



# 南昌大学实验报告

学生姓名： 丁俊 学 号： 8003119100 专业班级： 信息安全 193 班  
实验类型： ☐ 验证 ☐ 综合 ☐ 设计 ☐ 创新 实验日期： 2021.11.5 实验成绩： \_\_\_\_\_

## 一、实验项目名称

程序实现 DES 加密算法

## 二、实验目的

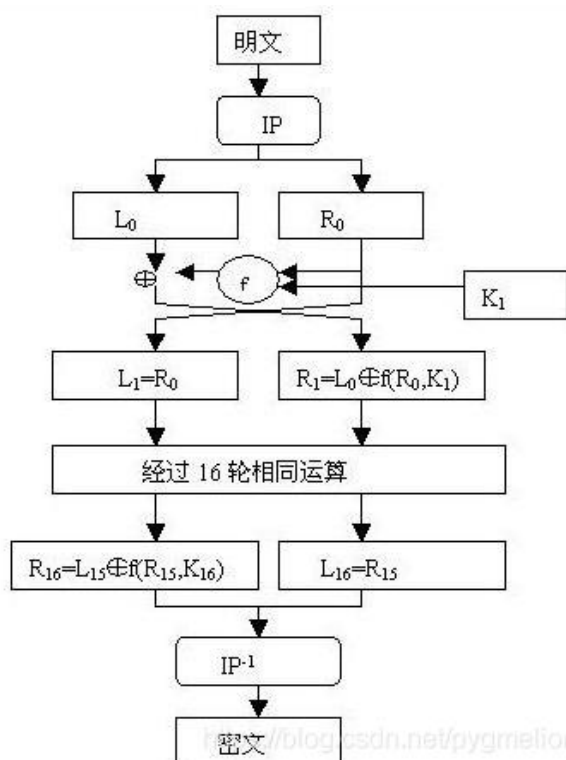
- 1、理解 DES 算法的概念及原理
- 2、理解 DES 算法的加密流程

## 三、实验基本原理

### 1. 算法原理

DES 算法为密码体制中的对称密码体制，又被称为美国数据加密标准，是 1972 年美国 IBM 公司研制的对称密码体制加密算法。明文按 64 位进行分组，密钥长 64 位，密钥事实上是 56 位参与 DES 运算（第 8、16、24、32、40、48、56、64 位是校验位，使得每个密钥都有奇数个 1）分组后的明文组和 56 位的密钥按位替代或交换的方法形成密文组的加密方法。

### 2. 基本结构



### 1. 求子钥 K1 - K16

K -> 置换得 K+ -> 得 C0, D0 -> 左移求 C1D1 - C16D16 -> 置换得 K1 - K16

## 2. 利用子钥求 L16R16

(1)  $M \rightarrow$  置换得  $M^+ \rightarrow$  得  $L_0, R_0$

(2) 进入 16 次循环:

$R_n$  扩展置换 (32 $\rightarrow$ 48)  $\rightarrow$  结果异或  $K_{n+1} \rightarrow$  异或后结果进入 S 盒 (48 $\rightarrow$ 32)  $\rightarrow$  S 盒输出进行 P 置换  $\rightarrow$  结果异或  $L_n \rightarrow$  结果赋给  $R_{n+1}$ ,  $L_{n+1} = R_n$

