Lab 04 (4 hrs): Public Key Encryption

Part 1 (2 hrs):

Program 1: Textbook RSA (on group)

Note: **group** as in group theory. You are not allowed to finish your work in group.

In this part, you are required to implement the textbook RSA algorithm from scratch. It contains the following three procedures, KeyGen, Encrypt, and Decrypt.

- KeyGen

    - Return the private key (N, d) and the corresponding public key (N, e). The prime number p and q should be approximately 512 bits and therefore the RSA modulus number N should be approximately 1024 bits. Note that finding an encryption exponent e might be time-consuming.

- Encrypt

    - Given a plaintext message $m \in \mathbb{Z}_N$ and a public key (N, e), return the encrypted message c.

- Decrypt

    - Given a ciphertext message $c \in \mathbb{Z}_N$ and a private key (N, d), return the decrypted message m'.

Your program does the following:

- Generate a textbook RSA key pair. You may use the **Lucas-Lehmer test** algorithm to determine whether an integer is a **Mersenne number**. Print the private key and the public key as multiple decimal strings.
- Read a decimal string representing a plaintext message m. Raise an exception if m is invalid.
- Encrypt the message m. Print the encrypted message c as a decimal string.
- Decrypt the encrypted message c. Print the decrypted message m' as a decimal string.
- If you think the textbook RSA algorithm is secure, print `secure`. Print `insecure` otherwise.

Note that in this program, you may only include third-party codes or libraries for:

- **Lucas-Lehmer test**
- **Extended Euclidean Algorithm**

Recall that including any third-party codes without claiming is considered as lack of academic integrity, and results in failing this course.

Example Input & Output

Input:

```
34862844108815430278935886114814204661242105806196134451262421197958661737288465541172
28052282264426727851058932660434223148007593063773733202981602586546035311597026639261601
0728522314566623967383381778634506543197676413955090472603990245045652258420455647032
1705267433321819673919640632299889369457498214445
```

Output:

```
Private key:
N:
72480887416135972061737686062889407161759160887103574047817069443537714713215543172947
83530734489117281009226795379461120259106966115799279495983875047920850600568798168602
58093326914314738092927649888685810993301494587588613911084108256251417386985070860629
10615219209815042032904395035912581683751821198857
d:
32680572261276319950892386078453159129961789301515586779730994965995850002546722461272
34799763381989553235576065507646928431521315642413233339996484423792583164625536594707
25703083590669888231653526200740789172830362047160446101384913323096514769024246548458
97041133816851219279187868791233937199309119813013
Public key:
N:
72480887416135972061737686062889407161759160887103574047817069443537714713215543172947
83530734489117281009226795379461120259106966115799279495983875047920850600568798168602
58093326914314738092927649888685810993301494587588613911084108256251417386985070860629
10615219209815042032904395035912581683751821198857
e:
33917284234023552492304328018336609591997179645740843023623954792230653601864281593260
66343509514646381824066015974213055088773251100245591355034309587510535389881074409602
46358240711152649432516095007220627456180308250152396818170736446412943903473906997087
26562812289026328860966096616801710266920990047581
Ciphertext:
c:
15537860445392860627791921113547942268433746816211127779088849816425871267717435366808
46977176367294233930601962603311260479027952125601838800450398728136944430846373705989
49009876885030376518237593520612640313270065385240350928087627746864061941144561683359
3940445716413905575583403097832722646599808641232
Plaintext:
m':
34862844108815430278935886114814204661242105806196134451262421197958661737288465541172
28052282264426728510589326604342231480075930637737332029816025865460353115970266392616
01072852231456662396738338177863450654319767641395509047260399024504565225842045564703
21705267433321819673919640632299889369457498214445
insecure
```

Part 2 (2 hrs):

Program 2: ElGamal (on group)

In this part, you are required to implement the ElGamal algorithm from scratch. It contains the following three procedures, KeyGen, Encrypt, and Decrypt.

- KeyGen
  - Return the private key (p, α, a) and the corresponding public key (p, α, β). The prime number p should be approximately 512 bits. Note that finding a primitive root α in $\mathbb{F}_p$ might be time-consuming.
- Encrypt
  - Given a plaintext message $m \in \mathbb{F}_p$ and a public key (p, α, β), return the encrypted message (r, t) and the secret key k.
- Decrypt
  - Given a ciphertext message (r, t) and a private key (p, α, a), return the decrypted message m'.

Your program does the following:

- Generate a private key and the corresponding public key. You may use the **Lucas-Lehmer test** algorithm to determine whether an integer is a **Mersenne number**. Print the private key and the public key as multiple decimal strings.
- Read a decimal string representing a plaintext message m. Raise an exception if m is invalid.
- Encrypt the message m. Print the encrypted message (r, t) as multiple decimal strings.
- Decrypt the encrypted message (r, t). Print the decrypted message m' as a decimal string.

Note that in this program, you may only include third-party codes or libraries for:

- **Lucas-Lehmer test**
- finding a primitive root modulo prime p

Example Input & Output

Input:

```
4137696876930090267522398697653550193405311689664069574322834683213199126531348263326
63372150404977967354472129825302119195842950384279292950877363098912
```

Output:

```
Private Key:
p:
1148316665858548134715660146165222874762827430482676449544229642142501525316181363411
5028572768478982068325434874240950329795338367115426954714853905429627
alpha:
9312361210673900259563710385567927129060681135208816314239276128613236057152973946513
124497622387244317947113336161405537229616593187205949777328006346729
a:
3101984266868748920462287182124446696068493916489350126886947863612185839382696504960
7102905193887399253648679189884365033722973815059514162028592744461749
Public Key:
p:
1148316665858548134715660146165222874762827430482676449544229642142501525316181363411
5028572768478982068325434874240950329795338367115426954714853905429627
alpha:
9312361210673900259563710385567927129060681135208816314239276128613236057152973946513
124497622387244317947113336161405537229616593187205949777328006346729
beta:
1159968293290431483618624548862401630355209517151486248093696597103338439113317368321
7064382008047274612113322639139614505140087062058968033287419225545389
Ciphertext:
r:
4270390275647605104323112550114089020700231211424317817144932009272298324070546918004
12526755130971009544880644710431495709985658397526227672932741898380 5
t:
3221108136460372613636905604674169025183939828688657275543956232356097903511339858673
3064643419869114844822347893103409297302459291101463342807369264943 09
Plaintext:
m':
4137696876930090267522398697653550193405311689664069574322834683213199126531348263326
63372150404977967354472129825302119195842950384279292950877363098912
```