**FILE TRANSFER PROJECT DOCUMENTATION**

By

BAO NGUYEN – BCN140030

MOFAN LI – MXL162930

CS 6349.001

**I. Project Description and Features**

*1. Description*

The problem is to create a file transfer system that utilizes Socket Programming, Connection Management, Reliable Communication, and security protocol that utilizes SHA-1. It allows 2 individuals (1 server and 1 client) on different computers to communicate via network. All messages transported over the network shall be secured with respect to Authentication, Confidentiality, and Integrity.

*2. Features*

We use Java to build this program. We create 2 different projects/modules, a *Client.java* and a *Server.java*, both lie within the *FileTransferApplication* project. The server sets up the sockets and wait for connection. The client sets up its socket and connect to the server. This program also records the start and end time of the session. This project supports Windows, Mac, and Linux.

The server does not initiate any message. It waits for the commands from the client and sends responses. The client can request server to view all the files the server contains, to download files from the server, or to upload files from local host to store in the server. There are several commands that allows the client to communicate to the server, as followings:

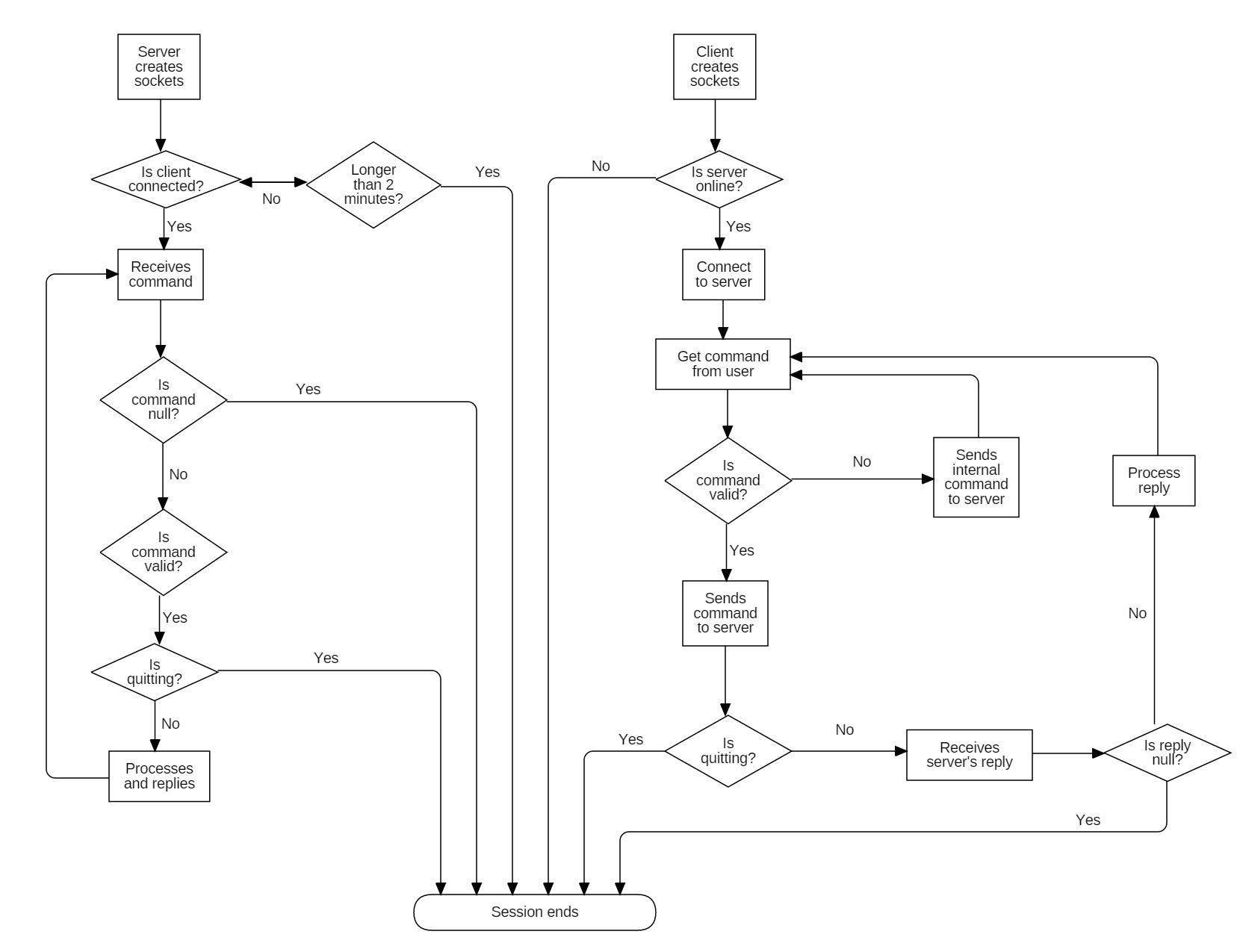
* List: request server to send a list of files that the server contains
* List-me: display a list of files it has downloaded from the server
* Stay: requests the server to keep the connection (this happen when the user enter invalid input, but Java will close the socket after each session – as states below – therefore has to provide a mechanism to keep the connection alive)
* Upload: upload a file to the server
* Download: download an existing file from the server’s library

