Fintech Homework3

Author: alanhc(曾宏鈞)

ID: r10944007

Date: 11/25

env

- python=3.9
- sagemath

Use the elliptic curve “secp256k1” as Bitcoin and Ethereum. Let G be the base point in the standard. Let d be the last 4 digits of your student ID number.

In [10]:
```
"""
參考上課ppt，使用uncompressed form建構橢圓曲線 (y^2 = x^3 + 7)
"""
d = 4007
## prime field size
p = 2^256-2^32-2^9-2^8-2^7-2^6-2^4-1
## 橢圓曲線係數
a = 0
b = 7

EC = EllipticCurve(GF(p), [a,b])
print(EC)
## 定義base point
GX = 0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798  #前128位元
GY = 0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8  #後128位元
G = EC(GX, GY)
```

```
Elliptic Curve defined by y^2 = x^3 + 7 over Finite Field of size 115792089237316195423570985008687907853269984665640564039457584007908834671663
```

1. Evaluate 4G.

In [11]:
```
print("1. 4G:\n", 4*G)
```

```
1. 4G:
 (103388573995635080359749164254216598308788835304023601477803095234286494993683 : 37057141145242123013015316630864329550140216928701153669873286428255828810018 : 1)
```

2. Evaluate 5G.

In [12]:
```
print("2. 5G:\n", 5*G)
```

```
2. 5G:
 (21505829891763648114329055987619236494102133314575206970830385799158076338148 : 98003708678762621233683240503080860129026887322874138805529884920309963580118 : 1)
```

3. Evaluate Q = dG, d=944007

In [13]:
```python
Q = d*G
print("3. dG:\n", Q)
```

```
3. dG:
 (110687230094785624329639812950862542631312783457675585801236088706412340814O7 : 29479005064789489990333750295528249572806001646454418366766781270476299137056 : 1)
```

4. With standard Double-and Add algorithm for scalar multiplications, how many doubles and additions respectively are required to evaluate dG?

In [14]:
```python
print(d, "binary:", bin(d)[2:])
_double = 0
_add = 0
# 從左到右，第一位不看
# 遇到1 double & add
# 遇到0 double
for i in str(bin(d))[3:]:
    if (i=='1'):
        _double+=1
        _add+=1
    else:
        _double+=1
print("double:", _double)
print("add:", _add)
```

```
4007 binary: 111110100111
double: 11
add: 8
```

5. Note that it is effortless to find −P from any P on a curve. If the addition of an inverse point is allowed, try your best to evaluate dG as fast as possible. Hint: 31P = 2(2(2(2(2P)))) −P.

In [15]:
```python
"""
根據Hint:
(原本)
31 = (11111)
這樣會做 4(double)+4(add)
但若是化簡成32-1，會變成
5(double) - 1
因為減法比加法快 (直接算-P)
所以演算法為
1. 找到大於n的最大2的次方 2^max_n
2. 2^max_n - n  = remain
3. 使用remain找小於remain的2次方相減，直到remain = 0
"""
myID_bin = str(bin(d))[2:]
max_digit = len(myID_bin)

diff = 1<<max_digit
diff -= d
diff_b = str(bin(diff))[2:]
ans_sub = []
i=len(diff_b)-1
for c in diff_b:
    if (c=="1"):
        ans_sub.append(i)
    i-=1

print("double:", max_digit)
print("substract:", len(ans_sub))
print("2^%s - 2^%s"%(max_digit,ans_sub))
ans = 0
for i in ans_sub:
    ans+=2^i
2^20 - ans
```

```
double: 12
substract: 4
2^12 - 2^[6, 4, 3, 0]
```

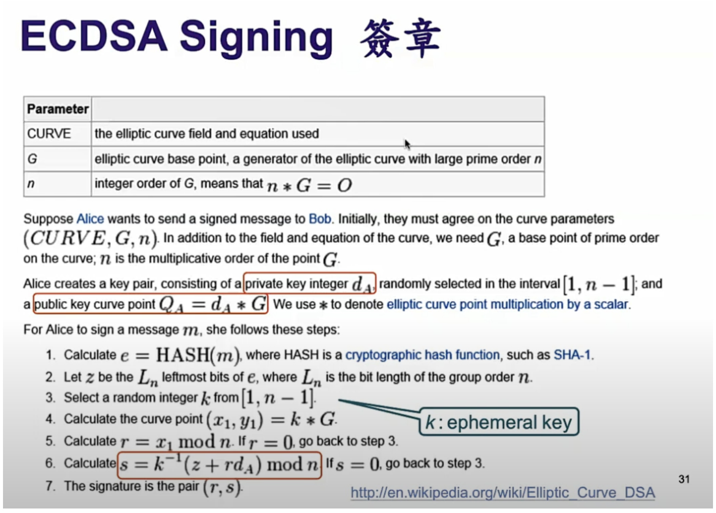
Out[15]:
```
1048487
```

Take a [Bitcoin transaction (https://www.blockchain.com/btc/tx/2b923c531fb2bb07bebdd160867c61ffce3a355988b17eae068c as you wish.

