Amrita Vishwa Vidyapeetham

Amrita School of Engineering, Coimbatore

B.Tech Continuous Assessments

Third Semester, Computer Science and Engineering (Cyber Security)

20CYS205 MODERN CRYPTOGRAPHY

PROGRAMMING ASSIGNMENT

Date:12/01/2024

Topic: Implementation of Elliptical Curve Signcryption Scheme

Team: Team 16

Team Members:

Kolluru Sai Supraj	-22065
S Parvathi	-22066
Amita Narayanan Kutty	-22067
Mukesh R	-22068

Faculty Mentor: Mr. Saurabh Srivastava

INDEX:

1. INTRODUCTION

2. SIGNCRYPTION SCHEME

    2.1. Parameter

3. CORRECTNESS

    3.1 . Correctness of the Scheme

4. CODES

    4.1. Python Code

    4.2. UI Code

5.OUTPUT SCREENSHOTS

6.ERRORS

7.USER MANUAL

    7.1 User Manual Specification

    7.2 User Manual Image

8.CONCLUSION

9.REFERENCES

List of figures:

7.2 Image Depicting the User Manual

1. INTRODUCTION

Elliptic Curve Signcryption (ECSC) is a cryptographic technique that combines the functionalities of both digital signature and encryption within the framework of elliptic curve cryptography (ECC). This innovative approach aims to provide a more efficient and streamlined solution for securing communication in various applications, particularly in resource-constrained environments.

Elliptic curve cryptography leverages the mathematical properties of elliptic curves to achieve strong security with shorter key lengths compared to traditional methods like RSA. In the context of ECSC, this mathematical foundation is harnessed to create a single cryptographic primitive that encompasses both signing and encrypting operations.

2. SIGNCRYPTION SCHEME

2.1. Parameter

Elliptical Curve Signcryption involves several parameters that are crucial for its implementation. Here are the key parameters involved :

2.1.1. Prerequisites

- A finite field GF(q) - order of q
- A prime q – length of l
- An elliptic curve E of the form $y^2=x^3+ax+b(\text{mod } q)$ – a,b should belong to GF(q) and $4a^2+27b^3=0(\text{mod } q)$
- Base point P of the curve E (ord(P)=n) – n is a large prime
- Co factor h=#E(GF(q))/n – h<<n
- #E(GF(q)) represents the number of points of the elliptic curve E defined on the finite field .
- Two hash fuctions H_1:G1 → {0,1} , H_2: {0,1} → Z_q
    - Parameters D={q,l,a,b,P,G1,n,h}

2.1.2. Key Generation

- The sender selects a random key SKs as their private key and Private key: $PK_R=SK_S$ P.
- Receiver private key $PK_R=SK_R$ P
- SK_S and SK_R are kept secret and PK_R and PK_S are exposed.

2.1.3. Signcrypt

- Select a random k from [1, n-1]
- Calculate kPK_R= K
- Calculate b=H_1(K)
- Calculate c=b XOR m (m=message)
- Calculate e=$H_2(m,K,PK_S,PK_R)$
- Calculate $s=k^{-1}(e+SK_S)$ If s=0 return to step 1.
- Get the signcryption σ=(c,e,s). This is sent to the receiver.

2.1.4. Unsigncrypt

The receiver gets the signcryption σ=(c,e,s) and uses PK_S and SK_R to unsigncrypt it :

- Calculate $w=s^{-1}$
- Calculate $X=ewPK_R + wPK_SSK_R$
- Calculate $b'=H_1(X)$
- Calculate m=b' XOR c
- Calculate $e'=H_2(m,X,PK_S,PK_R)$
- If e'=e, return m, other wise return "⊥"

3. CORRECTNESS

3.1. Correctness of the Scheme

$S=k^{-1}(e+SK_s)$

$s^{-1}=k(e+SK_s)^{-1}$

$X=ewPK_R + wPK_SK_R$

$=es^{-1}PK_R + s^{-1}PK_SSK_R$

$=es^{-1}SK_SSK_RP$

$=(e+SK_s)k(e+SK_s)^{-1}SK_RP$

$=kSK_RP$

$=kPK_R$

$=K$

So we have b’=b , e’=e.

b’=b ensures that the receiver can store the sender’s message m i.e the decryption process is correct,

e’=e ensures that the receiver can verify the correctness of the sender’s signature i.e the verification process is correct.

