BAO NGUYEN – BCN140030

MOFAN LI – MXL162930

CS 6349.001

**FILE TRANSFER PROJECT DOCUMENTATION**

**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 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 7 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 is able to view the files it contains, to 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.

Please read the included *READ\_ME.txt* file for more information about program execution.

**II. Security Protocol**

*1. Authentication*

Authentication mechanism utilizes both Public Key Crypto and Secre Key Crypto. 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 string (master key). They also know a pre-shared a random 69-character long secret key that contains all possible character of any message. (In this case, we generate a random string and hard-coded into the source code – see *AES.java*).

Details as follow:

Client:

Create unique ID based on its creation time.

Create random initial master key R.

Create encryption key E = R + 1 and Signature key S = R + 2

In each session from 1 -> n, update: E = E + 2 and S = S + 2

Session 0:

Request CA Certificate to verify if valid Server

Encrypt AES language using certified server’s public key

Encrypt R and ID using AES then send all

Session 1 -> n:

Encrypt ID using new E to get ID’ and send

Receive encrypted confirmation, decrypt and verify

Server:

Create encryption key E and signature key S

In each session from 1 -> n, update E = E + 2 and S = S + 2

Session 0:

Send CA Certificate

Receive encrypted AES language L’

Decrypt L’ using its own private key to get L

Set AES language using L

Received encrypted R, ID

Decrypt R’ and ID’ using AES

Set E = R + 1, S = R + 2

Session 1 -> n:

Decrypt ID' using new E to get client's ID

Compare with stored ID to authenticate

*2. Confidentiality*

We use SHA1 to encrypt our files which is generating and using one-time pad to XOR with data. The one-time pad shared key is generated using polyalphabetic cipher chosen at random from the character set (see more detail in the *AES.java* file). This gives the encryption unbreakable without knowing the key, since the characters is random, different from each session, and yields no statistical relationship to the language frequency.

File encryption:

b1 =SHA1(Kab|IV) c1=p1⊕b1

b2 =SHA1(Kab|c1) c2=p2⊕b2

… …

bi =SHA1(Kab|ci-1) ci=pi⊕bi

String encryption:

Polyalphabetic cipher.

*3. Integrity*

We also use SHA1 to calculate the MAC (Message authentication codes), such that:

MAC = SHA1(Kab|m)

**III. Threat Models**

*1. Eavesdropping*

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 because they are encrypted with 1-time pad keys.

*2. Initiate a Conversation Pretending to be Client*

The project has one-way authentication, so the server cannot verify the client. It assumes the first client who successfully connects is the legitimate client.

*3. Break-in Attack*

The attacker can break into server’s machine and get its 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.

*4. Man-in-the-Middle Attacks*

The attacker cannot get useful information if he doesn’t know all the keys (reason stated above).

The attacker cannot hijack the session using TCP hijacking because the authentication does not rely on IP address, but instead the content of the message.

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 recipient. Intrusion detection will notifies the client/server after receiving more than 5 out-of-sync messages, which causes them to terminate the connection. Moreover, the initial number of message sequence is chosen randomly by the client in the very first message, and the server just increment the message sequence onward. The attacker cannot simply counts the number of messages that has been sent to get the correct sequence number of a specific message. Therefore, the attacker cannot just create a message with his/her own sequence and send.

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

*5. Denial of Service Attack*

If the attacker send a certificate request to initiate the session, but then hangs and does not send anything else (tries to keep the server busy waiting for the next message), the server can detect the empty input from that connection. It then terminates the connection with the attacker and accepting new connection (as fresh start).

**IV. Contributions**

Bao Nguyen: Communication protocol, Secret key modification using poly-alphabetic cipher, Encryption & Decryption on Strings, Message sequence, Report, Intruder simulation – possible attacks

Mofan Li: AES Encryption & Decryption on file using Public & Private keys, Message integrity for file

*(\*) If for any reason you cannot run the project from the submitted zip file, please check out our project on Github at* [*https://github.com/baonguyen96/FileTransferApplication*](https://github.com/baonguyen96/FileTransferApplication)*. Here you can see the distribution break-down as well.*