School of Data and Computer Science, Sun Yat-sen University

DES Implementation

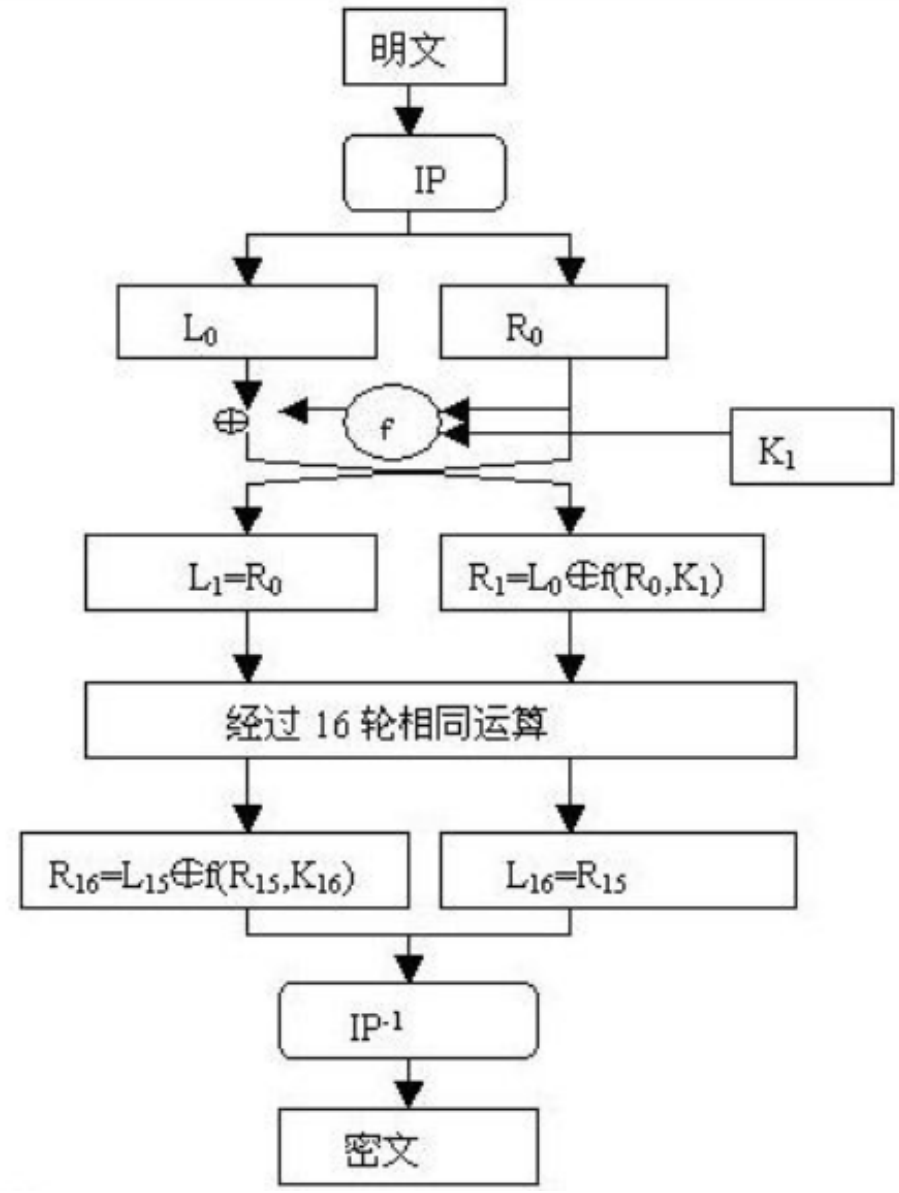
18340013 陈琮昊

December 24, 2020

Contents

1 DES 介绍

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。该算法将明文按 64 位进行分组，密钥的长度为 64 位，但只有 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。其简要流程图如下：



接下来对于每一步的置换操作进行介绍。

- 第一步是IP 置换：将输入的明文按照 IP 置换规则表进行置换，并得到左半部分 L_0 和右半部分 R_0；

- 第二步是密钥置换：首先根据规则将 64 位密钥压缩至 56 位（见 PC_KEY 表）；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：

  Step1：将 56 位的密钥分成两部分 C_0 和 D_0，每部分 28 位。

  Step2：这两部分分别循环左移 1 位或 2 位。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

  然后从 56 位中选出 48 位（如何选择见 PC_SUBKEY 表）。这个过程中，既置换了每位的顺序，又选择了子密钥。
- 第三步是E 扩展置换，其目的有两个：

