

# DES 算法的程序设计和实现

## DES 算法的程序设计和实现

实验目标

实验概述

算法原理

数据结构

Initial permutation (IP)

Final permutation (  $IP^{-1}$  )

Expansion function (E)

Permutation (P)

Permuted choice 1 ( PC-1 )

Permuted choice 2 ( PC-2 )

S-box

基本流程

实验过程

表置换

生成子密钥

Feistel 函数

实验结果

正确解码

密码错误

实验参考

# 实验目标

完成一个DES 算法的详细设计,内容包括:

- 算法概述;
- 总体结构;
- 数据结构。

## 实验概述

### 算法原理

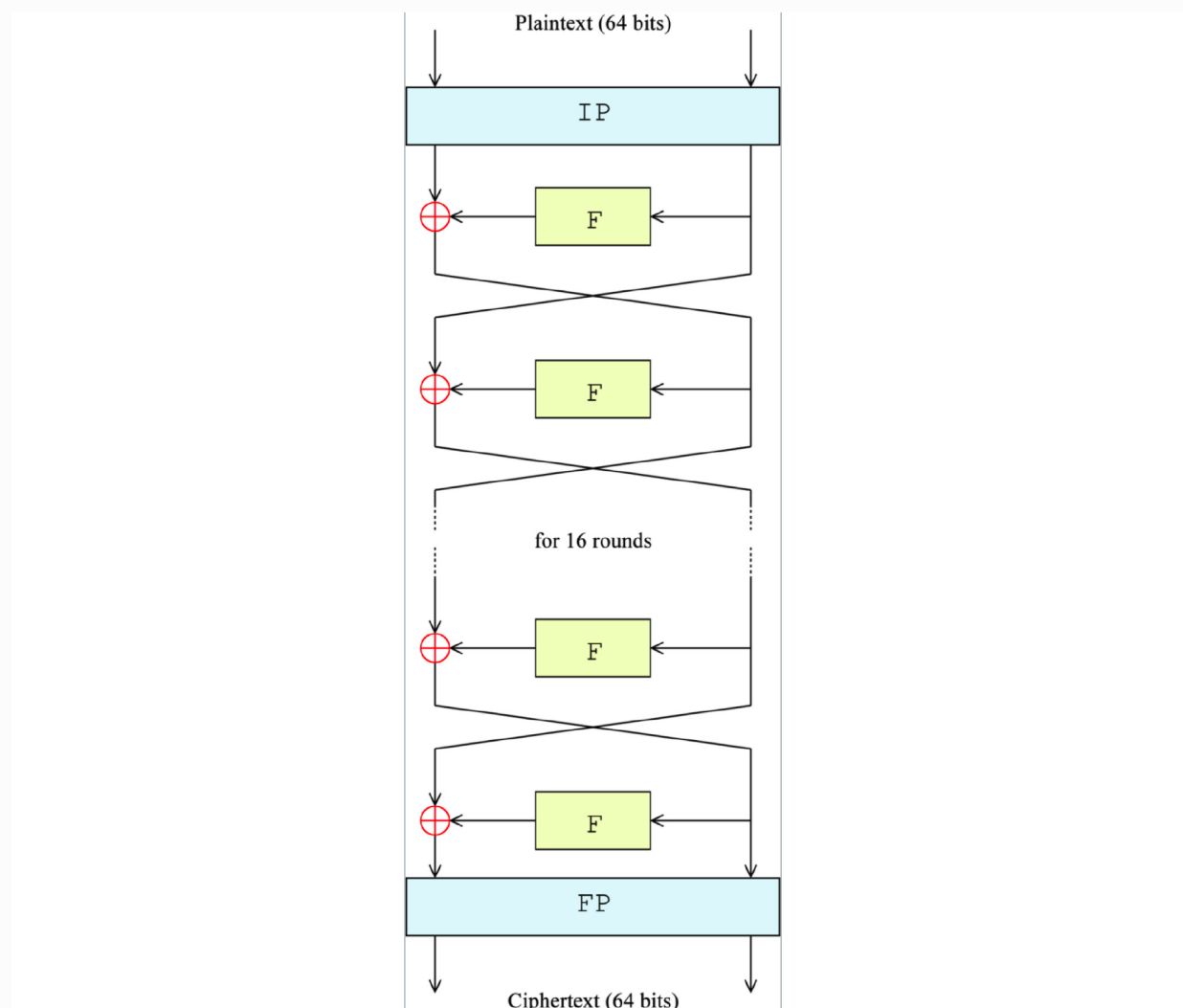
**DES (Data Encryption Standard)** 是一种用于电子数据加密的对称密钥块加密算法.它以64位为分组长度, 64位一组的明文作为算法的输入, 通过一系列复杂的操作, 输出同样64位长度的密文。DES 同样采用64位密钥, 但由于每8位中的最后1位用于奇偶校验, 实际有效密钥长度为56位。密钥可以是任意的56位的数, 且可随时改变。

