 ****

**实验报告二**

学生姓名： 陈华豪 学 号： 6130116238 专业班级： 16级网络工程

实验类型：√验证 □ 综合 □设计□创新 实验日期： 2018.11.17 实验成绩：

1. 实验项目名称

DES算法的实现。

1. 实验目的

学习常见的古典密码学算法，通过编程实现替代密码算法和置换密码算法，加深对古典密码体制的了解，为深入学习密码学奠定基础。

1. 实验基本原理

DES是美国联邦信息处理标准(FIPS)于1977年公开的分组密码算法，它的设计基于Feistel对称网络以及精心设计的S盒，在提出前已经进行了大量的密码分析，足以保证在当时计算条件下的安全性。不过，随着计算能力的飞速发展，现如今DES已经能用密钥穷举方式破解。虽然现在主流的分组密码是AES，但DES的设计原理仍有重要参考价值。在本实验中，为简便起见，就限定DES密码的明文、密文、密钥均为64bit，具体描述如下：

**明文**m是64bit序列。

**初始密钥**K是64 bit序列(含8个奇偶校验bit)。

**子密钥**K1, K2…K16均是48 bit序列。

**轮变换函数**f(A,J)：输入A(32 bit序列), J(48 bit序列)，输出32 bit序列。

**密文**c是64 bit序列。

1. **子密钥生成：**

输入初始密钥，生成16轮子密钥K1, K2…K16。

初始密钥(64bit)经过置换PC-1，去掉了8个奇偶校验位，留下56 bit，接着分成两个28 bit的分组C0与D0，再分别经过一个循环左移函数LS1，得到C1与D1，连成56 bit数据，然后经过置换PC-2，输出子密钥K1，以此类推产生K2至K16。

1. **轮变换函数f：**

f是DES加解密中每一轮的核心运算，输入A(32 bit), J(48 bit)，输出32 bit。

将A做一个扩展运算E，变成48 bit，记为E(A)。计算B=E(A)⊕J，将B分为8组B1…B8，每组Bi为6 bit，通过相应的S盒Si，输出Ci为4 bit，将所有Ci连成C(32 bit)，再通过置换P，得到最后的输出f(A,J)，为32 bit。在加密或解密的第i轮，A = Ri-1，J = Ki。

注意：每个S盒Si是4×16的矩阵，输入b0b1b2b3b4b5 (6 bit)，令L是b0b5对应的十进制数，n是b1b2b3b4对应的十进制数，输出矩阵中第L行n列所对应数的二进制表示。

详见下图：

4 bit

6 bit

A (32bit)

E(A) (48bit)

B1 B2 B3 B4 B5 B6 B7 B8

J (48bit)

E

S1

S2

S8

S7

S6

S5

S4

S3

C1 C2 C3 C4  C5 C6 C7 C8

P

f(A,J) (32bit)

1. **加/解密：**

DES的加密和解密几乎一样，不同之处在于加密时输入是明文，子密钥使

用顺序为K1K2…K16；解密时输入是密文，子密钥使用顺序为K16K15…K1**。**

以加密为例，输入明文分组64 bit，先进行一次初始置换IP，对置换后的数据X0分成左右两半L0与R0，根据第一个子密钥K1对R0实行轮变换f(R0, K1)，将结果与L0作逐位异或运算，得到的结果成为下一轮的R1，R0则成为下一轮的L1。如此循环16次，最后得到L16与R16。可用下列公式简洁地表示：



最后对64 bit数据R16, L16实行IP的逆置换IP-1，即得密文分组。

注意：第16轮变换后并未交换L16与R16，而直接将R16, L16作为IP-1的输入。

详见下图：

L0

输入

初始置换IP

R0

f

K1

R1=L0⊕f(R0,K1)

L1=R0

f

Ki

L15=R14

R15=L14⊕f(R14,K15)

f

R16=L15⊕f(R15,K16)

