VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Information System Analysis And Audit
Review 2

Acquiring Digital Signature using Elliptic curve Cryptography

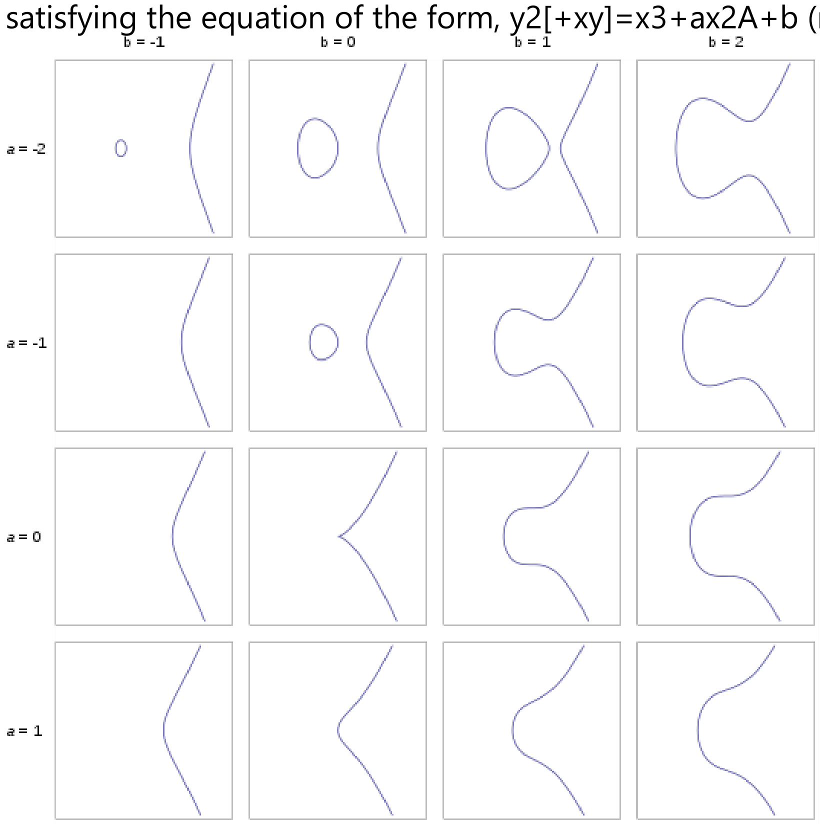
Reg no : **18BIT0168**
Name : Ashu Goyal
Slot : G1
Faculty : Prof. Sumaiya Thaseen I.
Digital Signature :

i) Design and Description of the System

The program calculates and verifies the electronic digital signature based on the Elliptic Curve Cryptography. SHA-1 is used to calculate the hash function.
The elliptic curve cryptosystems are paid more and more attention because its key string is shorter and its security is better than other public cryptosystems. The digital signature system based on elliptic curve (ECDSA) is one of the main stream digital signature systems. Elliptic Curve Digital signature represents one of the most widely used security technologies for ensuring un-forge-ability and non-repudiation of digital data. Its performance heavily depends on an operation called point multiplication. Furthermore, root cause of security breakdown of ECDSA is that it shares three points of the elliptic curve public ally which makes it feasible for an adversary to gauge the private key of the signer.

Elliptic Curve Cryptography

Elliptic curves are Cubic curves. Elliptic curves are called elliptic because of their rapport with elliptic integrals in mathematics which can be used to determine the length of arc of an ellipse. These may be defined as a set of discrete points on the co-ordinate plane, satisfying the equation of the form, y2[+xy]=x3+ax2A+b (mod p).



Eqn: y^{2}=x^{3}+ax+b

Fundamental of Elliptic Curves :

A. Prime Field :

The equation of the elliptic curve on a prime field Fq is V2(modq)=x3+a×x+b where 4a3+27b2(mod q)≠0. Here the elements of finite field are integer between 0 and q−1. All operations such as point-addition, point-subtraction, point-division and point-multiplication involves integer between 0 and q−1. The prime q is chosen such that there is finitely large number of points on the elliptic curve to make the cryptosystem secure.

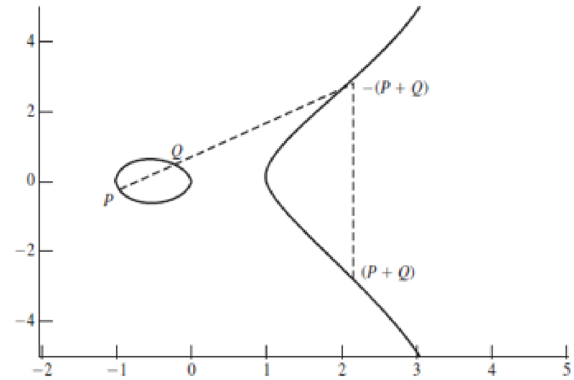
B. Binary Field

The equation of the elliptic curve on a binary field Fm2 is y2+x×y=x3+ax2+b, where b≠0. Here elements of a finite field are integers. These elements are chosen such that length of each should be at most m bits. These numbers can be regarded as a binary polynomial having degree m-1. In binary polynomial the coefficients can only be o or 1. All operations involves polynomial of degree m-1 or lesser. The m is chosen such that there is finitely large number of points on the elliptic curve to make the cryptosystem more secure.

Elliptic Curve Operation :

- **Point Addition**

  It is possible to obtain a third point R on the curve given two points P and Q with the aid of a set of rules. Such a possibility is termed as elliptic curve point addition. The symbol represents the elliptic curve addition P3=P1+P2. Point addition should not to be confused with scalar addition.

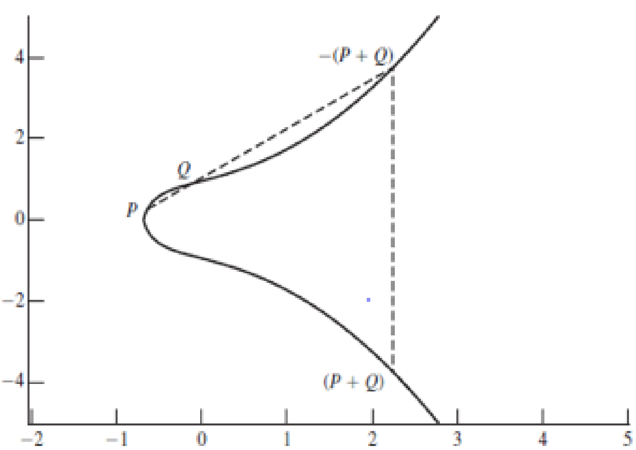


(a) $y^2 = x^3 - x$

- **Point Multiplication**

Consider a point $P(x_p,y_p)$ on elliptic curve E. To determine 2P, P is doubled. This should be an affine point on EC. Equation of the tangent at point P is: $S=[(3x_p^2+a)/2y_p] \pmod p$. Then 2P has affine coordinates (x_r,y_r) given by: $x_r=(S^2-2x_p) \bmod p$ $y_r=[S(x_p-x_r)-y_p] \pmod p$.

Now 3Pcan be determined by point addition of points P and 2P, treating 2P=Q. P has coordinates (x_p,y_p) and Q=2P has coordinates (x_q,y_q). Now the slope is: $S=[(y_q-y_p)/(x_q-x_p)] \bmod p$ $P+Q=-R$ $x_r=(S^2-x_p-x_q) \bmod p$ $y_r=(S(x_p-x_r)-y_p) \bmod p$. Thus $k \times p$ can be calculated by a series of point-doubling and point-addition operation.



(b) $y^2 = x^3 + x + 1$

ii)APPLICATION DEVELOPED

Point.java

```
package com.crypto.entity;

import java.math.BigInteger;

public class Point {

	//coordinates of a point on an elliptic curve over finite fields

	BigInteger pointX;
	BigInteger pointY;

	public BigInteger getPointX() {
		return pointX;
	}
	public void setPointX(BigInteger pointX) {
		this.pointX = pointX;
	}
	public BigInteger getPointY() {
		return pointY;
	}
	public void setPointY(BigInteger pointY) {
		this.pointY = pointY;
	}


}
```

Ecc.java

```
public static void main(String[] args) throws Exception {

		Scanner sc = new Scanner(System.in);

		boolean enableBitcoinParams = true;

		Random rand = new Random();

		//inital elliptic curve configuration (public)

		BigInteger mod;
		BigInteger order;

		if(enableBitcoinParams){

			mod = generatePrimeModulo();
			order = new
BigInteger("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141", 16);
```

```
            }
            else{

                  mod = new BigInteger("199"); // F199
                  order = new BigInteger("211"); //point of the finite field -
order of group

            }

            //curve equation: y^2 = x^3 + ax + b -> current curve: y^2 = x^3 + 7
            BigInteger a = new BigInteger("0");
            BigInteger b = new BigInteger("7");

            //base point on the curve
            Point basePoint = new Point();

            if(enableBitcoinParams){

                  basePoint.setPointX(new
BigInteger("55066263022277343669578718895168534326250603453777594175500187360389116729240"));
                  basePoint.setPointY(new
BigInteger("32670510020758816978083085130507043184471273380659243275938904335757337482424"));

            }
            else{

                  basePoint.setPointX(BigInteger.valueOf(2));
                  basePoint.setPointY(BigInteger.valueOf(24));

            }

            //-----------------------------------------------

            //brute force

            System.out.println("------------------------------");
            System.out.println("brute force addition");
            System.out.println("------------------------------");

            System.out.println("P: "+displayPoint(basePoint));

            Point newPoint = pointAddition(basePoint, basePoint, a, b, mod);
            System.out.println("2P: "+displayPoint(newPoint));
            //int n=sc.nextInt();

            for(int i=3;i<=10;i++) {


                        newPoint = pointAddition(newPoint, basePoint, a, b, mod);

                        System.out.println(i+"P: "+displayPoint(newPoint));

                        // will add order of grp using try and catch condition
```

