Final Term Project

Subject: Information Security

Professor: Junbeom Hur

Name : Hojoon Lee

ID : 2014190701

Submit data : 2018.12.21

Utils.py

```
def extended_euclidean_algorithm(b, m):
    a1, a2, a3 = 1, 0, m
    b1, b2, b3 = 0, 1, b

    while(1):
        #no inverse
        if b3 == 0:
            return 0
        elif b3 == 1:
            return (b2 + m) % m
        else:
            q = a3 // b3
            t1, t2, t3 = a1 - q * b1, a2 - q * b2, a3 - q *b3
            a1, a2, a3 = b1, b2, b3
            b1, b2, b3 = t1, t2, t3


def fast_power(base, power, modulus):
    r = 1
    while power:
        power, d = power // 2, power % 2
        if d:
            r = r * base % modulus
        base = base * base % modulus
    return r
```

In order to efficiently solve the problem, I made extra library in order to utilize in the problem solving.

One is the **extended Euclidean algorithm**, which is exactly same as the pseudo-code in the chapter 4.

The other is the **exponentiation under the modulus**. In order to perform fast exponentiation, I used the square and multiply algorithm.

Problem1

1. Approach to the problem

i) Find Xa'

She reuses x_A to compute a new secret key $x_A' \equiv x_A \ (mod\ q')$

She issues her new public key as a follow:

$\left(q', g', h' = g'^{x_A'}\right)$ where $h' = 118$

Since Alice uses Xa which is congruent to Xa’, using Xa’ will considerably reduce the key size of finding the Xa. Moreover, it is quite trivial to find Xa’ since the modular q’ is relatively small.

Therefore, perform brute force attack by comparing g^Xa’ with h’

ii) Find Xa

Since q’ = 223, we can find out that Xa = Xa’ + 223k, perform brute force attack on k by comparing g^Xa with h.

iii) Decrypt Elgamal

After we have figured out Xa, attacker can easily recover the session key h^r by (g^r)^Xa. We can recover the message M by multiplying the inverse of the session key.

2. Comment for my source code

Main

```
if __name__ == '__main__':
    print('Initialization Done',end='₩n₩n')
    mode = input('0: Test  1: Train₩n')
    print('Mode:', mode)

    print('Start finding x_a2...')
    x2 = findXa2()

    print('PreComputing Intervals...')
    start, interval = preComputeInterval()

    print('Start finding x_a1...')
    x1 = findXa1(mode)

    print('Start finding M1...')
    findM1()
```

i) Find Xa'

ii) Precompute $g^{q'}$, $g^{Xa'}$

iii) Find Xa

iv) Find M

i) Find Xa'

```
def findXa2():
    h_pred = 1
    for x2_pred in range(1, q2):
          h_pred *= g2
          h_pred %= q2
          if (h_pred == h2):
            print('[find!] x_2:%d' % (x2_pred), end='₩n₩n')
            return x2_pred
```

For the range between 1~q' perform brute force attack by comparing $g'^{Xa'}$ with h'.

ii) Precompute g^q', g^Xa'

```
def preComputeInterval():
        start = 1
        for _ in range(189):
                start *= g2
                start = start % q1

        interval = 1
        for _ in range(q2):
                interval *= g2
                interval = interval % q1

        print('3^189 % q1: ', start)
        print('3^223 % q1: ', interval,end='₩n₩n')
        return start, interval
```

Since Xa' = 189 and q' = 223, precompute g^Xa' and g^q' in order to utilize it in brute forcing Xa.

iii) Find Xa

```
def findXa1(mode):
    #mode: Test(0) / Train(1)

    start_time = time.time()
    h_pred = start  # 3^189 % q1

    if mode == '0':
        s_idx = 906463617
        h_pred *= fast_power(interval,s_idx,q1)
        h_pred = h_pred % q1

    elif mode =='1':
        s_idx = 1

    for i in range(s_idx, int(1e10)):
        h_pred *= interval  # 3 ^ (233 * i + 189)
        h_pred = h_pred % q1
        if i % 1e8 == 0:
            print('index:', i, time.time() - start_time)
        if h_pred == h1:
            x1_pred = (i+1) * q2 + x2
            print('[find!] index:%d x_a: %d' % (i+1, x1_pred),end='₩n₩n')
            return x1_pred
```

Perform brute force attack on k where Xa = Xa' + 223k. Compare g^Xa with h.

Since this procedure takes quite considerable amount of the time, skip the brute forcing procedure when the program is on the test mode.

iv) Find M

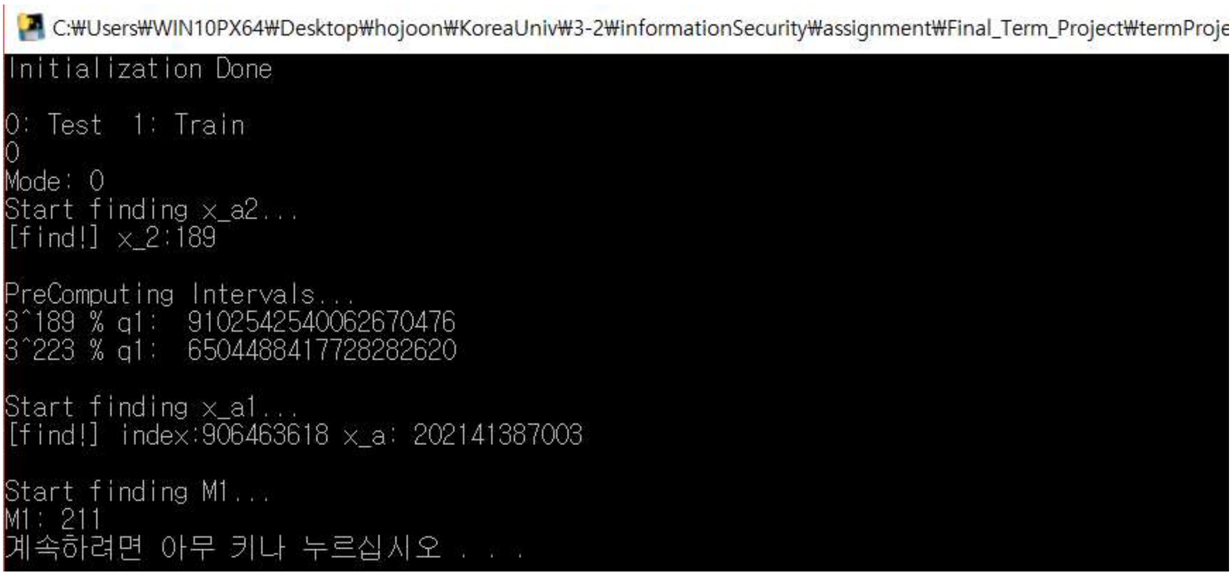
```
def findM1():
    K = fast_power(base=c1, power=x1, modulus=q1)  # h^r
    inv_K = extended_euclidean_algorithm(b=K, m=q1)
    M = c2 * inv_K % q1
    print('M1:',M)
```

K = session key (c1^x1 = (g^r)^Xa)

Multiply the inverse of the session key in order to recover from the K.

3. Screen Capture of the running program

Test Mode



Find the M1 without brute forcing the X_a

Train Mode

For the train mode, I perform the brute force attack in the Ipython with the same code.

```
In [6]:
start_time = time.time()
h_pred = start # 3^187 % q1

for i in range(1,int(1e10)):
    h_pred *= interval # 3 ^ (233 * i + 187)
    h_pred = h_pred % q1
    if i % 1e8 == 0:
        print('index:', i, time.time() - start_time)
    if h_pred == h1:
        x1 = i * q2 + x2
        print('[find!] index:%d x_a: %d'%(i,x1))
        break
```

```
index: 100000000 29.22079038619995
index: 200000000 58.31131076812744
index: 300000000 87.40251278877258
index: 400000000 115.73657298088074
index: 500000000 143.9417109489441
index: 600000000 173.1516842842102
index: 700000000 201.87481427192688
index: 800000000 230.91075921058655
index: 900000000 259.74120926856995
[find!] index:906463618 x_a: 202141387008
```

```
C:₩Users₩WIN10PX64₩Desktop₩hojoon₩KoreaUniv₩3-2₩informationSecurity₩assignment₩Final_Term_Project₩termProject2₩source_code₩dist
Initialization Done

0: Test  1: Train
1
Mode: 1
Start finding x_a2...
[find!] x_2:189

PreComputing Intervals...
3^189 % q1:  9102542540062670476
3^223 % q1:  6504488417728282620

Start finding x_a1...
```

If the program is on the train mode, this screen shows the program is trying to find the Xa with the brute force attack.

4. Answer

M = 211

Problem2

1. Approach to the problem

$$CT = (g^r, M_2 * g^{x_A r}) = (8312893525486221525, 7825868133432246571)$$

Since g^r and g^Xa are given, finding M2 will be the very trivial task. However, with the given hint *'use multiplicative inverse actively'*, this scenario would be not the case.

$$(PT, < CT >) = (m, < g^r, m * g^{x_A x_B r} >)$$

$$(727, < 8312893525486221525, 1009308953135723242 8 >)$$

Bob encrypts the plaintext with the random key r which is chosen randomly every time. However, with the given plaintext and ciphertext pairs, we can find out that one of the plaintext is encrypted with the same *r*. Therefore we can easily calculate the value of g^XaXbr used in the encryption, with multiplying the multiplicative inverse of the m.

$$CT = (g^r, M_2 * g^{x_A r}) = (8312893525486221525, 7825868133432246571)$$

Since all of the messages of the given plaintext and ciphertext pairs are relatively small, I decided to perform the brute force attack on the M2.

