Secure File System (SFS)

ECE 422 LEC B1 - Winter 2023
Reliable Secure Systems Design

Software Project
By
Ashutosh Lamba
And
Kyle Bricker

Abstract

The goal of this project was to implement a secure file system (SFS) that allows internal users to access, modify and create files, similar to a Unix file system. To increase security, all contents of the file system are encrypted with a master key held by the server, this protects user information and data from unauthorized external access. The system also utilizes encrypted communication with the client to ensure security from someone listening in.

Introduction

The goal of this project was to design and implement a secure file system (SFS) that allows its internal users to access, modify and create files, similar to a Unix file system but on an untrusted server. We assumed that the server we are storing the files to has external users who can access the file server and can access the user's files, keeping this in mind, our goal was to build a robust and secure file system that allows its users to safely store files on this untrusted server.

To achieve this, we start by encrypting all the communication between our server and client using the industry standard RSA encryption for the initial key exchange and then an AES encryption scheme implementation called Fernet so that anyone listening in on any communication would be able to decipher any information being sent between the client and the server. After this to ensure the confidentiality of the user's files and directories we use the same AES encryption scheme implementation to encrypt all the file/directory names and content when storing it in the file server, this ensures that any external users looking at the files and directories would not be able to read them, ensuring confidentiality of the files. Lastly we calculate and store an SHA-256 checksum for each file and directory to ensure the integrity of each file, this allows our system to detect and notify our users if any files or directories are modified without their permission. The user information, groups, file permissions and checksums are stored in the server on the filesystem but are also encrypted to ensure security and confidentiality.

Methodology

For our codebase we decided to write the entire project in python, using libraries such as sockets and cryptography to aid us. The implementation of the secure file system can be broken down into several parts, the implementation of the client and server architecture, the organization of the file system and the encryption techniques used.

Client and server architecture:

The client-server architecture was designed with a secure and robust connection in mind, for this, python's socket library was used. The server starts by creating a socket that listens on a specified host and port, upon receiving a client connection a thread is spawned to handle it. The server also spawns a thread to handle any instructions given to it by an admin, such as adding a user to a group. Once a client application starts up it connects to the server socket and a key exchange is performed, more information about this can be found in the encryption techniques section. After the key exchange is complete the client and server send messages to each other encrypted via a block cipher. The client provides the user with a command line interface allowing them to first create an account or login, and then it displays the users current directory within the system and allows them to make requests to the server. Correctness of commands given by the client are handled client side, meaning the client will only send commands to the server that it believes to be correct, this saves needless computation server side, as well as prevention of unexpected inputs for the server to handle.

Organization of file system:
The file system stores user and file information in its own utility folder called 'etc'. This folder contains 3 files:

1) Users: contains the usernames and passwords for all the signed up users. The usernames are encrypted using the python library Fernet encryption and the password is hashed using the bcrypt library before storing, bcrypt is irreversible and fernet guarantees that a message encrypted using it cannot be manipulated or read without the key.
2) Groups: contains the usernames and group names for all the users who are currently part of a group. Both usernames and group names are encrypted using fernet before storage.
3) Permissions: this stores information about all the files created by the users, it contains a realpath (encrypted because of filenames being encrypted), owner username (encrypted), permission (user/group/internal), and the latest SHA-256 checksum for the file.

Once the user is authenticated, they can use any of the commands outlined in the User Guide. The file system consists of a home directory, that cannot be directly accessed by the users, this home directory contains the home directories for all the users (named using their username but encrypted). Once the user is authenticated the server changes the current working directory to the user's home directory and then performs an integrity check on all the user's files by going though the users files and directories, calculating the checksum and comparing the current checksum to the stored checksum, the user is notified of any files or directories that fail the integrity check telling them that they have been modified by an external user. The file system uses OS commands and the utility files to perform all the file system functionality and updates the utility files accordingly, all

operations use the permissions file to check whether the user is allowed the action they want to perform and injection checks ensure that user is not able to navigate outside their home directory. If an external user created a file or directory, the user would be able to see them but not access them.

