信息安全技术期末项目—DES加密算法实现

李钰 19335112

1 引言

DES (Data Encryption Standard) 是常用的数据加密算法，于 1977 年被美国联邦政府的国家标准局授权在非密级政府通信中使用，随后在国际上被广泛使用。DES 算法使用了 Shannon 提出的混淆和扩散两个基本技术，综合运用了代数、代替、置换等多种密码技术。其目的是为了解决传统服务的身份认证及数据加密问题，避免恶意攻击者对密码系统的分析与统计，并在网络中提供加密通道供客户端与服务端进行数据传输，避免了明文在网络上传输的不安全问题。

2 DES加密算法

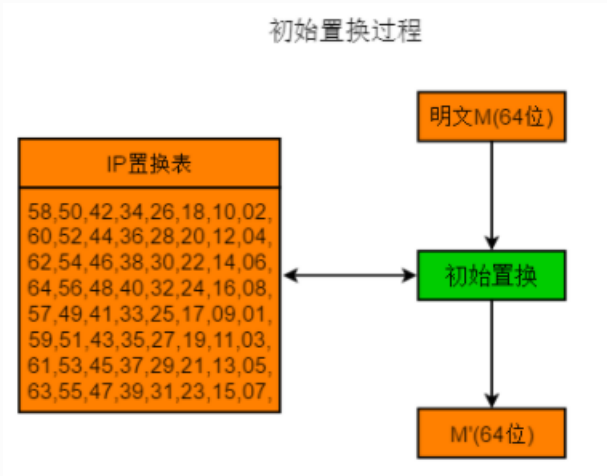
DES 加密算法中，明文和密文为 64 位分组。密钥的长度为 64 位，但是密钥的每个第八位设置为奇偶校验位，因此密钥的实际长度为56位。DES 加密算法为最为常见的分组加密算法。其主要思想在于数据位的置换与移位过程，通过16次的迭代加密与最终的逆置换得出最终的密文。DES 的解密方式只需按照加密的逆过程求解即可。由于DES 加密过程的算法是公开的，所以密钥K的保密就显得尤为重要，只有发送方与接收方采用相同的密钥进行加密解密才能获取明文数据。

DES 加密算法大致分为以下 4 个步骤：初始置换、生成子密钥、迭代过程、逆置换

整个过程流程图：

2.1 初始置换

初始置换是将原始明文经过IP置换表处理。置换过程如图：



例如：

输入64位明文数据M（64位）

明文M（64位）= 0110001101101111011011010111000001110101011101000110010101110010

选取密钥K（64位）

密钥K（64位）= 0001001100110100010101110111100110011011101111001101111111110001

IP置换表中的数据指的是位置，例如58指将M第58位放置第1位。

M经过IP置换后为M’

M’（64位） = 1111111110111000011101100101011100000000111111110000011010000011

取M’的前32位作为L0，则有

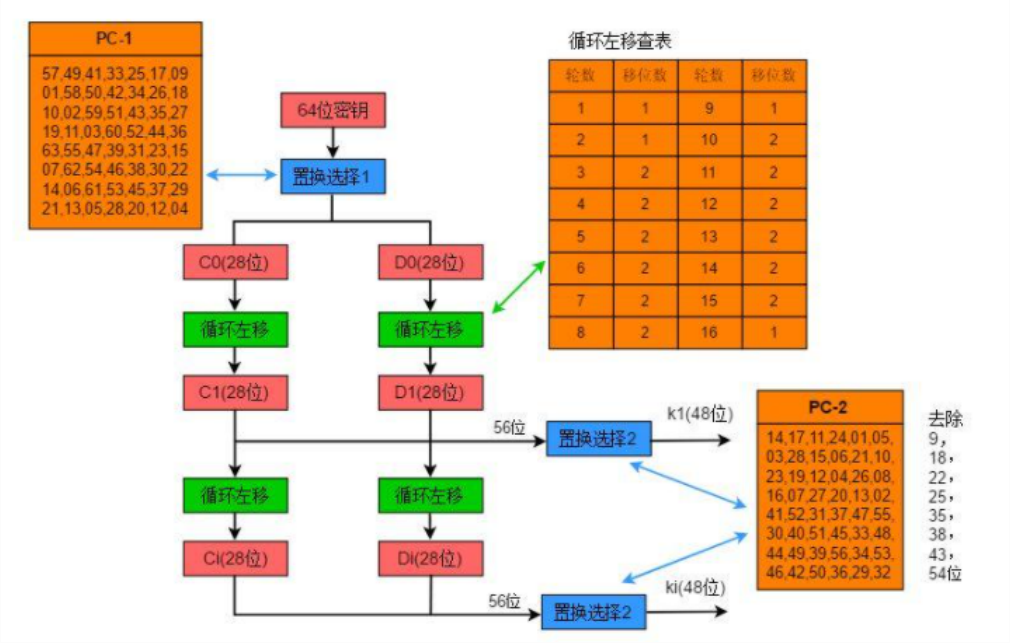
L0（32位）= 11111111101110000111011001010111

