Question 1

Recall the Dual DL PRNG {reference to Dual DL problem.} There is an actual crypto algorithm, called the Dual EC DL PRNG, where instead of an element in a multiplicative group mod a prime and exponentiation, we consider a point on an elliptic curve over a prime order finite field and scalar multiplication (See NIST SP-800-90.)

We need to define some auxiliary functions:

* *x*(*P*): maps the x-coordinate of an elliptic curve point, *P*, to the integer the smallest positive integer that maps to *x*, mod P.
* LSB*m*(*a*): returns the least significant *m* bits of integer *a*.

And we also denote the following values:

* *p*: a prime, with *n* bits.
* E: an elliptic curve over a finite field with *p* elements, given by equation *y*2 = *x*3 + *ax* + *b*.
* *P*: a point on *E*, with prime order *q* (for maximum security *q* should be roughly the same size as *p*.)
* *Q*: a point in the cyclic subgroup of E generated by *P*.

At the beginning of iteration *i* we have internal state *s*[*i*], and we define the following values:

1. t[i] = s[i]
2. s[i+1] = x(t[i]∙P)
3. r[i] = x(t[i]∙Q)
4. o[i] = LSBn-8 (r[i])

Here *o*[*i*] is the output of the *i*th iteration block, and *s*[*i*+1]

The following diagram shows the flow for generating one block of output with this Crypto Algorithm:

The following problems outline a similar problem with this algorithm as the one described in {reference to Dual DL problem.}

1. Implement a Sage function to generate a single output block from this algorithm (Your function should take an internal state represented as a list with the following elements [E, P, Q, si], where E is a Sage Elliptic Curve object, P is a point on E, with prime order q, and Q is a point on E, generated by Q.
2. Write a Sage function that takes an output of this PRNG (i.e. the x coordinate of a point with the top 8 bits truncated off) and returns the possible values for *R* = *t*[*i*]∙*Q* that could have generated that output [Hint: try the is\_x\_coordinate function on Elliptic Curve objects.]
3. Suppose you have E defined by y2 = x3 + 2x + 4, P = (42, 98095628488211854), Q = (6396452788131036613, 9671497098832291002), and you know that the *P* has order *q* = 1227273995918533091 and also *Q* = 99689∙*P*. Write a sage function that takes an output from one iteration of this function and returns a list of the possible next internal states.
4. Suppose you know that o[i] = 58246156843038996, and o[i+1] = 64511473570997445, use the fact that you have two subsequent outputs to determine the possible internal states that could have generated these two outputs.

Question 1

Recall the Dual DL PRNG {reference to Dual DL problem.} There is an actual crypto algorithm, called the Dual EC DL PRNG, where instead of an element in a multiplicative group mod a prime and exponentiation, we consider a point on an elliptic curve over a prime order finite field and scalar multiplication (See NIST SP-800-90.)

We need to define some auxiliary functions:

* *x*(*P*): maps the x-coordinate of an elliptic curve point, *P*, to the integer the smallest positive integer that maps to *x*, mod P.
* LSB*m*(*a*): returns the least significant *m* bits of integer *a*.

And we also denote the following values:

* *p*: a prime, with *n* bits.
* E: an elliptic curve over a finite field with *p* elements, given by equation *y*2 = *x*3 + *ax* + *b*.
* *P*: a point on *E*, with prime order *q* (for maximum security *q* should be roughly the same size as *p*.)
* *Q*: a point in the cyclic subgroup of E generated by *P*.

