S-DES 算法程序开发手册

1. 项目概述

S-DES（Simplified Data Encryption Standard）算法是一个用于教育和理解加密算法基本原理的简化版本，用于加密和解密 8 位二进制数据块。本项目提供了一个基于 C++的 S-DES 算法程序，用于演示和学习加密算法的基本原理。

2. 使用说明简介

（1）选择二进制加密或者 ACSII 字符串加密。
（2）输入 8 位的二进制明文或密文。
（3）输入 10 位的二进制密钥。
（4）选择加密或解密模式。
（5）程序将生成加密后的密文或解密后的明文。

3. 算法原理

3.1 分组长度

S-DES 使用 8 位数据块作为输入。

3.2 密钥长度

S-DES 使用 10 位密钥，其中 2 位用于奇偶校验，因此实际有效密钥长度为 8 位。

3.3 算法描述

3.3.1 加密算法

S-DES 加密算法包括以下步骤：
初始置换：将输入数据重新排列以进行混淆。
密钥生成：从 10 位密钥生成两个 8 位子密钥。
轮函数：包括扩展置换、子密钥混合、S-盒替代、P4 置换等步骤。
两轮加密：应用轮函数两次。
逆初始置换：对加密结果进行逆初始置换以生成密文。

$$C = IP^{-1}(f_{k_2}(SW(f_{k_1}(IP(P)))))$$

3.3.2 解密算法

S-DES 解密算法包括以下步骤：
初始置换：将输入数据重新排列以进行混淆。
密钥生成：从 10 位密钥生成两个 8 位子密钥。
轮函数：包括扩展置换、子密钥混合、S-盒替代、P4 置换等步骤。

两轮解密：应用轮函数两次，解密步骤与加密步骤相反。
逆初始置换：对解密结果进行逆初始置换以生成明文。

$$P = IP^{-1}(f_{k_1}(SW(f_{k_2}(IP(C)))))$$

3.3.3 密钥扩展

初始密钥(10 位)：这是输入给密钥扩展过程的初始密钥。它通常是一个 10 位的二进制数。
P10 置换：初始密钥中的 10 位被置换，得到一个 8 位的中间密钥。
分割中间密钥：将中间密钥分成两部分，每部分包含 4 位。这两个 4 位的部分将成为生成子密钥的基础。
循环左移：对每个 4 位部分分别进行循环左移 1 位。这意味着将每个 4 位部分的每一位向左移动一个位置，循环到最高位。
合并左移后的部分：将左移后的 4 位部分合并成一个 8 位的中间密钥。这个中间密钥将用于生成第一个子密钥。
P8 置换：对中间密钥进行 P8 置换，得到第一个子密钥。
循环左移和 P8 置换的重复：重复上述的循环左移和 P8 置换步骤，但这次使用第一次左移后的中间密钥的右半部分，产生第二个子密钥。

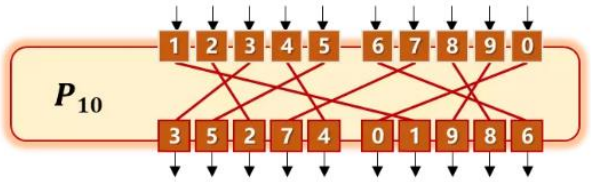
$$k_i = P_8(Shift^i(P_{10}(K))), (i = 1, 2)$$

3.4 转换装置设定

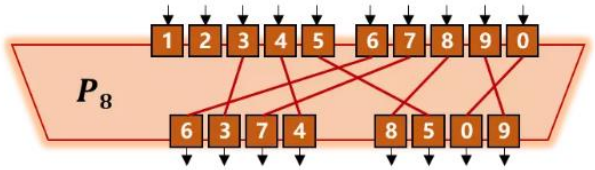
3.4.1 密钥扩展置换

密钥扩展置换包括 P10 和 P8 两个置换表，用于生成子密钥。
使用 P10 置换从 10 位输入密钥中选择 8 位，去除奇偶校验位。
将 8 位密钥分为左半部分和右半部分，分别进行循环左移 1 位。
使用 P8 置换将左右两部分组合成第一个子密钥。
再次进行循环左移 2 位，然后使用 P8 置换生成第二个子密钥。

