**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**：椭圆曲线离散对数问题实验 | **学号**：20131910023 | **上机实践日期**： **10.12** |
| **上机实践编号**：No.06 | **组号**： | **上机实践时间**：16:33 |

**一、实验目的**

1. 熟悉椭圆曲线离散对数问题(ECDLP)及其有关的密码体制
2. 实现与ECDLP有关的基本算法
3. 了解参数与参数规模
4. **实验内容**
5. 编程椭圆曲线离散对数问题(ECDLP)有关的算法
6. 编程实现Diffie-Hellman密钥交换协议的椭圆曲线版本。
7. 编程实现ElGamal加密体制的椭圆曲线版本。

说明：

* **基础有限域为素域GF(p)（p为大素数）的情形为必做实验**
* **基础有限域为GF（2\*\*m）的情形为选做实验**

**三、实验环境**

个人计算机，建议选择Java 8平台。

对于基础有限域为素域GF(p)的实验情形，可利用Java的BigInteger实现

对于基础有限域为GF（2\*\*m的实验情形，可采用支持有限域的任意平台（包括数学软件）进行研究型或验证型实验。

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. 利用MATLAB绘制椭圆曲线

①

k=0;

hold on

for x=-5:0.01:8

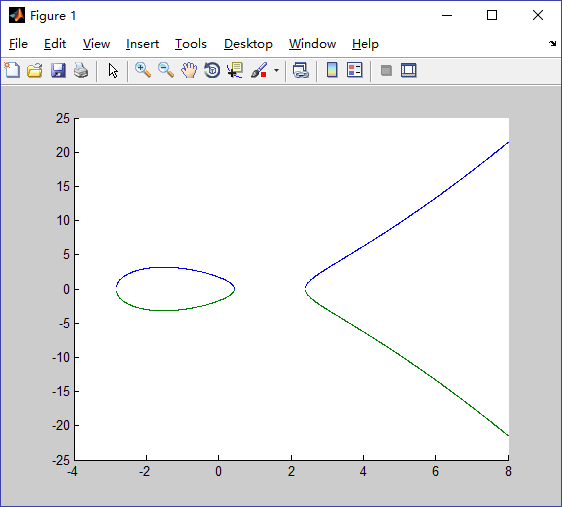
y2=x^3-7\*x+3;

if (y2>=0)

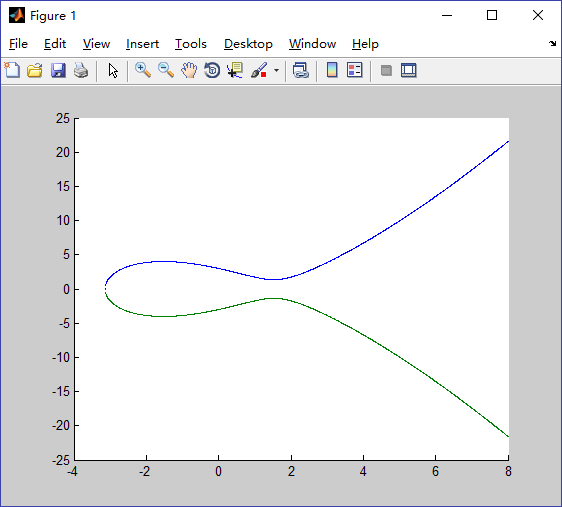
plot(x,sqrt(y2),x,-sqrt(y2))