4. CODE

4.1. Python Code

```
from flask import *
import random

app=Flask(__name__)

def points_on_curve(a,b,p):
    qr=list()
    for i in range(p):
        qr.append(i**2%p)
    points = []
    for x in range(p):
        y_squared = (x**3 + a * x + b) % p
        if y_squared in qr:
            y = qr.index(y_squared)
            points.append((x, y))
            if y != 0:
                points.append((x, p - y))
    return points


def base_point_order(*parameters):
    i=1

    if len(parameters)==5:
        (x2,y2,a,b,p)=parameters
        stop=len(points_on_curve(a,b,p))
        opt=1

    elif len(parameters)==6:
        (x2,y2,a,b,p,stop)=parameters
        opt=2
        stop%=len(base_point_order(x2,y2,a,b,p))+1

    stop%=(len(points_on_curve(a,b,p))+1)
    points=[(x2,y2)]

    if stop==1:
        return points[0]

    if y2==0:
        if opt==2:
```

```
            return points[0]
        return points


    lamda = (3*(x2**2) + a) * pow((2*y2),-1,p)%p
    x3=(lamda**2-2*x2)%p
    y3=(lamda*(x2-x3)-y2)%p

    i+=1

    points.append((x3,y3))

    (x1,y1)=(x3,y3)

    while i<stop+1:
        if x2!=x1:
            lamda=((y1-y2)*pow((x1-x2),-1,p))%p

            x3=(lamda**2-x1-x2)%p
            y3=(lamda*(x1-x3)-y1)%p

            points.append((x3,y3))

            (x1,y1)=(x3,y3)

            if i==stop:
                return points[stop-1]

            i+=1

        else:
            if i==stop and opt==2:
                return points[stop-1]
            i+=1
            return points


def possible_base_points(x2,y2,a,b,q):
    pbs=[]
    for i in (points_on_curve(a,b,q)):

            (x2,y2)=i

            if is_prime(len(base_point_order(x2,y2,a,q))+1):
                pbs.append((x2,y2))
    return pbs

def unsigncrypt(sigma,PKs,PKr,SKr,q,n,a,b):
    (c,e,s)=sigma
    w=pow(s,-1,n)
    x1=base_point_order(PKr[0],PKr[1],a,b,q,e*w)
    x2=base_point_order(PKs[0],PKs[1],a,b,q,w*SKr)
    if x1==None and x2==None:
```

```
        return None
    elif x1==None:
        X=x2
    elif x2==None:
        X=x1
    else:
        X=points_add(x1,x2,a,b,q)
    b1=Hash1(X[0])
    b1=int(b1,2)
    m=b1^c
    e1=Hash2(bin(m)[2:],X,PKs,PKr,q)
    if e==e1:
        return m
    else:
        return "⊥" #symbol ⊥demonstrates that the attempted decryption of a
ciphertext that does not pass the authenticity check

def signcrypt(PKs,PKr,SKs,m,n,a,b,q):
    set=True
    count=0
    while set:
        k=random.randint(1,n-1)
        K=base_point_order(PKr[0],PKr[1],a,b,q,k)
        b=Hash1(K[0])
        b=int(b,2)
        c=b^m
        e=Hash2(bin(m)[2:],K,PKs,PKr,q)
        s=(pow(k,-1,n)*(e+SKs))%n
        if s!=0:
            set=False
    sigma=(c,e,s)

    return sigma

def points_add(p1,p2,a,b,q):
    (x1,y1)=p1
    (x2,y2)=p2
    if p1!=p2:
        lamda=((y1-y2)*pow((x1-x2),-1,q))%q
        x3=(lamda**2-x1-x2)%q
        y3=(lamda*(x1-x3)-y1)%q
    else:
        (x3,y3)=base_point_order(x1,y1,a,b,q,2)

    return x3,y3

def  KeyGen(P,a,b,p,SKs,SKr):
    (x,y)= P
    PKs=base_point_order(x,y,a,b,p,SKs)
    PKr=base_point_order(x,y,a,b,p,SKr)
    return PKs,PKr

def is_prime(n):
```

```
  if n == 2 or n == 3: return True
  if n < 2 or n%2 == 0: return False
  if n < 9: return True
  if n%3 == 0: return False
  r = int(n**0.5)
  # since all primes > 3 are of the form 6n ± 1
  # start with f=5 (which is prime)
  # and test f, f+2 for being prime
  # then loop by 6.
  f = 5
  while f <= r:
    if n % f == 0: return False
    if n % (f+2) == 0: return False
    f += 6
  return True

def Hash2(binary_string, point1, point2, point3, prime_q):
    # Custom hash function using basic operations
    hash_value = 0

    # Process binary string
    for char in binary_string:
        hash_value = (hash_value * 31 + ord(char)) % prime_q

    # Process curve points
    for point in [point1, point2, point3]:
        hash_value = (hash_value * 31 + point[0]) % prime_q
        hash_value = (hash_value * 31 + point[1]) % prime_q

    return hash_value

def Hash1(input_number):

    hash_value = (input_number * 7) % 32

    return bin(hash_value)[2:]

@app.route('/')
def welcome():
    return render_template('index.html',k="")

@app.route('/points',methods=['POST'])
def gen_point():
    possible_base_points=[]
    possible_base_points_prime=[]
    a = int(request.form['inputA'])
    b = int(request.form['inputB'])
    q = int(request.form['inputQ'])

    if a==0 and b==0:
        k = "Both a and b are equal to zero"
        return render_template('index.html',k=k)
```

```
    elif (is_prime(q)==False):
        k = f"{q} is not a prime."
        return render_template('index.html',k=k)

    elif (4*(a**3)+27*(b**2))%q==0:
        k = f"The elliptic curve y² = x³+{a}x+{b} (mod {q}) is not singular"
        return render_template('index.html',k=k)


    for i in (points_on_curve(a,b,q)):

        (x2,y2)=i

        possible_base_points.append([(x2,y2),len(base_point_order(x2,y2,a,b,q))+
1])
        if is_prime(len(base_point_order(x2,y2,a,b,q))+1):
            possible_base_points_prime.append([(x2,y2),len(base_point_order(x2,y
2,a,b,q))+1])
    return
render_template('points.html',points=possible_base_points,p_points=possible_base
_points_prime,a=a,b=b,q=q)

@app.route('/sign',methods=['GET','POST'])
def sig_crypt():
    p = request.form['enteredPoint'].split(',')
    q = int(p[0][1:])
    r = int(p[1][:-1])
    (x2,y2) = (q,r)
    SKs = int(request.form['senderSecretKey'])
    SKr = int(request.form['receiverSecretKey'])
    m = int(request.form['message'])
    a = int(request.form['inputA'])
    b = int(request.form['inputB'])
    q = int(request.form['inputQ'])
    n=(len(base_point_order(x2,y2,a,b,q))+1)
    while SKs>=n or SKr>=n:
        if SKs>=n:
            print()
            SKs=int(request.form['senderSecretKey'])
        if SKr>=n:
            print()
            SKr=int(request.form['receiverSecretKey'])
    l=len(bin(q)[2:])

    PKs,PKr=KeyGen((x2,y2),a,b,q,SKs,SKr)

    sigma=signcrypt(PKs,PKr,SKs,m,n,a,b,q)

    return
render_template('unsigncrypt.html',sigma=sigma,P=(x2,y2),PKs=PKs,PKr=PKr,SKs=SKs
,SKr=SKr,m=m,n=n,a=a,b=b,q=q,l=l)

@app.route('/resign',methods=['POST'])
```

```
def resign():
    message =
    request.form['signature'].split(''
    )
    t = int(message[0][1:])
    r = int(message[1])
    s = int(message[2][:-1])
  sigma = (t,r,s)
  n = int(request.form['n'])
  a = int(request.form['inputA'])
  b = int(request.form['inputB'])
  q = int(request.form['inputQ'])
  PKs = request.form['senderPublicKey'].split(',')
  x = int(PKs[0][1:])
  y = int(PKs[1][:-1])
  PKs = (x,y)
  PKr = request.form['receiverPublicKey'].split(',')
  d = int(PKr[0][1:])
  e = int(PKr[1][:-1])
  PKr = (d,e)
  SKr = int(request.form['receiverSecretKey'])

  m1 =unsigncrypt(sigma,PKs,PKr,SKr,q,n,a,b)
```