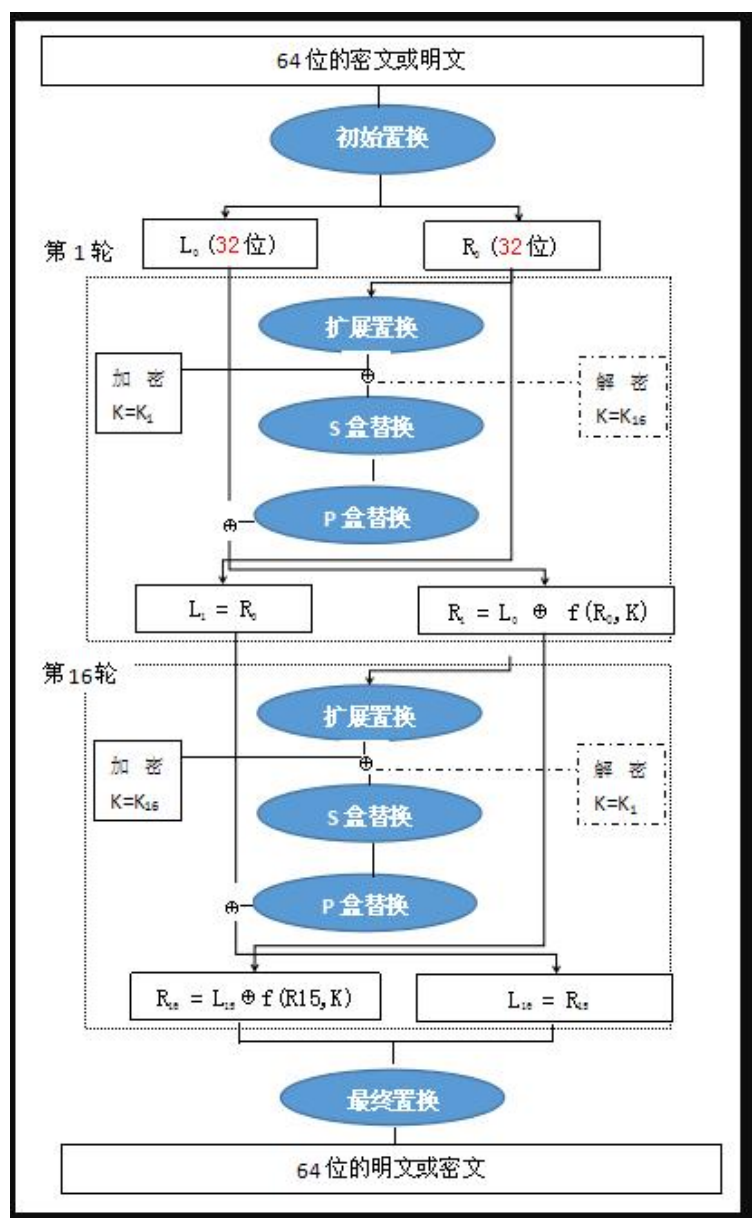
(3) 得到 L1R1 - L16R16

## 3. 生成密文

L16R16  $\rightarrow$  R16L16  $\rightarrow$  位置交换表置换得密文

## 4. 解密过程

解密过程基本与加密过程相同, 不过, 加密利用子钥的顺序为  $K_1 \sim K_{16}$ , 解密过程利用子钥的顺序为  $K_{16} \sim K_1$ 。