  1. 生成与密钥相同长度的数据以进行异或运算；

  2. 提供更长的结果，在后续的替代运算中可以进行压缩。
- 第四步是S 盒代替：S 盒的 6 个输入确定了其对应的输出在哪一行哪一列，输入的最高、最低两位做为行数 H，中间四位做为列数 L，在 S 表中查找第 H 行 L 列对应的数据即可。
- 第五步是P 盒置换：在上一步 S 盒代替的 32 位输出按照 P 表进行置换。P 盒置换的结果与最初的 64 位分组的左半部分 L_0 异或，然后左、右两半部分交换，接着开始另一轮。
- 最后一步是IP 逆置换：IP 逆置换为 IP 置换的逆过程。在 DES 的最后一轮，左、右两半部分并不进行交换，而是两部分合并成一个分组做为末置换的输入，置换后输出的结果即为密文。

至此加密过程全部结束，解密过程大部分同上，只是在每轮迭代时密钥顺序相反（加密过程的密钥为 K1-K16，解密过程的密钥为 K16-K1）。这就是 DES 的整个过程，接下来编写代码实现。

2 代码实现

本次 DES 的代码实现使用 C 语言，由于篇幅有限，这里只放关键部分的代码，完整代码见文件 DES.c。

本次代码实现定义的变量如下：

```
char Sec_Key[8];                    //密钥
char Plain[100];                    //明文
char ans[100];                      //存放结果
int M[64];                          //存放明文对应的二进制串
int M_IP[64];                       //IP置换后的结果
int K[64];                          //存放密钥对应的二进制串
int ReducedK[56];                   //56位密钥
int Cipher[64];                     //密文
int L[17][32], R[17][32];           //L,R(64位)
```

```
int C[17][28], D[17][28];                    //C,D(56位)
int SubKey[17][48];                          //子密钥
int real_len;                                //明文实际长度
int len;                                     //明文填充后的长度
```

然后还有几张表，这些表是做置换时用到的：

```
const int IP[64];                            //IP置换规则表
const int PC_KEY[56];                        //64->56位密钥表
const int PC_SUBKEY[48];                     //56->48位密钥表
const int E[48];                             //E位选择表
const int S[8][4][16];                       //S盒功能表
const int P[32];                             //P盒置换表
const int IPinv[64];                         //IP逆置换表
```

在这里列举出本次实现用到的所有函数，如下：

```
void ShowKey();                                          //打印子密钥
void DES(int mode);                                      //DES核心代码
void PlainProcess(char* s);                              //对明文的处理
void SecKeyProcess();                                    //对密钥的处理
void Reduced_CD();                                       //获得C[0],D[0]
void fill(char* s);                                      //填充
void IPchange();                                         //IP置换
void Cir_LeftShift(int a[][28], int k);                  //循环左移
void getSubKey(int k);                                   //获得子密钥
void XOR(int *result, int *p1, int *p2, int len);        //异或
void diff(int *R, int k, int mode);                      //加密和解密的区别步骤
void Schange(int *R, int *Eresult);                      //S盒置换
void Pchange(int *R);                                    //P盒置换
void IPinvChange(int *Cipher, int *R, int *L);           //IP逆置换
```

