云南大学数学与与统计学院
上机实践报告

课程名称：信息论基础实验	**年级：**2013	**上机实践成绩：**
指导教师：陆正福	**姓名：**金洋	
上机实践名称：公钥密码实验	**学号：**20131910023	**上机实践日期：2016/7/1**
上机实践编号：No.11	**组号：**	**上机实践时间： 22:58**

一、实验目的

理解公钥密码体制

二、实验内容

1. RSA 体制的实现与分析
2. Elgamal 体制的实现与分析
3. Rabin 体制的实现与分析（选做）
4. 椭圆曲线密码体制的实现与分析（选做）

要求：

（1） 实现密码体制。

（2） 任取一段输入数据作为明文，计算明文熵；将密码系统作用于明文，得到密文，计算密文熵。比较明文熵和密文熵。

（3）改变明文 1bit，观察密文的变化。改变密钥 1bit，观察密文的变化。

（4）改变密文 1bit，观察解密后的明文变化。

（5）分析（2）-（4）中的实验现象和原因。

三、实验环境

1. 个人计算机，任意可以完成实验的平台，如 Java 平台、Python 语言、R 语言、Matlab 平台、Magma 平台等。
2. 对于信息与计算科学专业的学生，建议选择 Java、Python、R 等平台。
3. 对于非信息与计算科学专业的学生，建议选择 Matlab、Magma 等平台。

四、实验记录与实验结果分析

（注意记录实验中遇到的问题。实验报告的评分依据之一是实验记录的细致程度、实验过程的真实性、实验结果的解释和分析。**如果涉及实验结果截屏，应选择白底黑字。**）

1. RSA 体制的实现与分析

（1） 实现密码体制。

Node.java

```
package IT11;
public class Node {
	private double pr;
	private char al;

	public void setp(double p) {
		this.pr = p;
	}

	public void setalpha(char a) {
		this.al = a;
	}

	public double getp() {
		return pr;
	}

	public char getalpha() {
		return al;
	}

	public Node(double p, char alpha) {
		this.pr = p;
		this.al = alpha;
	}


}
```

ENTROPY.java

```
package IT11;
import java.util.ArrayList;
public class ENTROPY {
	public static double entropy(String message) {

		ArrayList<Node> array = new ArrayList<Node>();
```

```
		array.clear();
		double num = message.length();
		for (int i = 0; i < num; i++) {
			boolean flag_exit = true;
			for (int j = 0; j < array.size(); j++) {
				if (array.get(j).getalpha() == message.charAt(i)) {
					flag_exit = false;
					array.get(j).setp(array.get(j).getp() + 1 /
num);
				}
			}
			if (flag_exit)
				array.add(new Node(1 / num, message.charAt(i)));
		}

		double entropy = 0;
		for (int i = 0; i < array.size(); i++) {
			double p1 = array.get(i).getp();
			entropy += (-p1 * (Math.log(p1) / Math.log(2)));
		}
		return entropy;
	}

}
```