## 四、主要仪器设备及耗材

Windows 操作系统, DevCpp

## 五、实验步骤

1 代码编写, 主要代码如下:

```
1.          #include<stdio.h>
2.          #include<string.h>
3.          #include<stdlib.h>
4.          /*-----*/
5.              定义枚举型全局变量
6.          -----*/
7.          //typedef enum
8.          // {
9.          //     false = 0;
10.         //     true  = 1;
11.         // } bool;
12.
13.         // 十六轮子密钥
14.         static bool SubKey[16][48]={0};
15.
16.         /*-----*/
17.         /*-----*/
18.
19.         各种置换表
20.         -----*/
21.         // IP 置换表
22.         const char IP_Table[64]={
23.             58,50,42,34,26,18,10, 2,60,52,44,36,28,20,12, 4,
24.             62,54,46,38,30,22,14, 6,64,56,48,40,32,24,16, 8,
25.             57,49,41,33,25,17, 9, 1,59,51,43,35,27,19,11, 3,
26.             61,53,45,37,29,21,13, 5,63,55,47,39,31,23,15, 7
27.         };
28.         // IP-1 置换表
29.         const char IPR_Table[64]={
30.             40, 8,48,16,56,24,64,32,39, 7,47,15,55,23,63,31,
31.             38, 6,46,14,54,22,62,30,37, 5,45,13,53,21,61,29,
32.             36, 4,44,12,52,20,60,28,35, 3,43,11,51,19,59,27,
33.             34, 2,42,10,50,18,58,26,33, 1,41, 9,49,17,57,25
34.         };
35.         // E 扩展表
36.         static char E_Table[48]={
```

```

37.          32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,
38.          8, 9,10,11,12,13,12,13,14,15,16,17,
39.          16,17,18,19,20,21,20,21,22,23,24,25,
40.          24,25,26,27,28,29,28,29,30,31,32, 1
41.      };
42.      // PC1 置换表
43.      static char PC1_Table[56]={
44.          57,49,41,33,25,17, 9, 1,58,50,42,34,26,18,
45.          10, 2,59,51,43,35,27,19,11, 3,60,52,44,36,
46.          63,55,47,39,31,23,15, 7,62,54,46,38,30,22,
47.          14, 6,61,53,45,37,29,21,13, 5,28,20,12, 4
48.      };
49.
50.      // pc2 表
51.      static char PC2_Table[48]={
52.          14,17,11,24, 1, 5, 3,28,15, 6,21,10,
53.          23,19,12, 4,26, 8,16, 7,27,20,13, 2,
54.          41,52,31,37,47,55,30,40,51,34,33,48,
55.          44,49,39,56,34,53,46,42,50,36,29,32
56.      };
57.      // 移位表
58.      static char Move_Table[16]={
59.          1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
60.      };
61.      // S 盒
62.      static char S_Box[8][4][16]={
63.          //S1
64.          14, 4,13, 1, 2,15,11, 8, 3,10, 6,12, 5, 9, 0, 7,
65.          0,15, 7, 4,14, 2,13, 1,10, 6,12,11, 9, 5, 3, 8,
66.          4, 1,14, 8,13, 6, 2,11,15,12, 9, 7, 3,10, 5, 0,
67.          15,12, 8, 2, 4, 9, 1, 7, 5,11, 3,14,10, 0, 6,13,
68.          //S2
69.          15, 1, 8,14, 6,11, 3, 4, 9, 7, 2,13,12, 0, 5,10,
70.          3,13, 4, 7,15, 2, 8,14,12, 0, 1,10, 6, 9,11, 5,
71.          0,14, 7,11,10, 4,13, 1, 5, 8,12, 6, 9, 3, 2,15,
72.          13, 8,10, 1, 3,15, 4, 2,11, 6, 7,12, 0, 5,14, 9,
73.          //S3
74.          10, 0, 9,14, 6, 3,15, 5, 1,13,12, 7,11, 4, 2, 8,
75.          13, 7, 0, 9, 3, 4, 6,10, 2, 8, 5,14,12,11,15, 1,
76.          13, 6, 4, 9, 8,15, 3, 0,11, 1, 2,12, 5,10,14, 7,
77.          1,10,13, 0, 6, 9, 8, 7, 4,15,14, 3,11, 5, 2,12,
78.          //S4
79.          7,13,14, 3, 0, 6, 9,10, 1, 2, 8, 5,11,12, 4,15,
80.          13, 8,11, 5, 6,15, 0, 3, 4, 7, 2,12, 1,10,14, 9,