- $P_{10} = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)$



- $P_8 = (6, 3, 7, 4, 8, 5, 10, 9)$

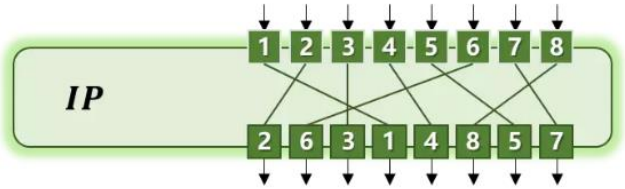


- $Left_Shift^1 = (2, 3, 4, 5, 1)$
- $Left_Shift^2 = (3, 4, 5, 1, 2)$

3.4.2 初始置换盒

初始置换盒（IP）用于对输入数据进行初始置换，混淆数据。

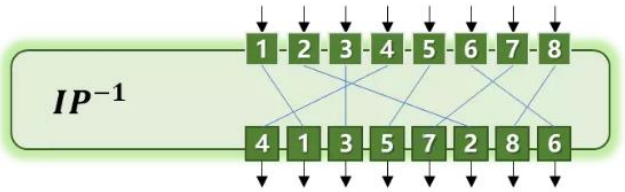
- $IP = (2, 6, 3, 1, 4, 8, 5, 7)$



3.4.3 最终置换盒

最终置换盒用于对加密结果进行逆初始置换，生成密文或解密结果。

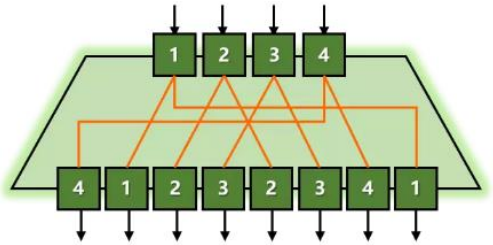
- $IP^{-1} = (4, 1, 3, 5, 7, 2, 8, 6)$



3.4.4 轮函数

轮函数包括扩展置换（EP）、子密钥混合、S-盒替代和 P4 置换等步骤，用于每一轮的加密和解密。

- $EPBox = (4, 1, 2, 3, 2, 3, 4, 1)$



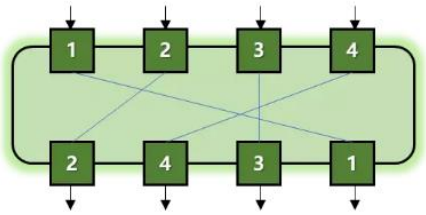
- $SBox_1 = [(1, 0, 3, 2); (3, 2, 1, 0); (0, 2, 1, 3); (3, 1, 0, 2)]$

	00	01	10	11
00	01	00	11	10
01	11	10	01	00
10	00	10	01	11
11	11	01	00	10

- $SBox_2 = [(0,1,2,3);(2,3,1,0);(3,0,1,2);(2,1,0,3)]$

	00	01	10	11
00	00	01	10	11
01	10	11	01	00
10	11	00	01	10
11	10	01	00	11

- $SPBox = (2,4,3,1)$



4. 代码组件

4. 1 主要函数

为了使代码模块化程度更高，逻辑更加清晰，编写多个核心函数来进行加解密。并且首先在外部定义加密过程中所需要的全局变量。

```
static int IP[]={2,6,3,1,4,8,5,7}; //初始置换
static int IP_1[]={4,1,3,5,7,2,8,6};  //最终置换

//密钥生成
static int P10[]={3,5,2,7,4,10,1,9,8,6}; //密钥扩展
static int P8[]={6,3,7,4,8,5,10,9}; //密钥压缩
//轮函数F
static int P4[]={2,4,3,1};  //SPBpx 直接置换
static int EP[]={4,1,2,3,2,3,4,1};  //EPBox 扩展置换
static int S0[4][4]={{1,0,3,2},{3,2,1,0},{0,2,1,3},{3,1,0,2}};
static int S1[4][4]={{0,1,2,3},{2,3,1,0},{3,0,1,2},{2,1,0,3}};
```

4. 1. 1 数组合并函数

由于加解密过程多处被分为两部分处理，需要合并函数将部分合并为整体。

```
//合并数组函数
void merge(char *num,char *x,char *y,int n){
    n/=2;
    for(int i=0;i<n;i++){
        num[i]=x[i];
        num[i+n]=y[i];  //长度为n/2的数组x和y合并成为数组num
    }
}
```

4. 1. 2 置换函数

对于加密过程中用到的 IP 初始置换、P10 扩展置换、P8 压缩置换等，统一编写置换函数来进行，提高代码模块化程度。

```
//置换函数
void change(char *z1,char *z2,int *P,int n){ //n为新数组长度
        for(int i=0;i<n;i++)
            z2[i]=z1[(P[i]-1)];  //将z1数组的第(P[i]-1) 个数置于z2数组的第i个
}
```

