Secure File System Project

(SFS)

ECE 422 - RELIABLE SECURE SYSTEMS DESIGN

Authors:

Chanpreet Singh

Gurbani Baweja

Abstract

This report provides a comprehensive overview of the Secure File System (SFS) that is developed to allow its users to store data on an untrusted file server. The SFS program was designed using Python and it uses a combination of libraries like os and time. It also uses encryption/decryption techniques for keeping the filenames and contents secure.

The report describes the technologies used for the system and the design of the system along with artifacts/required diagrams like Sequence Diagram and State Diagrams to better understand the design of the system. The report also discusses the deployment instructions for the SFS to allow its users to properly set up the SFS. A user guide about how to run the SFS program is also provided in the report. The overall goal of this report is to provide an overall picture of the design and use of the Secure File System (SFS).

Introduction

In today's time, the use of computers to store private information like files has tremendously increased. With an escalation towards the usage of computers for storing our private information comes several risks to the security of our information. Data corruption and unauthorized modifications to our files are some of the threats to storing our files on computer systems.

As a way to mitigate the threats that come as a result of storing our files on a computer system, we have designed a Secure File System(SFS) to allow users to store data on an untrusted file server. External users(i.e. users who are not part of the SFS) can get access to files and directories belonging to internal users on the file server. However, the SFS encrypts both the contents and names of the files, to protect them from any unauthorized external users. In the event that an external user tries to modify an internal user, the SFS notifies the internal user of such an attempt from an unauthorized user. The file system allows multiple internal users to exchange files with each other by keeping a track of access permissions for reading and writing to each file for each internal user. The SFS also supports the creation of groups and users like a Unix file system.

Overall, the Secure File System (SFS) provides a minimal shell environment that allows users to perform file operations like read, write, share, rename and delete. The report discusses the technologies and design of the SFS and provides information for its users for deployment and use of the SFS.

Technologies

In our SFS implementation we used os, time, and sqlite3 python libraries. The os library was primarily used for executing all the operating system level commands such as cat, ls, touch, etc so our SFS could handle directories, paths, and files in same manner as an operating system would.The time library enabled us to document and verify the time when a file was last updated, this enabled us to ensure if a file was edited after original user logged out or not and if it was modified then alert the original user when they login the next time. The sqlite3 library was used to run the database to store the file and user information in the SQL database. This enabled us to store information for SFS to operate and authenticate users/security of the system. We also used Cybera Rapid Cloud to store our SFS remotely and run it on cloud and used GitHub for online code repository, track versions and work collaboratively. The SFS is developed using python as the primary programming language. For filename encryption/decryption we use cipher algorithms which are cited in the reference section. For file content encryption/decryption we use python encrypt/decrypt library with ‘cp037’ encryption/decryption scheme.

Design

The following diagram provides a high level architectural view of our implementation of the Secure File System:

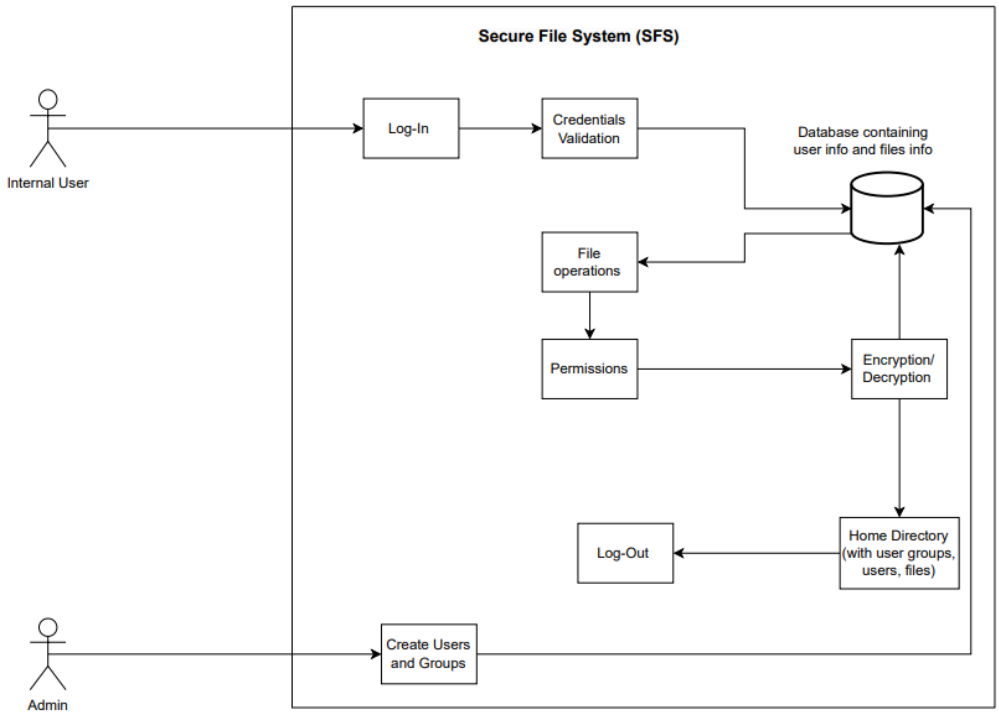

Figure 1: High Level Architectural View of the SFS

Our SFS is designed by using an SQL database, a root folder (SFsystem) for all files/user/groups/directories and a python script to control access and flow of SFS. Our database stores two tables i.e. sfsFiles and sfsUsers, with the username being the primary key of sfsUsers and the foreign key of SFS files.