```

```

81.          10, 6, 9, 0,12,11, 7,13,15, 1, 3,14, 5, 2, 8, 4,
82.          3,15, 0, 6,10, 1,13, 8, 9, 4, 5,11,12, 7, 2,14,
83.          //S5
84.          2,12, 4, 1, 7,10,11, 6, 8, 5, 3,15,13, 0,14, 9,
85.          14,11, 2,12, 4, 7,13, 1, 5, 0,15,10, 3, 9, 8, 6,
86.          4, 2, 1,11,10,13, 7, 8,15, 9,12, 5, 6, 3, 0,14,
87.          11, 8,12, 7, 1,14, 2,13, 6,15, 0, 9,10, 4, 5, 3,
88.          //S6
89.          12, 1,10,15, 9, 2, 6, 8, 0,13, 3, 4,14, 7, 5,11,
90.          10,15, 4, 2, 7,12, 0, 5, 6, 1,13,14, 0,11, 3, 8,
91.          9,14,15, 5, 2, 8,12, 3, 7, 0, 4,10, 1,13,11, 6,
92.          4, 3, 2,12, 9, 5,15,10,11,14, 1, 7, 6, 0, 8,13,
93.          //S7
94.          4,11, 2,14,15, 0, 8,13, 3,12, 9, 7, 5,10, 6, 1,
95.          13, 0,11, 7, 4, 0, 1,10,14, 3, 5,12, 2,15, 8, 6,
96.          1, 4,11,13,12, 3, 7,14,10,15, 6, 8, 0, 5, 9, 2,
97.          6,11,13, 8, 1, 4,10, 7, 9, 5, 0,15,14, 2, 3,12,
98.          //S8
99.          13, 2, 8, 4, 6,15,11, 1,10, 9, 3,14, 5, 0,12, 7,
100.         1,15,13, 8,10, 3, 7, 4,12, 5, 6,11, 0,14, 9, 2,
101.         7,11, 4, 1, 9,12,14, 2, 0, 6,10,13,15, 3, 5, 8,
102.         2, 1,14, 7, 4,10, 8,13,15,12, 9, 0, 3, 5, 6,11
103.     };
104.     //P 置换表
105.     static char P_Table[32]={
106.         16, 7,20,21,29,12,28,17, 1,15,23,26, 5,18,31,10,
107.         2, 8,24,14,32,27, 3, 9,19,13,30, 6,22,11, 4,25
108.     };
109.     /*-----*/
110.     -----*/
111.     /*-----自定义函数
112.     -----*/
112.     void SetKey(char My_key[8]); //生成 16 轮的子密钥;
113.     void ByteToBit(bool * Data_out,char * Data_in,int Num); //字节转
        换成位;
114.     void Change_bit(bool * Data_out,int Num); //二进制的位置进行转
        换;
115.     void BitToByte(char My_message[8],bool * Message_in,int Num); /
        /位转换成字节;
116.     void TableReplace(bool *Data_out,bool *Data_in,const char *Tabl
        e,int Num); //各种表的置换算法;
117.     void Bitcopy(bool * Data_out,bool * Data_in,int Num); //二进制
        数组的拷贝

```

```

118.         void Loop_bit(bool * Data_out,int movstep,int len); //左移位;
119.         void Run_Des(char My_message[8],char HexMessage[16]); //des 的轮加
            密算法
120.         void Xor(bool * Message_out, bool * Message_in,int Num); //执行
            异或
121.         void S_change(bool * Data_out, bool * Data_in); // S 盒变换;
122.         void HexToBit(bool * Data_out,char * Data_in,int Num); // 十六进
            制转二进制
123.         void BitToHex(char * Data_out,bool * Data_in,int Num); //二进制
            转换成十六进制;
124.         void Run_desDes(char My_message[8],char HexMessage[16]); // DES
            轮解密算法;
125.
126.         /*-----*/
127.
128.         /*-----主函数
            -----*/
129.         int main()
130.         {
131.             int i=0,j;
132.             char My_key[8]={0}; //记录加密密钥;
133.             char You_key[8]={0}; //解密密钥
134.             char My_message[8]={0}; //明文
135.             char Message_hex[16]={0}; //16 进制的密文
136.             printf("请输入你要加密的内容(8 Byte):\n");
137.             gets(My_message);
138.             printf("请输入你的加密密钥:\n");
139.             gets(My_key);
140.             i=strlen(My_key);
141.             while(i!=8)
142.             {
143.                 printf("请输入加密密钥(8 Byte)\n");
144.                 gets(My_key);
145.                 i=0;
146.                 i=strlen(My_key);
147.             }
148.             SetKey(My_key); //生成 16 轮的加密子密钥;
149.             Run_Des(My_message,Message_hex); //des 的轮加密过程
150.             printf("经过加密的密文为:\n");
151.             for(i=0;i<16;i++)
152.             {
153.                 printf("%c ",Message_hex[i]);
154.             }
155.             printf("\n");

