EECE 5850 - Computer Network Security
Project-II

Submitted by-

Md Zahidul Islam
(Contribution: Combining the RSA and IM Codes, Debugging, Implementing GUI, and Report Writing)

And

Parikshit Jadav
(Contribution: Debugging the RSA and IM Codes, Block Diagrams, and Report Writing)

<u>Overview of the RSA algorithm and its implementation for the Instant Messenger (IM) server:</u>

In this project, we have implemented the RSA algorithm for Encryption and Decryption and the chat server in python. As we know that before the beginning of the communication, the key pairs (private and public keys pairs) can be distributed to each client in the network by the Public Key Infrastructure (PKI) and Certificate Agency (CA), it is considered in the project that the clients are already aware about the public keys of the other clients and the private key of their own. For each client, a new private-public key-pairs are created using RSA algorithm by processing different pairs of prime numbers (unique for each client in the network) before beginning the communication. The public key of all the clients are shared among each other, which can be done using the IM server. However, for simplicity and demonstrating the concept of IM, the public keys are hard coded into the code files for the clients. This approach has been used to reduce the complexity of the key distribution part in the code. The overall process of the project is described below:

- The private and public keys are generated before beginning the communication. These pairs are generated through the rsa.py file. For the generating the keys, separate prime number pairs are used for each client and the key pairs are generated. These key pairs are hard coded in the client files. The main methods of rsa.py file are as follows:
  - The method *gcd*() is used to find the Greatest Common Divisor (gcd) for the parameters a and *b*.
  - The *multiplicative_inverse*() method is used to calculate the multiplicative inverse based on e and phi.
  - The method *is_prime*() is used to check whether the input number is prime or not. The method *generateLargePrime*() helps to generate *p* and *q* in RSA.
  - Method *generate_key_pair*() is used to calculate the public key is (*e*, *n*) and private key is (*d*, *n*) for the encryption and decryption functions.
- The first client establishes the connection with the server. After that, the second client establishes the connection with the server. Each client had its own pair of the private and public keys, and the clients are aware about the public key of the other clients as it is already present in the code for the encryption of the messages to be sent to other client in the network.
- The implementation uses the message from the user and encrypts the message with the public key of another client (which is broadcasted at the establishment of the connection.
- The code for encryption and decryption is added in the client files for each user. This file contains various methods (functions). The main methods are described below:
  - Method encrypt() is used for the encryption of the plaintext using the public key of the receiver and it return the cipher text.
  - Method decrypt() is used to decrypt the ciphertext received from the sender using private key. It return the plaintext after decrypting the cipher text.
- After encrypting the message, it is sent to another client. As the message is encrypted, there are less chances of leaking the information to another user, even though the intruder gets the encrypted message.

- After receiving the encrypted message, the client uses its private key to decrypt the message and the message is displayed in the terminal.
- The figures (Fig. 1, Fig. 2, Fig. 3) below show the final output process of the algorithm, right from taking the input from the sender, encrypting it and to send it to the receiver.
- A Graphical User Interface (GUI) is used for showing the messages of each client.

<u>Flow Charts:</u>

The flow chart below shows the process that take place in the entire conversation between two clients through a server.

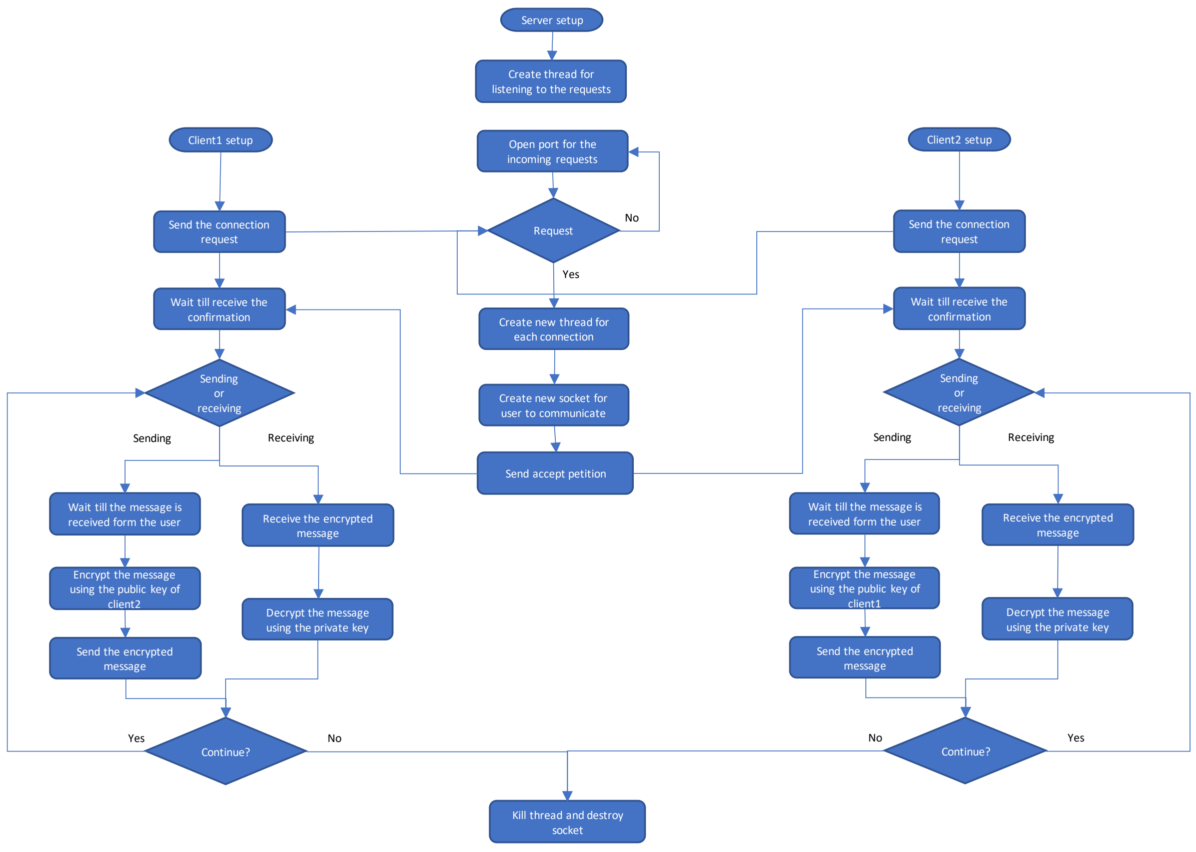

Server setup
Create thread for listening to the requests
Open port for the incoming requests
Request
No
Yes
Create new thread for each connection
Create new socket for user to communicate
Send accept petition
Client1 setup
Send the connection request
Wait till receive the confirmation
Sending or receiving
Sending
Receiving
Wait till the message is received form the user
Encrypt the message using the public key of client2
Send the encrypted message
Receive the encrypted message
Decrypt the message using the private key
Continue?
Yes
No
Client2 setup
Send the connection request
Wait till receive the confirmation
Sending or receiving
Sending
Receiving
Wait till the message is received form the user
Encrypt the message using the public key of client1
Send the encrypted message
Receive the encrypted message
Decrypt the message using the private key
Continue?
No
Yes
Kill thread and destroy socket

Fig. 1. Block diagram for the implementation of IM with RSA algorithm.

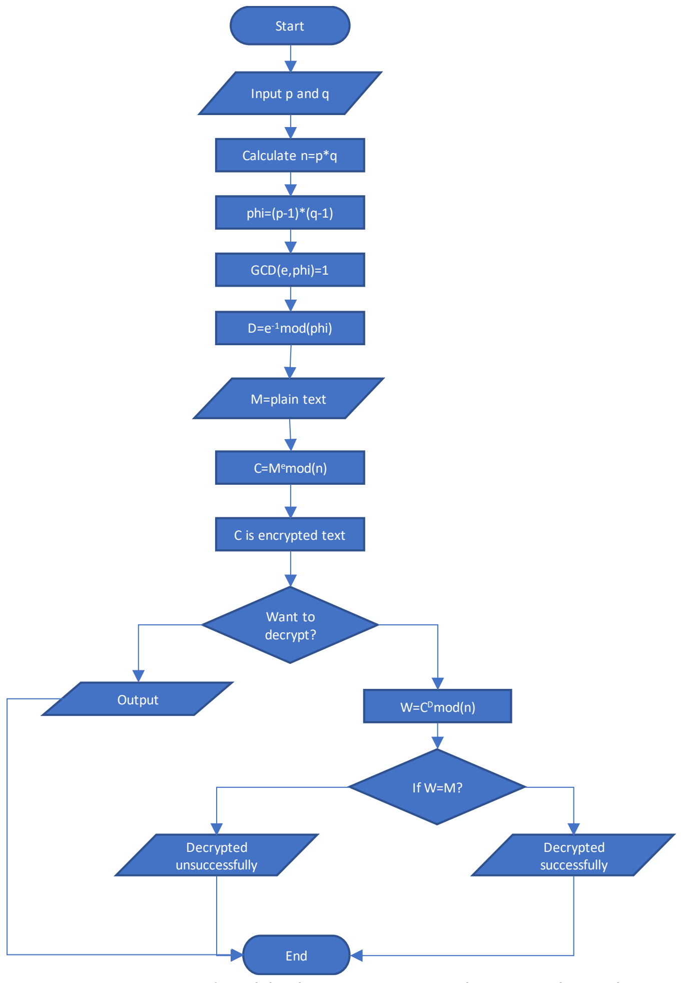


Fig. 2. Private and Public key generation by RSA algorithm.

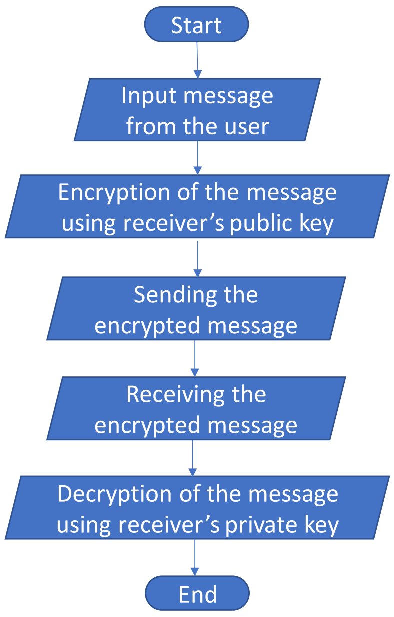
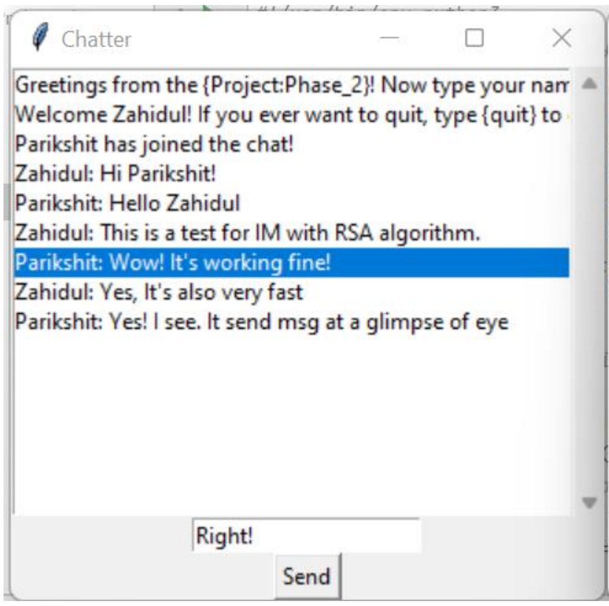

Start
Input message from the user
Encryption of the message using receiver's public key
Sending the encrypted message
Receiving the encrypted message
Decryption of the message using receiver's private key
End

Fig. 3. Encryption and Decryption process between two Clients.

Graphical User Interface (GUI):

The Final output is only shown in the GUI.

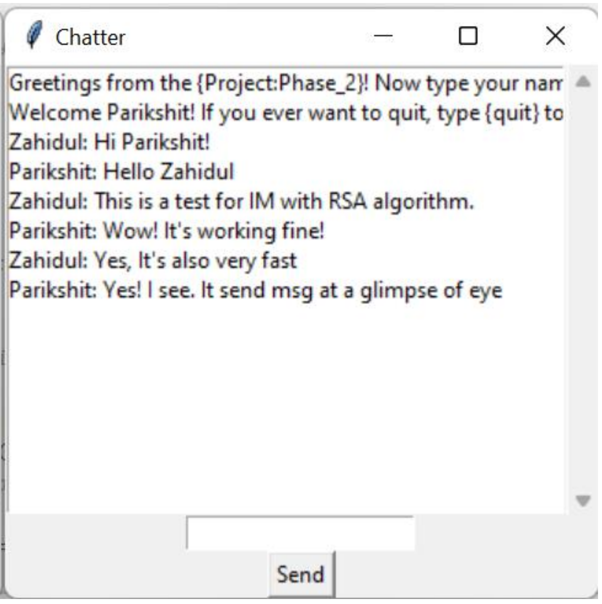
Client 1



```
Greetings from the {Project:Phase_2}! Now type your nam
Welcome Zahidul! If you ever want to quit, type {quit} to
Parikshit has joined the chat!
Zahidul: Hi Parikshit!
Parikshit: Hello Zahidul
Zahidul: This is a test for IM with RSA algorithm.
Parikshit: Wow! It's working fine!
Zahidul: Yes, It's also very fast
Parikshit: Yes! I see. It send msg at a glimpse of eye
```

```
Right!
```

Client 2



```
Greetings from the {Project:Phase_2}! Now type your nam
Welcome Parikshit! If you ever want to quit, type {quit} to
Zahidul: Hi Parikshit!
Parikshit: Hello Zahidul
Zahidul: This is a test for IM with RSA algorithm.
Parikshit: Wow! It's working fine!
Zahidul: Yes, It's also very fast
Parikshit: Yes! I see. It send msg at a glimpse of eye
```

Debug process:

The tricky part in the implementation process was to implement the method to generate key pairs. There is a need to find e and d in the process to create the key pairs (i.e. private and public keys). The main focus has to be kept on the complexity of the algorithm, to make the key pair generation faster even in the situation involving the use of large prime numbers. Another issue was faced while integrating the encryption and decryption code with the Instant Messaging (IM) code which we had referenced from the Github [1]. However, we need to include the RSA implementation with the IM, which is difficult to implement because of some points. First, in IM, the server broadcast a message after receiving it from the clients, whereas, in IM with RSA algorithm, the server should send the messages to the appropriate receiver so that the receiver can decrypt the received message. Second, as the entire message is end-to-end encrypted, the server does not have any information about the messages (e.g., who is the sender). To overcome this problem, some header message is added with the encrypted message to check information about the sender. Finally, the server also sends some common type of messages to all clients, which should be captured by all clients. These common message from server to clients is not encrypted in out implementation, as the purpose is to secure the end-to-end messaging.

Discussion:

This project has given us an opportunity to improve our programming skills through the hands on implementation of RSA algorithm and using it with IM for sending messages across multiple clients. The understanding developed through this project will be helpful in modifying it to enable the public key broadcasting of each client when they join the existing communication among other clients. In our implementation, the design is simplified by making some simple but realistic assumptions and it is easily application to implement the end-to-end secured IM for multiple users.

Reference:

1. Instant Messaging (IM) code: https://medium.com/swlh/lets-write-a-chat-app-in-python-f6783a9ac170
2. RSA refence: https://gist.github.com/angeloped/b225423200d5068f4e57707a5b3a8293