Encryption techniques:
A few encryption techniques are utilized in this system, first during the initial key exchange RSA encryption from python's library cryptography is utilized to securely send a symmetric session key that is then used for all communications between the client and server via fernet. For saving the passwords of users the python library bcrypt is used to hash passwords before storing them on the system. This means that even if an intruder gains access to the database and manages to decrypt it, they still will not be able to see the original passwords of users. The system database and files are encrypted by fernet which utilizes an AES encryption scheme, allowing us to use a master key for encryption and decryption. By default fernet incorporates a timestamp and random iv into its encryption algorithm, this would be problematic for our implementation relying on predictability of encryption, so these values were hardlocked when encrypting with the master key.

Design

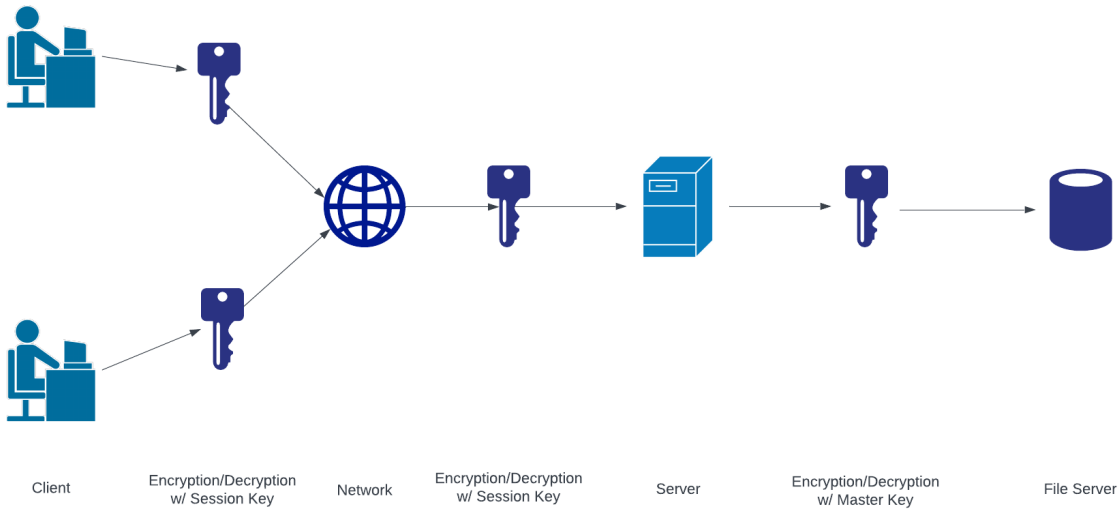

Figure 1: High-level architectural view

In Figure 1 we can see a high-level architectural view of the SFS system. Encryption via a session key is used to securely send information between the client and server. In order to read from the file server, the server holds a master key, which allows it to encrypt and decrypt data to and from the file server.