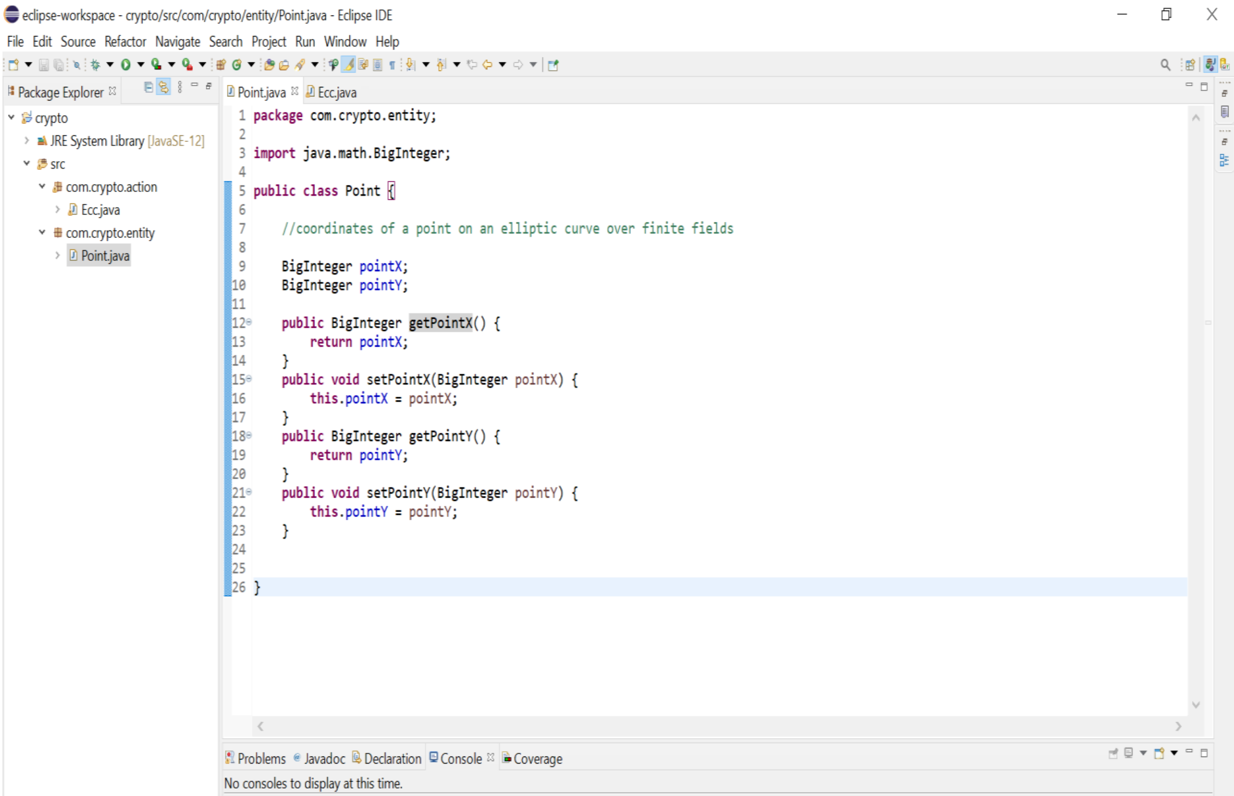
```
		}

		System.out.println();

		//-----------------------------------------------

		//key exchange

		System.out.println("------------------------------------------");
		System.out.println("Elliptic Curve Diffie Hellman Key Exchange");
		System.out.println("------------------------------------------");

		Date generationBegin = new Date();

		System.out.println("public key generation...");

		BigInteger kRam = new BigInteger("2010000000000017"); //Ram's private
key
		Point RamPublic = applyDoubleAndAddMethod(basePoint, kRam, a, b, mod);
		System.out.println("Ram public: \t"+displayPoint(RamPublic));

		BigInteger kShyam = new BigInteger("2010000000000061"); //Shyam's
private key
		Point ShyamPublic = applyDoubleAndAddMethod(basePoint, kShyam, a, b,
mod);
		System.out.println("Shyam public: \t"+displayPoint(ShyamPublic));

		Date generationEnd = new Date();

		System.out.println("public key generation lasts "
				+(double)(generationEnd.getTime() -
generationBegin.getTime())/1000+" seconds\n");

		//------------------------------------------------

		Date exchangeBegin = new Date();

		System.out.println("key exchange...");

		Point RamShared = applyDoubleAndAddMethod(ShyamPublic, kRam, a, b, mod);
		System.out.println("Ram shared: \t"+displayPoint(RamShared));

		Point ShyamShared = applyDoubleAndAddMethod(RamPublic, kShyam, a, b,
mod);
		System.out.println("Shyam shared: \t"+displayPoint(ShyamShared));

		Date exchangeEnd = new Date();

		System.out.println("shared key exchange lasts "
				+(double)(exchangeEnd.getTime() -
exchangeBegin.getTime())/1000+" seconds\n");


		//------------------------------------
```