K16

L16=R15

逆初始置换IP-1

输出

1. 主要仪器设备及耗材

运行Windows或Linux操作系统的PC机， C语言（C++）或JAVA编译环境。

1. 实验步骤

相关表格及置换等操作对应的数据，供参考。

static const unsigned char IP\_Table[64] = { //定义IP置换表

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};

static const unsigned char IPInv\_Table[64] = { //定义IP逆置换表

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};

static const unsigned char E\_Table[48] = { //定义E扩展表

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};

static const unsigned char P\_Table[32] = { //定义P置换表

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};

static const unsigned char PC1\_Table[56] = { //定义PC1置换表

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};

static const unsigned char PC2\_Table[48] = { //定义PC2置换表

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

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

41,52,31,37,47,55,30,40,51,45,33,48,

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

static const unsigned char LS\_Table[16] = { //定义左移位数表

1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

static unsigned char S\_Box[8][4][16] = { //S盒

//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

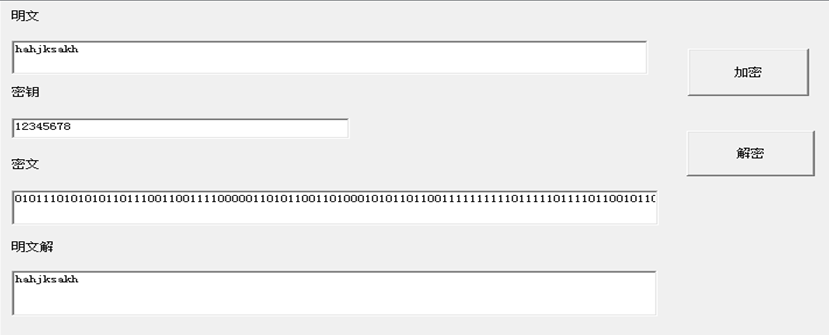
13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,

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,

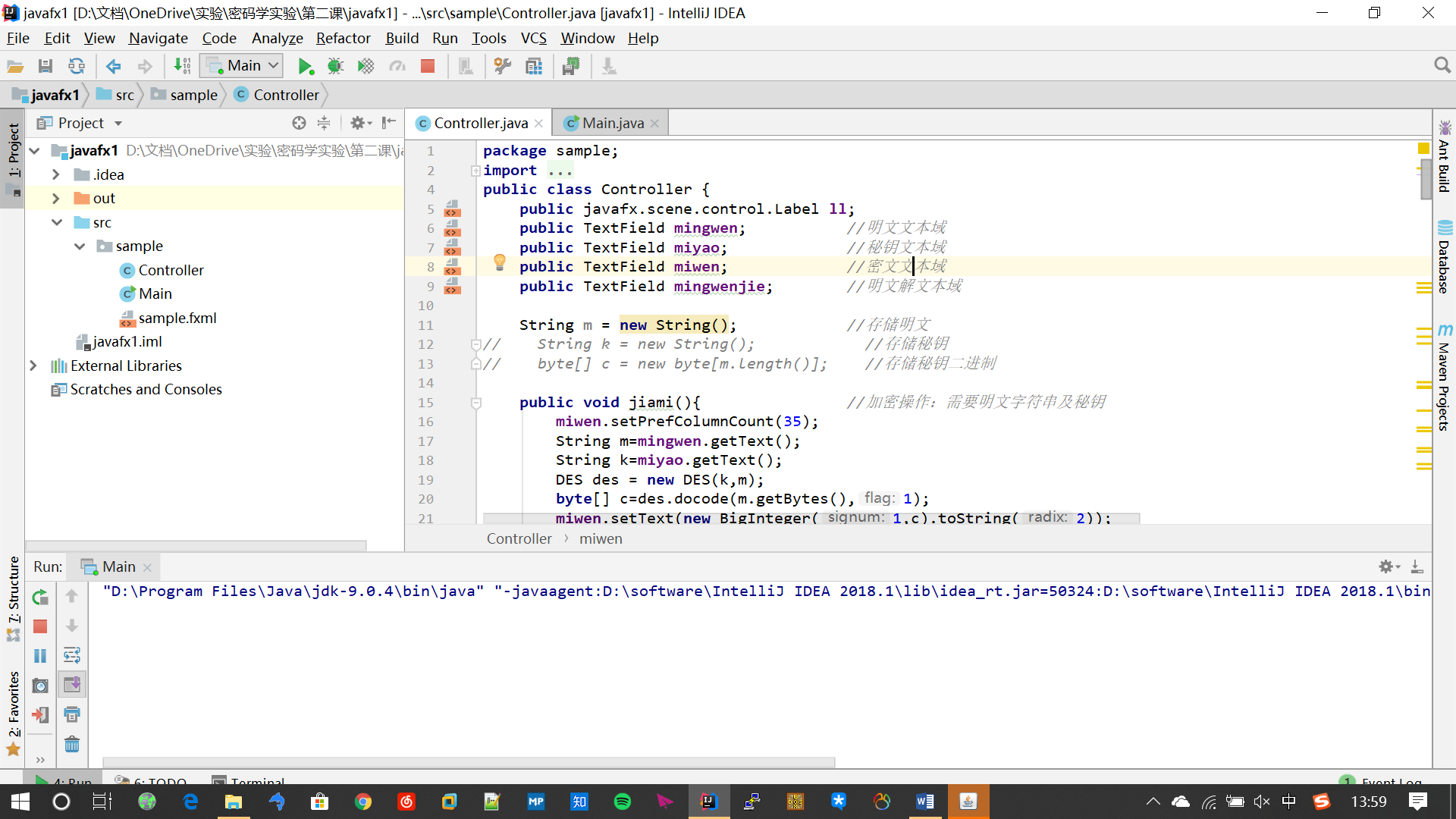
2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11};

