Team 4 Report

**b) Challenge/Response**

**Authentication:**

The server stores a database of registered users and their associated secret keys.

When a user attempts to connect to a server, it sends it’s username to the server via UDP packet. The server retrieves this username, checks to see if the user exists, pulls their secret key *U\_SK*, and generates a random number *rand* that will help to authenticate the user.

Using an MD5 hash, the server hashes (rand+U\_SK) and stores the result, *serverChallenge*. The server then sends the random number to the client.

The client runs the same hash function using their secret key and the received random number and sends over their *challengeResult*.

The server compares challengeResult to serverChallenge in order to verify.

**Encryption:**

The server and client utilize the bouncy castle encryption API and utilizes synchronous encryption. Client and server use AES/CBC/PKCS5Padding for messages with decryption via secret key.

In order for both the client and server to share the same secret key, the secret key is generated using an SHA-256 hashing of (rand+U\_SK).

**c) Hardware setup/ Configuration**

We are using the Amazon Web Service (AWS) Elastic Computing (EC2) virtual service in the cloud with the instance t2.micro. Which give us 1 virtual CPU (vCPU) and 1 GiB of memory. Plenty of power to run our Chat Server program anytime and anywhere. It creates virtual computing environment, known as instances. Each T2 instance type includes one or more instances sizes, balance, memory and network resources.

Our base hardware is an Intel i7 processor with 8 GB of RAM with internet connection. Using it to log on to AWS and create multiple T2 instances to simulate our Chat Server program.

**d) Protocol State Diagram**

UDP handshake

Server

Client B

Client A

Send login

Send login

No

No

No

No

No

Yes

Yes

Yes

Yes

End chat?

End chat?

Timeout/null

Timeout/null

Yes

Yes

Send message

Send message

No

No

Connected?

Connected?

Authentication

Chat session ends

**e) Screenshots**

**f) Issues**

One of our main issues was to come up with an encryption method that would satisfy the project’s requirement. We did not know how to implement the encryption between message for the clients. We decide to use DES encryption and could not get it to work. We did implement most tasks by using hash tables because it was fast and efficient. We even try to integrate hash table to do encryption by passing client unique key and hash it in server and send it back, vice versa. However, that did not meet the requirement of the project and failed the integrity protection. We end up doing a lot of research and come up with an idea to use an external API. As stated in part (b) we used bouncy house encryption API and import it to use with our program which was coded in JAVA. We succeeded with a SHA 256bit cipher key.

This project could be improved by implementing GUI, require new clients to create account and verify client information if already existed and if it doesn’t create that client. Stored all client’s data in an encryption space.

**g) Video clip**

link: