



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 :

Ashu

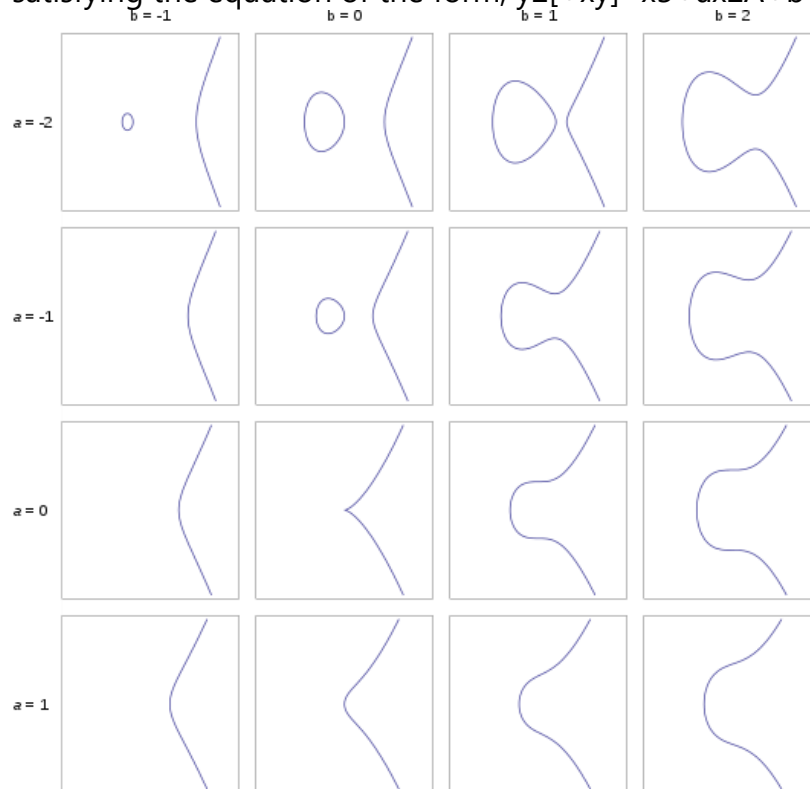
i) Design and Description of the System

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, $y^2 + xy = x^3 + ax^2 + b \pmod{p}$.



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

Fundamental of Elliptic Curves :

A. Prime Field :

The equation of the elliptic curve on a prime field F_q is $V^2(\text{mod } q) = x^3 + ax + b$ where $4a^3 + 27b^2(\text{mod } q) \neq 0$. 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.

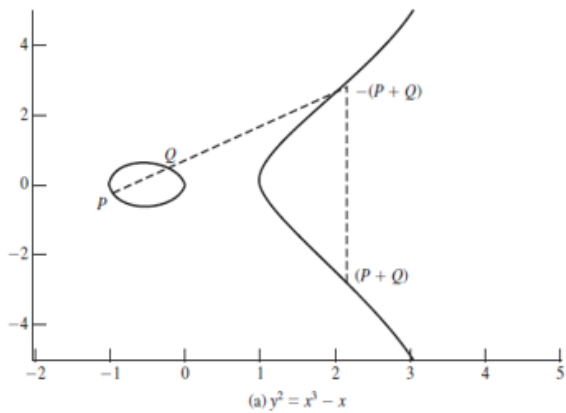
B. Binary Field

The equation of the elliptic curve on a binary field F_{m^2} is $y^2 + x \cdot y = x^3 + ax^2 + b$, where $b \neq 0$. 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 0 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 $P_3 = P_1 + P_2$. Point addition should not to be confused with scalar addition.



• Point Multiplication

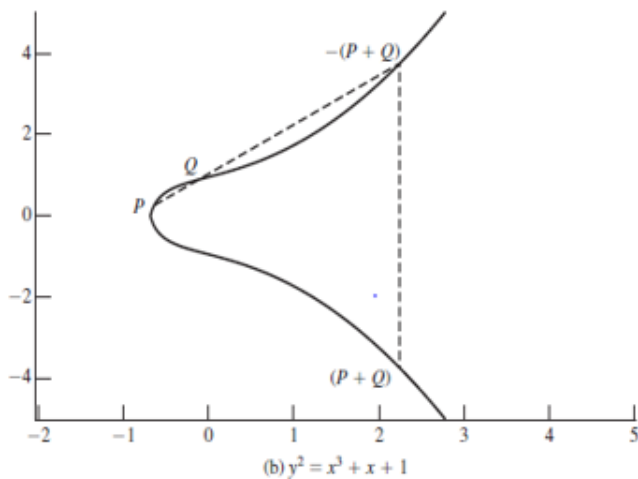
Consider a point $P(x_p, y_p)$ on elliptic curve E . To determine $2P$, P is doubled. This should be an affine point on EC . Equation of the tangent at point P is:

$S = [(3x_p^2 + a)/2y_p] \pmod{p}$. Then $2P$ has affine coordinates (x_r, y_r) given by:

$x_r = (S^2 - 2x_p) \pmod{p}$ $y_r = [S(x_p - x_r) - y_p] \pmod{p}$.

Now $3P$ can 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:

$S = [(y_q - y_p)/(x_q - x_p)] \pmod{p}$ $P + Q = -R$ $x_r = (S^2 - x_p - x_q) \pmod{p}$ $y_r = (S(x_p - x_r) - y_p) \pmod{p}$. Thus $k \times p$ can be calculated by a series of point-doubling and point-addition operation.



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();

    //initia elliptic curve configuration (public)

    BigInteger mod;
    BigInteger order;

    if(enableBitcoinParams){

        mod = generatePrimeModulo();
        order = new
        BigInteger("FFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141", 16);
```

```

    }
    else{

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

    }

    //curve equation:  $y^2 = x^3 + ax + b \rightarrow$  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("5506626302227734366957871889516853432625060345377759417550018736038911672
9240"));
        basePoint.setPointY(new
BigInteger("3267051002075881697808308513050704318447127338065924327593890433575733748
2424"));

    }
    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("201000000000017"); //Ram's private
key
    Point RamPublic = applyDoubleAndAddMethod(basePoint, kRam, a, b, mod);
    System.out.println("Ram public: \t"+displayPoint(RamPublic));

    BigInteger kShyam = new BigInteger("201000000000061"); //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");

    //-----

```

```

//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("7526351870759818498791637802193967358605561473195750759290443885178754239
5619");

Point publicKey = applyDoubleAndAddMethod(basePoint, privateKey, a, b,
mod);

System.out.println("public key: "+displayPoint(publicKey));

//BigInteger randomKey = new BigInteger("115");
//BigInteger randomKey = new
BigInteger("2869561854380584433211382972037328521042073943857088320383969651817641479
1234");
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("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);

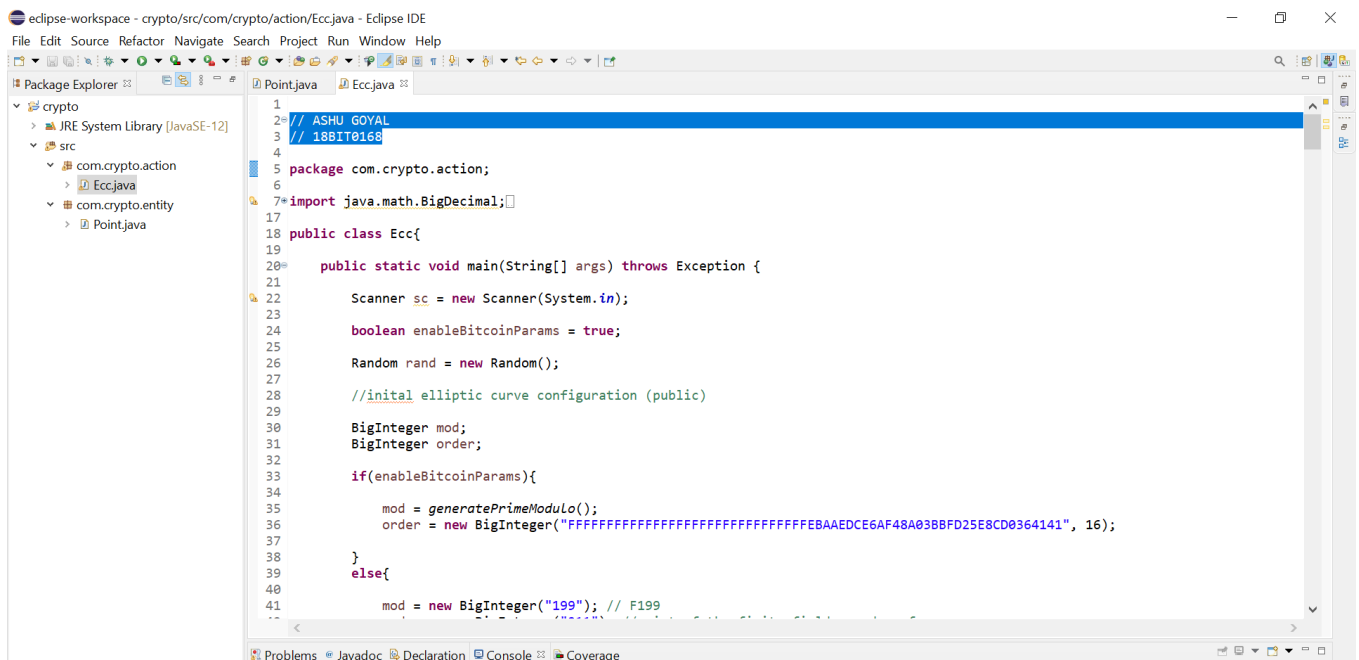
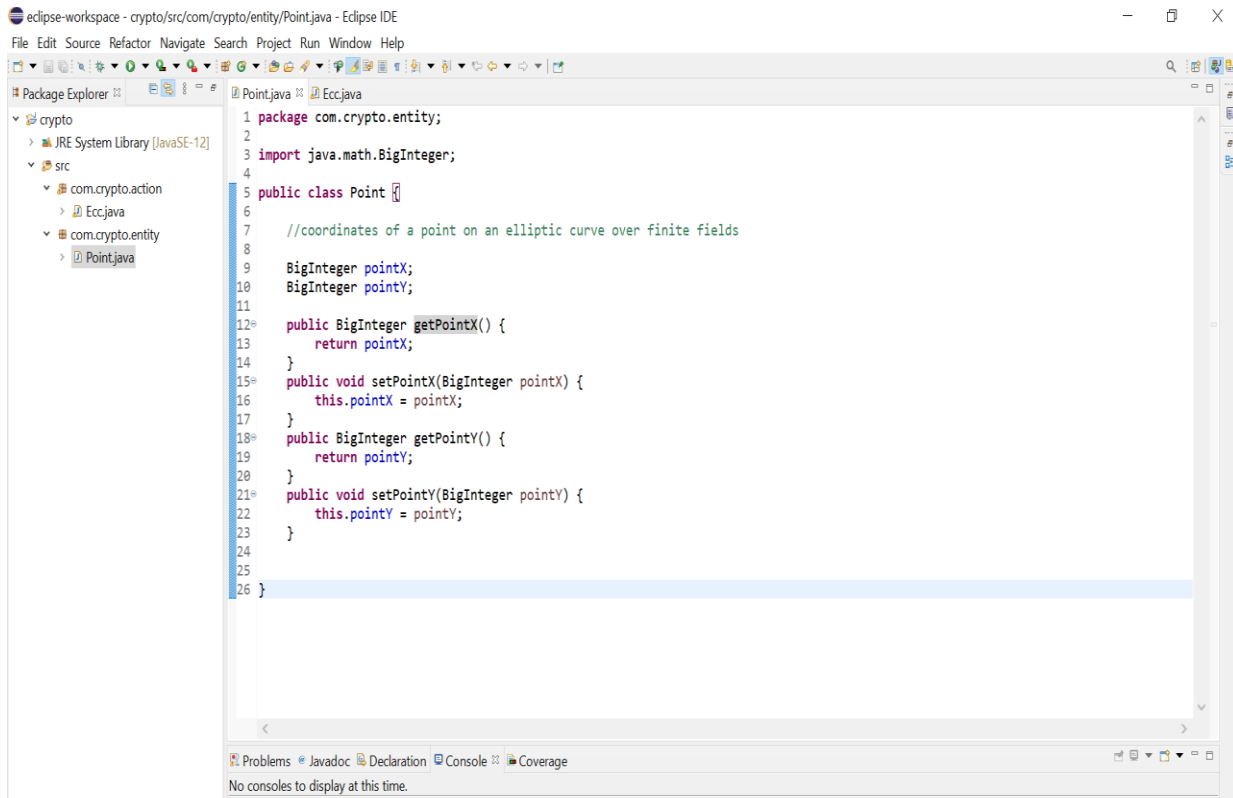
Point c2 = applyDoubleAndAddMethod(publicKey, randomKey, a, b, mod);
c2 = pointAddition(c2, plaintext, a, b, mod);