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的两个重要的安全特性是混淆和扩散。其中混淆是指通过密码算法使明文和密文以及密钥的关系非常复杂, 无法从数学上描述或者统计。**扩散**是指明文和密钥中的每一位信息的变动, 都会影响到密文中许多位信息的变动, 从而隐藏统计上的特性, 增加密码的安全。

DES算法的基本过程是换位和置换。如图, 有16个相同的处理阶段, 称为轮。还有一个初始和最终的排列, 称为 IP 和 FP, 它们是反向的 (IP 取消 FP 的作用, 反之亦然)。

在主轮之前，块被分成两个32位的一半和交替处理；这种纵横交错的方案被称为Feistel 方法。Feistel 结构确保了解密和加密是非常相似的过程——唯一的区别是在解密时子键的应用顺序是相反的。其余的算法是相同的。这大大简化了实现，特别是在硬件中，因为不需要单独的加密和解密算法。

⊕ 符号表示异或（XOR）操作。Feistel 函数将半块和一些键合在一起。然后，将Feistel 函数的输出与块的另一半组合在一起，在下一轮之前交换这一半。在最后一轮之后，两队交换了位置；这是 Feistel 结构的一个特性，使加密和解密过程类似。



## 数据结构

Initial permutation (IP)

IP 置换表指定64位块上的输入排列。其含义如下：输出的第一个比特来自输入的第58位;第二个位来自第50位，以此类推，最后一个位来自第7位输入。

**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

Final permutation (  $IP^{-1}$  )

最后的排列是初始排列的倒数。

$IP^{-1}$

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

## Expansion function (E)

展开函数被解释为初始排列和最终排列。注意，输入的一些位在输出时是重复的;输入的第5位在输出的第6位和第8位中都是重复的。因此，32位半块被扩展到48位。

## E

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

## Permutation (P)

P排列打乱了32位半块的位元。

## 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

## Permuted choice 1 ( PC-1 )

表的“左”和“右”部分显示了来自输入键的哪些位构成了键调度状态的左和右部分。输入的64位中只有56位被选中；剩下的8（8、16、24、32、40、48、56、64）被指定作为奇偶校验位使用。

PC-1													
Left							Right						
57	49	41	33	25	17	9	63	55	47	39	31	23	15
1	58	50	42	34	26	18	7	62	54	46	38	30	22
10	2	59	51	43	35	27	14	6	61	53	45	37	29
19	11	3	60	52	44	36	21	13	5	28	20	12	4

