**西安电子科技大学**

**《基于对称密钥体制的加密算法实现》课程实验报告**

网络与信息安全 学院 2118021 班

成 绩

姓名 盖乐 学号 21009200991

同作者

实验日期 2024 年 5 月 31 日

|  |
| --- |
| 指导教师评语：  指导教师：  年 月 日 |
| **实验报告内容**  一、实验目的  二、实验所用仪器（或实验环境）  三、实验基本原理及步骤（或方案设计及理论计算）  四、实验数据记录（或仿真及软件设计）  五、实验结果分析及回答问题（或测试环境及测试结果） |

**实验报告内容**

一、实验目的

密码学是研究数据加密和数据解密算法的科学。掌握私钥密码体制的原

理，了解常用的私钥密码算法，选择熟悉的对称密钥算法实现信息的加密和

解密，从而对对密码学的相关知识，尤其数据加密解密的流程有一个初步的

了解。通过实际编程，进一步了解对称分组密码算法 DES 的加密和解密过程，

及其运行原理和实验方法。

二、实验所用仪器（或实验环境）

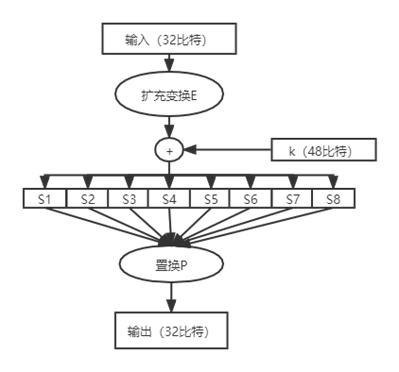
运行 Windows 或 Linux 操作系统的 PC 机，具有 gcc（linux）或 VC（Windows）

等的编译环境。

三、实验基本原理及步骤（或方案设计及理论计算）

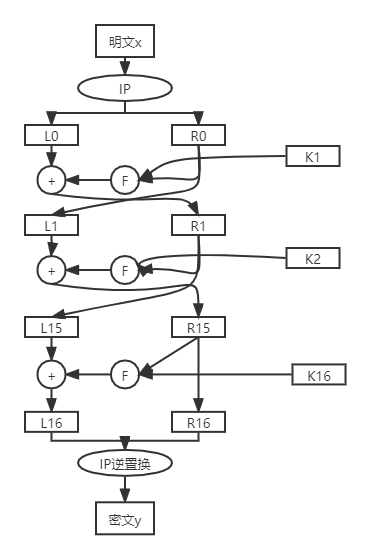
DES是迭代型分组密码，明文分组长度为64比特，轮函数是一个Feistel网络，F为32比特输入/32比特输出的钥控非线性函数，记为F(·,k)，k为密钥，若第j轮输入为（L(j-1)，R（j-1））,输出为（L(j)，R（j）），出该轮子密钥为K(j)，，则(L(j),R(j))=(F(R(j-1),K(j)),L(j-1)）

非线性函数F的结构：



DES加密算法为初始置换IP后跟16轮迭代，在最后一轮时不进行左右交换，之后再跟逆置换IP-1。

加密算法流程图如下:

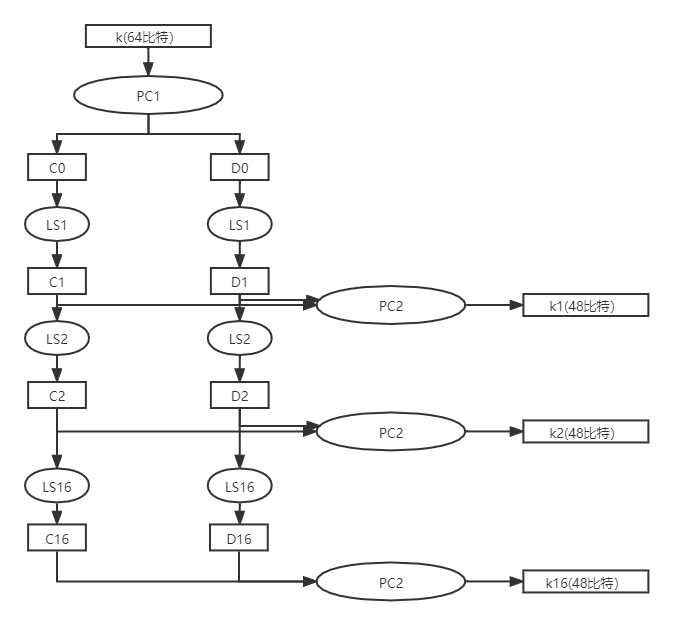


DES加密算法的密钥扩展算法需要用到两个缩减变换PC1和PC2，其中PC1是将64比特场的串缩减为56比特；PC2是将56比特长的串缩减为48比特。

密钥扩展算法还需要用到左移变换LSj，j=1-16。其中LSj表示生成第j轮子密钥k(j)时的循环左移变换。规定：当j=1,2,9,16时，LSj为循环左移1位；当j=3-8时或10-15时，LSj为循环左移2位。每一个循环左移变换的输入和输出都是28比特串。