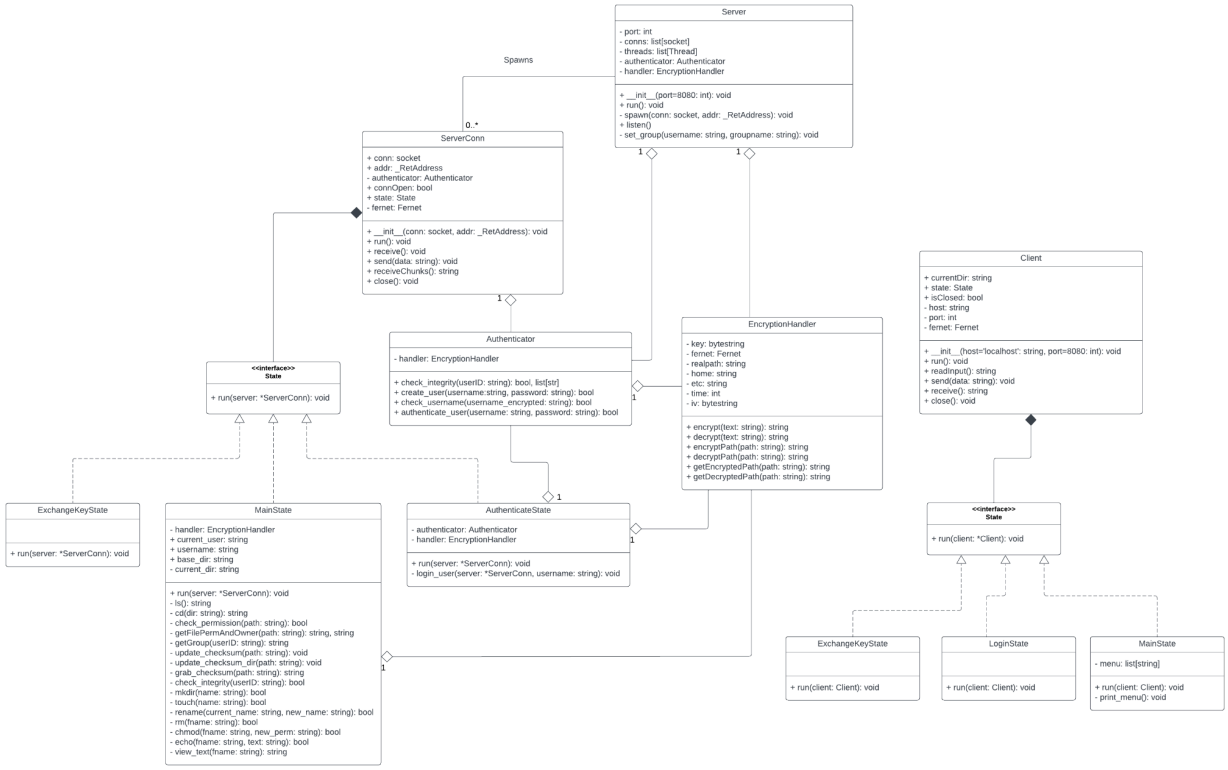

Figure 2: UML class diagram

Figure 2 shows the UML class diagram for our SFS system. The class diagrams can be separated into two sections, client and server. On the client side we have the main Client class which handles our socket connection, as well as IO with the user. Then we have three state classes which each contain the instructions for the client to run while it is in that state. On the server side we have our main Server class which handles IO with the admin as well as spawning ServerConn objects as threads to handle client requests. The Server object also contains an Authenticator and EncryptionHandler object which are passed down to its ServerConn object. The ServerConn object contains a socket which is directly connected to a single client, the ServerConn then operates in a State system similar to the Client, this is seen in the State interface and the objects implementing it.

