中山大學
SUN YAT-SEN UNIVERSITY

《信息安全技术》

实验报告

（Assignment 1：DES 算法）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 软件工程 1 班
学 生 姓 名 ：	陆记
学　　号	17343080
时　　间 ：	2019　年　10　月　12　日

一、 算法原理概述【参考网上博客】

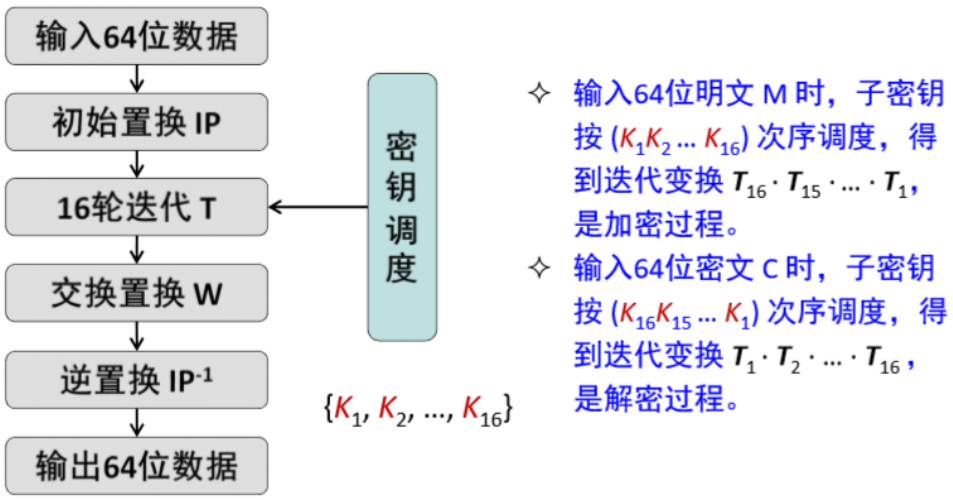
DES 是一种典型的块加密方法：它以 64 位为分组长度，64 位一组的明文作为算法的输入，通过一系列复杂的操作，输出同样 64 位长度的密文。DES 使用加密密钥定义变换过程，因此算法认为只有持有加密所用的密钥的用户才能解密密文。

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。

二、 总体结构

- DES 算法的总体结构 — *Feistel* 结构



三、 模块分解

DES 加密

- 加密过程
  - ✧ $C = E_k(M) = IP^{-1} \cdot W \cdot T_{16} \cdot T_{15} \cdot \ldots \cdot T_1 \cdot IP(M)$.
    - ○ M 为算法输入的64位明文块；
    - ○ E_k 描述以 K 为密钥的加密函数，由连续的过程复合构成；
    - ○ IP 为64位初始置换；
    - ○ $T_1, T_2, \ldots, T_{16}$ 是一系列的迭代变换；
    - ○ W 为64位置换，将输入的高32位和低32位交换后输出；
    - ○ IP^{-1} 是 IP 的逆置换；
    - ○ C 为算法输出的64位密文块。

该模块的 C++代码如下：

```
/* DES加密，输入64位明文，返回64位密文 */
bitset<64> encrypt(bitset<64> & clear) {
    bitset<64> cipher; //密文
    bitset<64> Rep_clear; //IP置换后的明文
    bitset<32> L; //明文IP置换后前32位
    bitset<32> R; //明文IP置换后后32位
    bitset<32> L_; //迭代过程的下一轮L
    // 第一步：IP置换
    for (int i = 0; i < 64; ++i)
        Rep_clear[i] = clear[IP[i] - 1]; //因为IP置换表中是从1开始，所以需要减1

    // 第二步：16轮迭代T置换得到L16，R16
    for (int i = 0; i < 32; ++i) //得到L0
        L[i] = Rep_clear[i];
    for (int i = 32; i < 64; ++i) //得到R0
        R[i - 32] = Rep_clear[i];

    for (int r = 0; r < 16; ++r) { //根据迭代规则进行16轮迭代得到L16和R16
        L_ = R;
        R = L ^ f(R, subkey[r]);
        R = L_;
    }
    // 第四步：W置换生成R16L16，即前32位为R16，后32位为L16
    for (int i = 0; i < 32; ++i)
        cipher[i] = R[i];
    for (int i = 32; i < 64; ++i)
        cipher[i] = L[i - 32];

    // 第五步：IP-1置换
    Rep_clear = cipher;
    for (int i = 0; i < 64; ++i)
        cipher[i] = Rep_clear[IP_1[i] - 1];

    return cipher;
}
```

具体步骤：

根据加密过程的公式结合 Feistel 结构，从右向左执行：

1、首先进行 IP 的初始置换

将给定的 64 位明文块 M 通过一个固定的 IP 置换表对 M 的二进制位进行重新排列，得到 $M_0=IP(M)=L_0R_O$, 其中 L_0 和 R_O 分别表示 M_0 的前后 32 位。

IP 置换表如下（表中的数字代表二进制位位置的编号 1-64）：

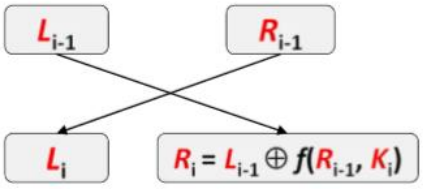
IP 置换表 (64位)							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

2、然后进行 16 轮的迭代 T 变换

迭代的规则如下：

- 根据 L_0R_0 按下述规则进行16次迭代 $T_1, T_2, ..., T_{16}$，即

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



- 这里 ⊕ 是32位二进制串按位异或运算，f 是输出32位的 *Feistel* 轮函数；
- 16个长度为48位的子密钥 K_i (i = 1 .. 16) 由密钥 K 生成；
- 16次迭代后得到 $L_{16}R_{16}$；

- 最后左右交换输出 $R_{16}L_{16}$。

其中 R_i 的得到需要调用到 f（R_{i-1}, K_i）Feistel 轮函数，其执行步骤如下：

(1) 将长度为32位的串 R_{i-1} 作 E-扩展，成为48位的串 $E(R_{i-1})$；
(2) 将 $E(R_{i-1})$ 和长度为48位的子密钥 K_i 作48位二进制串按位异或运算，K_i 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。各个分组分别经过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；
(4) 将 (3) 得到的分组结果顺序连接得到长度为32位的串；
(5) 将 (4) 的结果经过 P-置换，得到的结果作为轮函数 $f(R_{i-1}, K_i)$ 的最终32位输出。

C++代码如下：

```
/* Feistel轮函数f,输入32位串，和48位子密钥，返回一个32位的串 */
bitset<32> f(bitset<32> R, bitset<48> k) {
    bitset<48> ext_R; //扩展后的32位串
    // 第一步：E-扩展置换，将输入的32位串扩展至48位
    for (int i = 0; i < 48; ++i)
        ext_R[47 - i] = R[32 - E_ext[i]];

    // 第二步：将扩展后的R与子密钥k异或
    ext_R = ext_R ^ k;

    // 第三步：S-BOX压缩转换，将48位的串转换为36位串
    bitset<32> f_out; //f函数的最终输出

    int b = 0; //表示f_out每一位的序号
    for (int i = 0; i < 48; i = i + 6) { //循环将48位串分为8组，每组6位
        int row = ext_R[i] * 2 + ext_R[i + 5]; //用每组的第1位和第6位组成的二进制数转换成的十进制数作为行数，直接第一位的数×2加第六位数即可
        int col = ext_R[i + 1] * 8 + ext_R[i + 2] * 4 + ext_R[i + 3] * 2 + ext_R[i + 4]; //用每组的2、3、4、5位计算列数
        int num = S_BOX[i / 6][row][col]; //找出相应的S-BOX对应row和col的十进制数
        bitset<4> binary(num); //将十进制的num转化为4位的二进制数
        //将4位二进制数分别赋值给f_out对应的位
        f_out[b] = binary[0];
        f_out[b + 1] = binary[1];
        f_out[b + 2] = binary[2];
        f_out[b + 3] = binary[3];
        b += 4;
    }

    // 第四步：P-置换，得到f函数最后的输出
    bitset<32> tmp = f_out;
    for (int i = 0; i < 32; ++i)
        f_out[i] = tmp[P[i] - 1]; //因为P置换表是从1开始，所以要减1

    return f_out;
}
```

下面将逐步解释：

1）先将上一轮的 R 做 E-扩展（即将原来的 32 位串扩展至 48 位）得到 $E(R_{i-1})$，E-扩展规则如下，其中表中的数字代表 R_{i-1} 中的二进制数的位置，扩展方式为：前后各加一列（每列 8 位），前列为初始末列下移一位，后列为初始初列上移一位：

