REPORT

Information Security Project #2

ElGamal Cryptography

학과	컴퓨터학과
학번	2015410056
이름	김지윤
제출 일자	2018/12/12
담당 교수	허준범 교수님

Project #2: ElGamal Cryptography

■ Approach to the problems

ElGamal Cryptography는 exponentiation을 이용하여 encryption, decryption이 이루어지는 암호 방식으로 그 안전성은 discrete logarithm 계산이 매우 어렵다는 사실에 기반을 둔다. Private key를 X_A, public key를 Y_A라고 했을 때 Y_A로부터 X_A를 구하는 계산은 modulus q의 크기가 클수록 어려워진다.

문제에서 주어진 q는 15383399235709406497로 매우 크기 때문에 지수의 크기를 1씩 증가시켜가며 public key와 일치되는 것을 찾는 brute force attack으로는 private key를 구하는 것이 사실상 불가능하지만 powers mod 테이블에 존재하는 규칙성을 이용한다면 탐색 범위를 크게 줄일 수 있다.

1. Powers mod table의 규칙성

Modulus를 q라고 했을 때 0≤a<q를 만족하는 정수 a가 primitive root이려면 $\phi(q)$ = q-1의 모든 소인수 p에 대해 **$a^{(\phi(q))/p} \not\equiv 1 \pmod q$** 를 만족해야 한다. 그리고 이를 확장시키면 powers mod table속에 존재하는 규칙을 찾을 수 있다.

Powers mod 19

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Power mod 19 table에서 base a는 0을 제외한 19의 residue로 1부터 18까지 총 18개가 존재한다. 위 표에서 회색으로 칠해진 박스는 a의 지수승으로 나타낼 수 있는 residues를 나타낸다. 19의 primitive root인 2, 3, 10, 13, 14, 15는 group mod q를 generate할 수 있으므로 모든 범위를 포함한다.

그 외의 나머지 base는 일부만 포함하고 그 뒤부터는 구간이 반복되는 것을 알 수 있는데, base a의 지수승으로 커버할 수 있는 범위의 크기를 R(a)라고 했을 때 R(a)는 위에 기술한 특정 base가 primitive root인지 아닌지 확인하는 식을 이용해서 구할 수 있다.

① **$a^{(\phi(q))/p} \equiv 1 \pmod q$를 만족하는 p를 p_i라고 했을 때 $R(a) = \phi(q) / \prod p_{i\,(i=1,\dots,k)}$ 가 성립한다.**

(ϕ(q)의 prime factorization 결과에 x^n 형태가 존재할 경우 p에 $x^1, x^2, \dots, x^n$ 을 각각 대입하여 계산하고 가장 큰 n에 대해 x^n을 p_i 에 포함시킨다.)

- a가 primitive root일 경우

  위의 식을 만족하는 p는 1 외에 존재하지 않으므로 R(a)는 $\phi(q) = 18$이다.

- a = 4일 경우

  ϕ(q)의 소인수 2에 대해 위 식을 만족한다. $4^{18/2}=262144 \equiv 1 \pmod{19}$ 따라서 R(4) = 18/2 = 9이다.

- a = 7일 경우

  2와 3에 대해 위 식을 만족한다. $7^{18/2}=40353607 \equiv 1 \pmod{19}$, $7^{18/3}=117649 \equiv 1 \pmod{19}$ 따라서 R(7) = 18/(2*3) = 3이다.

- a = 18일 경우

  3과 3^2 대해 위 식을 만족한다. $18^{18/3}=34012224 \equiv 1 \pmod{19}$, $18^{18/9}=324 \equiv 1 \pmod{19}$ 따라서 $R(18) = 18/3^2 = 2$이다.

a^n mod q의 결과로 나오는 수를 b라고 했을 때 R(a)와 R(b)사이에는 포함관계가 존재한다.

② **$a^n \bmod q = b_n$ 일 때 $R(b_n) \le R(a)$ 이다.**

- a가 primitive root일 경우

  a^n mod q의 결과는 1이상 ϕ(19) 이하의 모든 수를 포함하고 R(a) = ϕ(19)로 최댓값이므로 위의 식이 성립한다.

- a = 4일 경우 (R(4)=9)

  4^n mod q의 결과로 나올 수 있는 수는 4, 16, 7, 9, 17, 11, 6, 5, 1 이고 R(4)=9, R(16)=9, R(7)=3, R(9)=9, R(17)=9, R(11)=3, R(6)=9, R(5)=9, R(1)=1으로 위의 식을 만족한다.

