32, Ri, 32);
    }
    //逆初始置换（IP^-1置换）
    afterFP = Final_Permutation(afterIP);
    plaintext = bitTochar(afterFP);
}
```