2）将 $E(R_{i-1})$与给定密钥 K 生成的 16 个 48 的子密钥进行抑或运算（子密钥的生成会在后面单独解释）

3）将第 2）步得到的抑或结果按每组 6 位分成 8 个小组（按照二进制位次序），然后通过 S-盒对每个小组依次进行 6-4 位转换，因此有 8 个 S-盒。S-盒是一类选择函数，每个 S-盒是一个 4 行（编号 0-3）、16 列（编号 0-15）的表，表中的元素为 0-15 之间的十进制数（可以用 4 位二进制数表示）。

S-盒 6-4 位的转换规则：假设输入的二进制数为 $b_1b_2b_3b_4b_5b_6$，则此二进制数对应 S-盒中的行号为 $n = (b_1b_6)_{10}$，即第一位和最后一位组成的二进制数的十进制（正好范围为 0-3），对应 S-盒中的列号为 $m = (b_2b_3b_4b_5)_{10}$，与行号同理，正好范围为 0-15。然后将 S-盒中行号列号分别为 n、m 的元素 $[S_i]_{n,m}$ 转换为 4 位二进制数，这就是 6-4 转换的结果。

下面是一个老师给的例子：

◌ 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

8 个 S-盒如下（只是目前公开的 S-盒，并不能证明是数学上最优的，我们国家也有自己的 S-盒）

✧ S-盒 S_1 - S_4

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-盒 S_5 - S_8

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）将第 3）步经过 S-盒 6-4 变换后的 8 个分组依次连接得到新的长度为 32 位的串。这样相当于通过 E-扩展和 S-盒实现了一次二进制位数由 32->48->32 的加密过程。

5）最后将第 4）步的结果通过 P-置换，得到的 32 位串作为 f（R_{i-1}, K_i）的结果。其中，P-置换表如下，表中的数字代表二进制位的序号（1-32）：

1

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

3、接着进行 W 置换

W 置换即将第 2 步 T 迭代最终得到的 $L_{16}R_{16}$ 交换得到 $R_{16}L_{16}$。

4、最后进行 IP^{-1} 逆置换

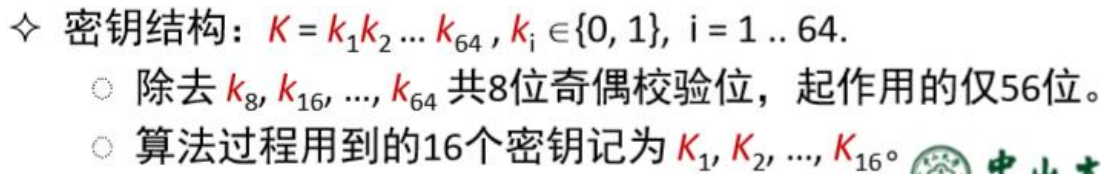
对 W 置换得到的结果 $R_{16}L_{16}$ 进行一个 IP^{-1} 置换就可以得到需要的密文，即：$C=IP^{-1}(R_{16}L_{16})$.置换规则和 IP 置换一致，IP^{-1} 置换表如下：