At the beginning of iteration *i* we have internal state *s*[*i*], and we define the following values:

1. t[i] = s[i]
2. s[i+1] = x(t[i]∙P)
3. r[i] = x(t[i]∙Q)
4. o[i] = LSBn-8 (r[i])

Here *o*[*i*] is the output of the *i*th iteration block, and *s*[*i*+1]

The following diagram shows the flow for generating one block of output with this Crypto Algorithm:

The following problems outline a similar problem with this algorithm as the one described in {reference to Dual DL problem.}

1. Implement a Sage function to generate a single output block from this algorithm (Your function should take an internal state represented as a list with the following elements [E, P, Q, si], where E is a Sage Elliptic Curve object, P is a point on E, with prime order q, and Q is a point on E, generated by Q.

def DualECDL\_prng\_x\_coord(point):

return point.xy()[0].lift()

def DualECDL\_prng\_truncate(x, m):

return x & (2^m - 1)

def DualECDL\_prng(rng\_state):

E = rng\_state[0]

P = rng\_state[1]

Q = rng\_state[2]

si = rng\_state[3]

n = E.base\_ring().characteristic().nbits()

ti = si

next\_si = DualECDL\_prng\_x\_coord(ti\*P)

ri = DualECDL\_prng\_x\_coord(ti\*Q)

oi = DualECDL\_prng\_truncate(ri, n-8)

rng\_state[3] = next\_si

return oi

1. Write a Sage function that takes an output of this PRNG (i.e. the x coordinate of a point with the top 8 bits truncated off) and returns the possible values for *R* = *t*[*i*]∙*Q* that could have generated that output [Hint: try the is\_x\_coordinate function on Elliptic Curve objects.]

def DualECDL\_reinflate\_points(E, oi):

p = E.base\_ring().characteristic()

n = p.nbits()

m = n-8

potential\_points = []

Epoly = E.defining\_polynomial()

for j in xrange(2^8):

x\_coord = j\*2^m + oi

if (x\_coord > p):

break

Ept = Epoly.subs(x=x\_coord, z=1).univariate\_polynomial()

if (not Ept.is\_irreducible()):

y\_coord = Ept.roots()[0][0]

P1 = E((x\_coord, y\_coord))

P2 = E((x\_coord, -y\_coord))

potential\_points.append(P1)

potential\_points.append(P2)

return potential\_points

1. Suppose you have E defined by y2 = x3 + 2x + 4, P = (42, 98095628488211854), Q = (6396452788131036613, 9671497098832291002), and you know that the *P* has order *q* = 1227273995918533091 and also *Q* = 99689∙*P*. Write a sage function that takes an output from one iteration of this function and returns a list of the possible next internal states.

def DualECDL\_predict\_next\_state(E, trapdoor, oi):

points = DualECDL\_reinflate\_points(E, oi)

states = []

for j in xrange(len(points)):

next\_si = DualECDL\_prng\_x\_coord(trapdoor\*points[j])

if not next\_si in states:

states.append(next\_si)

return states

1. Suppose you know that o[i] = 58246156843038996, and o[i+1] = 64511473570997445, use the fact that you have two subsequent outputs to determine the possible internal states that could have generated these two outputs.  
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
   This Sage output that shows the output of this function is:  
     
     
     
     
     
     
     
     
     
     
     
     
     
   So we have narrowed down the potential internal states to just one.

sage: p = 14727287954581601537

sage: F = GF(p)

sage: E = EllipticCurve(F, [0,0,0,2,4]); E.order().factor()

2^2 \* 3 \* 1227273995918533091

sage: q = 1227273995918533091

sage: P = E((42, 98095628488211854))

sage: b = 99689

sage: Q = E((6396452788131036613, 9671497098832291002))

sage: trapdoor = xgcd(b, q

sage: o1 = 2785191591765556

sage: o2 = 8741267312563186

sage: DualECDL\_narrow\_state\_guess(E, P, Q, trapdoor, o1, o2)

[26148098850266219]

def DualECDL\_narrow\_state\_guess(E, P, Q, trapdoor, o1, o2):

states1 = DualECDL\_predict\_next\_state(E, trapdoor, o1)

states2 = []

rng\_state = [E, P, Q, 0]

for j in xrange(len(states1)):

rng\_state[3] = states1[j]

possible\_o2 = DualECDL\_prng(rng\_state)

if (possible\_o2 == o2):

states2.append(rng\_state[3])

return states2