Secure File System

Final Report

ECE 422

Youssef Ismail, Ali Muneer, Rupin Bapuji

High-level architectural overview

This project is an implementation of a Secure File System (SFS) with a command-line interface (CLI) in Python. This system is designed to store data on an untrusted file server securely, ensuring that internal users' file and directory names and contents are encrypted and that data integrity is checked.

The following components make up the architecture of the project:

- Encryption Module
    - Utilizes the cryptography library to encrypt and decrypt file contents and names. This singleton class manages encryption keys and provides methods for working with JSON data and strings, ensuring that data is stored securely on disk.
- File I/O Module
    - Handles the creation, deletion, reading, and writing of files and directories on the filesystem. It integrates closely with the Encryption Module to ensure that all file operations are performed on encrypted data, maintaining the privacy and integrity of the data.
- Graph Module
    - Represents the file system's structure and permissions as a graph. Nodes in the graph represent files and directories, each with associated permissions for users and groups. This module manages access control, checks for readable and writable access, and supports operations like listing directory contents, creating files and directories, and modifying permissions.
- User Management
    - Manages user accounts and group memberships. It supports operations such as creating users, adding users to groups, and checking group memberships. This component also ensures that user authentication is handled securely, with passwords being properly hashed.
- CLI
    - Provides a command-line interface for interacting with the SFS. It supports commands for user login and registration, file and directory management (creating, reading, writing, deleting, renaming), and permission management. The CLI leverages the above modules to perform operations in a secure and user-friendly manner.

It can be represented using the following UML class entity diagram.

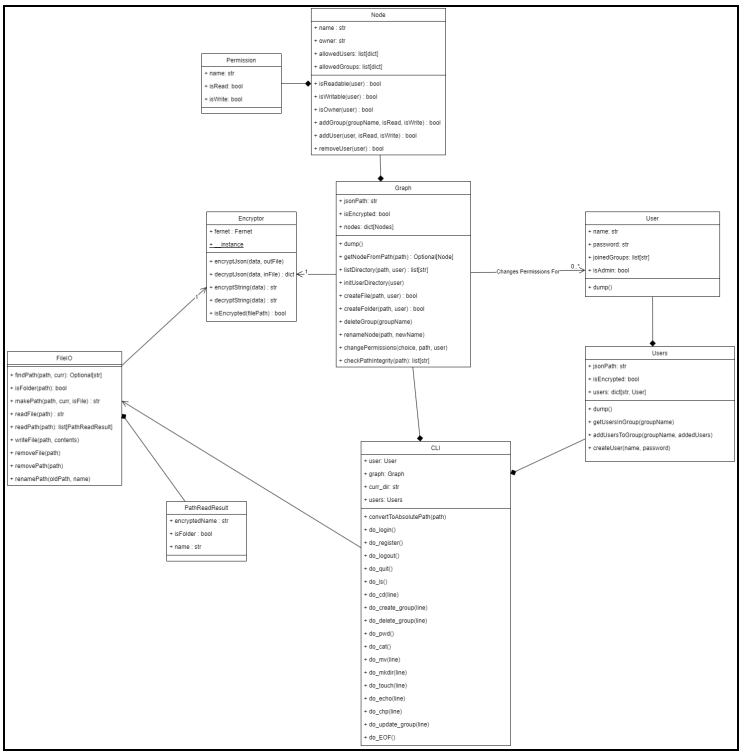


Figure 1: Diagram of the System Architecture

Design Description

User Information

We used an encrypted JSON file to store user information. This information consisted of a username, hashed password, and their joined groups. On startup, the encrypted JSON is decrypted and parsed by our user management module, which will create a set of User objects, each with the same fields as the JSON. Any change to this data (registration, add/remove/edit groups) is first changed in the local object set, then updated in the JSON. We opted to update the JSON with every change command so we can protect against force quits. This way, if data is updated, but the user force quits the system, the changes will still be preserved upon the next login.

Files & Folders

We used an encrypted JSON to store files & folders as well. We initially used a nested list for the JSON, similar to a real file system. We later changed the format to a flat list to decrease access time. Each entry in the JSON is identified by its unique path, and each entry has the following information: path, owner, allowed users, and allowed groups. We used a Graph class with Node objects to store the permissions data on startup. The encrypted JSON is parsed and stored as a directed graph. Any updates to the permissions or file structure of the system is immediately applied to the file system, the Graph object, and the JSON file.

