Secure Chat

Professor Gofman

Submitted by

Ryan Dencker - dencker@csu.fullerton.edu
Brandon Nguyen - nguyen.brandon771@csu.fullerton.edu
Brodan Whelan - bwhelan212@csu.fullerton.edu
Edgardo Arteaga - edgardoarteaga@csu.fullerton.edu
Hart Zhang- hart224444@csu.fullerton.edu
Ricky Truckner - rickytruckner@csu.fullerton.edu

Presentation Video

https://youtu.be/gWX44kYBMBA

Github

https://github.com/ryandencker/Secure-Chat

Department of Computer Science,
California State University, Fullerton

Table of Contents

Abstract

In this project, we are to implement a system that enables a group of users to chat securely. All users are registered with the chat server. When the user wants to chat with another registered user, he first connects to the chat server and enters his/her username and password. The server verifies the user name and password, and if correct, the user’s status is changed to “online“. Next, the user may enter the user IDs of users with whom he wishes to chat (could be more than one). At any given time the user should be able to check what other users are online and invite them to the ongoing conversation.

Introduction

This project aims to develop a secure chat that ensures confidential and authenticated communication among users. Users register with a central chat server and authenticate themselves with a username and password which are stored as secure hashes. Upon successful authentication, their status changes to "online", allowing them to initiate chat sessions with other online users. The server generates a symmetric key for each chat session, securely distributing it to all participants by encrypting it with their public keys. This ensures that only the intended recipients can decrypt the symmetric key and participate in the chat.

Design

The secure chat system ensures robust user authentication, secure key distribution, encrypted communication, and digital signature verification. Users register and authenticate with a central server using unique credentials, and their status is managed as online or offline. To initiate a chat, a user specifies online participants, and the server generates a symmetric key(AES) for the session, encrypting it with each participant's public key. Messages are encrypted with this symmetric key, ensuring confidentiality, and are broadcast to all participants. The system supports RSA and Digital Signature Algorithm (DSA) for digital signatures, allowing users to ensure message integrity and authenticity. A public key infrastructure (PKI) maintained by the server facilitates secure key distribution and signature verification. The user interface provides easy access to registration, login, initiating chats, and checking statuses, with real-time notifications about user availability and message delivery.

Security Protocols

The secure chat employs multiple security protocols to ensure communications' integrity, confidentiality, and authenticity. Digital signatures, using both RSA and Digital Signature Algorithm (DSA), are implemented to sign messages, verifying the sender's identity and maintaining message integrity. Symmetric key encryption (AES) is used for encrypting all chat messages with a session-specific symmetric key, which is securely distributed to participants by encrypting it with their public keys. The authentication protocol requires users to log in with their username and password, allowing the server to verify their identity and manage their online/offline status. Additionally, the system provides real-time notifications and status updates, enabling users to see the availability of others, receive session invitations, and stay informed about changes in participant status. These combined protocols ensure a secure user experience.

Implementation

Our project was created using the Python programming language. Our program is separated into a client and server file which run their specific command and functions respectively. Our client.py file includes the socket and threading library in order to connect the client to the server and run certain processes concurrently (***Fig. 1***). Our server-side similarly used socket and threading to connect to client functionalities and run tasks concurrently (***Fig. 2***). We also use the library bcrypt in order to hash the user passwords and is considered one of the strongest algorithms for this purpose(***Fig. 3***). We use random to help generate random and unique user IDs (***Fig. 3***). Another library we use is the Cryptodome package which contains many low-level functions used for cryptography. We specifically used it for Hashing in which we SHA256 for digital signatures. RSA is used for session key distribution/exchange from server to clients and also for digital signatures(***Fig. 4***). DSA and DSS are also used for digital signature implementation in which we use them to generate keys and several other functions (***Fig. 5***). We used parallel implementation in order to complete this project. In which we separated certain functionalities and assigned them to individual team members to complete to be worked on concurrently. We initially tried creating a single file to create the application, but found separating our client and server side to be easier to code, debug, and just better industry practice. We also used github for our version control and Visual Studio Code as our IDE.

```
def start_client(server_name, server_port):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_name, server_port))
```

Figure 1. Client.py example of sockets library in use

```
def handle_client(client_socket, addr):
    try:
        accout_option(client_socket)
        command_handler(client_socket)
    except Exception as e:
        print(f"An error occurred with {addr}: {e}")
    finally:
        remove_from_online(client_socket)
        if client_socket in clients:
            clients.remove(client_socket)
        client_socket.close()
```

Figure 2. Server.py example of sockets library in use

```
  hashedPassword = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
  user_id = generate_unique_user_id()
```

Figure 3. Server.py example of bcrypt and random libraries in use

```
keys = RSA.generate(2048)
pub_key = keys.publickey().export_key()
priv_key = keys.export_key()
```

Figure 4. RSA from Cryptodome package

```
key = DSA.generate(2048)
pub_key = key.publickey().export_key(format='PEM')
priv_key = key.export_key(format='PEM')
```

Figure 5. DSA from Cryptodome package

Conclusion

In this project, we developed a messaging application to transmit messages securely between users. Our application enhances security by storing user credentials as hashed passwords rather than plaintext. During the login process, the server hashes the provided user credentials and compares them to the stored hashes to authenticate users. Once authenticated, the server generates public and private keys to distribute symmetric keys, which users then utilize to encrypt and decrypt their messages. This ensures that users can communicate securely without concern for unauthorized people to read their messages.