그리고 R(b_n)이 R(a)와 동일한 값이 나오는지, 줄어든다면 얼만큼 줄어드는 지는 지수 n과 관련이 있다.

③ a^n mod q = b_n 일 때 R(b_n) = R(a) / gcd(R(a), n) 가 성립한다.

a가 4일 경우를 생각해보자 (R(a) = 9)

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9
1	1	1	1	1	1	1	1	1
4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1
9	5	7	6	16	11	4	17	1
11	7	1	11	7	1	11	7	1
16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1

- 4^5≡17 (mod 19), gcd(9, 5)=1 → R(17)=9/1=9
- 4^6≡11 (mod 19), gcd(9, 6)=3 → R(11)=9/3=3
- 4^9≡1 (mod 19), gcd(9, 9)=9 → R(1)=9/9=1

따라서 base a를 알고 a^X mod q를 안다면 X에 대해 유추하는 것이 가능해진다.

a^X mod q = b라고 할 때 규칙①을 통해 R(a)와 R(b)를 각각 알 수 있고 규칙③의 식을 변형하면 gcd(R(a), X) = R(a) / R(b) 가 되므로 이를 통해 X에 대한 정보를 얻을 수 있다.

예를 들어 주어진 base a를 4라고 하자. 그리고 4^X를 알고 있을 때 X에 대한 식을 세울 수 있다. (R(4) = 9)

- 4^X=17일 경우

  R(17) = 9 → 범위가 줄어들지 않았으므로 X는 9의 모든 소인수와 서로소임을 알 수 있다.

- 4^X=11일 경우

  R(11) = 3 → gcd(R(4), n) = R(4) / R(11) = 3 따라서 X는 3k임을 알 수 있다. 이때 k는 (R(4) / R(11))과 서로소여야 한다.

총 정리하면 아래와 같다.

Base가 a이고 a^X mod q = b 일 때 X는 $b^{(\mathbb{R}(a))/p} \equiv 1$ (mod q)를 만족하는 모든 p_i에 대하여 $\prod p_{i\ (i\ =\ 1,\ \ldots\ ,\ k\ )}$ * k 이다. (k는 R(a)/ $\prod p_{i\ (i\ =\ 1,\ \ldots\ ,\ k\ )}$ 와 서로소)

2. Modulus q에의 적용

과제의 Problem1, 2, 3은 공통된 q와 base g를 갖는다.

- ✓ q = 15383399235709406497
- ✓ g = 3

ElGamal Cryptography에 의하면 g는 q의 primitive root여야하지만 문제에서 주어진 g의 값 3은 사실 primitive root가 아니다. ($3^{(\phi(q))/2} \equiv 1$ (mod q)) R(3)은 ϕ(q) / 2 이므로 일단 private key 값이 될 수 있는 수의 범위가 반으로 줄어든다는 사실을 알 수 있다.. (지수 n이 ϕ(q) / 2 이상일 경우 n- ϕ(q) / 2의 결과와 동일)

그리고 위의 식을 이용해서 Private key X_A에 대한 관계식을 구할 수 있다.

R(3) = 2^4 * 3 * 7 * 281 * 81466061026253

public key를 Y_A = 3^{X_A}라고 했을 때 $Y_A{}^{(\mathbb{R}(3))/p} \equiv 1$ (mod q)을 만족하는 p를 찾는다. p$\in${2, 2^2, 2^3, 2^4, 3, 7, 281, 81466061026253}

만약 결과가 모두 $\not\equiv$ 1이 나온다면 X_A가 2, 3, 7, 281, 81466061026253와 서로소라는 사실을 알 수 있지만 여전히 탐색 범위가 크므로 Y_A에 3의 multiplicative inverse를 곱해서 X_A-1에 대한 계산을 통해 범위를 줄인다.

위의 과정을 반복하여 X_A에 대해 $\underline{(p_1 * p_2 * \ldots * p_n) * k + m}$ 형태의 식을 만들 수 있고 그 후에 brute force attack으로 k의 값을 찾는다. k는 X_A를 직접 brute force로 바로 찾는 것보다 훨씬 빠르게 찾을 수 있다.