4.2. UI Code

Main Page :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Elliptic Curve Cryptography</title>
    <style>


        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 20px;
            background-color: yellow;
        }

        .container {
            max-width: 600px;
            margin:  auto;
            color: rgb(0, 0, 0);
        }

        input, button {
```

```
            margin: 10px 0;
            padding: 8px;
            width: 100%;
            box-sizing: border-box;
        }

        button {
            background-color: #784caf;
            color: white;
            border: none;
            cursor: pointer;
        }

        button:hover {
            background-color: rgb(255, 0, 0);
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>SIGNCRYPTION USING ELLIPTIC CURVE CRYPTOGRAPHY</h1>

        <form action="/points" method="post">

            <h3> ENTER THE PARAMETERS OF THE CURVE</h3>

            <label for="inputA">Enter A:</label>
            <input type="number" name="inputA" required>

            <label for="inputB">Enter B:</label>
            <input type="number" name="inputB" required>

            <label for="inputQ">Enter Q:</label>
            <input type="number" name="inputQ" required>

            <button type="submit">Generate points on the curve</button>

        </form>
    </div>
    {%print(k)%}

</body>
</html>
```

Page 2 :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Elliptic Curve Cryptography - Result Page</title>
```

```
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 20px;
            background-color: #ffd700; /* Light Gold */
        }

        .container {
            max-width: 600px;
            margin: auto;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            background-color: white;
        }

        h2 {
            color: #784caf; /* Purple */
        }


        form {
            margin-top: 20px;
        }

        label {
            display: block;
            margin-top: 10px;
            font-weight: bold;
        }

        input {
            margin-top: 5px;
            padding: 10px;
            width: calc(100% - 20px);
            box-sizing: border-box;
        }

        button {
            background-color: #4CAF50;
            color: white;
            border: none;
            padding: 10px 20px;
            cursor: pointer;
            border-radius: 5px;
            margin-top: 20px;
        }

        button:hover {
            background-color: #45a049;
        }
        p {
```

```
            margin-top: 20px;
        }

        a {
            color: #45a049; /* Green */
            text-decoration: none;
            font-weight: bold;
        }

        a:hover {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <h3>Generated Points:</h3>
    <ul>
        <li>Point {{1}}: θ</li>
        {% for i in points %}
            <li>Point{{points.index(i)+2}}: {{ i[0] }} of order {{i[1]}}</li>
        {% endfor %}
    </ul>
    <br>
    <h3>Prime ordered Points:</h3>
    <ul>

        {% for i in p_points %}
            <li>Point{{p_points.index(i)+2}}: {{ i[0] }} of order {{i[1]}}</li>
        {% endfor %}
    </ul>
    <div class="container">
        <h2>Points Generated By the Curve</h2>
        <form action="/sign" method="post">
            <label for="inputA">a:</label>
            <input type="number" name="inputA" value={{ a }}>

            <label for="inputB">b:</label>
            <input type="number" name="inputB" value={{ b }}>

            <label for="inputQ">q:</label>
            <input type="number" name="inputQ" value={{ q }}>

            <label for="enteredPoint">Selected Point:</label>
            <input type="text" id="enteredPoint" name="enteredPoint" required>

            <label for="senderSecretKey">Sender's Secret Key:</label>
            <input type="number" id="senderSecretKey" name="senderSecretKey"
required>


            <label for="receiverSecretKey">Receiver's Secret Key:</label>
            <input type="number" id="receiverSecretKey" name="receiverSecretKey"
required>
```

```
            <label for="Message">Message to be encrypted:</label>
            <input type="number" id="message" name="message" required>

            <button type="submit">Submit</button>
        </form>
        <p><a href="/">Go back to the input page</a></p>
    </div>
</body>
</html>
```

