2 Secure chat application

You are provided with a simple chat application for two users with a central server. The server stores the usernames and passwords in clear. The clients send their login/password to authenticate and further conversations in clear to the server. Finally, the clients store the chat history in plaintext on disk. We refer to this architecture as the *plaintext chat*. Obviously, this design has many flaws. You will address individual problems in the questions below. For each question, create a separate project that reuses the provided codebase, which implements the plaintext chat. Eventually you will submit three (3) projects.

2.1 Plaintext passwords on the network

a) From the original plaintext codebase, we want to prevent man-in-the-middle attacks from passively listening to plaintext passwords. We also want to avoid the possibility for an attacker to impersonate a user by replaying the traffic sent by that user during authentication. Modify the plaintext codebase to achieve this goal.

b) Additionally, we want to protect the chat history logged by the client so that if a malicious user manages to copy the chat history file, she cannot read the history on another computer. The mechanism needs to be bound to the machine where the history is stored. Implement this feature.

2.2 Plaintext passwords on the server

a) Restarting from the original codebase, we want to avoid the server storing plaintext passwords. In particular, we want that an attacker who obtains a copy of the server's database will spend significant resources to recover the users' passwords. We also want to prevent attackers from using precomputed tables (e.g., rainbow tables). Modify the plaintext codebase to achieve this goal.

b) Additionally, we want to protect the chat history logged by the client so that it is not possible to read the history without login to the server. The server needs to send the history encryption key to the client after a successful authentication. Further, if the server's database falls into an attacker's hands, we want to make it as difficult for the attacker to decrypt a stolen chat history as impersonating a user to the server. Implement this feature.

2.3 Certificate-based authentication

As it seems difficult to combine the two previous protections, the administrator of the system concluded that passwords are not secure enough to authenticate clients, and decided to move for certificate-based authentication.

2.3.1 Authentication

Provide the command lines for OpenSSL that you use for these questions.

a) Using OpenSSL, generate a root certificate "MyRoot" with an RSA-4096 key, self-signed using SHA256.

b) Using OpenSSL, generate certificates and private keys for Alice, Bob and the central server, issued by your "MyRoot" certificate. These entities should use RSA-2048 keys.

c) Replace the use of passwords in the server with public keys from Alice and Bob.

d) Modify the clients to load user certificates and private keys to authenticate to the server. Clients are expected to know the server's public key. Replay attacks should not be feasible.

2.3.2 Encrypted traffic

Now that the authentication is taken care of, we need to keep the actual chats confidential.

a) Make the clients and server encrypt exchanged messages with the appropriate recipient's public key. Note that messages need to go through the server and clients do not know each others public keys.

b) In the question above, the length of the encrypted messages is bound by the key size. Negotiate symmetric keys between clients and the server to further encrypt sent and received messages.

c) Forward secrecy: Imagine that the private key of the server is compromised, all previously sent encrypted messages can be decrypted. Show how an attacker would decrypt past recorded traffic between a client and the server. Implement forward secrecy to prevent this type of attack.