密钥种子k为64比特长，但实际上只有56比特有效，而密钥种子中的第8,16,24、....、64位被舍去，因为它们通常是奇偶校验位。56比特的种子被扩展成48×16=768比特加密密钥。

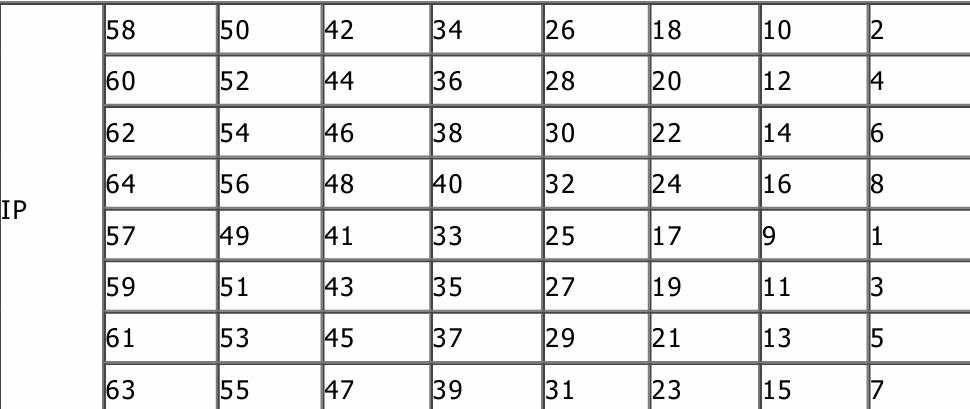
DES密钥扩展算法流程如下：



DES加解密相似，如果将加密算法记为y=DES(x,k1,k2,.....,k16)

其中(k1,k2,......,k16)为顺序各轮的子密钥，则解密算法为x=DES(y,k1,k2,......k16)

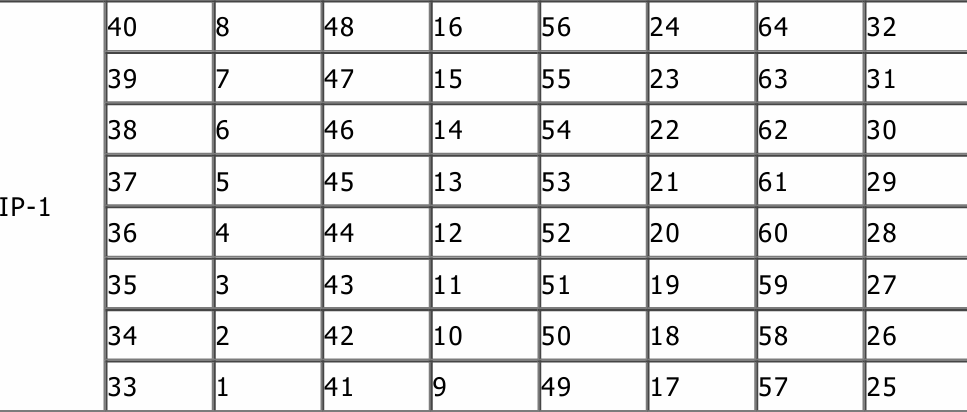
对DES算法加密过程图示的说明如下：待加密的64比特明文串m，经过IP置换后，得到的比特串的下标列表如下：



该比特串被分为32位的L0和32位的R0两部分。R0子密钥K1经过f变换f(R0,K1)输出32位的比特串f1，f1和L0做位异或运算。

运算结果被赋给R1，R0则原封不动地赋给L1。L1与R0又做与以上完全相同的运算，生成L2，R2……一共经过16次运算。最后生成R16和L16，其中R16为L15与f(R15,K16)做位异或运算的结果，L16是R15的直接赋值。

R16与L16合并成64位的比特串。值得注意的是R16一定要排在L16前面。R16与L16合并后成的比特串，经过置换IP-1后所得比特串的下标列表如下：



经过置换IP-1后生成的比特串就是密文e。

四、实验数据记录（或仿真及软件设计）

代码实现：

#include <stdlib.h>

#include <stdio.h>

//初始变换IP

const char IP\_Table[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

};