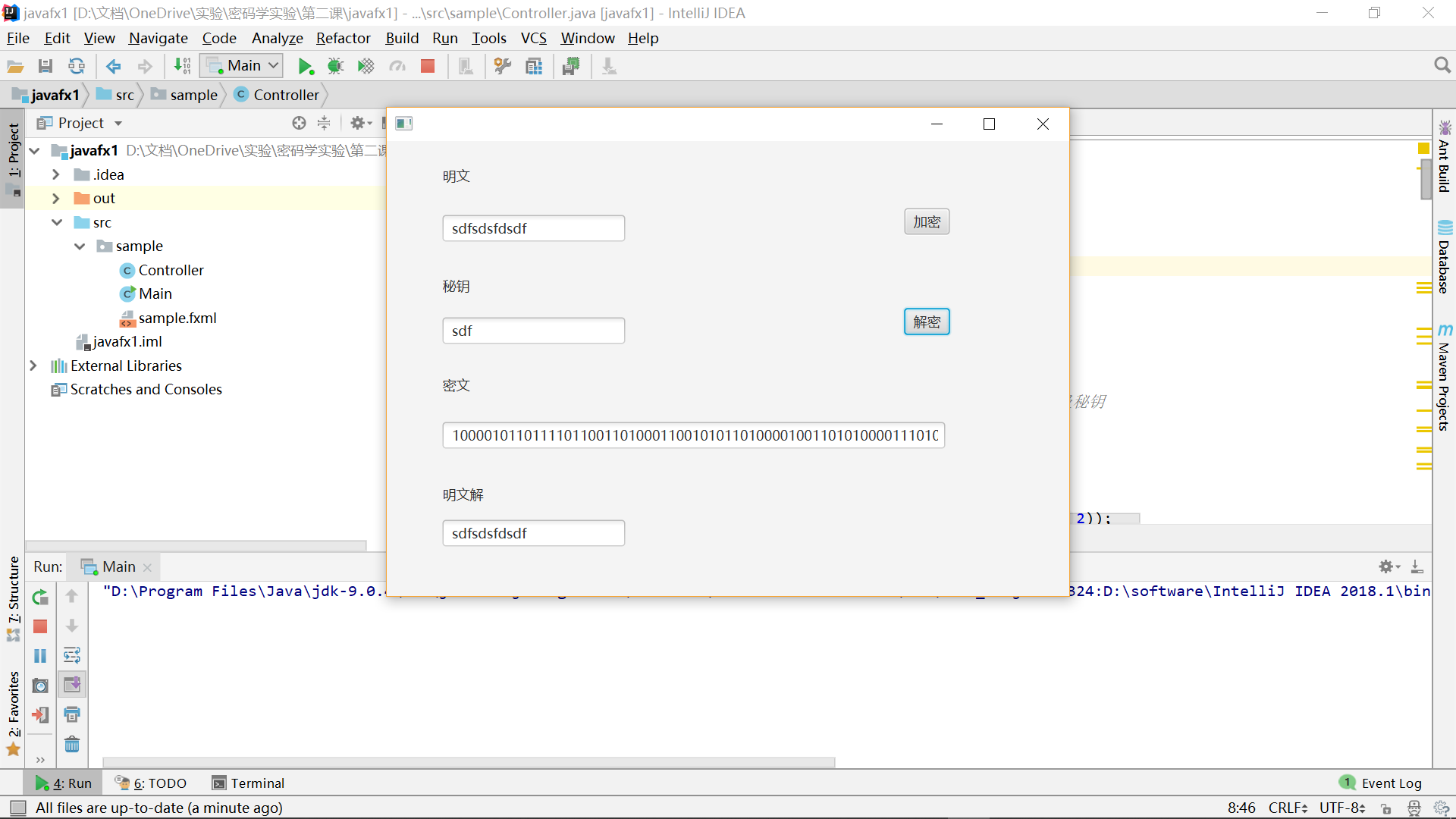
用类似界面实现加解密过程



1. 实验数据及处理结果

结果：





代码：

**package** sample;  
**import** java.math.BigInteger;  
**import** javafx.scene.control.TextField;  
**public class** Controller {  
 **public** javafx.scene.control.Label **ll**;  
 **public** TextField **mingwen**; *//明文文本域* **public** TextField **miyao**; *//秘钥文本域* **public** TextField **miwen**; *//密文文本域* **public** TextField **mingwenjie**; *//明文解文本域* String **m** = **new** String(); *//存储明文  
// String k = new String(); //存储秘钥  
// byte[] c = new byte[m.length()]; //存储秘钥二进制* **public void** jiami(){ *//加密操作：需要明文字符串及秘钥* **miwen**.setPrefColumnCount(35);  
 String m=**mingwen**.getText();  
 String k=**miyao**.getText();  
 DES des = **new** DES(k,m);  
 **byte**[] c=des.docode(m.getBytes(),1);  
 **miwen**.setText(**new** BigInteger(1,c).toString(2));  
 }  
 **public void** jiemi() { *//解密操作：只需要密文二进制字符串以及秘钥* String k = **miyao**.getText();  
 String maiden1 = **miwen**.getText();  
 BigInteger integer = **new** BigInteger(maiden1, 2);  
 **byte**[] c = integer.toByteArray();  
 DES des=**new** DES(c,k);  
 **byte**[]p=des.docode(c,0);  
 **mingwenjie**.setText(**new** String(p));  
 }  
  
 **class** DES {  
 **int**[] **IP\_Table**={ *//定义IP置换表* 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};  
 **int**[] **IPInv\_Table**={ *//定义IP逆置换表* 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};  
 **int**[] **E\_Table**={ *//定义E扩展表* 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};  
 **int**[] **P\_Table**={ *//定义P置换表* 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};  
   
 **int**[] **PC1\_Table**={ *//定义PC1置换表* 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};  
 **private int**[] **PC2\_Table**={ *//定义PC2置换表* 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,  
 51,45,33,48,44,49,39,56,  