Page 3:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
       body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 20px;
            background-color: #ffd700; /* Light Gold */
        }

        .container {
            max-width: 600px;
            margin: auto;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            background-color: white;
        }

        h2 {
            color: #784caf; /* Purple */
        }

        form {
            margin-top: 20px;
        }

        label {
            display: block;
            margin-top: 10px;
            font-weight: bold;
        }

        input {
            margin-top: 5px;
            padding: 10px;
            width: calc(100% - 20px);
```

```
            box-sizing: border-box;
        }

        button {
            background-color: #4CAF50;
            color: white;
            border: none;
            padding: 10px 20px;
            cursor: pointer;
            border-radius: 5px;
            margin-top: 20px;
        }

        button:hover {
            background-color: #45a049;
        }

        p {
            margin-top: 20px;
        }

        a {
            color: red; /* Green */
            text-decoration: none;
            font-weight: bold;
        }

        a:hover {
            text-decoration: underline;
        }
    </style>
    <title>Recovered Message</title>
</head>
<body>
    <Center>
    <h1><pre><strong>Recovered Message:</strong>    {{m1}}</pre></h1>
    <br>
    <p><a href="/">Go back to the input page</a></p>
</body>
</html>
```

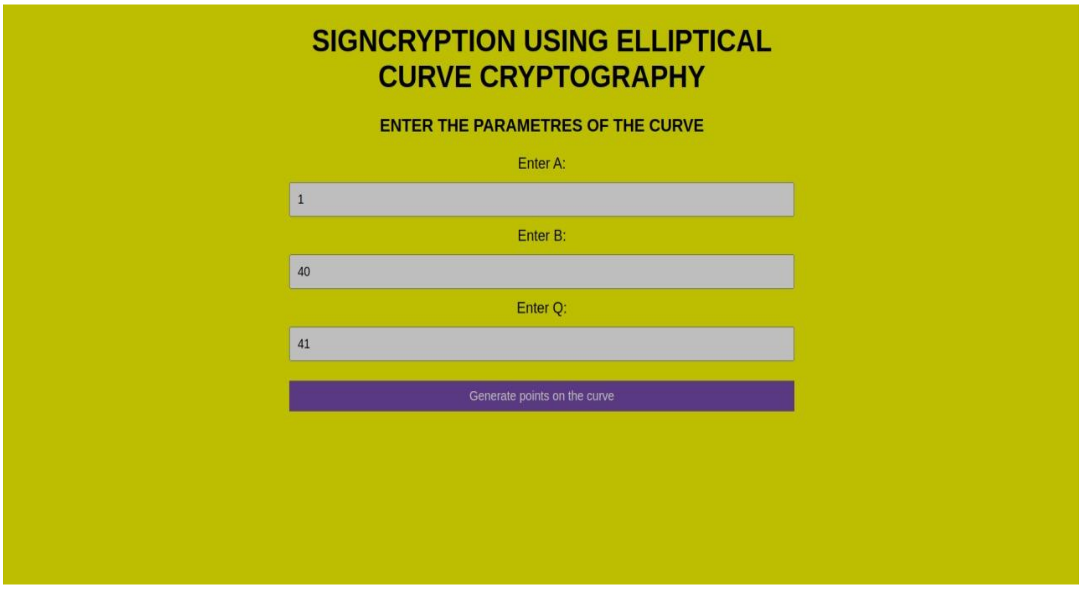
Page 4 :

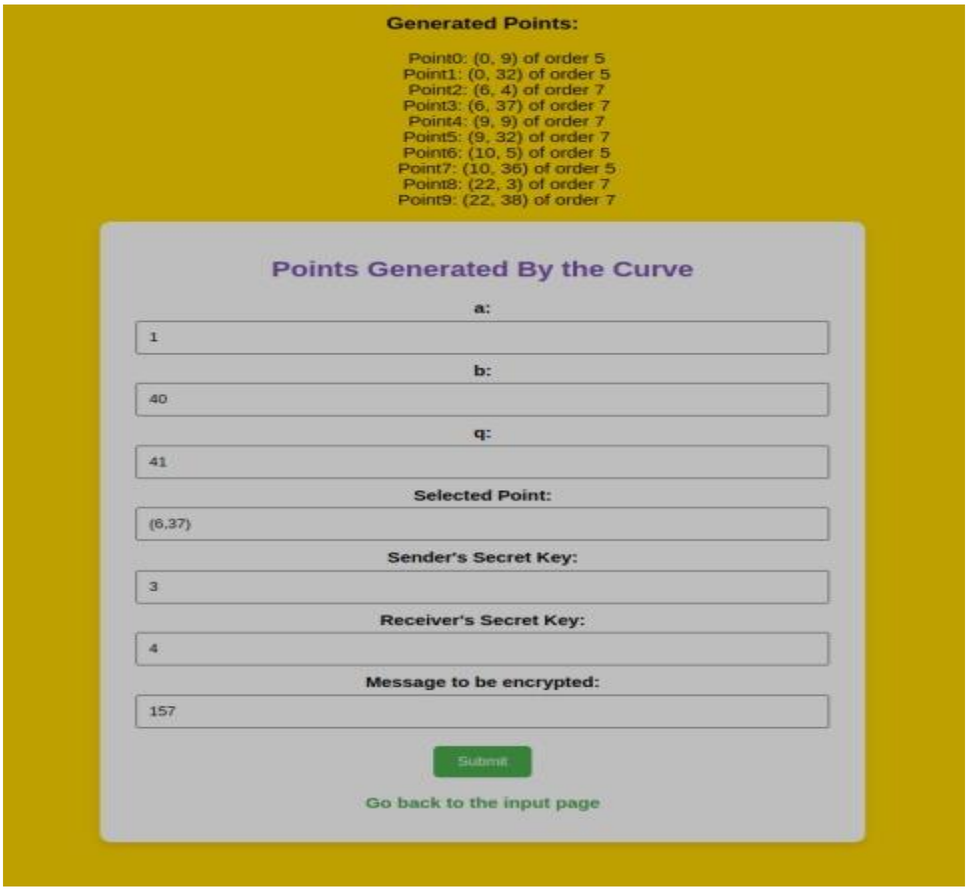
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Signature Display</title>
    <style>
        body {
            font-family: Arial, sans-serif;
```

