Report on

Analysis of standard elliptic curves for the implementation of ECC Cipher in resource-constrained environments

Submitted to Manipal University, Jaipur

Towards the partial fulfillment for the Award of the Degree of

BACHELOR OF TECHNOLOGY

In Computers Science and Engineering

2018-2022

By

Name - Jaidev Chaudhary
Registration No- 189301036
&
Name - Saransh Sharma
Registration No- 189301164

Under the guidance of

Dr. Umashankar Rawat [illegible]

Department of Computer Science and Engineering

School of Computing and Information Technology

Manipal University Jaipur

Jaipur, Rajasthan

1. Introduction:

Nowadays, many publicly observable Internet protocols offer cipher suites that contain elliptic curve based cryptographic algorithms. Today, ECC is increasingly used to implement public-key cryptography protocols, such as digital signatures and key agreement protocols. Bitcoin, secure shell (SSH), Transport Layer Security (TLS), and the Austrian e-ID card are some of the application protocols which make use of ECC in the real world. Many E-commerce applications also use elliptic curve cryptography, due to its security features. There are many international standards available for the selection of elliptic curves. The choice of the curve is dependent on the security requirement and the required efficiency of the curve in terms of computation speed. In the first release of cryptographic standards specifying elliptic curves for use in practice is given for the different key sizes. The standards that recommend curves for various security levels are defined over either prime or binary extension fields. Currently, across the internet, the elliptic curves deployed are mostly defined over prime fields. In this paper, such elliptic curves defined over prime fields are the subject of analysis.

In elliptic curve systems, a sub exponential-time algorithm is absent compared to systems based on the multiplicative group of a finite field, so it is possible to maintain the same level of security with an elliptic curve group that is smaller in size. The advantages of using smaller elliptic groups are smaller key sizes, bandwidth savings, and faster implementations which are useful for safety implementation in resource-constrained applications. So, while implementing ECC cryptosystem in the field of E-commerce it is important to find a suitable curve that can work efficiently with resource-constrained devices such as smart cards, personal computer (PC) cards, and wireless devices, etc.

1.1 Motivation:

In the current digital world and public-key cryptography segment, the majority of deployments are fulfilled by the RSA based cryptosystems. Cryptosystems based on elliptic curves emerge as an alternative to the RSA cryptosystems. The security of the RSA cryptosystem is based on the integer factorization problem (IFP) whereas the security of ECC is based on the elliptic curve discrete logarithm problem (ECDLP). The significant attraction towards ECC is that the best-known algorithm for solving the ECDLP takes full exponential time while for solving IFP of RSA takes sub exponential time. ECC takes less memory than RSA and is better than RSA, especially on memory-constrained devices. Moreover, ECC provides same level of security as the RSA but with reduced key size.

2. Literature:

Selection of Elliptic Curve -

All of the different international standards available today focus on the selection of safe and efficient elliptic curves for implementing ECC. Each standard of curve strives to maintain the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP). For the sake of efficiency, each standard has a particular form of the elliptic curve along with a recommendation for the specific values of the prime fields and the constants in the equation. The security of ECC is primarily dependent on the selection of curve. The main idea is to select a curve which is safe against the known attacks on ECDLP. If a weak curve is selected for any application, then its cryptographic security is also reduced.

<u>Analysis of Elliptic Curve –</u>

There are many elliptic curves available for use in ECC suggested by the different standards. The types of curve are basically classified based on prime field size and the form of the curve. Each selected curve has different field sizes which are nothing but the key sizes of ECC. The selected curves are analyzed by performing the two algorithms used in ECC, namely ECDH and ECDSA. The analysis is carried out using Sage Math which is open-source mathematical software using a system having an Intel Core i3 processor with 4 GB of RAM. The same algorithm is used while performing the analysis for each curve. Only the curve parameters are changed according to the type of curve. For each type of curve, the results are taken regarding computation time needed for different operations during ECDH and ECDSA.

Example: Let us consider the elliptic curve y^2 = x^3 + x^2 + x+ 6; p = 11.

Then, set of points of $E_{11}(1, 6)$ are

(2, 4)	(2, 7)	(3, 5)	(3, 6)
(5, 2)	(5, 9)	(7, 2)	(7, 9)
(8, 3)	(8, 8)	(10, 2)	(10, 9)

Let us consider G = (2, 7) and user B's private key $n_{ab} = 7$, then user B's public key

$$P_B = n_{ab} \times G = 7\,(2, 7) = (7, 2)$$

Suppose that user A wishes to send a message to user B that is encoded in the elliptic curve point $P_m = (10, 9)$ (say) and let A select a random number k = 3.