RSA.java

```
package IT11;

import java.math.BigInteger;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.RSAPrivateKeySpec;
import java.security.spec.RSAPublicKeySpec;
import java.util.HashMap;

import javax.crypto.Cipher;

public class RSA {

	/**
	 * 生成公钥和私钥
	 * @throws NoSuchAlgorithmException
```

```
     *
     */
    public static HashMap<String, Object> getKeys() throws
NoSuchAlgorithmException{
            HashMap<String, Object> map = new HashMap<String, Object>();
            KeyPairGenerator keyPairGen =
KeyPairGenerator.getInstance("RSA");
        keyPairGen.initialize(1024);
        KeyPair keyPair = keyPairGen.generateKeyPair();
        RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
        RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
        map.put("public", publicKey);
        map.put("private", privateKey);
        return map;
    }
    /**
     * 使用模和指数生成 RSA 公钥
     * 注意：【此代码用了默认补位方式，为 RSA/None/PKCS1Padding，不同 JDK 默
认的补位方式可能不同，如 Android 默认是 RSA
     * /None/NoPadding】
     *
     * @param modulus
     *          模
     * @param exponent
     *          指数
     * @return
     */
    public static RSAPublicKey getPublicKey(String modulus, String
exponent) {
            try {
                BigInteger b1 = new BigInteger(modulus);
                BigInteger b2 = new BigInteger(exponent);
                KeyFactory keyFactory = KeyFactory.getInstance("RSA");
                RSAPublicKeySpec keySpec = new RSAPublicKeySpec(b1, b2);
                return (RSAPublicKey) keyFactory.generatePublic(keySpec);
            } catch (Exception e) {
                e.printStackTrace();
                return null;
            }
    }

    /**
     * 使用模和指数生成 RSA 私钥
     * 注意：【此代码用了默认补位方式，为 RSA/None/PKCS1Padding，不同 JDK 默
认的补位方式可能不同，如 Android 默认是 RSA
     * /None/NoPadding】
     *
     * @param modulus
```

```
	 *            模
	 * @param exponent
	 *            指数
	 * @return
	 */
	public static RSAPrivateKey getPrivateKey(String modulus, String
exponent) {
		try {
			BigInteger b1 = new BigInteger(modulus);
			BigInteger b2 = new BigInteger(exponent);
			KeyFactory keyFactory = KeyFactory.getInstance("RSA");
			RSAPrivateKeySpec keySpec = new RSAPrivateKeySpec(b1, b2);
			return (RSAPrivateKey)
keyFactory.generatePrivate(keySpec);
		} catch (Exception e) {
			e.printStackTrace();
			return null;
		}
	}

	/**
	 * 公钥加密
	 *
	 * @param data
	 * @param publicKey
	 * @return
	 * @throws Exception
	 */
	public static String encryptByPublicKey(String data, RSAPublicKey
publicKey)
			throws Exception {
		Cipher cipher = Cipher.getInstance("RSA");
		cipher.init(Cipher.ENCRYPT_MODE, publicKey);
		// 模长
		int key_len = publicKey.getModulus().bitLength() / 8;
		// 加密数据长度 <= 模长-11
		String[] datas = splitString(data, key_len - 11);
		String mi = "";
		//如果明文长度大于模长-11 则要分组加密
		for (String s : datas) {
			mi += bcd2Str(cipher.doFinal(s.getBytes()));
		}
		return mi;
	}

	/**
	 * 私钥解密
	 *
```

```
	 * @param data
	 * @param privateKey
	 * @return
	 * @throws Exception
	 */
	public static String decryptByPrivateKey(String data, RSAPrivateKey privateKey)
			throws Exception {
		Cipher cipher = Cipher.getInstance("RSA");
		cipher.init(Cipher.DECRYPT_MODE, privateKey);
		//模长
		int key_len = privateKey.getModulus().bitLength() / 8;
		byte[] bytes = data.getBytes();
		byte[] bcd = ASCII_To_BCD(bytes, bytes.length);
		System.err.println(bcd.length);
		//如果密文长度大于模长则要分组解密
		String ming = "";
		byte[][] arrays = splitArray(bcd, key_len);
		for(byte[] arr : arrays){
			ming += new String(cipher.doFinal(arr));
		}
		return ming;
	}
	/**
	 * ASCII 码转 BCD 码
	 *
	 */
	public static byte[] ASCII_To_BCD(byte[] ascii, int asc_len) {
		byte[] bcd = new byte[asc_len / 2];
		int j = 0;
		for (int i = 0; i < (asc_len + 1) / 2; i++) {
			bcd[i] = asc_to_bcd(ascii[j++]);
			bcd[i] = (byte) (((j >= asc_len) ? 0x00 : asc_to_bcd(ascii[j++])) + (bcd[i] << 4));
		}
		return bcd;
	}
	public static byte asc_to_bcd(byte asc) {
		byte bcd;

		if ((asc >= '0') && (asc <= '9'))
			bcd = (byte) (asc - '0');
		else if ((asc >= 'A') && (asc <= 'F'))
			bcd = (byte) (asc - 'A' + 10);
		else if ((asc >= 'a') && (asc <= 'f'))
			bcd = (byte) (asc - 'a' + 10);
		else
			bcd = (byte) (asc - 48);
		return bcd;
```

```
    }
    /**
     * BCD 转字符串
     */
    public static String bcd2Str(byte[] bytes) {
        char temp[] = new char[bytes.length * 2], val;

        for (int i = 0; i < bytes.length; i++) {
            val = (char) (((bytes[i] & 0xf0) >> 4) & 0x0f);
            temp[i * 2] = (char) (val > 9 ? val + 'A' - 10 : val +
'0');

            val = (char) (bytes[i] & 0x0f);
            temp[i * 2 + 1] = (char) (val > 9 ? val + 'A' - 10 : val
+ '0');
        }
        return new String(temp);
    }
    /**
     * 拆分字符串
     */
    public static String[] splitString(String string, int len) {
        int x = string.length() / len;
        int y = string.length() % len;
        int z = 0;
        if (y != 0) {
            z = 1;
        }
        String[] strings = new String[x + z];
        String str = "";
        for (int i=0; i<x+z; i++) {
            if (i==x+z-1 && y!=0) {
                str = string.substring(i*len, i*len+y);
            }else{
                str = string.substring(i*len, i*len+len);
            }
            strings[i] = str;
        }
        return strings;
    }
    /**
     *拆分数组
     */
    public static byte[][] splitArray(byte[] data,int len){
        int x = data.length / len;
        int y = data.length % len;
        int z = 0;
        if(y!=0){
            z = 1;
```

