CS 161 Project 2 Design Document

Data structures:

What data structures are you going to use to organize data on Datastore?

The User Struct will store Username, Password, Private Key, Sign Key, CreatedFiles, ReceivedFiles, and CreatorMap. The Private Key is generated with the user's Public Key and the Sign Key is generated with the user's Verify Key. These keys will be useful for encryption/decryption. CreatedFiles is a map of filenames to a map of recipients to the UUID of the invitation the user created for this recipient. ReceivedFiles is a map of filenames to the UUID of the invitation the user received. CreatorMap is a map of filenames to the source key for the UUID of the file struct. These maps are useful for storing, appending, loading, and revoking files.

The UserHMAC Struct will contain the Encrypted User Struct and HMAC of the Encrypted User Struct.

The Ciphertext Struct will contain Encrypted Ciphertext, Encrypted UUID to Next Ciphertext Struct, HMAC of Encrypted Ciphertext, and HMAC of Encrypted UUID to Next Ciphertext.

The File Struct will contain the Encrypted Start Ciphertext UUID, HMAC of the Encrypted Start Ciphertext UUID, Encrypted End Ciphertext UUID, HMAC of the Encrypted End Ciphertext UUID, Encrypted Block Index, HMAC of the Encrypted Block Index.

The Invitation Struct will contain the Encrypted Source Key, HMAC of Encrypted Source Key, and Key for HMAC of Encrypted Source Key.

The InvitationHMAC Struct will contain the Invitation Struct, HMAC of Invitation Struct, Key for HMAC of Invitation Struct, and Signature.

User Authentication:

How will you authenticate users?

Each User struct is stored in the DataStore with its key as the UUID derived from the Argon2Key(user's username + user's password). So one can only access their user by inputting their correct username and password when using GetUser(username, password). Since UUID is unique, one cannot have a different password or username for the same UUID. In other words, the username and password would have to be exactly the same to access the User struct.

How will you ensure that multiple User objects for the same user always see the latest changes reflected?

We can ensure that multiple User objects for the same user always sees the latest change reflected by storing all information on DataStore. Since the user can access the File Struct with Argon2Key(Username + Password + File Name), it is not dependent on the User object as long as the user is properly logged in and knows its username and password. Additionally, by "storing" the file's ciphertext in the File Struct through a pointer, we can ensure that the user will see the latest changes as the plaintext can update outside the File Struct.

File Storage and Retrieval:

How does a user store and retrieve files?

When a file is created, its contents will be encrypted in Ciphertext Structs of fixed plaintext lengths. We structure this as a linked list where Ciphertext Structs contain ciphertext and UUID of the next ciphertext struct . We ensure its integrity with HMAC for both values. We then store the UUID of the first ciphertext struct, the UUID of the last ciphertext struct, and the block index (which is used for revoke) in a file struct. We encrypt these values with HashKDF keys we generate based on the random source key stored in CreatorMap. The UUID of the file struct will be created from the random source key. The user can retrieve this file by accessing the source key in CreatorMap, converting it to UUID. Then, the user can retrieve the file struct, decrypting the UUID of the Start Ciphertext Struct with HashKDF. We then iterate through the linked Ciphertext Structs, decrypting the ciphertext and the UUID of the next Ciphertext with HashKDF.

Efficient Append:

What is the total bandwidth used in a call to append?

By structuring the ciphertext struct to be a linked list, the only piece of data that we need to upload and download when appending is the content we want to append. This is because we store the UUID of the End Ciphertext Struct. Therefore, we can locate the end of the ciphertext linked list without iterating through the entire ciphertext linked list. The total bandwidth is solely dependent on the amount of bytes in the content we want to append.

File Sharing:

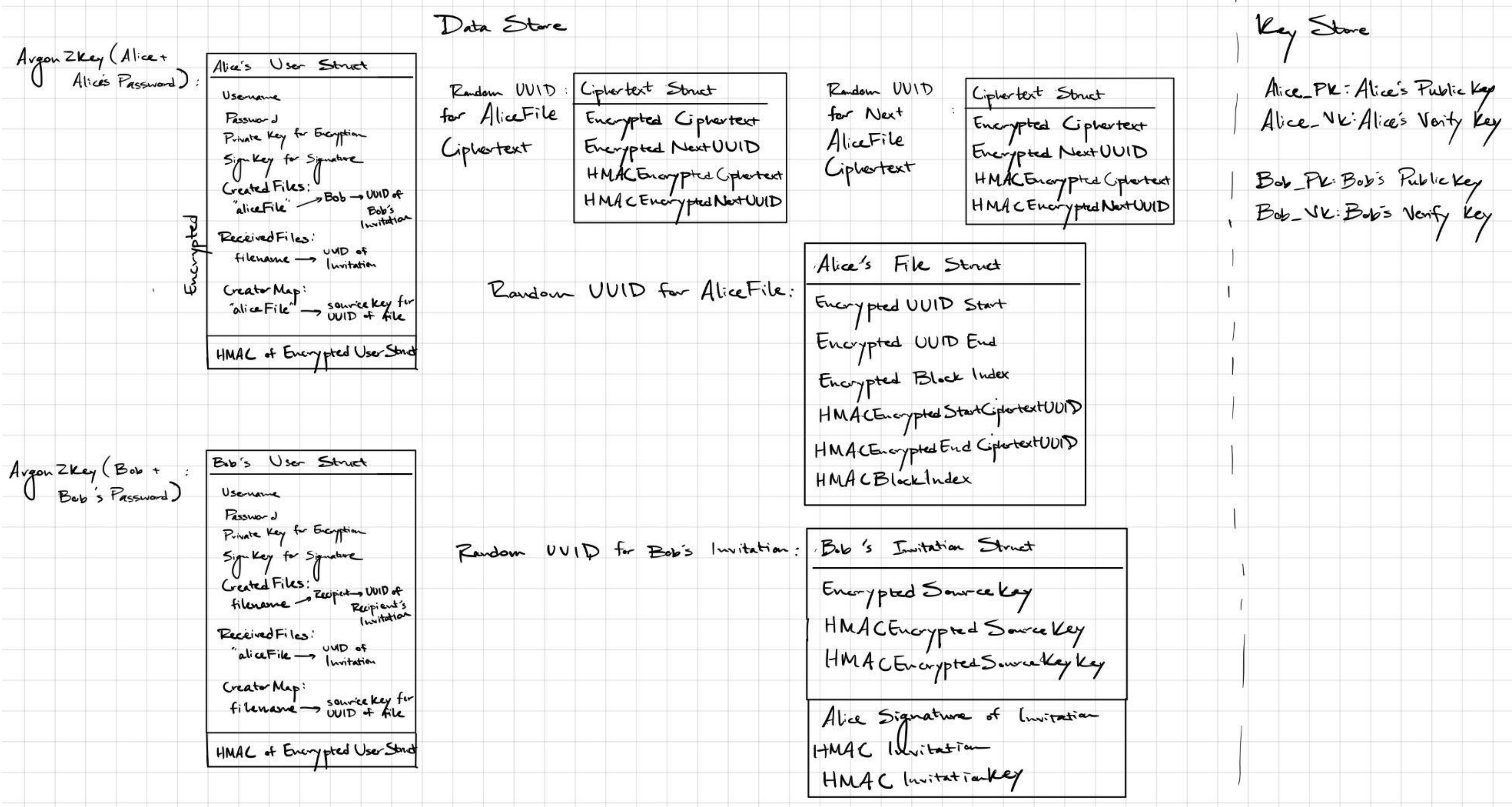
What gets created on CreateInvitation, and what changes on AcceptInvitation?

The Invitation Struct and InvitationHMAC Struct is created on CreateInvitation, with a random UUID created by the sender. The Username of the recipient and the random UUID of the InvitationHMAC Struct will be stored in the creator's CreatedFiles map in the User Struct. On AcceptInvitation, a key-value pair is created in the recipient's ReceivedFile map in the User Struct. This will map the (potentially new) filename after accepting the invitation and the UUID of the invitation. This invitation will have information to access the ciphertext linked list.

File Revocation:

What values need to be updated when revoking?

The creator's CreatedFile map must delete the revoked user and their invitation UUID. To ensure the file cannot be accessed by the revoked user and all other users that the file was shared to through the revoked user, we will set new random UUIDs for the file struct and all ciphertext structs. We will delete the file struct and ciphertext structs associated with the previous UUID's in the DataStore. Therefore, if the revoked recipient and anyone else that received the file from that recipient (their children) tries to access the file struct or ciphertext structs through the information from the invitation or by memorizing UUID's, they will find nothing in the DataStore.


Data Store
Key Store
Argon2Key (Alice + Alice's Password) :
Alice's User Struct
Username
Password
Private Key for Encryption
Sign Key for Signature
Created Files:
"aliceFile" → Bob → UUID of Bob's Invitation
Received Files:
filename → UUID of Invitation
Creator Map:
"aliceFile" → source key for UUID of file
HMAC of Encrypted User Struct
Encrypted
Argon2Key (Bob + Bob's Password) :
Bob's User Struct
Username
Password
Private Key for Encryption
Sign Key for Signature
Created Files:
filename → Recipient → UUID of Recipient's Invitation
Received Files:
"aliceFile → UUID of Invitation
Creator Map:
filename → source key for UUID of file
HMAC of Encrypted User Struct
Random UUID for AliceFile Ciphertext :
Ciphertext Struct
Encrypted Ciphertext
Encrypted Next UUID
HMAC Encrypted Ciphertext
HMAC Encrypted Next UUID
Random UUID for Next AliceFile Ciphertext :
Ciphertext Struct
Encrypted Ciphertext
Encrypted Next UUID
HMAC Encrypted Ciphertext
HMAC Encrypted Next UUID
Random UUID for AliceFile:
Alice's File Struct
Encrypted UUID Start
Encrypted UUID End
Encrypted Block Index
HMAC Encrypted Start Ciphertext UUID
HMAC Encrypted End Ciphertext UUID
HMAC Block Index
Random UUID for Bob's Invitation :
Bob's Invitation Struct
Encrypted Source key
HMAC Encrypted Source Key
HMAC Encrypted Source Key Key
Alice Signature of Invitation
HMAC Invitation
HMAC Invitation Key
Alice_PK: Alice's Public Key
Alice_VK: Alice's Verify Key
Bob_PK: Bob's Public Key
Bob_VK: Bob's Verify Key