取M’的后32位作为R0，则有

R0（32位）= 00000000111111110000011010000011

2.2 生成子密钥

DES 加密共执行16次迭代，每次迭代过程的数据长度为 48位，因此需要16个48位的子密钥来进行加密，生成子密钥的过程如下：



我们继续用上面的例子进行说明

（1）第一轮置换：

密钥 K = 0001001100110100010101110111100110011011101111001101111111110001需经过PC-1表置换，即执行置换选择1过程。PC-1表为8行7列的表，密钥K经PC-1后变为56位数据K’。

（PC-1表见上图或源码）

K’（56位）= 11110000110011001010101011111010101010110011001111100011111

取K’的前28位作为C0，则有

C0（28位）= 1111000011001100101010101111

取K’的后28位作为D0，则有

D0（28位）= 0101010101100110011110001111

获得C0，D0后进行左移操作需要查询移动位数表：

每轮移动的移动位数表如下：

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

进行第一轮移位，轮数为1，查表得左移位数为1。

C0左移1位为C1：

C1（28位） = 1110000110011001010101011111

D0左移1位为D1：

D1（28位） = 1010101011001100111100011110

将C1和D1合并后，经过PC-2表置换得到子密钥K1，PC-2表中去除了第9，18，22，25，35，38，43，54位。

PPC-2表为6X8的表，PC-2表见上图或源码

由于PC-2表为6X8的表，经PC-2置换后的数据为48位，置换后得到密钥K1，

K1（48位） = 000110110000001011101111111111000111000001110010

（2）第二轮置换

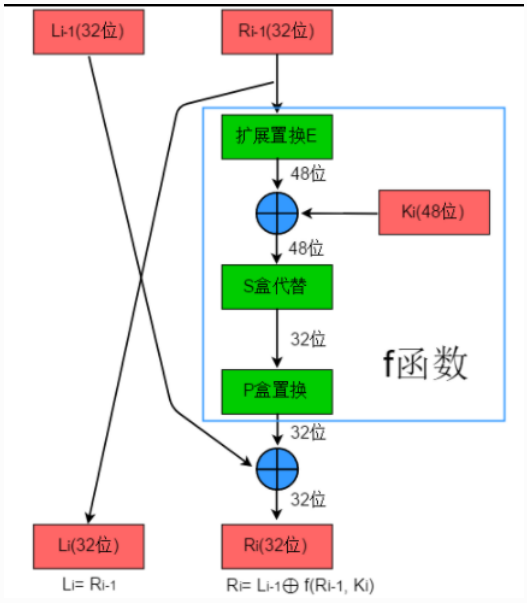
以此类推进行第二轮置换，C1和D1根据移动位数表左移得到C2和D2，之后合并，经过PC-2表置换得到密钥K2（48）位。以此类推，得到K3-K16子密钥。

2.3 迭代过程

设L_i（32位）和R_i（32位）为第i次迭代结果的左半部分与右半部分，子密钥Ki为第i轮的48位加密密钥。定义运算规则：

$$L_i = R_i - 1;$$
$$R_i = L_i \oplus f(R_{i-1}, K_i);$$

整个迭代过程如下图



令**扩展置换E、S-盒替代、P盒置换**的过程作为函数f,下面分别介绍这三个过程

2.3.1 扩展置换E

右半部分R_i的位数为32位，而密钥长度K_i为48位，为了能够保证R_i与K_i可以进行异或运算需要对R_i位数进行扩展，用于扩展置换表E如下：

扩展置换表E：

```
32,01,02,03,04,05,
04,05,06,07,08,09,
08,09,10,11,12,13,
12,13,14,15,16,17,
16,17,18,19,20,21,
20,21,22,23,24,25,
24,25,26,27,28,29,
28,29,30,31,32,01
```

