LOCATION BASED ELLIPTIC CURVE CRYPTOGRAPHY

FINAL YEAR PROJECT

SUBMITTED BY

RUPIN KEJRIWAL
001311001028

AND

ABU KAUNEN NAWAZ
001311001030

UNDER THE SUPERVISION OF
Mr. UTPAL KUMAR RAY

DEPARTMENT OF INFORMATION TECHNOLOGY
JADAVPUR UNIVERSITY
2017

ACKNOWLEDGEMENT

We would like to thank our project guide Prof. Utpal Kumar Ray, Assistant Professor of Jadavpur University along with Mr. Aritro Sengupta, PhD Scholar for their guidance and support and the faculty members of the department for their cooperation. We extend our sincere thanks to our classmates who have helped us in our hours of need.

ABSTRACT

We are living in the information age. We need to keep information about every aspect of our lives. In other words, information is an asset that has a value like any other asset. As an asset, information needs to be secured from attacks.
These informations are highly confidential, which should be accessed only by few authorized individuals. For example, information related to country's security is accessed only by few noted important officials. Many data in the form of encrypted messages are received by these officials via the network. These officials first decrypt the messages and then can read the information.

Such security can be processed to a much higher level if we devise some technology which would let these officials decrypt the messages only at the desired location which the sender wants and between a time frame so that no other individual at some other place could get his hands on the information.

Hence, this project report presents such an implementation in java based on location based encryption which enables the decryption of the message only at the desired location in the given time frame only.

TABLE OF CONTENTS

INTRODUCTION

Location based encryption enhances security by integrating position and time into encryption and decryption processes. We find that from a security perspective, it is not enough to simply enable or disable decryption based on location and time; these aspects must be integrated into the key construction process. Furthermore, keys or files in transit should not reveal anything regarding their locations/times of applicability.

The term “location-based encryption” is used here to refer to any method of encryption wherein the cipher text can only be decrypted at a specified location. If an attempt is made to decrypt the data at another location, the decryption process fails and reveals no information about the plaintext.

Location-based encryption can be used to ensure that data cannot be decrypted outside a particular facility, for example, the headquarters of a government agency or corporation, or an individual’s office or home. Alternatively, it may be used to confine access to a broad geographic region. Time as well as space constraints may be placed on the decryption location.

For implementing location based encryption, we used the concept of Elliptic Curve Cryptography.

CRYPTOGRAPHY & NETWORK SECURITY

Cryptography derived its name from a Greek word called “krypto’s” which means “Hidden Secrets”. Cryptography is the practice and study of hiding information. It is the Art or Science of converting a plain intelligible data into an unintelligible data and again retransforming that message into its original form. It provides Confidentiality, Integrity, and Accuracy.

Cryptography provides three security goals (refer Fig.2.1):

- **Confidentiality**: This means that the information must be protected from unauthorised access.
- **Integrity:** This means that changes need to be done only by authorised entities and through authorised mechanisms.
- **Availability:** This means that the information created & stored by an organisation needs to be available to authorised entities. Information is useless if it is not available.

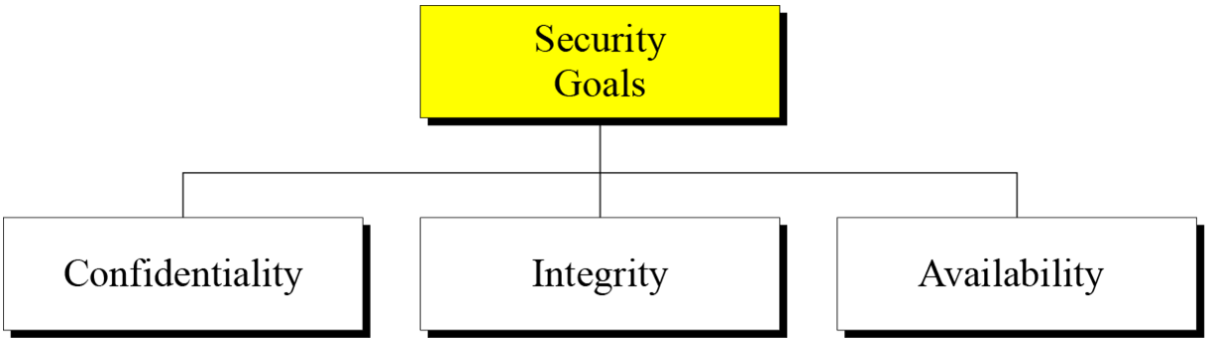

Fig. 2.1 Security Goals of cryptography.

Mechanisms in cryptography: Nowadays, cryptography is defined as involving three distinct mechanisms:

1. Symmetric-Key Encipherment
2. Asymmetric-Key Encipherment
3. Hashing

Symmetric-Key Encipherment: Also called *secret key cryptography*.

1. Alice can send a message to Bob over an insecure channel with an assumption that Eve cannot understand the contents of the message by simply *eavesdropping* over the channel.
2. Alice encrypts the message using an encryption algorithm and Bob decrypts the message using a decryption algorithm.
3. A single secret key is used for both encryption and decryption.

Fig.2.2 shows the encryption & decryption in secret key cryptography.

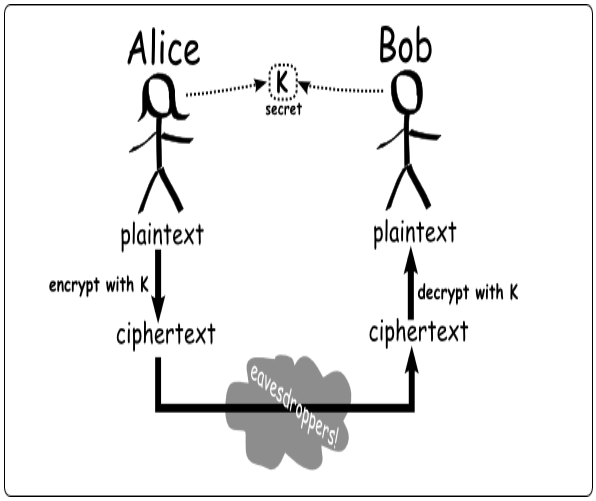

Fig.2.2 Symmetric Key Cryptography.

Asymmetric Key Encipherment: Also called *public key cryptography*. We have the same situation like the previous one, but with few exceptions. Instead of one key, we have two keys; one public key and one private key. To send a secured message to Bob, Alice first encrypts the message using Bob's public key. To decrypt the message, Bob uses his own private key. Fig.2.3 shows the encryption & decryption in public key cryptography.

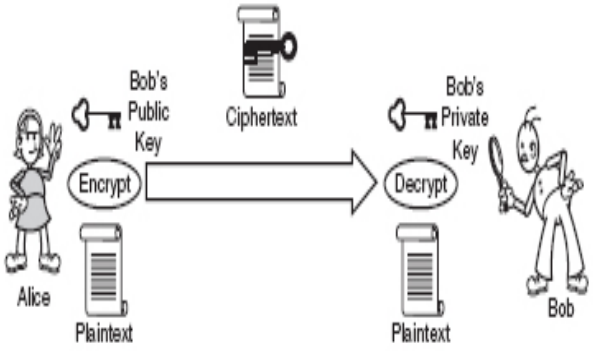

Fig.2.3 Public Key Cryptography

Hashing: In hashing, a fixed length message digest is created out of a variable length message. The digest is normally much smaller than the message. Both the message and the digest must be sent to Bob. Hash output will change substantially even if a single bit of the input is changed. Fig.2.4 shows the conversion of the variable length of data in fixed digest using hashing.

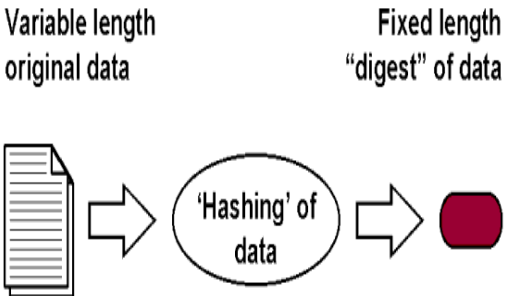

Fig.2.4 Hashing

ELLIPTIC CURVE CRYPTOGRAPHY

ECC is a secure *public key cryptosystem* that provides same level of security like the other public key systems but with smaller key sizes. ECC system is based on the theory of *elliptic curves*.