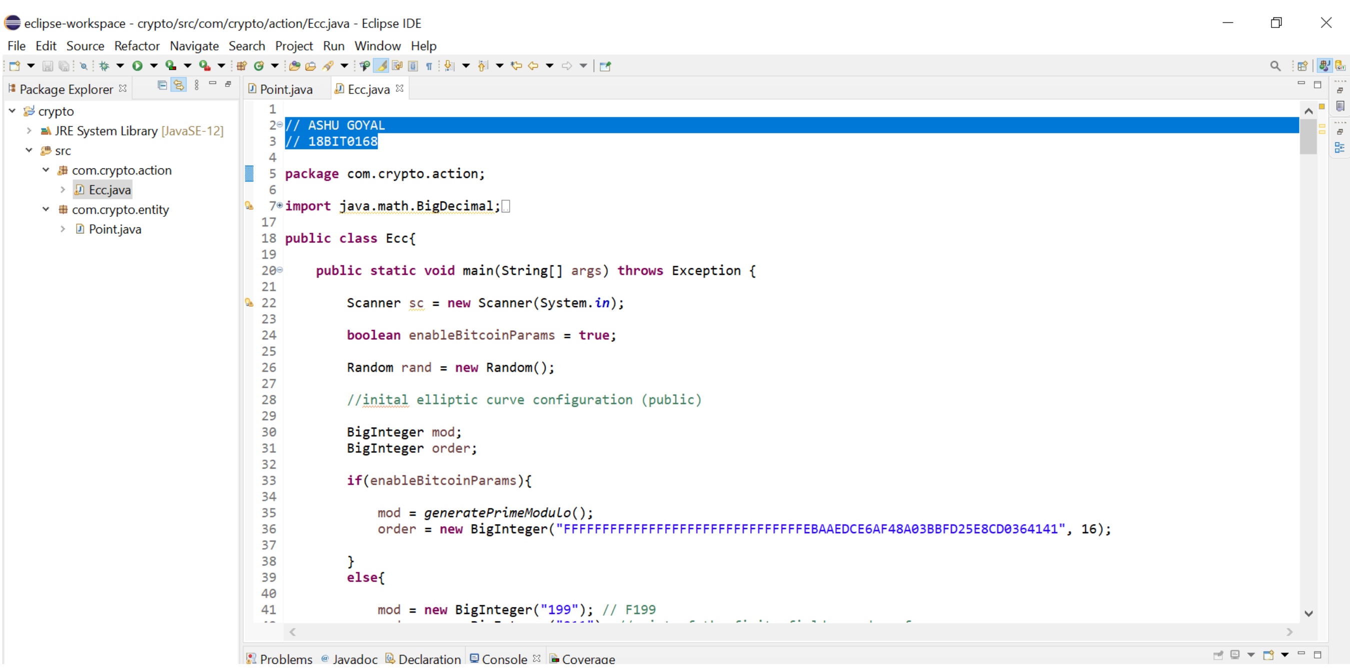
```java
		//ecdsa - elliptic curve digital signature algorithm
		
		System.out.println("---------------------------------------------------
");
		System.out.println("Elliptic Curve Digital Signature Algorithm - 
ECDSA");
		System.out.println("---------------------------------------------------
");
		
		//String text = "Ashu Goyal 18BIT0168";
		System.out.println("Write your message ");
		String text = sc.nextLine();
		
		MessageDigest md = MessageDigest.getInstance("SHA1");
		md.update(text.getBytes());
		byte[] hashByte = md.digest();
		
		BigInteger hash = new BigInteger(hashByte).abs();
		
		System.out.println("message: "+text);
		System.out.println("hash: "+hash);
		
		//------------------------------------
		
		//BigInteger privateKey = new BigInteger("151");
		BigInteger privateKey = new 
BigInteger("75263518707598184987916378021939673586055614731957507592904438851787542395619");
		
		Point publicKey = applyDoubleAndAddMethod(basePoint, privateKey, a, b, 
mod);
		
		System.out.println("public key: "+displayPoint(publicKey));
		
		
		//BigInteger randomKey = new BigInteger("115");
		//BigInteger randomKey = new 
BigInteger("28695618543805844332113829720373285210420739438570883203839696518176414791234");
		BigInteger randomKey = new BigInteger(128, rand);
		
		Point randomPoint = applyDoubleAndAddMethod(basePoint, randomKey, a, b, 
mod);
		
		System.out.println("random point: "+displayPoint(randomPoint));
		
		//------------------------------------
		
		//signing
		
		System.out.println("\nsigning...");
		
		Date signingBegin = new Date();
		
		BigInteger r = randomPoint.getPointX().remainder(order);
```

```
            BigInteger s =
(hash.add(r.multiply(privateKey)).multiply(multiplicativeInverse(randomKey,
order))).remainder(order);

            System.out.println("Signature: (r, s) = ("+r+", "+s+")");

            Date signingEnd = new Date();

            System.out.println("\nmessage signing lasts "
                         +(double)(signingEnd.getTime() -
signingBegin.getTime())/1000+" seconds\n");

            //-----------------------------------

            //verification

            Date verifyBegin = new Date();

            System.out.println("verification...");

            BigInteger r1=sc.nextBigInteger();
            BigInteger s1=sc.nextBigInteger();

            BigInteger w = multiplicativeInverse(s1, order);

            Point u1 = applyDoubleAndAddMethod(basePoint,
(hash.multiply(w).remainder(order)), a, b, mod);

            Point u2 = applyDoubleAndAddMethod(publicKey,
(r1.multiply(w).remainder(order)), a, b, mod);

            Point checkpoint = pointAddition(u1, u2, a, b, mod);

            System.out.println("checkpoint: "+displayPoint(checkpoint));

            System.out.println(checkpoint.getPointX()+" ?= "+r);

            if(checkpoint.getPointX().compareTo(r) == 0){

                  System.out.println("signature is valid...");

            }
            else{

                  System.out.println("invalid signature detected!!!");

            }

            Date verifyEnd = new Date();

            System.out.println("\nverification lasts "
                         +(double)(verifyEnd.getTime() -
verifyBegin.getTime())/1000+" seconds\n");
```

```
		//-----------------------------------

		//symmetric key encryption / decryption

		System.out.println("-----------------------------------");
		System.out.println("Elliptic Curve ElGamal Cryptosystem");
		System.out.println("-----------------------------------");

		Point plaintext = new Point();
		plaintext.setPointX(new
BigInteger("336149967351030618680861315033126277860770498883769660845427857731520433 8
1677"));
		plaintext.setPointY(new
BigInteger("845575943611910316099620620801289312009521636547123441624777695327769511 9
5137"));

		System.out.println("plaintext: "+displayPoint(plaintext));

		BigInteger secretKey = new
BigInteger("752635187075981849879163780219396735860556147319575075929044388517875423 9
5619");
		//Ram and Shyam both must know this secret key

		publicKey = applyDoubleAndAddMethod(basePoint, secretKey, a, b, mod);

		//encryption

		rand = new Random();

		randomKey = new BigInteger(128, rand); //2^128 - 1

		Point c1 = applyDoubleAndAddMethod(basePoint, randomKey, a, b, mod);

		Point c2 = applyDoubleAndAddMethod(publicKey, randomKey, a, b, mod);
		c2 = pointAddition(c2, plaintext, a, b, mod);

		System.out.println("\nciphertext:");
		System.out.println("c1: "+displayPoint(c1));
		System.out.println("c2: "+displayPoint(c2));

	}
```

