CIS 457 Project 3: TCP Encrypted Chat Program

Objective: Implement a Multi-client/server chat system. Learn how to encrypt/decrypt in code.

Deliverables: You must turn in your code on blackboard on the part two due date. Additionally you must turn in your documentation by the next lecture after the part two due date. You must demo your client and server meeting part 1 requirements on the part 1 due date in lab. You must arrange a time to demo the complete project after the part 2 due date.

Groups: This project may be done alone, or in groups of up to three students.

Grading: This project is worth 100 points (30 for part 1, 40 for part 2, 20 for part 3, and 10 for documentation), as described below.

Specifications

You may write this program in C, or C++, or Java. Other languages may be allowed by request. You may not use any non-standard libraries without prior permission. Your program must work either on the computers in the Datacomm lab or on mininet. You may program it wherever you like, but must demo it in one of these environments.

Your assignment is to write both the server and client parts of a chat program. You must use TCP sockets. The messages you send must be encrypted, using AES in CBC mode. For the purpose of encryption you should use the OpenSSL libcrypto library (C), or the classes in javax.crypto (Java). Use of these libraries will be explained in lab at the start of the second week of this project. If you are not sure which language to choose, consider that the Java cryptography libraries are somewhat simpler than the C/C++ ones.

Part 1

Due: Tuesday, November 29th

For part 1, you must support the following functionality:

- The server should allow for multiple simultaneous clients (you will likely need to use select or threads for this).

- Each client must have a username, which will be used in the client list and to specify which client to send a message to.
- The server should provide a way for the clients to get a list of all other connected clients.
- The server should support both clients sending to individual other clients, and clients sending to all other online users simultaneously.
- The client should be able to send message to the server to request the usernames of all other connected clients, send messages to other individual users, and send broadcast messages to all users.
- Administrative commands must work as described below

These administrative commands must only be available to an administrative user (except the command to become an admin). It is up to you how to distinguish these commands from chat messages. The specific commands you must support are:

- **Admin** - become an administrator. This command must be password protected. Input of the password must not be shown on the screen (if programming in C, and using terminal input, you can use the ECHO flag from the termios library for this; if programming in Java and using System.console for input, you can use the readPassword method).
- **Kick** - kick off another user. You must program this in a way where the server disconnects the other user regardless of if the other client cooperates. You must program the client in a way that when kicked, it detects that it is disconnected and handles it gracefully (does not wait until you try to send a message to know it is disconnected, does not go into an infinite loop of printing, etc).
- **?????** - Any other command of your choice. Be creative. If you don't want to be creative, some suggestions: Rename a user; make someone else an admin; silence a user (so they can get messages but not send them)

Part 1 Grading: In part 1, there will be a total of 50 points, divided as follows:

Criteria	Points
Multiple client connections	5
Broadcast message (to all clients)	10
Individual message	10
Client list	5
Admin command	5
Password not printed	5
Kick command	5
????? command	5

Part 2

Due: Wednesday, December 7th

For part 2, you must encrypt all messages being sent from client to server, and server back to client. Since all messages between two clients go through a server, the server will need to decrypt and then re-encrypt each message. Please refer to the sample cryptotest.java and cryptotest.c files for help in how to encrypt and decrypt in your programs.

The C version needs a RSA public and private key in PEM format. These can be produced with the commands:

- `openssl genpkey -algorithm RSA -out RSApriv.pem -pkeyopt rsa_keygen_bits:2048`
- `openssl rsa -pubout -in RSApriv.pem -out RSApub.pem`

The Java version needs the keys in DER format. You can produce the correct keys with the above commands plus:

- `openssl rsa -inform PEM -outform DER -pubin -pubout -in RSApub.pem -out RSApub.der`
- `openssl pkcs8 -topk8 -nocrypt -inform PEM -outform DER -in RSApriv.pem -out RSApriv.der`

If you are using Java, you may not use cipherstreams.

When clients join the chat server, they need to securely establish a symmetric key pair with the server. For this purpose, the server should have a public/private key pair for use with RSA. To establish a symmetric key, the client should randomly generate one, and then send it to the server encrypted with the server's RSA public key. The server then should decrypt it using the RSA private key. All subsequent messages should be sent encrypted with this symmetric key.

A random initialization vector is used in encryption to ensure unique encryptions of identical messages. You must properly generate a new IV for each message.

Part 2 Grading: In part 2, there will be a total of 40 points, divided as follows:

Criteria	Points
Randomly generate symmetric key	5
Encrypt symmetric key with RSA pub key	10
Decrypt symmetric key with RSA private key	10
Encrypting all chat messages with symmetric key	5
Decrypting all chat messages with symmetric key	5
Correct use of initialization vector	5

Documentation

Due: Friday, December 9th

Documentation of your program is worth 10 points. This should be a 1-3 page document describing the design of your program. This should not be a line by line explanation of your code, but should explain the overall structure and logic of the program, as well as what major challenges you encountered and how you solved them.