Then cipher text $C_m = \{kg., P_m + ke_{pt\}}$

$$= \{3\,(2, 7), (10,9) + 3(7,2)\}$$

$$= \{(8, 3), (10,9) + (3,5)\}$$

$$= \{(8, 3), (10,2)\}$$

Next User B will recover P_m from C_m as follows:

$$(P_m + ke_{pt}) - n_{ab}\ (kg.) = (10, 2) - 7\ (8,3)$$
$$= (10, 2) - (3,5)$$
$$= (10, 2) + (3,6)$$
$$= (10, 9) = P_m$$

Problem Statement -

Implementation of the Elliptic Curve Diffie-Hellman Algorithm and Elliptic Curve Digital Signature Algorithm.

Approach -

There are multiple standard Elliptical Curves that can be used with the ECC scheme

Each standard of curve strives to maintain the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP).

The main idea is to select a curve which is safe against the known attacks on ECDLP

The analysis is carried out using SageMath, which is a free open-source mathematics software system licensed under the GPL, using Python 3.

Security of ECC –

The security of elliptic curve cryptosystems such as the ECDSA is the apparent intractability of the following elliptic curve discrete logarithm problem (ECDLP):

"Given an elliptic curve *E* defined over, a point $P \quad E\,()$ of order *n*, and a point $Q \quad E\,()$, determine the integer *k*, $0 \leq k \leq n$-1, such that $Q = kip$, provided that such an integer exists".

Over the past many years, the ECDLP has received considerable attention from leading mathematician around the world, and no significant weaknesses have been reported till date. However, in order to achieve the maximum possible security level, *n* should be prime.

Comparison of bit size between RSA and ECC

Table 1.1

ECC-Based Scheme (Size of *n* in bits)	RSA (Modulus size in bits)
112	512
160	1024
224	2048
256	3072
384	7680
512	15360

3. Methodology and Framework:

The analysis will be performed by considering each curve for the implementation of the Elliptic Curve Diffie-Hellman (ECDH) algorithm and the Elliptic Curve Digital Signature Algorithm (ECDSA). The analysis is carried out using Sage Math, which is a free open-source mathematics software system licensed under the GPL, using Python 3.

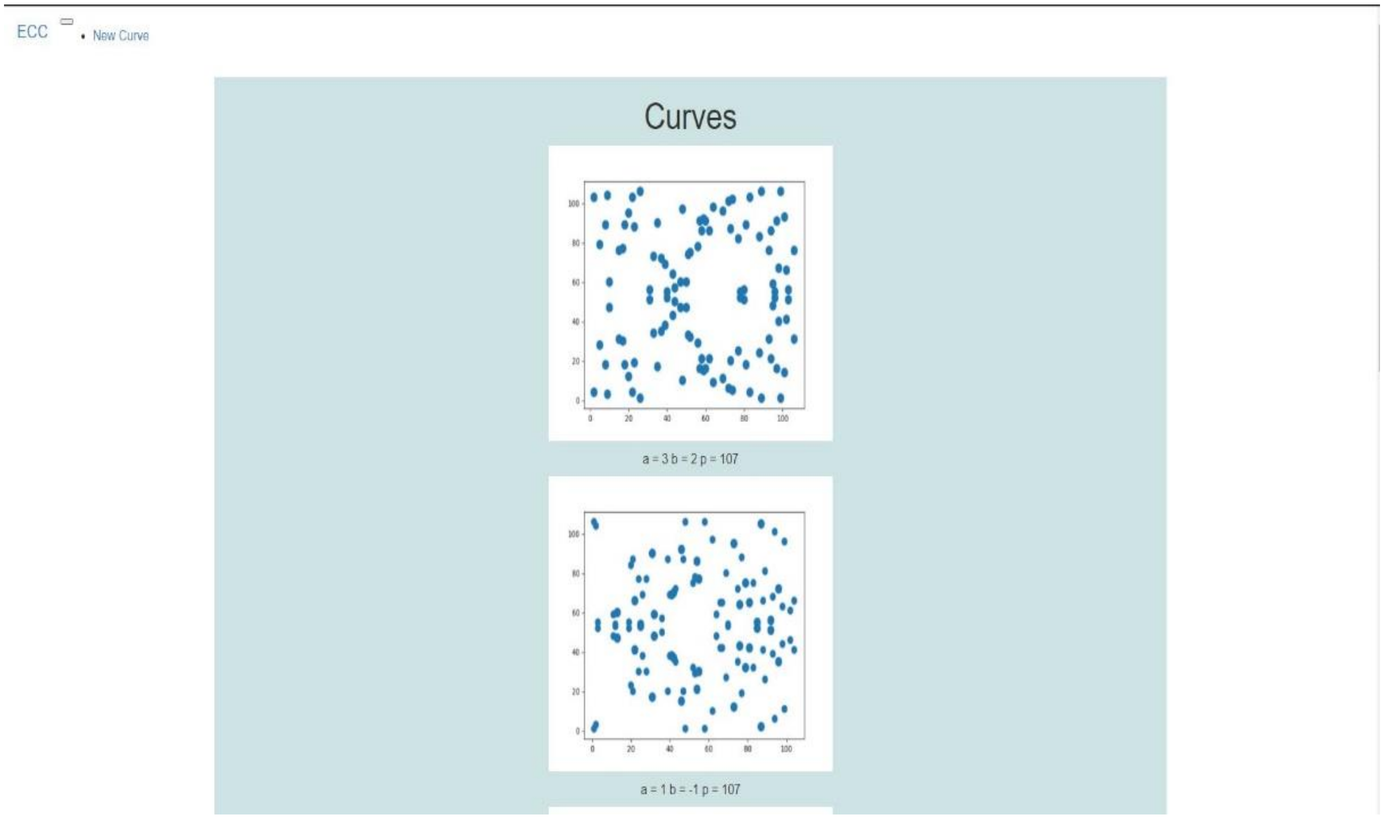
- SOFTWARE
  - Windows 10
  - Python 3.x

