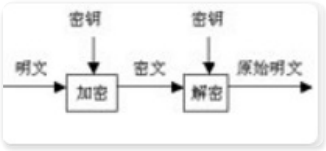
DES算法设计实验报告

学号：15331061
姓名：邓旺

算法原理概述

DES加密算法作为一种对称密码体制，具有如下的几个组成部分，可以用于加密也可以用于解密，并且加密和解密用的是同一个算法。

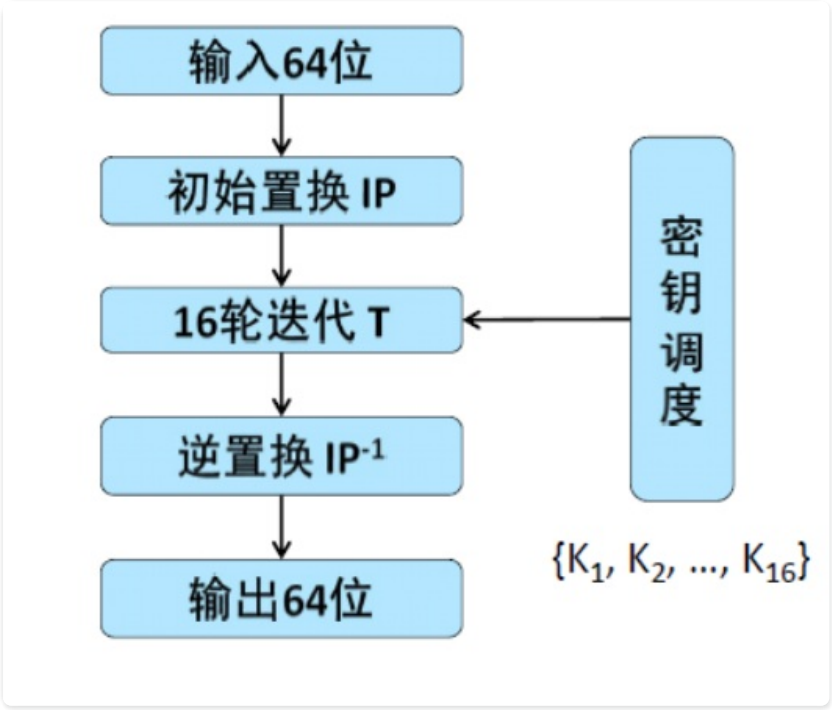


此外呢，DES加密算法还是一种典型的块加密方法，它以64位为分组长度，64位一组的明文作为算法的输入。它的密钥长度是56位（每个第8位都用作奇偶校验），密钥可以是任意的56位的数，而且可以任意时候改变，通过一系列复杂的操作，输出同样64位长度的密文。
算法的主要是通过多轮的换位和置换以及特定的数学运算实现的，主要的步骤如下：

- 置换：将数码中的某一位的值根据置换表的规定，用另一位代替。通过不规则的操作确保加密的安全性。
- 扩展：将一段数码用置换的方法，以扩展置换表来规定扩展后的数码每一位的替代值，从而扩展成比原来位数更长的数码。
- 压缩：将一段数码通过置换表压缩成比原来位数更短的数码。
- 异或：一种按位的二进制布尔代数运算。

总体结构

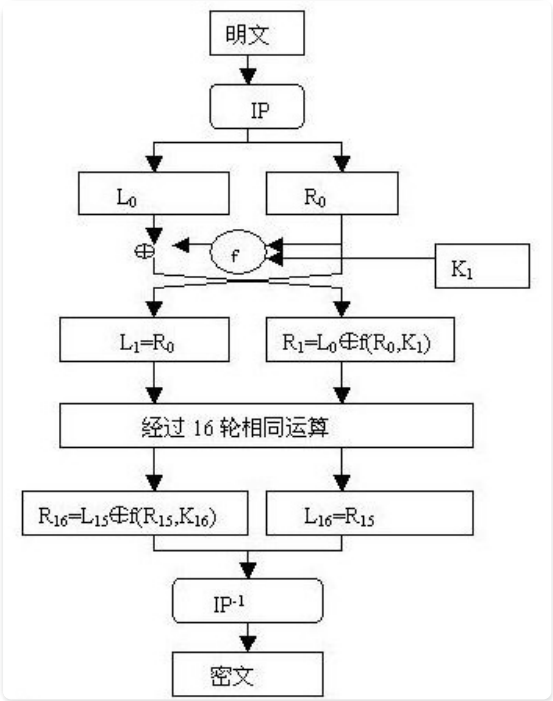
DES 算法的采用的是一种 Feistel 结构，是一种对称的密码结构。在DES 算法中的具体体现如下：