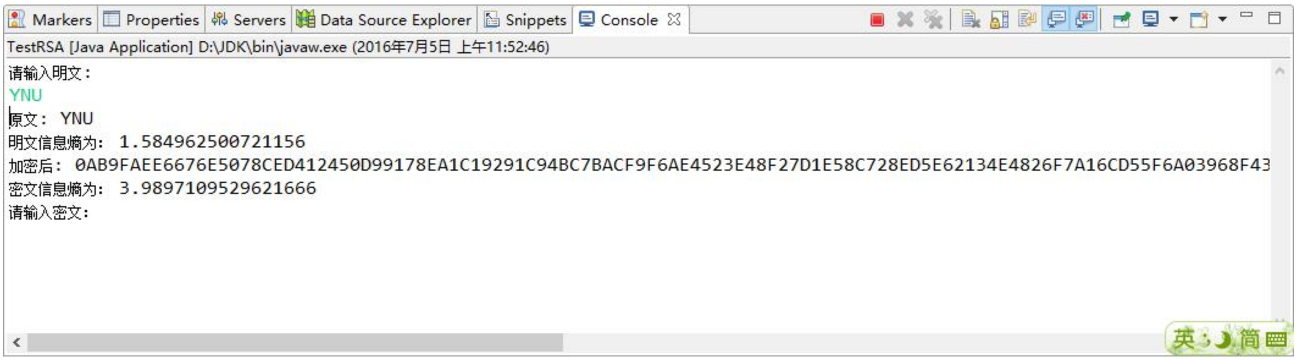
```
        }
        byte[][] arrays = new byte[x+z][];
        byte[] arr;
        for(int i=0; i<x+z; i++){
            arr = new byte[len];
            if(i==x+z-1 && y!=0){
                System.arraycopy(data, i*len, arr, 0, y);
            }else{
                System.arraycopy(data, i*len, arr, 0, len);
            }
            arrays[i] = arr;
        }
        return arrays;
    }
}
```

TestRSA.java

```
package IT11;

import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.util.HashMap;
import java.util.Scanner;

public class TestRSA  {
    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        HashMap<String, Object> map = RSA.getKeys();
        //生成公钥和私钥
        RSAPublicKey publicKey = (RSAPublicKey) map.get("public");
        RSAPrivateKey privateKey = (RSAPrivateKey) map.get("private");

        //模
        String modulus = publicKey.getModulus().toString();
        //公钥指数
        String public_exponent = 
publicKey.getPublicExponent().toString();
        //私钥指数
        String private_exponent = 
privateKey.getPrivateExponent().toString();

        //使用模和指数生成公钥和私钥
        RSAPublicKey pubKey = RSA.getPublicKey(modulus,
```

```
public_exponent);
            RSAPrivateKey priKey = RSA.getPrivateKey(modulus,
private_exponent);



            System.out.println("请输入明文:");
      Scanner input=new Scanner(System.in);
      String message=input.next();

        System.out.println("原文: " + message);
        System.out.println("明文信息熵为:   "+ ENTROPY.entropy(message));



        //加密后的密文
            String encryptData = RSA.encryptByPublicKey(message, pubKey);
        System.out.println("加密后: " + encryptData);
        System.out.println("密文信息熵为:   "+
ENTROPY.entropy(encryptData));

        System.out.println("请输入密文: ");

        String newEncryptData=input.next();

        String decryptData = RSA.decryptByPrivateKey(newEncryptData,
priKey);
        System.out.println("解密后: " + decryptData);


      }

}
```

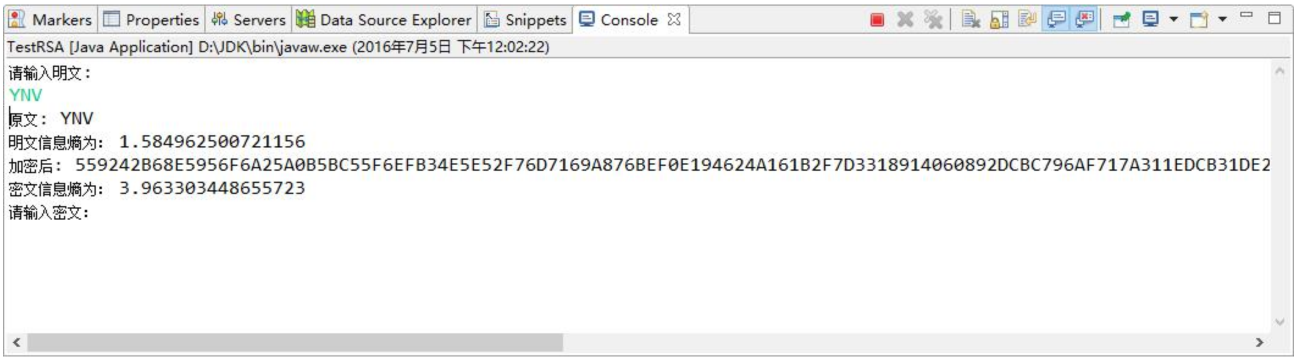
（2） 任取一段输入数据作为明文，计算明文熵；将密码系统作用于明文，得到密文，计算密文熵。比较明文熵和密文熵。



明文熵<密文熵.

