**云南大学数学与与统计学院**

**上机实践报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：近代密码学实验 | **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**：离散对数问题实验 | **学号**：20131910023 | **上机实践日期**： **9.14** |
| **上机实践编号**：No.03 | **组号**： | **上机实践时间**： **16:52** |

**一、实验目的**

熟悉离散对数问题(DLP)及其有关的密码体制

1. **实验内容**
2. 编程离散对数问题(DLP)有关的算法
3. 编程实现Diffie-Hellman体制
4. 编程实现ElGamal体制

**三、实验环境**

个人计算机，Java 8平台

对于非信息与计算科学专业的学生，可以选择任意编程平台

**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. 编程离散对数问题(DLP)有关的算法

需要用到快速模幂算法：

**public** **long** fastPowering(**int** g,**int** A,**int** N) {

**int** r=(**int**) (Math.*log*(A)/Math.*log*(2));

**long**[] a=**new** **long**[r+1];

**int**[] b=**new** **int**[r+1];

**long**[] tp=**new** **long**[r+1];

**int** i;

/\* 2^i \*/

tp[0]=1;

**for** (i=1;i<=r;i++) tp[i]=2\*tp[i-1];

i=r;

**int** G=A;

**while** (G>0) {

//System.out.println("Jin"+i+" "+G+" "+tp[i]);

**if** (G>=tp[i]) {

b[i]=1;

G-=tp[i];

}

**else** b[i]=0;

i--;

}

a[0]=g % N;

**for** (i=1;i<=r;i++) a[i]=a[i-1]\*a[i-1] % N;

**long** ans=1;

**for** (i=0;i<=r;i++)

**if** (b[i]==1) ans=ans \* a[i] % N;

**return** ans;

}

大整数版本：

**public** BigInteger fastPowering(BigInteger g,BigInteger A,BigInteger N) {

BigInteger a=g;

BigInteger b=**new** BigInteger("1");

BigInteger one=**new** BigInteger("1");

BigInteger two=**new** BigInteger("2");

**while** (A.compareTo(zero)!=0) {

**if** (A.mod(two).compareTo(one)==0) b=b.multiply(a).mod(N);

a=a.multiply(a);

