DES 算法 C 语言实现

18342066 鲁沛

一、 算法原理概述

DES 是一种典型的块加密方法：它以 64 位为分组长度，明文被分割成 64 位一组作为算法的输入，经过一系列计算过程，输出同样 64 位长度的密文。DES 使用加密密钥定义变换过程，因此被认为只有持有加密密钥的用户才能解密密文。DES 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。其中极少量的组合被认为是弱密钥，但能容易地避开它们。

二、 总体结构

原始明文：任意长度的字符串，算法得到输入后将其转换为 8 个字节的分组（64 位二进制）。其最后的分组不够 8 个字节(64 位)时，按 PKCS#5 (RFC 8018)规范进行字节填充：在末尾以字节填满，填入的字节取值相同，都是填充的字节数目；原始明文消息刚好分组完全时，在明文末尾额外填充 8 个字节(即增加一个完整分组)，每个字节取值都是 08。

分组结构：$M = m_1 m_2 \dots m_{64}$，$m_i \in \{0,1\}$，$i = 1 \dots 64$。

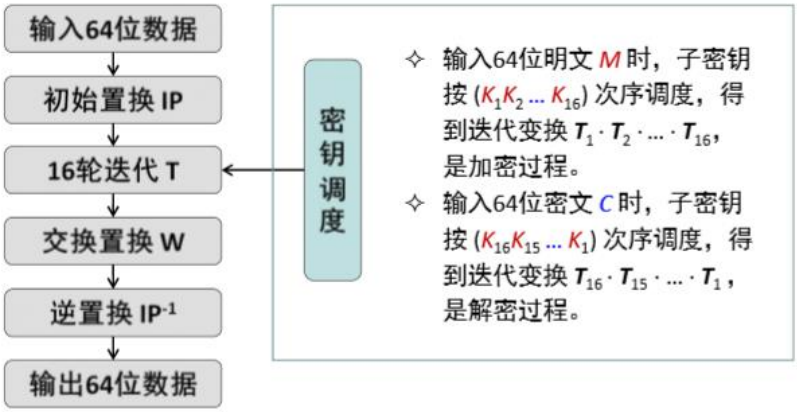
密钥：64 位长二进制数，以 8 位 16 进制数输入。

$K = k_1 k_2 \dots k_{64}$，$k_i \in \{0,1\}$，$i = 1 \dots 64$。共 8 位奇偶校验位，在密钥中起作用的仅 56 位。算法过程将会用到由密钥 K 生成的 16 个子密钥，分别记为$K_1, K_2, \dots, K_{16}$。

密文：$C = c_1 c_2 \dots c_{64}$，$c_i \in \{0,1\}$，$i = 1 \dots 64$。以 16 进制输出。

加密过程：$C = E_k(M) = [IP \cdot T_1 \cdot T_2 \cdot \dots \cdot T_{16} \cdot W \cdot IP^{-1}]$
输入 64 位明文 M 时，子密钥按$(K_1, K_2, \dots, K_{16})$ 次序调度，得到迭代变换$T_1 \cdot T_2 \cdot \dots \cdot T_{16}$。

解密过程：$M = D_k(C) = [IP \cdot T_{16} \cdot T_{15} \cdot \dots \cdot T_1 \cdot W \cdot IP^{-1}]$
输入 64 位密文 C 时，子密钥按$(K_{16}, K_{15}, \dots, K_1)$ 次序调度，得到迭代变换$T_{16} \cdot T_{15} \cdot \dots \cdot T_1$。



三、 模块分解

1. **读取一行未知长度的字符串，read()函数**

   算法：先假设输入字符串长度不超过 10，以 10 个字符为大小用 malloc 函数分配内存空间，如果读完 10 个字符还没读到换行符，则每次用 realloc 函数两倍的扩展分配的内存空间。全部读完之后将字符串的最后一位置'\0'。

2. **将明文分组并填充，group()函数和 fill()函数**

   算法：一个字符是 8 位（1 字节），于是只要将字符串长度除以 8 取余就可知需要填充几个字节，将字符串长度除以 8 加 1 即可知道最终会有几个分组。然后按照 PKCS#5 (RFC 8018)规范用一个 switch 语句根据余数的值填充即可。