end

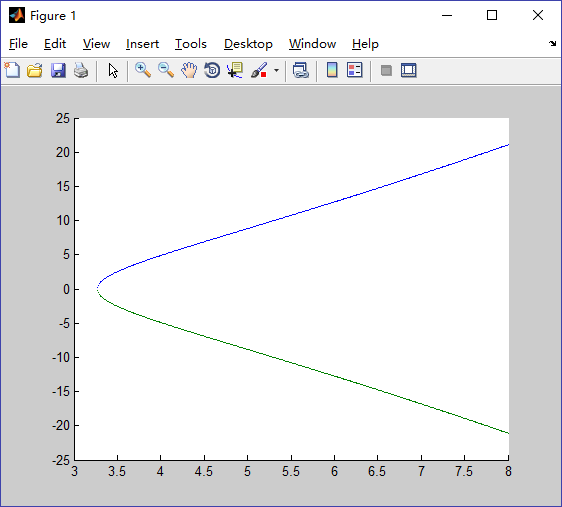
end



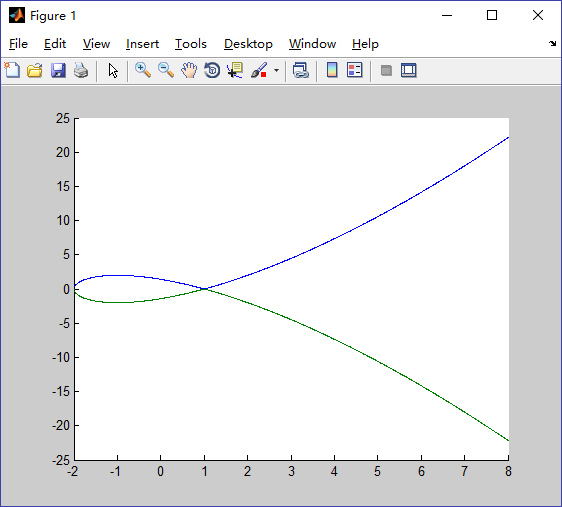
②



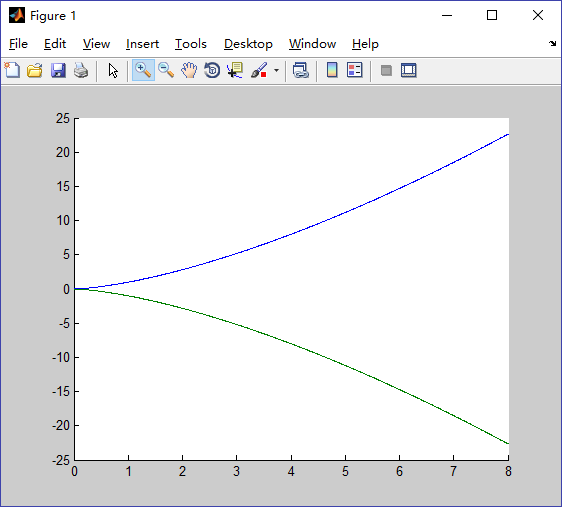
③



④



⑤



1. 编程椭圆曲线离散对数问题(ECDLP)有关的算法

①加法计算的实现：

**public** BigInteger[] calculateAnsPoint(BigInteger[] P,BigInteger[] Q) {

BigInteger lam;

BigInteger[] ans=**new** BigInteger[2];

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

/\*基本情况\*/

**if** (P[0].compareTo(***INF***)==0 && P[1].compareTo(***INF***)==0) **return** Q;

**if** (Q[0].compareTo(***INF***)==0 && Q[1].compareTo(***INF***)==0) **return** P;

**if** (P[0].compareTo(Q[0])==0 && (P[1].add(Q[1])).compareTo(p)==0 ) {

ans[0]=***INF***;

ans[1]=***INF***;

**return** ans;

}

**if** (P[0].compareTo(Q[0])==0 && P[1].compareTo(Q[1])==0) {

FA.extendedEuclidean(***TWO***.multiply(P[1]), p);

//"a^(-1)="+FA.getU().mod(p).abs();

lam=(***THREE***.multiply(P[0].multiply(P[0])).add(A)).multiply(FA.getU().mod(p).abs()).mod(p);//(3 x1^2+A)/(2y1)

}

**else** {

FA.extendedEuclidean(Q[0].subtract(P[0]), p);

lam=(Q[1].subtract(P[1])).multiply(FA.getU().mod(p).abs()).mod(p);// (y2-y1)/(x2-x1)