IP^{-1} 置换表 （64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

下面将介绍给定密钥 K 的 16 个子密钥的生成过程：

子密钥生成

子密钥生成是根据给定的密钥 K,生成 16 个 48 位的子密钥 K1-K16，供 Feistel 轮函数使用。其中密钥 K 的结构如下：



C++代码如下：

```
/* 子密钥生成 */
void sub_keys() {
    bitset<56> uncheckd_key; //去掉校验位的key
    bitset<28> C; //uncheckd_key的前28位
    bitset<28> D; //uncheckd_key的后28位
    bitset<48> sub_key; //48位子密钥
    //第一步，对key的非校验委进行PC-1置换
    for (int i = 0; i < 56; ++i)
        uncheckd_key[55 - i] = key[64 - PC_1[i]];

    //生成16个48位子密钥，保存在subkeys[]中
    for (int r = 0; r < 16; ++r) {

        for (int i = 0; i < 28; ++i)
            C[i] = uncheckd_key[i];
        for (int i = 28; i < 56; ++i)
            D[i - 28] = uncheckd_key[i];

        //第二步，根据shift_left数组规定的规则左移
        C = leftshift(C, shift_left[r]);
        D = leftshift(D, shift_left[r]);

        //第三步，PC-2压缩置换，将56位uncheckd_key压缩成48位形成子密钥
        for (int i = 0; i < 28; ++i)
            uncheckd_key[i] = C[i];
        for (int i = 28; i < 56; ++i)
            uncheckd_key[i] = D[i - 28];

        for (int i = 0; i < 48; ++i)
            sub_key[i] = uncheckd_key[PC_2[i] - 1];

        subkey[r] = sub_key;
    }
}
```

```
/* 子密钥生成过程中的移位函数，输入28位的串以及移位的位数，输出28位的新串 */
bitset<28> leftshift(bitset<28> k, int shift_digit) {
    bitset<28> tmp = k;
    for (int i = 27; i >= 0; --i) {
        if (i - shift_digit < 0)
            k[i] = tmp[i - shift_digit + 28];
        else
            k[i] = tmp[i - shift_digit];
    }
    return k;
}
```

生成步骤如下：

1）对 K 的 56 个非校验位进行 PC-1 置换，得到 C_0D_0，其中 C_0、D_0 分别表示置换后的前后 28 位。PC-1 置换表如下（表中数字代表二进制数的序号，可以看到没有 8、16…等 8 个校验位）：

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 $C_i = LS_i(C_{i-1})$ 以及 $D_i = LS_i(D_{i-1})$，其中 $LS_i(C)$ 函数的规则为：若 $i=1, 2, 9, 16$ 时，将二进制串 C 循环左移一个位置；否则循环左移两个位置。

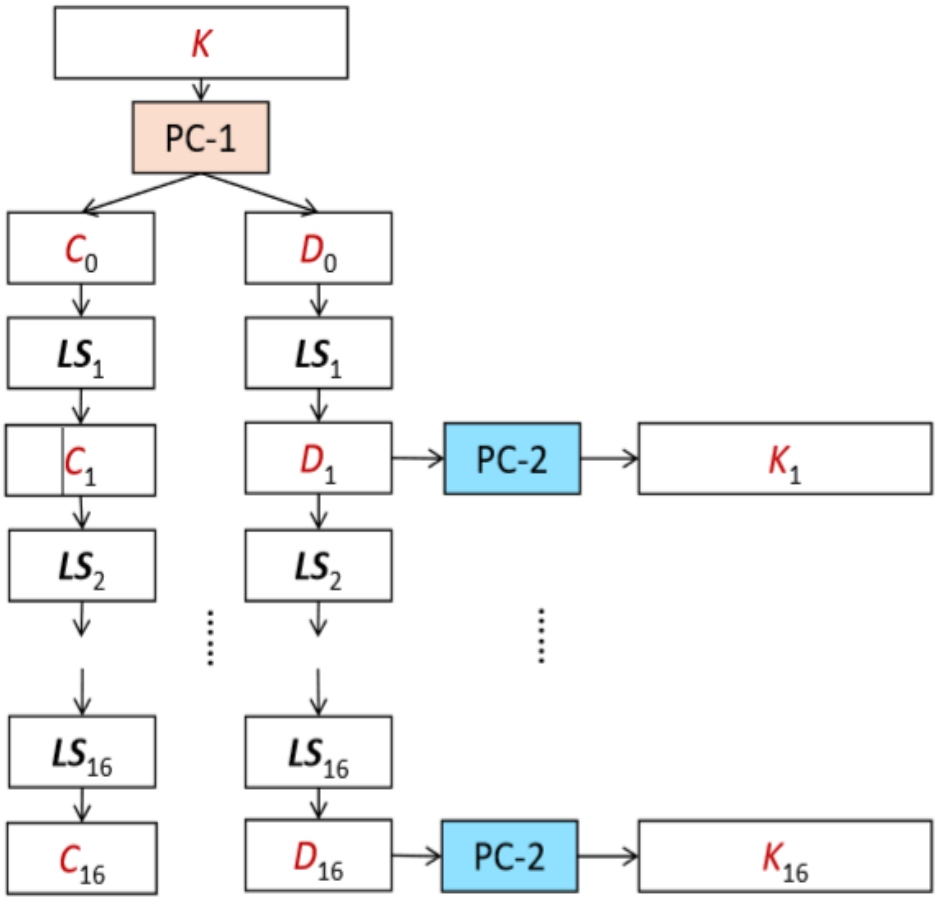
3）对第 2）步得到的 16 个 56 位 C_iD_i 进行 PC-2 压缩置换，得到 48 位的子密钥 K_i。得到 K_{16} 之后置换结束。