Only the administrator can make users and groups with every user being a record in sfsUsers SQL table. Both users and groups are also linked by a class as shown in the class diagram. When the admin logins into SFS and create a group, a folder is created in SFsystem, and then when the admin creates a user in a group, a new folder is created in group's folder to act as user's home directory along with a record being created in sfsUsers table.

A logged-in user can create, delete, rename, read, write or set permission for files in its directory, this is implemented using os commands and sqlFiles SQL table. While there is no restriction on creating non-.txt files, however, it's recommended that only .txt files are used. In our sfsFiles table we store the directory path, file name, file encrypted name, file content, file user name, file last modified timestamp, and users who have file read and write permissions. Note that the modified timestamp column enables us to store the time and date when the file is modified by the authorized logged-in user. When a user login, the system decrypts all the file names to which user has access and then compares the file last modification timestamp with the timestamp stored in the database if a timestamp does not match, it means file was modified when the user was not logged in, then the system gives an alert warning to the user. The warning appears only at subsequent login when the file was modified, after that the warning does not appear in subsequent logins for the initial modification.

The encryption of content is using the 'cp037' encryption, however, the file names are encrypted by using cipher_encrypt() and decrypted using cipher_decrypt(), both these functions' source is cited in references. They use string cipher encryption/decryption which is ideal compared to any other encryption/decryption scheme that might involve converting a string into binary, hex or another type. Only at login and logout the encryption/decryption happens for file/directory names, however, the file content is encrypted the moment file content is written using echo command. Each time we use cat command to read a file, we decrypt and show the content to the user, however, the file content stays encrypted all time for enhanced security. Databases do have decrypted content, but we assume in this project that the database is secure and thus don't need to be encrypted, however, it should be easy to encrypt the file content and store it in the database. To ensure we encrypt/decrypt the right files/directories name at login and logout, we use the at_end_encrypt_these_paths array, as it store the paths of files/directories that were decrypted at login and need to be encrypted at logout.

The system implements a build-in restriction on user reading/writing into files to which they do not have permission to, thus preventing any unauthorized reading/writing. Permissions for all

internal users are stored within the database which determines if a user can read and/or write to another user's file. For users in different groups, trying to go to other group's directory is also restricted, this is implemented by verifying the change directory path and ensuring that the current logged-in user belong to the group to whose directory its intending to access, if yes then the directory changes otherwise error message is displayed.

Our system overall work in sync, such as even for simple things such as file rename, it require changes into the database, at_end_encrypt_these_paths array, path updates and other flow changes, all of these things enable our SFS to work in a flow, its a rigid flow structure, to keep it secure and ensure multiple levels of checks.

An external user does not need to log in or use SFS, it rather directly goes to the SFsystem directory where it can navigate through all group directories and user directories, by seeing only encrypted names of all directories/files. The external user can use the operating system echo command and change any file content and leave the SFsystem.

To better understand the design of the SFS, the following design artifacts can be referred to:

Design Artifacts:

1. UML Class Diagram

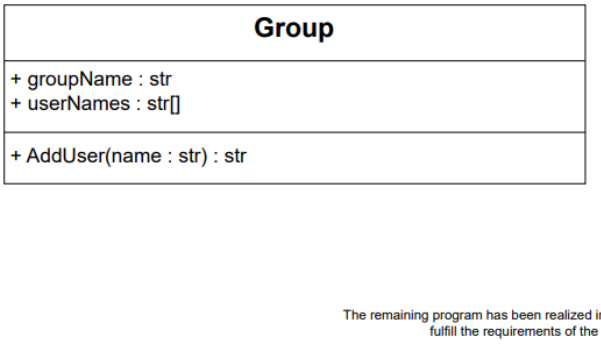

Figure 2: UML Class Diagram for the SFS

2. Sequence Diagram - Create File Requirement

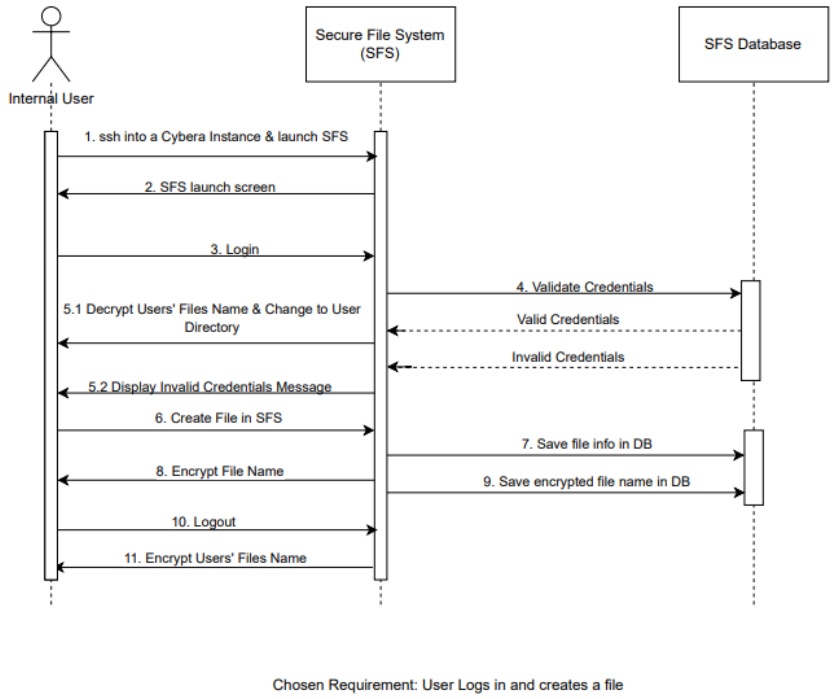

Figure 3: Sequence Diagram for Create File Requirement

3. State Diagrams:

3.1 Admin Interaction with SFS

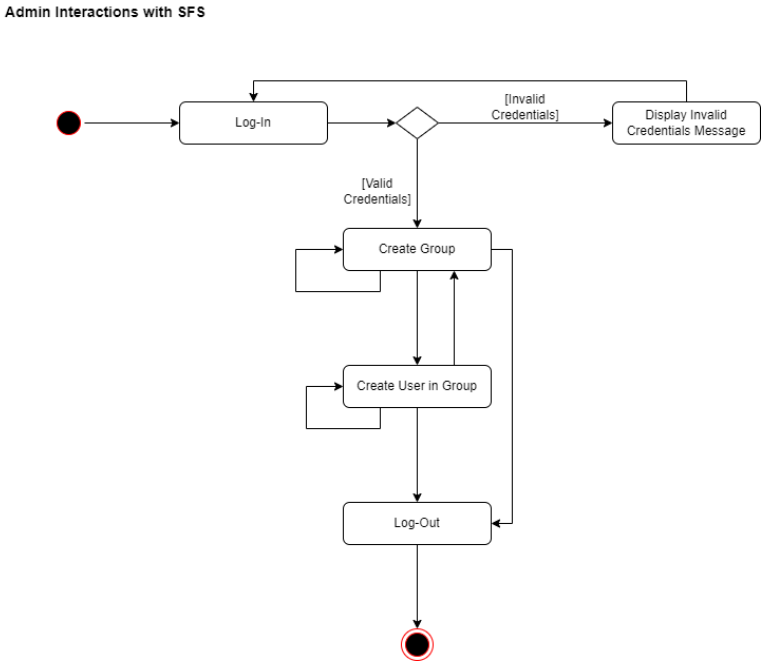

Figure 4: State Diagram - Admin Interaction with SFS

3.2 Create File

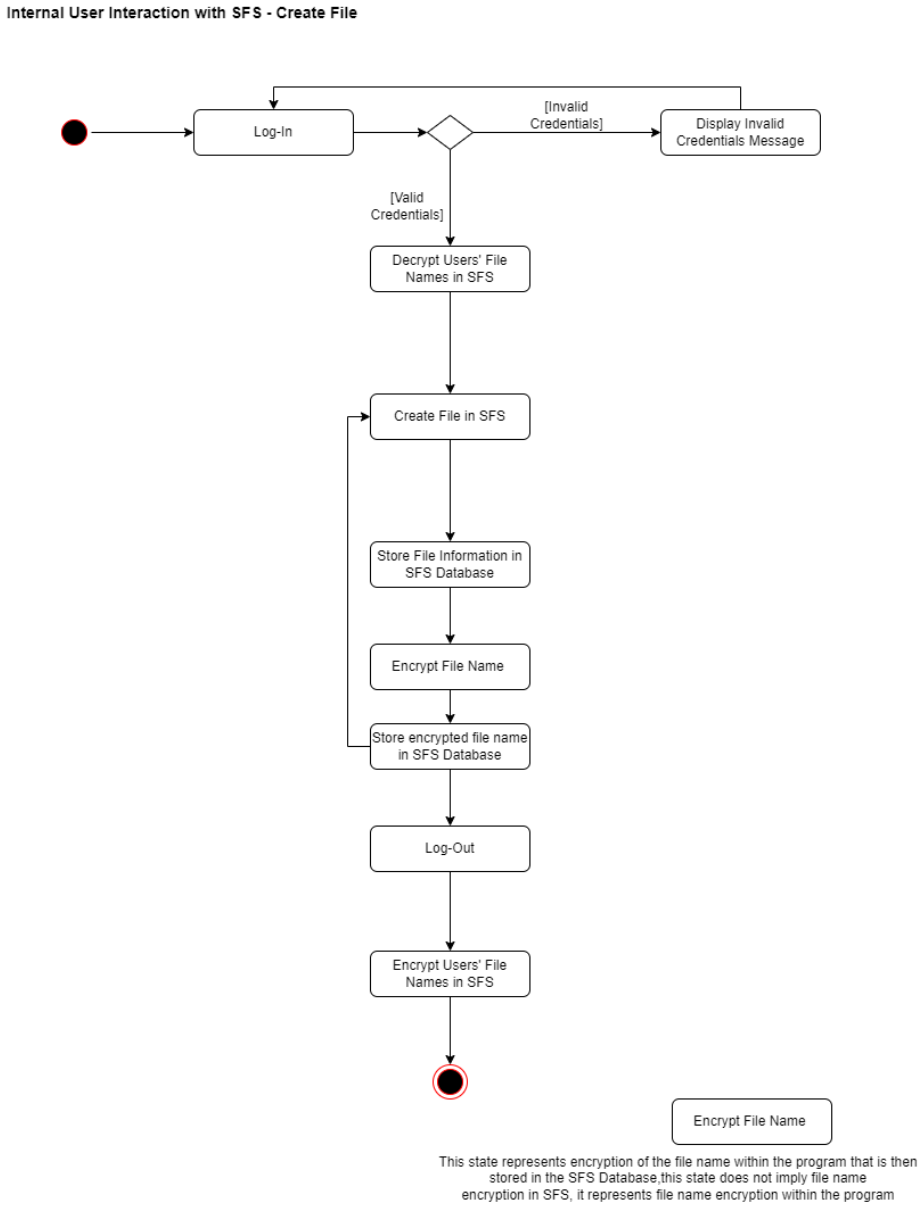

Figure 5: State Diagram - Create File

3.3 Delete File

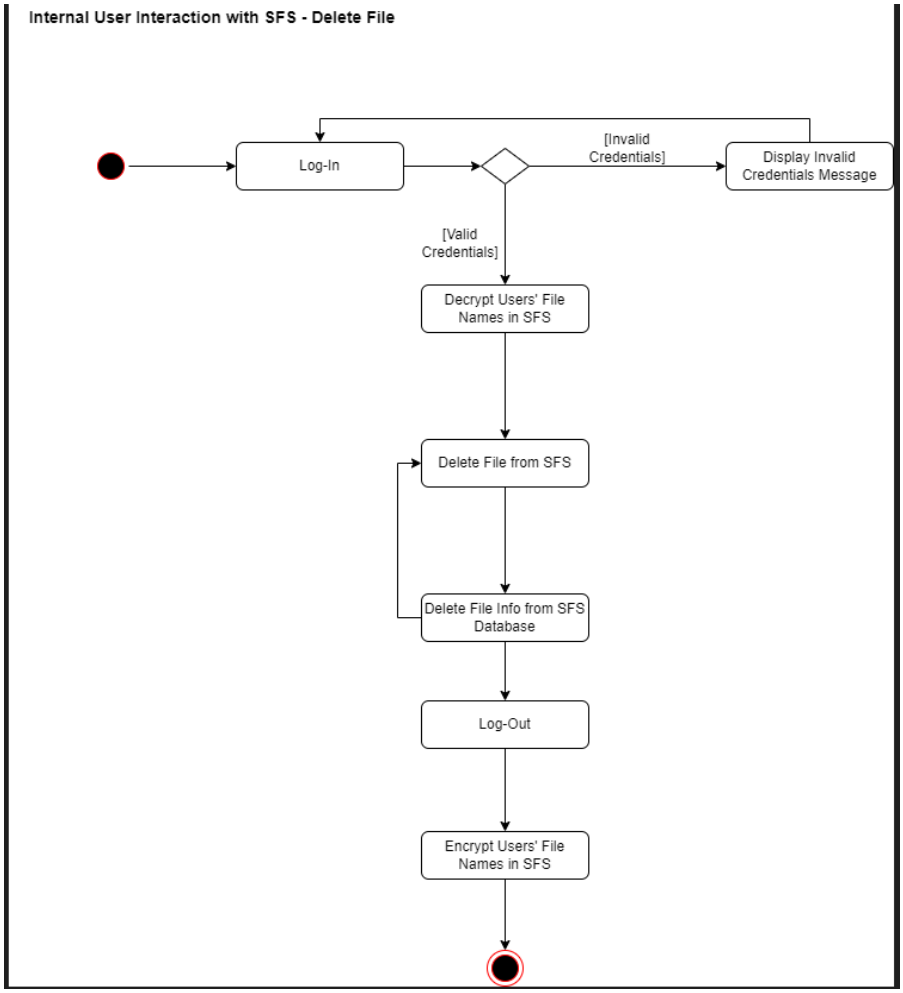

Figure 6: State Diagram - Delete File

3.4 Rename File

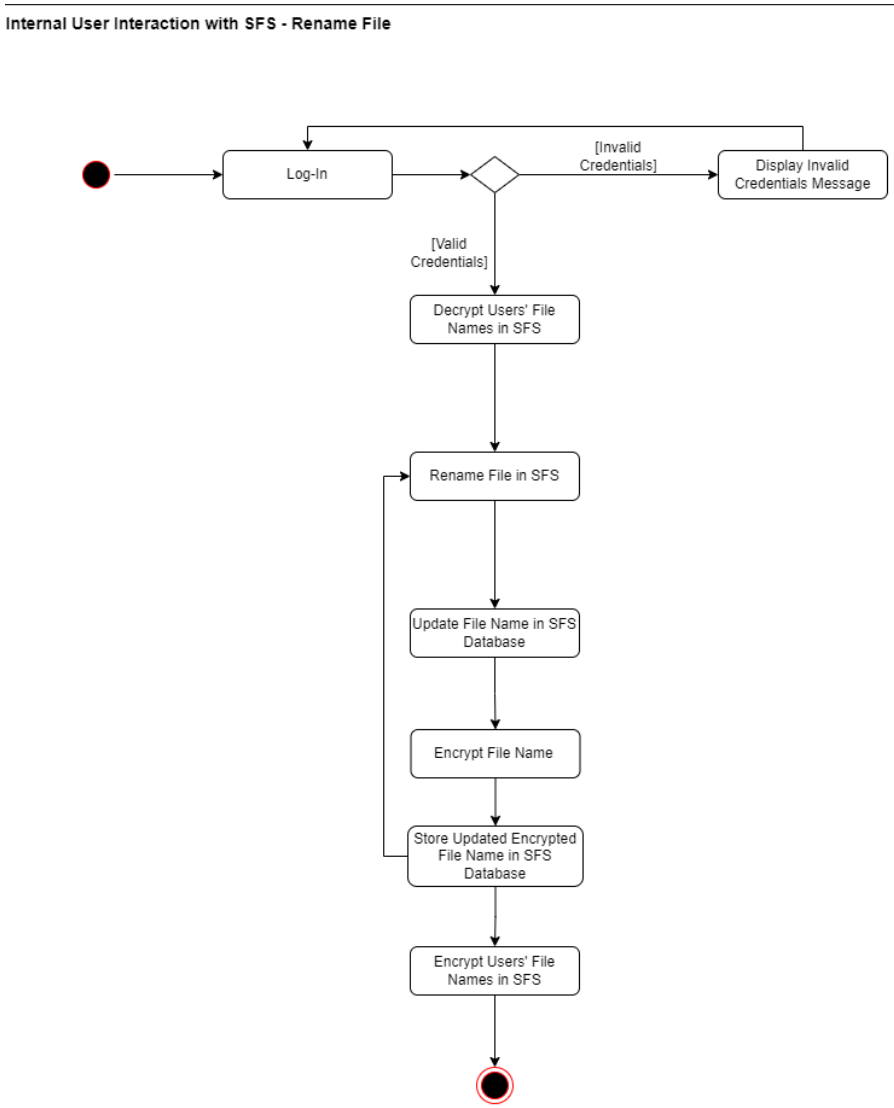

Figure 7: State Diagram - Rename File

3.5 Read File

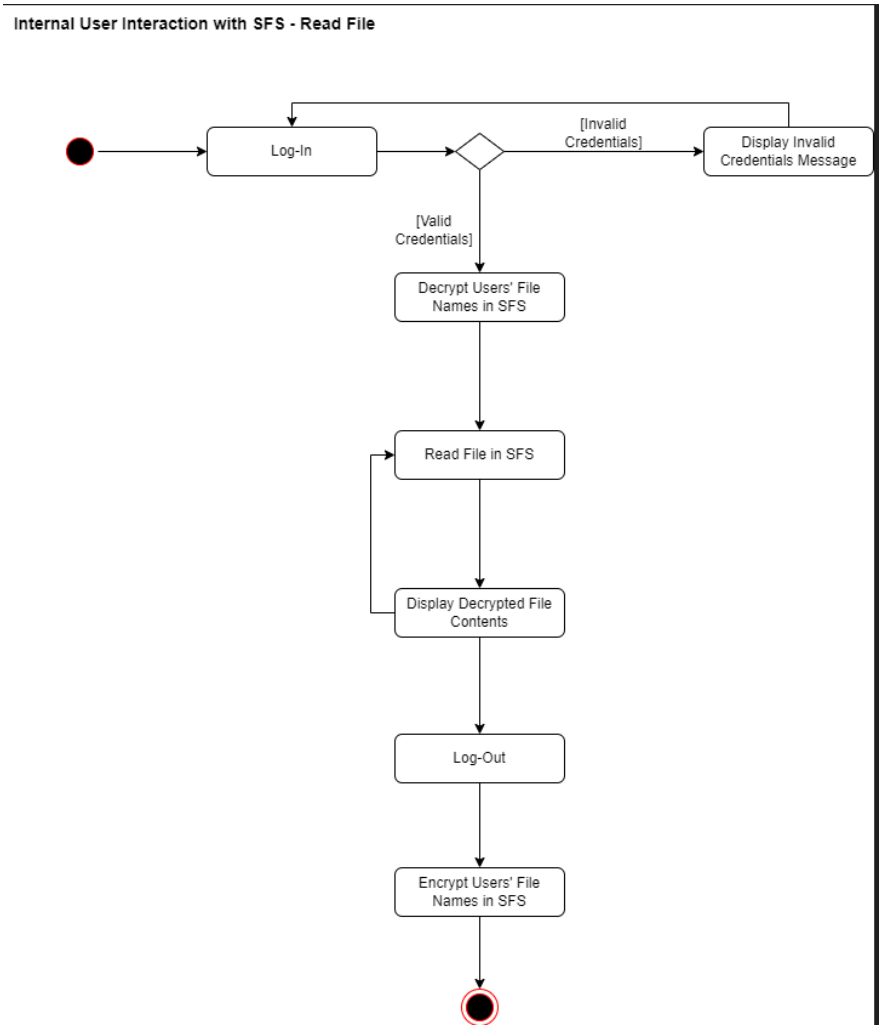

