Intro. To FinTech Homework 3 Report

B09901142 EE4 呂睿超

tags: `2023 Fall Intro. To Fintech`

Q1 - Q3

- Initialize the curve and base point

```
# Define the elliptic curve for secp256k1
E = EllipticCurve(GF(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
G = E.gen(0)  # Base point
```

- Evaluate the points using

```
# Define the elliptic curve for secp256k1
result_4G = 4*G
result_5G = 5*G
Q = d*G
```

Q4

- Follow the algorithm, double at every bit, add if bit = 1
  - I implmented two ways to check the counts, but the logic is the same as the above.

```
# d = 1142
binary = bin(d)
binary = binary[2:]
double = 0
add = 0

for bit in binary[1:]:
    double += 1
    if bit == '1':
        add += 1
```

- (double, add) = (10,5)

Q5

- After trial and error, I found the best decomposition of d = 1142 = 2^{10} + 2^7 - 2^3 - 2^1
- Following the similar logic as standard double and add but apply the subtraction when needed

```
for power in range(9,-1,-1):
    current_point2 = 2*current_point2
    double += 1
    if power in [3,1]:  # subtract G at these powers
        current_point2 = current_point2 - G
        add += 1
    elif power == 7:
        current_point2 = current_point2 + G
        add += 1
```

- (double, add) = (10,3)

Q6

- I just followed the standard step of signing a bitcoin transaction

```
# Step 0 : Initialize
d_a = d
n = E.order()
m = "Sample Bitcoin Transaction Signing"

# Step 1 : Calculating hashed message
e = int(hashlib.sha256(m.encode()).hexdigest(), 16)

# Step 2 : find z, k
Ln = len(bin(n)[2:])
z = int(str(e)[:Ln])
random.seed(int(40))
k = random.randint(1,n-1)

# Step 3 : Calculate x1 -> r
(x1,y1) = (k*G).xy()
r = lift(x1) % n

# Step 4 : Calculate r
s = ((z + r*d) * inverse_mod(k, n)) % n
```

- Note

  - I fixed the random seed just for reproducible results
  - I used inverse_mod() for k^{-1} mod n
  - I used lift() for modulum of coordinates of curves

Q7

- Similarly, I just followed the standard step of the verification of a bitcoin transaction

```
# Step 1 : Verify if the signature is valid
assert((1 <= r and r <= n-1) and (1 <= s and s <= n-1))

e_ver = int(hashlib.sha256(m.encode()).hexdigest(), 16)

# Step 2 : Find z, w -> u1, u2
Ln = len(bin(n)[2:])
z = int(str(e_ver)[:Ln])

w = inverse_mod(s, n) % n
u1 = (z*w) % n
u2 = (r*w) % n

# Step 3 : Find x2 -> r2
(x2,y2) = (u1*G + u2 * Q).xy()
r2 = lift(x2) % n

# Step 4 : Verify r2 with x1
assert(r2 == x1)
print("Verified")
```

- Note
  - I used assert() for checking the validity
  - Similar with Q6, I explained each step in the comment of the code

Q8

- Simply use the PolynomialRing function provided by Sage

```
R = PolynomialRing(Zmod(10007), 'x')
x = R.gen()
p = R.lagrange_polynomial([(1, 10), (2, 20), (3, 1142)])
```

- p = $556x^2 + 8349x + 1112$