Elliptic curves are cubic equations in two variables that are similar to the equations used to calculate the length of a curve in the circumference of an ellipse. The general equation of the elliptic curve is:

$$y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$$

Elliptic curves over real numbers use a special class of elliptic curves of the form:

$$y^2 = x^3 + ax + b$$

So, for every message to be sent, the message must be mapped to a point on the curve. All the operations which will be used will be point operations on the elliptic curve like point addition or point multiplication. Fig.3.1 and Fig.3.2 shows the computation process of Point addition & Point multiplication.

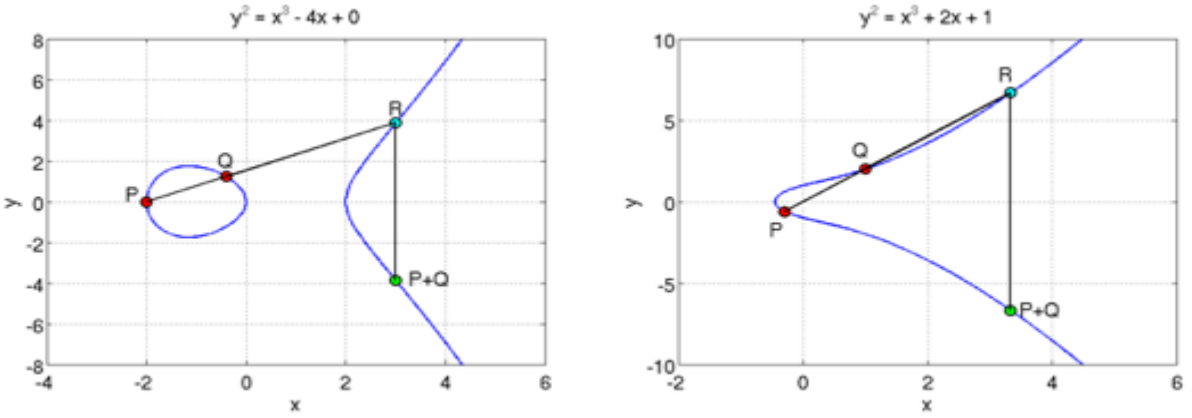

Fig.3.1 Point Addition

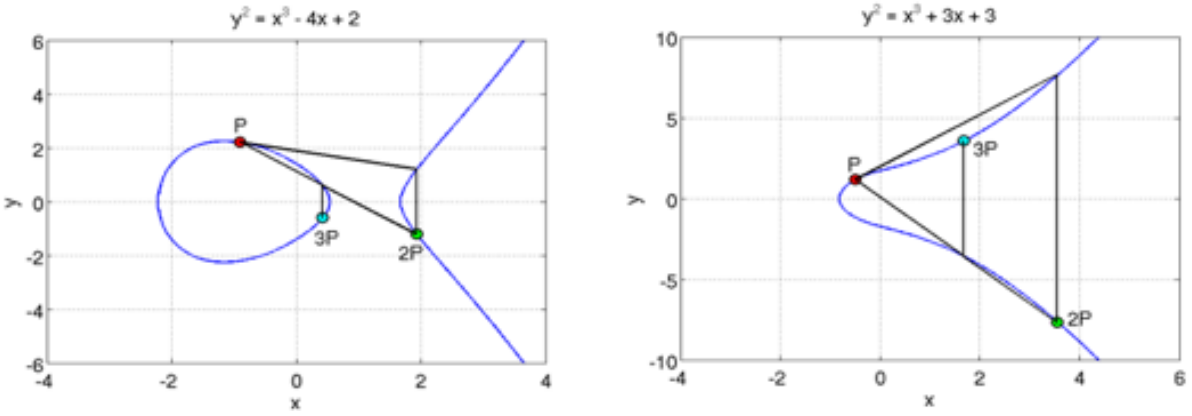

Fig.3.2 Point Multiplication

The Discrete Logarithm Problem For Elliptic Curves (ECDLP): Public-key cryptography is based on the intractability of certain mathematical problems. Early public-key systems are secure assuming that it is difficult to factor a large integer composed of two or more large prime factors. For elliptic-curve-based protocols, it is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is infeasible: this is the "elliptic curve discrete logarithm problem" (ECDLP). The security of elliptic curve cryptography depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original and product points. The size of the elliptic curve determines the difficulty of the problem.

Given an elliptic curve *E* and two points *A* and *B* on *E,* the discrete log problem for elliptic curves is finding an integer $1 \leq d \leq \#E$ such that

$P + P + \ldots + P = dP = T$

In cryptosystems d is the private key and T is the public key.

And, #E is Hasse's Bound which is

$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$ given a elliptic curve E modulo p.

So, the steps in **Elliptic Curve Cryptosystem** are as follows:

1. First an elliptic curve E (a,b) is generated defining its several parameters required.
2. A point on the curve is chosen $e_1(x_1, y1)$ (generator point) and an integer d is chosen. Then a point $e_2(x_2,y_2) = d * e_1(x_1,y_1)$ is calculated. This $e_2(x_2, y2)$ serves as the public key for the user and the integer d serves as the private key. This is done on the receiver side and then these are sent to the sender.
3. Then, the sender maps the message to a point P on the curve which is done by a **Message mapping algorithm**.
4. Now, two ciphers texts C1 and C2 are generated by the sender such that:

   $C_1 = r * e_1$

   $C_2 = P + r * e_2$

   where r is a random integer

   This ends the encryption part.
5. The sender sends these two cipher texts to the receiver.
6. Bob, after receiving C_1 and C_2, calculates P, the plaintext using the following formula:

   $P = C_2 - (d * C_1)$

   The "–" here means adding with the inverse.
7. Hence, the message is decrypted.

LOCATION BASED ECC

The term “location-based encryption” is used here to refer to any method of encryption wherein the cipher text can only be decrypted at a specified location. If an attempt is made to decrypt the data at another location, the decryption process fails and reveals no information about the plaintext.

Location based Cryptography is based on the theory that geographical positions can be used to derive the credentials which determine the key features of the identities used by the involved parties which are sharing the information. This form of cryptography employs a method of not only integrating the location of the participating nodes into encryption and decryption processes but also incorporates these locations into the process of construction of keys that are required to cipher and decipher the plain text data. We have tried to demonstrate that a particular location provided by the receiver node can be used as a key and incorporated into the encryption process to manipulate the plain text and convert it into a cipher text.

Traditional encryption assures that users who have certain kind of authorization can access secure content. Location based service integrates a mobile device's location related information such as latitude, longitude to provide extra security to user. Location-based encryption refers to a method of encryption in which the cipher text can only be decrypted at a specified locality like headquarters of a government agency or corporation, or an individual’s working place. If an attempt is made to decrypt the cipher text at some other place, the decryption process fails and reveals no information about the plain text. Also time, bio-statistics, space can be posted as additional constraints on the decryption location.

To implement location based ECC it is required that we form a location key which is another point on the elliptic curve. This location key is formed by the concatenation of location base point and the time key and then multiplying this concatenated value to the point e_1 on the curve.

We consider only the base point of the location coordinates into account. Then we form the location BP.
If the latitude is 22.789 and longitude is 88.126, then the co-ordinates are concatenated to form Location BP= 2278988126.

The map is divided into different grids of equal dimensions as shown in Fig.4.1. Each grid has a base point marked with red dot. BP is located at the left bottom corner of each grid as shown in Fig.4.2.

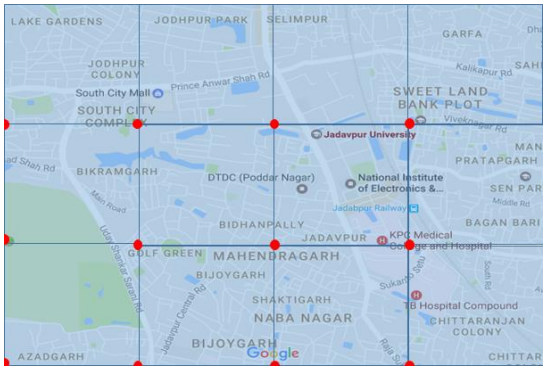

Fig.4.1 Division of map into grids

Fig.4.2 Base point of a grid

Unlike location of the receiver, Time Frame varies with time.
For example, JU campus is open from 10:00:00 to 17:30:00 every day. So if the sender sends data on 04/05/2017 to the receiver, then the receiver will be able to decrypt that message if

a. The receiver is physically located inside the campus
b. The receiver receives the message before 18:59:59 on that particular date.