Figure 8: State Diagram - Read File

3.6 Read File of Another User

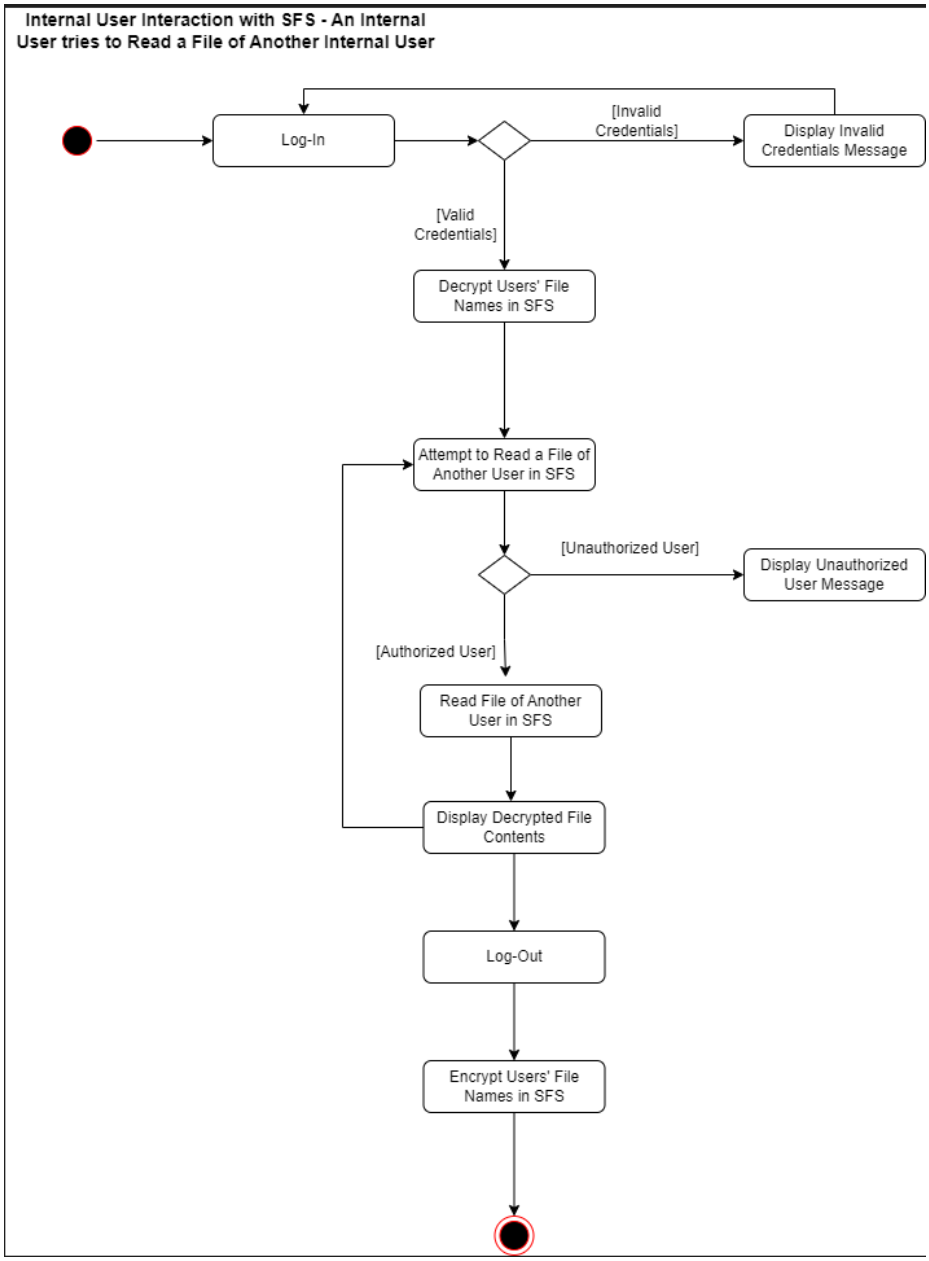

Figure 9: State Diagram - Read File of Another User

3.7 Write to File

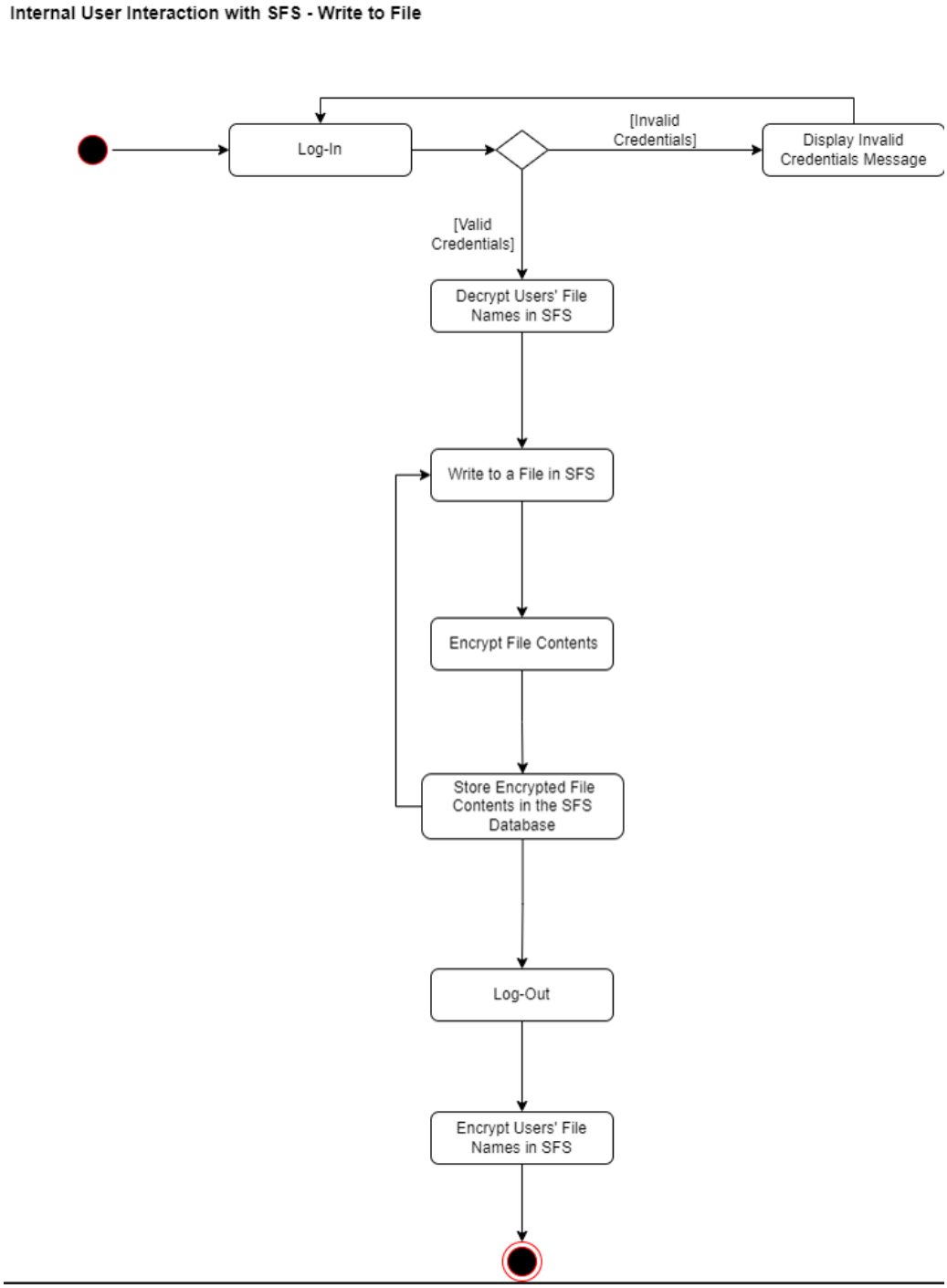

Figure 10: State Diagram - Write to File

3.8 Write to File of Another User

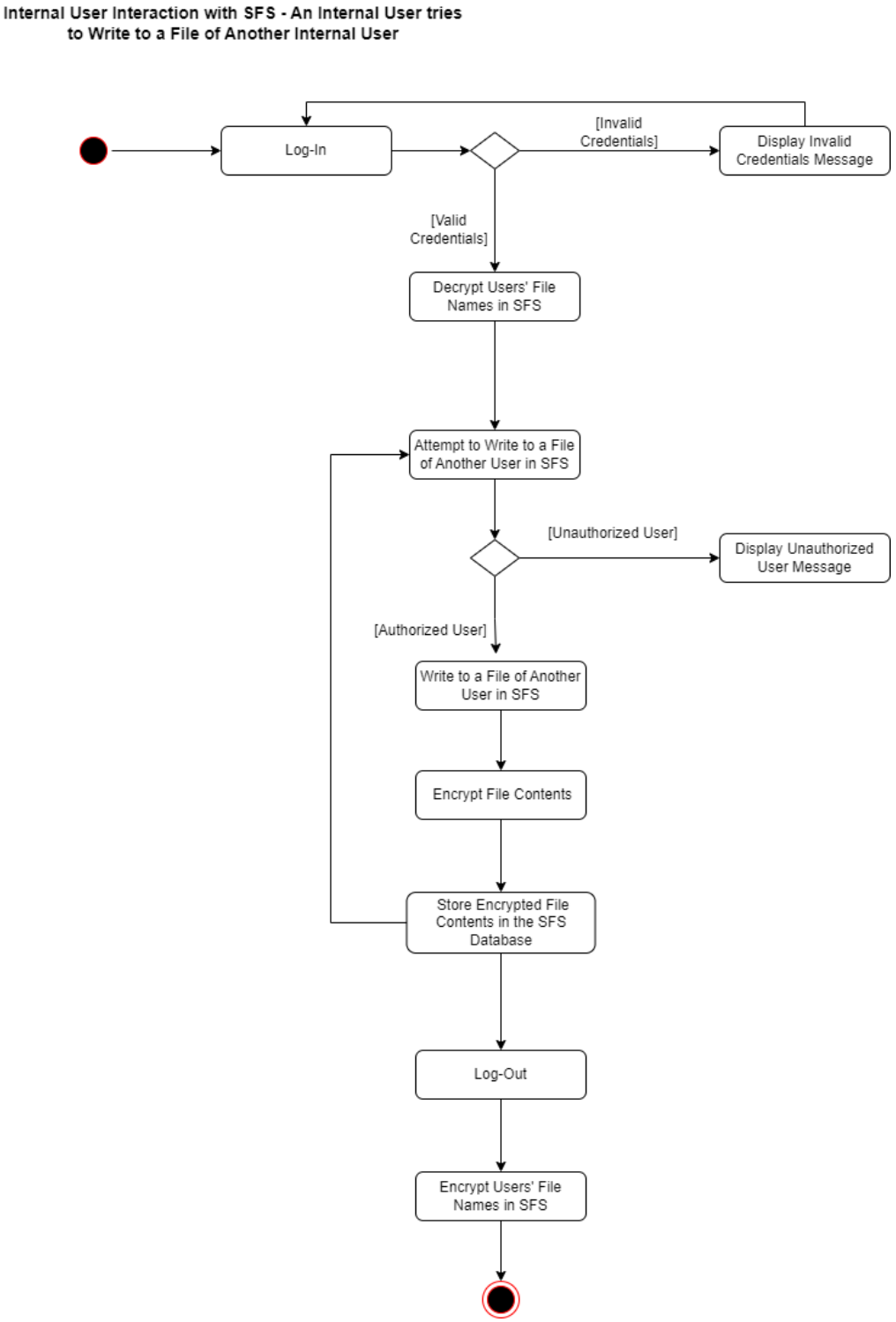

Figure 11: State Diagram - Write to File of Another User

3.9 External User Interaction with SFS

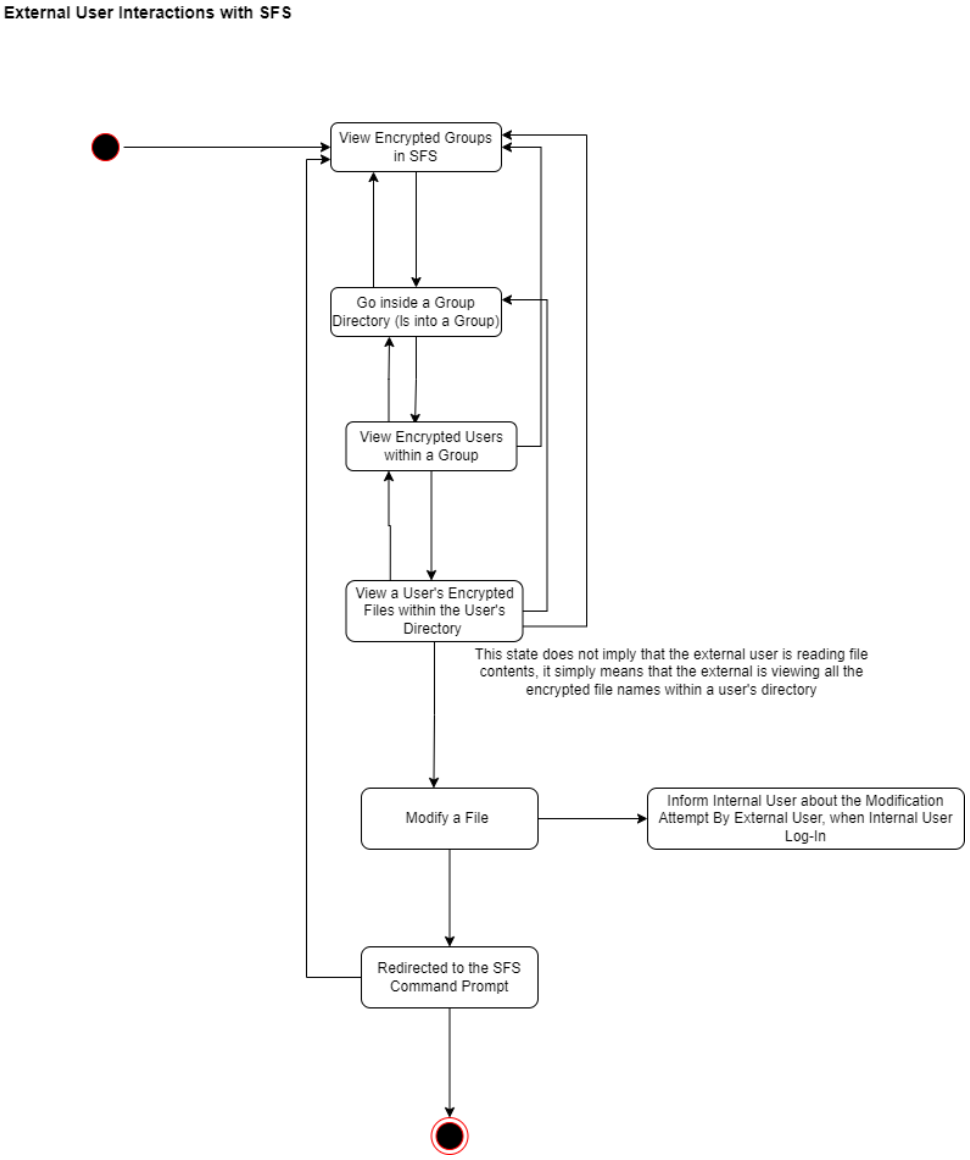

Figure 12: State Diagram - External User Interaction with SFS

Deployment Instructions

To set-up the SFS on Cybera:

1. Login to Cybera rapid cloud, create/login account, create a database instant, copy the IP address for the instance. Make sure that the instance security allows for your local IP access. Download your private key.