CLI

Pseudo code:

```
Initialize the CLI environment
  - Load the encrypted users and permissions from JSON files
  - Set the initial prompt and directory

While the CLI is running
  If the command is "login"
    If a user is already logged in
      Display an error message
    Else
      Prompt for username and password
      If the username and password match
        Set the current user and update the prompt
        Check for corrupted files in the user's directory
      Else
        Display an error message

  If the command is "register"
    If a user is already logged in
      Display an error message
    Else
      Prompt for username, password, and password confirmation
      If the passwords match and the username doesn't exist
        Create and log in the new user
        Initialize the user's directory
      Else
        Display an error message

  If the command is "logout"
    Clear the current user and reset the prompt

  If the command is "quit"
    Exit the CLI

  If the command is "ls"
    If no user is logged in
      Prompt to login
    Else
      List files in the current directory, filtering based on user permissions

  If the command is "cd"
    If no user is logged in
      Prompt to login
    Else
      Change the current directory, if accessible

  If the command is "create_group"
    If no user is logged in or the user is not an admin
      Display an error message
    Else
      Create a new group with specified users

  If the command is "delete_group"
    If no user is logged in or the user is not an admin
      Display an error message
    Else
      Delete the specified group from all users and nodes

  If the command is "pwd"
    Display the current working directory
```

```
  If the command is "cat"
    If no user is logged in
      Prompt to login
    Else
      Read and display the contents of a specified file, if accessible

  If the command is "mv"
    If no user is logged in
      Prompt to login
    Else
      Rename a specified file or directory, if permitted

  If the command is "mkdir"
    If no user is logged in
      Prompt to login
    Else
      Create a new directory at the specified path, if permitted

  If the command is "touch"
    If no user is logged in
      Prompt to login
    Else
      Create a new file at the specified path, if permitted

  If the command is "echo"
    If no user is logged in
      Prompt to login
    Else
      Overwrite a file with specified content, if permitted

  If the command is "chp"
    If no user is logged in
      Prompt to login
    Else
      Change permissions of a specified file or directory, if owner

  If the command is "update_group"
    If no user is logged in or the user is not an admin
      Display an error message
    Else
      Add or remove users from a specified group

Handle any unrecognized commands with an error message
On exit, perform cleanup if necessary
```

Tools & Technologies

Packages

BCrypt: To enhance security, we used BCrypt for password hashing. BCrypt's hashing algorithm ensures that passwords are securely stored by incorporating a salt value and multiple rounds of hashing, making it resistant to brute-force attacks. This bolstered our application's security measures and protected user credentials from potential threats.

Cryptography(Fernet): For encrypting sensitive data such as metadata files and secure user files and folders, we utilized the Fernet encryption method. Fernet offered a straightforward yet secure symmetric encryption approach, allowing us to encrypt and decrypt data efficiently while maintaining cryptographic integrity. Integrating Fernet into our application ensured that sensitive information remained protected from unauthorized access and tampering, bolstering our overall data security strategy.

We chose Fernet for encrypting sensitive data because of its straightforward symmetric encryption approach, making it easy to implement within our development workflow. Its strong cryptographic security features, such as key rotation and timestamp verification, ensure the integrity and confidentiality of encrypted data. Additionally, Fernet's compatibility with Python, which is widely used in our development stack, made it a convenient and seamless choice for integrating encryption functionalities into our application.

Version Control and Collaboration

Github: In order to collaborate in an Agile environment, we shall be using Github for collaboration. Its interactive GUI facilitates the use of code review and pair programming. Further, its branch and merge functionality allows us to engage in iterative development. Separate branches can be made for each user story to simulate continuous integration.

Docker: Docker is a containerization platform that allows us to package our applications and dependencies into containers, ensuring consistency and portability across different environments. We used Docker to deploy our application regardless of the OS our users are using. Docker also lets us share volumes to the local system which we used to store our user files and metadata objects.

Project Management Tools

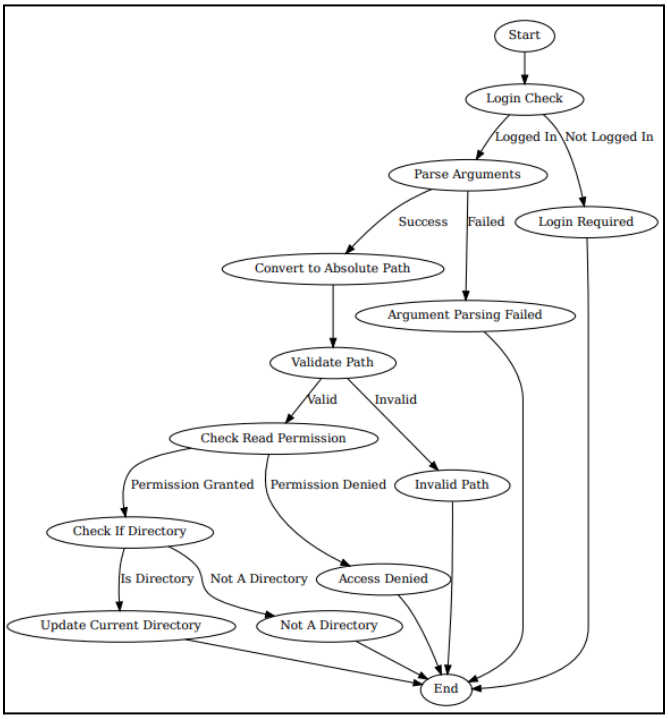
Github Projects: For tracking our progress we shall use the embedded project view within Github. This will tightly integrate our codebase with our project management system to allow us to track our overall progress on a commit-by-commit basis.

Communication Tools

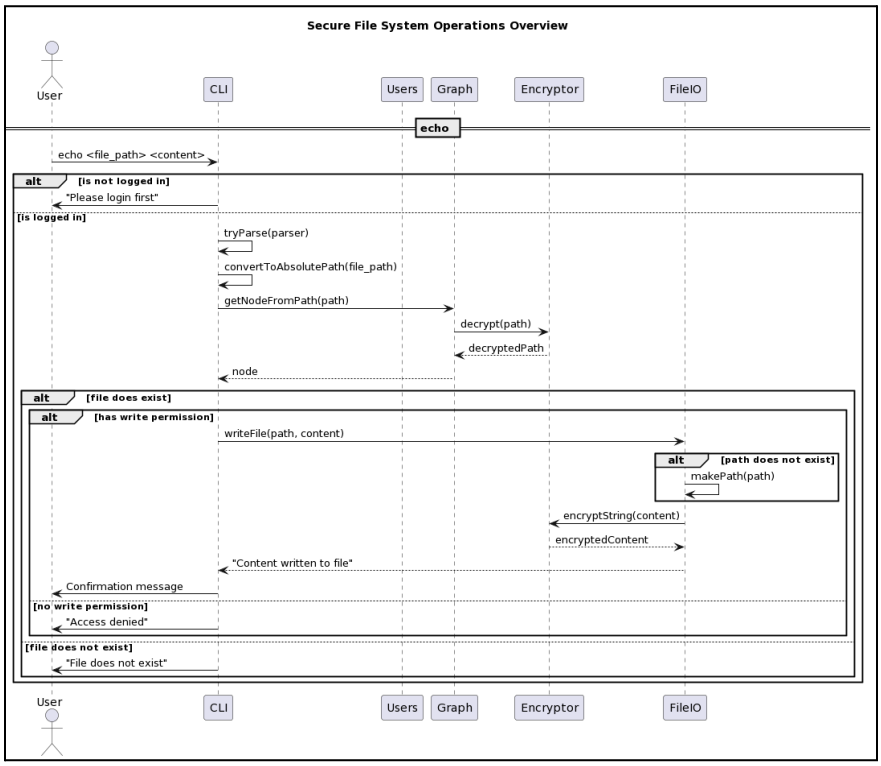
Discord: To communicate with our team and distribute our tasks accordingly we shall use Discord as an informal communication medium. Sprint planning and interim progress will be posted and discussed there.

Github Issues: More formal documentation of bugs and features will be done on different issues, by “commenting” on issues to keep a trackable line of communication.

State Diagram for “cd” command



Sequence Diagram for “echo” command



Deployment

Due to our containerized approach to development deploying the SFS is quite easy.

This assumes you have a running VM instance. You must also have Docker Compose installed on your VM.

For instructions on how to do that see this link: https://wiki.cybera.ca/display/RAC/Rapid+Access+Cloud+Guide%3A+Part+1 . Once you have done that, continue.

1. Clone the git repository on your VM instance:

```
git clone https://github.com/yismailuofa/securefilesystem.git
cd securefilesystem
```

2. Create a folder called files in the root directory:

```
mkdir files
```

3. Generate a fernet key for encryption:

```
pip3 install cryptography
python3 -c "from cryptography.fernet import Fernet; key =
Fernet.generate_key(); print(key.decode('utf-8'))" > fernet.key
```

4. Start the app in interactive mode:

```
docker compose run app
```

Once you have run this, you should be able to interact with the app right away,

User Guide

The app starts off with one user created called admin which has the password “admin”. This user has privileges to view anyone’s data.

Our codebase is self documenting as well. Once you start the app you can type *help* to see commands and information about them.

```
sfs> help
```

```
Documented commands (type help <topic>):
========================================
EOF  cd   create_group  echo  login   ls     mv   quit      touch
cat  chp  delete_group  help  logout  mkdir  pwd  register  update_group
```

You can also type *help <topic>* to see information on a command's usage.

```
sfs> help mv
Rename a file or directory. Usage: mv <source> <name>
```

In addition, a description of the commands is provided here:

Command	Description
EOF / Ctrl-C/ quit	Exit the program
cd	Change directory
create_group	Create a group of access permissions
update_group	Update an access group's members.
delete_group	Delete an access group
echo	Overwrite the contents of a file
login	Login as a user
ls	Show the contents of your current directory
mv	Rename a folder or file
touch	Create a new empty file
cat	Display the contents of a file
chp	Change a file's permissions.
logout	Logout the current user
register	Register a new user
pwd	Display the present working directory

Conclusion

In conclusion, the secure file system was successfully implemented using Python and Docker to create an authenticated and encrypted file storage system. Our system encrypts all user files and passwords while allowing users the flexibility to create and edit complex file and folder structures. We also allowed users to create robust permissions by sharing with other users and the ability to create groups which can abstract this process further. Of course, all this metadata was also securely encrypted as well. This project demonstrates the importance of designing systems that are not only functional but secure which is quite important in the real world where user data is extremely sensitive. Overall, we were able to learn a lot about designing these systems through this project.