 34,53,46,42,50,36,29,32};  
 **int**[] **LS\_Table**={1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1}; *//定义左移位数表* **final int**[][][] **S\_Box** = { *//S盒* {  
 { 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 } },  
 {  
 { 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 } },  
 {  
 { 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 } },  
 {  
 { 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 } },  
 {  
 { 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 } },  
 {  
 { 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 } },  
 {  
 { 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 } },  
 {  
 { 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 } }  
 };  
   
  
 **int** mlength; *//明文长度* String key; *//秘钥* **int**[][] sunkey=**new int**[16][48]; *//子秘钥* **public** DES(String key,String message){ *//构造方法* **this**.key=key;  
 mlength=message.getBytes().length;  
 initKeys(key);  
 }  
 **public** DES(**byte**[] c,String k){  
 **this**.**key**=k;  
 **mlength**=c.**length**;  
 initKeys(k);  
 }  
 **public byte**[] docode(**byte**[] p ,**int** flag)  
 { *//加密信息并返回密文* **int** length;  
 length=p.**length**;  
 **int** zuShu;  
 **int** tianChong;  
 zuShu=length/8; *//明文组数* tianChong=8-(length-zuShu\*8); *//计算mod8便于填充明文* **byte**[] p\_padding; *//填充后的明文字节数组* **if** (tianChong<8)  
 { *//填充明文* p\_padding=**new byte**[length+tianChong];  
 System.arraycopy(p,0,p\_padding,0,length);  
 **for**(**int** i=0;i<tianChong;i++)  
 {  
 p\_padding[length+i]=(**byte**)0; *//用"0"填充* }  
  
