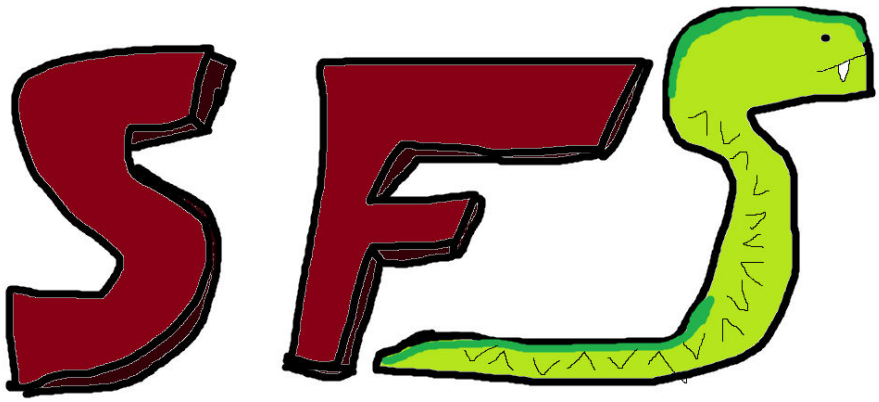
ECE 422 - PROJECT 1

Andrew Culberson - Logan Vaughan



Abstract

The purpose of this project is to develop a way to store files securely on an otherwise potentially insecure machine. We solved this problem by developing a secure file system server and client that communicate securely.

Introduction

In this project, we implemented a secure file system server (referred to as the SFS from now on) that includes both encryption of data in transit, and in storage. A corresponding client program was also written so that users can easily interact and access the SFS. We utilized three different forms of encryption: one asymmetric and two symmetric. The client CLI communicates with the SFS and allows users to create directories and files. It also allows users to manage permissions on all of their created resources. All user profiles and permission groups are managed and created by a server administrator account.

Technologies, Methodologies, Tools

The SFS is divided into a server and client that are both written in Python. The client is a command line interface that sends encrypted messages to the server which interprets them and sends back encrypted responses.

When the server is first started, the admin user must be defined. The admin user has the most control over the file system and must process certain types of user requests. After the admin user on the server is set up, the server will start waiting for a client connection. At this point, the client can be started.

When the client is started, the connection process between the server and client begins, and the server and client perform a key exchange. The key exchange is done with RSA asymmetric keys. First, the client sends a public key to the server. After receiving the client's public key, the server sends the client its public key. Then, the client sends the server a symmetric key that will be used by both the server and client to encrypt and decrypt their communications.

Now that the server and client have encrypted communication, the client can send commands to the server to interact with the file system. Originally, the user won't have access to most of the commands from the client. The user will have to either login as

the existing admin, or create a user in the client, wait for the admin to approve their user, and then login as that user. Note that user information will be stored in the server and the passwords will be hashed with SHA256.

Once the user is logged in they can start adding to their directory. The files in the SFS can be navigated with commands like cd, and ls from Unix. Also, similar to Unix, files have permissions that can be edited by the file owners. User groups also exist in the SFS which permits file sharing.

When users create files, the contents of the files are encrypted with AES in EAX mode and the file/directory names and encrypted using the Fernet Python library. Furthermore, files are given hashes which are verified on user login. The files in the SFS are also verified against a redundant structure.

Design Artifacts

See Appendix A for the design artifacts. The order of the artifacts are as follows: Overall architecture diagram, UML class diagram, Sequence diagram for user creation, and State diagram for client server communication.

Deployment Instructions

Follow these steps to set up the SFS server and client connection.

1. Procure a machine running Ubuntu
2. Download at minimum the Server and Utils folder to the machine
3. Change into the Server directory
4. Run `python main.py <port>` with port set to the port clients should connect to
5. Set the admin user password on startup
6. Ensure firewall rules are set within your environment to allow client server communication

The server should now be ready to receive client connections.

