ECE 422 Project 2

Prepared By Team Goon (Justin Javier, Saif Husnain)

GitHub: https://github.com/AnotherDayOfTrying/ece-422-lab2

Abstract

This report details the design, development, and progress of Secure File System which encrypts both files and directories. The implementation allows users to create their own files and directories and have protections based on permissions each user has. Any users that do not have permissions to view certain files will not be able to access them. This is done through encryption using python libraries. The project will display the importance of protecting user privacies via encryption and making sure your entire system is secure and users can put their trust in it and utilize it for their important information.

Detailed Design

Creating a user involves inputting a username and password. The password is salted and hashed first and then saved within our database. As an account is created, a key is generated for the user, this key is used to access documents the user owns. The user has a folder created for them at the root of the system, which is their home. Upon login, the user's info is confirmed with the database and if they match, the user is allowed into the system. Encryption is done using the crypto library in Javascript. We use the Advanced Encryption Standard algorithm in nodejs. We create a secret key using a general admin password and we create a 16 byte initialization vector. Then using the crypto library we create a cipher object and then update the input text and finalize the encryption by concatenating it with the initialization vector. For permissions, we have groups that can be created by admins. These groups carry permissions and have their own keys associated with them. These groups contain users that have privy to those permissions. Files have permissions on their metadata and if a user is a part of a group with the permissions to access those files are granted access via the group key. Overall, the implementation is tied together via a cli made using the yargs library. There are commands for a user to maneuver through the system, attempt to read a file, write to file, etc. This is how users navigate and use the SFS.

Tools and Technologies

Cybera: Hosting the applications. Cybera is a cloud resource management service that allows us to start up VMs to host our application on. Cybera was chosen because it was mandated in ECE 422.

Node.js: Choose node.js for implementation because of its high-level language capabilities in addition to our group’s programming experience in Node.js. Furthermore, Node.js has a built-in library “crypto” which gives us the ability to hash, encrypt, decrypt and sign data.

Typescript: Typescript is a superset of Javascript that gives us types. This helps with development by increasing readability, giving runtime errors (prevents compiling erroneous code), and enabling intellisense in the IDE.

NPM: Node Package Manager allows us to use community created packages. We plan on using packages to abstract components of the project such as the CLI.

Bun: Bun is a competitor to Node.js and provides a native runtime for typescript. In addition, bun compiles faster than Node.js.

Yargs: Yargs is a npm package that is used to create CLIs. We chose this package because of our familiarity with it and its simplicity.

Agile: Used to plan the development of the project. This was used due its iterative nature. Our current iteration for implementation is currently limited by our inexperience in creating a similar product. Agile allows us to amend and change plans as we encounter issues.

Burndown Chart: Tracks the progress of the project according to tasks. Allows us to visualize if we are on track to complete the project before the deadline.

Mongodb: We used mongodb as our database to save users and their data such as permissions. Mongodb was chosen because it has mapping for data structures in JS.

Design Artifacts

Class Diagram:

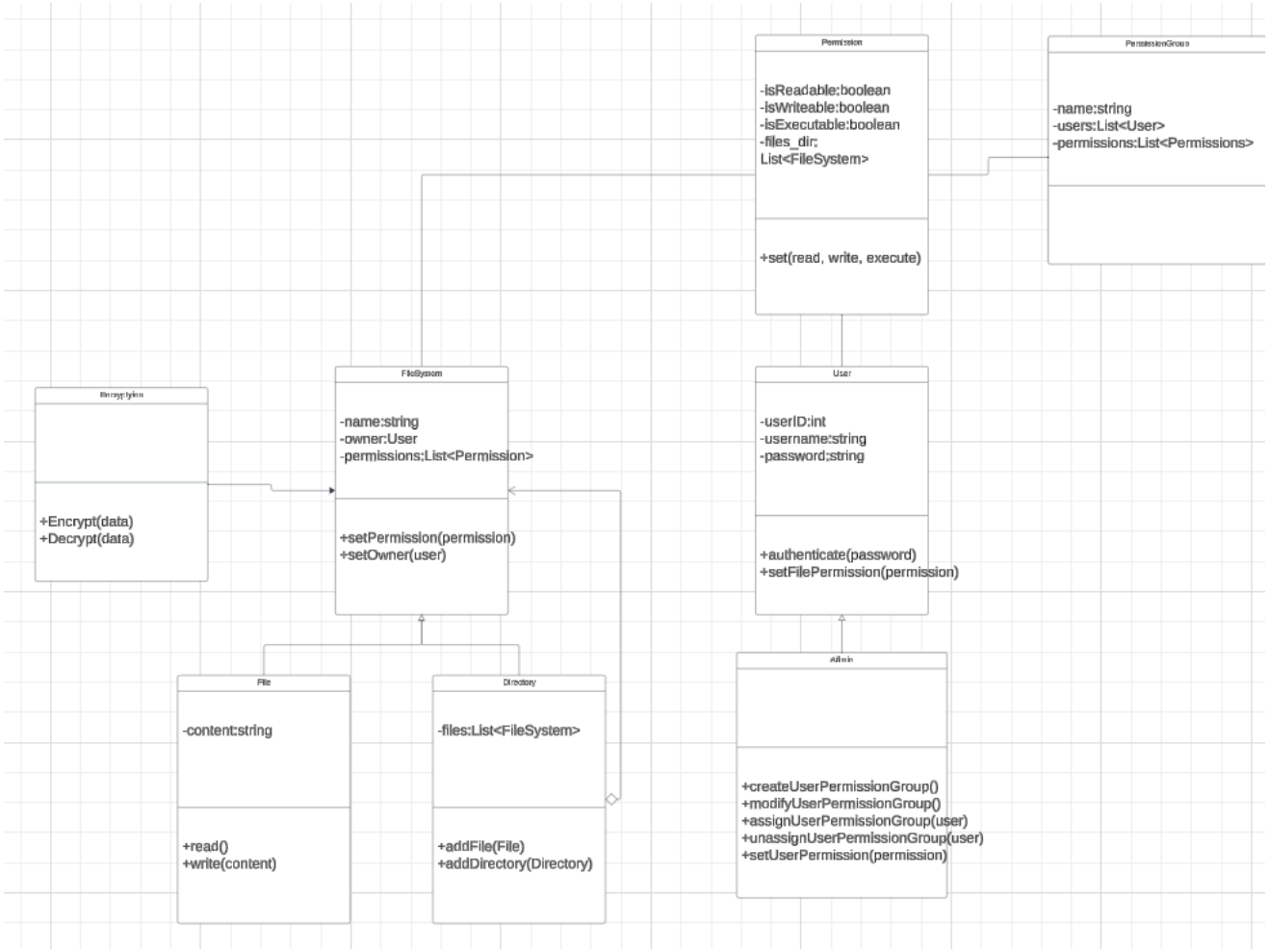

Figure 1: Class Diagram

The original class diagram that was created before beginning development on the project. The design encompasses a filesystem, which includes both directories and files. Directories can hold both more directories and files. Each file and directory are encrypted by an encryption service and decrypted by that same service. Whether a user has access to a file or directory can be told by the permissions that the user carries, which is indicated by their permission group and if they correlate to the permissions of said directory or file, otherwise they will not be allowed to see the content of that part of the filesystem.

Original Timeline:

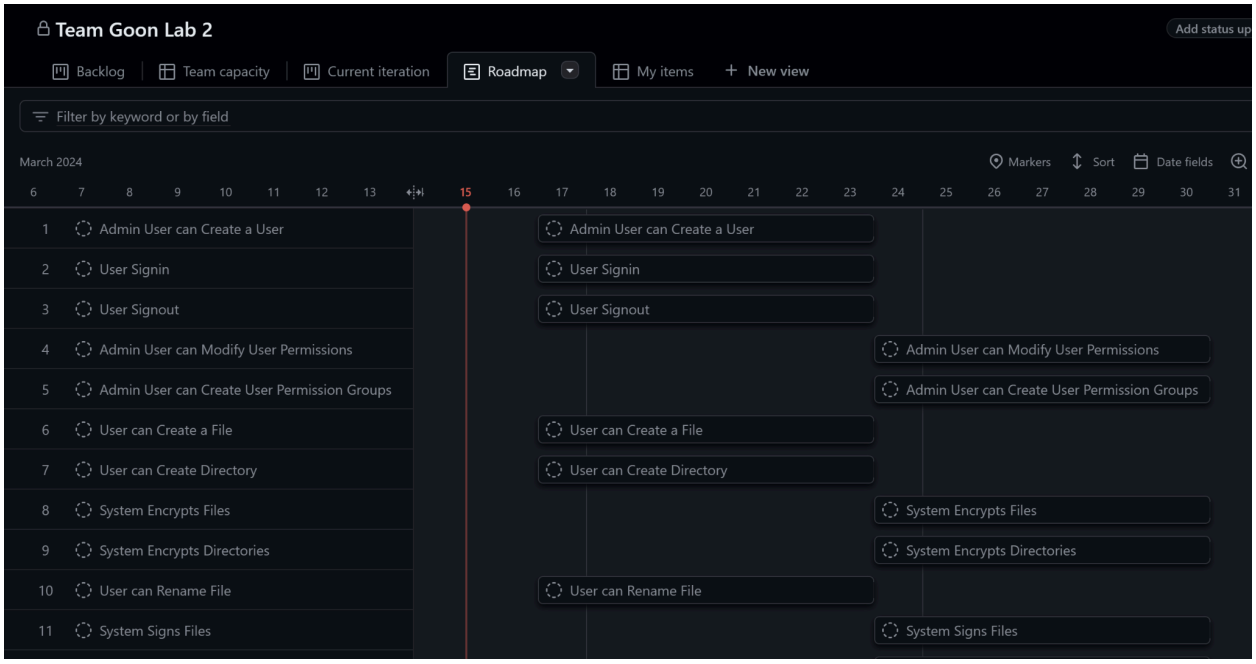

Figure 2: Original Goal Timeline

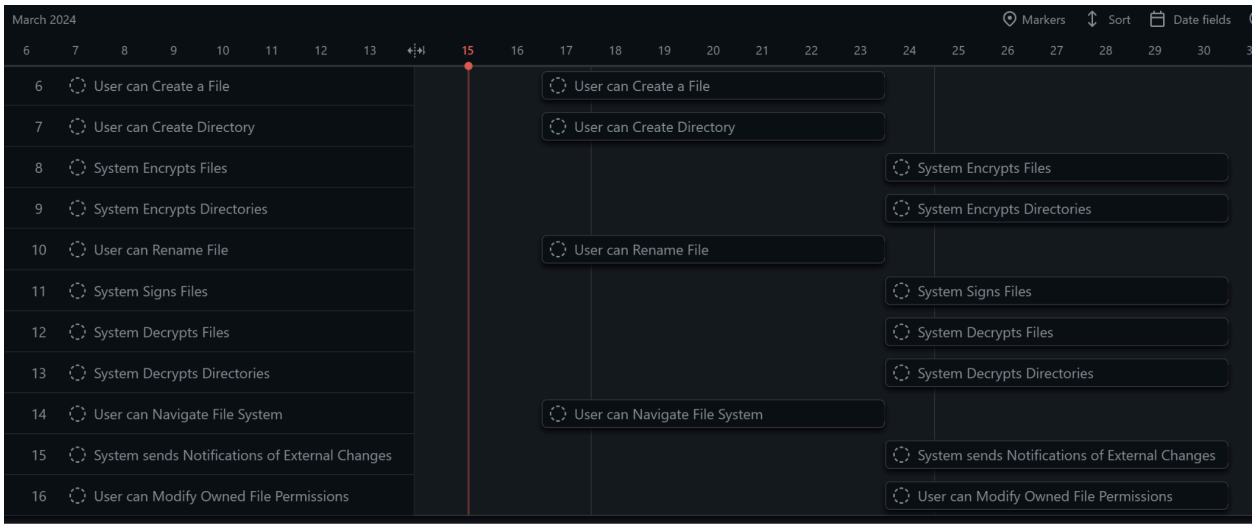

Figure 3: Original Goal Timeline Continued

Burndown Chart:

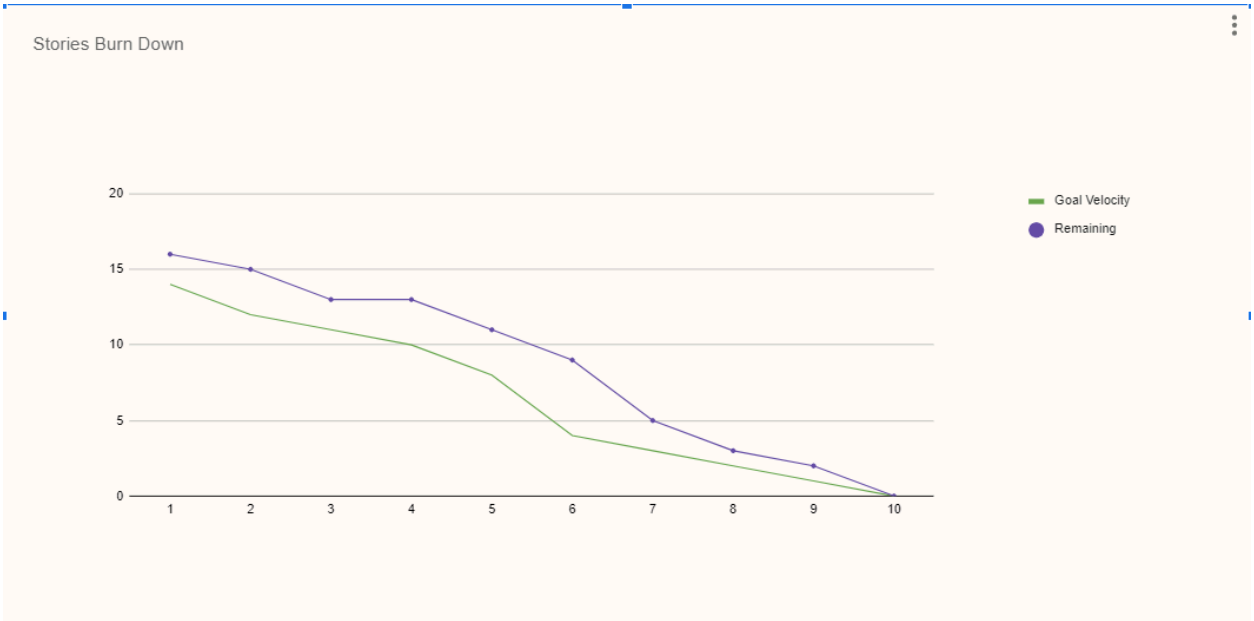

Figure 4: Burndown Chart

The burndown chart above shows the progress and timeline that was achieved. To explain the timeline, over a 10 day period we outlined how many user stories shall remain. The remaining line in contrast is what the actual timeline ended up being. Overall, we hoped to complete more user stories early on, but in the end, we ended up backloading our completion a little due to other projects taking up time. In the end, however, we were able to finish up in time and did not run into any major issues.

State Diagram:

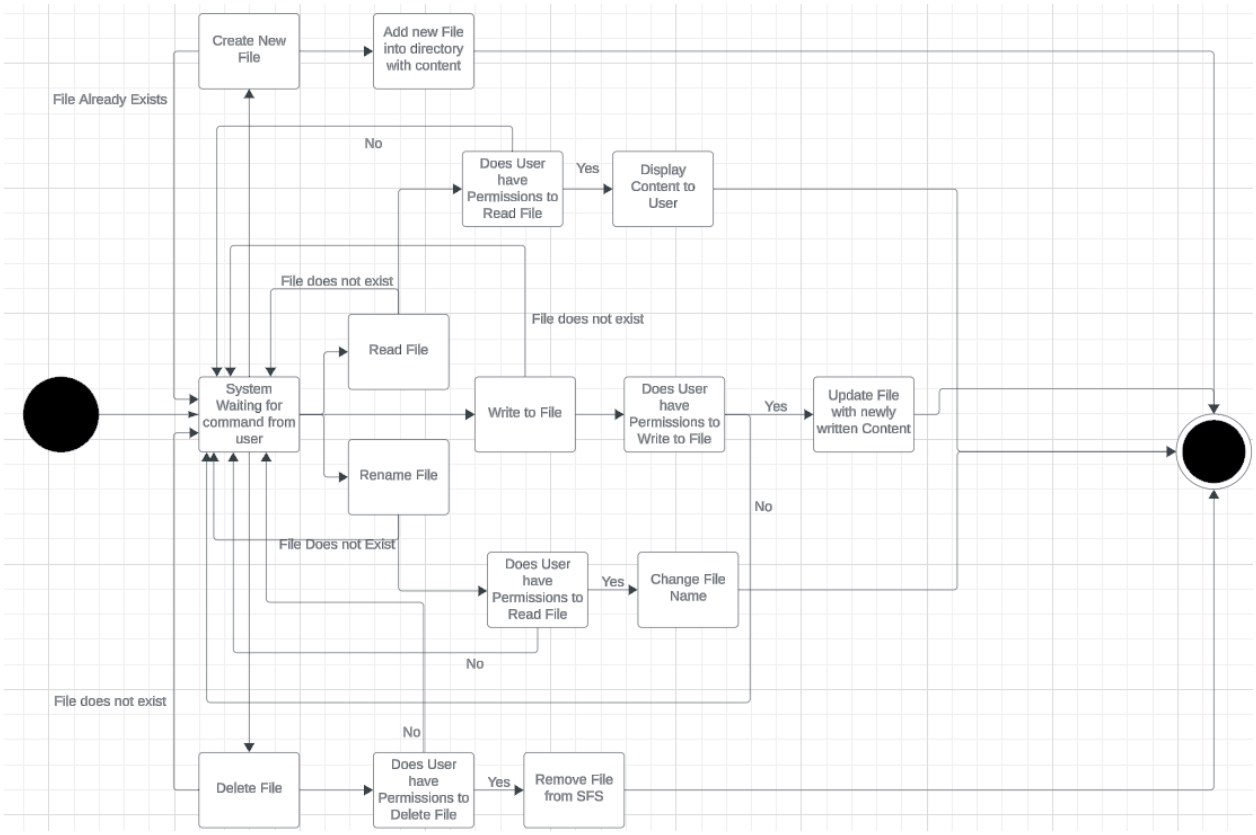

Figure 5: State Diagram

The UML state diagram is done for requirement 3: “SFS users can create, delete, read, write and rename files. However, SFS is not required to support bulk upload/download of file content (e.g. sftp or scp)”. It depicts how our system handles create, delete, read, write and rename once an internal user is logged in. Essentially it checks if the file exists, and then if the user has the correct permissions.

Sequence Diagram:

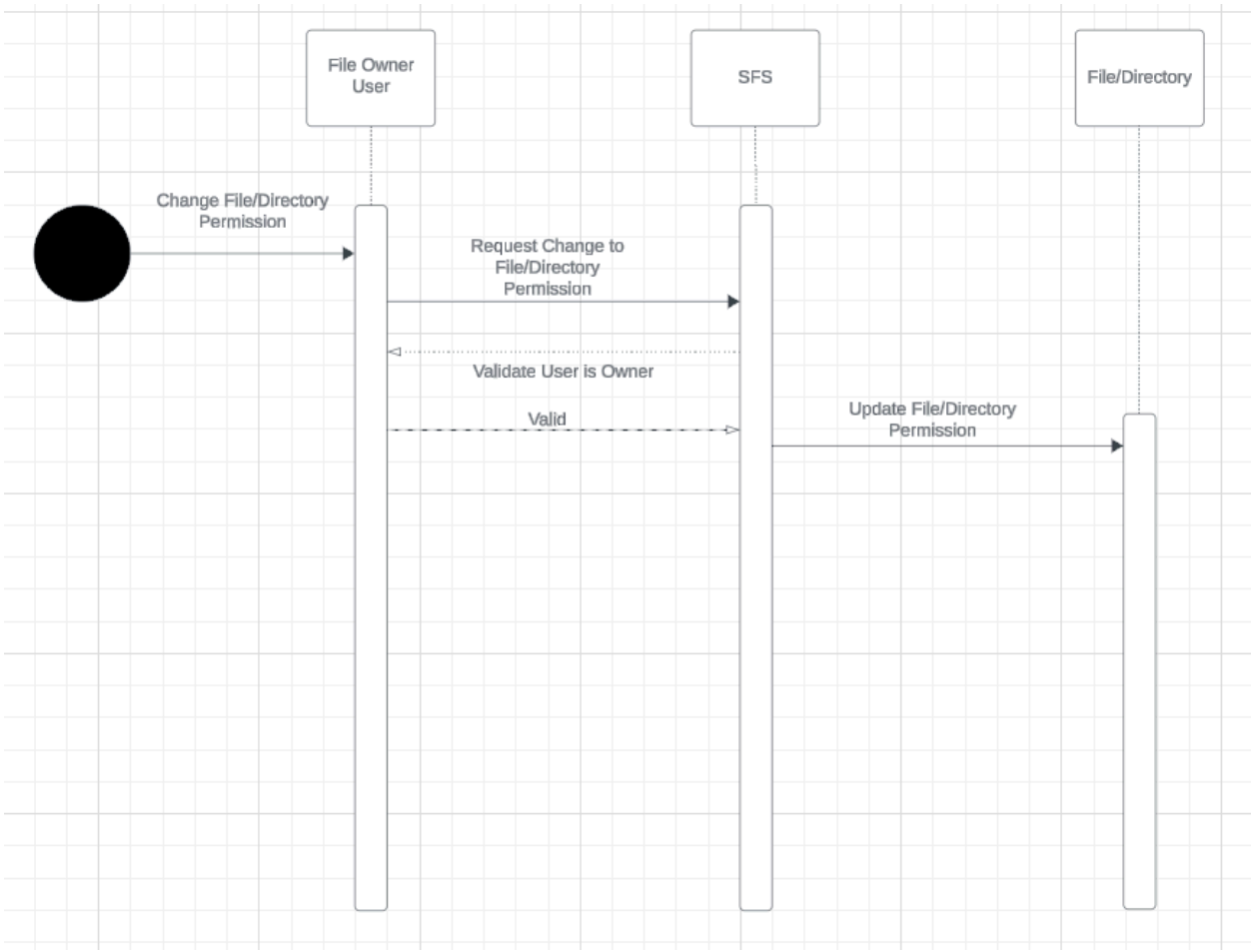

Figure 6: Sequence Diagram

The sequence diagram depicts requirement 5: “Internal users should be able to set owner/group/other permissions on files and directories. ” The owner of a file first attempts to change the permissions the file has and the system checks to see if it is in fact the owner attempting this and after it validates, the files permissions are updated as required.

Deployment Instructions

1. Clone https://github.com/AnotherDayOfTrying/ece-422-lab2
2. Install Node: https://nodejs.org/en/download/package-manager
3. Install node dependencies: npm i
4. Npm install bun: npm i bun
5. Build index.ts: bun build index.ts > index.js

6. Link sfs module: npm link
7. Install and run mongodb: https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/
8. Setup environment variables ADMIN_PASSWORD and ADMIN_SALT
   export ADMIN_PASSWORD:<hashed password in hex>
   export ADMIN_SALT:<salt in hex>
   ADMIN_PASSWORD is a hashed password. Use function crypto.pbkdf2Sync with 'sha512' encryption. Remember your password to use admin functions.

User Guide:

Simply run "sfs [command]", command list:

a. "ls" to view files in current directory
b. "mkdir [dir]" to create new directory
c. "cd [dir]" to move to directory
d. "touch [file]" to create a new file
e. "login [username] [password]" to log into a system
f. "logout" to logout
g. "rm [file]" to remove a file
h. "rmdir [dir]" to remove a directory
i. "cat" to read a file
j. "echo [file] [data]" to write to a file
k. "mv [file] [rfile]" to rename a file
l. "changePermissions [fiole]" to change permissions of a file
m. "whoami" to display which user you are
n. "pwd" to see which directory you are currently in

For admission the following commands are also available:

a. "admin createUser [username] [password] –adminpass [pass]" to create a new user
b. "admin createGroup [name] –adminpass [pass]" to create a new group
c. "admin addToGroup [username] [group] –adminpass [pass]" to add user to group
d. "admin removeFromGroup [username] [group] –adminpass [pass]" to remove user from group

Conclusion

Overall, we were able to create a SFS that was able to maintain integrity, by maintaining secure files and not allowing external users to access them. It teaches the importance

of having set rules and permissions for users, because if users cannot trust that a system they are using will not protect their private information and manage permissions properly, they will distance themselves from it. To grow an application, one must have the trust and faith of the users. To achieve this we were able to correctly implement permissions and successfully utilize encryption and decryption of both files and directories. Internal users were able to create, edit and delete their own files and set permissions for other users.