//IP逆变换

const char IPR\_Table[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

};

//E表格

static char E\_Table[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

};

//PC1表格

static char PC1\_Table[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

};

//左移变换

static char Move\_Table[16] = {

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

//PC2表格

static char PC2\_Table[48] = {

14,17,11,24, 1, 5, 3,28,15, 6,21,10,

23,19,12, 4,26, 8,16, 7,27,20,13, 2,

41,52,31,37,47,55,30,40,51,34,33,48,

44,49,39,56,34,53,46,42,50,36,29,32

};

//S盒

static char S\_Box[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, 0, 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, 0, 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

};

//P表格

static char P\_Table[32] = {

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个子密钥K(1~5)

static int SubKey[5][48] = { 0 };

void BitsCopy(int\* DatOut, int\* DatIn, int Len)//DatIn前Len位复制给DatOut

{

int i = 0;

for (i = 0; i < Len; i++)

{

DatOut[i] = DatIn[i];

}

}

void ByteToBit(int\* DatOut, char\* DatIn, int Num)//字节串转换为比特串

{

int i = 0;

for (i = 0; i < Num; i++)

{

DatOut[i] = (DatIn[i / 8] >> (i % 8)) & 0x01;

}

}

void BitToByte(char\* DatOut, int\* DatIn, int Num)//比特串转换为字节串

{

int i = 0;

for (i = 0; i < (Num / 8); i++)

{

DatOut[i] = 0;

}

for (i = 0; i < Num; i++)

{

DatOut[i / 8] |= DatIn[i] << (i % 8);

}

}

void BitToHex(char\* DatOut, int\* DatIn, int Num)//比特串转换为16进制

{

int i = 0;

for (i = 0; i < Num / 4; i++)

{

DatOut[i] = 0;

}

for (i = 0; i < Num / 4; i++)

{

DatOut[i] = DatIn[i \* 4] + (DatIn[i \* 4 + 1] << 1)

+ (DatIn[i \* 4 + 2] << 2) + (DatIn[i \* 4 + 3] << 3);

if ((DatOut[i] % 16) > 9)

{

DatOut[i] = DatOut[i] % 16 + '7'; // 余数大于9时处理 10-15 to A-F

} // 输出字符

else

{

DatOut[i] = DatOut[i] % 16 + '0'; // 输出字符

}

}

}

void HexToBit(int\* DatOut, char\* DatIn, int Num)//十六进制转二进制

{

int i = 0; // 字符型输入

for (i = 0; i < Num; i++)

{

if ((DatIn[i / 4]) > '9') // 大于9

{

DatOut[i] = ((DatIn[i / 4] - '7') >> (i % 4)) & 0x01;

}

else

{

DatOut[i] = ((DatIn[i / 4] - '0') >> (i % 4)) & 0x01;

}

}

}

void TablePermute(int\* DatOut, int\* DatIn, const char\* Table, int Num)//位表置换函数

{

int i = 0;

static int Temp[256] = { 0 };

for (i = 0; i < Num; i++) // Num为置换的长度

{

Temp[i] = DatIn[Table[i] - 1]; // 原来的数据按对应的表上的位置排列

}

BitsCopy(DatOut, Temp, Num); // 把缓存Temp的值输出

}

void LoopMove(int\* DatIn, int Len, int Num) //循环左移函数

{

static int Temp[256] = { 0 }; // 缓存 OK

BitsCopy(Temp, DatIn, Num); // 将数据最左边的Num位(被移出去的)存入Temp

BitsCopy(DatIn, DatIn + Num, Len - Num); // 将数据左边开始的第Num移入原来的空间

BitsCopy(DatIn + Len - Num, Temp, Num); // 将缓存中移出去的数据加到最右边

}

void Xor(int\* DatA, int\* DatB, int Num) //按位异或函数

{

int i = 0;

for (i = 0; i < Num; i++)

{

DatA[i] = DatA[i] ^ DatB[i];

}

}

void S\_Change(int DatOut[32], int DatIn[48]) // S盒变换

{

int i, X, Y; // i为8个S盒

for (i = 0, Y = 0, X = 0; i < 8; i++, DatIn += 6, DatOut += 4) // 每执行一次,输入数据偏移6位

{ // 每执行一次,输出数据偏移4位

Y = (DatIn[0] << 1) + DatIn[5]; // af代表第几行

X = (DatIn[1] << 3) + (DatIn[2] << 2) + (DatIn[3] << 1) + DatIn[4]; // bcde代表第几列

ByteToBit(DatOut, &S\_Box[i][Y][X], 4); // 把找到的点数据换为二进制

}

}

void F\_Change(int DatIn[32], int DatKi[48]) // F函数

{

static int MiR[48] = { 0 }; // 输入32位通过E选位变为48位

TablePermute(MiR, DatIn, E\_Table, 48);