（3） 改变明文 1bit，观察密文的变化。改变密钥 1bit，观察密文的变化。

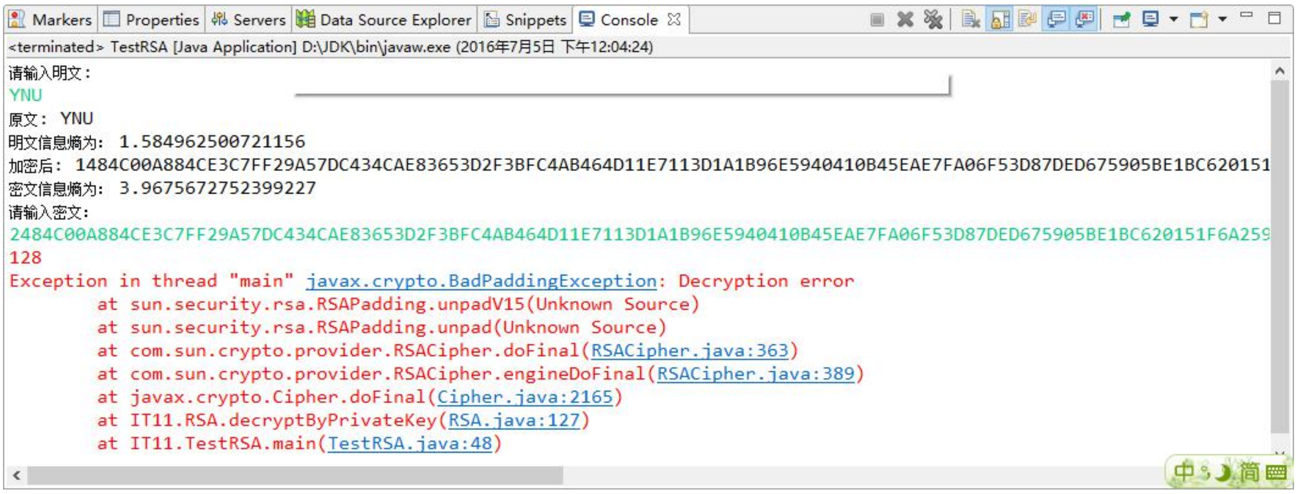
改变明文 1bit:



密文的内容发生了很大的变化，但是熵变化不大

（4） 改变密文 1bit，观察解密后的明文变化。

解密出错；



（5）分析（2）-（4）中的实验现象和原因。

2. Elgamal 体制的实现与分析

（1）实现密码体制。

ElGamalKeyPairGenerator.java

```java
package IT11;


import java.math.BigInteger;
import java.security.*;

public class ElGamalKeyPairGenerator extends KeyPairGeneratorSpi {
  private int mStrength = 0;
  private SecureRandom mSecureRandom = null;

  public void initialize(int strength, SecureRandom random) {
    mStrength = strength;
    mSecureRandom = random;
  }

  public KeyPair generateKeyPair() {
    if (mSecureRandom == null) {
      mStrength = 1024;
      mSecureRandom = new SecureRandom();
    }
    BigInteger p = new BigInteger(mStrength, 16, mSecureRandom);
    BigInteger g = new BigInteger(mStrength - 1, mSecureRandom);
    BigInteger k = new BigInteger(mStrength - 1, mSecureRandom);
    BigInteger y = g.modPow(k, p);
```

```
    ElGamalPublicKey publicKey = new ElGamalPublicKey(y, g, p);
    ElGamalPrivateKey privateKey = new ElGamalPrivateKey(k, g, p);
    return new KeyPair(publicKey, privateKey);
  }

}
```

ElGamalPrivateKey.java

```
package IT11;


import java.math.BigInteger;
import java.security.*;

public class ElGamalPrivateKey extends ElGamalKey implements PrivateKey {

     private BigInteger mK;

     protected ElGamalPrivateKey(BigInteger k, BigInteger g, BigInteger p)
{
     super(g, p);
      mK = k;
     }
     protected BigInteger getK() { return mK; }

     public String toString()
     {
        return  mK + ":" + getG() + ":" + getP();
     }

}
```

ElGamalPublicKey.java

```
package IT11;


import java.math.BigInteger;
import java.security.*;

public class ElGamalPublicKey extends ElGamalKey implements PublicKey {
```

```
    private BigInteger mY;

    protected ElGamalPublicKey(BigInteger y, BigInteger g, BigInteger p)
{
    super(g, p);
    mY = y;
  }
  protected BigInteger getY() { return mY; }

       public String toString()
     {
       return  mY + ":" + getG() + ":" + getP();
     }
}
```

