Author: Sheng Huang Yao Chen

MiniRSA

COLLABROTION

Sheng Huang and Yao Chen made this RSA-based chatting program. We mainly worked together in a working mode called pair programming. And this is the github address for our work: https://github.com/yaochen1025/MiniRSA .

WHAT IS IT?

This program is a mini chat application using RSA cryptosystem. It consists of 3 parts: RSA cryptosystem, terminal version of server and client, GUI version of chat program.
-RSA package contains a java class of tools for encrypting and decrypting with unit test for them. It also has three classes with main function to simulate encryption, decryption, and cracking the key.
-Terminal version asks user to input information and calculates both public and private key. It then sets up a connection between server and client and allows users chat with each other.
-GUI version generates user interface when program starts and allows visualized operations by user.

Structures and Approaches

Packages:

-gui

This package contains two java class. GUI.java sets up gui and acts as message sender. The Reader.java acts as another thread working to receive messages. All the chatting content will reflect in the chattingArea.

Another thing to mention is that, there is no distinction between a client and a server in this class. The object will first try to act as a client, and if there is no server available, it will become a server automatically, listening for others. **So the firstly-run program will be server, and the secondly-run program will become the client.**

-modular

This package contains two java class that have main method: Server.java and Client.java. Others are supportive. Both Server.java and Client.java extends the ChatProgram class which is a wrapper of setting up connection as well as sending-key and receiving-key. Two class: MessageSender and MessageRecver extends Thread, so that sending and receiving messages can be asynchronously. And inside these two class, they communicate via a full duplex socket which provides input stream and output stream. The encryptor class is just a wrapper for encrypting and decrypting.

-rsa

This class is the rsa system. According to the specification in the assignment, you can run Encryptor, Decryptor, and Cracker to see whether the whole cryptosystem works. RSA.java is a full box of methods, and RSATest is the unit test for it.

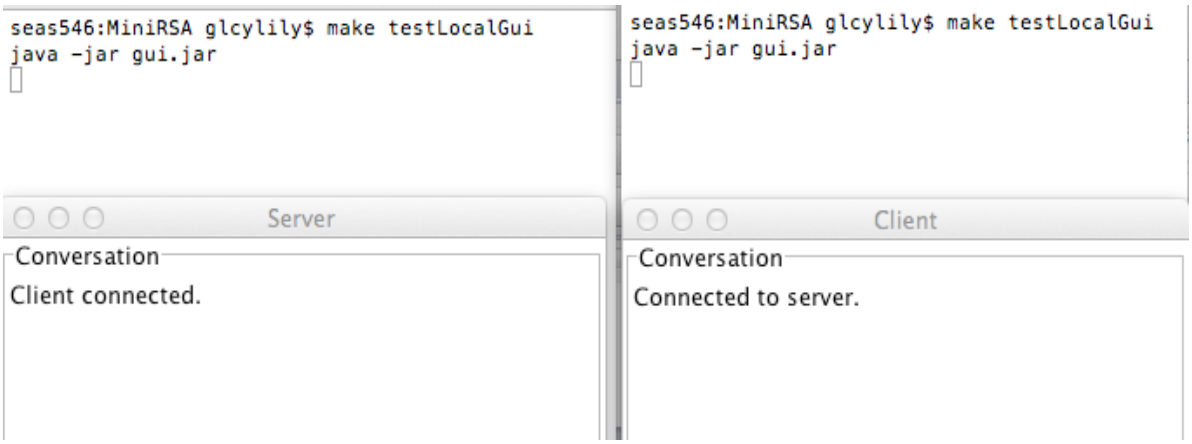
Sample Run:

1. ***make build*** (prepares all the jar files needed)

```
seas546:MiniRSA glcylily$ make build
ant gui
Buildfile: /Users/glcylily/Documents/workspace/MiniRSA/build.xml

gui:
      [jar] Building jar: /Users/glcylily/Documents/workspace/MiniRSA/gui.jar

BUILD SUCCESSFUL
Total time: 0 seconds
ant server
Buildfile: /Users/glcylily/Documents/workspace/MiniRSA/build.xml

server:
      [jar] Building jar: /Users/glcylily/Documents/workspace/MiniRSA/server.ja

BUILD SUCCESSFUL
Total time: 0 seconds
ant client
Buildfile: /Users/glcylily/Documents/workspace/MiniRSA/build.xml

client:
      [jar] Building jar: /Users/glcylily/Documents/workspace/MiniRSA/client.ja

BUILD SUCCESSFUL
```