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

}

```

Screenshot :



Brute force point addition:

eclipse-workspace - crypto/src/com/crypto/action/Ecc.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

```
Package Explorer  Point.java  Ecc.java
└─ crypto
   └─ JRE System Library [JavaSE-12]
      └─ src
         └─ com.crypto.action
            └─ Ecc.java
               └─ com.crypto.entity
                  └─ Point.java

66 //-----
67
68 //brute force
69
70 System.out.println("-----");
71 System.out.println("brute force addition");
72 System.out.println("-----");
73
74 System.out.println("P: "+displayPoint(basePoint));
75
76 Point newPoint = pointAddition(basePoint, basePoint, a, b, mod);
77 System.out.println("2P: "+displayPoint(newPoint));
78 //int n=sc.nextInt();
79
80 for(int i=3;i<=10;i++) {
81
82
83     newPoint = pointAddition(newPoint, basePoint, a, b, mod);
84
85     System.out.println(i+"P: "+displayPoint(newPoint));
86
87     // will add order of grp using try and catch condition
88 }
89
90 System.out.println();
91
92 //-----
```

Key Exchange:

```
Point.java  Ecc.java
93
94 //key exchange
95
96 System.out.println("-----");
97 System.out.println("Elliptic Curve Diffie Hellman Key Exchange");
98 System.out.println("-----");
99
100 Date generationBegin = new Date();
101
102 System.out.println("public key generation...");
103
104 BigInteger kRam = new BigInteger("2010000000000017"); //Ram's private key
105 Point RamPublic = applyDoubleAndAddMethod(basePoint, kRam, a, b, mod);
106 System.out.println("Ram public: \t"+displayPoint(RamPublic));
107
108 BigInteger kShyam = new BigInteger("2010000000000061"); //Shyam's private key
109 Point ShyamPublic = applyDoubleAndAddMethod(basePoint, kShyam, a, b, mod);
110 System.out.println("Shyam public: \t"+displayPoint(ShyamPublic));
111
112 Date generationEnd = new Date();
113
114 System.out.println("public key generation lasts "
115     +(double)(generationEnd.getTime() - generationBegin.getTime())/1000+" seconds\n");
116
117 //-----
118
119 Date exchangeBegin = new Date();
120
121 System.out.println("key exchange...");
122
123 Point RamShared = applyDoubleAndAddMethod(ShyamPublic, kRam, a, b, mod);
124 System.out.println("Ram shared: \t"+displayPoint(RamShared));
125
```

Problems Javadoc Declaration Console Coverage

No console to display at this time

Output :