```

```

156.         printf("请输入你的解密密钥(8 Byte):\n");
157.         gets(You_key);
158.         i=strlen(You_key);
159.         while(i!=8)
160.         {
161.             printf("请输入解密密钥(8 Byte)\n");
162.             gets(You_key);
163.             i=0;
164.             i=strlen(You_key);
165.         }
166.         SetKey(You_key); //生成 16 轮的解密子密钥;
167.         Run_desDes(My_message,Message_hex);//解密;
168.         printf("解密结果为:\n");
169.         for(i=0;i<8;i++)
170.         {
171.             printf("%c ",My_message[i]);
172.         }
173.         printf("\n");
174.         return 0;
175.     }
176.
177.     /*-----具体函数定义-----*/
178.     void Bitcopy(bool * Data_out, bool * Data_in,int Num) //二进制数
组拷贝
179.     {
180.         int i=0;
181.         for(i=0;i<Num;i++)
182.         {
183.             Data_out[i]=Data_in[i];
184.         }
185.
186.     }
187.     void Change_bit(bool * Data_out,int Num) //二进制的位置进行转
换;
188.     {
189.         int i,j;
190.         static bool Temp[8]={0};
191.         for(i=0;i<Num/8;i++)
192.         {
193.             Bitcopy(Temp,Data_out,Num/8);
194.             for(j=0;j<Num/8;j++)
195.             {
196.                 Data_out[j]=Temp[Num/8-1-j];
197.             }

```

```

198.         Data_out+=Num/8;
199.     }
200. }
201. void ByteToBit( bool * Data_out,char * Data_in,int Num) //字节转
    位
202. {
203.     int i,j;
204.     for(i=0;i<Num;i++)
205.     {
206.         Data_out[i]=(Data_in[i/8]>>(i%8))&0x01;
207.     }
208.     //Change_bit(Data_out,Num);
209. }
210. void BitToHex(char * Data_out, bool * Data_in,int Num) //二进制
    转十六进制
211. {
212.     int i;
213.     for(i=0;i<Num/4;i++)
214.     {
215.         Data_out[i]=0;
216.     }
217.     for(i=0;i<Num/4;i++)
218.     {
219.         Data_out[i]=Data_in[4*i]+Data_in[4*i+1]*2+Data_in[4*i+2]*4+Da
            ta_in[4*i+3]*8;
220.         if(Data_out[i]%16>9)
221.         {
222.             Data_out[i]=Data_out[i]%16+'7';
223.         }
224.         else
225.             Data_out[i]=Data_out[i]%16+'0';
226.     }
227. }
228. void HexToBit(bool * Data_out,char * Data_in,int Num) //十六进制
    转二进制
229. {
230.     int i;
231.     for(i=0;i<Num;i++)
232.     {
233.         if(Data_in[i/4]<='9')
234.         {
235.             Data_out[i]=((Data_in[i/4]-'0')>>(i%4))&0x01;
236.         }
237.         else

