DES 加密设计

2015 级软件工程计应　　15331151　　李佳

一、算法原理概述

DES 的原始思想可以参照二战德国的恩格玛机，其基本思想大致相同。传统的密码加密都是由古代的循环移位思想而来，恩格玛机在这个基础之上进行了扩散模糊。但是本质原理都是一样的。现代 DES 在二进制级别做着同样的事:替代模糊，增加分析的难度。美国国家标准局 1973 年开始研究除国防部外的其它部门的计算机系统的数据加密标准，于 1973 年 5 月 15 日和 1974 年 8 月 27 日先后两次向公众发出了征求加密算法的公告。加密算法要达到的目的（通常称为 DES 密码算法要求）主要为以下四点：

1.提供高质量的数据保护，防止数据未经授权的泄露和未被察觉的修改；

2.具有相当高的复杂性，使得破译的开销超过可能获得的利益，同时又要便于理解和掌握；

3.DES 密码体制的安全性应该不依赖于算法的保密，其安全性仅以加密密钥的保密为基础；

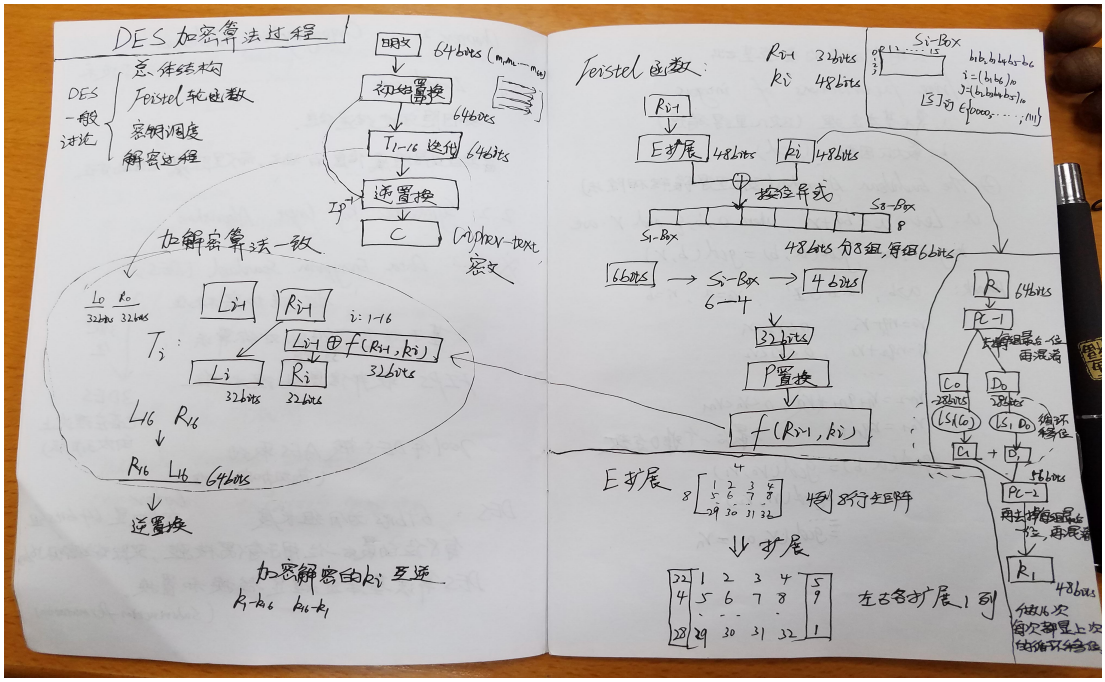
4.实现经济，运行有效，并且适用于多种完全不同的应用。

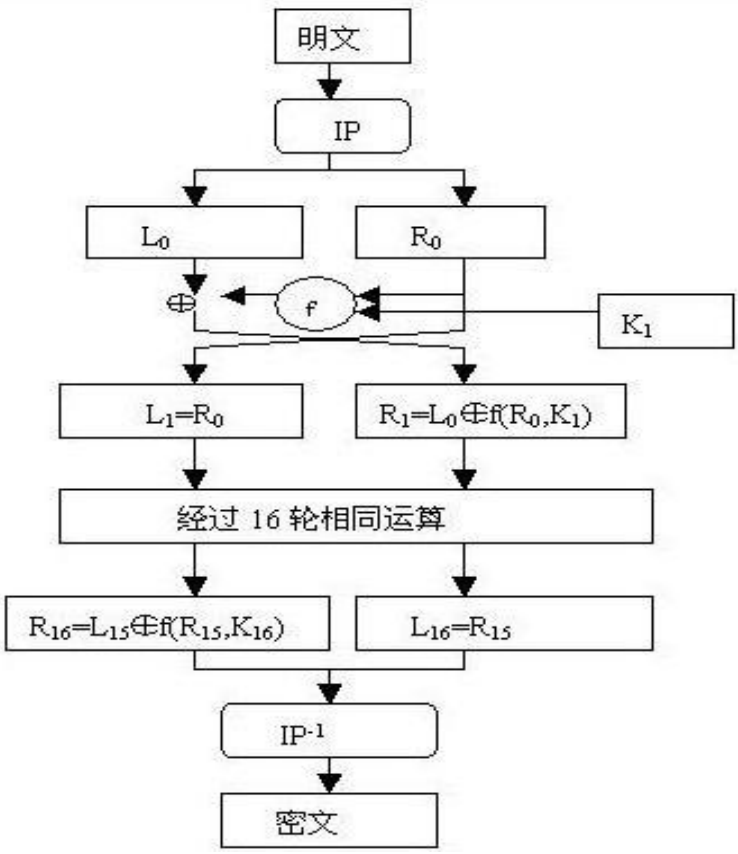
1977 年 1 月，美国政府颁布：采纳 IBM 公司设计的方案作为非机密数据的正式数据加密标准（DES）。

DES 算法的入口参数有三个：Key、Data、Mode。其中 Key 为 8 个字节共 64 位，是 DES 算法的工作密钥；Data 也为 8 个字节 64 位，是要被加密或被解密的数据；Mode 为 DES 的工作方式，有两种：加密或解密。 