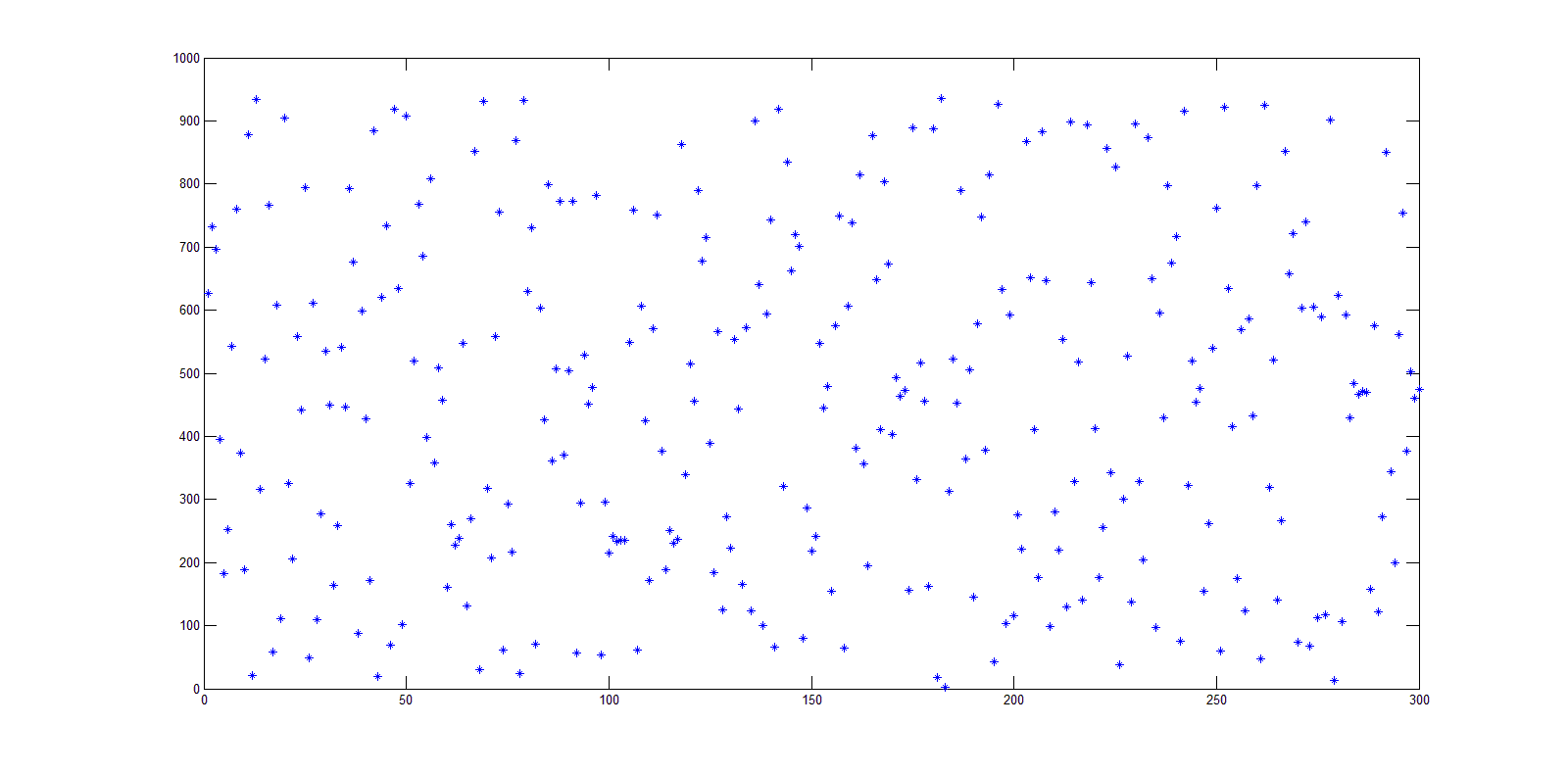
A=A.divide(two);

}

**return** b;

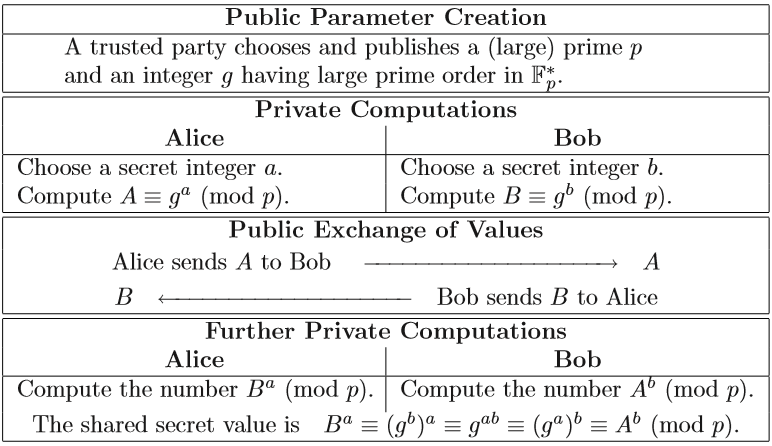
}

结合matlab画出了的图像



1. 编程实现Diffie-Hellman体制

Diffie-Hellman的主要过程如下



**public** **void** Diffie\_Hellman() {

BigInteger p,g,a,A,b,B,keyA,keyB;

p=createBigPrime(500);

g=createRandomInt();

System.***out***.println("A trusted party chooses and publishes a large prime p="+p+" and an integer g="+g);

System.***out***.println();

FundAl FA=**new** FundAl();

/\*Alice\*/

a=createRandomInt();

A=FA.fastPowering(g,a,p);

/\*Bob\*/

b=createRandomInt();

B=FA.fastPowering(g,b,p);

System.***out***.println("Alice sends "+A+" to Bob————————>A;");

System.***out***.println("B<————————Bob sends "+B+" to Alice;");

System.***out***.println();

keyA=FA.fastPowering(B,a,p);

keyB=FA.fastPowering(A,b,p);

