# DES 密码实验

## 一、实验要求：

1、了解分组密码的起源与涵义。

2、掌握 DES 密码的加解密原理。

3、用 Visual C++实现 DES 密码程序并输出结果。

## 二、实验内容：

1、1949 年，Shannon 发表了《保密系统的通信理论》，奠定了现代密码学的基础。他还指出混淆和扩散是设计密码体制的两种基本方法。扩散指的是让明文中的每一位影响密文中的许多位，混淆指的是将密文与密钥之间的统计关系变得尽可能复杂。而分组密码的设计基础正是扩散和混淆。在分组密码中，明文序列被分成长度为 n 的元组，每组分别在密钥的控制下经过一系列复杂的变换，生成长度也是 n 的密文元组，再通过一定的方式连接成密文序列。

2、DES 是美国联邦信息处理标准(FIPS)于 1977 年公开的分组密码算法，它的设计基于 Feistel 对称网络以及精心设计的 S 盒，在提出前已经进行了大量的密码分析，足以保证在当时计算条件下的安全性。不过，随着计算能力的飞速发展，现如今 DES 已经能用密钥穷举方式破解。虽然现在主流的分组密码是 AES，但 DES 的设计原理仍有重要参考价值。在本实验中，为简便起见，就限定 DES 密码的明文、密文、密钥均为 64bit，具体描述如下：

**明文** m 是 64bit 序列。

**初始密钥** K 是 64 bit 序列(含 8 个奇偶校验 bit)。

**子密钥** $K_1, K_2 \ldots K_{16}$ 均是 48 bit 序列。

**轮变换函数** f (A,J)：输入 A(32 bit 序列), J(48 bit 序列)，输出 32 bit 序列。

**密文** c 是 64 bit 序列。

**1) 子密钥生成：**

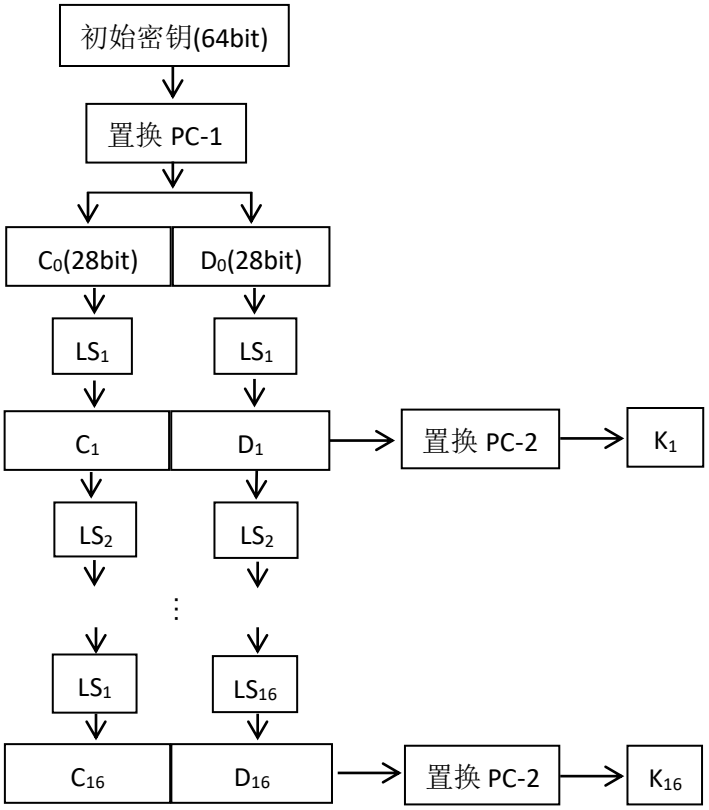
输入初始密钥，生成 16 轮子密钥 $K_1, K_2 \ldots K_{16}$。

初始密钥(64bit)经过置换 PC-1，去掉了 8 个奇偶校验位，留下 56 bit，接着分成两个 28 bit 的分组 $C_0$ 与 $D_0$，再分别经过一个循环左移函数 $LS_1$，得到 $C_1$ 与 $D_1$，连成 56 bit 数据，然后经过置换 PC-2，输出子密钥 $K_1$，以此类推产生 $K_2$ 至 $K_{16}$。