DES 算法是这样工作的：如 Mode 为加密，则用 Key 去把数据 Data 进行加密， 生成 Data 的密码形式（64 位）作为 DES 的输出结果；如 Mode 为解密，则用 Key 去把密码形式的数据 Data 解密，还原为 Data 的明码形式（64 位）作为 DES 的输出结果。在通信网络的两端，双方约定一致的 Key，在通信的源点用 Key 对核心数据进行 DES 加密，然后以密码形式在公共通信网（如电话网）中传输到通信网络的终点，数据到达目的地后，用同样的 Key 对密码数据进行解密，便再现了明码形式的核心数据。这样，便保证了核心数据（如 PIN、MAC 等）在公共通信网中传输的安全性和可靠性。 通过定期在通信网络的源端和目的端同时改用新的 Key，便能更进一步提高数据的保密性，这正是现在金融交易网络的流行做法。

DES 使用一个 56 位的密钥以及附加的 8 位奇偶校验位，产生最大 64 位的分组大小。这是一个迭代的分组密码，使用称为 Feistel 的技术，其中将加密的文本块分成两半。使用子密钥对其中一半应用循环功能，然后将输出与另一半进行"异或"运算;接着交换这两半，这一过程会继续下去，但最后一个循环不交换。DES 使用 16 个循环，使用异或，置换，代换，移位操作四种基本运算。

二、总体结构





DES 加密的总体结构如上两图所示，明文经过初始置换后进入 16 次迭代，迭代中使用 Feistel 轮函数，然后通过逆置换后得到密文。解密过程使用的结构和算法与加密过程相同，只是子密钥 ki 的使用顺序正好相反。