> 输入64位明文M 时，密钥按(K1K2 … K16)次序调度，是加密过程。
> 输入64位密文C 时，密钥按(K16K15 … K1)次序调度，是解密过程。

模块分解

DES算法的主要流程如下图所示，按照流程可大致分为以下几个模块：

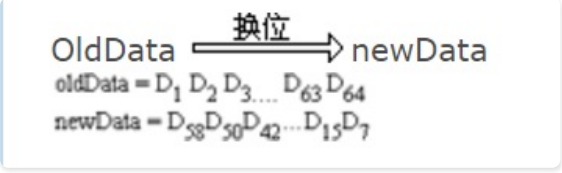


初始置换IP

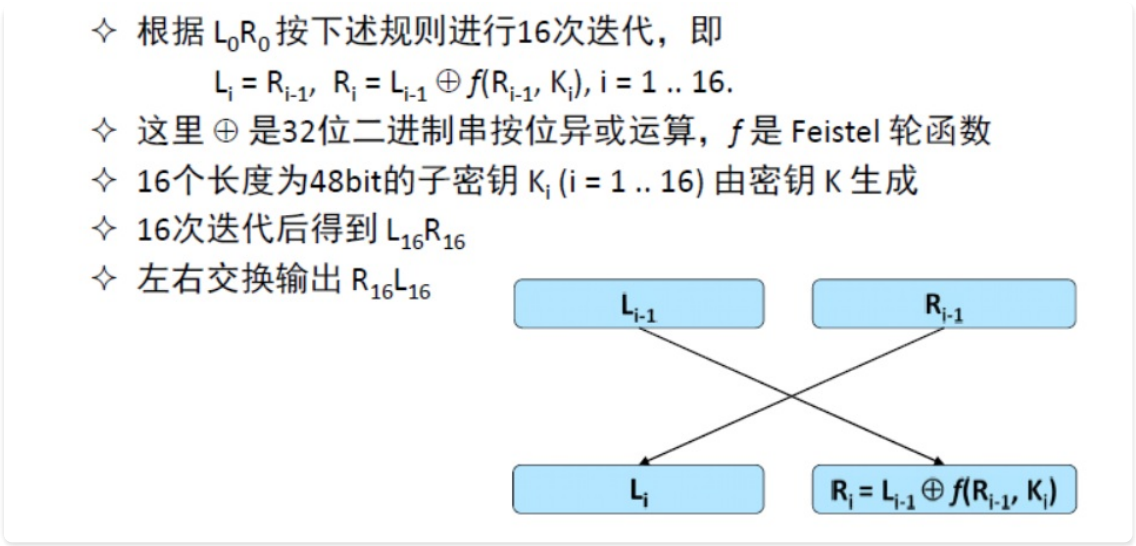
给定64位明文块M，通过一个固定的初始置换IP 来重排M 中的二进制位，得到二进制串M0 = IP(M) = L0R0，这里L0 和R0分别新数据的前32位和后32位。下表给出IP 置换后的下标编号序列。

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

表中的数字代表新数据在原输入数据中的位置，例如数字58代表输入的64位数据的原第58位换到第1位，原第50位换到第2位，依此类推，。。。，原第7位换到第64位，最后得到新的64位数据。