 }**else** {  
 p\_padding=p;  
 }  
 zuShu=p\_padding.length/8; *//明文组数* **byte**[] f\_p=**new byte**[8]; *//八个字节一组进行加密，存储每次加密的八个字节* **byte**[] result=**new byte**[p\_padding.length]; *//密文字节型数组* **for**(**int** i=0;i<zuShu;i++)  
 { *//加密* System.arraycopy(p\_padding,i\*8,f\_p,0,8);  
 System.arraycopy(decode(f\_p,sunkey,flag),0,result,i\*8,8);  
 }  
 **if** (flag==0)  
 { *//解密* **byte**[] p\_result=**new byte**[mlength];  
 System.arraycopy(result,0,p\_result,0,mlength);  
 **return** p\_result;  
 }  
 **return** result;  
  
 }  
 **public byte**[] decode(**byte**[] p,**int** k[][],**int** flag)  
 {  
 **int**[] p\_bit=**new int**[64]; *//整型存储明文分组二进制表示* StringBuilder stringBuilder=**new** StringBuilder(); *//字符串存储明文分组的二进制表示* **for**(**int** i=0;i<8;i++)  
 {  
 String p\_b=Integer.toBinaryString(p[i]&0xff);  
 **while** (p\_b.length()%8!=0)  
 {  
 p\_b=**"0"**+p\_b;  
 }  
 stringBuilder.append(p\_b);  
 }  
 String pstr=stringBuilder.toString(); *//明文分组的二进制表示* **for**(**int** i=0;i<64;i++)  
 { *//将数字变为01形式* **int** p\_t=Integer.valueOf(pstr.charAt(i));  
 **if**(p\_t==48)  
 {  
 p\_t=0;  
 }**else if**(p\_t==49){  
 p\_t=1;  
 }  
 p\_bit[i]=p\_t;  
 }  
 **int** [] p\_IP=**new int**[64];  
 **for** (**int** i=0;i<64;i++)  
 { *//IP置换* p\_IP[i]=p\_bit[IP\_Table[i]-1];  
 }  
 **if** (flag == 1)  
 { *//加密轮置换输出为p\_IP* **for** (**int** i = 0; i < 16; i++)  
 {  
 lunZhiHuan(p\_IP, i, flag, k[i]);  
 }  
 } **else if** (flag == 0)  
 { *//解密轮置换输出为p\_IP* **for** (**int** i = 15; i > -1; i--)  
 {  
 lunZhiHuan(p\_IP, i, flag, k[i]);  
 }  
 }  
 **int**[] c=**new int**[64];  
 **for**(**int** i=0;i<IPInv\_Table.length;i++)  
 { *//逆置换输出为数组c* c[i]=p\_IP[IPInv\_Table[i]-1];  
 }  
 **byte**[] c\_byte=**new byte**[8]; *//以字节型存储密文* **for**(**int** i=0;i<8;i++)  
 {  
 c\_byte[i]=(**byte**) ((c[8\*i]<<7)+(c[8\*i+1]<<6)+(c[8\*i+2]<<5)+(c[8\*i+3]<<4)+(c[8\*i+4]<<3)+(c[8\*i+5]<<2)+(c[8\*i+6]<<1)+(c[8\*i+7]));  
 }  
 **return** c\_byte;  
  
  
 }  
 **public void** lunZhiHuan(**int**[] M, **int** times, **int** flag, **int**[] sunkeyi)  
 { *//轮置换方法* **int**[] L0=**new int**[32];  
 **int**[] R0=**new int**[32];  
 **int**[] L1=**new int**[32];  
 **int**[] R1=**new int**[32];  
 **int**[] f=**new int**[32];  
 System.arraycopy(M,0,L0,0,32);  
 System.arraycopy(M,32,R0,0,32);  
 L1=R0;  
 f=function(R0,sunkeyi);  
 **for**(**int** j=0;j<32;j++)  
 {  
 R1[j]=L0[j]^f[j];  
 **if** (((flag == 0) && (times == 0)) || ((flag == 1) && (times == 15)))  
 {  
 M[j] = R1[j];  
 M[j + 32] = L1[j];  
 }  
 **else** {  
 M[j] = L1[j];  
 M[j + 32] = R1[j];  
 }  
 }  
  