三、模块分解

DES 算法把 64 位的明文输入块变为 64 位的密文输出块，它所使用的密钥也是 64 位，整个算法的主要置换模块如下：

(1)IP 置换

其功能是把输入的 64 位数据块按位重新组合，并把输出分为 L0、R0 两部分，每部分各长 32 位，其置换规则如下：

58,50,12,34,26,18,10,2,60,52,44,36,28,20,12,4,

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,

即将输入的第 58 位换到第一位，第 50 位换到第 2 位，…，依此类推，最后一位是原来的第 7 位。L0、R0 则是换位输出后的两部分，L0 是输出的左 32 位，R0 是右 32 位，例：设置换前的输入值为 D1D2D3……D64，则经过初始置换后的结果为：L0=D58D50…D8；R0=D57D49…D7。

(2)E 扩展

置换目标是 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。E 扩展置换目的有两个：生成与密钥相同长度的数据以进行异或运算；提供更长的结果，在后续的替代运算中可以进行压缩。E 扩展置换表：

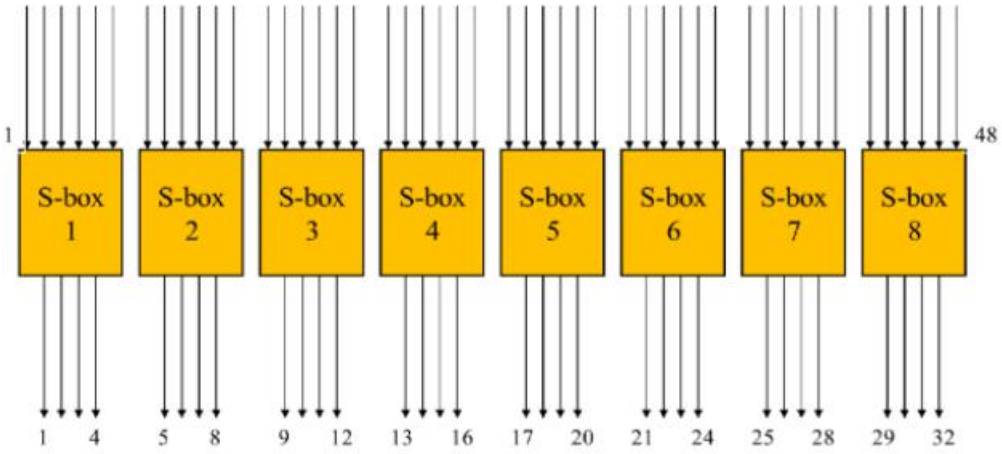
32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10,11,

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,

(3)S 盒

压缩后的密钥与扩展分组异或以后得到 48 位的数据，将这个数据送人 S 盒，进行替代运算。替代由 8 个不同的 S 盒完成，每个 S 盒有 6 位输入 4 位输出。48 位输入分为 8 个 6 位的分组，一个分组对应一个 S 盒，对应的 S 盒对各组进行代替操作。



在 f(Ri,Ki)算法描述图中，S1,S2…S8 为选择函数，其功能是把 6bit 数据变为 4bit 数据。下面给出选择函数 Si(i=1,2……)的功能表：

选择函数 Si :

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,

然后在S1表中查得对应的数，以4位二进制表示，此即为选择函数S1的输出。下面给出子密钥Ki(48bit)的生成算法 。初始 Key 值为 64 位，但 DES 算法规定，其中第 8、16、……64 位是奇偶校验位，不参与 DES 运算。故 Key 实际可用位数便只有 56 位。即：经过缩小选择换位表 1 的变换后，Key 的位数由 64 位变成了 56 位，此 56 位分为 C0、D0 两部分，各 28 位，然后分别进行第 1 次循环左移，得到 C1、D1，将 C1（28 位）、D1（28 位）合并得到 56 位，再经过缩小选择换位 2，从而便得到了密钥 K0（48 位）。依此类推，便可得到 K1、K2、……、K15，不过需要注意的是，16 次循环左移对应的左移位数要依据下述规则进行：