```



```

238.         {
239.             Data_out[i]=((Data_in[i/4]- '7')>>(i%4))&0x01;
240.         }
241.     }
242. }
243. void BitToByte(char My_message[8],bool * Message_in,int Num) //
    位转换成字节
244. {
245.     int i=0;
246.     for(i=0;i<(Num/8);i++)
247.     {
248.         My_message[i]=0;
249.     }
250.     for(i=0;i<Num;i++)
251.     {
252.         My_message[i/8]|=Message_in[i]<<(i%8);
253.     }
254. }
255. void TableReplace( bool *Data_out, bool * Data_in,const char *T
    able ,int Num) // 置换算法
256. {
257.     int i=0;
258.     static bool Temp[256]={0};
259.     for(i=0;i<Num;i++)
260.     {
261.         Temp[i]=Data_in[Table[i]-1];
262.     }
263.     Bitcopy(Data_out,Temp,Num);
264. }
265. void Loop_bit(bool * Data_out,int movstep,int len)
266. {
267.     static bool Temp[256]={0};
268.     Bitcopy(Temp,Data_out,movstep);
269.     Bitcopy(Data_out,Data_out+movstep,len-movstep);
270.     Bitcopy(Data_out+len-movstep,Temp,movstep);
271.     /*Temp=Data_out;
272.     Temp[movstep]='\0';
273.     Data_out=Data_out+movstep;
274.     Data_out+(len-movstep)=Temp;*/
275. }
276. void Xor(bool * Message_out,bool * Message_in,int Num)//执行异
    或
277. {
278.     int i;

```

```

279.         for(i=0;i<Num;i++)
280.         {
281.             Message_out[i]=Message_out[i]^Message_in[i];
282.         }
283.     }
284.     void SetKey(char My_key[8])
285.     {
286.         int i,j;
287.         static bool Key_bit[64]={0}; //Key 的二进制缓存;
288.         static bool *Key_bit_L,*Key_bit_R;
289.         Key_bit_L=&Key_bit[0]; //key 的左边 28 位;
290.         Key_bit_R=&Key_bit[28]; //key 的右边 28 位;
291.         ByteToBit(Key_bit,My_key,64);
292.         /* Change_bit(Key_bit,64) ;//二进制的位置进行转换;
293.         for(i=0;i<64;i++)
294.         {
295.             printf("%d ",Key_bit[i]);
296.         }
297.         printf("\n");
298.         printf("\n");*/
299.         TableReplace(Key_bit,Key_bit,PC1_Table,56); //pc-1 置换
300.         for(i=0;i<16;i++)
301.         {
302.             Loop_bit(Key_bit_L,Move_Table[i],28);
303.             Loop_bit(Key_bit_R,Move_Table[i],28);
304.             TableReplace(SubKey[i],Key_bit,PC2_Table,48); //pc-2 置换
305.         }
306.     }
307.     void S_change(bool * Data_out, bool * Data_in) //S 盒变换
308.     {
309.         int i;
310.         int r=0,c=0; //S 盒的行和列;
311.         for(i=0;i<8;i++,Data_in=Data_in+6,Data_out=Data_out+4)
312.         {
313.             r=Data_in[0]*2+Data_in[5]*1;
314.             c=Data_in[1]*8+Data_in[2]*4+Data_in[3]*2+Data_in[4]*1;
315.             ByteToBit(Data_out,&S_Box[i][r][c],4);
316.         }
317.     }
318.     void F_change(bool Data_out[32],bool Data_in[48]) // f 函数;
319.     {
320.         int i;
321.         static bool Message_E[48]={0}; //存放 E 置换的结果;
322.         TableReplace(Message_E,Data_out,E_Table,48); //E 表置换

