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:

```
3486284410881543027893588611481420466124210580619613445126242119795866173728846554117228052282264426728510589326604342231480075930637737332029816025865460353115970266392616010728522314566623967383381778634506543197676413955090472603990245045652258420455647032170526743332181967391964063229988936945749821444 5
```

Output:

```
Private key:
N:
72480887416135972061737686062889407161759160887103574047817069443537714713215543172947
```

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:

```
41376968769300902675223986976535501934053116896640695743228346832131991265313482633266
337215040497796735447212982530211919584295038427929295087736309809l2
```

Output:

```
Private Key:
p:
114831666585854813471566014616522287476282743048267644954422964214250152531618136341l5
0285727684789820683254348742409503297953383671154269547148539054296 27
alpha:
93123612106739002595637103855679271290606811352088163142392761286132360571529739465131
2449762238724431794711333616140553722961659318720594977732800634 6729
a:
31019842668687489204622871821244466960684939164893501268869478636121858393826965049607
10290519388739925364867918988436503372297381505951416202859274461749
Public Key:
p:
11483166658585481347156601461652228747628274304826764495442296421425015253161813634115
02857276847898206832543487424095032979533836711542695471485390542 9627
alpha:
93123612106739002595637103855679271290606811352088163142392761286132360571529739465131
24497622387244317947113336161405537229616593187205949777328006346729
beta:
11599682932904314836186245488624016303552095171514862480936965971033384391133173683217
06438200804727461211332263913961450514008706205896803328741922554539
Ciphertext:
r:
42703902756476051043231125501140890207002312114243178171449320092722983240705469180041
25267551309710095448806447104314957099856583975262276729327418983805
t:
32211081364603726136369056046741690251839398286886572755439562323560979035113398586733
06464341986911484482234789310340929730245929110146334280736926494309
Plaintext:
m':
41376968769300902675223986976535501934053116896640695743228346832131991265313482633266
33721504049779673544721298253021191958429503842792929508773630980912
```