循环左移位数 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1

(4)P 置换

S 盒代替运算的 32 位输出按照 P 盒进行置换。该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，也不能被略去，映射规则如下表：

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,

(5)IP 逆置换

逆置换正好是初始 IP 置换的逆运算，例如，第 1 位经过初始置换后，处于第 40 位，而通过逆置换，又将第 40 位换回到第 1 位，其逆置换规则如下表所示：

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,

四、数据结构

对于加密解密过程中用到的数据和各项表，定义如下：

```
//S盒
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}}};
//置换选择1
int PC_1[56] = {56,48,40,32,24,16,8,
                0,57,49,41,33,25,17,
                9,1,58,50,42,34,26,
                18,10,2,59,51,43,35,
                62,54,46,38,30,22,14,
                6,61,53,45,37,29,21,
                13,5,60,52,44,36,28,
                20,12,4,27,19,11,3};

//置换选择2
int PC_2[48] = {13,16,10,23,0,4,2,27,
                14,5,20,9,22,18,11,3,
                25,7,15,6,26,19,12,1,
                40,51,30,36,46,54,29,39,
                50,44,32,46,43,48,38,55,
                33,52,45,41,49,35,28,31};

//对左移次数的规定
int MOVE_TIMES[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};
```

加解密中使用的函数以及其中的数据类型声明如下：

```
int ByteToBit(ElemType ch,ElemType bit[8]);
int BitToByte(ElemType bit[8],ElemType *ch);
int Char8ToBit64(ElemType ch[8],ElemType bit[64]);
int Bit64ToChar8(ElemType bit[64],ElemType ch[8]);
int DES_MakeSubKeys(ElemType key[64],ElemType subKeys[16][48]);
int DES_PC1_Transform(ElemType key[64], ElemType tempbts[56]);
int DES_PC2_Transform(ElemType key[56], ElemType tempbts[48]);
int DES_ROL(ElemType data[56], int time);
int DES_IP_Transform(ElemType data[64]);
int DES_IP_1_Transform(ElemType data[64]);
int DES_E_Transform(ElemType data[48]);
int DES_P_Transform(ElemType data[32]);
int DES_SBOX(ElemType data[48]);
int DES_XOR(ElemType R[48], ElemType L[48],int count);
int DES_Swap(ElemType left[32],ElemType right[32]);
int DES_EncryptBlock(ElemType plainBlock[8], ElemType subKeys[16][48], ElemType cipherBlock[8]);
int DES_DecryptBlock(ElemType cipherBlock[8], ElemType subKeys[16][48], ElemType plainBlock[8]);
int DES_Encrypt(char *plainFile, char *keyStr,char *cipherFile);
int DES_Decrypt(char *cipherFile, char *keyStr,char *plainFile);
```

五、C++算法过程

串格式以及数据形式的转换：

```
int ByteToBit(ElemType ch, ElemType bit[8]){
    int cnt;
    for(cnt = 0;cnt < 8; cnt++){
        *(bit+cnt) = (ch>>cnt)&1;
    }
    return 0;
}

//二进制转换成字节
int BitToByte(ElemType bit[8],ElemType *ch){
    int cnt;
    for(cnt = 0;cnt < 8; cnt++){
        *ch |= *(bit + cnt)<<cnt;
    }
    return 0;
}

//将长度为8的字符串转为二进制位串
int Char8ToBit64(ElemType ch[8],ElemType bit[64]){
    int cnt;
    for(cnt = 0; cnt < 8; cnt++){
        ByteToBit(*(ch+cnt),bit+(cnt<<3));
    }
    return 0;
}

//将二进制位串转为长度为8的字符串
int Bit64ToChar8(ElemType bit[64],ElemType ch[8]){
    int cnt;
    memset(ch,0,8);
    for(cnt = 0; cnt < 8; cnt++){
        BitToByte(bit+(cnt<<3),ch+cnt);
    }
    return 0;
}
```