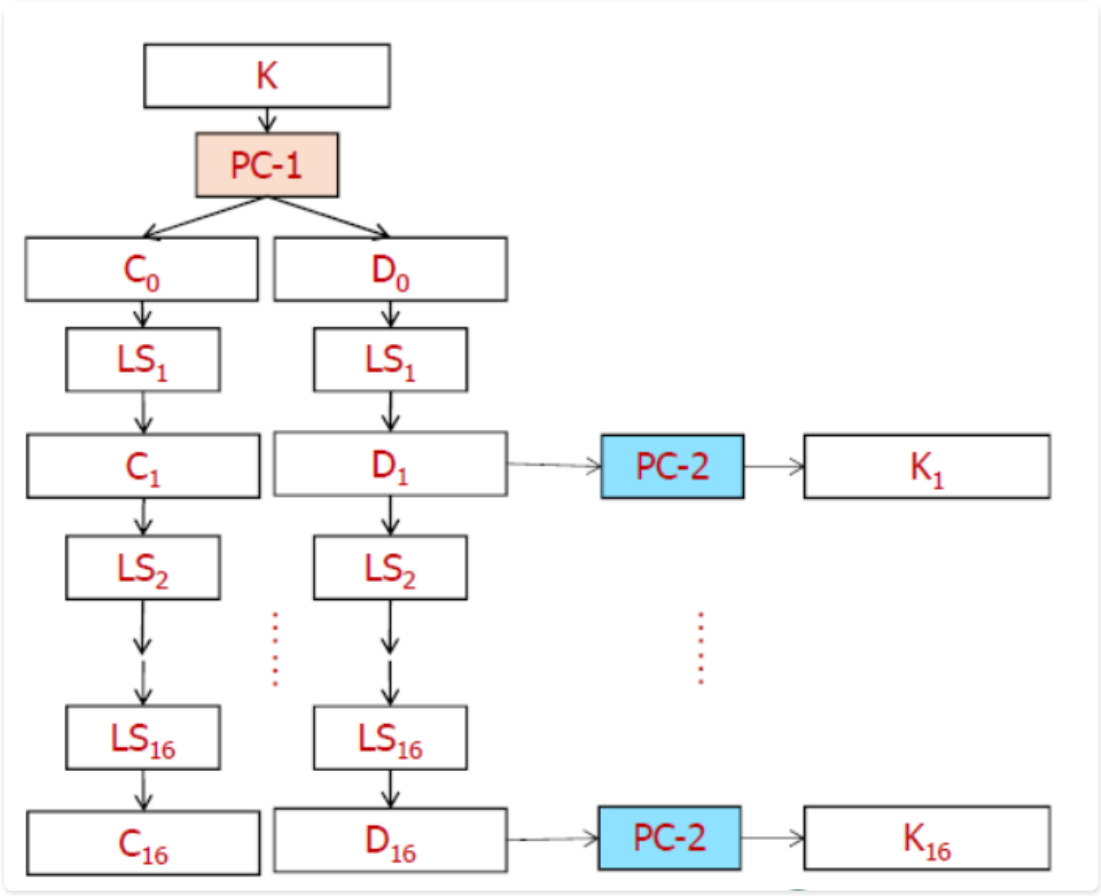


迭代T



其中，该步骤包括以下过程：

- 密钥置换



不考虑每个字节的第8位，DES的密钥由64位减至56位，每个字节的第8位作为奇偶校验位，产生的56位密钥如下（注意表中没有8,16,24,32,40,48,56和64这8位）

PC-1 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

在DES的每一轮中，从56位密钥产生出不同的48位子密钥，确定这些子密钥的方式如下：
1). 将56位的密钥分成两部分，每部分28位。
2). 根据轮数，这两部分分别循环左移1位或2位。其中移位就是将一段数码按照规定的位数整体性地左移或右移。循环右移就是当右移时，把数码的最后的位移到数码的最前头，循环左移正相反。每轮移动的位数如下表：

轮数	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

移动后，从56位中选出48位。这个过程中，既置换了每位的顺序，又选择了子密钥，因此称为压缩置换。压缩置换规则如下表（注意表中没有9，18，22，25，35，38，43和54这8位）