```

```

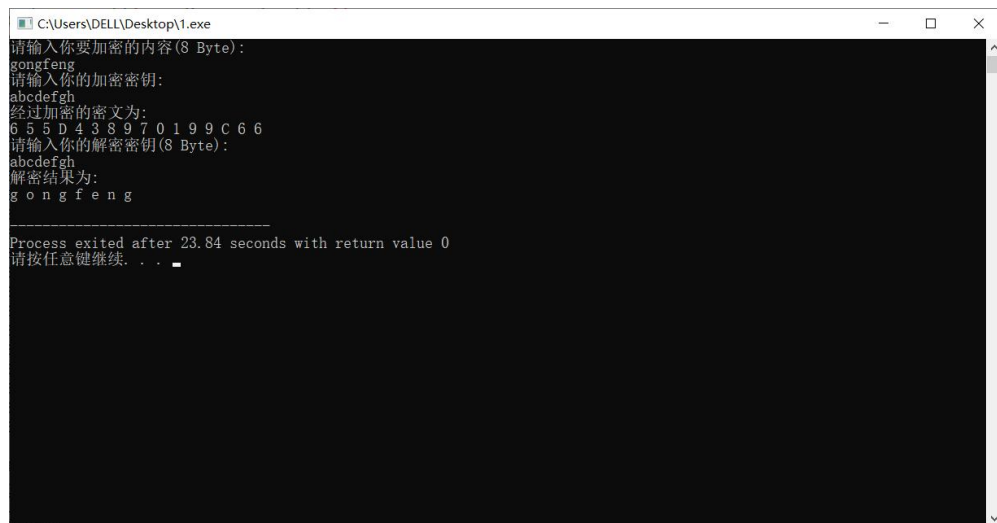
323.         Xor(Message_E,Data_in,48);
324.         S_change(Data_out,Message_E);           // S 盒变换
325.         TableReplace(Data_out,Data_out,P_Table,32); //P 置换
326.     }
327.     void Run_Des(char My_message[8],char HexMssage[16])//des 轮加密算
        法:
328.     {
329.         int i;
330.         static bool Message_bit[64]={0};
331.         static bool *Message_bit_L=&Message_bit[0],*Message_bit_R=&Mes
            sage_bit[32];
332.         static bool Temp[32]={0};
333.         ByteToBit(Message_bit,My_message,64);
334.         /*Change_bit(Message_bit,64) ;//二进制的位置进行转换;
335.         for(i=0;i<64;i++)
336.         {
337.             printf("%d ",Message_bit[i]);
338.         }
339.         printf("\n");
340.         printf("\n");*/
341.         TableReplace(Message_bit,Message_bit,IP_Table,64);
342.         for(i=0;i<16;i++)
343.         {
344.             Bitcopy(Temp,Message_bit_R,32);
345.             F_change(Message_bit_R,SubKey[i]);
346.             Xor(Message_bit_R,Message_bit_L,32);
347.             Bitcopy(Message_bit_L,Temp,32);
348.         }
349.         TableReplace(Message_bit,Message_bit,IPR_Table,64);
350.         BitToHex(HexMssage,Message_bit,64);//二进制转换成十六进制;
351.     }
352.     void Run_desDes(char My_message[8],char HexMessage[16])// DES 轮
        解密算法:
353.     {
354.         int i=0;
355.         static bool Message_bit[64]={0};
356.         static bool * Message_bit_L=&Message_bit[0], * Message_bit_R=&
            Message_bit[32];
357.         static bool Temp[32]={0};
358.         HexToBit(Message_bit,HexMessage,64);
359.         TableReplace(Message_bit,Message_bit,IP_Table,64);
360.         for(i=15;i>=0;i--)
361.         {
362.             Bitcopy(Temp,Message_bit_L,32);

```

```
363.          F_change(Message_bit_L,SubKey[i]);
364.          Xor(Message_bit_L,Message_bit_R,32);
365.          Bitcopy(Message_bit_R,Temp,32);
366.      }
367.      TableReplace(Message_bit,Message_bit,IPR_Table,64);
368.      BitToByte(My_message,Message_bit,64);
369.  }
```

## 六、实验数据及处理结果

运行结果如图 2:



```
C:\Users\DELL\Desktop\1.exe
请输入你要加密的内容(8 Byte):
gongfeng
请输入你的加密密钥:
abcdefgh
经过加密的密文为:
6 5 5 D 4 3 8 9 7 0 1 9 9 C 6 6
请输入你的解密密钥(8 Byte):
abcdefgh
解密结果为:
g o n g f e n g

-----
Process exited after 23.84 seconds with return value 0
请按任意键继续. . .
```

图 2

## 七、思考讨论题或体会或对改进实验的建议

无

## 八、参考资料

现代密码学第 4 版