CPSC-352 Final Project

Spring 2024

Secure File Sharing

Dr. Mikhail Gofman

Nick Goulart

Katherine Joy Guardiano

Dylan Zuniga

Allen Dai

Andy Huynh

Abstract

This project is a secure peer-to-peer file-sharing system that uses concurrent server concepts. The system involves three peers connected to a network, each storing files associated with keywords that can be used to search for the file. An indexing server tracks file locations on which peers and manages peer accounts affiliated with the server. Peers authenticate with the server, update file information, and query for specific files. When a peer queries for a specific file, the peer chooses from the list that is returned and initiates a file transfer operation. Files are securely transferred between peers, ensuring confidentiality and integrity using digital signatures and custom key distribution protocols. The salient features include a concurrent server network architecture, user with ID and password, a keyword-based file search and retrieval, a secure file transfer method using public keys and digital signatures, and supporting the choice of RSA or DSA for the digital signature.

Introduction

The project itself is composed of two parts: a client and a server program. The server file contains functions for database management, message sending and receiving, connection management, account management and authentication, and file management. The client file contains functions for sending messages and receiving messages and querying search requests to the server. The server holds and handles a bulk of the functionality, while the client simply creates requests and sends data, much like a real-life implementation of a client-server-based system. Security is provided by AES encryption of session data, including both keys and messages. Nonces are used to provide session security and prevention of replay attacks. The server allows file upload and download, with the ability to query for specific files via use of keywords attributed to them on upload. It also allows for the registering of users in order to manage unique keys.

Design

The full system contains 4 core components: the client, the file transfer service, the server, and the database management system. The client acts as the part of the system that initializes file upload and download. It makes requests to the server regarding database contents. Any and all user input relevant to the operation of the file-sharing system is taken from the client side. Within the specific client file for the system involves the generation of the client's unique keys, client-side encryption, and handling incoming and outgoing packets. The file transfer service exists as a part of the server and is what allows the client and server to interact with one another for data exchange and management. It handles the initialization of the system, choosing server or client interactions, and some aspects of identity verification. The server acts as the part of the system that acts as the intermediary between the client and the database. It handles and verifies the client's requests for database content, acting as data management. It also secures passwords for storage in the database via hashing and the addition of a salt. The database is the part of the system that handles data storage, responding to the client's requests only when verified by the server itself. Its data management duties include the storage of password hashes, user credentials such as user IDs and keys, and file information such as keywords and contents.

[Please see end of documentation for diagrams]

Security Protocols

When the client connects to the server, they have the option to generate new RSA/DSA key pairs that will be stored in the database upon login/registration and later used in the peer-to-peer file transfer. When each peer acts as sender and receiver, this is where the actual key exchange starts. First, the sender and receiver send their public RSA/DSA keys across the network. Then, the sender generates an AES key. The sender then encrypts the AES key using the receiver's public RSA key. The receiver decrypts the encrypted AES key using their private RSA key. Then, the DSA and RSA signatures are generated by the sender which are then sent to the receiver alongside the AES encrypted file. Then finally, the receiver decrypts the file byte stream using the AES key, and verifies that all of the file data has not been tampered with in transit using RSA/DSA verification methods. If the verification fails, the file will cancel the download, and the program will end. If the verification succeeds, the file will download. This ensures both confidentiality and integrity of the file transfer process, making it robust against tampering and unauthorized access.

Implementation

The project was written predominantly in Python for simplicity and familiarity, with the importation of multiple different libraries to cover security and data management. Libraries imported include Cryptography, Pycryptodome, Pysqlite3, Tqdm, Prettytable, and Bcrypt. Both Cryptography and Pycryptodome are used for encryption, including key generation and utilization, RSA and DSA implementation, and digital signature implementation. Pysqlite is used to construct the database, allowing for the storing and indexing of files as well as user authentication information (usernames, passwords, unique IDs, etc.). The Tqdm library is used for user interface purposes, adding loading bars to show users progress on their actions, such as the uploading or downloading of files. The Prettytable library is used to create tables for the user interface, adding to the user experience. Finally, the Bcrypt library is used for hashing and salting passwords before they are stored in the database. The overall implementation ensures that the system is secure, efficient, and user-friendly, providing a reliable platform for secure file sharing.