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

- E扩展

让IP置换后获得的右半部分R0，将32位输入扩展为48位(分为4位×8组)输出，从而生成的数据可以与48位的子密钥可以进行异或运算。

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

从表中可以看出，扩展的数据是从相邻两组分别取靠近的一位，4位变为6位。例如表中第二行的4取自上组中的末位，9取自下组中的首位。

- S盒代替

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$b_1\ b_2\ b_3\ b_4\ b_5\ b_6$，则由$n = (b_1\ b_6)10$ 确定行号，$m = (b_{21}b_3\ b_4\ b_5)_{10}$ 确定列号，$[Si]_{n,m}$ 元素的值的二进制形式即为所要的S_i 的输出。

8个S盒如下：

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

- P-置换

把输入的每位映射到输出位，任何一位不能被映射两次，映射规则如下表：

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

逆置换 IP^{-1}

初始置换的逆过程，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^{-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

类-C语言算法过程

注:大部分代码参考自这篇博客，自己完成了伪代码部分，但是在部分模块的实现上有一定的困难。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*IP置换矩阵*/
int 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};

//逆IP置换矩阵
int 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};
//E扩展矩阵
int E_Table[48] = {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};
//  P-置换矩阵
int P_Table[32] = {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};
 //PC-1置换矩阵
int PC1_Table[56] = {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};
// PC-2压缩置换矩阵
int PC2_Table[48] = {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};

//16次轮转循环左移矩阵
const static int LOOP_Table[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

//8个S盒，三维数组
int S_Box[8][15][16] = {
     //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}}
            };

//初始IP置换
void opIP(const int input[64],int output[64],int table[64])
{
    int i;
    for(i=0;i<64;i++)
    {
        output[i]=input[table[i]-1];
    }
};

//E扩展
void opE(const int input[32],int output[48],int table[48])
{
    int i;
    for(i=0; i<48;i++)
    {
        output[i]=input[table[i]-1];
    }
};

//P置换
void opP(const int input[32], int output[32], int table[32])
{
    int i;
    for(i=0; i<32; i++)
    {
        output[i]=input[table[i]-1];
    }
};

//逆IP
void IP_R(const int input[64],int output[64],int table[64])
{
    int i;
    for(i=0;i<64;i++)
    {
        output[i]=input[table[i]-1];
    }
};

//PC_1
void PC_1(const int input[64],int output[56],int table[56])
{
    int i;
    for(i=0;i<56;i++)
    {
        output[i]=input[table[i]-1];
    }
};

//PC_2
void PC_2(const int input[56],int output[48],int table[48])
```

```
void PC_2(const int input[56],int output[48],int table[48])
{
    int i;
    for(i=0;i<48;i++)
    {
        output[i]=input[table[i]-1];
    }
};

//S盒
void opS(const int input[48],int output[32], int table[8][16][16])
{
 int i;
    int j=0;
    int INT[8];
    for(i=0;i<48;i=i+6)
    {
        INT[j]=table[j][(input[i]<<1)+(input[i+5])][(input[i+1]<<3)+(input[i+2]<<2)+(i
nput[i+3]<<1)+(input[i+4])];
        j++;
    }
    for(j=0;j<8;j++)
    {
        for(i=0;i<4;i++)
        {
            output[3*(j+1)-i+j]=(INT[j]>>i)&1;
        }
    }
};


//异或操作
void opXor(int *INA,int *INB,int len)
{
    int i;
    for(i=0; i<len; i++)
    {
        *(INA+i) = *(INA+i)^*(INB+i);
    }
};

//Feistel 轮函数f(Ri-1, Ki)
void F_func(int input[32],int output[32], int subkey[48])
{
    int len=48;
    int temp[48]={0};
    int temp_1[32]={0};
    opE(input,temp,E_Table);
    opXor(temp,subkey,len);
    opS(temp,temp_1,S_Box);
    opP(temp_1,output,P_Table);
};

//密钥循环左移位
void RotateL(const int input[28],int output[28], int leftCount)
{
    int i;
    int len = 28 ;
    for(i=0;i<len;i++)
    {
        output[i]=input[(i+leftCount)%len];
    }
};