For each date, 10:00:00 is considered as the B.P of time frame. In this case the Time Key used by the sender is the concatenation date and BP of that particular frame = 04052017100000

Location Key = f(Time Key, Location BP)
If we consider f() as concatenation then Location Key=
04052017100000 || 2278988126
L_k= 04052017100000022789881 26

Encryption at Sender's end using ECC
$C_1 = k. e_1$
$C_2 = P_m + k. e_2 + L_k. e_1$
Send C_1 and C_2

Decryption at Receiver's end using ECC
Receiver has private key d
Receiver gets Location Key L_k from location and time frame.
Receiver calculates
$C_2 – d.C_1 – L_k. e_1 = P_m$

IMPLEMENTATION

POINT ADDITION:

```
public ECPoint add(ECPoint q) throws NoCommonMotherException{

       if (!hasCommonMother(q)) throw new NoCommonMotherException();

       if (this.iszero) return q;
       else if (q.isZero()) return this;

       BigInteger y1 = y;
       BigInteger y2 = q.gety();
       BigInteger x1 = x;
       BigInteger x2 = q.getx();

       BigInteger alpha;

       if (x2.compareTo(x1) == 0) {

           if (!(y2.compareTo(y1) == 0)) return new ECPoint(mother);
           else {
              alpha =
((x1.modPow(TWO,mother.getp())).multiply(THREE)).add(mother.geta());
              alpha =
(alpha.multiply((TWO.multiply(y1)).modInverse(mother.getp()))).mod(mother.getp());
           }

       } else {
           alpha =
((y2.subtract(y1)).multiply((x2.subtract(x1)).modInverse(mother.getp()))).mod(mothe
r.getp());
       }

       BigInteger x3,y3;
       x3 =
(((alpha.modPow(TWO,mother.getp())).subtract(x2)).subtract(x1)).mod(mother.getp());
       y3 = ((alpha.multiply(x1.subtract(x3))).subtract(y1)).mod(mother.getp());


       try{ return new ECPoint(mother,x3,y3); }
       catch (NotOnMotherException e){
           System.out.println("Error in add!!! Result not on mother!");
           return null;
       }

    }
```

POINT MULTIPLICATION:

```
public ECPoint multiply(BigInteger coef) {
       try{
           ECPoint result = new ECPoint(mother);
           byte[] coefb = coef.toByteArray();
           if(fastcache != null) {
```

```
            for(int i = 0; i < coefb.length; i++) {
                result = result.times256().add(fastcache[coefb[i]&255]);
            }
            return result;
        }
        if(cache == null) {
            cache = new ECPoint[16];
            cache[0] = new ECPoint(mother);
            for(int i = 1; i < cache.length; i++) {
                cache[i] = cache[i-1].add(this);
            }
        }
        for(int i = 0; i < coefb.length; i++) {
            result =
result.times16().add(cache[(coefb[i]>>4)&15]).times16().add(cache[coefb[i]&15]);
        }
        return result;
    } catch (NoCommonMotherException e) {
        System.out.println("Error in pow!!!");
        return null;
    }
}
```

ENCRYPTION:

```
public byte[] encrypt(byte[] input,int numbytes, Key key,String lKey) {
    ECKey ek = (ECKey) key;
    System.out.println("Encrypting...");
    //String lKey = "191220160859592278988126";
  BigInteger lk = new BigInteger(lKey);
  byte[] res=new byte[ek.mother.getPCS()+numbytes];
    hash.reset();


    BigInteger rk = new BigInteger(ek.mother.getp().bitLength() + 17, Rand.om);
    if (ek.mother.getOrder() != null) {
        rk = rk.mod(ek.mother.getOrder());
    }
    ECPoint gamma = ek.mother.getGenerator().multiply(rk);      //value of C1
    //System.out.println("Printing value of C1.....->");
    //System.out.println(gamma.toString());

    ECPoint sec = ek.beta.multiply(rk);       //value of r.PbKey
    System.out.println("Printing value of r.Pkey.....->");
    System.out.println(sec.toString());

    ECPoint lkey = ek.mother.getGenerator().multiply(lk); //value of Lkey.G
    System.out.println("Printing value of location Key.....->");
    System.out.println(lkey.toString());

    System.arraycopy(gamma.compress(),0,res,0,ek.mother.getPCS());



    /** copying gamma to C1 **/
    try{
           C1 = new ECPoint(gamma.getMother(),gamma.getx(),gamma.gety());
    }catch(NotOnMotherException ex){
```

```
            System.err.println("Point Doesn't lie on the curve!!");
      }


      /** Mapping input message to a point on Elliptic Curve **/

      String binary = new BigInteger(input).toString(2);
      System.out.println("Input Text As binary: "+binary);        //converting the
input text to binary representation

      StringBuilder sb = new StringBuilder(binary);
      sb.append("00000000");
      //appending 8 bits at end of the binary representation of input text
      String s=sb.toString();
      System.out.println("Input Text after appending 8 bits: "+ s);
      BigInteger x_cord = new BigInteger(s,2);             //converting the binary
to corresponding decimal integer value
      System.out.println("Decimal equivalent: "+ x_cord);
      ECPoint Pm=null;


      int flag = 0;  // value 0 indicates not a Quadratic residue and value 1
indicates is a Quadratic residue
      flag =
findY_Coordinate(x_cord,ek.mother.geta(),ek.mother.getb(),ek.mother.getp());
      while(flag == 0)
      {
            x_cord = x_cord.add(BigInteger.ONE);
            flag =
findY_Coordinate(x_cord,ek.mother.geta(),ek.mother.getb(),ek.mother.getp());
      }
      try{
            Pm = new ECPoint(ek.mother,x_cord,y_cord);
            System.out.println("Printing coordinates of Pm------>
"+Pm.toString());

            try{
                  C2 = Pm.add(sec.add(lkey));
                  System.out.println("After Ecrypting---->");
                  System.out.println("Printing coordinates of C1------
>"+C1.toString());
                  System.out.println("Printing Coordinates of C2 ----->"+
C2.toString());
            }
            catch(NoCommonMotherException ex){}
      }
      catch(NotOnMotherException ex){
            System.out.println("Point Doesnt not lie on the curve!!");
      }

      System.out.println("");
      return res;
   }
```

DECRYPTION:

```
public byte[] decrypt(byte[] input, Key key,String lKey_decrypt) {
       ECKey dk = (ECKey) key;
       System.out.println("Decrypting...");
       //String lKey = "191220160859592278988126";
    BigInteger lk = new BigInteger(lKey_decrypt);
       byte[] res=new byte[input.length-dk.mother.getPCS()];
       byte[] gammacom=new byte[dk.mother.getPCS()];
       hash.reset();
       byte[] arr=null;

       System.arraycopy(input,0,gammacom,0,dk.mother.getPCS());
       ECPoint gamma = new ECPoint(gammacom,dk.mother);
       ECPoint sec = gamma.multiply(dk.sk);
       ECPoint lkey = dk.mother.getGenerator().multiply(lk);
       System.out.println("Location Key: "+ lkey.toString());


       ECPoint skey = C1.multiply(dk.sk);
       try {
              ECPoint sum = skey.add(lkey);
              System.out.println("Coordinates of (nB.C1 + Lk.G ) before inverting---
-->");
              System.out.println(sum.toString());
              BigInteger y_cord = sum.gety();
              //y_cord = BigInteger.ZERO.subtract(y_cord);
              y_cord = sum.getMother().getp().subtract(y_cord);
              sum.sety(y_cord);
              System.out.println("Coordinates of (nB.C1 + Lk.G ) after inverting----
->");
              System.out.println(sum.toString());
              ECPoint plainText = C2.add(sum);
              System.out.println("Plaintext Coordinates After decryption 'Pm'-----
>");
              System.out.println(plainText.toString());
              BigInteger x_cord = plainText.getx();

              System.out.println("ReMapping point coordinates to string message---
>");
              String t = x_cord.toString(2);
              System.out.println("PlainText with 8 bits padding As binary: "+t);
              t= t.substring(0,t.length()-8);
              System.out.println("PlainText As binary: "+t);

              BigInteger bi = new BigInteger(t,2);
              System.out.println("Decimal Equivalent: "+bi);
              String output= new String(bi.toByteArray());
              System.out.println("After Decrypting---->");
              System.out.println("PlainText As text: "+output);
               arr = output.getBytes();
       } catch (NoCommonMotherException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
       }
       return arr;
    }
```

CONCLUSION

The focus of the project was to implement location based cryptography using elliptic curve cryptography and message mapping techniques. Though it has been implemented using java in eclipse IDE, the codes can be used in implementing it into developing an android chat application.