Screenshot :


eclipse-workspace - crypto/src/com/crypto/entity/Point.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
crypto
JRE System Library [JavaSE-12]
src
com.crypto.action
Ecc.java
com.crypto.entity
Point.java
Point.java
Ecc.java
package com.crypto.entity;
import java.math.BigInteger;
public class Point {
//coordinates of a point on an elliptic curve over finite fields
BigInteger pointX;
BigInteger pointY;
public BigInteger getPointX() {
return pointX;
}
public void setPointX(BigInteger pointX) {
this.pointX = pointX;
}
public BigInteger getPointY() {
return pointY;
}
public void setPointY(BigInteger pointY) {
this.pointY = pointY;
}
}
Problems
Javadoc
Declaration
Console
Coverage
No consoles to display at this time.


eclipse-workspace - crypto/src/com/crypto/action/Ecc.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
crypto
JRE System Library [JavaSE-12]
src
com.crypto.action
Ecc.java
com.crypto.entity
Point.java
Point.java
Ecc.java
// ASHU GOYAL
// 18BIT0168
package com.crypto.action;
import java.math.BigDecimal;
public class Ecc{
public static void main(String[] args) throws Exception {
Scanner sc = new Scanner(System.in);
boolean enableBitcoinParams = true;
Random rand = new Random();
//inital elliptic curve configuration (public)
BigInteger mod;
BigInteger order;
if(enableBitcoinParams){
mod = generatePrimeModulo();
order = new BigInteger("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141", 16);
}
else{
mod = new BigInteger("199"); // F199
Problems
Javadoc
Declaration
Console
Coverage

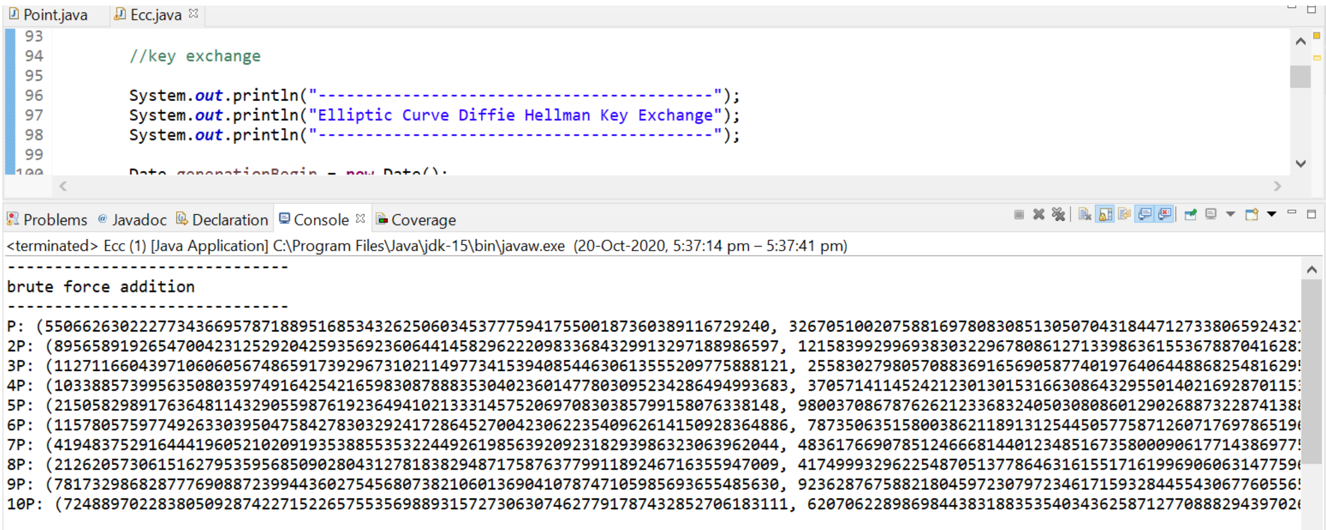
Brute force point addition:

```java
        //-----------------------------------------------

        //brute force

        System.out.println("-----------------------------");
        System.out.println("brute force addition");
        System.out.println("-----------------------------");

        System.out.println("P: "+displayPoint(basePoint));

        Point newPoint = pointAddition(basePoint, basePoint, a, b, mod);
        System.out.println("2P: "+displayPoint(newPoint));
        //int n=sc.nextInt();

        for(int i=3;i<=10;i++) {


                newPoint = pointAddition(newPoint, basePoint, a, b, mod);

                System.out.println(i+"P: "+displayPoint(newPoint));

                // will add order of grp using try and catch condition
        }

        System.out.println();

        //-----------------------------------------------
```

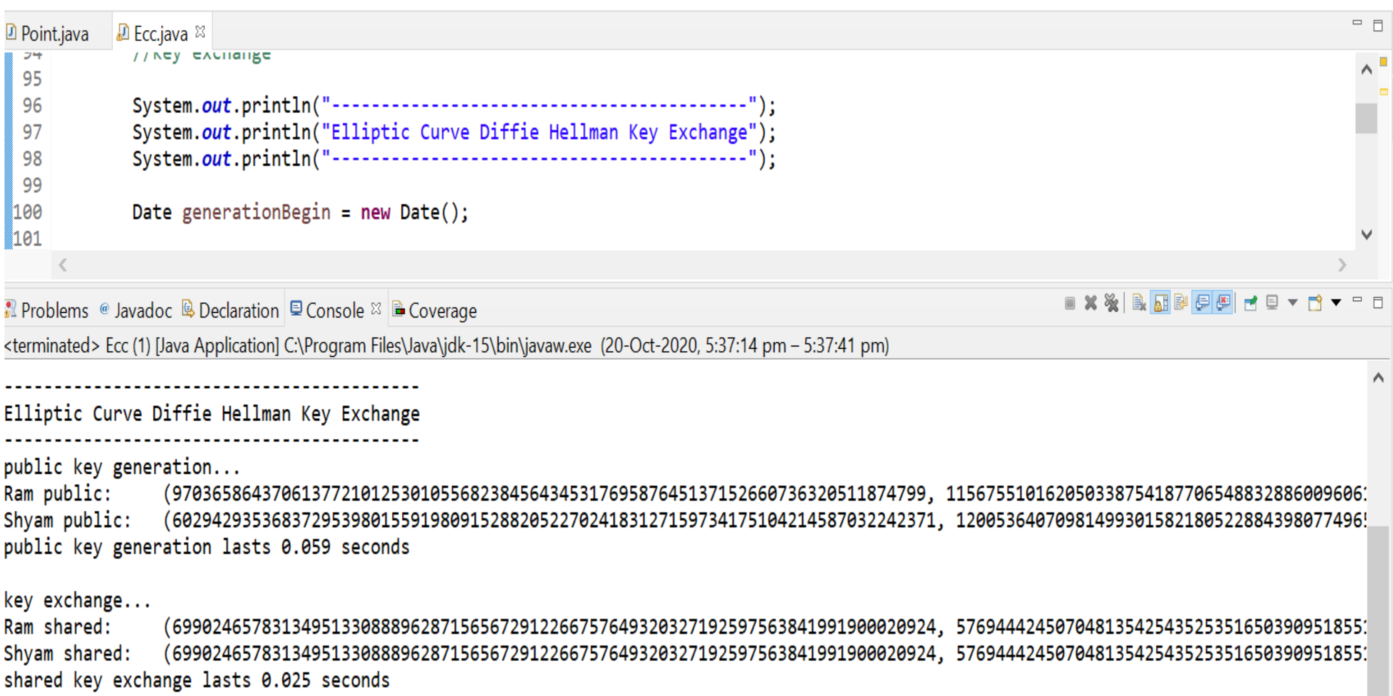
Key Exchange:

```java

        //key exchange

        System.out.println("------------------------------------------");
        System.out.println("Elliptic Curve Diffie Hellman Key Exchange");
        System.out.println("------------------------------------------");

        Date generationBegin = new Date();

        System.out.println("public key generation...");

        BigInteger kRam = new BigInteger("2010000000000017"); //Ram's private key
        Point RamPublic = applyDoubleAndAddMethod(basePoint, kRam, a, b, mod);
        System.out.println("Ram public: \t"+displayPoint(RamPublic));

        BigInteger kShyam = new BigInteger("2010000000000061"); //Shyam's private key
        Point ShyamPublic = applyDoubleAndAddMethod(basePoint, kShyam, a, b, mod);
        System.out.println("Shyam public: \t"+displayPoint(ShyamPublic));

        Date generationEnd = new Date();

        System.out.println("public key generation lasts "
                +(double)(generationEnd.getTime() - generationBegin.getTime())/1000+" seconds\n");

        //-----------------------------------------------

        Date exchangeBegin = new Date();

        System.out.println("key exchange...");

        Point RamShared = applyDoubleAndAddMethod(ShyamPublic, kRam, a, b, mod);
        System.out.println("Ram shared: \t"+displayPoint(RamShared));
```

Output :



Elliptic Curve Diffie Hellman Key Exchange:



Elliptic Curve Digital Signature Algorithm – ECDSA

Point.java Ecc.java

```java
        System.out.println("----------------------------------------------------");
        System.out.println("Elliptic Curve Digital Signature Algorithm - ECDSA");
        System.out.println("----------------------------------------------------");

        //String text = "Ashu Goyal 18BIT0168";
        System.out.println("Write your message ");
        String text = sc.nextLine();
```