User Guide

Running the CLI:

1. Change into the Client directory
2. Run `python main.py <ip> <port>` with ip and port as the SFS's ip and port
3. One connected, a prompt "> " will appear
4. Enter the `menu` command to see a list of commands

To create a user:

1. Run `user_create <username> <password>` with the username and password of the new user. This will make a new user request for the server admin.
2. Login as the server admin, or get the server admin to run `activate_user <username>` with the username of the new user
3. You can now login as the user with `login <username> <password>`

If you set up the server admin credentials on server startup, you can log in as the server admin user with `login admin <password set on startup>`.

On login, if your files have been tampered with somehow, you will be notified immediately.

File System Navigation And File Creation:

Once you are logged in as a user, you can create files with `create <filename>` or `write <filename> <o|a> <content>`. You can delete files you created with `delete <filename>`, and rename files you have permissions to access with `rename <path to file> <new name>`. Files you have permissions to read can be read with `read <filename>`.

You can move around the file system with `cd <path or relative path>` and list contents with `ls <path or relative path>`.

Permissions:

Each file has 3 numbers describing the associated permissions. The meaning of the numbers is as follows.

Each number has the following meaning:

- 0: Neither read nor write
- 1: Read only
- 2: Read and write

The position of the number means that the permissions apply to the following user types:

- First number: File owner
- Second number: Users in file owners groups
- Third number: All other users

You can modify permissions on files you have created by using `chmod <3 numbers for permissions> <filename>`.

Discussion

Vulnerabilities:

1. Certificates were not used in this project, so man in the middle attacks are possible
2. Password hashes are not salted. If the users.json file is leaked, passwords could be cracked easily (providing they are guessable to begin with).
3. We do not validate passwords for security (length, symbols, etc.). Users can use EXTREMELY insecure passwords (such as a single character).
4. We also do not support concurrency, so only one client should be connected at a time. (Though multiple CAN connect at a time, this would likely break everything almost instantly).

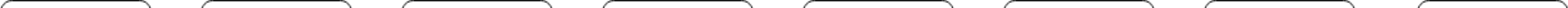
Conclusion

In conclusion, we implemented a secure file system server (referred to as the SFS from now on) that includes both encryption of data in transit, and in storage. A corresponding client program was also written so that users can easily interact and access the SFS. We utilized three different forms of encryption: one asymmetric and two symmetric. The client CLI communicates with the SFS and allows users to create directories and files. It also allows users to manage permissions on all of their created resources. All user profiles and permission groups are managed and created by a server administrator account.