}

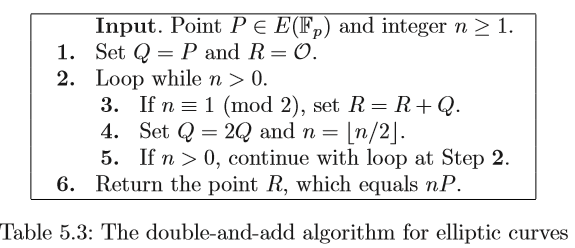
ans[0]=lam.multiply(lam).subtract(P[0]).subtract(Q[0]).mod(p);

ans[1]=lam.multiply(P[0].subtract(ans[0])).subtract(P[1]).mod(p);

**return** ans;

}

② The Double-and-Add Algorithm



**public** BigInteger[] doubleAndAdd(BigInteger n,BigInteger[] P){

BigInteger[] Q=**new** BigInteger[2];

BigInteger[] R=**new** BigInteger[]{***INF***,***INF***};

Q=P;

**while** (n.compareTo(***ZERO***)==1) {

**if** (n.mod(***TWO***).compareTo(***ONE***)==0 ) R=calculateAnsPoint(R,Q);

Q=calculateAnsPoint(Q,Q);

n=n.divide(***TWO***);

}

**return** R;

}

**ECDLP.java**

**package** MC06;

**import** java.math.BigInteger;

**import** MC02.FundAl;

**public** **class** ECDLP {

**protected** BigInteger A,B,p;

**private** **final** **static** BigInteger ***ZERO*** = **new** BigInteger("0");

**private** **final** **static** BigInteger ***ONE*** = **new** BigInteger("1");

**private** **final** **static** BigInteger ***TWO*** = **new** BigInteger("2");

**private** **final** **static** BigInteger ***THREE*** = **new** BigInteger("3");

**private** **final** **static** BigInteger ***INF*** = **new** BigInteger("-1");

**public** ECDLP() {

}

**public** ECDLP(BigInteger A,BigInteger B,BigInteger p) {

**this**.A=A;

**this**.B=B;

**this**.p=p;

}

**public** BigInteger[] calculateAnsPoint(BigInteger[] P,BigInteger[] Q) {

BigInteger lam;

BigInteger[] ans=**new** BigInteger[2];

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

/\*基本情况\*/

**if** (P[0].compareTo(***INF***)==0 && P[1].compareTo(***INF***)==0) **return** Q;

**if** (Q[0].compareTo(***INF***)==0 && Q[1].compareTo(***INF***)==0) **return** P;

**if** (P[0].compareTo(Q[0])==0 && (P[1].add(Q[1])).compareTo(p)==0 ) {

ans[0]=***INF***;

ans[1]=***INF***;

**return** ans;

}

**if** (P[0].compareTo(Q[0])==0 && P[1].compareTo(Q[1])==0) {

FA.extendedEuclidean(***TWO***.multiply(P[1]), p);

//"a^(-1)="+FA.getU().mod(p).abs();

lam=(***THREE***.multiply(P[0].multiply(P[0])).add(A)).multiply(FA.getU().mod(p).abs()).mod(p);//(3 x1^2+A)/(2y1)

}

**else** {

FA.extendedEuclidean(Q[0].subtract(P[0]), p);

lam=(Q[1].subtract(P[1])).multiply(FA.getU().mod(p).abs()).mod(p);// (y2-y1)/(x2-x1)

}

ans[0]=lam.multiply(lam).subtract(P[0]).subtract(Q[0]).mod(p);

ans[1]=lam.multiply(P[0].subtract(ans[0])).subtract(P[1]).mod(p);

**return** ans;

}

**public** BigInteger[] doubleAndAdd(BigInteger n,BigInteger[] P){

BigInteger[] Q=**new** BigInteger[2];

BigInteger[] R=**new** BigInteger[]{***INF***,***INF***};

Q=P;

**while** (n.compareTo(***ZERO***)==1) {

**if** (n.mod(***TWO***).compareTo(***ONE***)==0 ) R=calculateAnsPoint(R,Q);

Q=calculateAnsPoint(Q,Q);

n=n.divide(***TWO***);

}

**return** R;

}

}

**TestECDLP.java**

**package** MC06;

**import** java.math.BigInteger;

**import** java.util.Scanner;

**import** MC02.FundAl;

**public** **class** TestECDLP {

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

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

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("E: Y^2=X^3+AX+B over GF(p)");