■ Problem #1

문제에서 주어진 Ciphertext CT = (g^r, M*h^r)을 decrypt하여 Plaintext M을 구하려면 X_A를 알아야 하고, X_A에 대한 식은 X_A'를 통해 구할 수 있다. (q', g')의 값은 (223, 3)으로 q'가 brute-force를 적용할 수 있을 만큼 충분히 작기 때문에 X_A'를 금방 찾을 수 있다.

- ✓ X_A' =189

그리고 X_A' ≡ X_A (mod q')를 만족하므로 $\underline{X_A = 189+223*k}$라는 것을 알 수 있다. (k는 0이상의 정수) k의 범위는 0부터 ((q-1)/2-189)/223까지이다. (R(3)이 (q-1)/2이기 때문)

K를 구하는 brute force attack을 python으로 구현했을 때 답을 내기까지 10 ~ 15분 정도 걸렸다.

```python
Q = 15383399235709406497 # modulus Q
test = 9102542540062670476 # 3^189 mod q
mul_operand = 6504488417728282620 # 3^223 mod q

for k in range(0, 34491926537465036):
    # 34491926537465036 = ((Q-1)/2 -189)/223
    if(test % Q == 12036625823877237123):
        print("k = " + str(k) + "& Xa = " + str(189+223*k))
        break
    test = (test*mul_operand)%Q
```

```
k = 906463618& Xa = 202141387003
```

- ✓ k = 906463618
- ✓ X_A = 202141387003

X_A를 구한 후엔 ElGamal Cryptography의 decryption 과정을 따라간다.

$C1^{X_A}$ = K를 계산하고 extended Euclidean algorithm을 이용해서 K의 multiplicative inverse를 구한 후 C2에 곱하여 plaintext M을 구한다.

```
************** Recovering key K *****************

****************** ANSWER *********************
key K is 11155503656725568082
***********************************************

****************** ANSWER *********************
message M is 211
***********************************************
```

- ✓ K = 11155503656725568082
- ✓ **M = 211**

Answer to problem #1: 211

■ Problem #2

Parameter q와 g는 problem #1과 동일하다.

문제에서 X_A에 대한 관계식을 찾을 수 없지만 powers mod table의 규칙과 multiplicative inverse를 사용해서 X_A의 식을 직접 세울 수 있다. (…■Approach to the problems)

3^n mod q의 결과로 나올 수 있는 수 X에 대하여 $X^{(\mathbb{R}(3))/p} \equiv 1$ (mod q)를 만족하는 p를 반환하는 Algorithm을 extended primitive root test라고 하자.

1) 3^{X_A} mod q = 3255928389273017819에 대한 extended primitive root test 결과는 1로 X_A는 2, 3, 7, 281, 81466061026253과 서로소이다.

```
-----3255928389273017819-----

1
****************************************
```

Prime factor 2, 3, 7, 281, 81466061026253 중에 $X_A - d$에 대해 extended primitive root test를 했을 때 나올 수 있는 적당히 큰 수는 281이다. (d를 1 또는 2씩 증가시켜서 extended primitive root test를 수행해야 하는데, 81466061026253은 너무 크기 때문에 결과가 281로 나오는 d를 먼저 찾는 것이 더 효율적이다.)

X_A 은 2, 3, 7, 281과 서로소이기 때문에 $X_A - 1$에 대해 테스트하면 결과에 반드시 2가 포함된다. 하지만 큰 prime factor를 우선적으로 찾기 위해 $X_A - 2k$를 테스트한다. 결과에 281이 포함되어야 하기 때문에 k의 범위는 우선 1부터 140까지 잡는다. $X_A - 2k$에 대한 테스트는 3^{X_A}에 3^{-2}를 반복해서 곱하는 것으로 가능하다. 3^{-2}은 extended Euclidean algorithm으로 계산할 수 있다.

2) g^{X_A-2k} mod q에 대한 결과는 아래와 같다.

```
****************************************
*** xA - 2
-----2071036402775824924-----

1 3 7
****************************************
*** xA - 4
-----8776448064591428601-----

1 281
****************************************
*** xA - 6
-----9751608960657142 89-----

1
```

따라서 $\boldsymbol{X_A - 4 = 281*k}$임을 알 수 있다. (k는 2, 3, 7, 81466061026253와 서로소)

base를 3^{X_A-4}으로 잡고 prime factor 3 또는 7에 해당하는 지수 위치를 구하기 위해 $X_A - 4 - 2*281k$에 대해 테스트한다. Base에 3^{-2*281}씩 곱하면서 차례대로 테스트하고 그 결과를 살피는데, 단위가 2*281k인 이유는 위에서 얻은 조건 $X_A - 4 = 281*k$의 형태를 유지시키면서 k가 3, 7과 서로소가 아닌 것을 찾기 위함이다.