## Permuted choice 2 ( PC-2 )

这个排列从56位键调度状态为每轮选择48位的子键。

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

## S-box

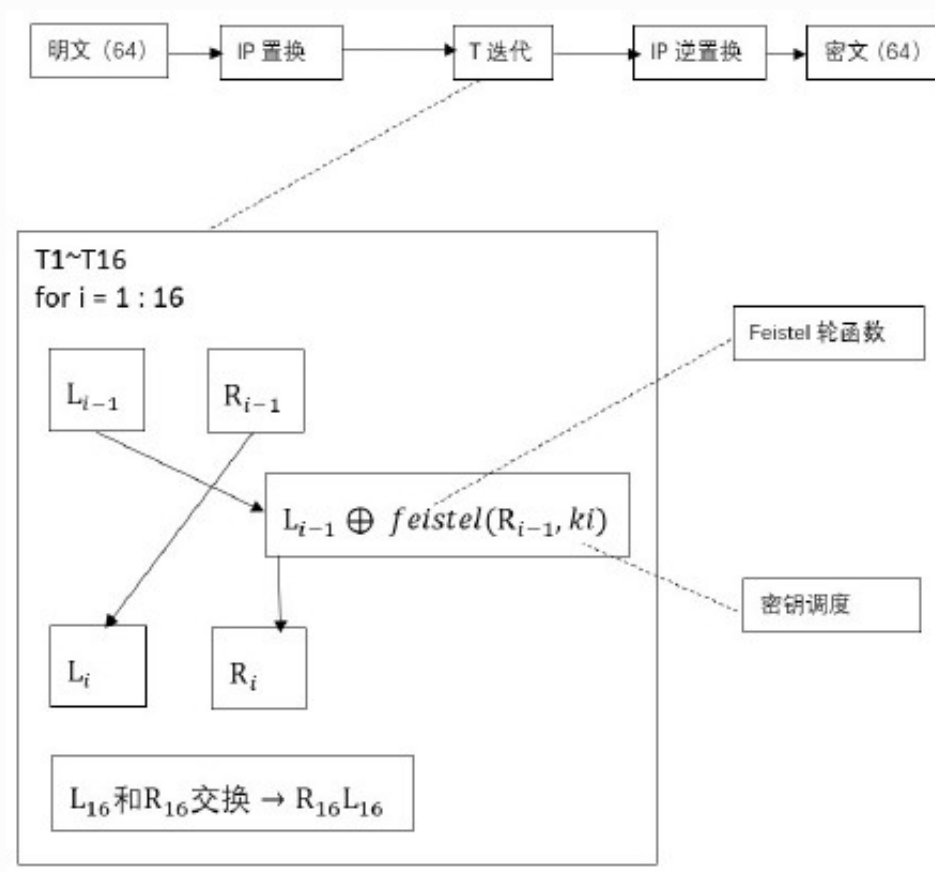
这个表列出了DES中使用的8个S-box，每个S-box用4位的输出替换6位的输入。给定一个6位输入，通过使用外部的两个位选择行，以及使用内部的四个位选择列，就可以找到4位输出。例如，一个输入“011011”有外部位“01”和内部位“1101”。第一行为“00”，第一列为“0000”，S-box S5对应的输出为“1001”(=9)，即第二行第14列的值。



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

# 基本流程

DES算法的基本流程图如下：



DES算法是典型的对称加密算法，在输入64比特明文数据后，通过输入64比特密钥和算法的一系列加密步骤后，可以得到同样为64比特的密文数据。反之，我们通过已知的密钥，可以将密文数据转换回明文。我们将算法分为了三大块：**IP置换**、**16次T迭代**和**IP逆置换**，加密和解密过程分别如下：

- $C = E_k(M) = IP^{-1} \cdot W \cdot T_{16} \cdot T_{15} \cdots T_1 \cdot IP(M)$
- $M = D_k(C) = IP^{-1} \cdot W \cdot T_1 \cdot T_2 \cdots T_{16} \cdot IP(C)$

符号	释意
<b>M</b>	算法输入的64位明文块
<b>E</b>	描述以K 为密钥的加密函数，由连续的过程复合构成
<b>IP</b>	64位初始置换
$T_n$	一系列的迭代变换
<b>W</b>	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 算法输出的64位密文块

## 实验过程

实验的设计模式是自顶向下的结构，用C语言去分别是先各个函数的功能，最后通过主函数将所有函数进行整合，让算法更加清晰客观。

### 表置换

通过IP置换表，根据表中所示下标，找到相应位置进行置换。

```
const char IP_Table[64]={
    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
};

void TablePermute(bool *DatOut,bool *DatIn,const
char *Table,int Num)
{
    int i=0;
    static bool Temp[256]={0};
    for(i=0;i<Num;i++)
    {
```

```

        Temp[i]=DatIn[Table[i]-1];
    }
    BitsCopy(DatOut,Temp,Num);
}