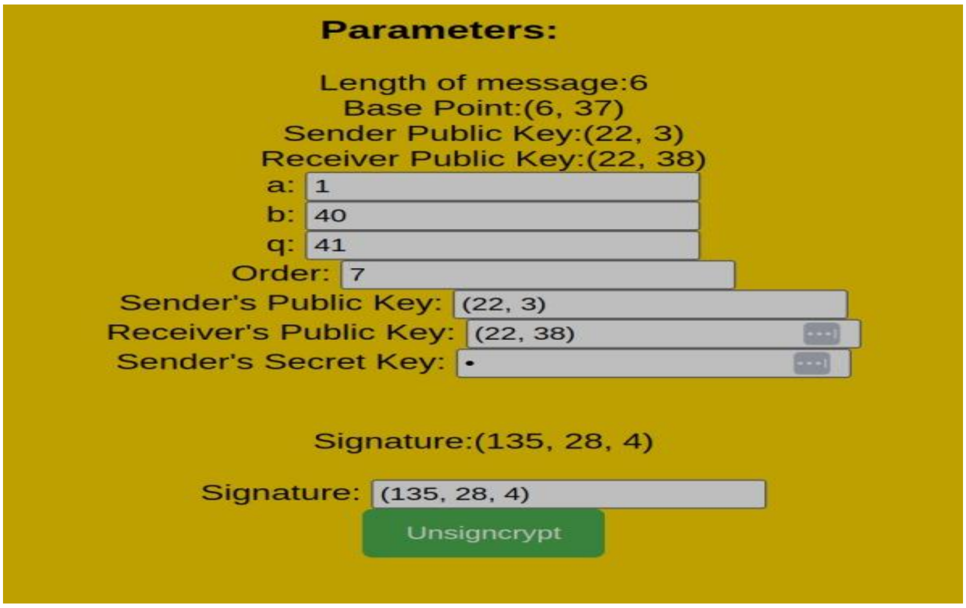
```
            text-align: center;
            margin: 20px;
            background-color:#ffd700;
        }

        .container {
            max-width: 400px;
            margin: auto;
            padding: 20px;
            border: 1px solid #ddd;
            border-radius: 8px;
            background-color: white;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }

        .signature-box {
            border: 1px solid #ddd;
            padding: 20px;
            margin-bottom: 20px;
        }

        button {
            background-color: #4CAF50;
            color: white;
            border: none;
            padding: 10px 20px;
            cursor: pointer;
            border-radius: 5px;
        }

        button:hover {
            background-color: #45a049;
        }

        .additional-box {
            border: 1px solid #ddd;
            padding: 20px;
        }
    </style>
</head>
<body>
    <form action="/resign" method="post">
        <h3>Parameters:</h3>
    <ul>
            <li>Length of message:{{l}}</li>
            <li>Base Point:{{P}}</li>
            <li>Sender Public Key:{{PKs}}</li>
            <li>Receiver Public Key:{{PKr}}</li>

            <label for="inputA">a:</label>
            <input type="number" name="inputA" value={{ a }}><br>

            <label for="inputB">b:</label>
```

```
            <input type="number" name="inputB" value={{ b }}><br>

            <label for="inputQ">q:</label>
            <input type="number" name="inputQ" value={{ q }}><br>

            <label for="inputQ">Order of {{P}}:</label>
            <input type="number" name="n" value={{ n }}><br>

            <label for="senderPublicKey">Sender's Public Key:</label>
            <input type="text" id="senderPublicKey" name="senderPublicKey"
required><br>

            <label for="receiverPublicKey">Receiver's Public Key:</label>
            <input type="text" id="receiverPublicKey" name="receiverPublicKey"
required><br>

            <label for="receiverSecretKey">Receiver's Secret Key:</label>
            <input type="password" id="receiverSecretKey"
name="receiverSecretKey"><br>

            <br>
            <br>
            <li>Signature:{{sigma}}</li><br>
            <label for="Signature">Signature:</label>
            <input type="text" id="signature" name="signature" required><br><br>
        <button type="submit">Unsigncrypt</button>
    </ul>

    </form>

</body>
</html>
```