3. **初始置换 IP，IP_displace()函数**

   给定 64 位明文块 M，通过一个固定的初始置换 IP 重排 M 中的二进制位，得到二进制串M_0= IP(M) = L_0R_0，这里L_0和R_0分别是M_0的前 32 位和后 32 位。下表是 IP 置换后的下标编号序列。

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

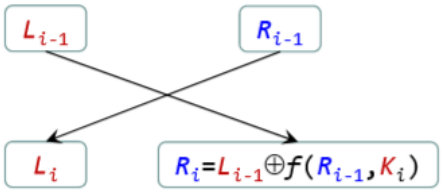
L_0 R_0

   算法：将 IP 置换表存为一维整形数组 IP，置换后的每一位为以 IP 数组中对应位的数值为下标在 M 中取出的数值。后续所有的矩阵置换都用同样的思路实现，后面不会再赘述。

4. **迭代序列 T，T_iteration()函数**

   根据L_0R_0按下述规则进行 16 轮迭代$T_1 \cdot T_2 \cdot \dots \cdot T_{16}$，即

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



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$L_{16}R_{16}$
异或函数算法：由于 DES 还要用到 48 位的异或，所以直接设计一个可以计算 32 位和 48 位异或的 xor_()函数，根据参数的不同，循环次数不同，每一项分别作异或运算即可。

5. **Feistel 轮函数$f(R_{i-1}, K_i)$**

1) 将长度为 32 位的串R_{i-1}作 E-扩展，得到一个 48 位的串$E(R_{i-1})$;

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

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;

S 盒介绍，S_boxing(()函数：
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$S_1, \cdots, S_8$，每个 S-盒是一个 4 行(编号十进制数 0-3)、16 列(编号十进制数 0-15) 的二维表，表中每个元素是一个十进制数，取值在 0-15 之间，用于表示一个 4 位二进制数。假设S_i的 6 位二进制输入为$b_1b_2b_3b_4b_5b_6$，则由 n=$(b_1b_6)_{10}$确定行号，由 m= $(b_2b_3b_4b_5)_{10}$确定列号，S_i[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	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

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
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

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(3)得到的分组结果顺序连接得到长度为 32 位的串;

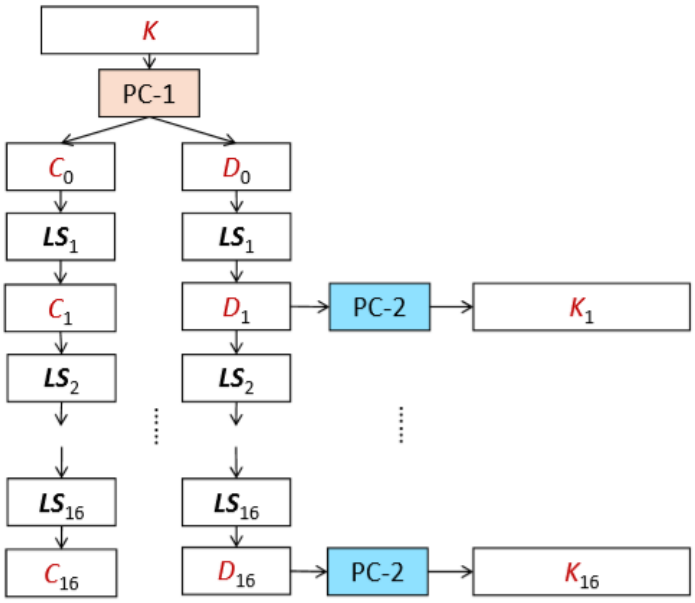
5) 将(4)的结果经过 P-置换, 得到的结果作为轮函数$f(R_{i-1}, K_i)$的最终 32 位输出。P 置换如下:

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

6. 子密钥的生成，get_sub_key()函数

根据给定的 64 位密钥 K, 生成 16 个 48 位的子密钥K_1-K_{16}, 供 Feistel 轮函数$f(R_{i-1}, K_i)$调用。

总体流程图如下:



1) 对 K 的 56 个非校验位实行置换 PC-1，得到C_0D_0，其中C_0和D_0分别由 PC-1 置换结果的前 28 位和后 28 位组成。

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

注意到密钥 K 的 8个校验位的下标不参与置换。

2) 计算$C_i = LS_i(C_{i-1})$和$D_i = LS_i(D_{i-1})$，当 i=1, 2, 9, 16 时，$LS_i(A)$表示将二进制串 A 循环左移一个位置；否则循环左移两个位置。用到 LS()函数。