DES 的核心代码实现如下：

```
//核心部分，mode=1代表加密，mode=2代表解密
void DES(int mode){
    if(mode == 1){
        fill(Plain);                                     //填充至8的整数倍
        int groups = len/8;                               //按64位分组
        for(int k = 0; k < groups; k++){
            PlainProcess(Plain+8*k);                     //将明文M转化为二进制串
            IPchange();                                  //首先进行IP置换
                    for(int i = 1; i < 17; i++){
                            int temp[32];
```

```
                for(int j = 0; j < 32; j++){
                        L[i][j] = R[i-1][j];                    //L[i]=R[i-1]
                        temp[j] = R[i-1][j];
                }
                diff(temp, i, mode);
               //加密和解密的唯一区别就在这一步，详见diff函数定义
                XOR(R[i], L[i-1], temp, 32);
            }
            IPinvChange(Cipher, R[16], L[16]);          //逆置换得到密文

            /*for(int i = 0; i < 64; i++){
                    printf("%d", Cipher[i]);
                    if((i+1)%8 == 0)          printf("\n");
            }
            printf("\n");*/

            for(int i = 0; i < 8; i++){
                    int ascii = 0;
                    for(int j = 7; j >= 0; j--){
                            ascii *= 2;
                            ascii += Cipher[8 * i + j];
                    }
                    //printf("%d ", ascii);
                    ans[8 * k + i] = ascii;
            }
        }
        ans[len] = '\0';                                //将密文的ASCII转为字符串并存入ans
        //printf("%s\n", ans);
    }
    //解密过程绝大部分和加密过程类似
    else if(mode == 2){
        int groups = len/8;
        for(int k = 0; k < groups; k++){
            PlainProcess(ans+8*k);
            IPchange();
            for(int i = 1; i < 17; i++){
                    int temp[32];
                    for(int j = 0; j < 32; j++){
                            L[i][j] = R[i-1][j];
                            temp[j] = R[i-1][j];
                    }
```

```
                diff(temp, i, mode);
                XOR(R[i], L[i-1], temp, 32);
            }
            IPinvChange(Cipher, R[16], L[16]);

            // for(int i = 0; i < 64; i++){
            //     printf("%d", Cipher[i]);
            //     if((i+1)%8 == 0)        printf("\n");
            // }
            // printf("\n");

            /* 注意：在加密的时候进行了填充，但在这里解密时
            不需要把填充的部分输出；只需讨论是否进行到最后一次循环，
            因为只有最后一次循环内的字符有可能包含填充内容。 */
            int lastgroup = (k == groups-1 ? real_len%8 : 8);
            /* 如果是最后一次循环，则要返回原字符串最后一组对应的字节数；
            如果不是则返回8（每组字节数）*/
            for(int i = 0; i < lastgroup; i++){
                int ascii = 0;
                for(int j = 7; j >= 0; j--){
                    ascii *= 2;
                    ascii += Cipher[8 * i + j];
                }
                //printf("%d ", ascii);
                ans[8 * k + i] = ascii;
            }
        }
        ans[real_len] = '\0';
            //将解密后的结果由ASCII转为字符串并存入ans
        //printf("%s\n", ans);
    }
}
```

主函数则是先调用 DES(1) 将明文加密为密文，然后再调用 DES(2) 将密文解密为明文：

```
//主函数
int main(){
    ShowKey();
    printf("\nMessage:%s\n", Plain);
    DES(1);
    printf("Encoding:%s\n", ans);
    DES(2);
```

```
    printf("Decoding:%s\n", ans);
}
```

3 运行结果

结果如下，Message 为要发送的消息，Encoding 为加密后的结果，结果显示为乱码（因为加密后可能会超出 ASCII 码范围）；Decoding 为解密后的结果，与 Message 相同。

```
PS C:\Users\czh\Desktop> cd "c:\Users\czh\Desktop\" ; if ($?) { gcc DES.c -o DES } ; if ($?) { .\DES }

Message:WONGSANZIT


Encoding:\]?j<膚季???


Decoding:WONGSANZIT
```

还可以看到中间过程的一些输出，这里只放出 48 位子密钥 K1-K16 的结果（中间过程的相关输出在代码文件内的注释部分可以看到）：

```
PS C:\Users\czh\Desktop> cd "c:\Users\czh\Desktop\" ; if ($?) { gcc DES.c -o DES } ; if ($?) { .\DES }
K1: 1000 1111 0000 1001 0001 1000 1111 1011 1111 0010 1101 1111
K2: 0100 0011 0100 0001 1100 1011 1111 1011 1010 1111 1111 1111
K3: 0011 1001 1100 0001 1000 0101 0011 1111 1111 1111 1001 1111
K4: 0001 0001 0000 1001 1000 1011 0111 1111 0111 0101 1111 0111
K5: 0011 0001 0010 0000 1011 0101 1110 1111 1110 1001 1110 1111
K6: 1001 0101 0000 1100 1000 0100 1110 0110 1111 1111 1101 1111
K7: 0101 0010 0010 0010 1011 0100 1111 1111 1001 0111 1111 1111
K8: 1001 1100 1001 0100 0010 0100 1101 1111 1101 1111 1110 1011
K9: 1100 0001 1010 1101 0000 0010 0111 1101 1111 1111 1110 1110
K10:0110 0000 1010 0110 1011 0011 1111 1100 1101 1100 1111 1111
K11:1111 0101 1001 0100 0000 0010 1100 1111 1111 1110 1111 1111
K12:0110 0010 1000 0010 1101 0010 1011 1111 1111 1111 1111 1001
K13:0011 1100 1101 0000 0001 0110 1011 1011 1101 1111 0111 0111
K14:0010 0110 0000 0001 0101 1010 1101 1111 1110 1111 1011 0110
K15:0010 1110 0100 0000 0011 0101 1111 1101 0110 1111 1101 1101
K16:0001 1101 1101 0101 0100 0001 1111 0111 1011 1110 1111 0101
```

4 参考资料

https://blog.csdn.net/pygmelion/article/details/83576666

https://www.cnblogs.com/songwenlong/p/5944139.html

https://blog.csdn.net/qq_27570955/article/details/52442092