5.OUTPUT SCREENSHOTS




Generated Points:
Point0: (0, 9) of order 5
Point1: (0, 32) of order 5
Point2: (6, 4) of order 7
Point3: (6, 37) of order 7
Point4: (9, 9) of order 7
Point5: (9, 32) of order 7
Point6: (10, 5) of order 5
Point7: (10, 36) of order 5
Point8: (22, 3) of order 7
Point9: (22, 38) of order 7
Points Generated By the Curve
a:
1
b:
40
q:
41
Selected Point:
(6,37)
Sender's Secret Key:
3
Receiver's Secret Key:
4
Message to be encrypted:
157
Submit
Go back to the input page

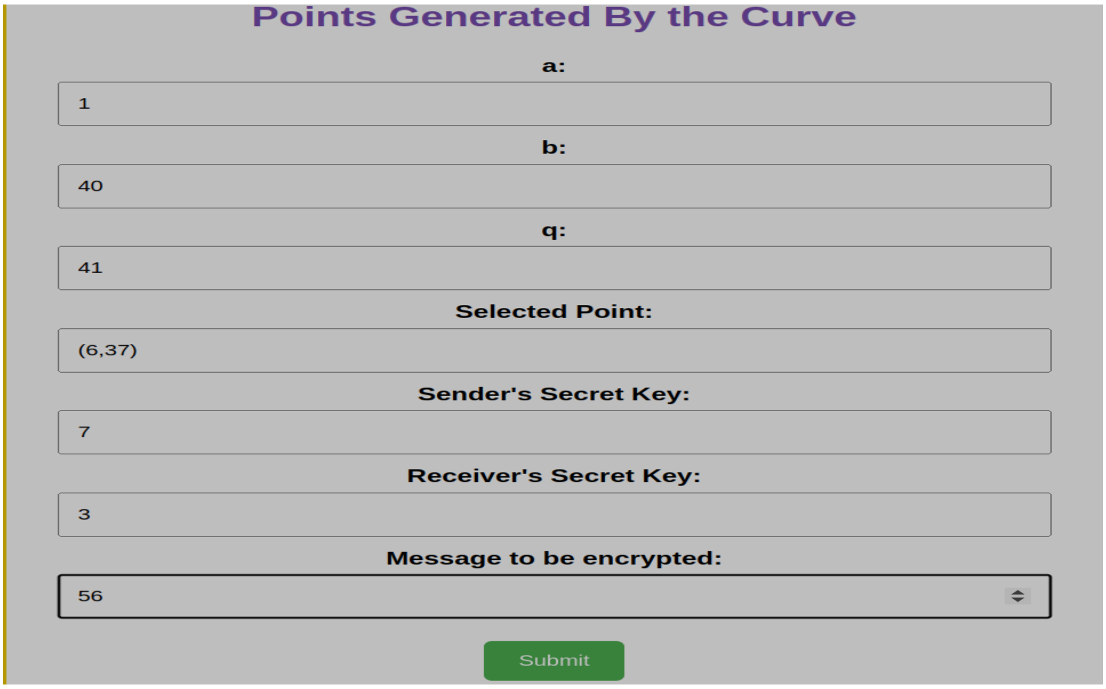

Parameters:
Length of message:6
Base Point:(6, 37)
Sender Public Key:(22, 3)
Receiver Public Key:(22, 38)
a: 1
b: 40
q: 41
Order: 7
Sender's Public Key: (22, 3)
Receiver's Public Key: (22, 38)
Sender's Secret Key: •
Signature:(135, 28, 4)
Signature: (135, 28, 4)
Unsigncrypt

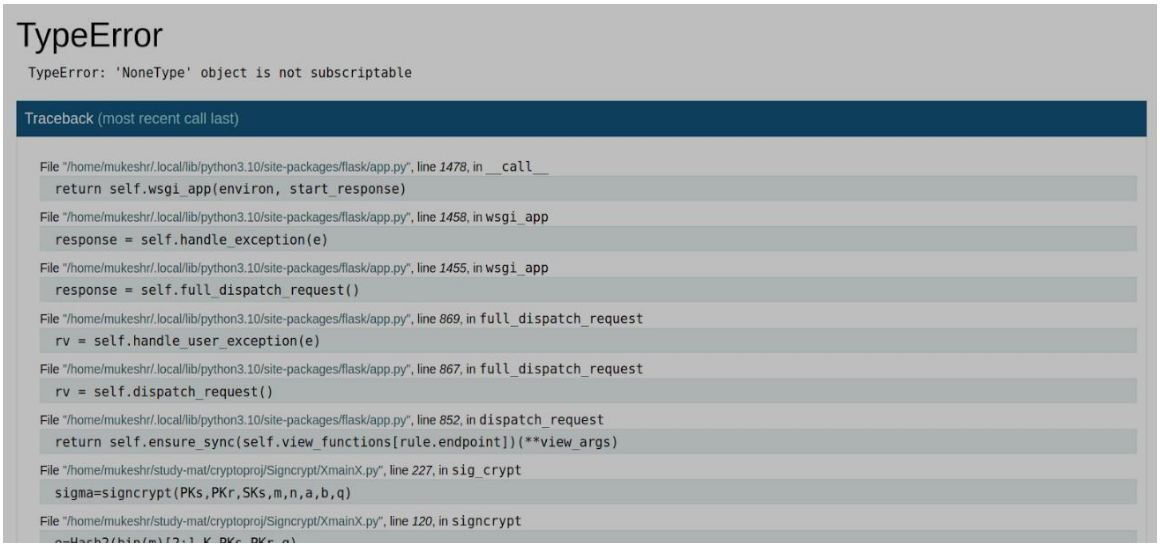

Recovered Message: 157

6.ERRORS

6.1 We made the user enter private keys less than the order of the point chosen, to avoid complications in the algorithm.