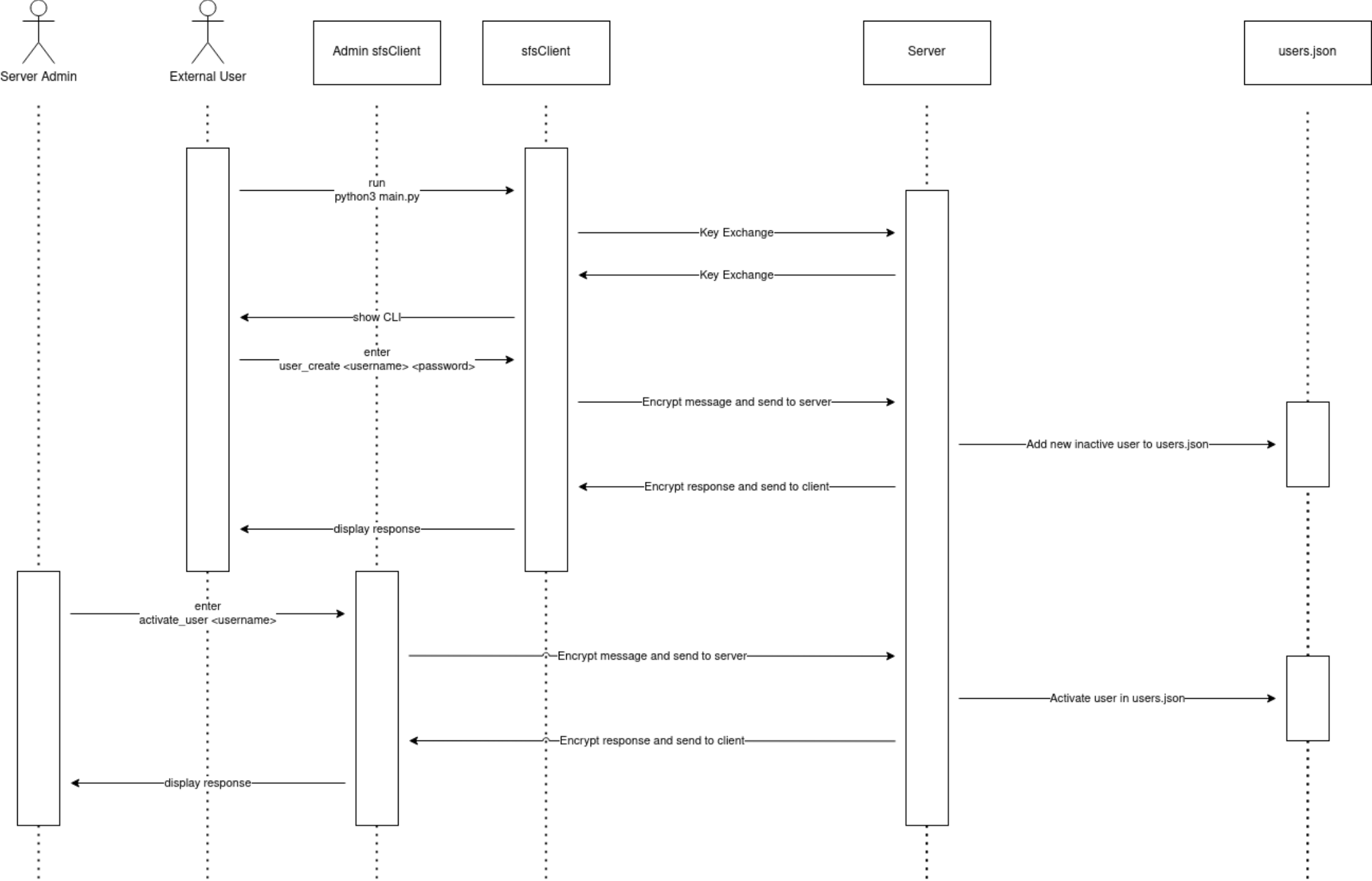
References

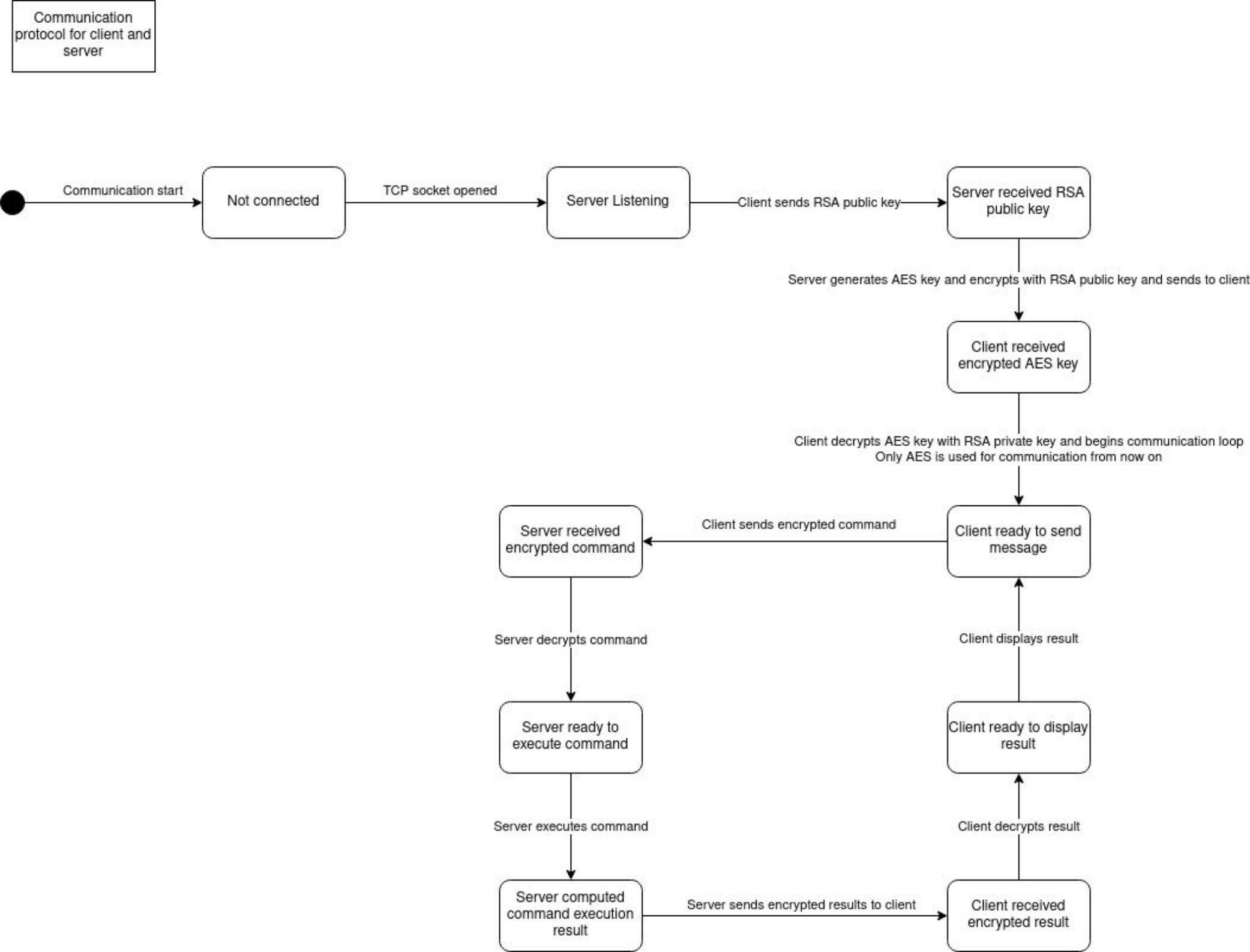
No external references used in this report.