算法：左移一位时，先用 temp 变量将 A[0]记录下来，然后循环 27 次，每次将 A[i+1]赋给 A[i]，最后将 temp 赋给 A[27]；左移两位时，先分别用 temp0 和 temp1 将 A[0 和 A[1]记录下来，然后循环 26 次，每次将 A[i+2]赋给 A[i]，最后将 temp0 赋给 A[26]，将 temp1 赋给 A[27]。

3) 对 56 位的C_iD_i压缩后实行 PC-2 置换，得到 48 位的K_i，i= i+1。

PC-2 压缩置换：去掉第 9, 18, 22, 25, 35, 38, 43, 54 位，将剩下的 48 位按照 PC-2 置换表作置换，得到K_i。

PC-2 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

4) 如果已经得到K_{16}，密钥生成过程结束；否则转(2)。

7. **加密和解密，encryption_or_decryption()函数**
   前文已介绍过加密和解密的流程是一样的，只是调用子密钥的次序相反，因此可以用同一个函数来实现，只是传入的参数不同。具体代码实现按照流程调用已有函数即可。

8. **辅助函数**
   bin_to_hex()函数，用于将 2 进制转换为 16 进制。
   hex_to_bin()函数，用于将 16 进制转换为 2 进制。
   bin_to_dec()函数，用于将 2 进制转换为 10 进制。

四、 数据结构设计

- 2 进制用一维整形数组存储，每一项为 0 或为 1；
- 8 个 S 盒用三维整形数组来存储，其余置换矩阵全部用一维整形数组存储；
- 16 进制用字符串来存储。

五、 C 语言源代码

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

const int IP[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
};//初始置换 IP

const int IP_1[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
};//逆置换 IP_1

const int E[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

};//E 扩展

const int PC_1[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_1,生成子密钥

const 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
};//PC_2,生成子密钥

const int S[8][4][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
}; //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 S1, …, S8。

const 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
};//P 置换

void IP_displace(int block[64], int left[32], int right[32])
{
    int temp[64];
    for(int i = 0; i < 64; ++i)
        temp[i] = block[IP[i] - 1];
    for(int i = 0; i < 32; ++i)
    {
        left[i] = temp[i];
        right[i] = temp[i + 32];
    }
}

void IP_1_displace(int before[], int after[])
{
    for(int i = 0; i < 64; ++i)
        after[i] = before[IP_1[i] - 1];
```

```
}

void PC_1_displace(int key[], int C[], int D[])
{
    int temp[56];
    for(int i = 0; i < 56; ++i)
        temp[i] = key[PC_1[i] - 1];
    for(int i = 0; i < 28; ++i)
    {
        C[i] = temp[i];
        D[i] = temp[i + 28];
    }
}

void PC_2_displace(int CD[], int sub_key[])
{
    for(int i = 0; i < 48; ++i)
        sub_key[i] = CD[PC_2[i] - 1];
}

void P_displace(int before_p[], int after_p[])
{
    for(int i = 0; i < 32; ++i)
        after_p[i] = before_p[P[i] - 1];
}

void bin_to_hex(int bin[][64], char hex[], int block_num)//2 进制转换 16 进制
{
    int count = 0;
    for(int i = 0; i < block_num; ++i)
    {
        for(int j = 0; j < 64; ++j)
        {
            if((j + 1) % 4 == 0)
            {
                int temp = bin[i][j] + bin[i][j - 1] * 2 + bin[i][j - 2] * 4 + bin[i][j - 3] * 8;
                if(temp > 9)
                    hex[count] = temp - 10 + 'A';
                else
                    hex[count] = temp + '0';
                ++count;
            }
```

