****

**JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY**

***CRYPTOGRAPHY ASSIGNMENT***

Submitted By

**KISORIO IAN KIGEN**

**CS282-6397/2012**

**BSC COMPUTER TECHNOLOGY**

**Write ECC code that displays the following curves**

1. y2 = x3 + x + 1

2. 𝑦2=𝑥3−25𝑥

3. 𝑦2=𝑥3+𝑥+6

4. y2 = x3 − 4x

5. y2 = x3 − 1

import matplotlib.pyplot as plt

import numpy as np

def graph\_draw(graph\_formula):

cartesian\_size = 8.0

x\_axis = np.linspace(-cartesian\_size, cartesian\_size, 300)

y\_axis = np.linspace(-cartesian\_size, cartesian\_size, 300)

# the X and Y here take their input from the equation values below

X, Y = np.meshgrid(x\_axis, y\_axis)

f = eval(graph\_formula)

plt.contour(X, Y, f, [0])

plt.title(graph\_formula)

plt.grid()

plt.show()

def graph\_describe():

graph\_draw("X\*\*3 - Y\*\*2 + X + 1")

graph\_draw("X\*\*3 - Y\*\*2 - (25\*X)")

graph\_draw("X\*\*3 - Y\*\*2 + X + 6")

graph\_draw("X\*\*3 - Y\*\*2 - (4\*X)")

graph\_draw("X\*\*3 - Y\*\*2 - 1")

if \_\_name\_\_ == "\_\_main\_\_":

graph\_describe() Page **3** of **5**

**Given** 𝜷=(𝟐,𝟕) 𝒑=𝟏𝟏 **write an ECC program that generates all the points on the curve with** 𝒑=𝟏𝟏 **and that the code should be able to perform point addition and doubling**

class Point(object):

# Construct a point with two given coordindates.

def \_\_init\_\_(self, x, y):

self.x, self.y = x, y

self.inf = False

# Construct the point at infinity.

@classmethod

def atInfinity(cls):

P = cls(0, 0)

P.inf = True

return P

def is\_infinite(self):

return self.inf

# Elliptic Curves over any Field ------------------------------------------------------------------

class Curve(object):

# Set attributes of a general Weierstrass cubic y^2 = x^3 + ax^2 + bx + c over any field.

def \_\_init\_\_(self, a, b, c, char, exp):

self.a, self.b, self.c = a, b, c

self.char, self.exp = char, exp

print(self)

# Elliptic Curves over Prime Order Fields ---------------------------------------------------------

class CurveOverFp(Curve):

# Construct a Weierstrass cubic y^2 = x^3 + ax^2 + bx + c over Fp.

def \_\_init\_\_(self, a, b, c, p):

Curve.\_\_init\_\_(self, a, b, c, p, 1)

def get\_points(self):

# Start with the point at infinity.

points = [Point.atInfinity()]

# Just brute force the rest.

for x in range(self.char):

for y in range(self.char):

P = Point(x, y)

if (y \* y) % self.char == (x \* x \* x + self.a \* x \* x + self.b \* x + self.c) % self.char: Page **4** of **5**

points.append(P)

return points

def invert(self, P):

if P.is\_infinite():

return P

else:

return Point(P.x, -P.y % self.char)

def add(self, P\_1, P\_2):

# Adding points over Fp and can be done in exactly the same way as adding over Q,

# but with of the all arithmetic now happening in Fp.

y\_diff = (P\_2.y - P\_1.y) % self.char

x\_diff = (P\_2.x - P\_1.x) % self.char

if P\_1.is\_infinite():

return P\_2

elif P\_2.is\_infinite():

return P\_1

elif x\_diff == 0 and y\_diff != 0:

return Point.atInfinity()

elif x\_diff == 0 and y\_diff == 0:

if P\_1.y == 0:

return Point.atInfinity()

else:

ld = ((3 \* P\_1.x \* P\_1.x + 2 \* self.a \* P\_1.x + self.b) \* mult\_inv(2 \* P\_1.y, self.char)) % self.char

else:

ld = (y\_diff \* mult\_inv(x\_diff, self.char)) % self.char

nu = (P\_1.y - ld \* P\_1.x) % self.char

x = (ld \* ld - self.a - P\_1.x - P\_2.x) % self.char

y = (-ld \* x - nu) % self.char

return Point(x, y)

# Extended Euclidean algorithm.

def euclid(sml, big):

# When the smaller value is zero, it's done, gcd = b = 0\*sml + 1\*big.

if sml == 0:

return big, 0, 1

else:

# Repeat with sml and the remainder, big%sml.

g, y, x = euclid(big % sml, sml)

# Backtrack through the calculation, rewriting the gcd as we go. From the values just

# returned above, we have gcd = y\*(big%sml) + x\*sml, and rewriting big%sml we obtain

# gcd = y\*(big - (big//sml)\*sml) + x\*sml = (x - (big//sml)\*y)\*sml + y\*big.

return g, x - (big // sml) \* y, y Page **5** of **5**

# Compute the multiplicative inverse mod n of a with 0 < a < n.

def mult\_inv(a, n):

g, x, y = euclid(a, n)

# If gcd(a,n) is not one, then a has no multiplicative inverse.

if g != 1:

raise ValueError('multiplicative inverse does not exist')

# If gcd(a,n) = 1, and gcd(a,n) = x\*a + y\*n, x is the multiplicative inverse of a.

else:

return x % n

a = CurveOverFp(1, 0, 0, 11)

a.add(2, 7)