Before:





After:

```
    while SKs>=n or SKr>=n:
        if SKs>=n:
            print()
            SKs=int(request.form['senderSecretKey'])
        if SKr>=n:
            print()
            SKr=int(request.form['receiverSecretKey'])
```

The page will hold till value of secret key of receiver/sender is changed to a value less than the order of the base-point

6.2 We had to change the hash2 function such that it included the whole point instead of one coordinate.

Before:

```
def Hash2(binary_string, point1, point2, point3, prime_q):
    # Custom hash function using basic operations
    hash_value = 0

    # Process binary string
    for char in binary_string:
        hash_value = (hash_value * 31 + ord(char)) % prime_q

    # Process curve points
    for point in [point1, point2, point3]:
        hash_value = (hash_value * 31 + point[0]) % prime_q

    return hash_value
```

After:

```python
def Hash2(binary_string, point1, point2, point3, prime_q):
    # Custom hash function using basic operations
    hash_value = 0

    # Process binary string
    for char in binary_string:
        hash_value = (hash_value * 31 + ord(char)) % prime_q

    # Process curve points
    for point in [point1, point2, point3]:
        hash_value = (hash_value * 31 + point[0]) % prime_q
        hash_value = (hash_value * 31 + point[1]) % prime_q

    return hash_value
```

Hash function changed to include y-coordinate too.

7.USER MANUAL

7.1User Manual Specifications

- Welcome to the SIGNCRYPTION USING ELLIPTIC CURVE CRYPTOGRAPHY application! This tool allows you to perform various cryptographic operations using Elliptic Curve Cryptography (ECC). ECC is a powerful and efficient encryption technique widely used for securing communication and data.

- Open your web browser and navigate to the input page of the application.
    - Enter the parameters of the elliptic curve in the provided form.
    - Enter A: Coefficient A of the curve.

- Enter B: Coefficient B of the curve.
- Enter Q: Prime order Q of the curve.
- Click the "Generate points on the curve" button.
- After submitting the input parameters, you will be redirected to a page displaying the points on the curve.

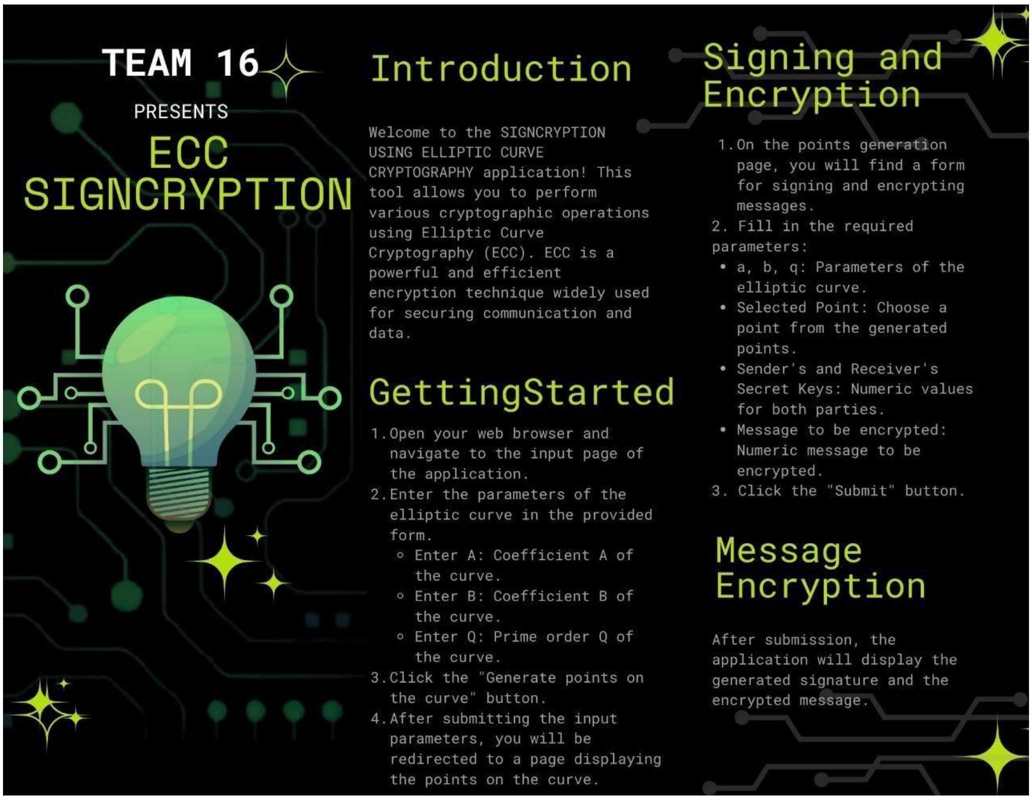
After submission, the application will display the generated signature and the encrypted message.