例如：

L0（32位） = 11111111101110000111011001010111

R0（32位） = 00000000111111110000011010000011

R0（32位）经过扩展置换后变为48位数据：

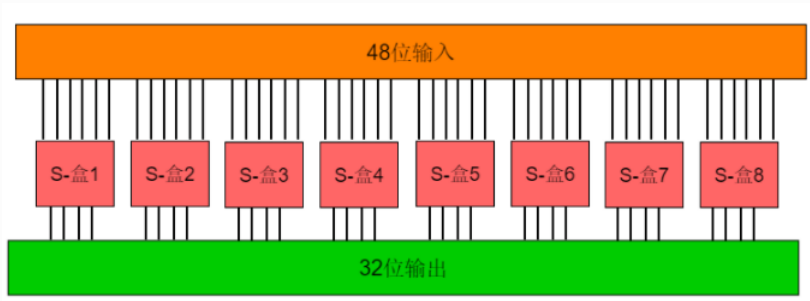
E(R0)（48位） = 100000000001011111111110100000001101010000000110

将E(R0)（48位）与K1（48位）作异或运算

E(R0)^K1（48位） = 100110110001010100010001011111001010010001110100

2.3.2 S-盒替代

代替运算由8个不同的代替盒（S盒）完成。每个S盒有6位输入，4位输出。代替运算流程如下：



（S盒数据内容见代码）

S盒的计算规则：

例如：若S-盒1的输入为110111，第一位与最后一位构成11，十进制值为3，则对应第3行，中间4位为1011对应的十进制值为11，则对应第11列。查找S-盒1表的值为14，则S-盒1的输出为1110。8个S盒将输入的48位数据输出为32位数据。

按照S-盒的计算过程，将

E(R0)^K1（48位） = 100110110001010100010001011111001010010001110100，通过 S- 盒替换得到的S盒输出为10001011110001000110001011101010（32位）。

2.3.3 P-盒置换

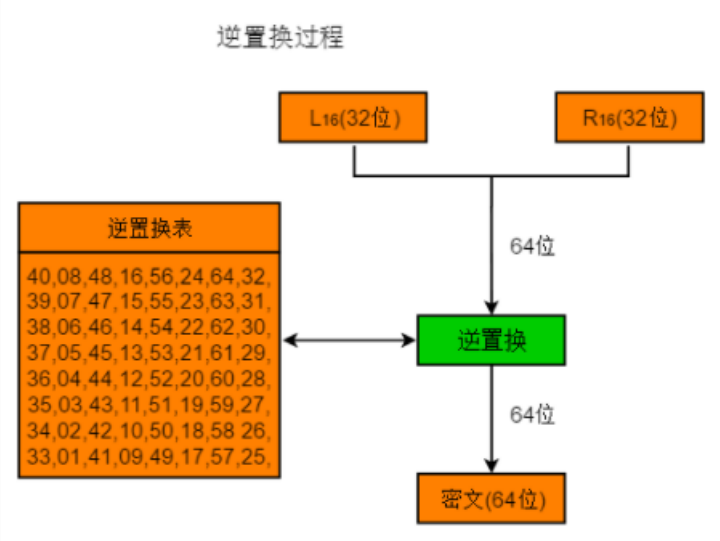
将S-盒替代的输出结果作为P-盒置换的输入。P-盒置换表如下：

```
16,07,20,21,29,12,28,17,01,15,23,26,05,18,31,10,
02,08,24,14,32,27,03,09,19,13,30,06,22,11,04,25,
```

将S盒输出10001011110001000110001011101010（32位）经过P盒置换，P-盒置换输出01001000101111110101010110000001

2.4 逆置换

将初始置换进行16次的迭代，即进行16层的加密变换，得到L16和R16，将此作为输入块，进行逆置换得到最终的密文输出块。逆置换是初始置换的逆运算。从初始置换规则中可以看到，原始数据的第1位置换到了第40位，第2位置换到了第8位。则逆置换就是将第40位置换到第1位，第8位置换到第2位。以此类推，逆置换规则表见下图。



最终输出为密文。

3 代码实现

本次代码运行之后，用户输入要加密的明文，之后输入密钥，随即程序输出明文通过加密之后。同时所有输入输出信息都被存储到"res.txt"文件中。

3.1 程序中需要用到的各类表

```
const static char IP_Table[64];      // 初始置换IP表
const static char Reverse_Table[64];  // 逆初始置换IP1表
static const char E_Table[48];       // 扩展置换E表
const static char P_Table[32];       // P盒置换表
const static char PC1_Table[56];     // 密钥置换表
const static char PC2_Table[48];     // 压缩置换表
const static char Move_Table[16];    // 每轮移动的位数
const static char S_Box[8][4][16];   // S盒设计
```

