Instant Messaging Application

Abstract:

In this era, when Instant Messaging apps like Whatsapp, messenger and Gtalk are widely used, creating a prototype of peer to peer messaging tool was intriguing and challenging. This model helps in establishing communication between two entities. By introducing encryption/decryption techniques and user authentication methods, achieved secured communication.

Introduction:

The aim is to design a secure instant peer to peer messaging tool providing a secure connection between two clients. For instance, if Alice and Bob are the clients and they want to communicate with each other. If intruder wants to eavesdrop he/she shouldn't be able to do it. To build a secure connection between clients we must use strong encryption and decryption algorithms. This messaging tool supports following principles:

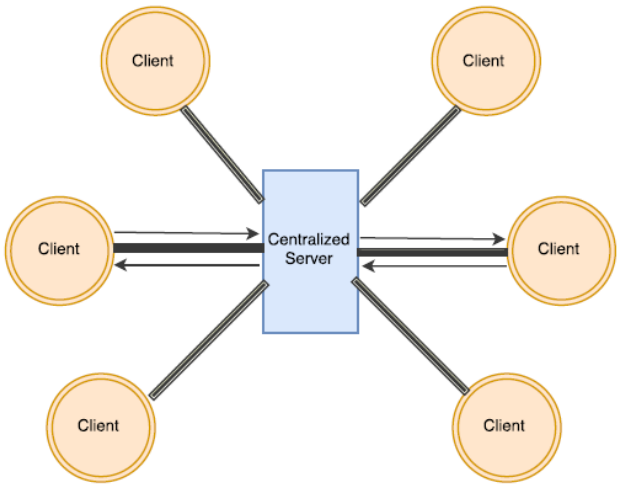
1) Alice and Bob use the messaging tool to send instant messages to each other.
2) Alice and Bob have their own passwords and login on authentication.
3) Every message is transmitted is encrypted using a 128 Bit key.
4) When Alice sends a message to Bob, Bob receives it when he logs into the system.

The technologies used are JAVA, Encryption/ Decryption API's, Socket programming API's.

Architecture:

The messaging application is based on Client – Server Network model. In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

Instant Messaging Application



Centralized client server architecture

The underlying network is built on TCP/IP protocol. TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. A TCP socket is one end-point of a two-way communication link between two programs running on the network. It is defined by IP address and a port. A socket is bound to a port number such that the TCP layer can identify the application to which the packet is destined to.

To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. Server will listen on a specific port and is said to be “bound” to it. Only one process(server) can listen on one port at any time. The client, when connecting to server, specifies the port on which it must communicate. If the server is listening on the same port, then communication is said to be successful. This implies that, the port that is specified by the client and the port to which the server is listening to must be the same. Hence, the client and the server each reads from and writes to the socket bound to the connection.

The Server can serve more than one client at one time. The current implementation supports maximum of 5 clients, but this can be extended to support higher number of clients.

Implementation:

Client:

Instant Messaging Application

- When a client logs in, he can register himself by setting up a password.
- Each client is given a clientID which is unique to him. This can be used as his username.
- The password is hashed and stored in the program using SHA-1.
- Once, the client logs in, his password is hashed and validated against the stored hash value. If it matches, access is granted.
- The client, when logging in, needs to give the IP address of server, his clientID, password followed by message and to_clientID in that order.
- Encryption is done on the client side using AES with CBC chaining mode. PKCS5 padding is used to the message for further enhancing security.
- Message is passed to output stream using PrintWriter objects.
- If he received a message, it is read from the socket using BufferedReader and Decryption is done.

Server:

- Server as the name itself suggests it's something that helps in communication. It receives a message from the client and then passes it on to the other participating client. This is achieved through socket programming.
- It also generates session keys and sends it to both the communicating parties.
- To support multiple client's system, we use multiple server threads.
- The Server will be in waiting status initially, when client initiates a connection to the server it will create a thread and service socket for each client.

Server Thread:

- This is the server thread that is forked in order to serve the client.
- It then creates a service socket for the client.
- Server reads the message from the socket using BufferedReader object.
- It checks if there are any messages to be sent to this client. If so, it will forward the message. Also, it will receive any response message back from the client.
- Messages are stored in an array indexed by client ID (Destination). All the messages are prefixed by client ID (Source). For example, if the message from client1 to client4 is "Hello", it is stored as "client1: *Hello" in messages [4].*
- When client4 logs in, it checks for existing messages and sends it to him. It then **deletes** the messages. If not he will receive "No Messages" comment.
- The server here is **Stateless**.