- HARDWARE
  - HDD/SSD with sufficient storage capacity
  - Core i5 Processor
  - At least 4GB RAM

4. Work Done:

Homepage-

Figure 1.1



- Understood the mathematical foundations reqd.
- Understood the algorithmic concepts reqd.
- Created a web-app that plots the curves that are pre-defined
- Basic interface ready
- Back-end majorly completed
- Currently gives O/P for predefined parameters

The above bulletins have been implemented using following techniques:

Frontend: Html, CSS, bootstrap

Backend: python3, Flask, SQLAlchemy

Packages: Matplotlib, Seaborn, wtforms

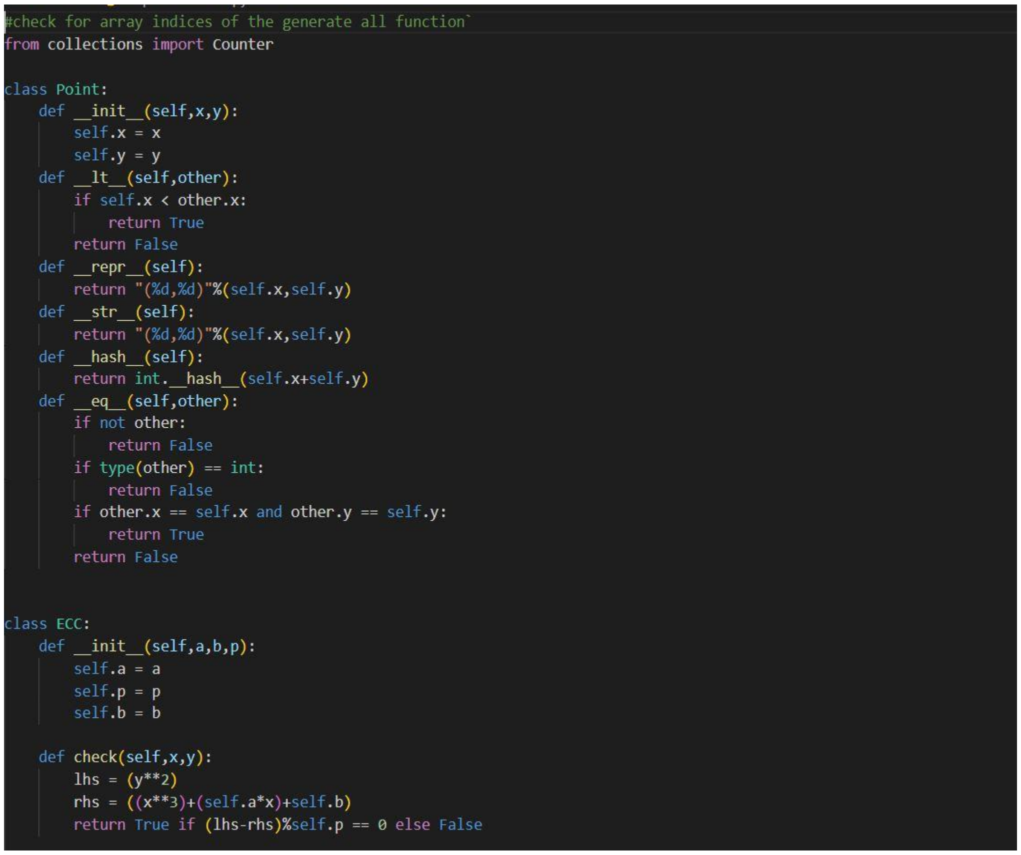
Flask: Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