```

对于16次  $T$  迭代，我们先将传入的经过 IP 混淆过的64位明文的左右两部分，分别为32位的  $L_0$  和32位的  $R_0$ 。之后我们将  $L_{16}$  和  $R_{16}$  进行交换，得到作为IP逆置换的输入：

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

## 生成子密钥

子密钥的生成，经历下面一系列步骤：首先对于64位密钥，进行置换选择，因为将用户输入的64位经历压缩变成了56位，所以我们将左面和右面的各28位进行循环位移。左右两部分分别按下列规则做循环移位：当  $flag = 1, 2, 9, 16$ ，循环左移1位；其余情况循环左移2位。最后将得到的新的左右两部分进行连接得到56位密钥。

```

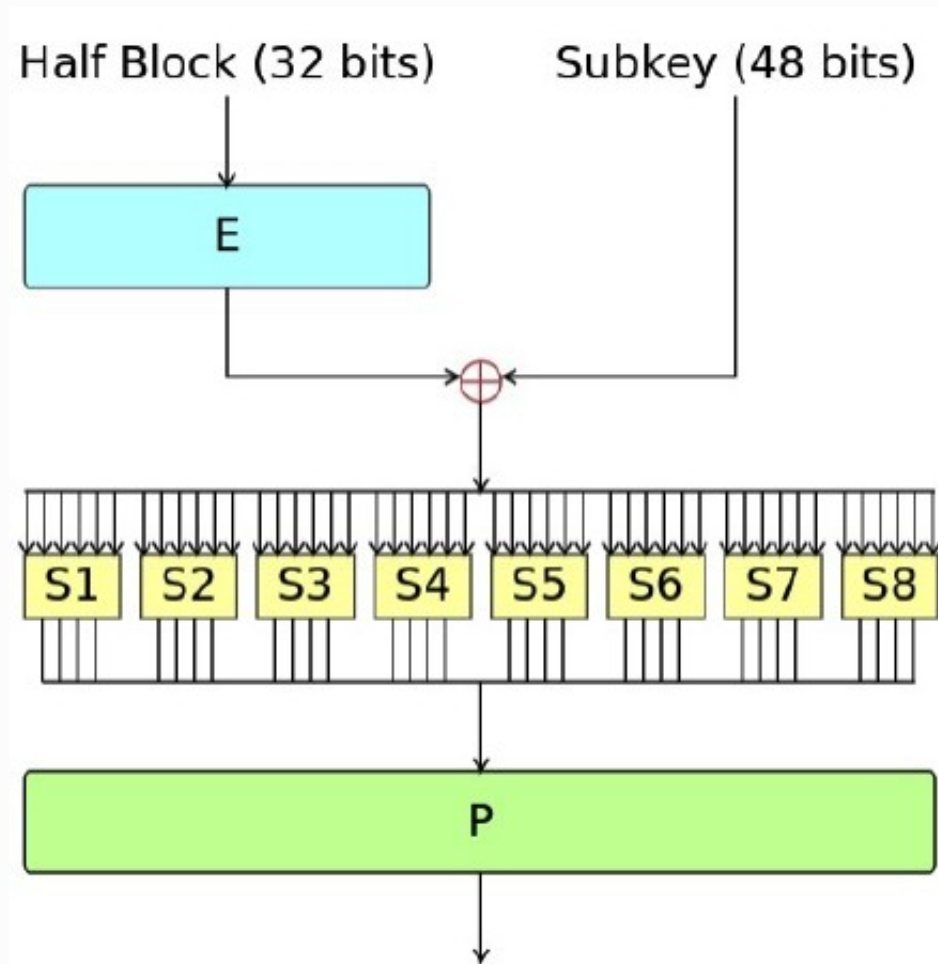
static char Move_Table[16]={
    1, 1, 2, 2, 2, 2, 2, 2,
    1, 2, 2, 2, 2, 2, 2, 1
};

void SetKey(char KeyIn[8])
{
    int i=0;
    static bool KeyBit[64]={0};
    static bool *KiL=&KeyBit[0], *KiR=&KeyBit[28];
    ByteToBit(KeyBit,KeyIn,64);
    TablePermute(KeyBit,KeyBit,PC1_Table,56);
    for(i=0;i<16;i++)
    {
        LoopMove(KiL,28,Move_Table[i]);
        LoopMove(KiR,28,Move_Table[i]);
    }
}

