Project #2: Secure File System

ECE 422 LEC B1

Due: Monday, 8 April 2024, 11:59 PM

Yihe Wang, Jianxi Wang, John Yu

Abstract

The project will explore the basic concepts of secure system design in terms of a secure file system by completing an integrated server-client program in Python and test its performance. We will also explore how encryption and encoding practices such as SHA-256 and Base64 can help with system security and identify malicious actions [1]. We will be deploying the program on Cybera Rapid Access platform. The task is to complete two major functionalities of the system, the first one focuses on the client side which allows the user to create, manipulate and delete files on the system, in addition, users can also change the permissions of the file in which they own. The second one is on the server side, the server shall authenticate the user and grants access while keep tracking the files stored on the server and alert the user should their files are modified without authorization [2]. We will practice the use of encryption and decryption procedures, examine how hashing mechanism can identify file modifications. We shall also strengthen our understanding on file management and client-server interactions in this project.

Design (see Appendix for UML sequence diagram and state machine diagram)

Authentication

First, user credentials are stored as an extended attribute "user.SFS.users" on the root folder of the system, it is the folder that contain all user folders, in which each of them contains user's private files.

```
    def get_all_users(self):
        encrypted_users = xattr.getxattr(self.root,
"user.SFS.users").decode()
        return json.loads(encryptor.decrypt_data(encrypted_users))
```

We used the "xattr" library to define our own extended attribute for the files and folders in this project. Note that the user credentials here are encrypted before assigning to the extended attribute.

Upon launching the program, the system will prompt for username and password, and try to find a match from all user credentials.

```
    def get_user(self, user_name):
        users = self.get_all_users()
        if user_name not in users:
            raise Exception(f"{user_name}: User does not exist.")
        return users[user_name]
```

The above code snippet tries to find the user itself via username, it will raise exception if username is not found. Note that the decryption of the existing user credential is done in get_all_users().

```
                    if current_user["password"] != password:
                        print("Invalid credential.")
                        init()
                        continue
```

This is the code for comparing the password, the system will prompt invalid credentials should an incorrect password is provided.

Encryption

We used an external library “Fernet” to create an encryptor:

```
from cryptography.fernet import Fernet
```

This encryptor will generate a unique key for the entire file system and handle the encryption and decryption for **file, filename, folder name, user credentials** and **access permission**. Note that since there is only one key for the file system, the same encryptor is shared among all users.

We defined two functions to handle the encryption and decryption of the data:

```
    def encrypt_data(self, decrypted_data):
        encrypted_data = self.encryptor.encrypt(decrypted_data.encode())
        return base64.urlsafe_b64encode(encrypted_data).decode()

    def decrypt_data(self, encrypted_data):
        encrypted_bytes =
base64.urlsafe_b64decode(encrypted_data.encode())
        return self.encryptor.decrypt(encrypted_bytes).decode()
```

These the encrypt_data() function will take any data (string or JSON) passed in, and return the encrypted string which is not human-readable. The decrypt_data() function works in reverse. These functions are integrated in the following commands:

encrypt_data		decrypt_data	
mkdir	To encrypt the folder name	ls	To decrypt the file names in the current folder
touch	To encrypt the file name	cd	To decrypt the folder name on the system for comparison
echo	To encrypt the information to be stored in the file	cat	To decrypt the information stored inside the file

mv	To encrypt the new folder name	login	To decrypt the saved user credentials for comparison
chmod	To encrypt the modified file/folder permission		
create_user	To encrypt the new user's credentials		

To summarize, the encryptor act as a translator that handles the interpretation of encrypted file or folders. Should the user wish to access any file or folder, the encryptor will decrypt the data for user display and if the user wishes to make changes to the system, the encryptor will encrypt the new data and pass to the system for storage [3].

Access permission

Like the user credentials, we utilize extended attributes "user.SFS.permission" to store the access permission for each file and folder.

```
    def get_path_permission(self, path):
        encrypted_permission = xattr.getxattr(path,
"user.SFS.permission").decode()
        return json.loads(encryptor.decrypt_data(encrypted_permission))


    def set_path_permission(self, path, permission):
        encrypted_permission =
encryptor.encrypt_data(json.dumps(permission))
        xattr.setxattr(path, "user.SFS.permission",
encrypted_permission.encode())
```

The access permission of a file or folder is a standard 3×3 matrix which lists the corresponding permission for the **owner**, the **users within the same group** and **all users**. Below is an example:

	Owner	**Group**	**All**
read	1	0	0
write	1	0	0
execute	1	1	0
Decimal	7	1	0

This is the default access we used when the user creates a new file via “touch” command. The binary number for each column can translate to a decimal number and all three together will be encrypted and stored to the extended attribute, in this case it is “710”.

The following table denotes what each access allows the user to do, and when they will be checked:

	File	Folder
read	To read the file ("cat")	To list the decrypted item names in the folder ("ls")
write	1. To modify the file ("echo") 2. To rename the file ("mv") 3. To remove the file (“rm”)	1. To create new folder in the current folder ("mkdir") 2. To create new file in the current folder ("touch") 3. To rename the folder ("mv") 4. To remove the folder (“rm”)
execute	N/A	To enter the specified folder ("cd")

Note that if the user has the execute access of a folder but not the read access, they can still enter the folder, but the “ls” command will only list encrypted folder/file names.

File Integrity

To check if the files have been tempered with, we used an external library “dirhash”.

```
from checksumdir import dirhash
```

This library will create a hash for a folder and all the files and subfolder within it. Therefore, we used it to generate a hash upon user logging out and saves it as a part of the extended attribute "user.SFS.users" which all users have their own hash for their user directory.

```
users[user_name]["lastHash"] = dirhash(user_home_path, 'sha256')
```

When the user attempts to login again, the system will generate another hash for the user directory and compare with the one that is save within the attribute. Should any file been renamed, changed, deleted, or moved, the hashes will not match, and the system will display an error message.

```
return users[user_name]["lastHash"] == dirhash(user_home_path,
'sha256')
```

Note that this method can only tell if there is a modification but cannot determine where the modification took place.

Tools

The following displays the tools we plan to use for SFS development:

1. Cybera Rapid Access Cloud: This is the cloud platform we used to host the SFS as we can deploy Ubuntu Linux instances on it.

2. Linux - Ubuntu 20.04: Linux is selected for its direct support of Linux commands, file management, and access control capabilities, which are essential for implementing SFS functionalities. Besides, Ubuntu 20.04 offers enhanced stability and functionality compared to earlier versions like Ubuntu 18.04.
3. VSCode (Remote SSH, Python): VSCode is chosen for remote development, enabling us to program directly on the server hosting SFS. VSCode offers syntax highlighting and other ease-of-use extensions, and overall, serves as a lightweight and efficient development environment.
4. Python: Python is chosen as our primary programming language due to its simplicity and versatility. Python's extensive library ecosystem will support us in setting up APIs, implementing encryption schemes, and fulfilling other SFS requirements.

Technologies

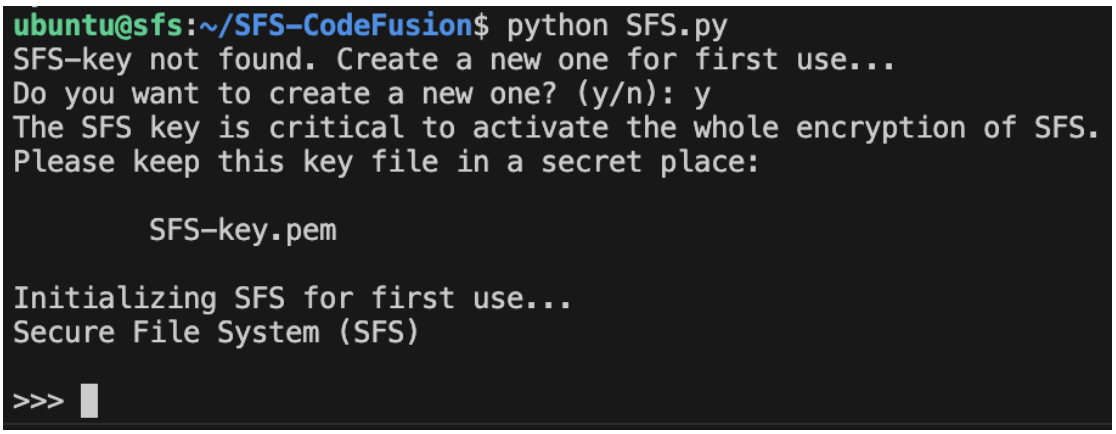
The following lists all technologies required to build the auto-scaling system:

1. Fernet: Fernet is a symmetric encryption method that makes it easy to encrypt and decrypt data securely. It uses a combination of HMAC for verification, AES for encryption, and a **timestamp** to ensure that encrypted messages are not tampered with and remain confidential and authentic over time.
2. dirhash: Dirhash is a software utility designed to calculate a single sha256 hash value for a directory and its contents, including all files and subdirectories. This process helps verifying the integrity and detecting changes in the contents of a directory over time [4].
3. xattr: xattr refers to "extended attributes," a feature that allows users to associate metadata with files and directories beyond the standard attributes provided by the filesystem. These attributes can store additional information like access permissions, encryption keys, or user-defined data.

Deployment instructions

The following contains steps required to deploy the SFS:

1. Connect to instance 10.2.9.68 on Cybera.
2. Download the code for SFS: https://github.com/CoasterJX/SFS-CodeFusion
3. Once downloaded, run “cd /path/to/ SFS-CodeFusion”
4. Run “python SFS.py” to activate SFS.
5. If you deploy SFS for the first time, the system will prompt the following. Type “y” and you should see a file named SFS-key.pem. Keep it in a private place so that nobody can decrypt the content under folder “file-system”:



User guide

The following contains guides for admin user:

1. To access any admin specific commands, you need the correct SFS-key.pem to be placed under SFS-CodeFusion/, otherwise you cannot login to admin account.
2. In SFS command line, type “login” and press ENTER, the command line will prompt you to enter username. Type “admin” and ENTER, and if SFS-key.pem is presented, the system will now login to admin.
3. The followings are command lines specifically for admin user:
    a. create-user: create a user in SFS, when creating the user, the system will ask for username, password, and groups the user belongs to (comma separated, with the first group as the user’s default group for files).
    b. All other commands for internal users (see user guide for internal user).

    Note that we do not have command “create-group” because all groups are created only when there is at least a user belongs to this group (for a tidier management, no “dangling” groups). Therefore, group creations are automatically done in create-user after system asks for groups this user belongs to.

The following contains guides for internal user:

1. Before accessing SFS, you need to ask administrator (admin user) to create a user with specified username, password, and groups for you.
2. In SFS command line, type “login” and press ENTER, the command line will prompt you to enter username. Enter your username and password followed by the command line guide, and you should login as your user account successfully.
3. The followings are commands supported for internal users:
    a. pwd: display current directory.
    b. ls: list all files and folders under current directory.

c. cd: change current directory (../ supported).
d. mkdir <folder_name>: create a new folder under current directory.
e. touch <file_name>: create a new file under current directory.
f. cat <file_name>: read a file content under current directory.
g. echo <file_name> [contents]: write contents to an existing file under current directory.
h. mv <old_name> <new_name>: rename the file/folder with old_name to new_name under current directory.
i. chmod <permission> <file/folder_name>: change permission of a file/folder (you must be the owner of that file/folder).
j. rm <file/folder_name>: remove a file or folder in current directory.

Note that as an internal user, you need permission to perform operations on specific files/folders.

The following are guidelines for all users:

- Make sure to type "logout" after you finished your work.
- The system will notify you once you login if there is any file/folder being modified by others after your last "logout", but the system will not recover data for you. Therefore, it is a good practice to backup those data.

Conclusion

Overall, the project is successful and all objectives for the projects are met. We were able to identify if a file or folder has been tempered with by an external user by checking the directory hash. We also explored the usage of extended attributes to store custom information on each folder and file. We have learned that encryption methods can help protect files from breach attempts effectively and together with hashing functions, we can alert the user that there has been a security breach [5]. We also learned how access permissions work in file systems. All in all, we have successfully completed all objectives in this lab as the file system runs smoothly.

References

[1] E. Sarafianou and M. Mogyorosi, "What's the difference between encryption, hashing, encoding and obfuscation?," Auth0, https://auth0.com/blog/how-secure-are-encryption-hashing-encoding-and-obfuscation/ (accessed Mar. 31, 2024).