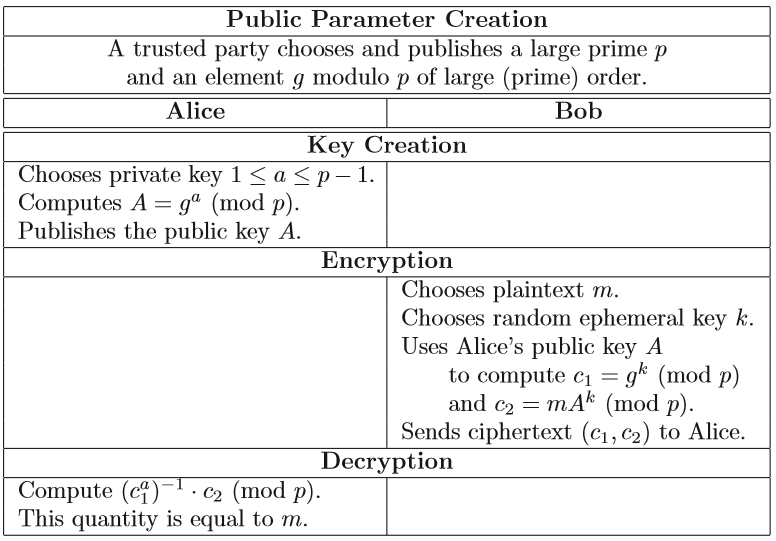
System.***out***.println("Alice cpmoutes the secret value keyA="+keyA);

System.***out***.println("Bob cpmoutes the secret value keyB="+keyB);

}

1. 编程实现ElGamal体制

ElGamal体制的主要过程如下：



**public** **void** ElGamal() {

BigInteger p,g,a,A,m,k,c1,c2;

p=createBigPrime(500);

g=createRandomInt();

System.***out***.println("A trusted party chooses and publishes a large prime p="+p+" and an integer g="+g);

System.***out***.println();

FundAl FA=**new** FundAl();

/\*Alice\*/

a=createRandomInt();

A=FA.fastPowering(g,a,p);

System.***out***.println("Alice chooses private key,then publiches the public key A="+A);

System.***out***.println();

/\*Bob\*/

m=createRandomInt();

System.***out***.println("Bob wants to send plaintext m="+m);

k=createRandomInt();

c1=FA.fastPowering(g,k,p);

c2=m.multiply(FA.fastPowering(A,k,p));

System.***out***.println("Bob sends ciphertext (c1,c2)=("+c1+","+c2+") to Alice.");

System.***out***.println();

/\*Alice\*/

BigInteger t=FA.fastPowering(c1,a,p);

FA.extendedEuclidean(t, p);

t=FA.getU().mod(p).abs();

t=t.multiply(c2).mod(p);

System.***out***.println("After computation, Alice gets message="+t);

}

1. 完整程序

**FundAl.java**

**package** MC03;

**import** java.math.BigInteger;

**public** **class** FundAl {

**protected** BigInteger u;

**protected** BigInteger v;

**protected** BigInteger zero;

**public** FundAl() {

zero=**new** BigInteger("0");

}

**public** BigInteger getU() {

**return** u;

}

**public** BigInteger getV() {

**return** v;

}

**public** BigInteger euclidean(BigInteger a,BigInteger b) {

**if** (b.compareTo(zero)==0) **return** a;

**return** euclidean(b,a.mod(b));

}

**public** BigInteger extendedEuclidean(BigInteger a,BigInteger b) {

**if** (b.compareTo(zero)==0) {

u=**new** BigInteger("1");

v=**new** BigInteger("0");

**return** a;

}

BigInteger r= extendedEuclidean(b,a.mod(b));

BigInteger t=u;

u=v;

v=t.subtract(a.divide(b).multiply(v));

**return** r;

}

**public** BigInteger fastPowering(BigInteger g,BigInteger A,BigInteger N) {

BigInteger a=g;

BigInteger b=**new** BigInteger("1");

BigInteger one=**new** BigInteger("1");

BigInteger two=**new** BigInteger("2");

**while** (A.compareTo(zero)!=0) {

**if** (A.mod(two).compareTo(one)==0) b=b.multiply(a).mod(N);

a=a.multiply(a).mod(N);