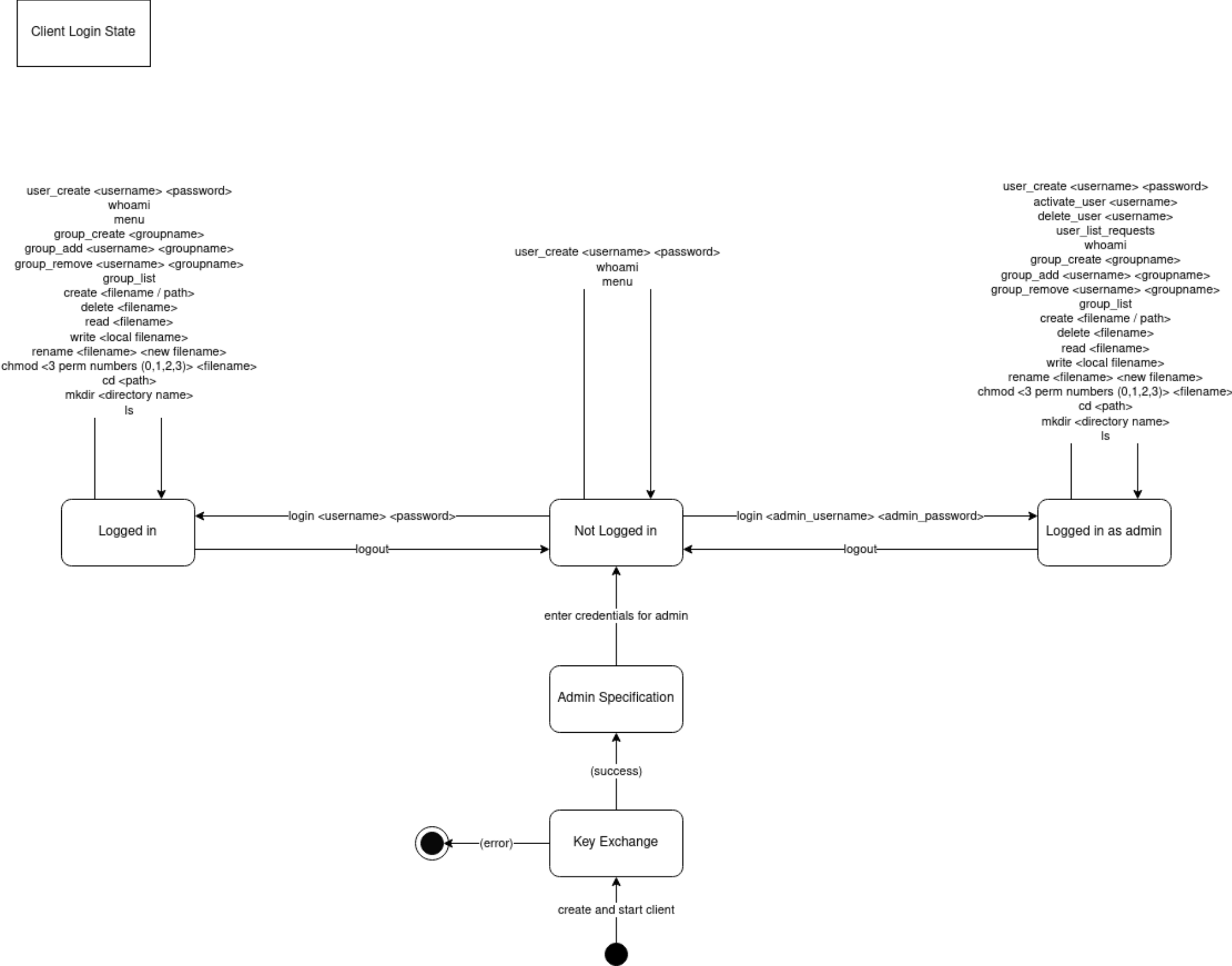
Appendix A


Client
Decryption/encryption with AES key
Internet
Cybera Firewall
Decryption/encryption with AES key
Server
Decryption/Encryption with Fernet and AES
Server Hard Drive


Client.Session
conn_crypto : AsymmetricCryptoManager
conn : socket
symmetric_cipher : SymmetricCryptoManager
key_exchange() : bool
handle() : None
Server.Session
conn : socket
pwd : str
username : str
keys : SymmetricCryptoManager
get_cwd() : str
set_cwd() : None
get_username() : str
set_username() : None
get_keys() : SymmetricCryptoManager
get_conn() : socket
logout() : bool
Utils.AsymmetricCryptoManager
private_key : RsaKey
public_key : RsaKey
master_cipher : PKCS10AEP_Cipher
encrypt(contents : bytearray) : bytes
decrypt(contents : bytearray) : bytes
get_public_key() : bytes
Utils.SymmetricCryptoManager
key : bytes
encrypt(contents : bytearray) : bytes
decrypt(contents : bytearray) : bytes
get_key() : bytes
Utils.FernetCryptoManager
key : bytes
cipher : Fernet
encrypt(contents : bytearray) : bytes
decrypt(contents : bytes) : bytes
get_key() : bytes
Server.SecureFileSystemServer
authenticator : Authenticator
filemanager : FileManager
handle_user_create(args : list, session : Session) : None
handle_whoami(session : Session) : None
handle_login(args : list, session : Session) : None
handle_logout(session : Session) : None
handle_menu(session : Session) : None
handle_pwd(session : Session) : None
handle_cd(args : list, session : Session) : None
handle_ls(args : list, session : Session) : None
handle_mkdir(args : list, session : Session) : None
handle_write(message : str, session : Session) : None
handle_read(args : list, session : Session) : None
handle_chmod(args : list, session : Session) : None
handle_group_create(args : list, session : Session) : None
handle_group_add(args : list, session : Session) : None
handle_group_remove(args : list, session : Session) : None
handle_group_list(args : list, session : Session) : None
handle_group_join(args : list, session : Session) : None
handle_group_list_requests(args : list, session : Session) : None
handle_user_list_requests(args : list, session : Session) : None
handle_create(args : list, session : Session) : None
handle_delete(args : list, session : Session) : None
handle_rename(args : list, session : Session) : None
handle_activate_user(args : list, session : Session) : None
handle_delete_user(args : list, session : Session) : None