In this project, we have implemented the location based elliptic curve cryptography using java but the location coordinates and the time frames have been fed manually. Adding location key makes the cryptosystem more secure, thereby making cryptanalysis more difficult.

APPENDIX A: ECC interfaces

ECParameters.java

```
package ECC;

/** Specifications completely defining an elliptic curve. Used to define an
 *elliptic curve by EllipticCurve.define(ECParamters ecp).
 *NOTE: This is designed for an elliptic curve on the form:
 *      y^2 = x^3 + ax + b (mod p)
 *--with fixed generator and precomputed order.
*/

import java.math.BigInteger;

public interface ECParameters {

    public BigInteger a();

    public BigInteger b();

    public BigInteger p();

    /** returns the x value of the generator*/
    public BigInteger generatorX();

    /** returns the y value of the generator*/
    public BigInteger generatorY();

    public BigInteger order();

    public String toString();
}
```

Key.java

```
package ECC;
import java.io.*;
/** */
public interface Key {
    public Key readKey(InputStream in) throws IOException;
    public void writeKey(OutputStream out) throws IOException;
    public Key getPublic();
    public boolean isPublic();
}
```

CryptoSystem.java

```
package ECC;

import java.io.*;

/** This interface is used to model a modern cryptosystem. It contains methods
to encrypt and decrypt and methods to generate keys for the specific cryptosystem.
In an actual implementation it would be a good idea to initialize a key inside
the constructor method.*/
public interface CryptoSystem{

    /**  Encrypts the string p.
     *@param plain the plaintext to be encrypted.
     *@param ek The (public) key to use for encryption.
     *@return the input string encrypted with the current key.
     */
    public byte[] encrypt(byte[] plain, int numbytes, Key ek,String lKey);

    /**  Decrypts the string c.
     *@param cipher the ciphertext to be decrypted.
     *@param sk the (secret) key to use for decryption.
     *@return the input string decrypted with the current key.
     */
    public byte[] decrypt(byte[] cipher, Key dk,String lKey_decrypt);

    /** This method generates a new key for the cryptosystem.
     *@return the new key generated*/
    public Key generateKey();

    /** This method returns the maximum size of blocks it can encrypt.
     *@return the maximum block size the system can encrypt. */
    public int blockSize();

    /** Returns a String describing this CryptoSystem*/
    public String toString();
}
```

APPENDIX B: Elliptic Curves

Secp112r1.java

```
package ECC;
import java.math.BigInteger;
public class secp112r1 implements ECParameters {
    private static final BigInteger p
       = new BigInteger("DB7C"+"2ABF62E3"+"5E668076"+"BEAD208B", 16);

    private static final BigInteger a
       = new BigInteger("DB7C"+"2ABF62E3"+"5E668076"+"BEAD2088", 16);

    private static final BigInteger b
       = new BigInteger("659E"+"F8BA0439"+"16EEDE89"+"11702B22", 16);

    private static final BigInteger S
       = new BigInteger("00F50B02" + "8E4D696E" + "67687561" + "51752904"
                     + "72783FB1", 16);

    private static final BigInteger gx
       = new BigInteger("09487239"+"995A5EE7" + "6B55F9C2" + "F098", 16);

    private static final BigInteger gy
       = new BigInteger("A89C" + "E5AF8724" + "C0A23E0E" + "0FF77500", 16);

    private static final BigInteger n
       = new BigInteger("DB7C" + "2ABF62E3" + "5E7628DF" + "AC6561C5", 16);

    public BigInteger a() {
       return a;
    }

    public BigInteger b() {
       return b;
    }

    public BigInteger p() {
       return p;
    }

    public BigInteger generatorX() {
       return gx;
    }

    public BigInteger generatorY() {
       return gy;
    }

    public BigInteger order() {
       return n;
    }

    public String toString(){
       return "secp112r1";
    }

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

```
        System.out.println("a:  "+a);
        System.out.println("b:  "+b);
        System.out.println("p:  "+p);
        System.out.println("gx: "+gx);
        System.out.println("gy: "+gy);
        System.out.println("n:  "+n);
        System.out.println("p.toByteArray().length: "+p.toByteArray().length);
    }


}
```

Secp160r1.java

```
package ECC;
import java.math.BigInteger;
public class secp112r1 implements ECParameters {
    private static final BigInteger p
       = new BigInteger("DB7C"+"2ABF62E3"+"5E668076"+"BEAD208B", 16);

    private static final BigInteger a
       = new BigInteger("DB7C"+"2ABF62E3"+"5E668076"+"BEAD2088", 16);

    private static final BigInteger b
       = new BigInteger("659E"+"F8BA0439"+"16EEDE89"+"11702B22", 16);

    private static final BigInteger S
       = new BigInteger("00F50B02" + "8E4D696E" + "67687561" + "51752904"
                     + "72783FB1", 16);

    private static final BigInteger gx
       = new BigInteger("09487239"+"995A5EE7" + "6B55F9C2" + "F098", 16);

    private static final BigInteger gy
       = new BigInteger("A89C" + "E5AF8724" + "C0A23E0E" + "0FF77500", 16);

    private static final BigInteger n
       = new BigInteger("DB7C" + "2ABF62E3" + "5E7628DF" + "AC6561C5", 16);

    public BigInteger a() {
       return a;
    }

    public BigInteger b() {
       return b;
    }

    public BigInteger p() {
       return p;
    }

    public BigInteger generatorX() {
       return gx;
    }

    public BigInteger generatorY() {
       return gy;
    }
```

```
    public BigInteger order() {
       return n;
    }

    public String toString(){
       return "secp112r1";
    }

    public static void main(String[] args) {
       System.out.println("a:  "+a);
       System.out.println("b:  "+b);
       System.out.println("p:  "+p);
       System.out.println("gx: "+gx);
       System.out.println("gy: "+gy);
       System.out.println("n:  "+n);
       System.out.println("p.toByteArray().length: "+p.toByteArray().length);
    }


}
```

Secp256r1.java

```
package ECC;
import java.math.BigInteger;
public class secp112r1 implements ECParameters {
    private static final BigInteger p
       = new BigInteger("DB7C"+"2ABF62E3"+"5E668076"+"BEAD208B", 16);

    private static final BigInteger a
       = new BigInteger("DB7C"+"2ABF62E3"+"5E668076"+"BEAD2088", 16);

    private static final BigInteger b
       = new BigInteger("659E"+"F8BA0439"+"16EEDE89"+"11702B22", 16);

    private static final BigInteger S
       = new BigInteger("00F50B02" + "8E4D696E" + "67687561" + "51752904"
                     + "72783FB1", 16);

    private static final BigInteger gx
       = new BigInteger("09487239"+"995A5EE7" + "6B55F9C2" + "F098", 16);

    private static final BigInteger gy
       = new BigInteger("A89C" + "E5AF8724" + "C0A23E0E" + "0FF77500", 16);

    private static final BigInteger n
       = new BigInteger("DB7C" + "2ABF62E3" + "5E7628DF" + "AC6561C5", 16);

    public BigInteger a() {
       return a;
    }

    public BigInteger b() {
       return b;
    }

    public BigInteger p() {
       return p;
    }
```

```
    public BigInteger generatorX() {
       return gx;
    }

    public BigInteger generatorY() {
       return gy;
    }

    public BigInteger order() {
       return n;
    }

    public String toString(){
       return "secp112r1";
    }

    public static void main(String[] args) {
       System.out.println("a:  "+a);
       System.out.println("b:  "+b);
       System.out.println("p:  "+p);
       System.out.println("gx: "+gx);
       System.out.println("gy: "+gy);
       System.out.println("n:  "+n);
       System.out.println("p.toByteArray().length: "+p.toByteArray().length);
    }


}
```

APPENDIX C: Elliptic curve points & operations

EllipticCurve.java

```
package ECC;

/**An implementation of an elliptic curve over a -finite- field.
 *
*/
import java.io.*;
import java.math.BigInteger;

public class EllipticCurve {

    private BigInteger a, b, p, order;
    private ECPoint generator;
    private BigInteger ppodbf;
    private int pointcmpsize;
    private String name;

    public static final BigInteger COEFA = new BigInteger("4");
    public static final BigInteger COEFB = new BigInteger("27");
    public static final int PRIMESECURITY = 500;