SQLAlchemy: SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks.

Matplotlib: Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

WTForms: Flask-WTForms is a great tool to help with form validation (e.g., avoidance of CSRF. Flask-WTForms can help create and use web forms with simple Python models, turning tedious and boring form validation into a breeze

<u>Code of implementation of Algorithm:</u>



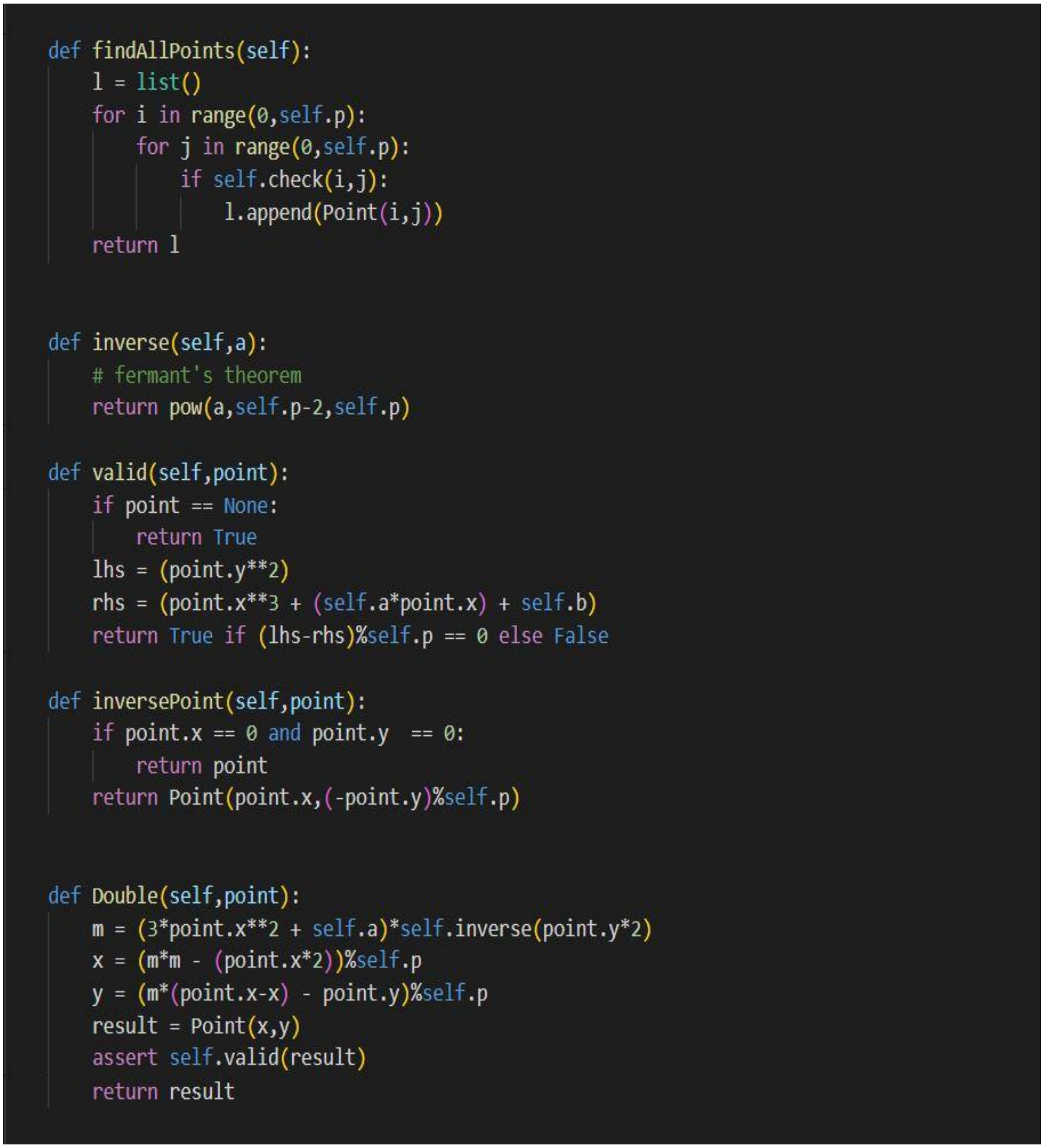
```
#check for array indices of the generate all function`
from collections import Counter

class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def __lt__(self,other):
        if self.x < other.x:
            return True
        return False
    def __repr__(self):
        return "(%d,%d)"%(self.x,self.y)
    def __str__(self):
        return "(%d,%d)"%(self.x,self.y)
    def __hash__(self):
        return int.__hash__(self.x+self.y)
    def __eq__(self,other):
        if not other:
            return False
        if type(other) == int:
            return False
        if other.x == self.x and other.y == self.y:
            return True
        return False


class ECC:
    def __init__(self,a,b,p):
        self.a = a
        self.p = p
        self.b = b

    def check(self,x,y):
        lhs = (y**2)
        rhs = ((x**3)+(self.a*x)+self.b)
        return True if (lhs-rhs)%self.p == 0 else False
```