A=A.divide(two);

}

**return** b;

}

}

**DLP.java**

**package** MC03;

**import** java.util.Random;

**import** java.math.BigInteger;

**import** java.util.Scanner;

**public** **class** DLP {

**public** DLP() {

}

**public** BigInteger createBigPrime(**int** len) {

BigInteger p;

**do** {

p=**new** BigInteger(len, 10, **new** Random());//此构造函数用于构造一个随机生成正BigInteger的可能是以指定的len的素数。可能性超过1-2^(-10)

} **while** (!p.isProbablePrime(10));//是素数则跳出构造

**return** p;

}

**public** BigInteger createRandomInt() {

Random rand = **new** Random();

**return**(**new** BigInteger(rand.nextInt(8999)+1000+""));//产生一个四位整数

}

**public** **void** Diffie\_Hellman() {

BigInteger p,g,a,A,b,B,keyA,keyB;

p=createBigPrime(500);

g=createRandomInt();

System.***out***.println("A trusted party chooses and publishes a large prime p="+p+" and an integer g="+g);

System.***out***.println();

FundAl FA=**new** FundAl();

/\*Alice\*/

a=createRandomInt();

A=FA.fastPowering(g,a,p);

/\*Bob\*/

b=createRandomInt();

B=FA.fastPowering(g,b,p);

System.***out***.println("Alice sends "+A+" to Bob————————>A;");

System.***out***.println("B<————————Bob sends "+B+" to Alice;");

System.***out***.println();

keyA=FA.fastPowering(B,a,p);

keyB=FA.fastPowering(A,b,p);

System.***out***.println("Alice cpmoutes the secret value keyA="+keyA);

System.***out***.println("Bob cpmoutes the secret value keyB="+keyB);

}

**public** **void** ElGamal() {

BigInteger p,g,a,A,m,k,c1,c2;

p=createBigPrime(500);

g=createRandomInt();

System.***out***.println("A trusted party chooses and publishes a large prime p="+p+" and an integer g="+g);

System.***out***.println();

FundAl FA=**new** FundAl();

/\*Alice\*/

a=createRandomInt();

A=FA.fastPowering(g,a,p);

System.***out***.println("Alice chooses private key,then publiches the public key A="+A);

System.***out***.println();

/\*Bob\*/

m=createRandomInt();

System.***out***.println("Bob wants to send plaintext m="+m);

k=createRandomInt();

c1=FA.fastPowering(g,k,p);

c2=m.multiply(FA.fastPowering(A,k,p));

System.***out***.println("Bob sends ciphertext (c1,c2)=("+c1+","+c2+") to Alice.");

System.***out***.println();

/\*Alice\*/

BigInteger t=FA.fastPowering(c1,a,p);

FA.extendedEuclidean(t, p);

t=FA.getU().mod(p).abs();

t=t.multiply(c2).mod(p);

System.***out***.println("After computation, Alice gets message="+t);

}

}

**TestDLP.java**

**package** MC03;

**public** **class** TestDLP {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

DLP dlp=**new** DLP();

System.***out***.println("Diffie-Hellman体制:");

dlp.Diffie\_Hellman();

System.***out***.println();