4.2 密钥生成

首先将初始密钥进行 P10 置换，再通过两个临时数组存放密钥的左右部分并分别进行处理，包含循环左移和 P8 置换，得到子密钥 K1 和 K2。

```
//子密钥生成
void key(char *mkey,char *k1,char *k2)
{
    char temp1[5];char temp2[5];//S1-L,S2-L 存放左/右半部分
    char mkey1[10];//临时数组
    change(mkey,mkey1,P10,10);//初始密钥进行P10置换
    for(int j=0;j<5;j++)//左右拆分
    {
        temp1[j]=mkey1[j];
        temp2[j]=mkey1[j+5];
    }
    char x=temp1[0];char y=temp2[0];//左移一位
    for(int k=0;k<4;k++)
    {
        temp1[k]=temp1[k+1];
        temp2[k]=temp2[k+1];
    }
    temp1[4]=x;temp2[4]=y; //最开头的一位到最后
    merge(mkey1,temp1,temp2,10);//合并
    change(mkey1,k1,P8,8);//P8置换 得子密钥k1
    x=temp1[0];y=temp2[0];//左移一位
    for(int k=0;k<4;k++)
    {
        temp1[k]=temp1[k+1];
        temp2[k]=temp2[k+1];
    }
    temp1[4]=x;temp2[4]=y; //最开头的一位到最后

    merge(mkey1,temp1,temp2,10);//合并
    change(mkey1,k2,P8,8);//P8置换 得子密钥k2
}
```

4.3 二进制加/解密

由于进行 SBoxes 替换过程中涉及到二进制整数型与字符型之间的转换，编写函数来进行。

```
//二进制整数型转字符型函数
char intz(int n){
    if(n==1)
    return '1';
    else
    return '0';
}
```

模块化编写加密函数，首先进行明文拆分，右半部分 Rm 进行 EP 扩展后与子密钥 K 按位异或,再进行 SBoxes 替换，最终进行 P4 置换后于最开始的 Lm 按位异或得到结果。

```
//加密主过程
void DES(char *temp,char *k1,char *Rm,char *Lm1){ //temp为明文
    char Lm[4];  //左半部分
    int i=0;
    char Rm1[8];  //右半部分Rm进行EP扩展后得Rm1
    for(i=0;i<4;i++){//拆分
        Lm[i]=temp[i];
        Rm[i]=temp[i+4];
    }
    for(i=0;i<8;i++){//EP扩展置换和Rm1与子密钥K1按位异或 结果为Rm1
        Rm1[i]=Rm[(EP[i]-1)];

        if(Rm1[i]==k1[i])
        Rm1[i]='0';
        else
        Rm1[i]='1';
    }
    int x1,y1,x2,y2;
    x1=(Rm1[1]-'0')*2+(Rm1[2]-'0');//S0盒替换
    y1=(Rm1[0]-'0')*2+(Rm1[3]-'0');
    x2=(Rm1[5]-'0')*2+(Rm1[6]-'0');//S1盒替换
    y2=(Rm1[4]-'0')*2+(Rm1[7]-'0');
    int s0=S0[y1][x1];
    int s1=S1[y2][x2];
    Rm1[0]=intz((s0/2)%2); //Rm1进行SBoxes替换
    Rm1[1]=intz(s0%2);
    Rm1[2]=intz((s1/2)%2);
    Rm1[3]=intz(s1%2);
    char Rm2[4];
    for(int i=0;i<4;i++){//Rm1进行P4置换得Rm2再和Lm做异或 结果为Lm1
        Rm2[i]=Rm1[(P4[i]-1)];
        if(Lm[i]==Rm2[i])
        Lm1[i]='0';
        else
        Lm1[i]='1';
    }
}
```