Problems Javadoc Declaration Console Coverage

<terminated> Ecc (1) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (20-Oct-2020, 5:37:14 pm – 5:37:41 pm)

```
----------------------------------------------------
Elliptic Curve Digital Signature Algorithm - ECDSA
----------------------------------------------------
Write your message
ashu goyal
message: ashu goyal
hash: 252028629355812873146649262218585244878992613032
public key: (86123958339353589454334613954037009250298301442165544159467110006827437489844, 248861675833951013317041420088293786758817457
random point: (112653625677348478557860612005332412675079056773553768391208247689289590699286, 620918986827275454673247446653469179639938

signing...
Signature: (r, s) = (112653625677348478557860612005332412675079056773553768391208247689289590699286, 1111667251956534577260335356692070149

message signing lasts 0.0 seconds

verification...
112653625677348478557860612005332412675079056773553768391208247689289590699286
111166725195653457726033535669207014962660464938771455008001770234901662814917
checkpoint: (112653625677348478557860612005332412675079056773553768391208247689289590699286, 62091898682727545467324744665346917963993880
112653625677348478557860612005332412675079056773553768391208247689289590699286 ?= 11265362567734847855786061200533241267507905677355376839
signature is valid...

verification lasts 15.387 seconds
```

Point.java Ecc.java

```java
        //symmetric key encryption / decryption

        System.out.println("----------------------------------");
        System.out.println("Elliptic Curve ElGamal Cryptosystem");
        System.out.println("----------------------------------");

        Point plaintext = new Point();
```

Problems Javadoc Declaration Console Coverage

<terminated> Ecc (1) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (20-Oct-2020, 5:37:14 pm – 5:37:41 pm)

```
random point: (112653625677348478557860612005332412675079056773553768391208247689289590699286, 620918986827275454673247446653469179639938

signing...
Signature: (r, s) = (112653625677348478557860612005332412675079056773553768391208247689289590699286, 1111667251956534577260335356692070149

message signing lasts 0.0 seconds

verification...
112653625677348478557860612005332412675079056773553768391208247689289590699286
111166725195653457726033535669207014962660464938771455008001770234901662814917
checkpoint: (112653625677348478557860612005332412675079056773553768391208247689289590699286, 62091898682727545467324744665346917963993880
112653625677348478557860612005332412675079056773553768391208247689289590699286 ?= 11265362567734847855786061200533241267507905677355376839
signature is valid...

verification lasts 15.387 seconds

----------------------------------
Elliptic Curve ElGamal Cryptosystem
----------------------------------
plaintext: (33614996735103061868086131503312627786077049888376966084542785773152043381677, 845575943611910316099620620801289312009521636
5
ciphertext:
c1: (85087345177053822579116300209139284533330707526327569748548591392920645922276, 394978340386288424182879419100499924482368536214273662
c2: (70347052281205961372649190299279887301447584832055866747617225053034577485811, 172431479479855168683355982250517982924740921353157389
```

iii) Key Generation

Hashing Algorithm:

SHA-1

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. SHA-1 is used to verify that a file has been unaltered. This is done by producing a checksum before the file has been transmitted, and then again once it reaches its destination. The transmitted file can be considered genuine only if both checksums are identical. Even a small change in the message will, with overwhelming probability, result in many bits changing due to the avalanche effect.

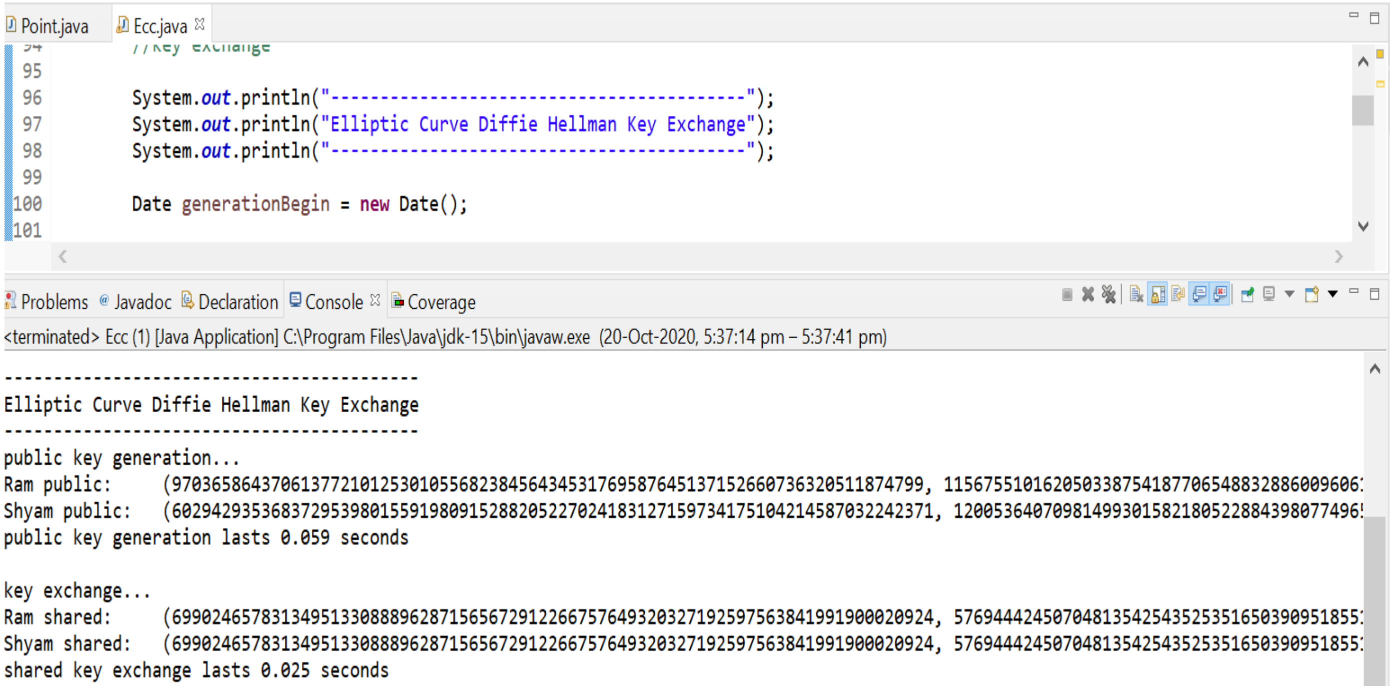
Code :

```
//String text = "Ashu Goyal 18BIT0168";
System.out.println("Write your message ");
String text = sc.nextLine();

MessageDigest md = MessageDigest.getInstance("SHA1");
md.update(text.getBytes());
byte[] hashByte = md.digest();

BigInteger hash = new BigInteger(hashByte).abs();

System.out.println("message: "+text);
              System.out.println("hash: "+hash);
```