IP 置换以及逆置换：

```
//IP置换
int DES_IP_Transform(ElemType data[64]){
    int cnt;
    ElemType temp[64];
    for(cnt = 0; cnt < 64; cnt++){
        temp[cnt] = data[IP_Table[cnt]];
    }
    memcpy(data,temp,64);
    return 0;
}

//IP逆置换
int DES_IP_1_Transform(ElemType data[64]){
    int cnt;
    ElemType temp[64];
    for(cnt = 0; cnt < 64; cnt++){
        temp[cnt] = data[IP_1_Table[cnt]];
    }
    memcpy(data,temp,64);
    return 0;
}
```

子密钥生成以及密钥置换：

```
//生成子密钥
int DES_MakeSubKeys(ElemType key[64],ElemType subKeys[16][48]){
    ElemType temp[56];
    int cnt;
    DES_PC1_Transform(key,temp);//PC1置换
    for(cnt = 0; cnt < 16; cnt++){//16轮迭代，产生16个子密钥
        DES_ROL(temp,MOVE_TIMES[cnt]);//循环左移
        DES_PC2_Transform(temp,subKeys[cnt]);//PC2置换，产生子密钥
    }
    return 0;
}

//密钥置换1
int DES_PC1_Transform(ElemType key[64], ElemType tempbts[56]){
    int cnt;
    for(cnt = 0; cnt < 56; cnt++){
        tempbts[cnt] = key[PC_1[cnt]];
    }
    return 0;
}

//密钥置换2
int DES_PC2_Transform(ElemType key[56], ElemType tempbts[48]){
    int cnt;
    for(cnt = 0; cnt < 48; cnt++){
        tempbts[cnt] = key[PC_2[cnt]];
    }
    return 0;
}
```

循环左移：

```
//循环左移
int DES_ROL(ElemType data[56], int time){
    ElemType temp[56];

    //保存将要循环移动到右边的位
    memcpy(temp,data,time);
    memcpy(temp+time,data+28,time);

    //前28位移动
    memcpy(data,data+time,28-time);
    memcpy(data+28-time,temp,time);

    //后28位移动
    memcpy(data+28,data+28+time,28-time);
    memcpy(data+56-time,temp+time,time);

    return 0;
}
```

E 扩展置换以及 P 置换和异或：

```
//扩展置换
int DES_E_Transform(ElemType data[48]){
    int cnt;
    ElemType temp[48];
    for(cnt = 0; cnt < 48; cnt++){
        temp[cnt] = data[E_Table[cnt]];
    }
    memcpy(data,temp,48);
    return 0;
}

//P置换
int DES_P_Transform(ElemType data[32]){
    int cnt;
    ElemType temp[32];
    for(cnt = 0; cnt < 32; cnt++){
        temp[cnt] = data[P_Table[cnt]];
    }
    memcpy(data,temp,32);
    return 0;
}

//异或
int DES_XOR(ElemType R[48], ElemType L[48] ,int count){
    int cnt;
    for(cnt = 0; cnt < count; cnt++){
        R[cnt] ^= L[cnt];
    }
    return 0;
}
```

S-Box 置换和交换：

```
//S盒置换
int DES_SBOX(ElemType data[48]){
    int cnt;
    int line,row,output;
    int cur1,cur2;
    for(cnt = 0; cnt < 8; cnt++){
        cur1 = cnt*6;
        cur2 = cnt<<2;

        //计算在S盒中的行与列
        line = (data[cur1]<<1) + data[cur1+5];
        row = (data[cur1+1]<<3) + (data[cur1+2]<<2)
            + (data[cur1+3]<<1) + data[cur1+4];
        output = S[cnt][line][row];

        //化为2进制
        data[cur2] = (output&0X08)>>3;
        data[cur2+1] = (output&0X04)>>2;
        data[cur2+2] = (output&0X02)>>1;
        data[cur2+3] = output&0x01;
    }
    return 0;
}

//交换
int DES_Swap(ElemType left[32], ElemType right[32]){
    ElemType temp[32];
    memcpy(temp,left,32);
    memcpy(left,right,32);
    memcpy(right,temp,32);
    return 0;
}
```