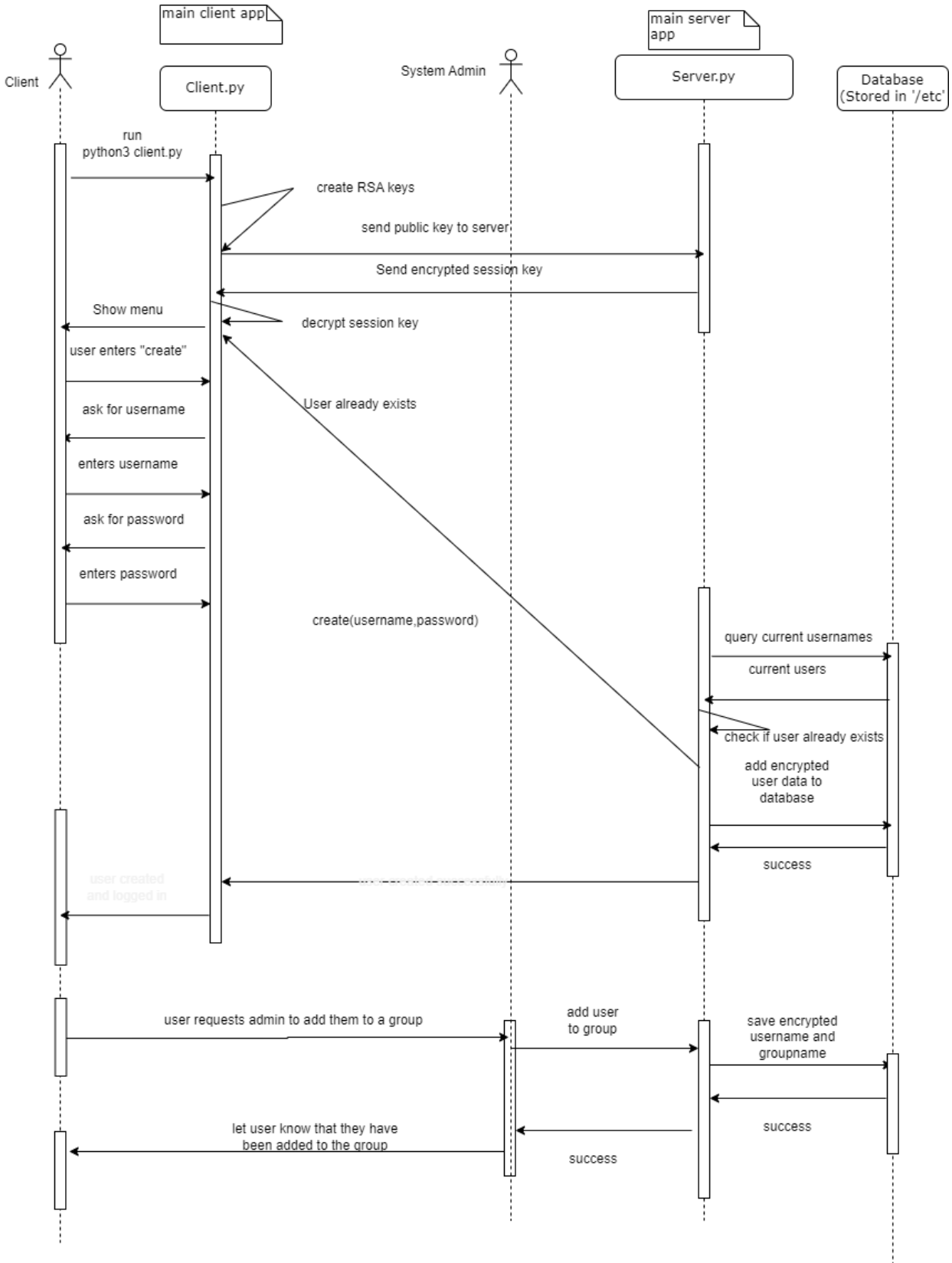

Figure 3: UML sequence diagram - create user and add group

The diagram above shows a sequence diagram for a user signing up on the SFS and being added to a group by a system administrator, we allow users to create accounts as long as someone else with the same username isn't already a user but we don't allow users to assign themselves to groups (as we consider it a security risk), that is handled on the server side and thus must be done by a system administrator.