 }  
  
  
  
 **public int**[] function(**int** [] r\_key,**int** [] key)  
 {  
 **int**[] result=**new int**[32];  
 **int**[] e\_k=**new int**[48];  
 **for**(**int** i=0;i<E\_Table.length;i++)  
 {  
 e\_k[i]=r\_key[E\_Table[i]-1]^key[i];  
 }  
 **int**[][] s=**new int**[8][6];  
 **int**[]s\_after=**new int**[32];  
 **for**(**int** i=0;i<8;i++)  
 {  
 System.arraycopy(e\_k,i\*6,s[i],0,6);  
 **int** r=(s[i][0]<<1)+ s[i][5];  
 **int** c=(s[i][1]<<3)+(s[i][2]<<2)+(s[i][3]<<1)+s[i][4];  
 String str=Integer.toBinaryString(S\_Box[i][r][c]);  
 **while** (str.length()<4)  
 {  
 str=**"0"**+str;  
 }  
 **for**(**int** j=0;j<4;j++)  
 {  
 **int** p=Integer.valueOf(str.charAt(j));  
 **if**(p==48)  
 {  
 p=0;  
 }**else if**(p==49){  
 p=1;  
 }  
 s\_after[4\*i+j]=p;  
 }  
  
 }  
 **for**(**int** i=0;i<P\_Table.length;i++)  
 {  
 result[i]=s\_after[P\_Table[i]-1];  
 }  
 **return** result;  
  
 }  
 **public void** initKeys(String key) *//生成子秘钥* {  
 **while** (key.length()<8)  
 {  
 key=key+key;  
 }  
 key=key.substring(0,8);  
 **byte**[] keys=key.getBytes();  
 **int**[] k\_bit=**new int**[64];  
 **for**(**int** i=0;i<8;i++)  
 {  
 String k\_str=Integer.toBinaryString(keys[i]&0xff);  
 **int** c = k\_str.length();  
 **if**(k\_str.length()<8)  
 {  
 **for**(**int** t=0;t<8-c;t++)  
 {  
 k\_str=**"0"**+k\_str;  
 }  
 }  
 **for**(**int** j=0;j<8;j++)  
 {  
 **int** p=Integer.valueOf(k\_str.charAt(j));  
 **if**(p==48)  
 {  
 p=0;  
 }**else if**(p==49){  
 p=1;  
 }  
 k\_bit[i\*8+j]=p;  
 }  
 }  
 **int** [] k\_new\_bit=**new int**[56];  
 **for**(**int** i=0;i<PC1\_Table.length;i++)  
 {  
 k\_new\_bit[i]=k\_bit[PC1\_Table[i]-1];  
 }  
 **int**[] c0=**new int**[28];  
 **int**[] d0=**new int**[28];  
 System.arraycopy(k\_new\_bit,0,c0,0,28);  
 System.arraycopy(k\_new\_bit,28,d0,0,28);  
 **for**(**int** i=0;i<16;i++)  
 {  
 **int**[] c1=**new int**[28];  
 **int**[] d1=**new int**[28];  
 **if**(LS\_Table[i]==1)  
 {  
 System.arraycopy(c0,1,c1,0,27);  
 c1[27]=c0[0];  
 System.arraycopy(d0,1,d1,0,27);  
 d1[27]=d0[0];  
 }**else if**(LS\_Table[i]==2) {  
 System.arraycopy(c0,2,c1,0,26);  
 c1[26]=c0[0];  
 c1[27]=c0[1];  
  
 System.arraycopy(d0,2,d1,0,26);  
 d1[26]=d0[0];  
 d1[27]=d0[1];  
 }  
 **int**[] tmp=**new int**[56];  
 System.arraycopy(c1,0,tmp,0,28);  
 System.arraycopy(d1,0,tmp,28,28);  
 **for** (**int** j=0;j<PC2\_Table.length;j++)  
 {  
 sunkey[i][j]= tmp[PC2\_Table[j]-1];  
 }  
 c0=c1;  
 d0=d1;  
 }  
  
 }  
 }  
  
}

**package** sample;  
  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.stage.Stage;  
  
**import** java.util.Scanner;  
  
**public class** Main **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception{  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"sample.fxml"**));  
 Scene scene = **new** Scene(root);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}