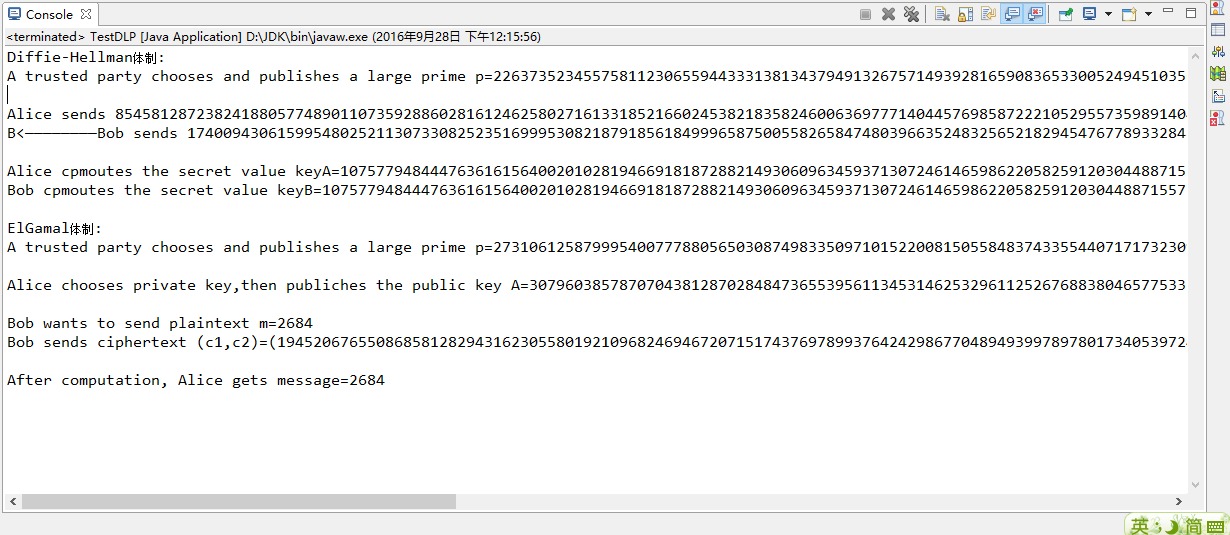
System.***out***.println("ElGamal体制:");

dlp.ElGamal();

}

}

运行结果截图：



即：

Diffie-Hellman体制:

A trusted party chooses and publishes a large prime p=2263735234557581123065594433313813437949132675714939281659083653300524945103562906065484401904402869482526066078068343994171157335853635938731902927407 and an integer g=8431

Alice sends 854581287238241880577489011073592886028161246258027161331852166024538218358246006369777140445769858722210529557359891404759536686496484135021800565625 to Bob————————>A;

B<————————Bob sends 1740094306159954802521130733082523516999530821879185618499965875005582658474803966352483256521829454767789332848576126548080369775665777830402926887480 to Alice;

Alice cpmoutes the secret value keyA=1075779484447636161564002010281946691818728821493060963459371307246146598622058259120304488715577956518207306806059602633203319650431197962361753576467

Bob cpmoutes the secret value keyB=1075779484447636161564002010281946691818728821493060963459371307246146598622058259120304488715577956518207306806059602633203319650431197962361753576467

ElGamal体制:

A trusted party chooses and publishes a large prime p=2731061258799954007778805650308749833509710152200815055848374335544071717323094044685563826407469312984305621973030662033284230235053019015802440318461 and an integer g=7164

Alice chooses private key,then publiches the public key A=307960385787070438128702848473655395611345314625329611252676883804657753331065331317825521590565553163972281865283652499447173585235434238636685646523

Bob wants to send plaintext m=2684

Bob sends ciphertext (c1,c2)=(1945206765508685812829431623055801921096824694672071517437697899376424298677048949399789780173405397241734332797031007024967258388685268261595294969545,3010896818416518943980036656139352861554338124971172926332465509711492528535382127831240479907260414455357681253281079854752065209363492205681757092789164) to Alice.

After computation, Alice gets message=2684

**五、实验体会**

**（请认真填写自己的真实体会）**

1. 在以上两个体制中，需要先由可信方提供一个大素数，（一般认为目前500位以上为可行），可以使用BigInteger(int bitLength, int certainty, Random rnd)：此构造函数用于构造一个随机生成正BigInteger的可能是以指定的bitLength的素数。

这个生成的数不一定是素数，故使用了isProbablePrime()方法来进一步判断，本实验中是素数的概率高于1-2^(-10),但是仍然无法保证100%该数为素数。

1. 在ElGamal中需要计算模拟，若使用费马小定理结合快速模幂运算，只需找到一个与a互素的数p，以p-2作为次数来计算。本实验中曾经尝试直接使用可信方发布的500位大整数p，导致模幂运算花费了很多时间。本实验直接采用了拓展欧几里得算法来求模逆。

**六、参考文献**

1. 主讲课教材（数学密码学导论）第二章

**2.（如有其它参考文献，请列出）**