The below function is continuation of the above function i.e., part of the same class ECC.



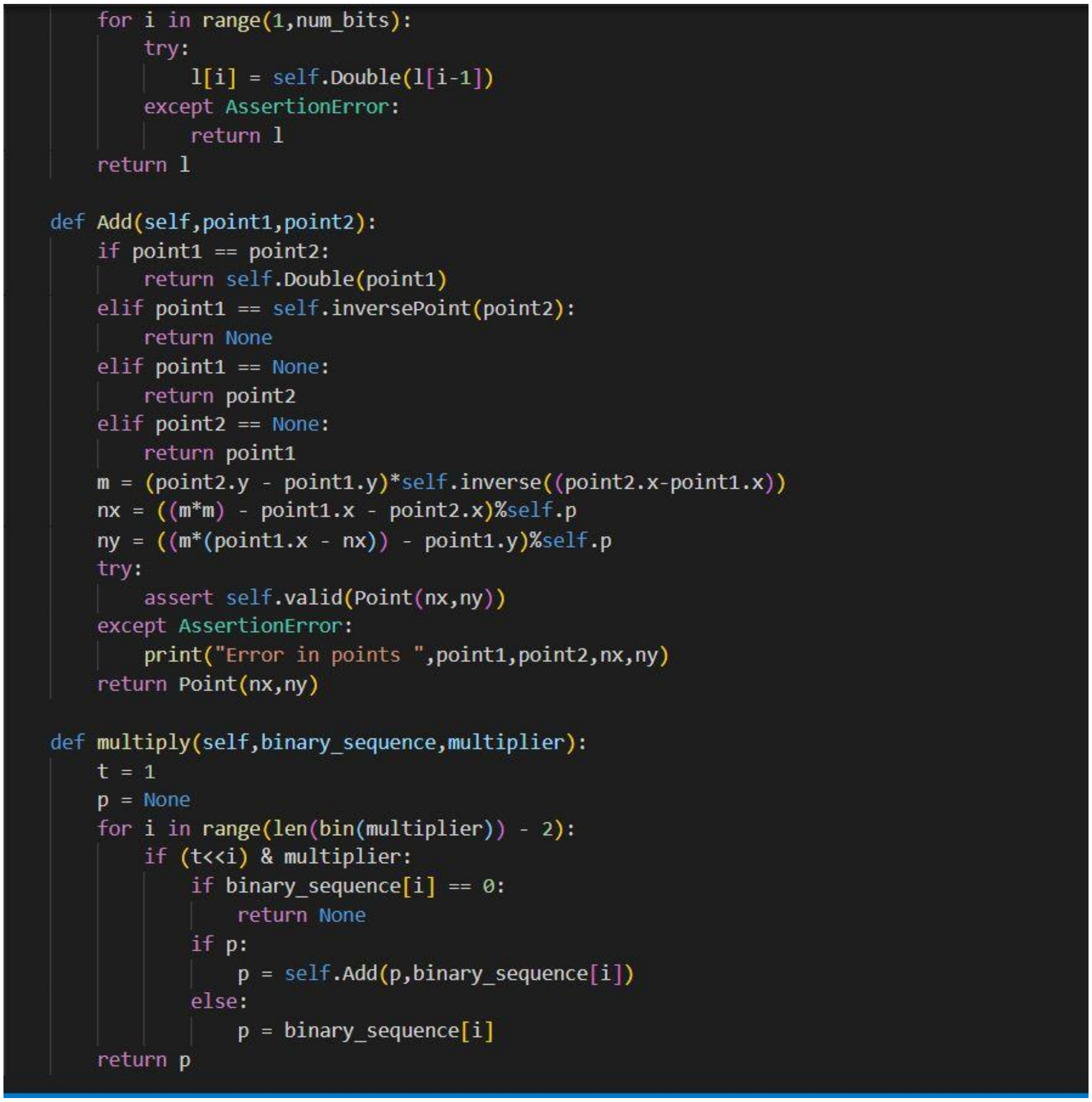
```python
def findAllPoints(self):
    l = list()
    for i in range(0,self.p):
        for j in range(0,self.p):
            if self.check(i,j):
                l.append(Point(i,j))
    return l


def inverse(self,a):
    # fermant's theorem
    return pow(a,self.p-2,self.p)

def valid(self,point):
    if point == None:
        return True
    lhs = (point.y**2)
    rhs = (point.x**3 + (self.a*point.x) + self.b)
    return True if (lhs-rhs)%self.p == 0 else False

def inversePoint(self,point):
    if point.x == 0 and point.y  == 0:
        return point
    return Point(point.x,(-point.y)%self.p)


def Double(self,point):
    m = (3*point.x**2 + self.a)*self.inverse(point.y*2)
    x = (m*m - (point.x*2))%self.p
    y = (m*(point.x-x) - point.y)%self.p
    result = Point(x,y)
    assert self.valid(result)
    return result
```



```
        for i in range(1,num_bits):
            try:
                l[i] = self.Double(l[i-1])
            except AssertionError:
                return l
        return l

    def Add(self,point1,point2):
        if point1 == point2:
            return self.Double(point1)
        elif point1 == self.inversePoint(point2):
            return None
        elif point1 == None:
            return point2
        elif point2 == None:
            return point1
        m = (point2.y - point1.y)*self.inverse((point2.x-point1.x))
        nx = ((m*m) - point1.x - point2.x)%self.p
        ny = ((m*(point1.x - nx)) - point1.y)%self.p
        try:
            assert self.valid(Point(nx,ny))
        except AssertionError:
            print("Error in points ",point1,point2,nx,ny)
        return Point(nx,ny)