Details

Hash	2b923c531fb2bb07bebdd160867c61ffce3a355988b17eae068cdf4b9f5eac6f
Status	Confirmed
Received Time	2021-11-26 10:10
Size	352 bytes
Weight	1,081
Included in Block	711326

6. Sign the transaction with a random number k and your private key d



In [16]:

```
import hashlib
from sage.rings.finite_rings.integer_mod import IntegerMod

n = G.order()
m = b"R10944007"
# step 1
my_e = hashlib.sha256(m).hexdigest()
e = 0x2b923c531fb2bb07bebdd160867c61ffce3a355988b17eae068cdf4b9f5eac6f  #上面截圖的hash
# step 2 找最左邊Ln個bit
Ln = 44
z = bin(e)[2:2+Ln]
# step 3
while(True):
    k = ZZ.random_element(n)
    # step 4
    x1, y1, _ = k*G
    # step 5 (r = x1 mod n)
    r = IntegerMod(GF(n), x1)
    # step 6
    k_inver = pow(k, -1, n)
    s = IntegerMod(GF(n), k_inver * (int(z, 2)+r*d))
    if r!=0 and s!=0:
        print("result: (r,s)=(\n%s,\n%s\n)"%(hex(r), hex(s)))
        break
```

```
result: (r,s)=(
0xaf5d6d8c60a9d1798328955384995fadc6acc2a52d57d128e50fb5b2e4925dc6,
0x36275afae831f16ab64d7e09c640fb4f887164428c220e7ea581eefe6fdefe627
)
```

ECDSA Verification 驗章

For Bob to authenticate Alice's signature, he must have a copy of her public-key curve point Q_A. Bob can verify Q_A is a valid curve point as follows:

1. Check that Q_A is not equal to the identity element O, and its coordinates are otherwise valid
2. Check that Q_A lies on the curve
3. Check that $n * Q_A = O$

After that, Bob follows these steps:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \bmod n$.
5. Calculate $u_1 = zw \bmod n$ and $u_2 = rw \bmod n$.
6. Calculate the curve point $(x_1, y_1) = u_1 * G + u_2 * Q_A$.
7. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.

Note that using Straus's algorithm (also known as Shamir's trick) a sum of two scalar multiplications $u_1 * G + u_2 * Q_A$ can be calculated faster than with two scalar multiplications.[3]

32

http://en.wikipedia.org/wiki/Elliptic_Curve_DSA

7. Verify the digital signature with your public key Q.

In [17]:

```
# step 1
if (r<1 or r>n-1 or
    s<1 or s>n-1 ):
    print("error")
# step 2
e = 0x2b923c531fb2bb07bebdd160867c61ffce3a355988b17eae068cdf4b9f5eac6f
# step 3
Ln = 44
z = bin(e)[2:2+Ln]
# step 4
w = pow(s, -1, n) # 計算乘法反元素s^-1 mod n
# step 5
u1 = int(z,2)*w % n #IntegerMod(GF(n), int(z,2)*w)
u2 = r*w % n #IntegerMod(GF(n), r*w)
# step 6
x1, x2, _ = int(u1)*G+int(u2)*Q
# step 7
if (r == IntegerMod(GF(n), x1)):
    print("succeed!")
else:
    print("faild...")
```

```
succeed!
```

Lagrange Interpolation

- Problem: Construct a quadratic polynomial $p(x)$ with

$$p(1) = 5,\ p(2) = 9,\ \text{and } p(3) = 7.$$

- Solution: $p(x)$

$$= 5 \cdot \frac{(x-2)(x-3)}{(1-2)(1-3)} + 9 \cdot \frac{(x-1)(x-3)}{(2-1)(2-3)} + 7 \cdot \frac{(x-1)(x-2)}{(3-1)(3-2)}$$

Lagrange Interpolation

- Lagrange Interpolation Formula

$$p(x) = \sum_{i=0}^{k} p_i(x) = \sum_{i=0}^{k} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

is the unique polynomial of degree $\le k$ passing through the $k+1$ points (x_i, y_i), where $x_i \neq x_j$ for $i \neq j$

8. Over Z10007, construct the quadratic polynomial p(x) with p(1) = 10, p(2) = 100, and p(3) = 944007.

In [18]:
```
points = [(1,10), (2,100), (3,d)]
print(points)
F = GF(10007) # 有限體
R = F['x']
R.lagrange_polynomial(points) # 用 sage內建 lagrange_polynomial 解
```

```
[(1, 10), (2, 100), (3, 4007)]
```

Out[18]:
```
6912*x^2 + 9375*x + 3737
```

ref

- https://en.bitcoin.it/wiki/Secp256k1 (https://en.bitcoin.it/wiki/Secp256k1)
- https://ask.sagemath.org/question/39732/lagrange-interpolation-over-a-finite-field/ (https://ask.sagemath.org/question/39732/lagrange-interpolation-over-a-finite-field/)