4.3.1 二进制加密

在 main 函数中通过添加 choose 变量来判断用户需要进行的操作类型，直接调用预先模块化编写好的函数即可进行加密操作，二进制加密部分如下。

```
int main(){
    int choose;
    cout<<"Please enter 1 for encryption, 2 for decryption,  3 for finding a key and 4 FOR ASCII"<<endl;
    cin>>choose;
    //加密
    if(choose==1){
    char mkey[10],m[8],temp[8];//主密钥，明文，临时数组
    char k1[8],k2[8];//子密钥K1，K2
    char Rm[4],Lm1[4];//分段
    while(cin>>mkey>>m){
        key(mkey,k1,k2);//生成子密钥 K1 K2
        change(m,temp,IP,8);//明文IP置换
        DES(temp,k1,Rm,Lm1);//第一次循环,产生Rm和Lm1
        char m1[8],Rm1[4],Lm11[4];
        merge(m1,Rm,Lm1,8);//将Rm和Lm1左右互换合成m1
        DES(m1,k2,Rm1,Lm11);//第二次循环,产生Rm1和Lm11
        char mw[8],mw1[8];//最终结果密文为mw1
        merge(mw,Lm11,Rm1,8);//将Rm1和Lm11合成mw
        change(mw,mw1,IP_1,8);//mw 进行IP-1置换，得到密文mw1
        for(int i=0;i<8;i++)
            cout<<mw1[i];
        cout<<endl;
        }
    }
```

4.3.2 二进制解密

由于 S-DES 算法是可逆的，因此只需要交换子密钥 K1 与 K2 使用的先后顺序即可实现解密过程。

```
//解密
if(choose==2){
char mkey[10],m[8],temp[8];//主密钥，密文，临时数组
char k1[8],k2[8];//子密钥K1，K2
char Rm[4],Lm1[4];//分段
while(cin>>mkey>>m){ //这里的m即为密文
    key(mkey,k1,k2);//生成子密钥 K1 K2
    change(m,temp,IP,8);//密文IP置换
    DES(temp,k2,Rm,Lm1);//第一次循环,产生Rm和Lm1 先用k2
    char m1[8],Rm1[4],Lm11[4];
    merge(m1,Rm,Lm1,8);//将Rm和Lm1左右互换合成m1
    DES(m1,k1,Rm1,Lm11);//第二次循环,产生Rm1和Lm11 后用k1
    char mw[8],mw1[8];//最终结果明文为mw1
    merge(mw,Lm11,Rm1,8);//将Rm1和Lm11合成mw
    change(mw,mw1,IP_1,8);//mw 进行IP-1置换，得到明文mw1
    for(int i=0;i<8;i++)
        cout<<mw1[i];
    cout<<endl;
    }
}
```

4.4 ASCII 码加/解密

由于加解密中涉及到 ASCII 字符串和二进制字符串之间的转换，编写两个函数来实现。

```
// 将ASCII字符串转换为二进制字符串
void asciiToBinaryString(const char *ascii, char *binary) {
    for (int i = 0; i < 8; i++) {
        char c = ascii[i];
        for (int j = 0; j < 8; j++) {
            binary[i * 8 + j] = ((c >> (7 - j)) & 1) ? '1' : '0';
        }
    }
}

// 将二进制字符串转换为ASCII字符串
void binaryStringToAscii(const char *binary, char *ascii) {
    for (int i = 0; i < 8; i++) {
        char c = 0;
        for (int j = 0; j < 8; j++) {
            c = (c << 1) | (binary[i * 8 + j] - '0');
        }
        ascii[i] = c;
    }
}
```

本程序支持任意位数的 ASCII 码加解密，因此在最初使用 String 类型获取用户输入，同时取得用户输入的字符个数，再将其转换为定长的字符数组。通过 8bit 临时数组 cut 以及循环标识 flag 的使用，以 1byte 为一组进行加/解密。其中加解密过程的选择通过用户输入 choose1 和 if 语句来实现。

```
// 字符串加/解密
if (choose == 4) {
    cout<<"Enter 1 for encryption, 2 for decryption"<<endl;
    int choose1;
    cin>>choose1;
    char mkey[10], temp[8],cut[8]; // 主密钥，临时数组，切割数组
    char k1[8], k2[8]; // 子密钥 K1，K2
    char Rm[4], Lm1[4]; // 分段
    cin>>mkey;
    string mm; //利用字符串特性输入统计输入的字符个数  明/密文
    cin>>mm;
    int length =mm.length(); //输入的字符个数
   // cout<<length<<endl;
    char m[length]; //字符串转字符数组
    strcpy(m, mm.c_str());
  //将ASCII明文转换为二进制字符串
    char binaryM[length*8];
    asciiToBinaryString(m, binaryM);
    int k=0;
    int flag=0;
    //按照8位一组进行加密，分别输出
    key(mkey,k1,k2);//生成子密钥 K1 K2
    while(flag<length*8)
        {
        for(int i=0;i<8;i++)
        {
            cut[i]=binaryM[k];
            k++;
        } //切割数组存放8位
```