注意：置换 PC-1、PC-2 会在下文中会具体给出；

函数 $LS_i$ 为向左循环移动 { 1 个位置(i=1,2,9,16)
2 个位置(i=3,4,5,6,7,8,10,11,12,13,14,15)。

详见下图：



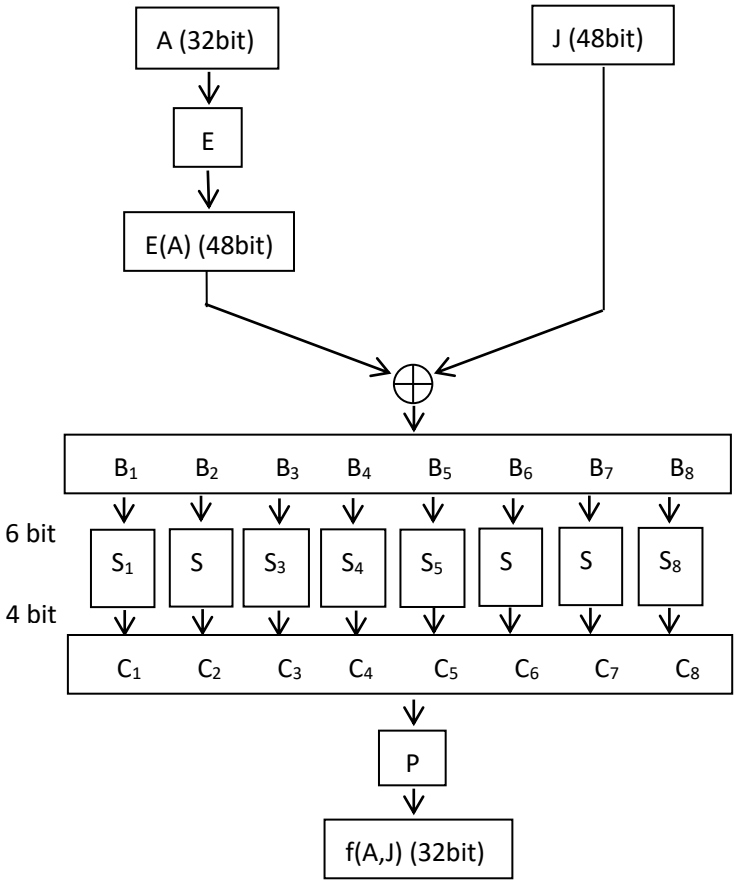

**2) 轮变换函数 f：**

f 是 DES 加解密中每一轮的核心运算，输入 A(32 bit), J(48 bit)，输出 32 bit。

将 A 做一个扩展运算 E，变成 48 bit，记为 E(A)。计算 B=E(A)⊕J，将 B 分为 8 组 $B_1 \cdots B_8$，每组 $B_i$ 为 6 bit，通过相应的 S 盒 $S_i$，输出 $C_i$ 为 4 bit，将所有 $C_i$ 连成 C(32 bit)，再通过置换 P，得到最后的输出 f(A,J)，为 32 bit。在加密或解密的第 i 轮，$A = R_{i-1}$，$J = K_i$。

注意：每个 S 盒 $S_i$ 是 4×16 的矩阵，输入 $b_0b_1b_2b_3b_4b_5$ (6 bit)，令 L 是 $b_0b_5$ 对应的十进制数，n 是 $b_1b_2b_3b_4$ 对应的十进制数，输出矩阵中第 L 行 n 列所对应数的二进制表示。

详见下图：




**3) 加/解密：**

DES 的加密和解密几乎一样，不同之处在于加密时输入是明文，子密钥使用顺序为 $K_1K_2\cdots K_{16}$；解密时输入是密文，子密钥使用顺序为 $K_{16}K_{15}\cdots K_1$。