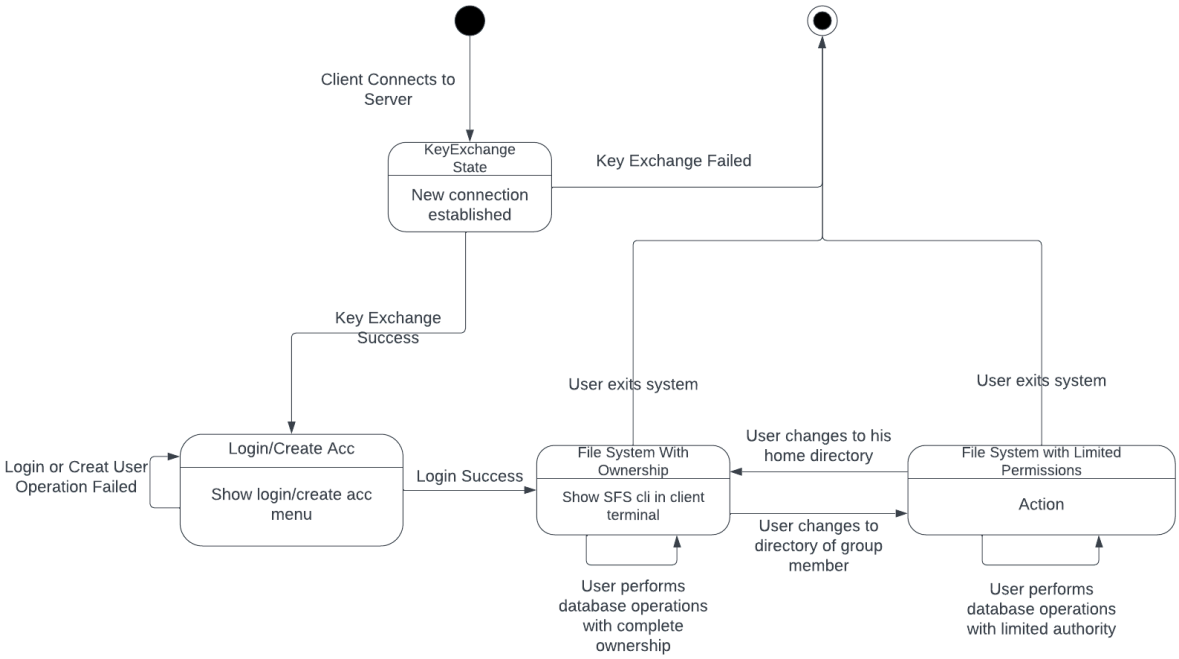

Figure 4: State machine diagram

Above is a diagram outlining the finite state elements of the SFS system, here we have four main states; key exchange, login/create account, file system with ownership and file system with limited permissions. In the key exchange state the server and client securely exchange a session key to be used for data transfer for the rest of their connection. Next they move to the login/create account state, where the client gets input from the user to login to an existing account or create a new account, this information is then authenticated by the server, and on success the client is logged into the system and moved to their home directory. While in their home directory the client is in the file system with ownership state, giving the user full control over files in this directory. If the user decides to change to a team members directory the system changes to a file system with limited permissions state where the user is only allowed to read or write to files that they have explicit permissions for, as defined by the directory owner. At any point while logged into the system, the client can use the 'exit' command to log out and quit the SFS system.

Deployment Instructions

Prerequisites
To install the dependencies, you will need Python 3.x, and Pip

On Ubuntu, they can be installed with:

- sudo apt-get install python3-pip python-dev build-essential

Then update PIP:

- sudo pip install --upgrade pip

Clone the project:

- Git clone https://github.com/ECE422-kyle-ash/ECE422-SFS.git

Cd into the project directory:

- cd ECE422-SFS

Install dependencies:

- pip install -r requirements.txt
- pip install --upgrade cryptography
- sudo apt-get install build-essential cargo

Running the client:

- Cd client/
- Python3 client.py <optional_portnumber> # default is 8080

Running the server:

- Cd server/
- Python3 server.py <optional_portnumber> # default is 8080

User Guide

Login Page

Commands:

- create
    - Create an account in the SFS system.

    - Format:
        - Username
        - Password
        - Confirm Password
- login
    - Login to an existing account in the SFS system.
    - Format:
        - Username
        - Password

Main System

Commands:

- ls - list contents of working directory.
- cd - change the working directory.
    - usage: cd <dirName>
- cat - print out the contents of a text file.
    - usage: cat <fileName>
- mkdir - create a directory.
    - usage: mkdir <dirName>
- touch - create an empty file.
    - usage: touch <fileName>
- echo - write to a file.
    - usage: echo <fileName> <text>
- rm - delete a file.
    - usage: rm <fileName>
- rename - rename a file.
    - usage: rename <fileName> <new_fileName>
- chmod - change file/directory permissions.
    - usage: chmod <fileName/dirName> <permission>
    - valid permissions:
        - internal - all internal users on system
        - user - owner only
        - group - all users in group
- menu - show menu.
- exit - logout from and exit the SFS Shell.

Administrator System

Commands:

- add user - add a user to a group
    - Format:
        - username
        - group name

Conclusion

In conclusion, our system relies on industry standard encryption techniques and integrity checks to give users a robust, secure and reliable File management system so that it successfully achieves the goals of this project. Our file management system allows users to share, create and update files and directories on an untrusted storage server while giving them a guarantee of their data's confidentiality and integrity.

References

1. https://docs.python.org/3/howto/sockets.html
2. https://realpython.com/python-sockets/
3. https://pypi.org/project/cryptography/
4. https://www.cybera.ca/rapid-access-cloud/
5. https://stackoverflow.com/