ElGamalKey.java

```
package IT11;


import java.math.BigInteger;
import java.security.*;

public class ElGamalKey implements Key {
  private BigInteger mP, mG;

  protected ElGamalKey(BigInteger g, BigInteger p) {
    mG = g;
    mP = p;
  }

  protected BigInteger getG() { return mG; }
  protected BigInteger getP() { return mP; }

  public String getAlgorithm() { return "ElGamal"; }
  public String getFormat() { return "NONE"; }
  public byte[] getEncoded() { return null; }
}
```

ElGamalEncryption.java

```
package IT11;
```

```
import java.math.BigInteger;
import java.security.*;

public class ElGamalEncryption{

    protected ElGamalKey mKey;

    protected static BigInteger kOne = BigInteger.valueOf(1);

    protected void engineInitEncrypt(PublicKey key) throws
InvalidKeyException {
    if (!(key instanceof ElGamalPublicKey)) throw new
InvalidKeyException("Invalid ElGamalPublicKey.");
    mKey = (ElGamalKey)key;
    }

    protected void engineInitDecrypt(PrivateKey key) throws
InvalidKeyException {
    if (!(key instanceof ElGamalPrivateKey)) throw new
InvalidKeyException("Invalid ElGamalPrivateKey.");
    mKey = (ElGamalKey)key;
    }

    protected BigInteger[] engineEncrypt(BigInteger M){
    BigInteger y = ((ElGamalPublicKey)mKey).getY();
    BigInteger g = mKey.getG();
    BigInteger p = mKey.getP();

    BigInteger k;
    do {
      k = new BigInteger(p.bitLength() - 1, new SecureRandom());
    } while (k.gcd(p).equals(kOne) == false);

    BigInteger a = g.modPow(k, p);

    BigInteger temp = y.modPow(k, p);
    BigInteger C = (M.multiply(temp)).mod(p);
    BigInteger[] result = new BigInteger[2];
    result[0] = a;
    result[1] = C;

    return result;
  }

protected BigInteger engineDecrypt(BigInteger[] result){
    BigInteger k = ((ElGamalPrivateKey)mKey).getK();
    BigInteger p = mKey.getP();

    BigInteger a = result[0];
```

```
    BigInteger C = result[1];
    BigInteger temp = a.modPow(k, p).modInverse(p);


    return C.multiply(temp).mod(p);
  }


}
```

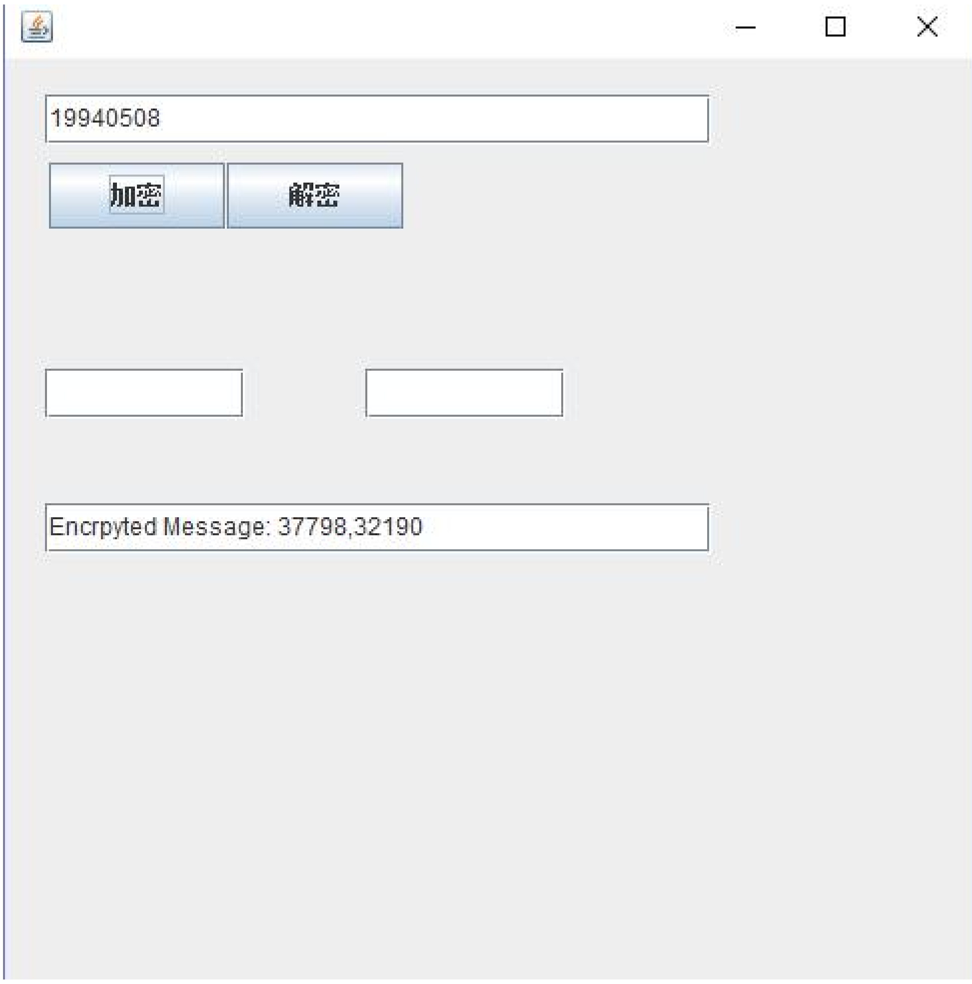
Ekeygen.java

```
package IT11;



import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.math.BigInteger;
import java.security.*;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Ekeygen {

    static ElGamalEncryption encrypt;
    static ElGamalKeyPairGenerator ekpg;
    static KeyPair epair;

    public static void main(String[] args){
      JFrame jiami=new JFrame();
      final ElGamalPrivateKey eprik;
        final ElGamalPublicKey epubk;
        ekpg = new ElGamalKeyPairGenerator();
        ekpg.initialize(16, new SecureRandom());
        epair = ekpg.generateKeyPair();

        eprik = (ElGamalPrivateKey) epair.getPrivate();
        epubk = (ElGamalPublicKey) epair.getPublic();


        System.out.println("Private Key: k = " + eprik.getK() );
```

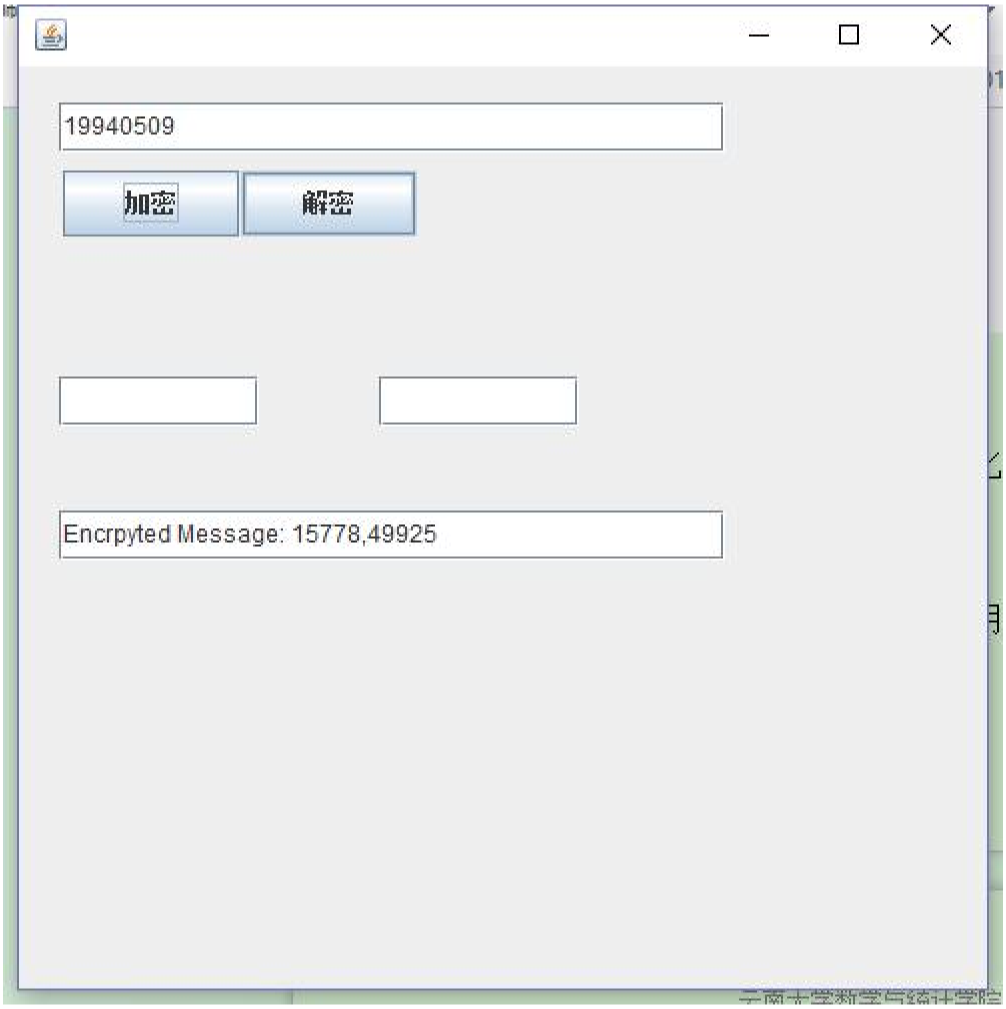
```
        System.out.println("Public Key: y = " + epubk.getY() + ", g = " +
epubk.getG() + ", p = " + epubk.getP());

       /* try
        {

             String str = "45678";

             System.out.println("Message : " + str);

             System.out.println();

             BigInteger C;
             //encrypt.engineInitDecrypt(eprik);

        }   */
        //catch(InvalidKeyException ike)
        //{
           // System.out.println("Invalid Key!");
        //}

      jiami.setSize(500, 500);
      final JTextField xianshi=new JTextField();
      xianshi.setBounds(20, 222, 333, 25);
      jiami.setLayout(null);
      jiami.add(xianshi);
      final JTextField elgam=new JTextField();
      elgam.setBounds(20, 18, 333, 25);
      jiami.setLayout(null);
      jiami.add(elgam);
      final JTextField elga1=new JTextField();
      elga1.setBounds(20, 155, 100, 25);
      jiami.setLayout(null);
      jiami.add(elga1);
      final JTextField elga2=new JTextField();
```

```
    elga2.setBounds(180, 155, 100, 25);
    jiami.setLayout(null);
    jiami.add(elga2);
    JButton an=new JButton();
    an.setBounds(22, 52, 88, 33);
    an.setText("加密");
    an.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        String ssa=elgam.getText();
        encrypt = new ElGamalEncryption();
        try {
                encrypt.engineInitEncrypt(epubk);
            } catch (InvalidKeyException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        System.out.println(ssa);
        BigInteger msg_num = new BigInteger(ssa);

        BigInteger[] encryptedmsg = encrypt.engineEncrypt(msg_num);
        xianshi.setText("Encrpyted Message: " + encryptedmsg[0] + ","
+ encryptedmsg[1]);
        }
    });
    jiami.add(an);
    JButton an2=new JButton();
    an2.setBounds(111, 52, 88, 33);
    an2.setText("解密");
    jiami.add(an2);
    an2.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){

        String ss1=elga1.getText();
        String ss2=elga2.getText();

        encrypt = new ElGamalEncryption();
        try {
                encrypt.engineInitDecrypt(eprik);
            } catch (InvalidKeyException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }


            BigInteger encryptedmsg = new BigInteger(ss1);
            BigInteger encryptedmsg1 = new BigInteger(ss2);
            BigInteger[] enmsg = new BigInteger[2];
          enmsg[0]=encryptedmsg;
          enmsg[1]=encryptedmsg1;
```