3) $g^{X_A-4-562*k}$ mod q에 대한 결과는 아래와 같다

```
*** xA - 4 - 6744
-----1411173721 5922925078-----

1 281
****************************************
*** xA - 4 - 7306
-----8668338444687937161-----

1 3 7 281
****************************************
*** xA - 4 - 7868
-----8498353418214241835-----

1 281
```

따라서 $\boldsymbol{X_A - 4 - 7306 = 281*3*7*k}$임을 알 수 있다. (k는 2, 81466061026253와 서로소)

마지막으로 prime factor 2에 대해 테스트한다.

4) $g^{X_A-4-7306-281*3*7*k}$ mod q에 대한 결과는 아래와 같다.

```
*** xA - 4 - 7306 - 11802
-----3281146543662420077-----

1 3 7 281
**************************************
*** xA - 4 - 7306 - 17703
-----6184693921469044413-----

1 2 2 2 2 3 7 281
**************************************
*** xA - 4 - 7306 - 23604
-----12906650217232497070-----

1 3 7 281
```

따라서 **X_A – 4 – 7306 - 17703 = 2^4*3*7*281*k**임을 알 수 있다. (k는 81466061026253과 서로소)

결과적으로 식을 **X_A = 25013 + 94416k**로 정리할 수 있다. K의 값으로 올 수 있는 수는 1 부터 ((q-1)/2-25013)/94416까지이다. (R(3)이 (q-1)/2이기 때문)

Brute-force로 k의 값을 찾는다.

```python
Q = 15383399235709406497 # modulus Q
test = 8913546557048774327 # 3^25013 mod q
mul_operand = 13128116860492580201 # 3^94416 mod q

for k in range(0, 81466061026253):
    # 81466061026253 ~ ((Q-1)/2 - 25013)/94416
    if(test % Q == 3255928389273017819):
        print("k = " + str(k) + "& Xa = " + str(25013+94416*k))
        break
    test = (test*mul_operand)%Q
```

```
k = 7642041& Xa = 721530968069
```

- ✓ k = 762041
- ✓ X_A = 721530968069

(답을 내기까지 10초 내외 소요)

그 뒤의 과정은 problem #1과 같이 ElGamal Cryptography의 decryption 과정을 그대로 따라가면 된다.

```
************** Recovering key K *****************

******************* ANSWER *********************
key K is 12154419646935549966
************************************************

******************* ANSWER *********************
message M2 is 35281
************************************************
```

- ✓ K = 12154419646935549966

- ✓ **M_2 = 35281**

Answer to problem #2: 35281

■ Problem #3

Problem #3의 답은 두 가지 방법으로 계산할 수 있다.

1) **Private key로 M_3를 직접 구한 후 M_2*M_3를 계산**

   Private key X_A를 problem #2에서 구했기 때문에 M_3를 decrypt하는 것이 가능하다.

```
< Problem #3 - 1 >

************** Recovering key K *****************

****************** ANSWER *********************
key K is 11085755423107228565
************************************************

****************** ANSWER *********************
message M3 is 46237
************************************************

final answer --> M2 * M3 is 1631287597
************************************************
```

   - ✓ M_3 = 46237
   - ✓ **M_2*M_3 = 35281*46237 mod q ≡ 1631287597**

2) **M_2, M_3에 대한 ciphertext를 곱한 후 decryption**

   M_2, M_3의 ciphertext ($C1_2$, $C2_2$), ($C1_3$, $C2_3$)가 주어졌을 때 M_2*M_3의 ciphertext는 ($C1_2$*$C1_3$, $C2_2$*$C2_3$)와 같다. (…■Reference 두 번째 링크)

   이때 $C1_2$*$C1_3$의 값이 3으로 나오기 때문에, $C2_2$*$C2_3$= M_2*M_3*g^{X_A} 가 성립하고 g^{X_A}가 그대로 K가 되므로 private key의 값을 몰라도 결과를 낼 수 있다.

```
< Problem #3 - 2 >

** C1 = 3
--> r1 * r2 = 1

****************** ANSWER *********************
final answer --> M2 * M3 is 1631287597
************************************************
```