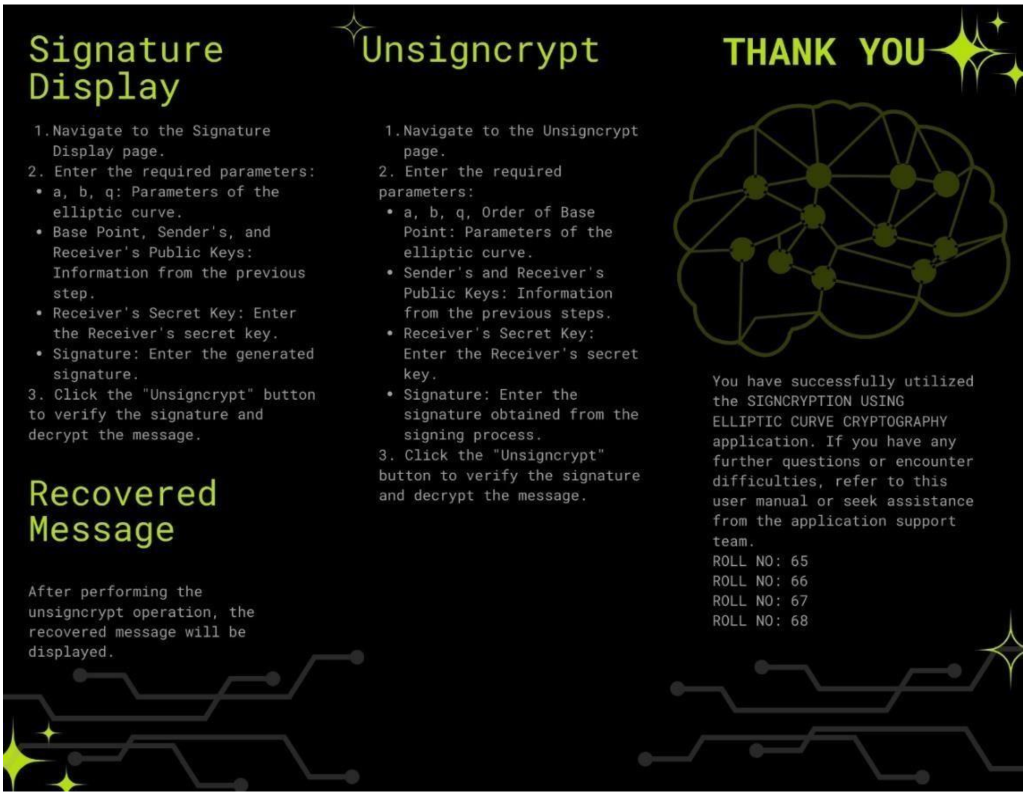
- Navigate to the Signature Display page.
  - Enter the required parameters:
  - a, b, q: Parameters of the elliptic curve.
  - Base Point, Sender's, and Receiver's Public Keys: Information from the previous step.
  - Receiver's Secret Key: Enter the Receiver's secret key.
  - Signature: Enter the generated signature.
  - Click the "Unsigncrypt" button to verify the signature and decrypt the message.

After performing the unsigncrypt operation, the recovered message will be displayed.

- Navigate to the Unsigncrypt page.
  - Enter the required parameters:
  - a, b, q, Order of Base Point: Parameters of the elliptic curve.
  - Sender's and Receiver's Public Keys: Information from the previous steps.
  - Receiver's Secret Key: Enter the Receiver's secret key.
  - Signature: Enter the signature obtained from the signing process.
  - Click the "Unsigncrypt" button to verify the signature and decrypt the message.
- You have successfully utilized the SIGNCRYPTION USING ELLIPTIC CURVE CRYPTOGRAPHY application.

7.2 User Manual Image





8.CONCLUSION

In conclusion, Elliptic Curve Signcryption (ECSC) presents a sophisticated and efficient cryptographic scheme that seamlessly integrates the functionalities of digital signature and encryption, leveraging the mathematical properties of elliptic curve cryptography. By combining these two essential cryptographic operations into a single, streamlined process, ECSC offers a compelling solution for applications where computational resources and bandwidth are at a premium.The security of ECSC relies on the proven hardness of the elliptic curve discrete logarithm problem, ensuring a strong foundation against potential cryptographic attacks. The careful selection and configuration of elliptic curve parameters, hash functions, and combining functions play a crucial role in tailoring ECSC to specific security requirements.

As technological landscapes continue to evolve, ECSC stands as a modern and adaptive cryptographic approach, addressing the challenges posed by constrained environments without compromising on the core principles of information security. Its versatility, efficiency, and robust security make Elliptic Curve Signcryption a valuable tool in securing communications and data in diverse applications, from IoT deployments to secure messaging protocols.

9.REFERENCE(S)

https://www.hindawi.com/journals/wcmc/2022/7499836/

https://ethesis.nitrkl.ac.in/5980/1/E-147.pdf

https://www.youtube.com/watch?v=2aHkqB2-46k&list=PL6N5qY2nvvJE8X75VkXglSrVhLv1tVcfy

https://www.ic.unicamp.br/~rdahab/cursos/mo421-mc889/Welcome_files/Stinson-Paterson_CryptographyTheoryAndPractice-CRC%20Press%20%282019%29.pdf