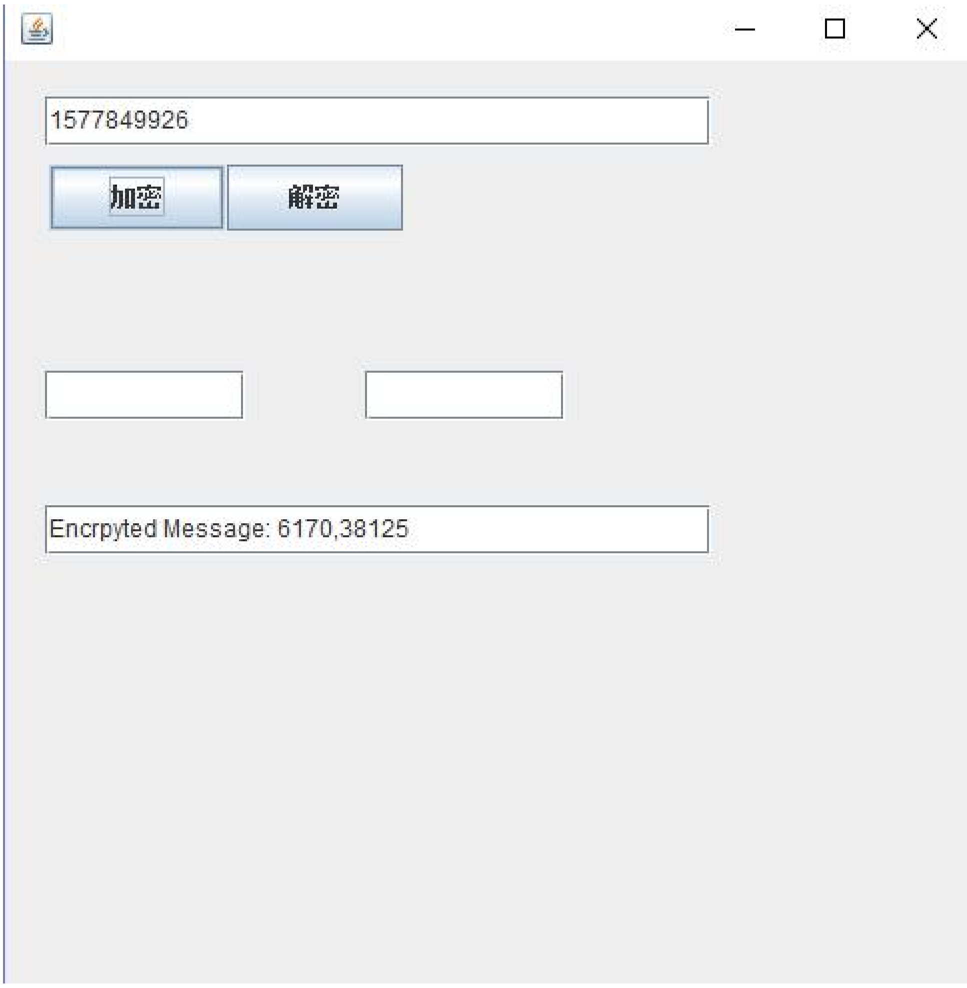
```
            BigInteger decryptedmsg = encrypt.engineDecrypt(enmsg);
            xianshi.setText("Decrypted Message: " +decryptedmsg);
            }
        });



        jiami.setVisible(true);

    }

}
```