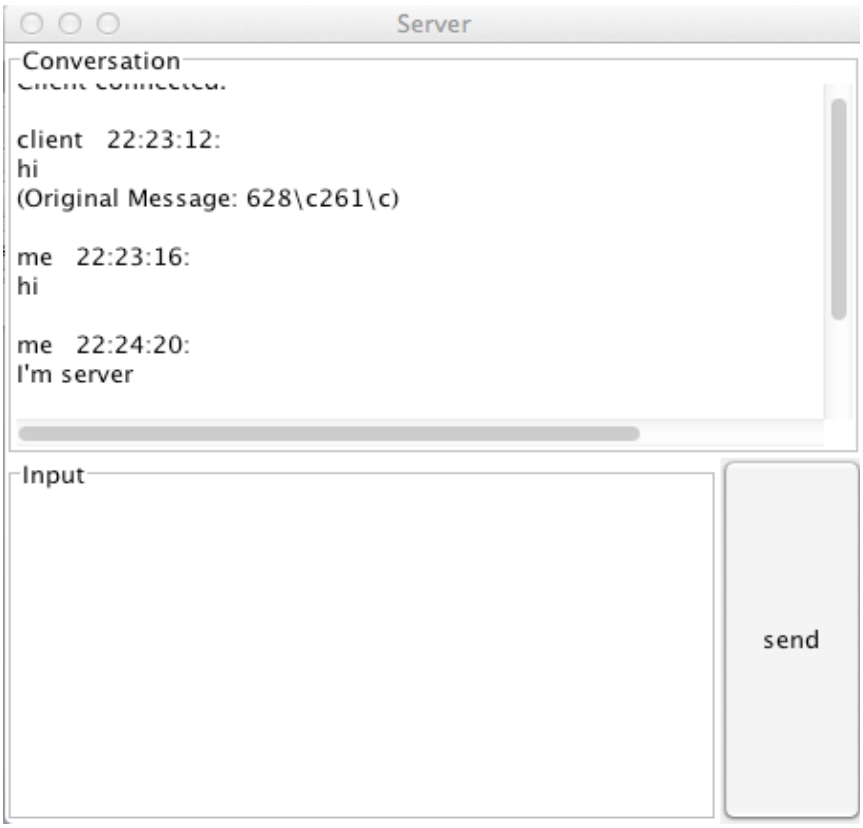
2. To test both end of GUI on local host:

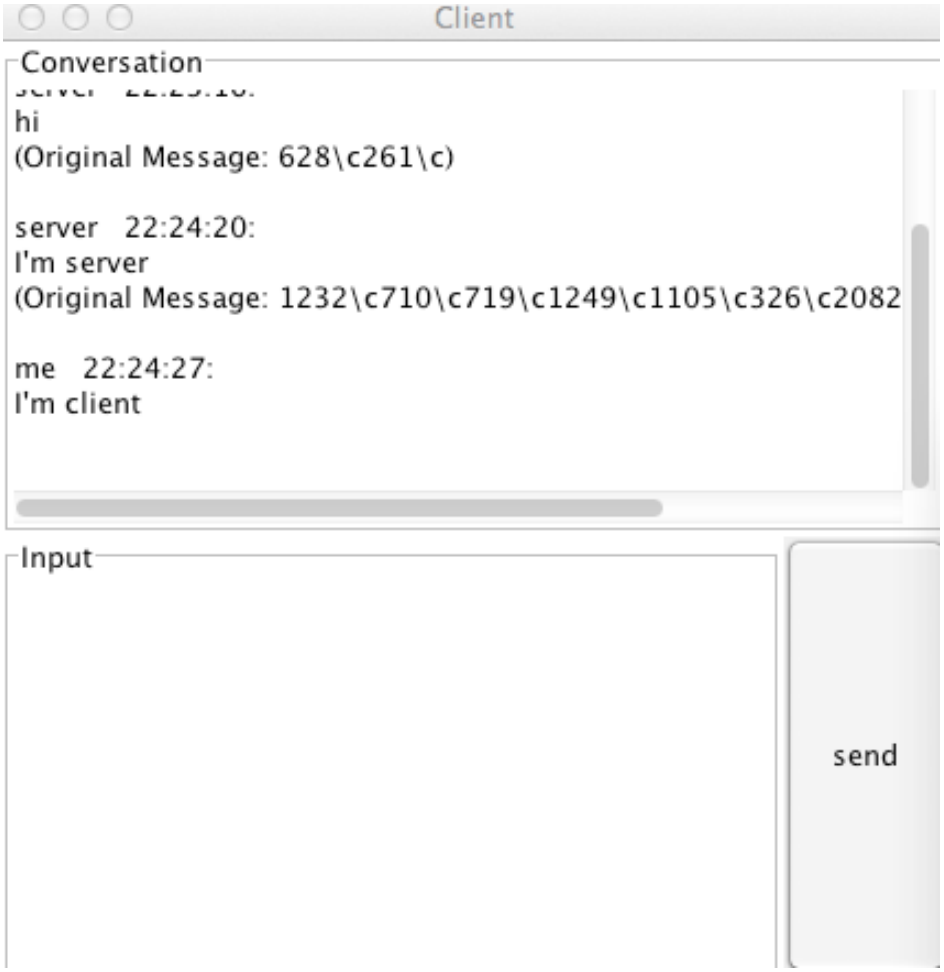
In our GUI version, there is no difference between server and client. User can simply open two terminals and input command line twice: ***make testLocalGui.*** The one runs first would automatically be the server and latter one would be client.



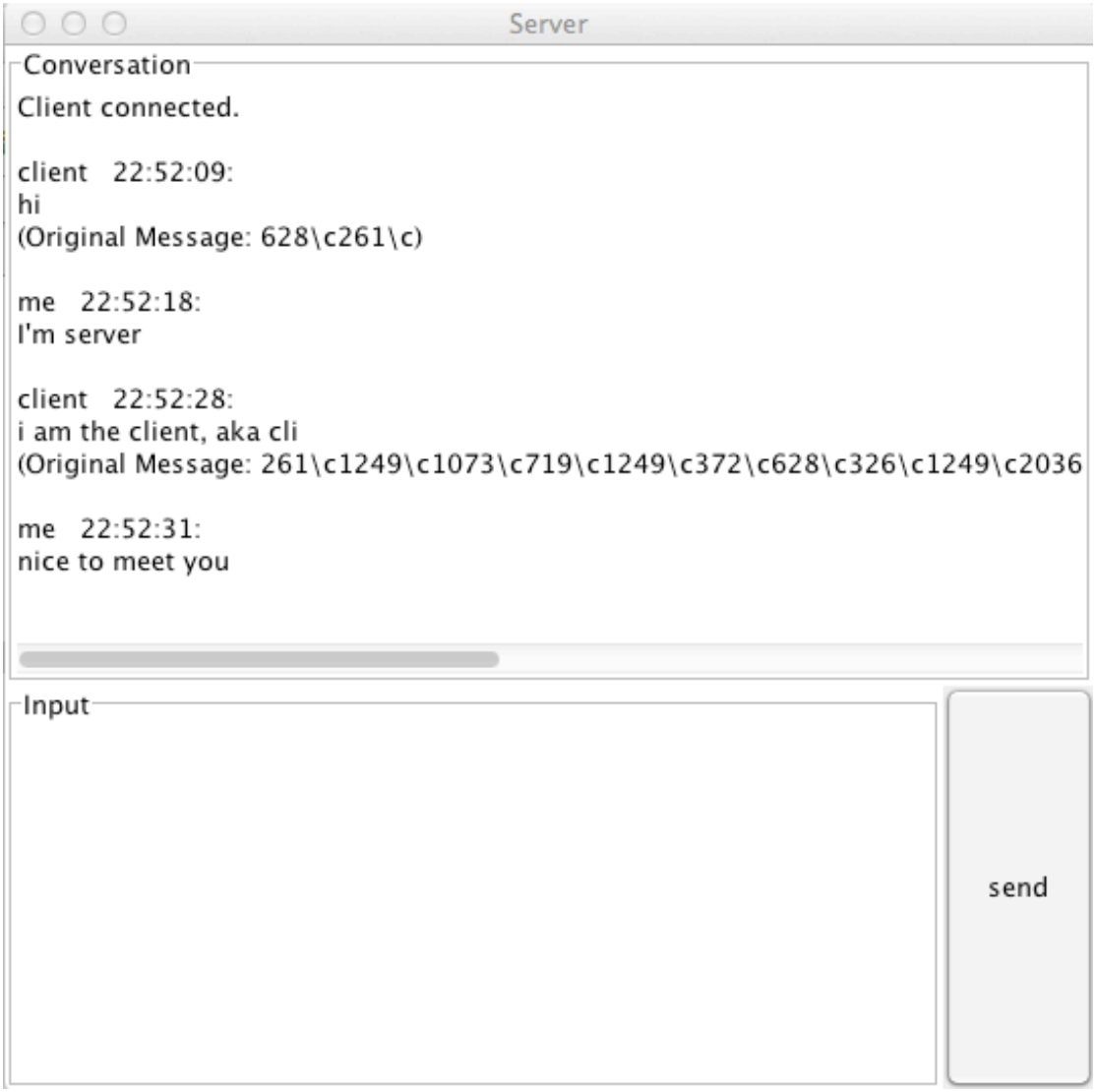
Author: Sheng Huang Yao Chen

GUI:





3. To test remote chatting:
For one end, ***java –jar gui.jar <ipaddress> <port>***
For the other end, ***java –jar gui.jar <ipaddress> <port>***
The port number should be the same



4. To test encrypt:

```
seas546:MiniRSA glcylily$ make testEncrypt
java -jar encrypt.jar
Please enter the public key value (e, c): first e, then c
451 2623
Please enter a sentence to encrypt
Hello!
1148
326
1145
1145
1780
2041
```

5. To test decrypt:

```
seas546:MiniRSA glcylily$ make testDecrypt
java -jar decrypt.jar
Please enter the public key value (d, c): first d, then c
1531 2623
Enter next char cipher value as an int, type quit to quit
1148
H 72
Enter next char cipher value as an int, type quit to quit
326
e 101
Enter next char cipher value as an int, type quit to quit
1145
l 108
Enter next char cipher value as an int, type quit to quit
1780
o 111
Enter next char cipher value as an int, type quit to quit
2041
! 33
Enter next char cipher value as an int, type quit to quit
quit
```

6. To test cracker

```
seas546:MiniRSA glcylily$ make testCracker
java -jar cracker.jar
Enter the public key value (e):
451
Enter the c that goes with the public key:
2623
a was 43 b was 61
The totient is 576
D was found to be 1531
Enter a number to decrypt, or "quit" to exit:
1148
This char is decrypted to 72
This letter is H
Enter a number to decrypt, or "quit" to exit:
326
This char is decrypted to 101
This letter is e
Enter a number to decrypt, or "quit" to exit:
1145
This char is decrypted to 108
This letter is l
Enter a number to decrypt, or "quit" to exit:
1780
This char is decrypted to 111
This letter is o
```

7. To test terminal version:

make runTerminalServer
make runTerminalClient

```
seas546:MiniRSA glcylily$ make runTerminalServer
java -jar server.jar
Enter the nth prime to compute:(a number greater than 5)
100
Enter the mth prime to compute:(a number greater than 5)
1001
100th prime = 541, 1001th prime = 7927, c = 4288507, m = 428004
e = 4172617096250338483, d = 496147, Public Key = (417261709625
8483, 4288507), Private Key = (496147, 4288507),
Waiting for client...
client connected
Receiving the other one's public key: (176263844484814493, 4815
)
Sending out my public key: (4172617096250338483, 4288507)
-------------
You can say something: (type "\bye" to quit)
-------------
Received from the other end:
hi
-------------
I'm server
-------------
```

```
seas546:MiniRSA glcylily$ make runTerminalClient
java -jar client.jar
Enter the nth prime to compute:(a number greater than 5)
1002
Enter the mth prime to compute:(a number greater than 5)
111
1002th prime = 7933, 111th prime = 607, c = 4815331, m = 4806792
, e = 176263844484814493, d = 4570229, Public Key = (17626384448
4814493, 4815331), Private Key = (4570229, 4815331),
connected to server!
Sending out my public key: (176263844484814493, 4815331)
Receiving the other one's public key: (4172617096250338483, 4288
507)
-------------
You can say something: (type "\bye" to quit)
hi
-------------
Received from the other end:
I'm server
-------------
I'm client
```