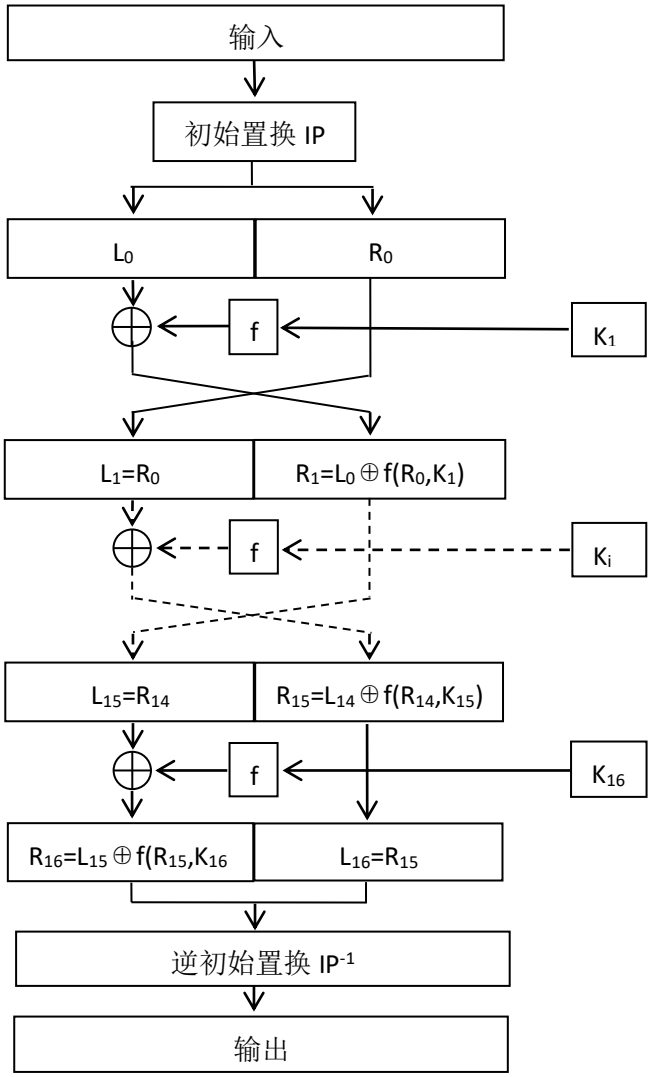
以加密为例，输入明文分组 64 bit，先进行一次初始置换 IP，对置换后的数据 $X_0$ 分成左右两半 $L_0$ 与 $R_0$，根据第一个子密钥 $K_1$ 对 $R_0$ 实行轮变换 $f(R_0, K_1)$，将结果与 $L_0$ 作逐位异或运算，得到的结果成为下一轮的 $R_1$，$R_0$ 则成为下一轮的 $L_1$。如此循环 16 次，最后得到 $L_{16}$ 与 $R_{16}$。可用下列公式简洁地表示：

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$
$$L_i = R_{i-1}, \qquad i = 1,2,\cdots,16$$