System.***out***.println("请输入A B p");

BigInteger A=input.nextBigInteger();

BigInteger B=input.nextBigInteger();

BigInteger p=input.nextBigInteger();

ECDLP EC=**new** ECDLP(A,B,p);

BigInteger[] P=**new** BigInteger[2];

BigInteger[] Q=**new** BigInteger[2];

System.***out***.println("请输入P的坐标");

P[0]=input.nextBigInteger();

P[1]=input.nextBigInteger();

System.***out***.println("请输入Q的坐标");

Q[0]=input.nextBigInteger();

Q[1]=input.nextBigInteger();

BigInteger[] ansR=**new** BigInteger[2];

ansR=EC.calculateAnsPoint(P,Q);

System.***out***.println("R'=P+Q=("+ansR[0]+","+ansR[1]+") in E(GF("+p+")).");

System.***out***.println("请输入n=");

BigInteger n=input.nextBigInteger();

BigInteger[] ansnP=**new** BigInteger[2];

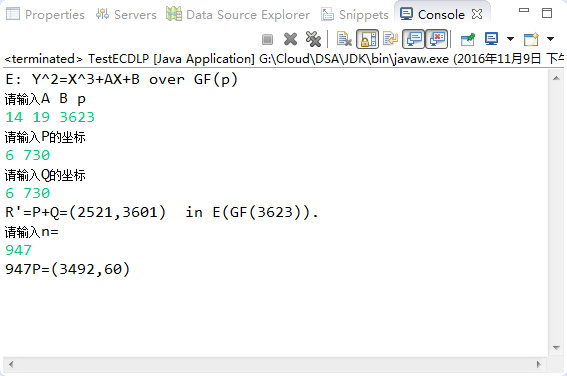
ansnP=EC.doubleAndAdd(n,P);

System.***out***.println(n+"P=("+ansnP[0]+","+ansnP[1]+")");

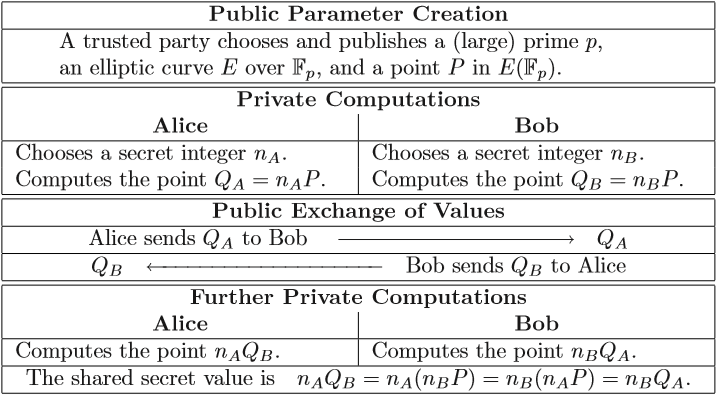
}

}

**针对课本p293 Example 5.16测试结果如下：**



1. 编程实现Diffie-Hellman密钥交换协议的椭圆曲线版本

（完整程序见4）

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

BigInteger p,A,B;

BigInteger nA,nB;

BigInteger[] P=**new** BigInteger[2];

BigInteger[] QA=**new** BigInteger[2];

BigInteger[] QB=**new** BigInteger[2];

BigInteger[] keyA=**new** BigInteger[2];

BigInteger[] keyB=**new** BigInteger[2];

p=createBigPrime(100);

A=createRandomInt();

B=createRandomInt();

P[0]=createRandomInt();

P[1]=createRandomInt();

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

System.***out***.println("E=X^3+"+A+"X+"+B);

System.***out***.println("P=("+P[0]+","+P[1]+")");

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

ECDLP ecdlp=**new** ECDLP(A,B,p);

/\*Alice\*/

nA=createRandomInt();

QA=ecdlp.doubleAndAdd(nA,P);

/\*Bob\*/

nB=createRandomInt();

QB=ecdlp.doubleAndAdd(nB,P);

System.***out***.println("Alice sends ("+QA[0]+","+QA[1]+") to Bob————————>QA;");