    def multiply(self,binary_sequence,multiplier):
        t = 1
        p = None
        for i in range(len(bin(multiplier)) - 2):
            if (t<<i) & multiplier:
                if binary_sequence[i] == 0:
                    return None
                if p:
                    p = self.Add(p,binary_sequence[i])
                else:
                    p = binary_sequence[i]
        return p
```

The entire code has been uploaded on GitHub:

https://github.com/Jaidev810/ECC-Cipher.git

*If you want to run the code:

Step1: install requirement.txt file using code **pip install -r requirement.txt**

Step2: Go in webview and run application.py using command **python application.py**

5. Conclusion and Future Plan:

Till now ECC algorithm has been implemented in python and its implementation has been displayed using GUI. A web app has been created which shows the curves for predefined values. The App has been implemented using various technologies and the visualization has been done using matplotlib.

- We plan to learn sage math and implement in our code.
- Selection of 5 best points from the curve.
- Calculation of average time taken by a point.
- Calculation of average deviation of a point.
- Improve the UI of the app.
- Take user input so that the user can plot his/her curve.
- Fixing bugs on the backend.

References:

- Cryptography and Network Security, 4th Edition, William Stallings
- https://www.researchgate.net/publication/322348730_Analysis_of_standard_elliptic_curves_for_the_implementation_of_elliptic_curve_cryptography_in_resource-constrained_E-commerce_applications
- Flask Documentation: https://flask.palletsprojects.com/en/1.1.x/
- Matplotlib library: https://matplotlib.org/stable/contents.html
- Sqlalchemy: https://docs.sqlalchemy.org/en/14/