   - ✓ **M_2*M_3 = 1631287597**

Answer to problem #3: 1631287597

■ Code Descriptions

Exponentiation을 빠르게 계산하는 함수, extended Euclidean Algorithm, extended primitive root test를 수행하는 부분 등 대부분의 로직은 C++로 구현하였다. 하지만 C++에서 지원하는 정수의 최대 크기는 18,446,744,073,709,551,615로 이와 비슷하게 큰 두 개의 숫자를 곱하는 연산은 overflow가 발생해 bitwise로 계산하는 등의 방법을 써야하기 때문에 이 과정을 반복 수행하는 brute-force에서 시간이 상당히 오래 걸렸다. 따라서 brute-force 부분만 integer 크기에 제한이 없는 python을 사용해서 구현하여 계산 시간을 줄였다.

- ✓ 개발언어: C++, python
- ✓ 개발환경: Visual Studio 2017, jupyter notebook

1) find_msb, bit_substract, bit_compare

bitwise 연산을 위한 보조함수이다. 각각 주어진 수의 최상위비트 위치를 찾아주고, 두 수의 뺄셈을 수행하고, 두 수를 비교하는 기능을 한다. bit연산을 할 때는 C++의 bitset 클래스를 이용했다.

2) mul_mod_q

두 수를 인자로 받아 bitset 형태로 변환하고 곱한 후 modulo 연산을 수행한다.

3) exp_mod_q

base와 exponent를 인자로 받아 mul_mod_q를 exponent만큼 반복한다.

4) faster_exp_mod_q

exponentiation을 빠르게 수행하는 함수이다.

exponent로 들어온 인자를 자릿수대로 쪼갠 후 숫자의 길이만큼 $base^{10}$ 단위로 계산해 vector 클래스에 저장한다. 먼저 base와 10을 exp_mod_q의 인자로 넣어서 $base^{10}$을 구한 후 $base^{100}$는 $base^{10}$을 이용하여 계산한다. 자릿수만큼의 $base^{10k\ (k=0\sim자릿수-1)}$을 모두 계산한 후엔 exponent의 각 자릿수에 해당하는 숫자를 다시 지수로 계산하고 결과를 모두 multiplicative modulo 연산 하여 최종 결과를 리턴한다.

아래의 실행 화면은 각 자릿수별 exponentiation의 결과를 보여준다.

```
*** base = 2695597157275121
*** exp = 202141387003

base ^ 1 = 2695597157275121
base ^ 10 = 5642176270811103753
base ^ 100 = 6975620393316307755
base ^ 1000 = 12650273962511867221
base ^ 10000 = 6383488838625357
base ^ 100000 = 6590926499630897838
base ^ 1000000 = 13248935792163352465
base ^ 10000000 = 2162241419958605821
base ^ 100000000 = 12815674513824954589
base ^ 1000000000 = 10405127962193306132
base ^ 10000000000 = 7521695082107547903
base ^ 100000000000 = 15082202904969780217

base ^ (3 * 10^0) = 3979802843144796053
base ^ (0 * 10^1) = 1
base ^ (0 * 10^2) = 1
base ^ (7 * 10^3) = 3475688998745412549
base ^ (8 * 10^4) = 3309888273607029801
base ^ (3 * 10^5) = 14101732808475007146
base ^ (1 * 10^6) = 13248935792163352465
base ^ (4 * 10^7) = 1666191930190436426
base ^ (1 * 10^8) = 12815674513824954589
base ^ (2 * 10^9) = 3046467728881537276
base ^ (0 * 10^10) = 1
base ^ (2 * 10^11) = 13734522757350796903
```

5) extended_Euclidean

extended_Euclidean algorithm을 구현해 놓은 것으로 multiplicative inverse를 구할 때 호출한다. Multiplicative inverse 또는 두 인자의 gcd를 리턴한다.

6) recovering_K

K의 값을 알아내는 함수이고 내부 로직은 faster_exp_mod_q와 동일하다.

7) extended_prt

extended primitive root test를 구현한 함수이다. 주어진 인자 X에 대해 $X^{(\mathrm{R}(3))/p} \equiv 1 \pmod{q}$를 만족하는 소인수 p를 차례대로 출력한다. 이때 기본적으로 1은 포함하여 출력하도록 한다.