System.***out***.println("QB<————————Bob sends ("+QB[0]+","+QB[1]+") to Alice;");

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

keyA=ecdlp.doubleAndAdd(nA,QB);

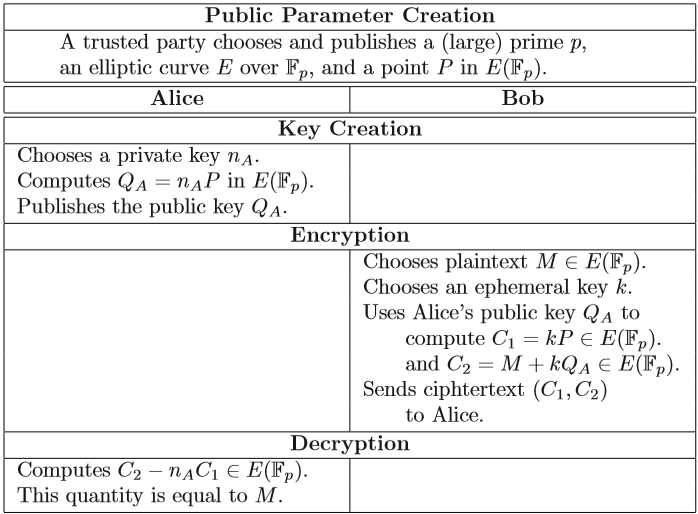
keyB=ecdlp.doubleAndAdd(nB,QA);

System.***out***.println("Alice cpmoutes the secret value keyA=("+keyA[0]+","+keyA[1]+")");

System.***out***.println("Bob cpmoutes the secret value keyB=("+keyB[0]+","+keyB[1]+")");

}

1. 编程实现ElGamal加密体制的椭圆曲线版本

（完整程序见4）

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

BigInteger p,A,B;

BigInteger nA;

BigInteger[] P=**new** BigInteger[2];

BigInteger[] QA=**new** BigInteger[2];

BigInteger k;

BigInteger[] M=**new** BigInteger[2];

BigInteger[] C1=**new** BigInteger[2];

BigInteger[] C2=**new** BigInteger[2];

p=createBigPrime(100);

A=createRandomInt();

B=createRandomInt();

P[0]=createRandomInt();

P[1]=createRandomInt();

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

System.***out***.println("E=X^3+"+A+"X+"+B);

System.***out***.println("P=("+P[0]+","+P[1]+")");

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

ECDLP ecdlp=**new** ECDLP(A,B,p);

/\*Alice\*/

nA=createRandomInt();

QA=ecdlp.doubleAndAdd(nA,P);

System.***out***.println("Alice chooses private key,then publiches the public key QA=("+QA[0]+","+QA[1]+")");

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

/\*Bob\*/

M[0]=createRandomInt();

M[1]=createRandomInt();

System.***out***.println("Bob wants to send plaintext M=("+M[0]+","+M[1]+")");

k=createRandomInt();

C1=ecdlp.doubleAndAdd(k,P);

C2=ecdlp.calculateAnsPoint(M, ecdlp.doubleAndAdd(k,QA));

System.***out***.println("Bob sends ciphertext (C1,C2)=(("+C1[0]+","+C1[1]+"),("+C2[0]+","+C2[1]+")) to Alice.");

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

/\*Alice\*/

BigInteger[] t=**new** BigInteger[2];

t=ecdlp.doubleAndAdd(nA,C1);

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

t[1]=ZERO.subtract(t[1]);

t=ecdlp.calculateAnsPoint(C2, t);

System.***out***.println("After computation, Alice gets message=("+t[0]+","+t[1]+")");

}

1. Elliptic curve cryptography

**ECC.java**

**package** MC06;

**import** java.util.Random;

**import** java.math.BigInteger;

**import** java.util.Scanner;

**public** **class** ECC {

**public** ECC() {

}

**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,A,B;

BigInteger nA,nB;

BigInteger[] P=**new** BigInteger[2];

BigInteger[] QA=**new** BigInteger[2];

BigInteger[] QB=**new** BigInteger[2];

BigInteger[] keyA=**new** BigInteger[2];

