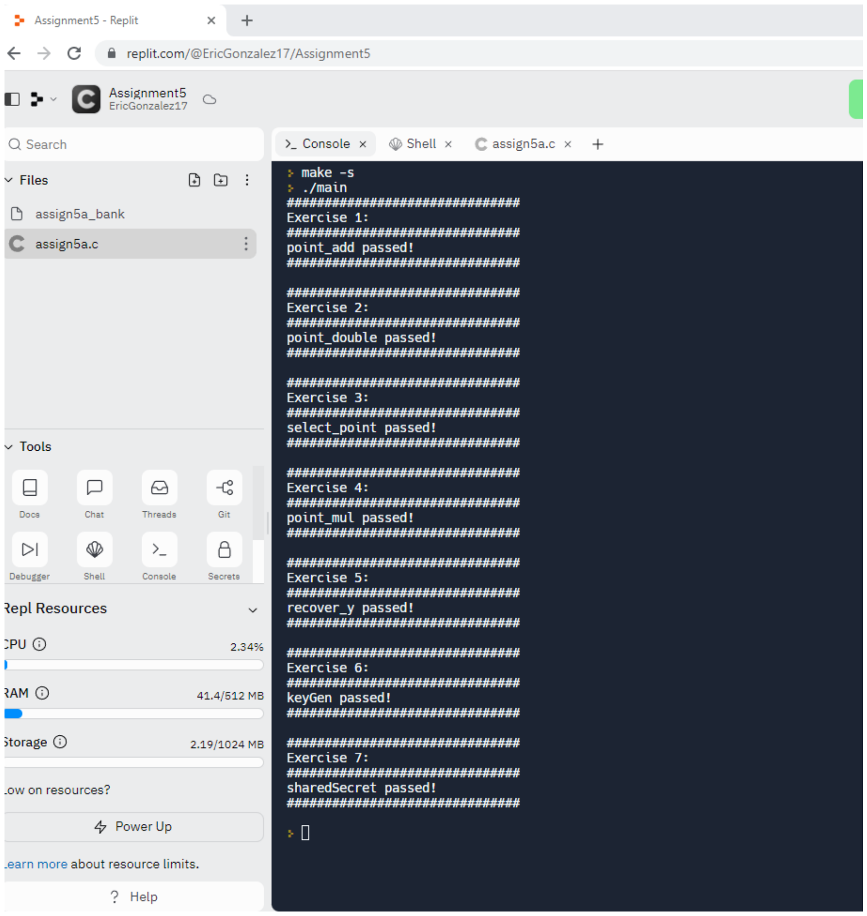
Eric Gonzalez
Z23411215
CDA 5326: Cryptographic Engineering
Assignment 5

Assignment A



```
› make -s
› ./main
###############################
Exercise 1:
###############################
point_add passed!
###############################

###############################
Exercise 2:
###############################
point_double passed!
###############################

###############################
Exercise 3:
###############################
select_point passed!
###############################

###############################
Exercise 4:
###############################
point_mul passed!
###############################

###############################
Exercise 5:
###############################
recover_y passed!
###############################

###############################
Exercise 6:
###############################
keyGen passed!
###############################

###############################
Exercise 7:
###############################
sharedSecret passed!
###############################

›
```

Assignment B:

1.)
a.)
sk = 6c3a9f1d7b84cd5a7661b2c38cc1c9e4cd413fbc9eb788dc97531a171727e690
pk = 5fa300ad515a4f8c5ebbe53ff1a75ac646f869d9a990a517f813df7df217ec82

b.)
ss= 2407d87db971b0908541612888d74110a446276dbe89ed1d2e7900eb32b3a70a
c.)
What is the approximate security level? Is it secure?

- **Our security is ECC with size approximately 256 which gives a security in bits of 112 to 128 bits. This is a very secure implementation and a standard used worldwide.**

d.)
Can Alice and Bob detect whether or not the messages have been modified by Eve?

- **No, Alice and Bob will not be able to tell that their communications have been tampered with and this is because they do not know their public keys ahead of time and their is no hashing or third party verification method in fact the third party is actually sabatuer using a man-in-the-middle attack for this reason. The encryption standard is too difficult to be broken, so they go for an unsecured traffic stream.**

e.)
Can Alice and Bob detect that the messages have not been read or modified by a 3rd party (Eve)? If yes, show how. If not, show why

- **In this instance it is possible for Alice and Bob to tell that the messages have not been read or modified. With knowledge of the public keys and the ability to see that the public keys have been modified the two parties will be able to see that the two shared secrets they have are not equal if the messages have been altered.**

f.) Two benefits of ECDH over DH:

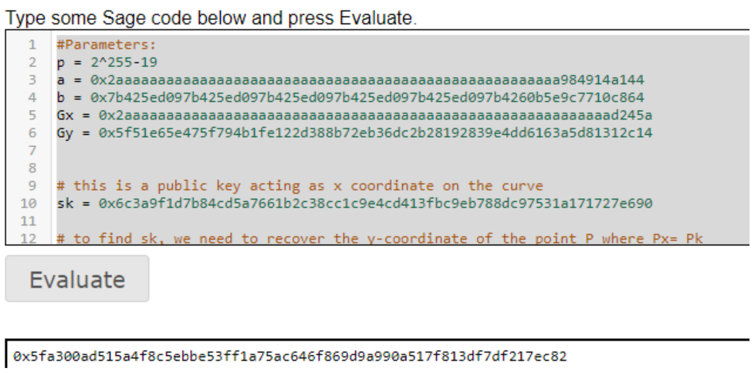
- **One is the amount of bits of security is greater for smaller key size**
- **Because you are using smaller key sizes the computational requirements are lessened and the applications that can be protected are greater**

2.)

a.)

```
p = 2^255-19
a = 0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa984914a144
b = 0x7b425ed097b425ed097b425ed097b425ed097b425ed097b4260b5e9c7710c864
Gx = 0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad245a
Gy = 0x5f51e65e475f794b1fe122d388b72eb36dc2b28192839e4dd6163a5d81312c14


# this is a public key acting as x coordinate on the curve
sk = 0x6c3a9f1d7b84cd5a7661b2c38cc1c9e4cd413fbc9eb788dc97531a171727e690

# to find sk, we need to recover the y-coordinate of the point P where Px= Pk


#finite field fp
U=GF(p)


# Finite field Fp
# Create an Elliptic Curve over Fp with equation: y2 = x3 + ax + b
E = EllipticCurve(GF(p),[a,b])
# Point at infinity.
# Note when printing point at infinity, sage will show (0:1:0)
#O = E(0)
G=E(Gx,Gy)

#Generate a secret key

pk=(sk*G)[0]

print(hex(pk))


def mod_sqrt(s,p):
   e_const=0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffd
   u=2*(s%p)
   v=pow(u,e_const,p)
   w=(u*v^2)%p
   y=(s*v*(w-1))%p
   return y
```

Output:

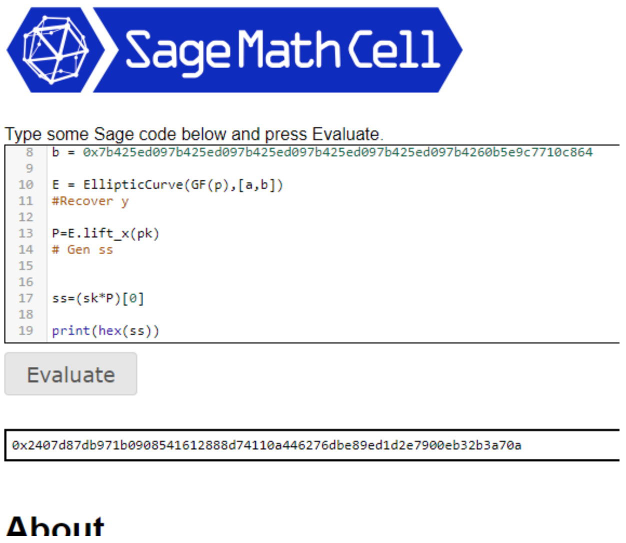


Type some Sage code below and press Evaluate.

```
#Parameters:
p = 2^255-19
a = 0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa984914a144
b = 0x7b425ed097b425ed097b425ed097b425ed097b425ed097b4260b5e9c7710c864
Gx = 0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad245a
Gy = 0x5f51e65e475f794b1fe122d388b72eb36dc2b28192839e4dd6163a5d81312c14


# this is a public key acting as x coordinate on the curve
sk = 0x6c3a9f1d7b84cd5a7661b2c38cc1c9e4cd413fbc9eb788dc97531a171727e690

# to find sk, we need to recover the y-coordinate of the point P where Px= Pk
```

```
0x5fa300ad515a4f8c5ebbe53ff1a75ac646f869d9a990a517f813df7df217ec82
```

```
#Part B
#2B.)
sk = 0x6c3a9f1d7b84cd5a7661b2c38cc1c9e4cd413fbc9eb788dc97531a171727e690
pk = 0x70a4b2148fdd56d85cbaf80a012f312787bd9b13bcd69d0be2927f6d843bf57c
p = 2^255-19
a = 0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa984914a144
b = 0x7b425ed097b425ed097b425ed097b425ed097b425ed097b4260b5e9c7710c864

E = EllipticCurve(GF(p),[a,b])
#Recover y

P=E.lift_x(pk)
# Gen ss


ss=(sk*P)[0]

print(ss)
```



Type some Sage code below and press Evaluate.

```
b = 0x7b425ed097b425ed097b425ed097b425ed097b425ed097b4260b5e9c7710c864

E = EllipticCurve(GF(p),[a,b])
#Recover y

P=E.lift_x(pk)
# Gen ss


ss=(sk*P)[0]

print(hex(ss))
```

```
0x2407d87db971b0908541612888d74110a446276dbe89ed1d2e7900eb32b3a70a
```