```
187.         }
188.     }
189.     hex[count] = '\0';
190. }
191.
192. void hex_to_bin(char * hex, int bin[])//16 进制转换 2 进制
193. {
194.     int length = strlen(hex);
195.     for(int i = 0; i < length; ++i)
196.     {
197.         int temp;
198.         if(hex[i] > 'A' || hex[i] == 'A')
199.             temp = hex[i] - 'A' + 10;
200.         else
201.             temp = hex[i] - '0';
202.         for(int j = 0; j < 4; ++j)
203.         {
204.             bin[(i + 1) * 4 - 1 - j] = temp % 2;
205.             temp = temp / 2;
206.         }
207.     }
208. }
209.
210. void bin_to_dec(int block[][64], int a[], int block_num)//2 进制转换 10 进制
211. {
212.     int count = 0;
213.     for(int i = 0; i < block_num; ++i)
214.     {
215.         for(int j = 0; j < 64; j = j + 8)
216.         {
217.             a[count] = block[i][j] * 128 + block[i][j + 1] * 64 + block[i][
    j + 2] * 32 + block[i][j + 3] * 16
218.                         + block[i][j + 4] * 8 + block[i][j + 5] * 4 + block[
    i][j + 6] * 2 + block[i][j + 7] * 1;
219.             ++count;
220.          }
221.     }
222. }
223.
224. void xor_(int t1[], int t2[], int res[], int mode)//需要用到 32 位异或和 48 位
    异或
225. {
226.     for(int i = 0; i < mode; ++i)
227.         res[i] = t1[i] ^ t2[i];
```

```
}

void fill(int binary[], int start, int remainder)//按照 PKCS#5 (RFC 8018）规范进行字节填充
{
    switch (remainder)
    {
    case 0:
        for (int i = 0; i < 8; ++i)
        {
            for (int j = 0; j < 8; ++j)
                binary[start + (8 * (i + 1)) - 1 - j] = (8 >> j) & 1;
        }
        break;
    case 1:
        for (int i = 0; i < 7; ++i)
        {
            for (int j = 0; j < 8; ++j)
                binary[start + (8 * (i + 1)) - 1 - j] = (7 >> j) & 1;
        }
        break;
    case 2:
        for (int i = 0; i < 6; ++i)
        {
            for (int j = 0; j < 8; ++j)
                binary[start + (8 * (i + 1)) - 1 - j] = (6 >> j) & 1;
        }
        break;
    case 3:
        for (int i = 0; i < 5; ++i)
        {
            for (int j = 0; j < 8; ++j)
                binary[start + (8 * (i + 1)) - 1 - j] = (5 >> j) & 1;
        }
        break;
    case 4:
        for (int i = 0; i < 4; ++i)
        {
            for (int j = 0; j < 8; ++j)
                binary[start + (8 * (i + 1)) - 1 - j] = (4 >> j) & 1;
        }
        break;
    case 5:
        for (int i = 0; i < 3; ++i)
```

```
271.         {
272.             for (int j = 0; j < 8; ++j)
273.                 binary[start + (8 * (i + 1)) - 1 - j] = (3 >> j) & 1;
274.         }
275.         break;
276.     case 6:
277.         for (int i = 0; i < 2; ++i)
278.         {
279.             for (int j = 0; j < 8; ++j)
280.                 binary[start + (8 * (i + 1)) - 1 - j] = (2 >> j) & 1;
281.         }
282.         break;
283.     case 7:
284.         for (int i = 0; i < 1; ++i)
285.         {
286.             for (int j = 0; j < 8; ++j)
287.                 binary[start + (8 * (i + 1)) - 1 - j] = (1 >> j) & 1;
288.         }
289.         break;
290.     }
291. }
292.
293. void group(int block[][64], char *s, int block_num, char mode) //分组函数,将明文分成 8 个字节一组
294. {
295.     int remainder;//判断需要填充的字节个数
296.     int str_len = strlen(s);
297.     int binary[block_num * 64];//先将字符串转换成二进制并将其填充，再对其分组
298.     if(mode == 'E')//加密模式下对 char 型的字符作二进制转换
299.     {
300.         remainder = str_len % 8;
301.         for (int i = 0; i < str_len; ++i)
302.         {
303.             for (int j = 0; j < 8; ++j)
304.                 binary[(8 * (i + 1)) - 1 - j] = (s[i] >> j) & 1;
305.         }
306.         fill(binary, str_len * 8, remainder);
307.     }
308.     else//解密模式下对 16 进制的字符作二进制转换
309.         hex_to_bin(s, binary);//无需填充
310.     for(int i = 0; i < block_num; ++i)
311.     {
312.         for(int j = 0; j < 64; ++j)
313.             block[i][j] = binary[i * 64 + j];
```