（2）改变明文 1bit，观察密文的变化。改变密钥 1bit，观察密文的变化。



（4）改变密文 1bit，观察解密后的明文变化。



1 bit 的变化使得结果完全不同。

五、实验体会

1. ElGamal 算法既能用于数据加密也能用于数字签名，其安全性依赖于计算有限域上离散对数这一难题。

密钥对产生办法。首先选择一个素数 p，两个随机数, g 和 x，g, x < p, 计算 y = g^x (mod p)，则其公钥为 y, g 和 p。私钥是 x。g 和 p 可由一组用户共享。

ElGamal 用于数字签名。被签信息为 M，首先选择一个随机数 k, k 与 p - 1 互质，计算 a = g^k (mod p)，再用扩展 Euclidean 算法对方程求解 b：M = xa + kb (mod p - 1)，签名就是(a, b)。随机数 k 须丢弃。

验证时要验证下式：y^a * a^b (mod p) = g^M (mod p)

同时一定要检验是否满足 $1 <= a < p$。否则签名容易伪造。

ElGamal 用于加密。被加密信息为 M，首先选择一个随机数 k，k 与 p - 1 互质，计算 a = g^k (mod p)，b = y^k M (mod p)，(a, b)为密文，是明文的两倍长。解密时计算 M = b / a^x (mod p)。

ElGamal 签名的安全性依赖于乘法群(IFp)* 上的离散对数计算。素数 p 必须足够大，且 p-1 至少包含一个大素数因子以抵抗 Pohlig & Hellman 算法的攻击。M 一般都应采用信息的 HASH 值(如 SHA 算法)。ElGamal 的安全性主要依赖于 p 和 g，若选取不当则签名容易伪造，应保证 g 对于 p-1 的大素数因子不可约。D.Bleichenbache“GeneratingElGamal Signatures Without Knowing the Secret Key”中提到了一些攻击方法和对策。ElGamal 的一个不足之处是它的密文成倍扩张。

2.公钥密码体制的核心思想是：加密和解密采用不同的密钥。这是公钥密码体制和传统的对称密码体制最大的区别。对于传统对称密码而言，密文的安全性完全依赖于密钥的保密性，一旦密钥泄漏，将毫无保密性可言。但是公钥密码体制彻底改变了这一状况。在公钥密码体制中，公钥是公开的，只有私钥是需要保密的。知道公钥和密码算法要推测出私钥在计算上是不可行的。这样，只要私钥是安全的，那么加密就是可信的。

显然，对称密码和公钥密码都需要保证密钥的安全，不同之处在于密钥的管理和分发上面。在对称密码中，必须要有一种可靠的手段将加密密钥（同时也是解密密钥）告诉给解密方；而在公钥密码体制中，这是不需要的。解密方只需要保证自己的私钥的保密性即可，对于公钥，无论是对加密方而言还是对密码分析者而言都是公开的，故无需考虑采用可靠的通道进行密码分发。这使得密钥管理和密钥分发的难度大大降低了。

3.加密会增加数据的冗余这会导致密文的熵变大；且由信源绝对信息率定义为 $R_0 = \log|A|$，信源的近似信息率定义为 $R_n = \dfrac{\log|B_n|}{n}$，可得明文熵 $H(X^n) = nR_n = \log|B_n|$，密文熵 $H(Y^n) = nR_0 = n\log|A|$，后者显然更大；

六、参考文献

1. Thomas M. Cover, Joy A. Thomas. Elements of Information Theory (2nd Edition) [M]. John Wiley & Sons, Inc.

2.（如有其它参考文献，请列出）