Secure Chat and File Transfer

1 Introduction

In this assignment you will implement a client which performs secure chat and file transfer. We'll call the tool *SCAFT (Secure Chat and File Transfer)* for convenience in this description. As part of developing SCAFT you will learn how to use the *Advanced Encryption Standard (AES)* cipher with a 256 bit key and the *Counter (CTR)* mode for block cipher encryption. SCAFT will rely on TCP communication for chat as well as file transfer.

SCAFT lets the user perform two fundamental tasks. First, a user can set up a secure chat session between multiple SCAFT clients using encrypted application-level multicast (TCP conversations). Second, the user can send a file he chooses to one member of the group using a separate TCP point-to-point connection. In both cases, all communications will be encrypted with AES using CTR mode. SCAFT communicates in a *peer-to-peer manner*, meaning that there is no server or central point of communication or coordination between the tools. A detailed user story and specific technical requirements are given below. Each SCAFT client is configured with the following parameters:

1. A user name. A user name is a string consisting of printable UTF8 characters which will fit in Java String. The user name **may not** contain whitespace characters or the '@' character.
2. The list of neighbors in the group - each neighbor has an IP address and port.
3. A text password. The password is converted to an AES key by:
   (1) Converting the password to bytes using the UTF8 encoding
   (2) Hashing the resulting bytes with SHA2-256 hash algorithm
   (3) Use the 256 output bits as the key for the AES cipher

2 Main User Story

Alice, Bob, Claire, and Dan want to form a shared secret chat group using TCP communication and AES encryption. They also want to be able to send files to each other with encryption to prevent outsiders from easily revealing their contents. To set up the group, Alice, Bob, Claire, and Dan meet up and exchange their home IP addresses and free port numbers (they may not all have the same port free on their home router) and a shared secret password.

After the parameters have been agreed upon, each person goes home, turns on SCAFT, and enters in the configuration information: neighbor list (IP addresses/ports) and shared password. Alice enters in her user name (*alice*) and connects to the group. When she connects, her SCAFT sends a HELLO message to everyone in the multicast group with her user name (*alice*). Bob is already connected to the group and once Alice connects and sends her HELLO, Bob sees her added to his list of *connected users*. Bob's SCAFT sends a HELLO message back to Alice, so she quickly sees Bob's username (*bob*) in her list of connected users.

Alice and Bob chat for a while using SCAFT. Each message that Alice sends is encrypted using AES and the shared password using CTR. For security, each message is sent using a different *random 128-bit initialization vector (IV)*. To ensure that the recipient can decrypt the message, the message's IV is added to the beginning of the message.

Soon, Claire decides to join the group using her SCAFT. Once she connects, her SCAFT sends out a HELLO message with her user name (*claire*) to everyone on the list (only Alice and Bob are currently online), so she is added to the connected users lists of Alice and Bob. Alice's and Bob's SCAFT tools send HELLO messages in response to Claire, so Claire sees Alice and Bob's user names in her list of connected users. Dan soon joins the group too, sends a HELLO, is added to the connected users list on Alice's, Bob's, and Claire's SCAFT tools, and sees all three of the others in his list.

From then on, every message sent by any member of the group is received by all the other members' SCAFT tools, decrypted, and shown on the screen. The SCAFT screen shows each message sent from the time the user joined the group, including the name of the person who sent the message, the time it was sent, and the content of the message. This lets the users see who wrote what in the past and when.

Alice decides to send a private file (`cool-cat-photo.png`) to Claire, a file whose contents aren't of interest to Bob and Dan. Alice and Claire will use the shared password as a key, so the file could in theory be read by Bob and Dan if they would intercept it. Alice and Claire trust Bob and Dan fully, so they are not concerned about that. Alice chooses Claire's user name (*claire*) from the list of connected users, selects a file to send, and her SCAFT sends a SENDFILE message to Claire's SCAFT. Bob and Dan don't see the SENDFILE message on their SCAFT. Since it's meant for her, Claire's SCAFT shows the SENDFILE message and asks if she wants to

accept the file `cool-cat-photo.png` from the user name *alice*. Claire agrees and her tool sends back an OK message to Alice's SCAFT. Once Alice's SCAFT receives the OK, the two SCAFTs set up a separate TCP connection on a randomly chosen port. Alice's SCAFT encrypts the file with the shared AES key and a new random IV and sends it over the TCP connection. Once the file finishes arriving at Claire's side, her SCAFT decrypts the file automatically and opens up its location for her to see the cool photo. Claire receives a notice when the file has finished arriving. Alice also sees a message indicating that the file completed being sent to Claire. If Claire declines to receive the file, her SCAFT sends back a NO message and the file transfer isn't performed. Alice sees a message about Claire's refusal.