3.2 子函数

下面是一些核心函数需要的子功能函数

- 置换函数：通过查表`table`来重组输入元素得到输出

```
void transform(bool *out, bool *in, const char *table, int len)
{
    for(int i=0; i<len; ++i)
        Tmp[i] = in[ table[i]-1 ];
    memcpy(out, Tmp, len);
}
```

- 异或函数：对输入A和B进行异或运算，之后存到A中

```
void xor(bool *inA, const bool *inB, int len)
{
    for(int i=0; i<len; ++i)
        inA[i] ^= inB[i];//异或运算，相同为0，不同为1
}
```

- 进行左移函数

```
void RotateL(bool *in, int len, int loop)
{
    memcpy(Tmp, in, loop);//Tmp接受左移除的loop个字节
    memcpy(in, in+loop, len-loop);//in更新即剩下的字节向前移动loop个字节
    memcpy(in+len-loop, Tmp, loop);//左移除的字节添加到in的len-loop的位置
}
```

- S函数：经过S盒进行数值转换

```
void S_func(bool out[32], const bool in[48])  //将8组，每组6 bits的串，转化为8组，每组4 bits
{
    for(char i=0,j,k; i<8; ++i,in+=6,out+=4)
    {
        j = (in[0]<<1) + in[5];//取第一位和第六位组成的二进制数为S盒的纵坐标
        k = (in[1]<<3) + (in[2]<<2) + (in[3]<<1) + in[4];//取第二、三、四、五位组成的二进制
数为S盒的横坐标
        Byte2Bit(out, &S_Box[i][j][k], 4);
    }
}
```

- F函数：包含了E扩展、异或运算以及S盒运算、P盒计算。

```
void F_func(bool in[32], const bool Ki[48])
{
    static bool MR[48];
    transform(MR, in, E_table, 48);           //先进行 E 扩展
    xor(MR, Ki, 48);                          //再异或
    S_func(in, MR);                           //各组字符串分别经过各自的 S 盒
    transform(in, in, P_table, 32);           //最后 P 变换
}
```

3.3 核心函数—进行DES加密和解密

首先，我们利用`IP`表来对明文进行初步置换，之后进入16轮迭代，生成子密钥并且对子密钥进行f函数以及异或的计算，最后再合并。解密过程是加密的逆过程，解密时颠倒一下顺序即可。

```
void do_DES(char out[8], char in[8], const PSubKey subkey, bool Type)
{
    static bool M[64], tmp[32], *Li=&M[0], *Ri=&M[32];  //64 bits明文 经过IP置换后，分成左右两份
    Byte2Bit(M, in, 64);
    transform(M, M, IP_table, 64);
    if( Type == ENCRYPT )                                //加密
    {
        for(int i=0; i<16; ++i)                          //加密时：子密钥 K0~K15
        {
            memcpy(tmp, Ri, 32);
            F_func(Ri, (*subkey)[i]);                   // 调用F函数
            xor(Ri, Li, 32);                             //Li与Ri异或
            memcpy(Li, tmp, 32);
        }
    }
    else                 //解密
    {
        for(int i=15; i>=0; --i)                         // 解密时：Ki的顺序与加密相反
        {
            memcpy(tmp, Li, 32);
            F_func(Li, (*subkey)[i]);
            xor(Li, Ri, 32);
            memcpy(Ri, tmp, 32);
        }
    }
    transform(M, M, Reverse_table, 64);           //最后经过逆初始置换IP-1，得到密文/明文
    Bit2Byte(out, M, 64);
}
```

4 实验结果

编译指令：`g++ des.cpp -o des`

运行：`.\des`

我们随机输入一些明文，设置一个密钥进行实验。

res.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

clear text:
测试DES加密　　输入明文

Key　　输入密钥
0001001100110100010101110111100110011011101111001101111111110001

Ciphertext:　　输出密文
;*
蘿y?葬賺 躠□L愤%鄦0□|頨□Cz,O?呿{□Zn@r墩??犋9??L葩環_穟扒r墩??@r墩???呿{□Zn
鶊?籤駪?呿{□Zn

After decryption:　　输出经过解密之后的文本
测试DES加密

res.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

clear text:
测试随机密钥

Key
我是密钥

Ciphertext:
1bY□A□?X??+鐵f^q7dFl　雕ù□_q　垔長V^?T?鄶 mY?8□B⊥□
□丙{祐=T?鄶　　m=T?鄶　m]垔長V^諈wY聚~畎]垔長V^?

After decryption:
测试随机密钥