//生成子密钥
void subKey_fun(const int input[64], int Subkey[16][48])
{
    int loop=1,loop_2=2;
    int i,j;
    int c[28],d[28];
```

```
    int pc_1[56]={0};
    int pc_2[16][56]={0};
    int rotatel_c[16][28]={0};
    int rotatel_d[16][28]={0};
    PC_1(input,pc_1,PC1_Table);
    for(i=0;i<28;i++)
    {
        c[i]=pc_1[i];
        d[i]=pc_1[i+28];
    }
    int leftCount = 0;
    for(i=1;i<17;i++)
    {
        if(i==1||i==2||i==9||i==16)
        {
             leftCount += loop;
             RotateL(c,rotatel_c[i-1],leftCount);
             RotateL(d,rotatel_d[i-1],leftCount);
        }
        else
        {
             leftCount += loop_2;
             RotateL(c,rotatel_c[i-1],leftCount);
             RotateL(d,rotatel_d[i-1],leftCount);
        }
    }
    for(i=0;i<16;i++)
    {
        for(j=0;j<28;j++)
        {
            pc_2[i][j]=rotatel_c[i][j];
            pc_2[i][j+28]=rotatel_d[i][j];
        }
    }
    for(i=0;i<16;i++)
    {
        PC_2(pc_2[i],Subkey[i],PC2_Table);
    }
};

//把CHAR转换为INT
void CharToBit(const char input[], int output[])
{
    int i,j;
    for(j=0; j<8; j++)
    {
        for(i=0; i<8; i++)
        {
            output[7*(j+1)-i+j] = (input[j] >> i) & 1 ;
        }
    }
};

//把INT转换为CHAR
void BitToChar(const int intput[],char output[])
{
    int i,j;
    for(j=0;j<8;j++)
    {
        for(i=0;i<8;i++)
        {
            output[j]=output[j]*2+intput[i+8*j];
        }
    }
};

void Des_encrypt(char input[8],char key_in[8], int output[64])
{
    int Ip[64]={0};//存储初始置换后的矩阵
    int output_1[64]={0};
```

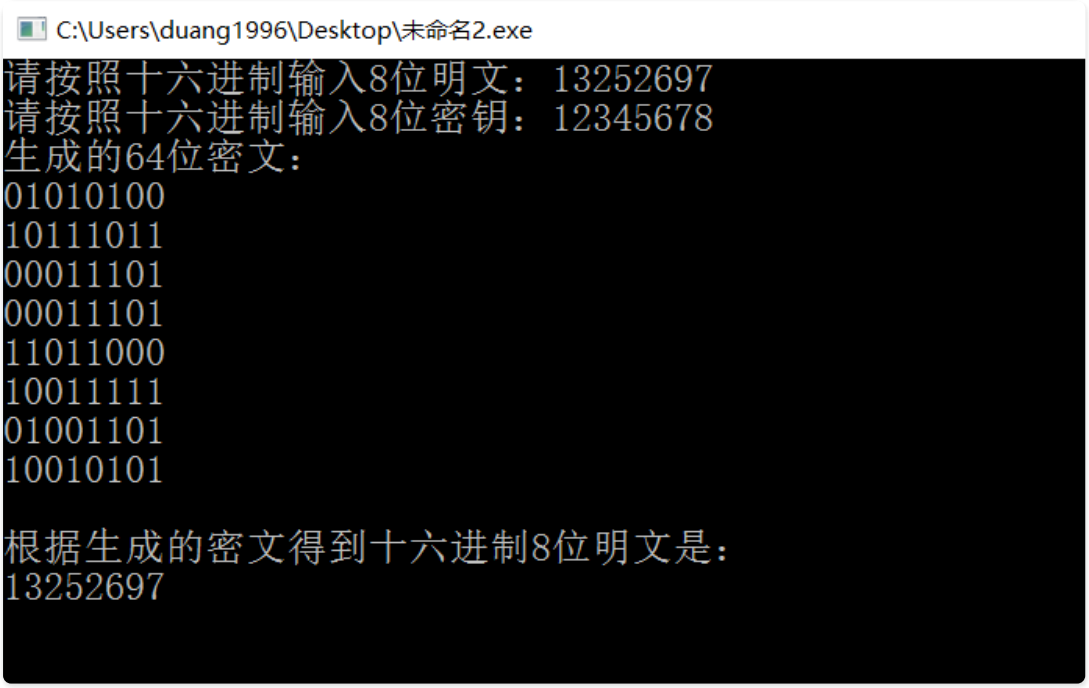
```
    int subkeys[16][48];
    int chartobit[64]={0};
    int key[64];
    int l[17][32],r[17][32];
    CharToBit(input,chartobit);//正确,转换为64个二进制数的操作正确!
    opIP(chartobit,Ip,IP_Table);//正确, IP初始置换!
    CharToBit(key_in,key);//正确!
    subKey_fun(key,subkeys);//正确!
    for(int i=0;i<32;i++)
    {
        l[0][i]=Ip[i];
        r[0][i]=Ip[32+i];
    }
    for(int j=1;j<16;j++)//前15轮的操作
    {
        for(int k=0;k<32;k++)
        {
            l[j][k]=r[j-1][k];
        }
            F_func(r[j-1],r[j],subkeys[j-1]);
            opXor(r[j],l[j-1],32);
    }
    int t=0;
    for(t=0;t<32;t++)//最后一轮的操作
    {
        r[16][t]=r[15][t];
    }
        F_func(r[15],l[16],subkeys[15]);
        opXor(l[16],l[15],32);
        for(t=0;t<32;t++)
    {
        output_1[t]=l[16][t];
        output_1[32+t]=r[16][t];
    }
        IP_R(output_1,output,IPR_Table);
};