When Dan decides to leave the chat group, he clicks "disconnect" on his SCAFT. His SCAFT first sends a BYE message to all connected members in the group and then disconnects. When the BYE messages reach Alice, Bob, and Claire's SCAFTs, Dan's user name is removed from the connected users list.

3 Requirements

Aside from implementing all elements of the user story above, your tool must meet the following requirements:

1. The tool must be written in Java.
2. The tool must use the Java Crypto APIs or the BouncyCastle cryptographic library.
3. The tool must have a JavaFX based graphical user interface (GUI)
4. The tool must let the user configure the neighbor list (IP address/port) and the shared text password using the graphical user interface.
5. The tool must not artificially limit the number of neighbors possible.
6. The configuration must be done using the GUI or in a dedicated configuration file which can be edited by the user at any time.
7. It must be possible to change the configuration information and use the tool with the new information without needing to close and restart the tool.
8. The tool must let the user choose to login to the group, causing the tool to connect to and send HELLO messages to all available neighbors in the group.
9. The tool must let the user choose to logout from the group, causing the tool to send BYE messages and disconnect from all currently connected neighbors in the group, but leaving the tool still open (*i.e.* logout does not cause the tool to quit).
10. The tool must let the user login and logout from the group as many times as the user wants without the need to close and restart the tool.
11. The tool must show a list of connected users, including the user name and IP address of each other user.
12. The tools must work **without** a central server or other centralized point of coordination.
13. The chat conversations must be based on TCP application-level multicast.
14. File transfer must be done using TCP point to point communication.
15. The tool must let the user select a recipient from the connected users list, a file to send, and send the file to the recipient.
16. The tool must ensure that all messages sent (*i.e.* HELLO, BYE, SENDFILE, OK, NO) are encrypted using AES in CTR mode.
17. Each message or file must be sent using a different random 128-bit CTR initialization vector (IV).
18. The IV for each message and file sent must be chosen randomly by the tool without user intervention.
19. The tool must catch all communication, file processing, and input/output errors.
20. The tool must not crash due to uncaught exceptions.
21. When one user sends a file sending message to a second user, the second user must have the choice toaccept or decline the file.
22. When a user chooses to accept a file, the user is given a chance to choose the directory where the filewill be saved. The file will be saved with the same file name as the sender used. If a file with the same name exists in the chosen directory, the user will be

asked if he wants to overwrite the file. If he does, the file will be overwritten. If he declines, the user will be given another chance to choose a different directory.

23. The tool must be multithreaded, allowing multiple concurrent file transfers, file receiving, and chattingat the same time.
24. **Do not** hard code absolute file paths, encryption keys, initialization vectors (IVs), IP addresses, or ports into the program source code.
25. The tool must keep a log of all messages sent and received in the format described below.
26. The log file must be stored in the working directory of the tool.

You may design your own interface, message formats, and message fields as necessary to fulfill the task.

3.1 Log File

The log file must contain all messages sent and received by the tool, even ones that are not shown on the screen due to errors or bad keys. The following fields must be recorded for all messages:

- The time and date the message was sent or received
- The IP address and port of the sender of the message
- The user name of the sender of the message
- The type of the message (*e.g.* HELLO, BYE, OK, NO, MESSAGE, SENDFILE).
- The decrypted contents of the message (including any message type or header)
- The IV of the message in hexadecimal notation

3.2 On Debugging

Keep in mind that when testing SCAFT you will likely need more than one computer for testing the group communication. You can use localhost for testing provided that you put each node on a different port, but the real test of the file transfer will be between computers. When testing, you may want to use the Wireshark packet sniffer tool to look at the packets sent of the network. Wireshark will help you track down sending and receiving errors and can give you more confidence that the packets are actually being sent with encryption.