文件读取以及加密分组：

```
//加密文件
int DES_Encrypt(char *plainFile, char *keyStr,char *cipherFile){
    FILE *plain,*cipher;
    int count;
    ElemType plainBlock[8],cipherBlock[8],keyBlock[8];
    ElemType bKey[64];
    ElemType subKeys[16][48];
    if((plain = fopen(plainFile,"rb")) == NULL){
        return PLAIN_FILE_OPEN_ERROR;
    }
    if((cipher = fopen(cipherFile,"wb")) == NULL){
        return CIPHER_FILE_OPEN_ERROR;
    }
    //设置密钥
    memcpy(keyBlock,keyStr,8);
    //将密钥转换为二进制流
    Char8ToBit64(keyBlock,bKey);
    //生成子密钥
    DES_MakeSubKeys(bKey,subKeys);

    while(!feof(plain)){
        //每次读8个字节，并返回成功读取的字节数
        if((count = fread(plainBlock,sizeof(char),8,plain)) == 8){
            DES_EncryptBlock(plainBlock,subKeys,cipherBlock);
            fwrite(cipherBlock,sizeof(char),8,cipher);
        }
    }
    if(count){
        //填充
        memset(plainBlock + count,'\0',7 - count);
        //最后一个字符保存包括最后一个字符在内的所填充的字符数量
        plainBlock[7] = 8 - count;
        DES_EncryptBlock(plainBlock,subKeys,cipherBlock);
        fwrite(cipherBlock,sizeof(char),8,cipher);
    }
    fclose(plain);
    fclose(cipher);
    return OK;
}
```

```
//加密单个分组
int DES_EncryptBlock(ElemType plainBlock[8], ElemType subKeys[16][48], ElemType cipherBlock[8]){
    ElemType plainBits[64];
    ElemType copyRight[48];
    int cnt;

    Char8ToBit64(plainBlock,plainBits);
    //初始置换（IP置换）
    DES_IP_Transform(plainBits);

    //16轮迭代
    for(cnt = 0; cnt < 16; cnt++){
        memcpy(copyRight,plainBits+32,32);
        //将右半部分进行扩展置换，从32位扩展到48位
        DES_E_Transform(copyRight);
        //将右半部分与子密钥进行异或操作
        DES_XOR(copyRight,subKeys[cnt],48);
        //异或结果进入S盒，输出32位结果
        DES_SBOX(copyRight);
        //P置换
        DES_P_Transform(copyRight);
        //将明文左半部分与右半部分进行异或
        DES_XOR(plainBits,copyRight,32);
        if(cnt != 15){
            //最终完成左右部的交换
            DES_Swap(plainBits,plainBits+32);
        }
    }
    //逆初始置换（IP^1置换）
    DES_IP_1_Transform(plainBits);
    Bit64ToChar8(plainBits,cipherBlock);
    return 0;
}
```

解密过程用到的函数和加密一致，但是输入的子密钥的顺序相反。

DES 算法总体来说具有极高安全性，到目前为止，除了用穷举搜索法对 DES 算法进行攻击外，还没有发现更有效的办法。但是在课堂上蔡老师说到，随着科学技术的进步，DES 算法并不是不可攻破，也显得没有之前那么安全。DES 算法中只用到 64 位密钥中的其中 56 位，而第 8、16、24、……64 位 8 个位并未参与 DES 运算，即 DES 的安全性是基于其余 56 位的组合变化才得以保证的。现在我们需要用更长位数的 KEY 来设计加密过程，或者采用其他更安全严谨的加密算法如 RSA 等以最大程度保证数据的安全传输。