[2] J. H. Saltzer and M. D. Schroeder, "The protection of information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975. doi:10.1109/proc.1975.9939

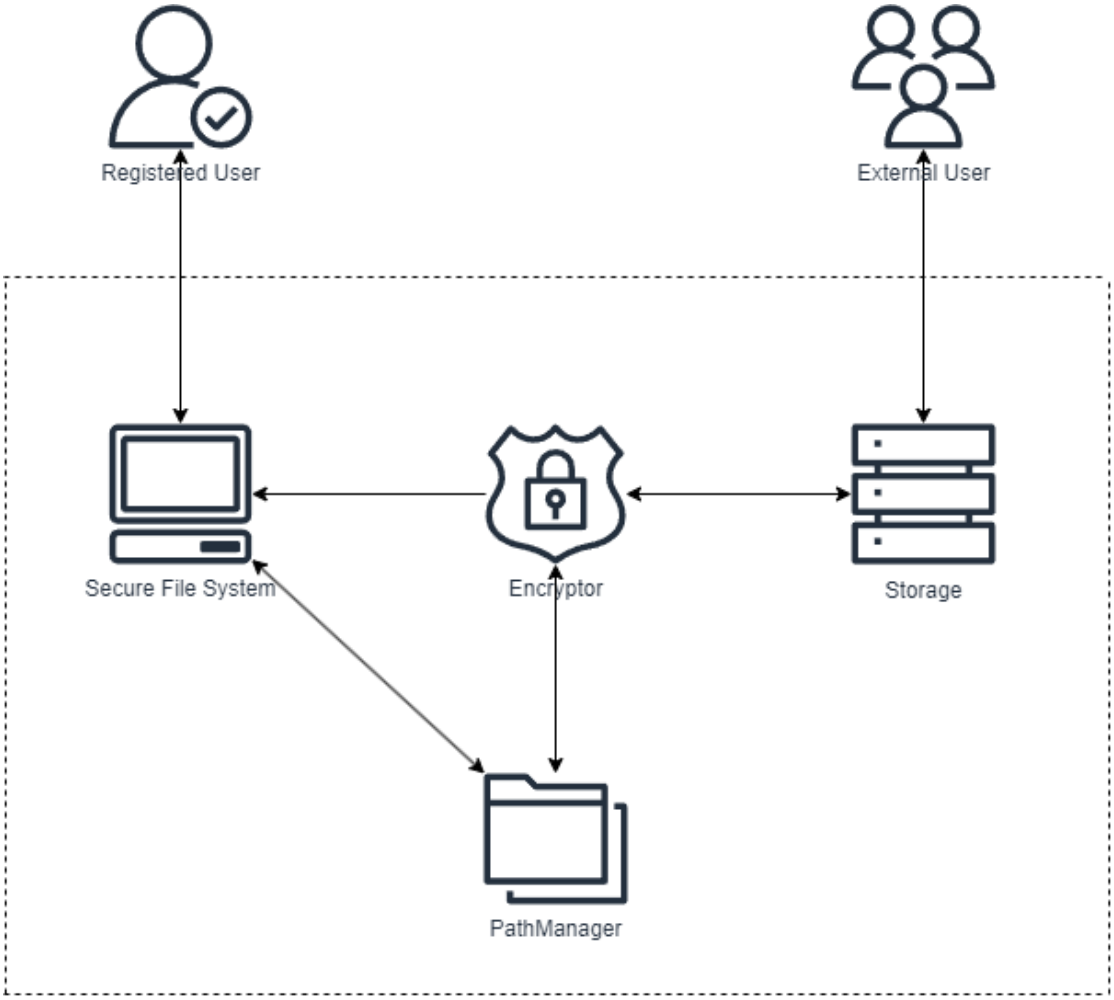
[3] D. Pillsbury, “Python secure password management: Hashing and encryption,” DEV Community, https://dev.to/dpills/python-secure-password-management-hashing-and-encryption--1246 (accessed Mar. 31, 2024).

[4] Mr. Shrikanta Jogar and Mr. Darshan S Handral, “Secure file storage on cloud using hybrid cryptography,” *International Journal of Advanced Research in Science, Communication and Technology*, pp. 540–551, Jul. 2022. doi:10.48175/ijarsct-5861

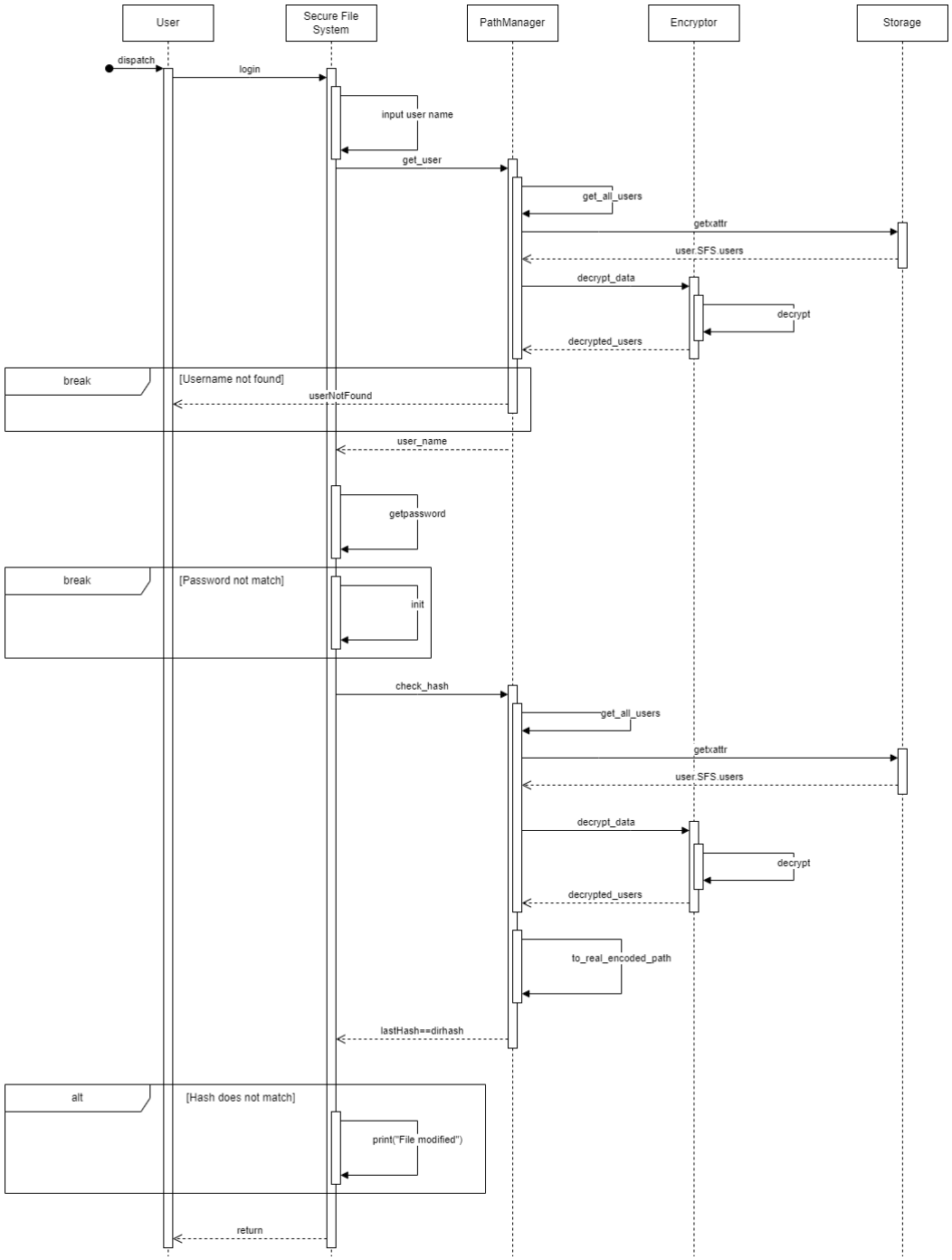
[5] S. P. G. *et al.*, “A secure data deduplication system for integrated cloud-edge networks,” *Journal of Cloud Computing*, vol. 9, no. 1, Nov. 2020. doi:10.1186/s13677-020-00214-6

Appendix

Architectural overview



Sequence diagram (login and detect corruption)



State machine diagram (create, delete, read, write, rename files)