最后对 64 bit 数据 $R_{16}$, $L_{16}$ 实行 IP 的逆置换 $IP^{-1}$，即得密文分组。

注意：第 16 轮变换后并未交换 $L_{16}$ 与 $R_{16}$，而直接将 $R_{16}$, $L_{16}$ 作为 $IP^{-1}$ 的输入。

详见下图：




3、使用 Visual C++编写程序，实现 DES 密码及输出界面，

主要步骤：

1) 新建一个空项目，取名 des_test。

2) 在左边的解决方案资源管理器中添加 cpp 文件，取名为 des_test.cpp。

3) 在 des_test.cpp 中先写入

```
#include<iostream>
using namespace std;
static bool SubKey[16][48];        //定义子密钥，使得所有函数都能调用
static const unsigned char IP_Table[64] = {                    //定义 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};
static const unsigned char IPInv_Table[64] = {          //定义 IP 逆置换表
    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};
static const unsigned char E_Table[48] = {              //定义 E 扩展表
    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};
static const unsigned char P_Table[32] = {              //定义 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};
static const unsigned char PC1_Table[56] = {            //定义 PC1 置换表
    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};
static const unsigned char PC2_Table[48] = {            //定义 PC2 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};
```

```
static const unsigned char LS_Table[16] = {                 //定义左移位数表
    1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};
static unsigned char S_Box[8][4][16] = {                    //S 盒
    //S1
    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
    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
    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
    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
    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
    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

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

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) 编写各个模块函数，例如：

void ByteToBit(bool* Out, const unsigned char* In, int bits);

//将字符数组 In 转化为 bool 数组 Out，bits 控制转换 bit 数

void HalfByteToBit(bool* Out, const unsigned char* In, int bits);

//将半字符数组 In(4 bit)转化为 bool 数组 Out

void BitToByte(unsigned char* Out, const bool* In, int bits);

//将 bool 数组 In 转化为字符数组 Out

void Transform(bool* Out, bool* In, const unsigned char* Table, int len);

//利用置换表 Table 将 bool 数组 In 转换成长度为 len 的 bool 数组 Out

void RotateL(bool* In, int len, int loop);

//对长度为 len 的 bool 数组 In 循环左移 loop 位

void Des_SetSubKey(unsigned char Key[8]);

//利用初始密钥 Key 生成 16 轮子密钥 SubKey，

注意前文已定义 static bool SubKey[16][48];

void Xor(bool* InA, bool* InB, int len);

//将 bool 数组 InB 与 InA 逐位做异或，结果存入 InA

void S_Func(bool Out[32], bool In[48]);

//利用 S 盒 S_Box 将 bool 数组 In 转换为 bool 数组 Out

void F_Func(bool In[32], bool Ki[48]);

//利用子密钥 Ki 对 bool 数组 In 做轮变换函数，结果仍存入 In

void Des_Run(unsigned char Out[8], unsigned char In[8], bool flag);

//Des 加解密：输入字符数组 In，输出字符数组 Out，

flag=1 时代表加密，flag=0 时代表解密。

5) 编写主函数，调试程序，目的是测试各个函数编写正确。

6) 所有函数测试都正确后，新建一个对话框项目 DES，包含明文框、密钥框、密文框、解密文框等，以及加密，解密，清空按钮，将以上的置换表、S 盒以及所有子函数声明及实现代码全部复制到 DESDlg.cpp 中的合适位置，再编写按钮事件，实现加解密功能。

提示点：

1) 输入输出尽量使用 cin、cout 这两个函数，编写函数时多输出中间变量查看结果。

2) 函数 memcpy(void *dest, const void *src, size_t n) 用于将源地址 src 处 n 个单位数据拷贝至目的地址 dest 处，适用于 bool 数组的拷贝。

3) 部分函数参考代码