send(session : Session, message : str) : None
handle_command(message : str, session : Session) : None
handle_conn(conn : socket) : None
Server.FileManager
file_crypto : SymmetricCryptoManager
filename_crypto: FernetCryptoManager
files : dict
home_path : str
base_path: str
get_user_info : func
new_user_dir(username : str) : None
remove_user_dir(username : str) : None
encode_filename(filename : str) : str
decode_filename(filename : str) : str
has_permission(file : dict, session : Session) : str | None
save() : None
fix_path(path : list, session : Session) : list
find_encrypted_filename(filename : str, file_obj : dict) : str
get_file_list(file_obj : dict, session : Session) : str
make_os_directories(encrypted_path : list) : bool
write_os_file(encrypted_path : list, content : str) : None
read_os_file(encrypted_path : list) : bytes
delete_os_file(encrypted_path : list) : bool
rename_os_file(encrypted_path : list, new_name : str) : bool
cd(path : list, session : Session) : str | bool
ls(path : list, session : Session) : str
mkdir(name : str, session : Session) : bool
write(path : list, overwrite_flag : str, content : str, session : Session) : bool
read(path : list, session : Session) : str
chmod(perms : str, path : list, session : Session) : bool
delete(path : list, session : Session) : bool
rename(path : list, new_name : str, session : Session) : bool
create(path : list, session : Session) : bool
verify_integrity(username : str) : list
verify_integrity_dfs(file_obj : dict, total_path : str, path : list) : list
verify_integrity_hash(file_obj : dict, total_path : str, path : list) : list
encrypt_file_contents(plain_contents : str) : bytearray
decrypt_file_contents(encrypted_contents : bytes) : bytearray
Server.Authenticator
users : dict
groups: dict
admin_name : str
user_save() : None
group_save() : None
new_user(username : str, password : str) : bool
delete_user(username : str, session : Session) : bool
user_list_requests(session : Session) : str | bool
activate_user(username : str, session : Session) : bool
authenticate_user(username : str, password : str) : bool
get_user(username : str) : dict
group_create(group_name : str, session : Session) : bool
group_add(group_name : str, new_user : str, session : Session) : bool
group_remove(group_name : str, user : str, session : Session) : bool
group_list(session : Session) : str
group_join(group_name : str, session : Session) : str
group_list_requests(group_name : str, session : Session) : str