Xor(MiR, DatKi, 48); // 和子密钥异或

S\_Change(DatIn, MiR); // S盒变换

TablePermute(DatIn, DatIn, P\_Table, 32); // P置换后输出

}

void SetKey(char KeyIn[8]) // 设置密钥 获取子密钥Ki

{

int i = 0;

static int KeyBit[64] = { 0 }; // 密钥二进制存储空间

static int\* KiL = &KeyBit[0], \* KiR = &KeyBit[28]; // 前28,后28共56

ByteToBit(KeyBit, KeyIn, 64); // 把密钥转为二进制存入KeyBit

TablePermute(KeyBit, KeyBit, PC1\_Table, 56); // PC1表置换 56次

for (i = 0; i < 16; i++)

{

LoopMove(KiL, 28, Move\_Table[i]); // 前28位左移

LoopMove(KiR, 28, Move\_Table[i]); // 后28位左移

TablePermute(SubKey[i], KeyBit, PC2\_Table, 48);

// 二维数组 SubKey[i]为每一行起始地址

// 每移一次位进行PC2置换得 Ki 48位

}

}

void PlayDes(char MesOut[8], char MesIn[8]) // 执行DES加密

{ // 字节输入 Bin运算 Hex输出

int i = 0;

static int MesBit[64] = { 0 }; // 明文二进制存储空间 64位

static int Temp[32] = { 0 };

static int\* MiL = &MesBit[0], \* MiR = &MesBit[32]; // 前32位 后32位

ByteToBit(MesBit, MesIn, 64); // 把明文换成二进制存入MesBit

TablePermute(MesBit, MesBit, IP\_Table, 64); // IP置换

for (i = 0; i < 5; i++) // 迭代5次

{

BitsCopy(Temp, MiR, 32); // 临时存储

F\_Change(MiR, SubKey[i]); // F函数变换

Xor(MiR, MiL, 32); // 得到Ri

BitsCopy(MiL, Temp, 32); // 得到Li

}

TablePermute(MesBit, MesBit, IPR\_Table, 64);

BitToHex(MesOut, MesBit, 64);

}

void KickDes(char MesOut[8], char MesIn[8]) // 执行DES解密

{ // Hex输入 Bin运算 字节输出

int i = 0;

static int MesBit[64] = { 0 }; // 密文二进制存储空间 64位

static int Temp[32] = { 0 };

static int\* MiL = &MesBit[0], \* MiR = &MesBit[32]; // 前32位 后32位

HexToBit(MesBit, MesIn, 64); // 把密文换成二进制存入MesBit

TablePermute(MesBit, MesBit, IP\_Table, 64); // IP置换

for (i = 4; i >= 0; i--)

{

BitsCopy(Temp, MiL, 32);

F\_Change(MiL, SubKey[i]);

Xor(MiL, MiR, 32);

BitsCopy(MiR, Temp, 32);

}

TablePermute(MesBit, MesBit, IPR\_Table, 64);

BitToByte(MesOut, MesBit, 64);

}

int main()

{

int i = 0;

char MesHex[16] = { 0 };

char key\_1[8] = { 0 };

char key\_2[8] = { 0 };

char Input\_Msg[8]={0};

char Output\_Msg[8]={0};

printf("输入64比特字符串：");

scanf("%s",Input\_Msg);

printf("输入64比特密钥：");

scanf("%s",key\_1);

while (key\_1[i] != '\0')

{

i++;

}

while (i != 8)

{

printf("密钥长度错误,重新输入密钥：");

scanf("%s",key\_1);

i = 0;

while (key\_1[i] != '\0')

{

i++;

}

}

SetKey(key\_1);

PlayDes(MesHex, Input\_Msg);

printf("加密结果：");

for (i = 0; i < 16; i++)

{

printf("%c", MesHex[i]);

}

printf("\n");

printf("输入密钥：");

scanf("%s",key\_2);

SetKey(key\_2);

KickDes(Output\_Msg, MesHex);

printf("解密结果：");

for (i = 0; i < 8; i++)

{

printf("%c", Output\_Msg[i]);

}

printf("\n");

system("pause");

}

五、实验结果分析及回答问题（或测试环境及测试结果）

代码运行结果：