```
void Transform(bool* Out, bool* In, const unsigned char* Table, int len)
{
    bool Temp[256];                    //临时数组长度定义成足够大的即可
    for(int i=0;i<len;i++)
        Temp[i] = In[Table[i]-1];
    memcpy(Out, Temp, len);
}
 void ByteToBit(bool* Out, const unsigned char* In, int bits)
 {   for (int i=0;i<bits;i++)
        Out[i] = (In[i/8]>>(7-i%8)) & 1;   }
 void HalfByteToBit(bool* Out, const unsigned char* In, int bits)
 {   for (int i=0;i<bits;i++)
        Out[i] = (In[i/4]>>(3-i%4)) & 1;   }
```

void BitToByte(unsigned char* Out, const bool* In, int bits)

{ for(int i=0;i<bits;i++)

Out[i/8] = (In[i]<<(7-i%8)) + Out[i/8]; }

DES 密码程序代码如下：(所有模块函数代码以及按钮事件代码)

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from PyQt5 import QtCore, QtGui, QtWidgets
import sys
from Ui_DES import Ui_MainWindow
from ctypes import memset

class Min(QtWidgets.QMainWindow,Ui_MainWindow):
    def __init__(self,parent=None): #ui 部分
        super().__init__()
        self.setupUi(self)
        self.jm.clicked.connect(self.func1)
        self.jm2.clicked.connect(self.func2)
        self.qk.clicked.connect(self.clear)
        self.m1=[]
        self.c=[]
        self.IP_Table=[ 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,]
        self.IPInv_Table=[  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]
        self.E_Table = [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]
        self.P_Table = [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]
        self.PC1_Table =[   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]
        self.PC2_Table = [  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]
        self.LS_Table = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]
        self.S_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]
                        ],
                        [
                            [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]
                        ],
                        [
                            [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]
                        ],
                        [
                            [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]
                        ],
                        [
                            [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]
                        ],
                        [
                            [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]
                    ],
                    [
                        [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]
                    ],
                    [
                        [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]
                    ]
                ]

    def func1(self):
        """
        加密
        """
        #读取明文
        self.m=list(self.mw.toPlainText())
        for i in range(len(self.m)):
            self.m[i]=ord(self.m[i])
        self.block = len(self.m) // 8 +1
        #读取密钥
        self.key=list(self.my.toPlainText())
        for i in range(len(self.key)):
            self.key[i]=ord(self.key[i])
        self.Des_SetSubKey()
        #分组加密
        for i in range(self.block-1):
            m_block=[]
            c_block=[]
            for j in range(8):
                m_block.append(self.m[8*i+j])
            c_block=self.Des_Run(m_block,1)
            for j in range(8):
                self.c.append(str(c_block[j]))
        #最后一组
        m_block=[]
        c_block=[]
        for j in range(8):
```

```
            m_block.append(0)
            self.m.append(0)
        for j in range(8):
            m_block[j]=self.m[8*(self.block-1)+j]
        c_block=self.Des_Run(m_block,1)
        for j in range(8):
            self.c.append(str(c_block[j]))
        #显示密文
        self.mw2.setPlainText(''.join(self.c))

    def func2(self):
        """
        解密
        """
        #分组解密
        for i in range(self.block):
            m_block=[]
            c_block=[]
            for j in range(8):
                c_block.append(int(self.c[8*i+j]))
            m_block=self.Des_Run(c_block,0)
            for j in range(8):
                self.m1.append(chr(m_block[j]))
            #显示明文
        self.jmw.setPlainText(''.join(self.m1))

    def clear(self):
        self.mw.setPlainText('')
        self.mw2.setPlainText('')
        self.my.setPlainText('')
        self.jmw.setPlainText('')
        self.m1=[]
        self.c=[]

    def Des_Run(self, In, Flag):
        M=[]
        M = self.ByteToBit(In,64)
        Li=M[:32]
        Ri=M[32:]
        if Flag == 1:
            M = self.Transform(M,self.IP_Table,64)
            for i in range(16):
                Temp = Ri
                Ri=self.F_Func(Ri,self.SubKey[i])
```

```
                Ri = self.Xor(Ri,Li)
                Li=Temp
            Li,Ri = Ri,Li
            M = Li + Ri
            M = self.Transform(M,self.IPInv_Table,64)
        else:
            M = self.Transform(M,self.IP_Table,64)
            for i in range(15,-1,-1):
                Temp = Ri
                Ri=self.F_Func(Ri,self.SubKey[i])
                Ri = self.Xor(Ri,Li)
                Li=Temp
            Li,Ri = Ri,Li
            M = Li + Ri
            M = self.Transform(M,self.IPInv_Table,64)
        Out=self.BitToByte(M,64)
        return Out

    def Des_SetSubKey(self):
        self.SubKey=[]
        self.K=self.ByteToBit(self.key,64)
        self.KL=self.K[:28]
        self.KR=self.K[28:]
        self.K=self.Transform(self.K,self.PC1_Table,56)
        for i in range(16):
            self.KL=self.RotateL(self.KL,self.LS_Table[i])
            self.KR=self.RotateL(self.KR,self.LS_Table[i])
            self.K=self.KL+self.KR
            self.SubKey.append(self.Transform(self.K,self.PC2_Table,48)
)

    def F_Func(self,In,Ki):
        MR = self.Transform(In,self.E_Table,48)
        MR = self.Xor(MR,Ki)
        In = self.S_Func(In,MR)
        In = self.Transform(In,self.P_Table,32)
        return In

    def S_Func(self,Out,In):
        Out=[]
        for i in range(8):
            j=(In[0+6*i]<<1)+In[5+6*i]
            k=(In[1+6*i]<<3)+(In[2+6*i]<<2)+(In[3+6*i]<<1)+In[4+6*i]
            Sbox=list('{:0>4s}'.format(bin(self.S_Box[i][j][k])[2:]))
```

```
            for i in range(len(Sbox)):
                Sbox[i]=int(Sbox[i])
            Out+=Sbox
        return Out

    def Transform(self, In, Table, len):
        Temp=[]
        for i in range(len):
            Temp.append(In[Table[i]-1])
        return Temp

    def Xor(self,InA,InB):
        for i in range(len(InA)):
            InA[i]=InB[i] ^ InA[i]
        return InA

    def RotateL(self,In,loop):
        return In[loop:] + In[:loop]

    def ByteToBit(self, In, bits):
        Out=[]
        for i in range(bits):
            Out.append((In[i//8]>>(7-i%8)) & 1)
        return Out

    def HalfByteToBit(self, In, bits):
        Out=[]
        for i in range(bits):
            Out.append((In[i//4]>>(3-i%4)) & 1)
        return Out

    def BitToByte(self, In, bits):
        Out=[]
        for i in range(bits//8):
            Out.append(0)
        for i in range(bits):
            Out[i//8] = (In[i]<<(7-i%8)) + Out[i//8]
        return Out

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ui=Min()
    ui.show()
    sys.exit(app.exec_())
```

DES 密码输出界面截屏如下：