Since Java forces to close the stream in order to finish sending a file as byte stream, we has to close the socket after each transaction (send message and receive message together is consider a transaction) and reopen new socket (same server’s IP and port with the previous socket) for another transaction. Therefore, there are multiple sessions between client and server during run time. The first session, called S0, performs a major authentication (key exchanges, server remembers this client, etc.) and comes up with a session key K0. In the follow-up sessions, the server first will make sure it still talks to the same client by validating client’s IP address and ID. This is the only authentication that the server performs on the client. [Describe how client authenticate server]

There are several occurrences when the program ends (the flow chart of the program is provided in the next section of this report):

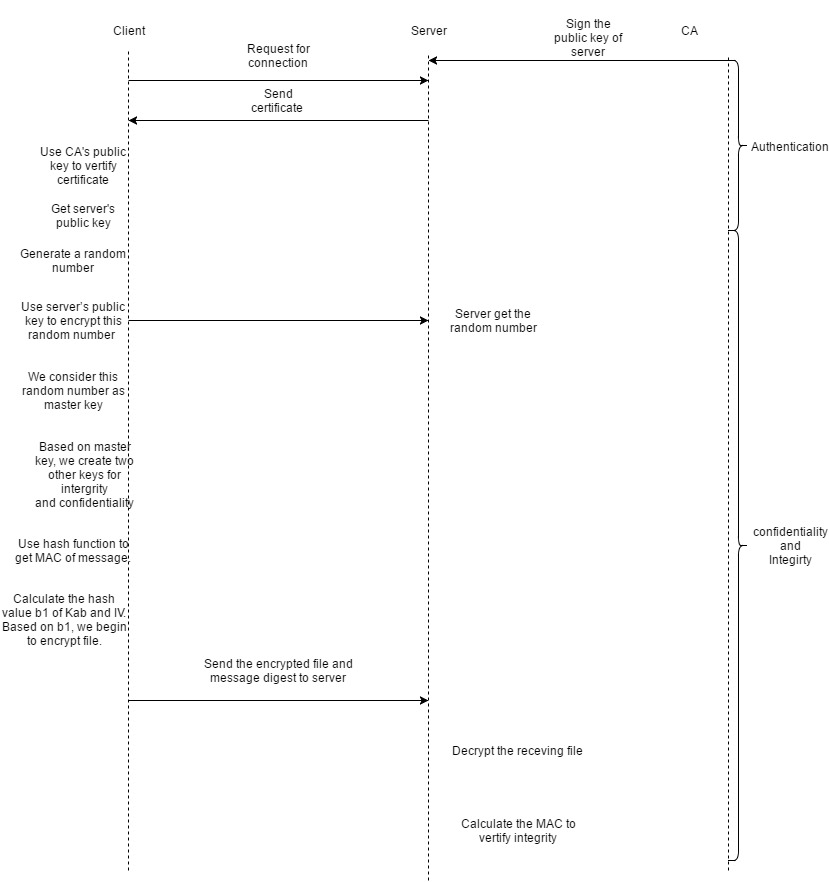
* If the server waits too long without any connection from the client, the program ends at that point.
* If the client tries to establish a connection without an active server (on the other word, the client starts before the server does), the program ends immediately.
* After the connection is established, they can continue communicating (the client starts first) until the client says “quit”. When this event occurs, all sockets are closed and the program ends.
* During the session, if either the server or the client goes offline for any reason (indicating by a null message received by the other party), the one who remains online displays an error message to the screen to indicate the event and the session ends safely (without crash).

**II. Implementation**

*1. Abstract Flow Chart*

*Basic flow of program. Authentication mechanism will be added as separated diagram.*

*2. Security Protocol*

**

**III. Code Executions**

*1. Configurations:*

Before running the program, user has to make sure the client and server are on the same network. If they are on different networks, the server machine has to setup port forwarding before it can operate with a remote client. Also, if there is an error saying “Socket corrupted” when the Client tries to connect to the Server (and the Server is running), then it is a Firewall configuration problem 🡪 please check Firewall to enable communication on your current type of network.

For the project to be successfully compiled and run, its structure must be preserved. The core modules structure is ([] denotes directory):

[] Client

|----- [] FilesDirectory

|----- [] src

|----- CAPublicKey.txt

|----- Client.java

|----- FakeClient.java

[] Commons

|----- [] src

|----- AESf.java

|----- InvalidMessageException.java

|----- Message.java

|----- Peer.java

[] Server

|----- [] FilesDirectory

|----- [] src

|----- CA-Certificate.crt

|----- CAPublicKey.txt

|----- FakeServer.java

|----- PrivateKey.txt

|----- PublicKey.txt

|----- Server.java

*2. Execution:*

There are several ways to run the system.

a. To run program via IDE:

Open Server and Client projects on separated windows.

Then individually click "Run" to run each code in parallel.

(May have to configure classpath to successfully compile and run)

b. To run program in the command line environment:

UNIX-like environment:

Open terminal and navigate to Server/src folder

Type: javac -cp ..\..\Commons\src Server.java

Type: java -cp ..\..\Commons\src:. Server

Open another terminal and navigate to Client/src folder

Type: javac -cp ..\..\Commons\src Client.java

Type: java -cp ..\..\Commons\src:. Client

Windows:

Open cmd (NOT PowerShell) and navigate to Server/src folder

Type: javac -cp ..\..\Commons\src Server.java

Type: java -cp ..\..\Commons\src;. Server

Open another cmd and navigate to Client/src folder

Type: javac -cp ..\..\Commons\src Client.java

Type: java -cp ..\..\Commons\src;. Client

c. To run program as jar files:

To be added later

**IV. Detail About Security Protocol**

*1. Authentication*

Server has a certificate which is its public key signed by CA’s private key. The client has CA’s public key. When server receiving the connection request from client, server will send the certificate to client. Client uses CA’s public key to verify the certificate and get server’s public key. After authentication, client generates a random number. We consider this random number as a master key.

Encrypt master key with server’s public key and send it to server.

*2. Confidentiality*

In this project, we use Sha1 to encrypt our files which is generating and using one-time pass to XOR with data.

b1 =Sha1(Kab|IV) c1=p1⊕b1

b2 =Sha1(Kab|c1) c2=p2⊕b2

… …

bi =Sha1(Kab|ci-1) ci=pi⊕bi

*3. Integrity*

We also use Sha1 to calculate the MAC(Message authentication codes).

MAC =SHA1(Kab|m)

**V. Threat Models**

*1. Eavesdrop*

Even if attacker can watch the message between client and server pass over the network, the attacker still can’t learn the contents of message between client and server.

*2. Initiate a Conversation Pretending to be Client*

The project has one-way authentication, so it can’t verify the client.

*3. Lie in Wait at Server’s Network Address and Accept a Connection from Client*

The server will use certificate signed by CA to verify itself.

*4. Read Server’s Database*

The attacker can get server’s private key to decrypt the master key from client. But since each session has a new key, and the master key is not used for encryption/decryption, stealing server’s private key does not help attacker to decrypt the captured messages.

*5. Man-in-the-Middle Attack*

The attacker cannot get useful information if he doesn’t know all the keys (server’s private key and the unique sessions’ keys).

The attacker can steal a message and does not forward to a party, in which case after sometimes both parties will end the connection due to timeout. Even though each party cannot do anything to prevent such an event, they still can detect the message has been stolen.

The attacker cannot successfully execute replay attack because each message is marked with a sequence number. If the receiving message does not have the expected sequence number, it will be dropped by the server/client.

The attacker cannot tamper with the message without being detected because each message is appended with a checksum and is signed by the sender.