COL334 Assignment 2

Lakshay Saggi, 2017CS50412
Siddhant Mago, 2017CS50419

September 2019

1 Basic Chat Application

Here, we made a simple chat applications, which requires a server code to be run at some port on a computer, and mediates between 2 client applications. Whenever Client.java code is run on a node, the client sends request to server to open 2 sockets, on for sending and other for receiving messages tries to connect to a server, if the server assigns the same to the client.

Then, initially the client registers over the network and can send the messages to other clients over the network(Only if client follows the proper protocol). If the user doesn't follow proper headers and protocols for message sending, then the server automatically closes its connection with the client after sending an error message, since it doesn't know how to parse the data anymore. We also added an UNREGISTER message after which the user is removed from the sending and receiving list of the server and hence, effectively disconnected.

To deal with user directly exiting by Ctrl + C, we can have a setting that server keeps sending some polling message to the user to which expects some acknowledgement by the user. If server doesn't receive the acknowledgement for some consecutive polling messages(say 10) till some time, then the server assumes user is disconnected and deals with it accordingly(i.e. removes it from its sending and receiving lists).

To deal with offline users, the user will have to store the message sent by the client in its buffer and try to keep sending the message to the recipient till it receives an acknowledgment from the same. After this, the server can store the acknowledgment to be sent to the sender that the other person has seen/received the message and send it whenever the user comes online.

2 Encryption

Here, we encrypted the messages to be sent using the RSA type algorithm(using public and private keys) so that the chats remain secure. We only encrypted the message and not the headers. Initially, before sending any message, the sender requests the public key of the receiver, which the server provides from its database, and the sender encrypts the message using that key and server sends it to the receiver, which decrypts it using its private key.

We shouldn't send the encrypted data in binary format and should rather encode it in a string because binary data can be confused as control signals and certain operation on binary data can lead to different results. Also, since we are sending and receiving data in strings, so we would have to deal with binary data separately, but by encoding it to a string, we require no extra measures to read the data.

3 Message Signatures

To check message integrity, the sender hashes the encrypted message and then encrypts this hash also by using his/her own private key and then sends this message signature in the headers also. Then, the receiver checks if the message has not been tampered with by decrypting the hash using the public key of the sender and then obtaining the value of hash obtained, and compares it with the actual encrypted message to check that the message has not been tampered with.