Homework #6

R08922037 郭逸琳

d = 922037

Problems

1. Calculate 4G (sage)

In [1]:
```
# set secp256k1 parameters
a = 0
b = 7
p =  2^256 - 2^32 - 977
```
0.014 seconds 1

In [2]:
```
# construct ECC curve
EC = EllipticCurve(GF(p), [a, b])

# base point of the standard
baseX = 55066263022277343669578718895168534326250603453777594175500187360389116729240
baseY = 32670510020758816978083085130507043184471273380659243275938904335757337482424
```
0.022 seconds 2

In [3]:
```
# construct the base point
G = EC(baseX, baseY)
```
0.011 seconds 3

In [4]:
```
## problem 1: 4G
Q = 4 * G
x = Q.xy()[0]
y = Q.xy()[1]
print 'x = ', x
print 'y = ', y
```
0.016 seconds 4

```
x =  103388573995635080359749164254216598308788835304023601477803095234286494993683
y =  37057141145242123013015316630864329550140216928701153669873286428255828810018
```

$$x = 103388573995635080359749164254216598308788835304023601477803095234286494993683$$
$$y = 37057141145242123013015316630864329550140216928701153669873286428255828810018$$

2. Calculate 5G (sage)

In [5]:
```
## problem 2: 5G
Q = 5 * G
x = Q.xy()[0]
y = Q.xy()[1]
print 'x = ', x
print 'y = ', y
```
0.008 seconds 5

```
x =  21505829891763648114329055987619236494102133314575206970830385799158076338148
y =  98003708678762621233683240503080860129026887322874138805529884920309963580118
```

$$x = 21505829891763648114329055987619236494102133314575206970830385799158076338148$$
$$y = 98003708678762621233683240503080860129026887322874138805529884920309963580118$$

3. Calculate dG (sage)

In [6]:
```
## problem 3: dG
d = 922037
Q = d * G
x = Q.xy()[0]
y = Q.xy()[1]
print 'x = ', x
print 'y = ', y
```
0.008 seconds 6

```
x =  97082309625624035349452542765333639318461486547441372200301262429618469233251
y =  39859575488738210964582896344700538325387536495084355221978637921499904842492
```

$$x = 97082309625624035349452542765333639318461486547441372200301262429618469233251$$
$$y = 39859575488738210964582896344700538325387536495084355221978637921499904842492$$

4. Evaluate dG (python 3.7)

In [1]:
```
d = 922037
binarystr = bin(d)[2:]
strLen = len(binarystr)

num = 0
action = []
```

In [2]:
```
for i in range(strLen):
    if i == 0 and binarystr[i] == '1':
        num = 1

        action.append('init')
    elif binarystr[i] == '1':
        num = num * 2
        num = num + 1

        action.append('d')
        action.append('a')
    elif binarystr[i] == '0':
        num = num * 2

        action.append('d')
```

In [3]:
```
#print(num)
print('double:', action.count('d'))
print('add:', action.count('a'))
#print(action)
```

```
double: 19
add: 9
```

Strategy:

- scan from left to right
- (1) if meet 1: do double and add
- (2) if meet 0: do double

Result:

- double: 19 times
- add: 9 times
- total: 28 times

5. Evaluate dG (python 3.7)

In [1]:
```
d = 922037
binarystr = bin(d)[2:]
strLen = len(binarystr)

num = 0
cnt = 0
action = []
```

Strategy:

- scan from left to right
- (1) if see 1 which len ≤ 2 : do double and add
- (2) if see 1 which len > 2 : do add, double len times, subtract
- (–) if 1's len > 2, do (2) because the instruction count will be less than do (1)
- (3) if see 0 : do double

In [2]:
```
for i in range(strLen):
    if i == 0 and binarystr[i] == '1':
        num = 1

        action.append('init')
    elif binarystr[i] == '0':
        num = num * 2

        action.append('d')
    elif binarystr[i] == '1':
        cnt = cnt + 1
        if i == strLen - 1 or binarystr[i+1] == '0':
            if cnt > 2:
                # the faster way
                num = num + 1
                num = num * pow(2, cnt)
                num = num - 1

                action.append('a')
                for j in range(cnt):
                    action.append('d')
                action.append('s')

                cnt = 0
            elif cnt == 2 or cnt == 1:
                # the original way (problem 4)
                for j in range(cnt):
                    num = num * 2
                    num = num + 1

                    action.append('d')
                    action.append('a')

                cnt = 0
```

In [3]:
```
#print(num)
print('double:', action.count('d'))
print('add:', action.count('a'))
print('subtract:', action.count('s'))
#print(action)
```

```
double: 19
add: 9
subtract: 0
```

Result:

- double: 19 times
- add: 9 times
- subtract: 0 times
- total: 28 times
- it's same with problem 4 because of my student ID
- other cases (student ID) will have a better result :D

6. Sign the transaction (sage)

In [1]:
```
## problem 6: sign a Bitcoin transaction
# set secp256k1 parameters
a = 0
b = 7
p = 2^256 - 2^32 - 977

# construct ECC curve
EC = EllipticCurve(GF(p), [a, b])

# base point
baseX = 55066263022277343669578718895168534326250603453777594175500187360389116729240
baseY = 32670510020758816978083085130507043184471273380659243275938904335757337482424
G = EC(baseX, baseY)

# group order
order = G.order()

# key pair: (private key, public key)
d = 922037
Q = d * G

# message
txHash = 0xd07ff0c512344cac5331d1e4c69826d7488267c3cc484459fec79f1e94dc45f4

# sign
k = 2
P = k * G
r = int(P.xy()[0]) % order
s = (inverse_mod(k, order) * (txHash + d * r)) % order

print 'r = ', r
print 's = ', s
```

0.177 seconds 1

```
r =  89565891926547004231252920425935692360644145829622209833684329913297188986597
s =  63382629143472104136338554160827449121225131901955806841843994872503071416790
```

$$r = 89565891926547004231252920425935692360644145829622209833684329913297188986597$$

$$s = 63382629143472104136338554160827449121225131901955806841843994872503071416790$$

- randomly choose k = 2 because it has modular inverse
- <transaction link (click here :D)>

7. Verify the signature (sage)

In [2]:
```
## problem 7: verify the digital signature
w = inverse_mod(s, order) % order
u1 = (txHash * w) % order
u2 = (r * w) % order
L = (u1 * G) + (u2 * Q)

#print 'r = ', r
#print 'L.x = ', L.xy()[0]

if r == L.xy()[0]:
    print 'Valid'
else:
    print 'Invalid'
```

0.052 seconds 2

```
Valid
```

Verified success!