```

```
TablePermute(SubKey[i],KeyBit,PC2_Table,48);  
    }  
}
```

## Feistel 函数



对半块的 Feistel 操作分为以下五步：

1. **展开**：32位的半块通过重复一半位的展开排列扩展到48位。输出由8个6位（48位）块组成，每个块包含4个相应的输入位的副本，加上从每个输入块到任意一侧的相邻位的副本。
2. **密钥混合**：结果与使用 XOR 操作的子密钥结合。16个48位的子键（每个轮一个）由主键派生，使用下面描述的键调度。
3. **替换**：将子键混合后，将块分割成8个6位的块，再用 S-box 或替换盒进行处理。根据一个非线性变换，八个 S-box 中的每一个都用四个输出位替换了它的六个输入位。S-box 提供了 DES 安全的核心，如果没有它们，密码就会是线性的，而且容易被破解。

4. *排列*: 根据一个固定的排列, 即 P-box, 将s-box的32个输出重新排列。这样设计的目的是, 经过排列后, 这一轮中每个 S-box 输出的比特被分散到下一轮的4个不同的 S-box 上。
5. *置换*: 最后的 IP 逆置换同之前的 IP 置换基本相同, 我们通过 IP 逆置换表, 根据表中所示下标, 找到相应位置进行置换。

```
const char IPR_Table[64]={
    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
};

void F_Change(bool DatIn[32],bool DatKi[48])
{
    static bool MiR[48]={0};
    TablePermute(MiR,DatIn,E_Table,48);
    Xor(MiR,DatKi,48);
    S_Change(DatIn,MiR);
    TablePermute(DatIn,DatIn,P_Table,32);
}
```

## 实验结果

如下三图展示了实验的结果（黄色代表明文，蓝色代表暗码，红色表示密码，绿色表示结果）

### 正确解码

```
NinodeMacBook-Pro:src nino$ ./a.out
Welcome! Please input your Message(64 bit):
warning: this program uses gets(), which is unsafe.
NinoLau
Please input your Secret Key:
16340154
Your Message is Encrypted!:
2 7 9 4 C 6 0 4 B 2 9 6 8 4 F 9

Please input your Secret Key to Deciphering:
16340154
Deciphering Over !!:
N i n o L a u
sh: pause: command not found
NinodeMacBook-Pro:src nino$ _
```

```
NinodeMacBook-Pro:src nino$ ./a.out
Welcome! Please input your Message(64 bit):
warning: this program uses gets(), which is unsafe.
I'm Nino.
Please input your Secret Key:
88888888
Your Message is Encrypted!:
2 6 2 6 F 8 C B 9 8 0 F D 8 9 5

Please input your Secret Key to Deciphering:
88888888
Deciphering Over !!:
I ' m   N i n o
```


如上二图表明，在给出正确的密码后，可以得到对应的明文。

## 密码错误



```
NinodeMacBook-Pro:src nino$ ./a.out
Welcome! Please input your Message(64 bit):
warning: this program uses gets(), which is unsafe.
Liushuo
Please input your Secret Key:
00000000
Your Message is Encrypted!:
2 7 F 9 D 6 1 E A 9 5 4 1 7 4 D
Please input your Secret Key to Deciphering:
88888888
Deciphering Over !!:
? ? ? ? 5 ?
```

wrong  
password  
won't  
decipher  
correctly



若密码错误，将解码出错误答案。

## 实验参考

- 【1】 Data Encryption Standard
- 【2】 DES算法的详细设计（简单实现）
- 【3】 深入理解并实现DES算法
- 【4】 DES算法原理完整版
- 【5】 安全体系（一）—— DES算法详解