    /** Constructs an elliptic curve over the finite field of 'mod' elements.
     *The equation of the curve is on the form : y^2 = x^3 + ax + b.
     *@param a the value of 'a' where y^2 = x^3 + ax + b
     *@param b the value of 'b' where y^2 = x^3 + ax + b
     *@param mod The number of elements in the field.
     IMPORTANT: Must a prime number!
    *@exception InsecureCurveException if the curve defined by a and b are
singular,
    *supersingular, trace one/anomalous.
    *This ensures well defined operations and security.*/
    public EllipticCurve(BigInteger a, BigInteger b, BigInteger p) throws
InsecureCurveException {

       this.a = a;
       this.b = b;
       this.p = p;
       if (!p.isProbablePrime(PRIMESECURITY)) {
           //System.out.println("THIS CANNOT HAPPEN!!! "+p+" is not a prime!");
           //throw new
InsecureCurveException(InsecureCurveException.NONPRIMEMODULUS,this);
       }
       if (isSingular()) throw new
InsecureCurveException(InsecureCurveException.SINGULAR, this);

       byte[] pb = p.toByteArray();
       if(pb[0] == 0) pointcmpsize = pb.length;
       else pointcmpsize = pb.length + 1;
       //ppodbf = (p.add(BigInteger.ONE)).shiftRight(2);
       name = "";


       //FIXME compute the order of the group
       //FIXME compute a generator for the group
    }
```

```
    public EllipticCurve(ECParameters ecp) throws InsecureCurveException {
       this(ecp.a(),ecp.b(),ecp.p());
       order = ecp.order();
       name = ecp.toString();
       try{
           generator = new ECPoint(this, ecp.generatorX(), ecp.generatorY());
           generator.fastCache();
       }
       catch (NotOnMotherException e){
           System.out.println("Error defining EllipticCurve: generator not on
mother!");
       }
    }

    public void writeCurve(DataOutputStream output) throws IOException {
       byte[] ab = a.toByteArray();
       output.writeInt(ab.length);
       output.write(ab);
       byte[] bb = b.toByteArray();
       output.writeInt(bb.length);
       output.write(bb);
       byte[] pb = p.toByteArray();
       output.writeInt(pb.length);
       output.write(pb);
       byte[] ob = order.toByteArray();
       output.writeInt(ob.length);
       output.write(ob);
       byte[] gb = generator.compress();
       output.writeInt(gb.length);
       output.write(gb);
       byte[] ppb = getPPODBF().toByteArray();
       output.writeInt(ppb.length);
       output.write(ppb);
       output.writeInt(pointcmpsize);
       output.writeUTF(name);
    }

    protected EllipticCurve(DataInputStream input) throws IOException {
       byte[] ab = new byte[input.readInt()];
       input.read(ab);
       a = new BigInteger(ab);
       byte[] bb = new byte[input.readInt()];
       input.read(bb);
       b = new BigInteger(bb);
       byte[] pb = new byte[input.readInt()];
       input.read(pb);
       p = new BigInteger(pb);
       byte[] ob = new byte[input.readInt()];
       input.read(ob);
       order = new BigInteger(ob);
       byte[] gb = new byte[input.readInt()];
       input.read(gb);
       generator = new ECPoint(gb, this);
       byte[] ppb = new byte[input.readInt()];
       input.read(ppb);
       ppodbf = new BigInteger(ppb);
       pointcmpsize = input.readInt();
       name = input.readUTF();
```

```
   generator.fastCache();
}

public boolean isSingular(){

   BigInteger aa = a.pow(3);
   BigInteger bb = b.pow(2);

   BigInteger result = ((aa.multiply(COEFA)).add(bb.multiply(COEFB) ) ).mod(p);

   if ( result.compareTo(BigInteger.ZERO) == 0 ) return true;
   else return false;

}

//FIXME!!!!!!!!!!
public BigInteger calculateOrder(){
   return null;
}

//FIXME!!!!!!!!
public ECPoint calculateGenerator(){
   return null;
}

public boolean onCurve(ECPoint q){

   if (q.isZero()) return true;
   BigInteger y_square = (q.gety()).modPow(new BigInteger("2"),p);
   BigInteger x_cube = (q.getx()).modPow(new BigInteger("3"),p);
   BigInteger x = q.getx();

   BigInteger dum = ((x_cube.add(a.multiply(x))).add(b)).mod(p);

   if (y_square.compareTo(dum) == 0) return true;
   else return false;

}

/** Returns the order of the group */
public BigInteger getOrder(){
   return order;
}

public ECPoint getZero(){
   return new ECPoint(this);
}

public BigInteger geta(){
   return a;
}

public BigInteger getb(){
   return b;
}

public BigInteger getp(){
   return p;
}
```

```
    public int getPCS() {
       return pointcmpsize;
    }

    /** Returns a generator for this EllipticCurve.*/
    public ECPoint getGenerator(){
       return generator;
    }

    public String toString(){
       if (name == null) return "y^2 = x^3 + "  + a + "x + " + b + " ( mod " + p +
" )";
       else if (name.equals("")) return "y^2 = x^3 + "  + a + "x + " + b + " ( mod
" + p + " )";
       else return name;
    }

    public BigInteger getPPODBF(){
       if(ppodbf == null) {
           ppodbf = p.add(BigInteger.ONE).shiftRight(2);
       }
       return ppodbf;
    }
}
```

ECPoint.java

```
package ECC;

import java.math.BigInteger;
import java.util.Arrays;
import java.io.*;

public class ECPoint {

    public final static BigInteger TWO = new BigInteger("2");
    public final static BigInteger THREE = new BigInteger("3");

    private EllipticCurve mother;

    private BigInteger x, y;
    private boolean iszero;

    private ECPoint[] fastcache = null;
    private ECPoint[] cache = null;

    public void fastCache() {
       try {
           if(fastcache == null) {
              fastcache = new ECPoint[256];
              fastcache[0] = new ECPoint(mother);
              for(int i = 1; i < fastcache.length; i++) {
                  fastcache[i] = fastcache[i-1].add(this);
              }
           }
       } catch (NoCommonMotherException e) {
           System.out.println("ECPoint.fastcache: THIS CANNOT HAPPEN!!!");
```

```
       }
    }

    /** Constructs a point on an elliptic curve.
     *@param mother The elliptic curve on wich the point is surposed to lie
     *@param x the x coordinate of the point
     *@param y the y coordinate of the point
     *@exception Throws a NotOnMotherException if (x,y) is not on the mother
curve.*/
    public ECPoint(EllipticCurve mother, BigInteger x, BigInteger y) throws
NotOnMotherException{
       this.mother = mother;
       this.x = x;
       this.y = y;
       if (!mother.onCurve(this)) throw new NotOnMotherException(this);
       iszero = false;
    }

    /** Decompresses a compressed point stored in a byte-array into a new ECPoint.
     *@param bytes the array of bytes to be decompressed
     *@param mother the EllipticCurve the decompressed point is supposed to lie
on.*/
    public ECPoint(byte[] bytes, EllipticCurve mother) {
       this.mother = mother;
       if (bytes[0] == 2){
           iszero = true;
           return;
       }
       boolean ymt = false;
       if(bytes[0] != 0) ymt = true;
       bytes[0] = 0;
       x = new BigInteger(bytes);
       if(mother.getPPODBF() == null) System.out.println("Error!!!");
       y =
x.multiply(x).add(mother.geta()).multiply(x).add(mother.getb()).modPow(mother.getPP
ODBF(),mother.getp());
       if(ymt != y.testBit(0)) {
           y = mother.getp().subtract(y);
       }
       iszero = false;
    }

    /** IMPORTANT this renders the values of x and y to be null! Use this
constructor
     *only to create instances of a Zero class!*/
    public ECPoint(EllipticCurve e){
       x = y = BigInteger.ZERO;
       mother = e;
       iszero = true;
    }

    public byte[] compress() {
       byte[] cmp = new byte[mother.getPCS()];
       if (iszero){
           cmp[0] = 2;
       }
       byte[] xb = x.toByteArray();
       System.arraycopy(xb, 0, cmp, mother.getPCS()-xb.length, xb.length);
       if(y.testBit(0)) cmp[0] = 1;
```

```
       return cmp;
    }

    /** Adds another elliptic curve point to this point.
     *@param q The point to be added
     *@return the sum of this point on the argument
     *@exception Throws a NoCommonMotherException if the two points don't lie on
the same elliptic curve.*/
    public ECPoint add(ECPoint q) throws NoCommonMotherException{

       if (!hasCommonMother(q)) throw new NoCommonMotherException();

       if (this.iszero) return q;
       else if (q.isZero()) return this;

       BigInteger y1 = y;
       BigInteger y2 = q.gety();
       BigInteger x1 = x;
       BigInteger x2 = q.getx();