```
314.     }
315. }
316. 
317. char *read()//读入一行字符串
318. {
319.     int str_len = 10; //假设输入字符串长度不超过 10
320.     char *input = (char *)malloc(sizeof(char) * str_len);
321.     char c;
322.     int count = 0;
323.     while ((c = getchar()) != '\n')
324.     {
325.         ++count;
326.         if (count > str_len)
327.         {
328.             str_len = str_len * 2; //每次不够用了就两倍的扩展内存空间
329.             input = (char *)realloc(input, sizeof(char) * str_len);
330.         }
331.         input[count - 1] = c;
332.     }
333.     input[count] = '\0';
334.     return input;
335. }
336. 
337. void LS(int A[], int flag)
338. {
339.     if((flag == 1) || (flag == 2) || (flag == 9) || (flag == 16))
340.     {
341.         int temp = A[0];
342.         for(int i = 0; i < 27; ++i)
343.             A[i] = A[i + 1];
344.         A[27] = temp;
345.     }
346.     else
347.     {
348.         int temp0 = A[0];
349.         int temp1 = A[1];
350.         for(int i = 0; i < 26; ++i)
351.             A[i] = A[i + 2];
352.         A[26] = temp0;
353.         A[27] = temp1;
354.     }
355. }
356. 
357. void S_boxing(int bit_48[], int bit_32[])//S 盒函数
```

```
{
    int bit_6[8][6];
    int bit_4[8][4];
    for(int i = 0; i < 8; ++i)
    {
        for(int j = 0; j < 6; ++j)
            bit_6[i][j] = bit_48[i * 6 + j];
        int n = 2 * bit_6[i][0] + bit_6[i][5];
        int m = 8 * bit_6[i][1] + 4 * bit_6[i][2] + 2 * bit_6[i][3] + bit_6[i][4];
        int temp = S[i][n][m];
        for(int j = 0; j < 4; ++j)
            bit_4[i][ 3 - j] = (temp >> j) & 1;
    }
    for(int i = 0; i < 8; ++i)
    {
        for(int j = 0; j < 4; ++j)
            bit_32[i * 4 + j] = bit_4[i][j];
    }
}

void get_sub_key(int key[], int sub_key[16][48])//将密钥分成 16 个子密钥
{
    int C[28], D[28];
    PC_1_displace(key, C, D);
    for(int i = 0; i < 16; ++i)
    {
        LS(C, i + 1);
        LS(D, i + 1);
        int CD[56];
        for(int i = 0; i < 28; ++i)
        {
            CD[i] = C[i];
            CD[i + 28] = D[i];
        }
        PC_2_displace(CD, sub_key[i]);
    }
}

void Feistel(int right[], int sub_key[], int res[])
{
    int E_R[48];
    for(int i = 0; i < 48; ++i)
        E_R[i] = right[E[i] - 1];
```

```
401.     xor_(E_R, sub_key, E_R, 48);
402.     int after_s[32];
403.     S_boxing(E_R, after_s);
404.     P_displace(after_s, res);
405. }
406. 
407. void T_iteration(int left[32], int right[32], int sub_key[16][48])//T 迭代
408. {
409.     int temp_right[32];
410.     int temp_left[32];
411.     for(int i = 0; i < 16; ++i)
412.     {
413.         for(int i = 0; i < 32; ++i)
414.         {
415.             temp_right[i] = right[i];
416.             temp_left[i] = left[i];
417.             left[i] = right[i];
418.         }
419.         int after_f[32];
420.         Feistel(right, sub_key[i], after_f);
421.         xor_(temp_left, after_f, right, 32);
422.     }
423. }
424. 
425. void encryption_or_decryption(char * input, int output[][64], int sub_key[1
    6][48], int block_num, int mode)//加密和解密用同一个函数实现
426. {
427.     int block[block_num][64];
428.     group(block, input, block_num, mode);
429.     for(int i = 0; i < block_num; ++i)
430.     {
431.         int left[32], right[32];
432.         IP_displace(block[i], left, right);
433.         T_iteration(left, right, sub_key);
434.         int temp[64];
435.         for(int j = 0; j < 32; ++j)//交换置换 W
436.         {
437.             temp[j] = right[j];
438.             temp[j + 32] = left[j];
439.         }
440.         IP_1_displace(temp, output[i]);
441.     }
442. }
443. 
```