8) main

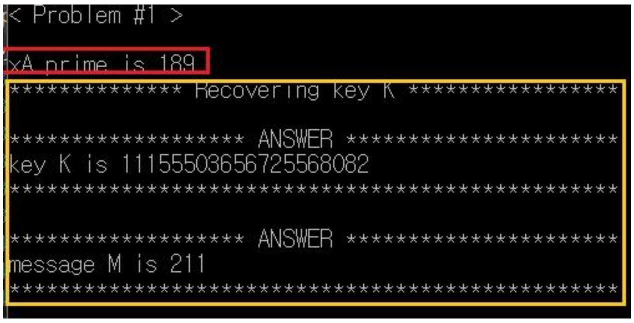
각 problem 별로 recovering_K 함수를 호출하는 것과 message M을 출력하는 것은 공통이고 problem #1에서는 X_A'을 계산하는 부분을, problem #2에는 X_A에 관한 식을 구하기 위한 step 1 ~ 4 부분을 추가했다.

■ Result

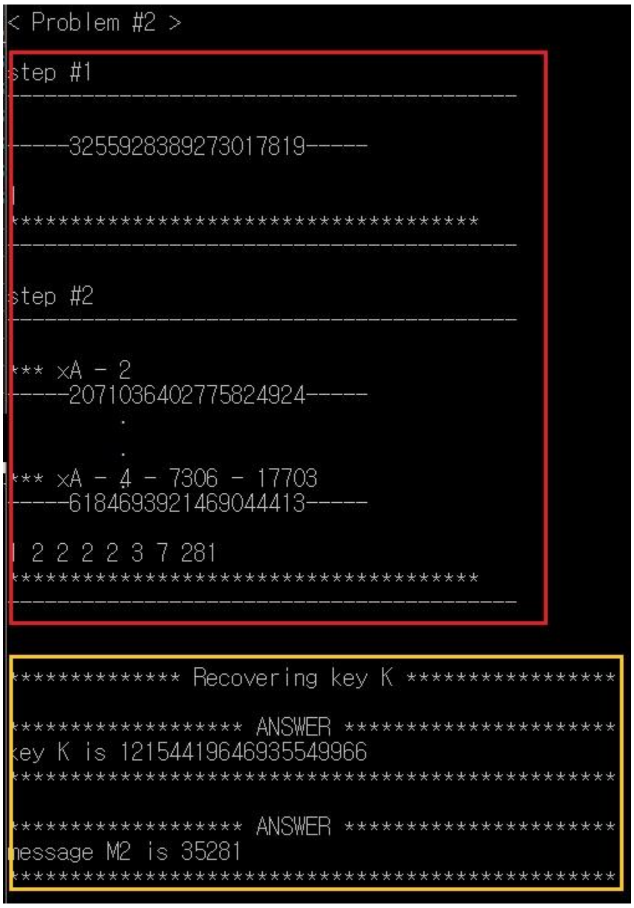
1. Screen capture of the running program

노란색 네모가 세 problem의 공통적인 부분 실행 화면이고 빨간색 네모 친 부분이 문제별 추가적인 실행 화면이다.

- Problem #1



- Problem #2



- Problem #3

```
< Problem #3 - 1 >
************** Recovering key K *****************

******************* ANSWER **********************
key K is 11085755423107228565
*************************************************

******************* ANSWER **********************
message M3 is 46237
*************************************************

final answer --> M2 * M3 is 1631287597
*************************************************
< Problem #3 - 2 >

** C1 = 3
--> r1 * r2 = 1
******************* ANSWER **********************
final answer --> M2 * M3 is 1631287597
*************************************************
```

(brute force 실행 화면은 위의 ■Problem #1, ■Problem #2에 각각 첨부했다.)

2. Answer

- Problem #1: **211**
- Problem #2: **35281**
- Problem #3: **1631287597**

■ References

✓ https://crypto.stackexchange.com/questions/8504/how-to-test-if-a-number-is-a-primitive-root

✓ https://crypto.stackexchange.com/questions/20262/how-does-chosen-ciphertext-attack-on-elgamal-work

✓ https://primes.utm.edu/curios/includes/primetest.php

✓ https://kulms.korea.ac.kr/bbcswebdav/pid-2269398-dt-content-rid-6349669_1/courses/20182R0136COSE35402/project2%282%29.pdf

✓ https://kulms.korea.ac.kr/bbcswebdav/pid-2254882-dt-content-rid-6075588_1/courses/20182R0136COSE35402/ch08_pdf.pdf