CS161 Project 2 Design Document
Saifullah Jailani and Haroun Khaleel

Data Structures

- **Map PubUsers:** A map that contains the user name as key and the RSA public key as value. It will be used in file transfers.
- **Map DataStore:** A map that contains the username as key and salted_password as value.
- **Map AllUsers:** A map that contains the username as key and path to where the user struct is stored as value.
- **Shared Directory:** A map that contains the username as key and another map as value that holds the sender as key and a path as value.
- **Struct User**
    - Username: The username is passed into the InitUser() and GetUser() functions (See "User Authentication" below) and is unique.
    - Password: Key generated using the PBKDF with the password passed in as an argument by the user when InitUser() is called combined with the username as a salt (Since username is unique). Store the salted password to the Map of PassUsers.
    - Key Pair: Public and Private Key generated using the PKEKeyGen() function
    - Owned Files: A map with references to the File objects that the user owns as well as a mapping of each File object to the top level users that it is shared with
    - Shared Files: A map with references to the File objects that have been shared with the user as well as a mapping of each File object to the top level users that it is shared with
    - **Fetch things from Datastore rather than fetching them locally in order to get this part**
- **Struct UserInteg (User Helper Struct) (not needed if we store hash as first line in datastore)**
    - User: Stores a reference to the User struct and is created along with that struct when InitUser() is called (See "Functions" below)
    - UserHMAC: An HMAC of the User struct
- **Struct File**
    - Struct File prev, Struct File next, Struct FileInteg integ.
- **Struct FileInteg (File Helper Struct)**
    - File: Stores a reference to the File Struct and is created along with the File Struct itself when StoreFile() is called
    - FileHMAC: An HMAC of the File Struct
    - FileKey
- **Struct Sharing**
    - String Keys, File HMAC, Struct File

User Authentication:

- InitUser(username string, password string):

    - Create a new user struct using the passed in arguments.
    - Generates the RSA private and public keys. Stores the private key and adds the public key to the users. Generate owned directory key salted with the user name and password salted with some deterministic random variable.
    - Initialize the owned files map, and store the encrypted bytes to the owned directory.
    - Store the struct in Datastore with the key derived from the PBKDF as Key and the HMAC of the User struct concatenated with the entire structure converted to bytes (and encrypted somehow) as the Value.
    - Adds the username and the path to the user struct to AllUsers.
- GetUser(username string, password string):
    - Uses PBKDF and the provided username and password to generate a key. We then check the Datastore for this key. If the corresponding data exists, we get the struct file from AllUsers and first read the first 64 bytes of data (the HMAC) and then compare it to the HMAC of the rest of the data. If these values match, return the struct data. If these values do not match or if there is no corresponding data, return an error.

File Storage and Retrieval:

- LoadFile(file name): this function will check the files in the Owned file directory, and shared file directory. If the file is found then we load the file and use the fileKey, and fileHmac to decrypt this file.
- StoreFile(filename string, content []byte): Adds the file to the owned files directory, and encrypts the data using HMAC key and file key and file name and len of the content as the salt. This part will be encrypted using a cipher block text algorithm where the hash of the bytes of all previous blocks will be the salt for the next part. And this part is done in a reversed order such that the last part is the head.
- AppendToFile(filename, contect []byte): For this part we will be loading the encrypted linked lists and using their bytes and the user name calculate the encryption key and push it to the front of the file struct.

File Sharing and Revocation:

- CreateInvitation(file, username): create shared file struct and add the necessary keys required for file operations. Encrypted with the RSA public key of the recipient and add it to the shared Directory map of the user. Also generate the hmac using the recipient password hash and sender password hash salted with the length of the content.
- AcceptInvitation(sender,file): It will find the file that the sender shared from a shared Directory. Then it will decrypt it using its own private key. After that it is decrypted. It will calculate the HMAC using the self password hash and the sender's password hash and the length of the content as salt. It will compare the HMAC if the file exists than indeed to a self shared directory. It will add the path to the self shared directory.
- RevokeFile(): Load the file, change the keys. Store it again and delete the old one.

Test Cases (6):

Design Requirement: The client MUST support a single user having multiple active sessions at the same time. All file changes MUST be reflected in all current user sessions immediately. [3.2.2]

1. Define CreateMulSessions(String username, String password, int NumSessions), a method which first calls InitUser() with the username and password. It then creates and returns as many sessions for that user as specified in a list.
2. Use one of these sessions (Session A) to create a file.
3. Assert that this file shows up in another session's (Session B) files without closing Session A.
4. Append to the file using Session B and assert that the edits show up on Session A's file without closing Session B.

Design Requirement: If the client is restarted, it must be able to pick up where it left off given only a username and password. Any data requiring persistent storage MUST be stored in either Keystore or Datastore.

1. Create a new user using a username and password
2. Create a new file
3. Append to the file
4. Save the contents of the file as a variable
5. Close the user
6. Open the user again with just the username and password
7. Assert that the contents of the File are exactly the same as they were before we closed the user

Design Requirement: Filenames MAY be any length, including zero (empty string). [3.5.6]

1. Create a new user using a username and password
2. Create a new file with an empty string as the name
3. Append to that file
4. Close/Reopen the user
5. Ensure that the file contents are still the same

Design Requirement: The client MUST NOT assume that filenames are globally unique. [3.5.7]

1. Create a new user using a username and password (User A)
2. Create another new user using a username and password (User B)
3. Create a file using User A and append something to it
4. Create a file using User B and append content that it different from that of User A's file
5. Close/Reopen both users
6. Assert that each user's file contents are the same as they were before

Design Requirement: The client MUST enforce authorization for all files. The only users who are authorized to access a file using the client include: (a) the owner of the file; and

(b) users who have accepted an invitation to access the file and that access has not been revoked. [3.6.1]

1. Create 3 new Users. (Users A, B, and C)
2. Create a file using User A
3. Share this file with Users B and C
4. Revoke User C's access to this file
5. Append a word to the file using User A
6. Close/Reopen all Users
7. Assert that the word is present in the file for Users A and B but not for User C

Design Requirement: The client MUST prevent any revoked user from using the client API to take any action on the file from which their access was revoked. [3.6.10]

1. Create 2 new Users (User A and B)
2. Create a new file using User A
3. Share this file with User B
4. Append to the file using User B
5. Assert that the content appended to the file matches the content of the same file when viewed by User A
6. Revoke permission for the file from User B
7. Attempt to append to the file using User B
8. Assert that this call returns an error message and does not allow User B to append
9. Attempt to append to the file using User A
10. Assert that this call does not return an error message