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

PC-2 压缩置换表如下：

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

子密钥生成的流程图如下：



DES 解密

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

因为 DES 加密中的迭代、置换、抑或和循环移位过程都是可逆的，所以 DES 解密可使用相同的算法和密钥。如上面的公式，不同点只是在于解密过程中输入的是密文，输出的是明文，同时在 T 迭代时，需要从加密时的子密钥从 K_1-K_{16} 的调度顺序颠倒变为 K_{16}-K_1 的顺序。至于公式的证明，可以直接根据 T 迭代的规则进行即可，因为相对比较容易，所以就不做过多阐述了。

C++代码如下：

```
/* DES解密，输入64位密文，返回64位明文，与加密过程调度子密钥的顺序相反 */
bitset<64> decrypt(bitset<64> & cipher) {
    bitset<64> clear; //明文
    bitset<64> Rep_cipher; //IP置换后的密文
    bitset<32> L; //明文IP置换后前32位
    bitset<32> R; //明文IP置换后后32位
    bitset<32> L_; //迭代过程的下一轮L
    // 第一步：IP置换
    for (int i = 0; i < 64; ++i)
        Rep_cipher[i] = cipher[IP[i] - 1]; //因为IP置换表中是从1开始，所以需要减1

    // 第二步：16轮迭代T置换得到L16，R16
    for (int i = 0; i < 32; ++i) //得到L0
        L[i] = Rep_cipher[i];
    for (int i = 32; i < 64; ++i) //得到R0
        R[i - 32] = Rep_cipher[i];

    for (int r = 0; r < 16; ++r) { //根据迭代规则进行16轮迭代得到L16和R16，与加密调度顺序相反，从子密钥K16开始调度
        L_ = R;
        R = L ^ f(R, subkey[15 - r]);
        R = L_;
    }
    // 第四步：W置换生成R16L16，即前32位为R16，后32位为L16
    for (int i = 0; i < 32; ++i)
        clear[i] = R[i];
    for (int i = 32; i < 64; ++i)
        clear[i] = L[i - 32];

    // 第五步：IP-1置换
    Rep_cipher = clear;
    for (int i = 0; i < 64; ++i)
        clear[i] = Rep_cipher[IP_1[i] - 1];

    return clear;
}
```

四、 数据结构

基本数据结构：64 位密钥、48 位子密钥组、IP 置换表以及 IP 逆置换表

```
bitset<64> key;                 // 64位密钥
bitset<48> subkey[16];          // 存放16轮子密钥

// IP置换表
int 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 };

//IP逆置换表
int IP_1[] = { 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 };
```

子密钥生成过程用到的数据结构：PC-1 置换表、PC-2 置换表以及用于左移位的移位数组

```
/* 子密钥生成过程 */
// 密钥K非校验位PC-1置换表
int PC_1[] = { 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 };

//子密钥生成时的循环左移位数，第1、2、9、16个子密钥左移一位，其他子密钥移两位
int shift_left[] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

// PC-2压缩置换表，将循环左移后的16个56位密钥压缩成48位子密钥
int PC_2[] = { 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 };
```

Feistel 轮函数用到的数据结构：E-扩展表、S-盒以及 P 置换表

```
/* Feistel轮函数 */
//E-扩展表，将32位L\R扩展至48位
int E_ext[] = { 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 };
```

```
//8个s盒，实现6->4位的压缩置换
int S_BOX[8][4][16] = {
        //S1-BOX
        {
            { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
            { 0, 15, 7, 4, 14, 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 }
        },
        //S2-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 }
        },
        //S3-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 }
        },
        //S4-BOX
        {
            { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
            { 13, 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 }
        },
```

```
        //S5-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 }
        },
        //S6-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 }
        },
        //S7-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 }
        },
        //S8-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 }
        }
};
```

```
// P置换表，将S-BOX压缩后的串进行置换作为最终的轮函数结果
int P[] = { 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 };
```

五、 C 语言源代码

Github 地址：https://github.com/luji17343080/Information-security-technology/tree/master

六、 编译运行结果

控制台结果：

加密文件.txt：

解密文件.txt：