Instant Messaging Application

Password Authentication:

The Client/user logs in in with their password and this is being authenticated using **SHA-1**. SHA-1(message and its digest) is a one-way cryptographic hash function and hence useful for password authentication of client when he/she logs in to chat. Here, we have used the basic implementation of SHA-1 which uses 160-bit hash value as message digest. Pre-image attack will require 2^159 possible attempts which will make it infeasible to guess the message from its digest. This has advanced security by using deliberately slow schemes, such as **PBKDF1** (inbuilt with SHA-1).

Encryption/Decryption:

AES (Advanced Encryption Standard):

We used AES which is a symmetric key encryption technique. AES depends on substitution permutation methodology. It is quick in both programming and hardware. AES is a variant of Rijndael cipher which has a fixed block size of **128 bits**, and a key size of 128, 192, or 256 bits. AES operates on a 4 × 4 column-major order matrix of bytes. Most AES calculations are done in a finite field. The key size used for an AES cipher gives the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of cycles of repetition are 10 cycles of repetition for 128-bit keys, 12 cycles of repetition for 192-bit keys, 14 cycles of repetition for 256-bit keys. Each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

PKCS5 Padding:

Almost all cryptographic hash functions process messages in fixed-length block size, all but the earliest hash functions include some sort of padding (the process of adding required number of bits) scheme. It is important for cryptographic hash functions to employ termination schemes that prevent a hash from being vulnerable to length extension attacks. Many padding schemes are based on appending predictable data to the final block. For example, the pad could be derived from the total length of the message. PKCS (Public key Cryptographic standards) standards are published by RSA security. PKCS#5 is used for block ciphers, every implementation of java platform supports this. In PKCS #5 the password-based encryption

schemes here are based on an underlying, conventional encryption scheme, where the key for the conventional scheme is derived from the password.

CBC MODE:

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block. In CBC encryption is sequential, the message is padded to a multiple of cipher block size with ciphertext stealing. This is used for processing of messages that are not evenly divisible into blocks without resulting in any expansion of the ciphertext. **Ciphertext stealing** for CBC mode doesn't require the plaintext to be longer than one block. In the case where the plaintext is one block long or less, the **Initialization vector (IV)** can act as the prior block of ciphertext. This may not be possible in situations where the sender cannot freely choose the IV when the ciphertext is sent (e.g., when the IV is a derived or pre-established value), and in this case ciphertext stealing for CBC mode can only occur in plaintexts longer than one block. Decrypting with the incorrect IV causes the first block of plaintext to be corrupt but subsequent plaintext blocks will be correct. This is because each block is XORed with the ciphertext of the previous block, not the plaintext, so one does not need to decrypt the previous block before using it as the IV for the decryption of the current one. This means that a plaintext block can be recovered from two adjacent blocks of ciphertext.

Conclusion:

Working Model:

1) **Input: "**IP Address of server" "Source_ClientID" "password" "message" "To_ClientID"

2) Client4 does not have any new messages.

```
run:
Received:

__________
cipher:No Messages
Sent:

______
Plain:Hey
Client4:oNpEh7OOUxm2yOjN2Xbeew==
BUILD SUCCESSFUL (total time: 2 seconds)
```

3) Client 2 tries to login.

```
192.168.0.17 4444 2 2324Hellloo 4
```

4) Client side authentication is failed due to password mismatch. Hence, access is not granted

```
run:
Authentication failed
BUILD SUCCESSFUL (total time: 0 seconds)
```

5) Client2 tries to login with correct password.

Arguments: 192.168.0.17 4444 2 2222 Hellloo 4

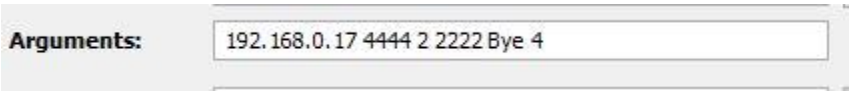
6) Client2 receives the messages client4 sent. Message from client2 is then sent to server.

```
Received:

__________
cipher:Client4:oNpEh7OOUxm2yOjN2Xbeew==
Plain:Hey
Sent:

______
Plain:Hellloo
Client2:Vn4L5E37QqUqpuQ9QiDSpg==
BUILD SUCCESSFUL (total time: 1 second)
```

7) Client2 types the message “Bye” and logs out.



2 is logged out.

Limitations:

1. The current implementation does not have a GUI window for chat sessions.
2. As the GUI was not implemented, the order of input must be maintained and the user has the burden of remembering it.
3. Though the system can support more number of clients, password setup supports only 5 users. This can be improved.

Feasible Advancements:

1. To improve security, key can be updated at regular intervals.
2. Diffie Hellman algorithm for key exchange could have been used to enhance security when symmetric key encryption/decryption algorithms like AES is used.
3. Advanced security can be achieved by using password salt and with other deliberately slow schemes like bcrypt or scrypt with sufficient work-factor.
4. A secured database to store messages and passwords can be used.