       BigInteger alpha;

       if (x2.compareTo(x1) == 0) {

           if (!(y2.compareTo(y1) == 0)) return new ECPoint(mother);
           else {
              alpha =
((x1.modPow(TWO,mother.getp())).multiply(THREE)).add(mother.geta());
              alpha =
(alpha.multiply((TWO.multiply(y1)).modInverse(mother.getp()))).mod(mother.getp());
           }

       } else {
           alpha =
((y2.subtract(y1)).multiply((x2.subtract(x1)).modInverse(mother.getp()))).mod(mothe
r.getp());
       }

       BigInteger x3,y3;
       x3 =
(((alpha.modPow(TWO,mother.getp())).subtract(x2)).subtract(x1)).mod(mother.getp());
       y3 = ((alpha.multiply(x1.subtract(x3))).subtract(y1)).mod(mother.getp());


       try{ return new ECPoint(mother,x3,y3); }
       catch (NotOnMotherException e){
           System.out.println("Error in add!!! Result not on mother!");
           return null;
       }

    }

    public ECPoint multiply(BigInteger coef) {
       try{
           ECPoint result = new ECPoint(mother);
           byte[] coefb = coef.toByteArray();
           if(fastcache != null) {
              for(int i = 0; i < coefb.length; i++) {
                  result = result.times256().add(fastcache[coefb[i]&255]);
```

```
            }
            return result;
         }
         if(cache == null) {
            cache = new ECPoint[16];
            cache[0] = new ECPoint(mother);
            for(int i = 1; i < cache.length; i++) {
                cache[i] = cache[i-1].add(this);
            }
         }
         for(int i = 0; i < coefb.length; i++) {
            result =
result.times16().add(cache[(coefb[i]>>4)&15]).times16().add(cache[coefb[i]&15]);
         }
         return result;
      } catch (NoCommonMotherException e) {
         System.out.println("Error in pow!!!");
         return null;
      }
   }

   private ECPoint times16() {
      try {
         ECPoint result = this;
         for(int i = 0; i < 4; i++) {
            result = result.add(result);
         }
         return result;
      } catch (Exception e) {
         System.out.println("ECPoint.times16: THIS CANNOT HAPPEN!!!");
         return null;
      }
   }

   private ECPoint times256() {
      try {
         ECPoint result = this;
         for(int i = 0; i < 8; i++) {
            result = result.add(result);
         }
         return result;
      } catch (Exception e) {
         System.out.println("ECPoint.times256: THIS CANNOT HAPPEN!!!");
         return null;
      }
   }

   public void setx(BigInteger xx){
      this.x = xx;
   }
   public void sety(BigInteger yy){
      this.y = yy;
   }

   public BigInteger getx(){
      return x;
   }

   public BigInteger gety(){
```

```
        return y;
    }

    public EllipticCurve getMother(){
        return mother;
    }

    public String toString(){
        return "(" + x.toString() + ", " + y.toString() + ")";
    }

    public boolean hasCommonMother(ECPoint p){
        if (this.mother.equals(p.getMother())) return true;
        else return false;
    }

    public boolean isZero(){
        return iszero;
    }
}
```

ECKey.java

```
package ECC;

import ECC.Key;
import ECC.Rand;
import java.math.BigInteger;
import java.io.*;

public class ECKey implements Key {
    /** There are to kinds of keys secret and public */
    /*
        protected boolean secret;
    protected BigInteger sk;
    protected ECPoint beta;
    protected EllipticCurve mother;
    */
        public boolean secret;
    public BigInteger sk;
    public ECPoint beta;
    public EllipticCurve mother;

    /** constructor **/

    public ECKey(EllipticCurve ec,ECPoint pt)
    {
      mother = ec;
      secret = false;
      //sk = 0;
      beta = pt;
      beta.fastCache();
    }

    /** ECKey generates a random secret key (contains also the public key)*/
    public ECKey(EllipticCurve ec) {
        mother = ec;
        secret = true;
        sk=new BigInteger(ec.getp().bitLength() + 17,Rand.om);
```

```
        if (mother.getOrder() != null) sk=sk.mod(mother.getOrder());
        beta=(mother.getGenerator()).multiply(sk);
        beta.fastCache();
    }

    public String toString() {
        if (secret) return ( "Secret key: "+ sk +" "+ beta +" "+ mother);
        else return("Public key:"+ beta +" "+ mother);
    }

    public boolean isPublic() {
        return (!secret);
    }

    public void writeKey(OutputStream out) throws IOException {
        DataOutputStream output = new DataOutputStream(out);
        mother.writeCurve(output);
        output.writeBoolean(secret);
        if(secret) {
            byte[] skb = sk.toByteArray();
            output.writeInt(skb.length);
            output.write(skb);
        }
        byte[] betab = beta.compress();
        output.writeInt(betab.length);
        output.write(betab);
    }

    public Key readKey(InputStream in) throws IOException {
        DataInputStream input = new DataInputStream(in);
        ECKey k = new ECKey(new EllipticCurve(input));
        k.secret = input.readBoolean();
        if(k.secret) {
            byte[] skb = new byte[input.readInt()];
            input.read(skb);
            k.sk = new BigInteger(skb);
        }
        byte[] betab = new byte[input.readInt()];
        input.read(betab);
        k.beta = new ECPoint(betab, k.mother);
        return k;
    }

    /** Turns this key into a public key (does nothing if this key is public) */
    public Key getPublic() {
        Key temp = new ECKey(mother);
        ((ECKey)temp).beta = beta;
        ((ECKey)temp).sk = BigInteger.ZERO;
        ((ECKey)temp).secret = false;
        System.gc();
        return temp;
    }
}
```

ECCryptoSystem.java

```
package ECC;

import ECC.*;
import java.math.BigInteger;
import java.io.*;
import java.util.*;
import java.security.MessageDigest;

public class ECCryptoSystem implements CryptoSystem {
    MessageDigest hash;
    MessageDigest h;
    private static final BigInteger TWO = BigInteger.valueOf(2);
    private static BigInteger y_cord;

    /** For testing Decryt method defining following static variables **/

    private static ECPoint C1,C2;
    private EllipticCurve ec;

    public ECCryptoSystem(EllipticCurve ec) {
       this.ec = ec;
       try {
           hash = MessageDigest.getInstance("SHA-1");
           h = MessageDigest.getInstance("SHA-1");
       } catch (java.security.NoSuchAlgorithmException e) {
           System.out.println("RSACryptoSystem: THIS CANNOT HAPPEN\n"+e);
           System.exit(0);
       }
    }

    public int blockSize() {
       return 20;
    }



/**    findY_coordinate Method **/

    public int findY_Coordinate(BigInteger x,BigInteger a,BigInteger b,BigInteger
p){

       int flag = 0;

       BigInteger x_cube = x.modPow(new BigInteger("3"),p);
       BigInteger y_cord_sqr = ((x_cube.add(a.multiply(x))).add(b)).mod(p);

       Legendre ret = new Legendre(p);
       int val = ret.calculate(y_cord_sqr);

       if ( val == 0 )
       {
              ECCryptoSystem.y_cord = BigInteger.ZERO;
              flag = 1;
       }
       else if ( val == 1 )
       {
              BigInteger temp1 = p.add(BigInteger.ONE);
```

```
            BigInteger temp2 = new BigInteger("4");
            temp1 = temp1.divide(temp2);

            BigInteger y1 = y_cord_sqr.modPow(temp1, p);
            BigInteger y2 = p.subtract(y1);

            if(y2.compareTo(y1) > 0)
                  ECCryptoSystem.y_cord = y1;
            else
                  ECCryptoSystem.y_cord = y2;
            flag = 1;

      }
      else
      {
            flag = 0;
            //System.err.println("Not a Quadratic residue modulo p");
      }
      return flag;

}



/********************/
/** encrypt method **/
/********************/

public byte[] encrypt(byte[] input,int numbytes, Key key,String lKey) {
      ECKey ek = (ECKey) key;
      System.out.println("Encrypting...");
      //String lKey = "191220160859592278988126";
BigInteger lk = new BigInteger(lKey);
byte[] res=new byte[ek.mother.getPCS()+numbytes];
      hash.reset();


      BigInteger rk = new BigInteger(ek.mother.getp().bitLength() + 17, Rand.om);
      if (ek.mother.getOrder() != null) {
          rk = rk.mod(ek.mother.getOrder());
      }
      ECPoint gamma = ek.mother.getGenerator().multiply(rk);      //value of C1
      //System.out.println("Printing value of C1.....->");
      //System.out.println(gamma.toString());