Moreover, if we can find out the Xb, we can easily compute M2^Xb * g^XaXbr. Since we know the value of the g^XaXbr, perform the brute force attack on k^Xb comparing with the M2^Xb

2. Comment for my source code

Main

```
if __name__ == '__main__':
    print('Initialization Done',end='₩n₩n')

    mode = input('0: Test  1: Train₩n')
    print('Mode:',mode)
    print('Start finding x_b...')
    x_b = findXb(mode)
    print('Start finding M^b...')
    M_b = findMexpB()
    print('Start finding M2 ...')
    M2 = findM2()
```

i) Find Xb by the brute force attack

iii) Compute M2^Xb

iv) Find M2 by the brute force attack.

i) Find Xb

```
def findXb(mode):
    #mode: Test(0) / Train(1)
    start_time = time.time()

    y_b_pred = 1
    s_idx = 1

    if mode == '0':
        s_idx = 4365000001
        y_b_pred = fast_power(g,4365000000,q)

    for i in range(s_idx, int(1e10)):
        y_b_pred *= g
        y_b_pred = y_b_pred % q
        if i % 1e9 == 0:
            print('index:', i, time.time() - start_time)
        if y_b_pred == y_b:
            x_b = i
            print('[find!] x_b:%d' % (x_b),end='₩n₩n')
            return x_b
```

Perform the brute force attack to Xb with comparing the value with yb(=g^Xb)

ii) Compute M2^Xb

```
def findMexpB():
    inv_m = extended_euclidean_algorithm(PT[3], q)
    g_xabr = C2[3] * inv_m % q
    inv_g_xabr = extended_euclidean_algorithm(g_xabr, q)
    M_b_mul_g_xabr = fast_power(M_mul_g_xar, x_b, q)
    M_b = M_b_mul_g_xabr * inv_g_xabr % q
    print('M^b = %d'%(M_b),end='₩n₩n')
    return M_b
```

1) get the inverse of the message m (=727)

2) get the g^XaXbr with multiplying the m-1 with the corresponding ciphertext

3) Exponentiation Xb to the M2*g^Xar and get M2^b * g^XaXbr

4) get M2^b with multiplying the inverse of the g^XaXbr

iii) Find M2

```
def findM2():
    for m in range(100000):
        M_b_pred = fast_power(m, x_b, q)
        if (M_b_pred == M_b):
            print('[find] M2:', m,end='₩n₩n')
            return m
```

Perform the brute force attack to k with comparing the k^Xb with the M2^Xb

3. Screen Capture of the running program

Test Mode

Train Mode

For the train mode, I perform the brute force attack to Xb in the Ipython with the same code.

```
In [5]:
start_time = time.time()

y_b_pred = 1
for i in range(1,int(1e10)):
    y_b_pred *= g
    y_b_pred = y_b_pred % q
    if i % 1e9 == 0:
        print('index:', i, time.time() - start_time)
    if(y_b_pred == y_b):
        x_b = i
        print('[find!] x_b:%d'%(x_b))
        break
```

```
index: 1000000000 239.48550701141357
index: 2000000000 497.1725034713745
index: 3000000000 757.0056359767914
index: 4000000000 1016.3188467025757
[find!] x_b:4365732901
```

```
C:₩Users₩WIN10PX64₩Desktop₩hojoon₩KoreaUniv₩3-2₩informationSecurity₩assignment₩Final_Term_Project₩termProject2₩source_co
Initialization Done

0: Test  1: Train
1
Mode: 1
Start finding x_b...
```

If the program is on the train mode, this screen shows the program is trying to find the Xb with the brute force attack.

4. Answer

M = 35281

Problem3

1. Approach to the problem

$$CT = (g^r, M_2 * g^{x_A r}) = (8312893525486221525,7825868133432246571)$$

$$CT = (g^r, M_3 * g^{x_A r}) = (4232920939787140673,12594363607212086362)$$

Suppose r1 as the r used in the first encryption and r2 as the r used in the second encryption.

With multiplying the second ciphertext of the both encryption, we can easily find out the M2 * M3 * g^Xa(r1+r2). Since the question is to find the M2 * M3, we have to find out the inverse of the g^Xa(r1+r2).

So, If we can find out the r1+r2, the question will be trivial.

2. Comment for my source code

Main

```
if __name__ == '__main__':
    print('Initialization Done!')
    print('Start finding (g^r1+r2)...')
    g_r1r2 = g_r_1 * g_r_2 % q
    print('g^(r1+r2) = ', g_r1r2)
    print('r1+r2 = ', 1,end='₩n₩n')

    print('Start finding M2*M3...')
    M2M3_mult_gxa = M2_mul_g_xar1 * M3_mul_g_xar2 % q
    inv_gxa = extended_euclidean_algorithm(y_a, q)
    M2M3 = M2M3_mult_gxa * inv_gxa % q
    print('M2*M3=', M2M3)
```

i) Get (r1+r2)

```
In [75]:
g_r1r2 = g_r_1 * g_r_2 % q
print('g^(r1+r2) = ',g_r1r2)
print('r1+r2 = ',1)

g^(r1+r2) =  3
r1+r2 =  1
```

We can easily find out that r1+r2 is equal to 1.

ii) Find M2*M3

Therefore, g^Xa(r1+r2) is equal to g^Xa which is y_a.

We can easily get M2*M3 by multiplying the inverse of the y_a to the M2*M3*g^Xa(r1+r2)

3. Screen Capture of the running program

4. Answer

M2 * M3 = 1631287597