在切割以及转换完成后，按照二进制加解密过程执行即可。

```
change(cut, temp, IP, 8);    //明文 IP置换
if(choose1==1)
DES(temp,k1,Rm,Lm1);//第一次循环,产生Rm和Lm1
else
DES(temp,k2,Rm,Lm1);//第一次循环,产生Rm和Lm1

char m1[8],Rm1[4],Lm11[4];
merge(m1,Rm,Lm1,8);//将Rm和Lm1左右互换合成m1
if(choose1==1)
DES(m1,k2,Rm1,Lm11);//第二次循环,产生Rm1和Lm11
else
DES(m1,k1,Rm1,Lm11);//第二次循环,产生Rm1和Lm11
char mw[8],mw1[8];//最终结果明文为mw1
merge(mw,Lm11,Rm1,8);//将Rm1和Lm11合成mw
cmhange(mw,mw1,IP_1,8);//mw 进行IP-1置换，得到明文mw1
int sum1=0;

//将8位二进制密文转为对应的字符
int mw1_int;
for(int i=0;i<8;i++)
    {
    if(mw1[i]=='0')mw1_int=0;
    else mw1_int=1;
    sum1+=mw1_int*(pow(2,7-i));
    }
cout<<char(sum1);
flag+=8;
    }
}
```

4.5 暴力破解

主要思路为，穷举二进制密钥，将每一种可能代入加密过程，验证加密出的密文与用户所输入的密文的一致性。密钥穷举的过程通过全排列来实现。

```
if(choose==3){
bool get = false; //确认是否找到了密钥
char m[8],mw[8],temp[8];//明文，密文，临时数组
char mkey[10] = {'0','0','0','0','0','0','0','0','0','0'};//初始化密钥全为0
char k1[8],k2[8];//子密钥K1，K2
char Rm[4],Lm1[4];//分段
cout<<"Please enter the plaintext:"<<endl;
cin>>m;
cout<<"Please enter the ciphertext:"<<endl;
cin>>mw;
int sum=0;
while(sum!=11)
{
    for(int i=0;i<10;i++)
    {
        if(mkey[i]=='0'){
        mkey[i]='1';
        break;
        }
        if(mkey[i]=='1'){
        mkey[i]='0';
        }
    } //二进制数组全排列
    sum=0;
    for(int i=0;i<10;i++)
    {
        if(mkey[i]=='1')sum++;
    }
```

获得本轮密钥后，代入加密过程，生成密文比对。穷举结束的标志为 mkey=11111111，通过计数器 count 对密钥中字符'1'的个数进行记录来判断是否达到了结束条件。

```
    //开始尝试：对明文加密看结果是否和密文相符
    key(mkey,k1,k2);//生成子密钥 K1 K2
    change(m,temp,IP,8);//明文IP置换
    DES(temp,k1,Rm,Lm1);//第一次循环,产生Rm和Lm1
    char m1[8],Rm1[4],Lm11[4];
    merge(m1,Rm,Lm1,8);//将Rm和Lm1左右互换合成m1
    DES(m1,k2,Rm1,Lm11);//第二次循环,产生Rm1和Lm11
    char mw1[8],mw11[8];//最终结果密文为mw11
    merge(mw1,Lm11,Rm1,8);//将Rm1和Lm11合成mw1
    change(mw1,mw11,IP_1,8);//mw1 进行IP-1置换，得到密文mw11
    int count=0;
    for(int i=0;i<8;i++) //比对加密出的密文和输入的密文
    {
        if(mw[i]==mw11[i])count+=1;
    }
    if(count==8)
    {
        cout<<"破译成功，密钥为："<<endl;
        get = true;
        for(int i=0;i<10;i++)
        cout<<mkey[i];
        cout<<endl;
    }
    if(sum==10)sum+=1; //全排列结束条件 数组全为1的试完

}
 if(get == false) cout<<"无对应密钥";

}
```

5. API 文档

SDES::encrypt(string plaintext, string key)
功能：使用 S-DES 算法加密明文。
参数：
plaintext：8 位的二进制明文字符串。
key：10 位的二进制密钥字符串。

返回值：加密后的 8 位二进制密文字符串。

SDES::decrypt(string ciphertext, string key)
功能：使用 S-DES 算法解密密文。
参数：
ciphertext：8 位的二进制密文字符串。
key：10 位的二进制密钥字符串。
返回值：解密后的 8 位二进制明文字符串。