■Design document

For InitUser function, since the username must be confidential to the dataStore and other attackers, we decided to take bytesToUUID(HMACEval(key1, username)) as a user's UUID key for the dataStore. We used bytesToUUID because we needed a deterministic way to map a username to a UUID, and we used HMACEval on the username to prevent usernames with the same last 16 bytes to map to the same UUID. Meanwhile, the password has a "good" entropy, so we can use the password to generate a high-entropy sourceKey using Argon2Key function and we will use the sourceKey to generate more keys for encryption/decryption and digital signature. Because the dataStore is untrusted, we store the encrypted userdata using AES-CBC symmetric encryption and an HMAC on the encrypted data in order to ensure confidentiality and integrity. We use symmetric keys for encryption & signing since we can derive the symmetric key from the user's password, hence, we don't need to store the symmetric keys anywhere in the server. The encrypted userdata will then contain the asymmetric RSA encryption secret key, digital signature secret key and the mapping from shared_filename to its shared keys so that we can use those keys when the file is shared.

We had two options of key management between users and files in the datastore. One is to generate one symmetric encryption key and one HMAC key that are both associated with each user and use those keys for all files. The other option is to generate new keys associated with each file. We decided to generate new keys k3 and k4 for encryption and HMAC using the sourceKey with HKDF for each file, where k3 and k4 are derived from the secret filename. The file contents will also be stored in a list of encrypted data, so future calls to AppendFile will simply add another encrypted value to the list. When we LoadFile, these list entries can be concatenated into one list again to read and write.

When Alice shares file1 with Bob and Carol, Bob and Carol can only access the file that was shared. More specifically, when Alice calls ShareFile to share file1 with Bob, she will first delete the original_file1 entry encrypted with k3 and k4, and create a new_file1 entry with contents encrypted with the shared keys (k6 and k7) between Alice and Bob. The value "sharing" will be the encryption of k6 and k7, which will be shared from Alice to Bob. Now, when Bob calls ReceiveFile, Bob then receives k6 and k7 and uses them to calculate the new_file1 entry in the dataStore to read/write the file. If Bob shares this file with Carol, since Bob is not the owner, he will not create another new_file1 entry, but will simply share the encrypted values of k6 and k7 to Carol. When RevokeFile is called, Alice can easily delete the entry in the datastore by deleting new_file1 and rederiving the original_file1 encrypted with k3 and k4. The choice of resetting her file's private keys is also if Alice

calls ShareFile then calls RevokeFile before Bob calls ReceiveFile, Bob still shouldn't access Alice's file because the keys would have changed back.
Another point of concern was how filenames would be stored in the datastore. We needed to make sure same filenames were allowed and make sure situations where filename = another username and same filename for two different users are also allowed. To do this, we encrypted filenames using the key which was derived from the owner's password, by taking bytestoUUID(HMAC(key4, filename)), where the key4 is based on the owner's password, allowing files with the same filename from different users to map to different UUIDs in the datastore.

■Security Analysis

- <u>Brute Forcing the UUID</u>: One possible attack is brute forcing the key for dataStore. Depending on how we compute the UUID for the dataStore, an attacker can easily brute force the UUID. In order to make sure that all UUIDs for username and filename have high entropy, we used HMAC to create a high entropy key which will always be preserved when converted to UUID. This holds true because the key for HMAC is derived from the sourceKey which was derived from Argon2Key function given user's password and we can assume in this project that users have good passwords.

- <u>Man In the Middle Attack</u>: When Alice shares a file with Bob, there could be an Eavesdropper who sees the sharing and all other messages transmitted through network. If we simply use the sharing as a key for the dataStore, Eve can overwrite/delete the data easily. In order to defend from this vulnerability, we used HMAC(key6, sharing) as a key for the dataStore where key6 is shared with Alice so that Eve cannot compute the UUID from sharing. This increases the security level because Eve now has to randomly guess the UUID in order to overwrite the data. Also, we must have confidentiality to defend from Eve. We used RSA public key encryption to make sure that the symmetric key and the sharing message is encrypted throughout the communication. We also made sure that those encrypted data has integrity by using RSA digital signature so that Bob can detect if the data that Bob received was not modified.

- <u>Overwrite the data stored in dataStore</u>: If an attacker has a list of keys for dataStore, an attacker can overwrite or delete the data stored in the dataStore. In order to prevent an attacker with keys to be able to overwrite data, we ensure integrity on all entries stored in the dataStore. Before calling loadFile and getUser, we will always use our private symmetric keys to verify that the file and user entries are valid and untampered with. That way, we can detect if any entries we own in the dataStore have been overwritten or not.