Server Admin
External User
Admin sfsClient
sfsClient
Server
users.json
run
python3 main.py
Key Exchange
Key Exchange
show CLI
enter
user_create <username> <password>
Encrypt message and send to server
Add new inactive user to users.json
Encrypt response and send to client
display response
enter
activate_user <username>
Encrypt message and send to server
Activate user in users.json
Encrypt response and send to client
display response


Communication protocol for client and server
Communication start
Not connected
TCP socket opened
Server Listening
Client sends RSA public key
Server received RSA public key
Server generates AES key and encrypts with RSA public key and sends to client
Client received encrypted AES key
Client decrypts AES key with RSA private key and begins communication loop
Only AES is used for communication from now on
Client ready to send message
Client sends encrypted command
Server received encrypted command
Server decrypts command
Server ready to execute command
Server executes command
Server computed command execution result
Server sends encrypted results to client
Client received encrypted result
Client decrypts result
Client ready to display result
Client displays result


Client Login State
user_create <username> <password>
whoami
menu
group_create <groupname>
group_add <username> <groupname>
group_remove <username> <groupname>
group_list
create <filename / path>
delete <filename>
read <filename>
write <local filename>
rename <filename> <new filename>
chmod <3 perm numbers (0,1,2,3)> <filename>
cd <path>
mkdir <directory name>
ls
Logged in
login <username> <password>
logout
user_create <username> <password>
whoami
menu
Not Logged in
login <admin_username> <admin_password>
logout
user_create <username> <password>
activate_user <username>
delete_user <username>
user_list_requests
whoami
group_create <groupname>
group_add <username> <groupname>
group_remove <username> <groupname>
group_list
create <filename / path>
delete <filename>
read <filename>
write <local filename>
rename <filename> <new filename>
chmod <3 perm numbers (0,1,2,3)> <filename>
cd <path>
mkdir <directory name>
ls
Logged in as admin
enter credentials for admin
Admin Specification
(success)
Key Exchange
(error)
create and start client