```
int main()
{
    int key[64], sub_key[16][48], block_num;
    char mode;
    char *input, *key_16;
    printf("choose your mode, encryption/E, decryption/D: ");
    scanf("%c", &mode);
    getchar();
    printf("input your plaintext/ciphertext: ");
    input = read();
    printf("input your key: ");
    key_16 = read();
    hex_to_bin(key_16, key);//key 作为 16 进制读入后先转换为二进制
    get_sub_key(key, sub_key);
    if(mode == 'E')
    {
        block_num = (strlen(input) / 8) + 1;
        char ciphertext[block_num * 16 + 1];
        int output[block_num][64];
        encryption_or_decryption(input, output, sub_key, block_num, mode);

        bin_to_hex(output, ciphertext, block_num);
        printf("%s", ciphertext);
    }
    else
    {
        block_num = (strlen(input) / 16);
        int plaintext[block_num * 8];
        int output[block_num][64], reverse_sub_key[16][48];
        for(int i = 0; i < 16; ++i)
        {
            for(int j = 0; j < 48; ++j)
                reverse_sub_key[i][j] = sub_key[15 - i][j];
        }
        encryption_or_decryption(input, output, reverse_sub_key, block_num, mode);
        bin_to_dec(output, plaintext, block_num);
        int fill = output[block_num- 1][60] * 8 + output[block_num - 1][61] * 4 + output[block_num - 1][62] * 2 + output[block_num - 1][63];
        for(int i = 0; i < (block_num * 8) - fill; ++i)
            printf("%c", plaintext[i]);
    }
    free(input);
    free(key_16);
```

```
485. }
```

六、 编译运行结果

明文：I love you!

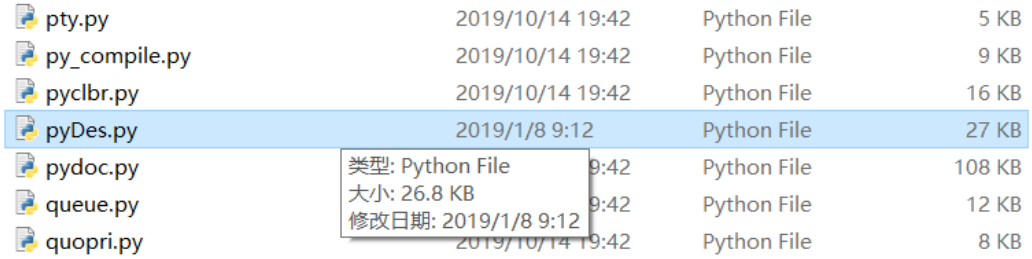
密钥：4142434445464748

自己实现的输出：

```
PS C:\Users\18052\Desktop\DES> ./a
choose your mode, encryption/E, decryption/D: E
input your plaintext/ciphertext: I love you!
input your key: 4142434445464748
3C2DF70C25C4B55349655C5D78AF21D5
PS C:\Users\18052\Desktop\DES> ./a
choose your mode, encryption/E, decryption/D: D
input your plaintext/ciphertext: 3C2DF70C25C4B55349655C5D78AF21D5
input your key: 4142434445464748
I love you!
PS C:\Users\18052\Desktop\DES>
```

用 python 中的 pyDES 库来验证程序的正确性

安装过程：去 https://github.com/twhiteman/pyDes 下载源码并解压后后将 pyDES.py 复制到 python 的 lib 文件夹下，即可使用 python 的 DES 库函数。



python 代码如下：

```
import base64
import binascii
from pyDes import *

def _example_des_():
        from time import time

        # example of DES encrypting in CBC mode with the IV of "\0\0\0\0\0\0\0\0
        print ("Example of DES encryption using CBC mode\n")
        k = des("ABCDEFGH", ECB, "\0\0\0\0\0\0\0\0")

        data = "I love you!"

        print ("Key      : 4142434445464748")
        print ("Data     : %r" % data)

        d = k.encrypt(data, padmode=PAD_PKCS5)
        ret = binascii.b2a_hex(d)
        print ("Encrypted: %r" % ret)

        d = k.decrypt(d)
        print ("Decrypted: %r" % d)

_example_des_()
```

输出如下：

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:\Users\18052\Desktop\DES\test_in_python.py ============
Example of DES encryption using CBC mode

Key      : 4142434445464748
Data     : 'I love you!'
Encrypted: b'3c2df70c25c4b55349655c5d78af21d5'
Decrypted: b'I love you!\x05\x05\x05\x05\x05'
```

可以看到与我自己的实现一致，则 DES 实现正确。