<u>**Key Exchange :**</u>

Code

```
Date generationBegin = new Date();

		System.out.println("public key generation...");

		BigInteger kRam = new BigInteger("2010000000000017"); //Ram's private key
		Point RamPublic = applyDoubleAndAddMethod(basePoint, kRam, a, b, mod);
		System.out.println("Ram public: \t"+displayPoint(RamPublic));

		BigInteger kShyam = new BigInteger("2010000000000061"); //Shyam's private key
		Point ShyamPublic = applyDoubleAndAddMethod(basePoint, kShyam, a, b, mod);
		System.out.println("Shyam public: \t"+displayPoint(ShyamPublic));

		Date generationEnd = new Date();

		System.out.println("public key generation lasts "
				+(double)(generationEnd.getTime() - generationBegin.getTime())/1000+" seconds\n");

		//-------------------------------------------------

		Date exchangeBegin = new Date();

		System.out.println("key exchange...");

		Point RamShared = applyDoubleAndAddMethod(ShyamPublic, kRam, a, b, mod);
		System.out.println("Ram shared: \t"+displayPoint(RamShared));

		Point ShyamShared = applyDoubleAndAddMethod(RamPublic, kShyam, a, b, mod);
		System.out.println("Shyam shared: \t"+displayPoint(ShyamShared));

		Date exchangeEnd = new Date();

		System.out.println("shared key exchange lasts "
				+(double)(exchangeEnd.getTime() - exchangeBegin.getTime())/1000+" seconds\n");
```



iv) Encryption :

Code :

```
//symmetric key encryption / decryption

             System.out.println("-----------------------------------");
             System.out.println("Elliptic Curve ElGamal Cryptosystem");
             System.out.println("-----------------------------------");

             Point plaintext = new Point();
             plaintext.setPointX(new
BigInteger("3361499673510306186808613150331262778607704988837696608454278577315204338
1677"));
             plaintext.setPointY(new
BigInteger("8455759436119103160996206208012893120095216365471234416247776953277695119
5137"));

             System.out.println("plaintext: "+displayPoint(plaintext));

             BigInteger secretKey = new
BigInteger("7526351870759818498791637802193967358605561473195750759290443885178754239
5619");
             //Ram and Shyam both must know this secret key

             publicKey = applyDoubleAndAddMethod(basePoint, secretKey, a, b, mod);

             //encryption
```

```
rand = new Random();

randomKey = new BigInteger(128, rand); //2^128 - 1

Point c1 = applyDoubleAndAddMethod(basePoint, randomKey, a, b, mod);

Point c2 = applyDoubleAndAddMethod(publicKey, randomKey, a, b, mod);
c2 = pointAddition(c2, plaintext, a, b, mod);

System.out.println("\nciphertext:");
System.out.println("c1: "+displayPoint(c1));
System.out.println("c2: "+displayPoint(c2));
```

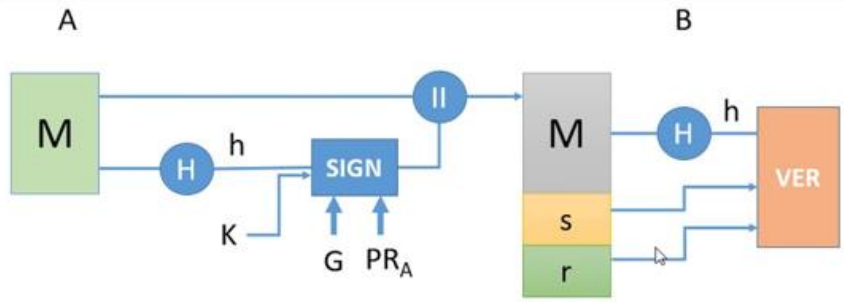
Elliptic Curve Digital Signature Algorithm

The steps involved in ECDSA are formation of key-pair, signature-generation and signature-verification. The digital signature is typically created using the hash function. The transmitter sends the encrypted data along with signature to the receiver. The receiver in possession of sender's public key and domain parameters can authenticate the signature.

The prime q of the finite field Fq, the equation of the elliptic curve E, the point P on the curve and its order n, are the public domain parameters. Furthermore, a randomly selected integer d from the interval [1, n-1] forms a private key. Multiplying P by the private key d, which is called scalar multiplication, will generate the corresponding public key Q.

The pair (Q, d) forms the ECC public-private key pair with Q is the public key and d is the private key. The generating point G, the curve parameter ‘a’ and ‘b’, together with few more constants constitute the domain parameters of ECC.

The public key is a point on the curve and the private key is a random number selected by signer. The public key is obtained by multiplying the private key with the generating point on the curve.



Signature = {s, r}

K → Random number

G → Global element