      ECPoint sec = ek.beta.multiply(rk);       //value of r.PbKey
      System.out.println("Printing value of r.Pkey.....->");
      System.out.println(sec.toString());

      ECPoint lkey = ek.mother.getGenerator().multiply(lk); //value of Lkey.G
      System.out.println("Printing value of location Key.....->");
      System.out.println(lkey.toString());

      System.arraycopy(gamma.compress(),0,res,0,ek.mother.getPCS());



      /** copying gamma to C1 **/
```

```
	try{
		C1 = new ECPoint(gamma.getMother(),gamma.getx(),gamma.gety());
	}catch(NotOnMotherException ex){
		System.err.println("Point Doesn't lie on the curve!!");
	}


	/** Mapping input message to a point on Elliptic Curve **/

	String binary = new BigInteger(input).toString(2);
	System.out.println("Input Text As binary: "+binary);	//converting the
input text to binary representation

	StringBuilder sb = new StringBuilder(binary);
	sb.append("00000000");
	//appending 8 bits at end of the binary representation of input text
	String s=sb.toString();
	System.out.println("Input Text after appending 8 bits: "+ s);
	BigInteger x_cord = new BigInteger(s,2);		//converting the binary
to corresponding decimal integer value
	System.out.println("Decimal equivalent: "+ x_cord);
	ECPoint Pm=null;


	int flag = 0;  // value 0 indicates not a Quadratic residue and value 1
indicates is a Quadratic residue
	flag =
findY_Coordinate(x_cord,ek.mother.geta(),ek.mother.getb(),ek.mother.getp());
	while(flag == 0)
	{
		x_cord = x_cord.add(BigInteger.ONE);
		flag =
findY_Coordinate(x_cord,ek.mother.geta(),ek.mother.getb(),ek.mother.getp());
	}
	try{
		Pm = new ECPoint(ek.mother,x_cord,y_cord);
		System.out.println("Printing coordinates of Pm------>
"+Pm.toString());

		try{
			C2 = Pm.add(sec.add(lkey));
			System.out.println("After Ecrypting---->");
			System.out.println("Printing coordinates of C1------
>"+C1.toString());
			System.out.println("Printing Coordinates of C2 ----->"+
C2.toString());
		}
		catch(NoCommonMotherException ex){}
	}
	catch(NotOnMotherException ex){
		System.out.println("Point Doesnt not lie on the curve!!");
	}



	System.out.println("");
	/*
	try{
	ECPoint nk = sec.add(lkey);
```

```
   //System.out.println("Testing Addition Operation....-> "+nk.toString());
   hash.update(nk.getx().toByteArray());
   hash.update(nk.gety().toByteArray());
   byte[] digest = hash.digest();
   for(int j = 0; j < (numbytes); j++) {
       res[j+ek.mother.getPCS()]=(byte) (input[j]^digest[j]);
       //System.out.println(res[j+ek.mother.getPCS()]);
   }




   }
   catch(NoCommonMotherException ex)
   {
          System.err.println("Points lie on different curves!!");
   }
   */
   /*hash.update(sec.getx().toByteArray());
   hash.update(sec.gety().toByteArray());
   byte[] digest = hash.digest();

   h.reset();
   h.update(lkey.getx().toByteArray());
   h.update(lkey.gety().toByteArray());
   byte[] dig =h.digest();

   for(int j = 0; j < ( numbytes); j++) {
       res[j+ek.mother.getPCS()]=(byte) (input[j]^digest[j]^dig[j]);
       System.out.println(res[j+ek.mother.getPCS()]);
   }*/
   //System.out.println(res);
   return res;
}



/********************/
/** decrypt method **/
/********************/

public byte[] decrypt(byte[] input, Key key,String lKey_decrypt) {
   ECKey dk = (ECKey) key;
   System.out.println("Decrypting...");
   //String lKey = "191220160859592278988126";
BigInteger lk = new BigInteger(lKey_decrypt);
   byte[] res=new byte[input.length-dk.mother.getPCS()];
   byte[] gammacom=new byte[dk.mother.getPCS()];
   hash.reset();
   byte[] arr=null;

   System.arraycopy(input,0,gammacom,0,dk.mother.getPCS());
   ECPoint gamma = new ECPoint(gammacom,dk.mother);
   ECPoint sec = gamma.multiply(dk.sk);
   ECPoint lkey = dk.mother.getGenerator().multiply(lk);
   System.out.println("Location Key: "+ lkey.toString());


   /** C2 - nB.C1 - Lk.G = Pm **/
```

```
	/** C2 - (nB.C1 + Lk.G )    = Pm **/
	/** C2 - sum           = Pm**/
	/** C2 + (-sum)              = Pm**/

	ECPoint skey = C1.multiply(dk.sk);
	try {
		ECPoint sum = skey.add(lkey);
		System.out.println("Coordinates of (nB.C1 + Lk.G ) before inverting---
-->");
		System.out.println(sum.toString());
		BigInteger y_cord = sum.gety();
		//y_cord = BigInteger.ZERO.subtract(y_cord);
		y_cord = sum.getMother().getp().subtract(y_cord);
		sum.sety(y_cord);
		System.out.println("Coordinates of (nB.C1 + Lk.G ) after inverting----
->");
		System.out.println(sum.toString());
		ECPoint plainText = C2.add(sum);
		System.out.println("Plaintext Coordinates After decryption 'Pm'-----
>");
		System.out.println(plainText.toString());
		BigInteger x_cord = plainText.getx();

		System.out.println("ReMapping point coordinates to string message---
>");
		String t = x_cord.toString(2);
		System.out.println("PlainText with 8 bits padding As binary: "+t);
		t= t.substring(0,t.length()-8);
		System.out.println("PlainText As binary: "+t);

		BigInteger bi = new BigInteger(t,2);
		System.out.println("Decimal Equivalent: "+bi);
		String output= new String(bi.toByteArray());
		System.out.println("After Decrypting---->");
		System.out.println("PlainText As text: "+output);
		 arr = output.getBytes();
	} catch (NoCommonMotherException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}

	/*
	try{
		ECPoint nk = sec.add(lkey);
		//System.out.println("Testing Addition Operation....->
"+nk.toString());
		hash.update(nk.getx().toByteArray());
		hash.update(nk.gety().toByteArray());
		byte[] digest = hash.digest();
		for(int j = 0; j < input.length-dk.mother.getPCS(); j++) {
		    res[j]=(byte) (input[j+dk.mother.getPCS()]^digest[j]);
		}

	}
	catch(NoCommonMotherException ex)
	{
		System.err.println("Points lie on different curves!!");
	}
	*/
```

```
        /*if(sec.isZero()) {
            hash.update(BigInteger.ZERO.toByteArray());
            hash.update(BigInteger.ZERO.toByteArray());
        } else {
            hash.update(sec.getx().toByteArray());
            hash.update(sec.gety().toByteArray());
        }
        byte[] digest = hash.digest();

        h.reset();
        h.update(lkey.getx().toByteArray());
        h.update(lkey.gety().toByteArray());
        byte[] dig =h.digest();

        for(int j = 0; j < input.length-dk.mother.getPCS(); j++) {
            res[j]=(byte) (input[j+dk.mother.getPCS()]^digest[j]^dig[j]);
        }
        */
        return arr;
    }



    /** This method generates a new key for the cryptosystem.
     *@return the new key generated*/
    public Key generateKey() {
        return new ECKey(ec);
    }

    public String toString(){
        return "ECC - " + ec.toString();
    }

}
```

APPENDIX D: Helper functions

Rand.java

```
package ECC;
import java.security.SecureRandom;
public class Rand {
    public static final SecureRandom om = new SecureRandom();
}
```

Legendre.java

```
package ECC;
import java.math.BigInteger;
public class Legendre {

       /** Method will return this if legendre symbol equal to zero **/
       public static final int DIVISIBLE = 0;

       /** Method will return this if legendre symbol equal to minus 1 **/
       public static final int NONRESIDUE = -1;

       /** Method will return this if legendre symbol equal to one **/
       public static final int RESIDUE = 1;

       private BigInteger primeField;

       /** Defining Constructor to initialize primeField **/
       Legendre(BigInteger p)
       {
              this.primeField = p;
       }

       /**
     * The Legendre symbol can be calculated by using the following formula
     *
     * |a| (p-1)/2 --- = a mod p |p|
     *
     * @param value
     *            Value to check to see if it is a quadratic residue modulo p
     * @return whether or not an integer is a quadratic residue modulo p
     */
    public int calculate(BigInteger value) {