Conclusion

Our secure peer-to-peer file-sharing system lets three peers connect to a network and reliably transfer files with each other. This was accomplished with the indexing server that will store the files with associated keywords, track file location, and manage accounts for the client. On the client's side, they will be able to connect and send search requests to the server. For the security of the system, each peer would need to authenticate with the server to be allowed to update file information and query specific files. Asymmetric cryptographic protocols are utilized to generate a secret AESkey between users that is used for secure communication and encryption. Digital signatures are employed with either DSA or RSA to verify that the files are coming from the expected sender and are not altered. The protocols used provide confidentiality, integrity, and authenticity. Using Python, implementing the secure-filling sharing system allowed libraries and frameworks to handle network connections, security, and database management efficiently.

There are plenty of things to take away from this project, but most notably, we have learned about concurrent server concepts, security protocols, and database management. This helps make our system reliable and operate at a sufficient level. These elements in the program provide a platform for peers to share files in a secure manner, where confidentiality, integrity, and authenticity are of utmost importance.

Relevant Links:

Presentation

https://www.youtube.com/watch?v=_aTffDwL2p8

GitHub repo

https://github.com/NickPrivate/Secure-File-Sharing

Diagrams and Figures

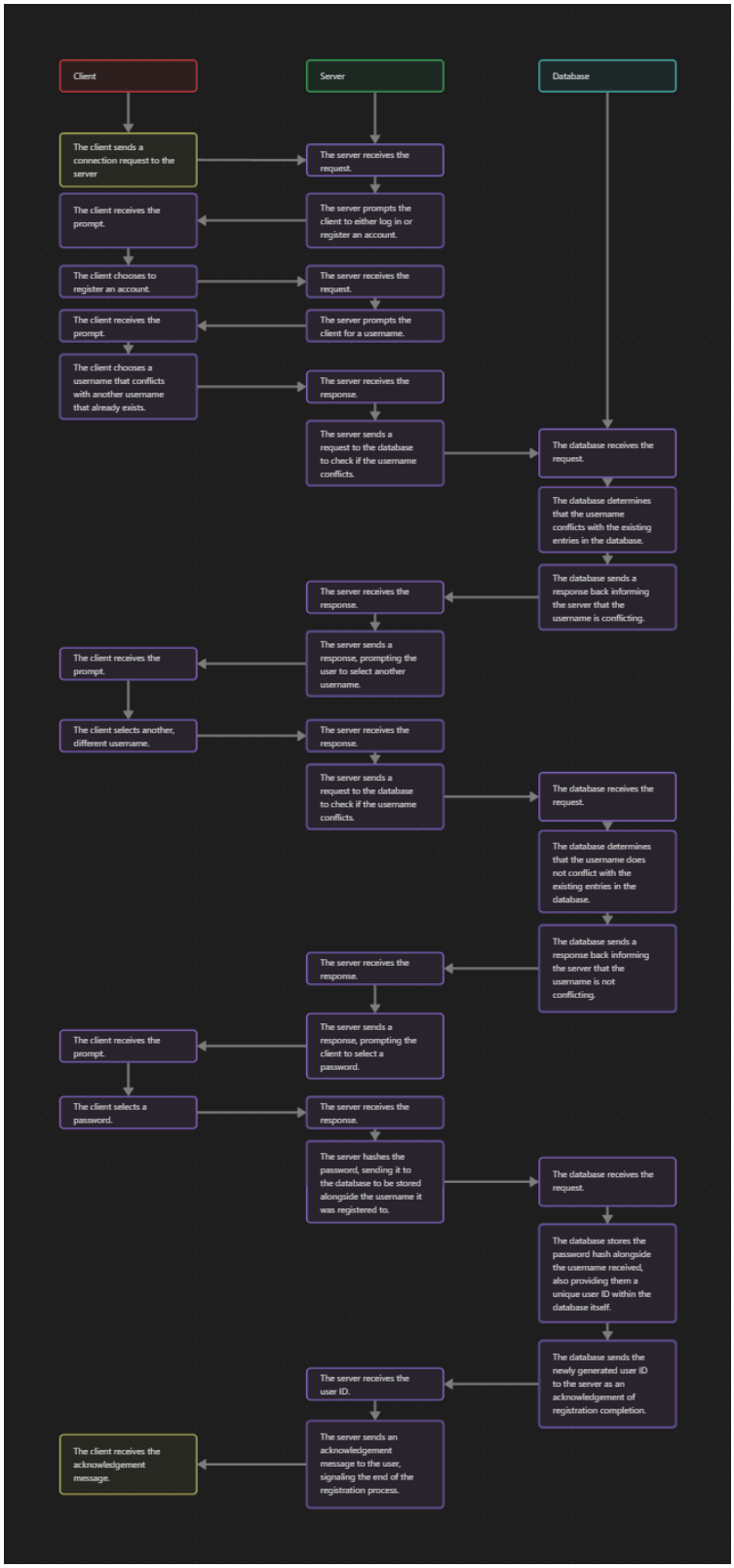

Figure 1, a depiction of a common use case for user registration.