BigInteger[] keyB=**new** BigInteger[2];

p=createBigPrime(100);

A=createRandomInt();

B=createRandomInt();

P[0]=createRandomInt();

P[1]=createRandomInt();

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

System.***out***.println("E=X^3+"+A+"X+"+B);

System.***out***.println("P=("+P[0]+","+P[1]+")");

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

ECDLP ecdlp=**new** ECDLP(A,B,p);

/\*Alice\*/

nA=createRandomInt();

QA=ecdlp.doubleAndAdd(nA,P);

/\*Bob\*/

nB=createRandomInt();

QB=ecdlp.doubleAndAdd(nB,P);

System.***out***.println("Alice sends ("+QA[0]+","+QA[1]+") to Bob————————>QA;");

System.***out***.println("QB<————————Bob sends ("+QB[0]+","+QB[1]+") to Alice;");

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

keyA=ecdlp.doubleAndAdd(nA,QB);

keyB=ecdlp.doubleAndAdd(nB,QA);

System.***out***.println("Alice cpmoutes the secret value keyA=("+keyA[0]+","+keyA[1]+")");

System.***out***.println("Bob cpmoutes the secret value keyB=("+keyB[0]+","+keyB[1]+")");

}

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

BigInteger p,A,B;

BigInteger nA;

BigInteger[] P=**new** BigInteger[2];

BigInteger[] QA=**new** BigInteger[2];

BigInteger k;

BigInteger[] M=**new** BigInteger[2];

BigInteger[] C1=**new** BigInteger[2];

BigInteger[] C2=**new** BigInteger[2];

p=createBigPrime(100);

A=createRandomInt();

B=createRandomInt();

P[0]=createRandomInt();

P[1]=createRandomInt();

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

System.***out***.println("E=X^3+"+A+"X+"+B);

System.***out***.println("P=("+P[0]+","+P[1]+")");

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

ECDLP ecdlp=**new** ECDLP(A,B,p);

/\*Alice\*/

nA=createRandomInt();

QA=ecdlp.doubleAndAdd(nA,P);

System.***out***.println("Alice chooses private key,then publiches the public key QA=("+QA[0]+","+QA[1]+")");

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

/\*Bob\*/

M[0]=createRandomInt();

M[1]=createRandomInt();

System.***out***.println("Bob wants to send plaintext M=("+M[0]+","+M[1]+")");

k=createRandomInt();

C1=ecdlp.doubleAndAdd(k,P);

C2=ecdlp.calculateAnsPoint(M, ecdlp.doubleAndAdd(k,QA));

System.***out***.println("Bob sends ciphertext (C1,C2)=(("+C1[0]+","+C1[1]+"),("+C2[0]+","+C2[1]+")) to Alice.");

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

/\*Alice\*/

BigInteger[] t=**new** BigInteger[2];

t=ecdlp.doubleAndAdd(nA,C1);

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

t[1]=ZERO.subtract(t[1]);

t=ecdlp.calculateAnsPoint(C2, t);

System.***out***.println("After computation, Alice gets message=("+t[0]+","+t[1]+")");

}

}

**TestECC.java**

**package** MC06;

**public** **class** TestECC {

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

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

ECC ecc=**new** ECC();

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

ecc.Diffie\_Hellman();

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

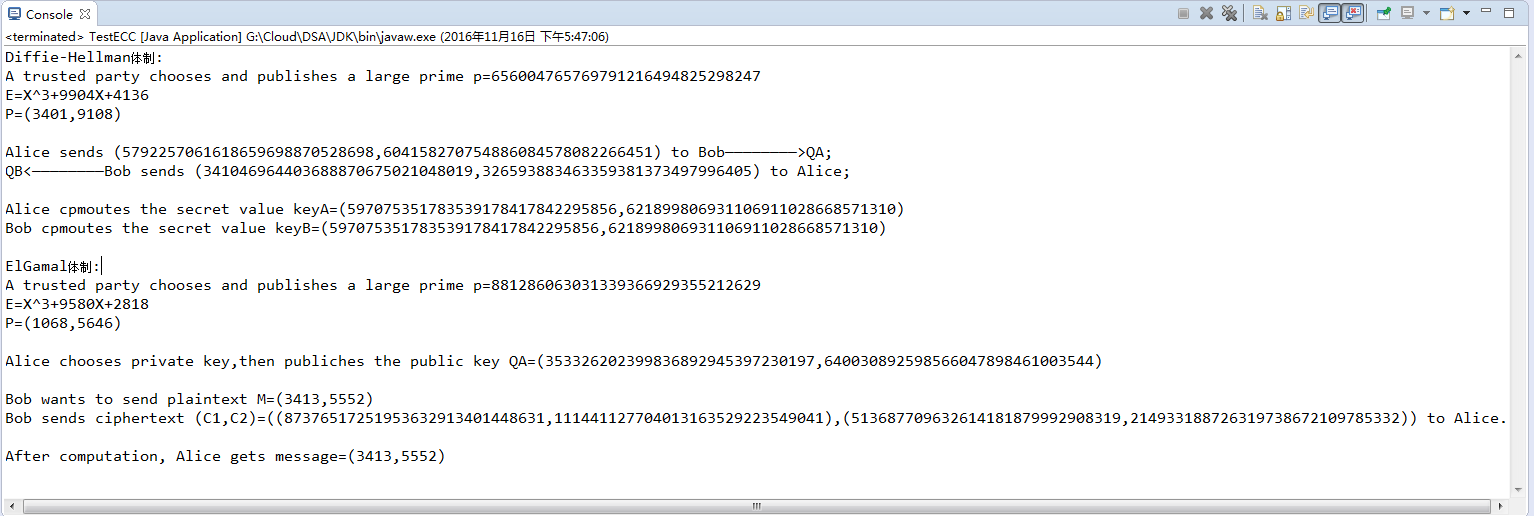
System.***out***.println("ElGamal体制:");

ecc.ElGamal();

}

}

测试结果：



即：

Diffie-Hellman体制:

A trusted party chooses and publishes a large prime p=656004765769791216494825298247

E=X^3+9904X+4136

P=(3401,9108)

Alice sends (5792257061618659698870528698,604158270754886084578082266451) to Bob————————>QA;

QB<————————Bob sends (341046964403688870675021048019,326593883463359381373497996405) to Alice;

Alice cpmoutes the secret value keyA=(597075351783539178417842295856,621899806931106911028668571310)

Bob cpmoutes the secret value keyB=(597075351783539178417842295856,621899806931106911028668571310)

ElGamal体制:

A trusted party chooses and publishes a large prime p=881286063031339366929355212629

E=X^3+9580X+2818

P=(1068,5646)

Alice chooses private key,then publiches the public key QA=(353326202399836892945397230197,640030892598566047898461003544)

Bob wants to send plaintext M=(3413,5552)

Bob sends ciphertext (C1,C2)=((87376517251953632913401448631,111441127704013163529223549041),(513687709632614181879992908319,214933188726319738672109785332)) to Alice.

After computation, Alice gets message=(3413,5552)

**五、实验体会**

**（请认真填写自己的真实体会）**

1. 用ECC,可以使用更小的秘钥获得同水平的安全。小秘钥是重要的，特别是在一个越来越多的密码学方法应用在小功率设备（比如手机）的世界里。虽然同时乘以两个质数比把结果因式分解成它的因子要简单，但当质数开始变得非常长，甚至只是乘法步骤都要花费一段时间在一个低功率设备。虽然你或许能通过增加秘钥长度来延长保持RSA的安全性，但随之而来的是在客户端上更缓慢的密码学性能。ECC似乎提供了一个较好的权衡：使用短并且快的秘钥达到高安全性。
2. ECC中得到两个点相加得到第三个点的运算需要编程实现，我们使用了下列方法来实现BigInteger[] calculateAnsPoint(BigInteger[] P,BigInteger[] Q)；

3.确保椭圆曲线是光滑的。既曲线的所有点都没有两个或两个以上不同的切线；

**六、参考文献**

1. 主讲课教材（数学密码学导论）第五章

**2.（如有其它参考文献，请列出）**