        // (a/p) = a^((p-1)/2) mod p where (a/p) is the legendre symbol
        BigInteger exponent = primeField.subtract(BigInteger.ONE);
        exponent = exponent.divide(new BigInteger("2"));

        BigInteger result = value.modPow(exponent, primeField);
        if (result.compareTo(BigInteger.ZERO) == 0)
            return DIVISIBLE;
         else if (result.compareTo(BigInteger.ONE) == 0)
            return RESIDUE;
         else
            return NONRESIDUE;
    }
}
```

APPENDIX E: Main function

TestECCrypto.java

```
package ECC;

import ECC.*;
import ECC.*;
import java.util.*;
import java.io.*;
import java.math.BigInteger;

public class TestECCrypto {
    public static void main(String[] args)throws IOException {
       try {
           EllipticCurve ec = new EllipticCurve(new secp256r1());

           CryptoSystem cs = new ECCryptoSystem(ec);

           Key sk = (ECKey)cs.generateKey();
           Key pk = sk.getPublic();

         //to obtain x and y coordinates of public key

           BigInteger x_cord=((ECKey)pk).beta.getx();
           BigInteger y_cord=((ECKey)pk).beta.gety();

           byte[] xb = x_cord.toString().getBytes();
           byte[] yb = y_cord.toString().getBytes();

           String xs = new String(xb);
           String ys = new String(yb);
           String msg = new String(x_cord.toString() + "," + y_cord.toString());

           byte[] msgb = msg.getBytes();
           String recv = new String(msgb);

          System.out.println("Curve Details--------->");
          System.out.println("Name: "+ ec.toString());
          System.out.println("a: "+ ec.geta());
          System.out.println("b: "+ ec.getb());
          System.out.println("p: "+ ec.getp());
          System.out.println("Order: "+ ec.getOrder());
          System.out.println("Generator: "+ ec.getGenerator().toString());
          System.out.println("");


         String text = "!! Location Based Elliptic Curve Cryptography !!";

           System.out.println("InputText----->\n" +text );

           InputStreamReader is = new InputStreamReader(System.in);
           BufferedReader br = new BufferedReader(is);

           System.out.println("Enter the location Coordinates:");
           System.out.println("Latitude: ");
           String latitude= br.readLine();
           System.out.println("Longitude: ");
```

```
String longitude= br.readLine();

System.out.println("Enter the date (ddmmyyyy) :");
String dt = br.readLine();

System.out.println("Enter the time frame (hhmmss) :");
System.out.println("Start time:");
String start = br.readLine();
System.out.println("End time:");
String end = br.readLine();

String lKey = dt + start + latitude + longitude ;

//String lKey = "191220160859592278988126";


System.out.println("Location Key: " + lKey);
//System.out.println(lKey);

/** Receiver's details**/
System.out.println("Receiver's LKey--->");
System.out.println("Enter the location Coordinates:");
System.out.println("Latitude: ");
String rlatitude= br.readLine();
System.out.println("Longitude: ");
String rlongitude= br.readLine();
System.out.println("Enter the date (ddmmyyyy) :");
String rdt = br.readLine();
System.out.println("Enter the time (hhmmss) :");
String rstart = br.readLine();

//Constructing an ECPoint

try{

ECPoint pt=new ECPoint(ec,x_cord,y_cord);
Key pubKey=new ECKey(ec,pt);
System.out.println("Printing Public Key formed----->");
System.out.println(pubKey.toString());

StringBuilder sb= new StringBuilder("");
   int chunk = 5;
   int i=0;
   int flag =0;
   while(flag == 0 && i< text.length()){

          String x;
          if((i+chunk)<text.length())
                 x = text.substring(i, i+chunk);
          else
                 {
                 x = text.substring(i, text.length());
                 flag=1;
                 }
                 i+=chunk;
          System.out.println("Current chunk---> "+x);
byte[] t1 = x.getBytes();
byte[] t2 = cs.encrypt(t1,t1.length,pubKey,lKey);
String str = new String(t2);
```

```
            //System.out.println("After Encryption---->");
            //System.out.println(str);



            /** Checking whether receiver lies in the time range and location!! **/

            if(latitude.equals(rlatitude) && longitude.equals(rlongitude) &&
dt.equals(rdt) && (start.compareTo(rstart)<=0) && (end.compareTo(rstart)>=0))
            {
            String lKey_decrypt = rdt + start + rlatitude + rlongitude ;
            byte[] t3 = str.getBytes();
            byte[] t4 = cs.decrypt(t2, sk,lKey_decrypt);
            String str2 = new String(t4);
            sb.append(str2);
            System.out.println("");
           // System.out.println("After decryption---->");
           // System.out.println(str2);
            }
            else
               System.err.println("Message cannot be Decrypted!!");

                }//end of while
               System.out.println("Complete Text After decryption:\n" +
sb.toString());
           }//end of try
            catch(NotOnMotherException e){
               System.err.println("Error");
            }

       } catch (InsecureCurveException e) {
            System.out.println("TestCryptoStreams: "+e);
       }
    }
}
```

Sample Output:

The screenshots of the output is:

```
Curve Details--------->
Name: secp256r1
a: 115792089210356248762697446949407573530086143415290314195533631308867097853948
b: 41058363725152142129326129780047268409114441015993725554835256314039467401291
p: 115792089210356248762697446949407573530086143415290314195533631308867097853951
Order: 115792089210356248762697446949407573529996955224135760342422259061068512044369
Generator: (48439561293906451759052585252797914202762949526041747995844080717082404635286, 36134250956749795798585127919587881956611106672985015071877198253568414405109)

InputText----->
Location Based ECC
Enter the location Coordinates:
Latitude:
22789
Longitude:
88126
Enter the date (ddmmyyyy) :
19122016
Enter the time frame (hhmmss) :
Start time:
085959
End time:
185959
Location Key: 191220160859592278988126
Receiver's LKey--->
Enter the location Coordinates:
Latitude:
22789
Longitude:
88126
Enter the date (ddmmyyyy) :
19122016
Enter the time (hhmmss) :
130000
Printing Public Key formed----->
Public key:(237492512092480414338730601452673033795287713333709976642296248707465955902888, 68357325632094627838018962836348341714332507387467190329301170182812963313536) secp256r1


Current chunk---> Location Based ECC
Encrypting...
Printing value of location Key.....->
(896544479731124073394561603569498633932570677871269207853354264487789012547791, 782224046147717884589272756234351330833755271029881699757880157047196056230 4)
Input Text As binary: 1001100011011110110001101100001011101000110100101101111011011000100000010000100110000101110011011001010110010000100000010001010100001101000011
Printing coordinates of Pm------> (17045599163206384271860250813798347276510338 56, 47899110760267622817420094022639358981008305874751397829969814134504225754159)
After Ecrypting---->
Printing coordinates of C1------>(11521465541238453351069342340591842878601762230492994507648700933316468484745 2, 14546053617615203165286380626146400534620751206220518903352938747510871969371)
Printing Coordinates of C2 ----->(32255907999290705562772637106684920393218286301502266382805160648886720119261, 10707694859500208910403388917327390654685601153791345199440126653073381996177 5)

Decrypting...
Location Key: (896544479731124073394561603569498633932570677871269207853354264487789012547791, 782224046147717884589272756234351330833755271029881699757880157047196056230 4)
Plaintext Coordinates After decryption 'Pm'----->
(17045599163206384271860250813798347276510338 56, 47899110760267622817420094022639358981008305874751397829969814134504225754159)
ReMapping point coordinates to string message--->
PlainText As binary: 1001100011011110110001101100001011101000110100101101111011011000100000010000100110000101110011011001010110010000100000010001010100001101000011
After Decrypting---->
PlainText As text: Location Based ECC

Complete Text After decryption:
Location Based ECC
```

REFERENCES

1. Cryptography & Network Security – Behrouz A. Forouzan
2. Journal: Message mapping and reverse mapping in elliptic curve cryptosystem - Aritro Sengupta and Utpal Kumar Ray
3. Journal: A Location Based Encryption Technique and Some of Its Applications - Logan Scott and Dorothy E. Denning
4. https://www.oracle.com/java/
5. https://developer.android.com/studio/intro/index.html
6. http://www.tutorialsface.com/2015/08/building-your-own-android-chat-messenger-app-similar-to-whatsapp-using-xmpp-smack-4-1-api-from-scratch-part-1/