Lab 4 : Client-Server Chat Application

Advanced Network Security

Paul Aguilar

Winter ‘26

Abstract

This project implements a basic client-server chat application to demonstrate the implementation of the Diffie-Hellman (DH) Key exchange protocol. This is done using several Python libraries, including the socket, threading, math, and tkinter libraries. Tkinter is used to create a basic graphical user interface (GUI) for the application, which allows users to select a username, connect to the server, request the secure key exchange, and exchange messages. Along with the DH key exchange, an RSA cipher is also applied as a digital signature to verify the authenticity of the public values sent between both parties.

1. Application Layer: The Message Protocol

The fundamental design of the messaging protocol for this application follows a "Header | Payload" architecture, in which the first segment dictates the routing or action to be taken. The system uses a pipe-delimited string protocol over a persistent TCP connection and handles 3 separate types of communication within the application. These include: a standard instant messaging conversation between two clients without any security/key exchange having taken place. System messages are also handled, which notify the user of system functions, e.g., welcome messages when connecting, key exchange initiations and status, and several warnings/errors regarding the connection. The third and final message type are security handshakes, which relate specifically to the DH key exchange process and are used to initiate relevant steps in the algorithm, including the initial request/initialization between both clients, the sharing of the public numbers, the generation and dissemination of the public value and the resulting shared secret that is passed between both clients.

Packet Structures

Standard Chat: Recipient|Message_Body

System Messages: System|Command_or_Update

Security Handshake: Recipient|MSG_TYPE|Math_Payload

The client handles the dispatching logic for each message based on the message index. Index 0 identifies the source of the packet, identifying the sender of a message for display on the GUI. Index 1 specifies the type/payload of the message. Specifically, whether or not any security actions are required.

Routing Logic

The server acts as a transparent **Identity-Based Router and maintains** a mapping of username and socket. When a packet is received, it splits the header using the pipe characters as delineation. It then strips the recipient header and prepends the sender header in the message to ensure that the receiver always knows the source of the message, freeing the sender from self-identifying in each payload.

2. Cryptographic Protocol: Authenticated Handshake

As a result of the DH key exchange implementation, a station-to-station type exchange was enacted, which prevents Man in the Middle attacks through the RSA signatures. The Diffie-Hellman exchange RFC (3526) was used to decide the standard public number to use. For the sake of simplicity and connectivity, the 1536-bit MODP group was chosen, as issues with public numbers larger than the RSA's hash resulted in unverifiable signatures and keys.

3. Key Exchange Workflow

The security is handled in a three-step asynchronous process within the Security_Manager class. During this process, an initiation message is sent (DH_INIT); this is the first part of the handshake when one client requests to begin the process. A transient DH private key is generated, and a public value is computed from it. This value is signed with the RSA private key and sent to the recipient. This response (DH_RESPONSE) begins the verification process for the recipient. First, the RSA signature is verified using the public key that is sent at the same time as the public value. If the signature is authenticated, then the shared secret is computed, and the recipient's public value and public RSA key are sent back to the sender. Finally, the same process repeats, validation then computation, until the sender also has the same shared secret, without it needing to be sent over the network.

4. Network Implementation: Transport Layer

The network implementation is handled by the socket library using the TCP protocol (AF_INET). This, in tandem with the threading library, effectively implements a multi-thread TCP server. Each client is managed by a client handler thread, which prevents any single users' process from blocking another. Asynchorny is also implemented using a receive loop on the client side, which allows for security updates to continue in the background. As a result, since the security handshake is handled in the background, the GUI remains responsive, though some slow down is noticeable at the initial connection due to the size of the RSA key required.

5. Security Considerations

Two main security concepts are implemented as a result of the DH key exchange and RSA signatures. The first of these being forward secrecy. This is the concept that no party is able to decrypt past chat sessions in a conversation, even if a key was recovered, because new keys are exchanged at the start of every session, by the users's choice. The second consideration would be identity binding. This is not as beneficial as forward secrecy, however due to the nature of the RSA signature implementation. Including the RSA public keys in the handshake binds the DH exchange to a specific and verified identity, which does contradict the purpose of end-to-end encryption, and these security protocols designed to protect and encrypt communications.