2. Clone the code from GitHub

git clone https://github.com/Chanpreet-Singh-UofA/Secure-File-System.git

3. Add SF to the cloned files base directory folder,
   cd Secure-File-System
   mkdir SFsystem
4. Upload the code to Cybera instance
   scp -i path/of/sfs.pem -r path/to/Secure-File-System ubuntu@\[<instance IP address>\]:
5. Now you can login into the SFS on Cybera.

To run the SFS on Cybera:

1. Connect to the Cybera instance using the ssh.
   ssh -i path/of/sfs.pem ubuntu@2605:fd00:4:1001:f816:3eff:fe1f:f1ec
2. Change directory to Secure-File-System
   cd Secure-File-System
3. Run the python script
   python3 SFS.py

User Guide

- To Login
  login <username> <password>
- To logout
  logout
- To Create a group and/or user
  login admin admin
  mkgroup <groupName>
  mkuser <groupName> <username> <password>
- To create a file
  touch <fileName>
- To write content into a file
  echo <fileName> <content>
- To read a file
  cat <fileName>
- To check the current path
  pwd
- To check all the directories and files in current directory
  ls
- To change directory
  cd <full new directory path>
- To rename a file

rename <full current file path> <full new file path>

- To set permissions

  setpremission <file name> <userName to who permission is be granted> <command - r, w, rw>

- To see all valid commands

  help

- To exit the SFS

  exit

Conclusion

A Secure File System (SFS) is critical as the use of computers for storing private information like files grows substantially. Conventional storage methods exposing our files to various threats like modification of files by unauthorized users. Our implementation of the SFS provides users to securely store data on an untrusted file server.

Various technologies such as os, time, and sqlite3 python libraries were used for the implementation of SFS. An SQLite Database was used to store information for SFS to operate and authenticate users/security of the system. Cybera Rapid Cloud was utilized to store our SFS implementation remotely. Cipher algorithms were used for the encryption/decryption of the filename and encrypt/decrypt library with 'cp037' encryption/decryption scheme was used for the encryption/decryption of the file content.

Detailed diagrams were also provided as design artifacts to better understand the design of the SFS. Deployment Instructions and a User Guide was also provided to help users set-up the SFS and run it smoothly. Finally, it can be stated that this implementation of the Secure File System provides a minimal shell environment that allows users to perform file operations like read, write, share, rename and delete. The SFS prevents modifications to its users' files by unauthorized users and thus it can be classified as a secure way of storing files for its users.

References

- File name encryption algorithm - https://likegeeks.com/python-caesar-cipher/
- File name decryption algorithm - https://likegeeks.com/python-caesar-cipher/
- Secure File System (SFS) Document provided on EClass