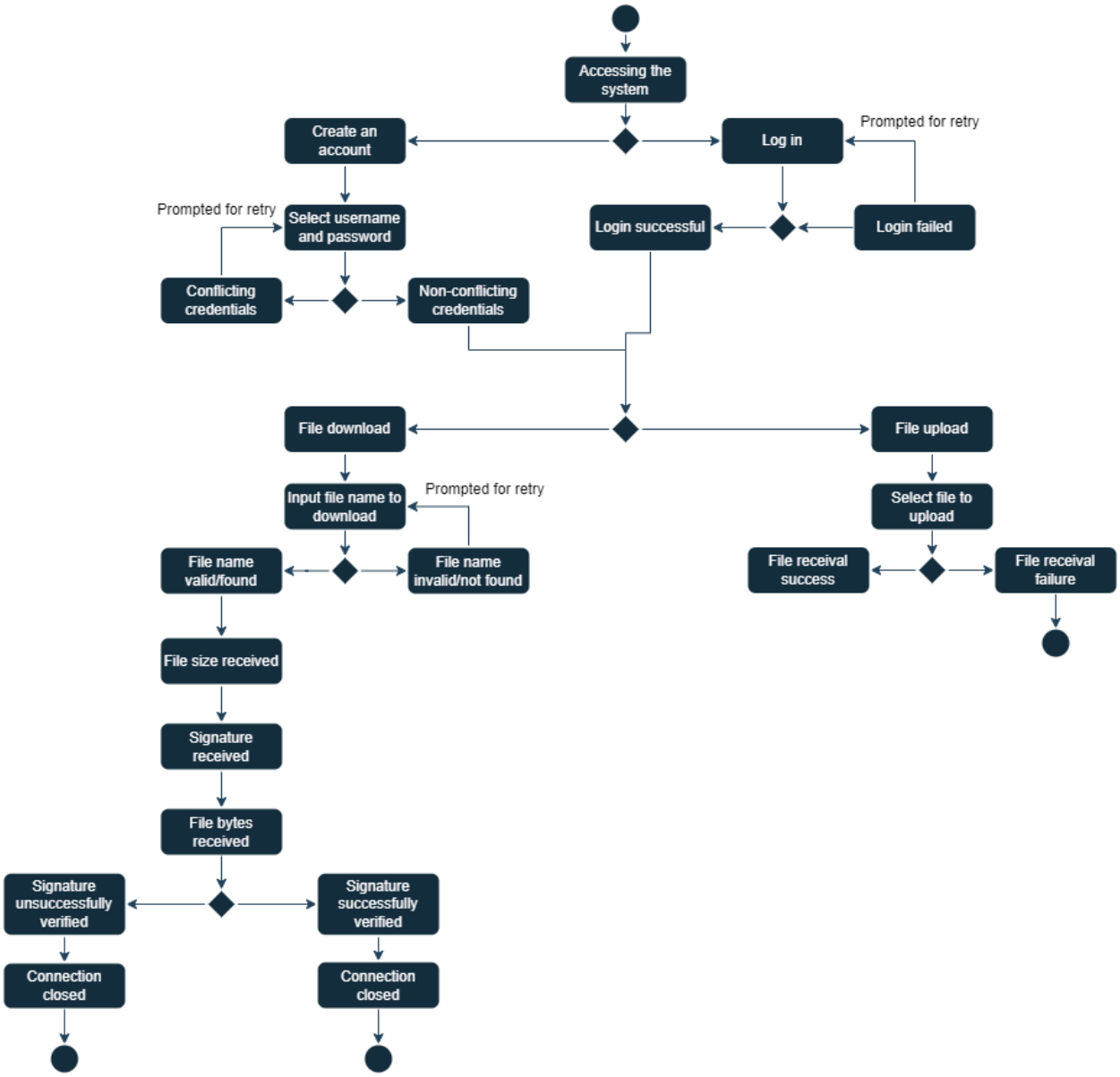

Figure 2, an activity diagram detailing possible user interactions.