```
Point.java Ecc.java
93
94 //key exchange
95
96 System.out.println("-----");
97 System.out.println("Elliptic Curve Diffie Hellman Key Exchange");
98 System.out.println("-----");
99
100 Date generationBegin = new Date();
101
```

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)

brute force addition

P: (55066263022277343669578718895168534326250603453777594175500187360389116729240, 326705100207588169780830851305070431844712733806592432;
2P: (89565891926547004231252920425935692360644145829622209833684329913297188986597, 12158399299693830322967808612713398636155367887041628;
3P: (112711660439710606056748659173929673102114977341539408544630613555209775888121, 2558302798057088369165690587740197640644886825481629;
4P: (103388573995635080359749164254216598308788835304023601477803095234286494993683, 3705714114524212301301531663086432955014021692870115;
5P: (21505829891763648114329055987619236494102133314575206970830385799158076338148, 98003708678762621233683240503080860129026887322874138;
6P: (115780575977492633039504758427830329241728645270042306223540962614150928364886, 7873506351580038621189131254450577587126071769786519;
7P: (41948375291644419605210209193538855353224492619856392092318293986323063962044, 48361766907851246668144012348516735800090617714386977;
8P: (21262057306151627953595685090280431278183829487175876377991189246716355947009, 41749993296225487051377864631615517161996906063147759;
9P: (7817329868287769088723994436027545680738210601369041078747105985693655485630, 92362876758821804597230797234617159328445543067760556;
10P: (72488970228380509287422715226575535698893157273063074627791787432852706183111, 6207062289869844383188353540343625871277088829439702;

Elliptic Curve Diffie Hellman Key Exchange:

```
Point.java Ecc.java
94 //key exchange
95
96 System.out.println("-----");
97 System.out.println("Elliptic Curve Diffie Hellman Key Exchange");
98 System.out.println("-----");
99
100 Date generationBegin = new Date();
101
```

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 Diffie Hellman Key Exchange

public key generation...

Ram public: (9703658643706137721012530105568238456434531769587645137152660736320511874799, 115675510162050338754187706548832886009606;
Shyam public: (60294293536837295398015591980915288205227024183127159734175104214587032242371, 12005364070981499301582180522884398077496;
public key generation lasts 0.059 seconds

key exchange...

Ram shared: (69902465783134951330888962871565672912266757649320327192597563841991900020924, 57694442450704813542543525351650390951855;
Shyam shared: (69902465783134951330888962871565672912266757649320327192597563841991900020924, 57694442450704813542543525351650390951855;
shared key exchange lasts 0.025 seconds

Elliptic Curve Digital Signature Algorithm – ECDSA

```
Point.java  Ecc.java
138
139      System.out.println("-----");
140      System.out.println("Elliptic Curve Digital Signature Algorithm - ECDSA");
141      System.out.println("-----");
142
143      //String text = "Ashu Goyal 18BIT0168";
144      System.out.println("Write your message ");
145      String text = sc.nextLine();

-----
Write your message
ashu goyal
message: ashu goyal
hash: 252028629355812873146649262218585244878992613032
public key: (86123958339353589454334613954037009250298301442165544159467110006827437489844, 248861675833951013317041420088293786758817457)
random point: (112653625677348478557860612005332412675079056773553768391208247689289590699286, 62091898682727545467324744665346917963993880)

signing...
Signature: (r, s) = (112653625677348478557860612005332412675079056773553768391208247689289590699286, 111166725195653457726033535669207014)

message signing lasts 0.0 seconds

verification...
112653625677348478557860612005332412675079056773553768391208247689289590699286
111166725195653457726033535669207014962660464938771455008001770234901662814917
checkpoint: (112653625677348478557860612005332412675079056773553768391208247689289590699286, 62091898682727545467324744665346917963993880)
112653625677348478557860612005332412675079056773553768391208247689289590699286 != 112653625677348478557860612005332412675079056773553768391208247689289590699286
signature is valid...

verification lasts 15.387 seconds
```

```
Point.java  Ecc.java
234      //symmetric key encryption / decryption
235
236      System.out.println("-----");
237      System.out.println("Elliptic Curve ElGamal Cryptosystem");
238      System.out.println("-----");
239
240      Point plaintext = new Point();

-----
signing...
Signature: (r, s) = (112653625677348478557860612005332412675079056773553768391208247689289590699286, 111166725195653457726033535669207014)

message signing lasts 0.0 seconds

verification...
112653625677348478557860612005332412675079056773553768391208247689289590699286
111166725195653457726033535669207014962660464938771455008001770234901662814917
checkpoint: (112653625677348478557860612005332412675079056773553768391208247689289590699286, 62091898682727545467324744665346917963993880)
112653625677348478557860612005332412675079056773553768391208247689289590699286 != 112653625677348478557860612005332412675079056773553768391208247689289590699286
signature is valid...

verification lasts 15.387 seconds

-----
Elliptic Curve ElGamal Cryptosystem
plaintext: (33614996735103061868086131503312627786077049888376966084542785773152043381677, 8455759436119103160996206208012893120095216365)

ciphertext:
c1: (8508734517705382257911630020913928453330707526327569748548591392920645922276, 39497834038628842418287941910049992448236853621427366)
c2: (70347052281205961372649190299279887301447584832055866747617225053034577485811, 17243147947985516868335598225051798292474092135315738)
```

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);
```

Key Exchange :

Code

```
Date generationBegin = new Date();

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

    BigInteger kRam = new BigInteger("201000000000017"); //Ram's private
key
    Point RamPublic = applyDoubleAndAddMethod(basePoint, kRam, a, b, mod);
    System.out.println("Ram public: \t"+displayPoint(RamPublic));

    BigInteger kShyam = new BigInteger("201000000000061"); //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");
```


The screenshot shows an IDE with two tabs: 'Point.java' and 'Ecc.java'. The 'Ecc.java' tab is active, displaying the following code:

```

95
96     System.out.println("-----");
97     System.out.println("Elliptic Curve Diffie Hellman Key Exchange");
98     System.out.println("-----");
99
100     Date generationBegin = new Date();
101

```

Below the code editor, the 'Console' tab is selected, showing the output of the program:

```

<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 Diffie Hellman Key Exchange
-----
public key generation...
Ram public:    (9703658643706137721012530105568238456434531769587645137152660736320511874799, 115675510162050338754187706548832886009606:
Shyam public:  (60294293536837295398015591980915288205227024183127159734175104214587032242371, 12005364070981499301582180522884398077496:
public key generation lasts 0.059 seconds

key exchange...
Ram shared:    (69902465783134951330888962871565672912266757649320327192597563841991900020924, 57694442450704813542543525351650390951855:
Shyam shared:  (69902465783134951330888962871565672912266757649320327192597563841991900020924, 57694442450704813542543525351650390951855:
shared key exchange lasts 0.025 seconds

```

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));

```

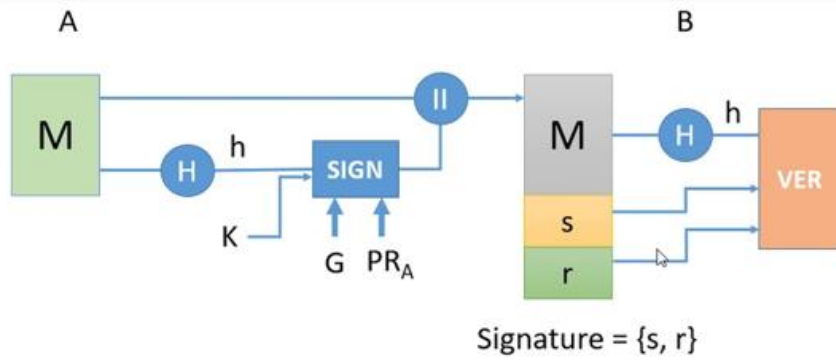
Elliptic Curve Digital Signature Algorithm

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 F_q , 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.



$K \rightarrow$ Random number
 $G \rightarrow$ Global element

Abhy: