Rishidev Prabhakar 2017EE30545

Chat Application With Encryption and Signature

There are two applications :

1. Client Application
2. Server Application

These Applications can run in given three modes :

1. Unencrypted
2. Encrypted
3. Encrypted with Signature

Incase of Unencrypted mode messages are not not encrypted and send over server as it is in string format. The server can actually read the content of message.

For Encrypted case the message is encrypted by the sender and then sent over to the server. Now this encrypted message can not be decrypted by server so server can not read the content of the message.

In Encrypted with Signature mode along with the encrypted message the sender sends a signature to ensure the receiver that this message is being originally sent by the sender.

The stepwise working of the application is described below:

Registration:

Firstly Server application needs to be run. On start server asks for the mode in which you want to run the application. Once the mode is set server starts listening for connection requests. Only client with this mode will be able to register on this server.
For registration, Client first opens two TCP connections to the server on the specified port. It registers one TCP connection for sending messages and other for listening to incoming messages. After registration both connections run on separate threads on the client. Server on accepting the connection creates a new thread and manage the connection there. If the connection is for receiving messages then server saves this socket in the database maintained by it and stops the thread. Data is stored in given data structure in server:

Hashtable<String, Hashtable<String, Object>> database;

Where key is the username of the user. database.get(username) will give another Hashtable where receiving socket, sending socket and other data related to given user is stored.
If Client is running in any of the encryption mode it generates Public and Private key pairs and stores them and also sends its public key to the server during registration. Server stores this key in the database.

User can also unregister by typing “unregister” in terminal. Server will remove all the data related to the client and client application will terminate.

Messaging:

In unencrypted mode client sends messages as it is in string format to server and waits for the response. Server then look up for the recipient’s receiving socket in the database. If the socket is not found then server send’s back the error to the sender informing that message can not be sent to the given recipient. Otherwise it forwards the message to recipient and waits for its response and then forwards the response back to the sender accordingly.

In Encrypted mode before sending a message sender needs to encrypt the message with the public key of the recipient. This encoded message can only be decrypted with the private key of recipient which is only with the recipient. Hence message content can not be read by someone else. So before sending message clients asks for the public key of the recipient from server. If server has the key saved in its database then it sends back the key to sender otherwise it sends Error that key doesn’t exist.
No data is sent in bytes. So before sending public key or any encrypted data it is Encoded to Base64. The receiver on receiving the message decodes it with its private key. If any error occurs like header of the message is missing it responds to server accordingly.

In Encryption with Signature everything is same as Encryption mode except an additional Signature is sent by sender to to ensure that message is actually sent by him.

H = hash(Encrypted Message)

H’ = K[pvt key of sender] (H)

This signature H’ is sent along with Encrypted message. At receiver end it requests the Public key of sender from server and then get back H as

H=K[pub key of sender](H’)

Then it equates

H=hash(Encrypted Message)

If it is equal then it ensures that message has not been tempered and that the sender is genuine. As private key is not shared by anyone else, others can not pretend to be the sender because it does not have the private key, so signature would not match.

Questions:

Question 1. Users may also disconnect arbitrarily by pressing Ctrl-C and not send an UNREGISTER message. How would you deal with such a scenario?

I have taken care of this issue in the application. On pressing Ctrl-C on Client application Server will remove the data of the user from its database. Any further send request to this user will be discarded by server telling the sender that user does not exist.
The Server's thread which manages Client's request is always listening for the requests on the client's sending socket inputstream. On pressing Ctrl-C client application terminates and socket connection ends. So the inputstream associated with socket returns -1 because end of stream has reached. The BufferReader associated with this stream will return null which will be detected and hence the user's data will be deleted from the server followed by closing the thread. I have implemented this by catching the NullPointerException which is raised when the server tries to operate on the string message received from inputstream which is null in this case.

Question 2. How would you extend the client and server applications to deal with offline Users?

We can maintain a received message queue for each user to store the messages needed to be delivered to this user. When this user is online, this queue will be dequeued and message will be forwarded to the user and respective response will be sent back to the sender of the message. If the sender is offline then the response will be enqueued to it's queue.
Also sender will not wait for the response from receiver after sending message as sender might be offline at that time. It will just wait for a response from the Server that the message has been enqueued to the respective user's queue. This will ensure the sender that message has been sent. When this user becomes online it will start receiving the messages queued into its queue. Then it will send back the response that message has been delivered which will ensure sender that message has been delivered and read.

Here are the instructions to run the application:

You need to compile the files in the src folder using:

javac TCPServer.java
javac TCPClient.java

First you need to run the Server application in one terminal.
Then run the Client application on other terminal.

SERVER APPLICATION:

To run the SERVER application open a terminal in src folder and run the following command:
java TCPServer

Then SERVER will start and ask for the mode in which you want to run the application. Modes are:
1. Unencrypted Mode
2. Encrypted Mode
3. Encryption with Signature
Then type 1, 2, 3 depending on the mode you want to run the server and press enter.

NOTE that the server will ONLY ACCEPT the REGISTER request of the Client running in same mode. Any connection request from Client running on different mode will be rejected.

CLIENT APPLICATION:

To run the CLIENT application open a terminal in src folder and run the following command:
java TCPClient

Then CLIENT will start and ask for the mode in which you want to run the application. Modes are:
1. Unencrypted Mode
2. Encrypted Mode
3. Encryption with Signature
Then type 1, 2, 3 depending on the mode you want to run the server and press enter.

Then CLIENT will ask for username. NOTE that username can ONLY contain alphabets and numbers without spaces. Type your username and press enter.

Then CLIENT will ask for IP Address of the SERVER. Type the IP address where SERVER is running. If it is on the same device type "localhost" without quotes.
If everything goes well you will receive registration successful message. Otherwise application will inform you of any Error.
You can send a message to the users connected to THE SAME SERVER you are connected by writing as "@usertosend message" where "usertosend" is the username of the user you want to send the message. Add "@" before the username.