void Des_decrypt(int input[64],char key_in[8],char output[8])
{
    int Ip[64]={0};//存储初始置换后的矩阵
    int output_1[64]={0};
    int output_2[64]={0};
    int subkeys[16][48];
    int chartobit[64]={0};
    int key[64];
    int l[17][32],r[17][32];
    opIP(input,Ip,IP_Table);//正确, IP初始置换!
    CharToBit(key_in,key);//正确!
    subKey_fun(key,subkeys);//正确!
    for(int i=0;i<32;i++)
    {
        l[0][i]=Ip[i];
        r[0][i]=Ip[32+i];
    }
    for(int j=1;j<16;j++)//前15轮的操作
    {
        for(int k=0;k<32;k++)
        {
            l[j][k]=r[j-1][k];
        }
            F_func(r[j-1],r[j],subkeys[16-j]);
            opXor(r[j],l[j-1],32);
    }
    int t=0;
    for(t=0;t<32;t++)//最后一轮的操作
    {
        r[16][t]=r[15][t];
    }
    F_func(r[15],l[16],subkeys[0]);
    opXor(l[16],l[15],32);
```

```
    for(t=0;t<32;t++)
    {
        output_1[t]=l[16][t];
        output_1[32+t]=r[16][t];
    }
        IP_R(output_1,output_2,IPR_Table);
        BitToChar(output_2,output);
};

int main()
{
 int output1[64] = {0};
 char output2[64] = {0};
 char data[9] = {0};
 char key[9] = {0};
 printf("请按照十六进制输入8位明文: ");
 gets(data);
 printf("请按照十六进制输入8位密钥: ");
 gets(key);
 Des_encrypt(data, key, output1);
 printf("生成的64位密文: \n");
 for(int i=0; i<64; i++)
    {
        printf("%d", output1[i]);
        if((i+1) % 8 == 0)
            printf("\n");
    }
 printf("\n根据生成的密文得到十六进制8位明文是: \n");
 Des_decrypt(output1, key, output2);
 for(int i=0; i<64; i++)
    {
        printf("%c", output2[i]);
        if((i+1) % 8 == 0)
            printf("\n");
    }
    return 0;
}
```

实验结果



文献参考

安全体系（一）—— DES算法详解
DES加密算法原理
工程实践——DES算法的C语言实现