Design Question:

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

We will be using Structs as our main data structures to organize data on Datastore. Our structs will be encrypted to hide the contents within the struct and then MAC'd for integrity. We will have numerous different type of structs such as User Struct which will hold information about the user, FileNode Struct nodes will be implemented in a linked list data structure that will hold the contents of the file, and many other structs such as Invitation, KeyFile, Intermediate, and FileLocator structs that will help play a role in our implementation of file sharing and revocation.

Helper functions: As you come up with a design, think about any helper functions you might write in addition to the cryptographic functions included here.

We've decided to write numerous helper functions to help encrypt and decrypt structs stored within our Datastore. Our helper functions include but are not limited to: HybridEncryption/ HybridDecryption which performs hybrid encryption to store our invitational structs. EncFileNode/ VerifyThenDecFileNode which performs symmetric encryption for our file nodes and also checks for integrity when decrypting through the use of hmacs. Moreover we created the helper function GenerateKeys to help create the keys used for the encryption process as well as for hmacs.

User Authentication: How will you authenticate users?

We store each user Struct in datastore with a unique UUID associated with the hashed username of the user. When authenticating users, we first check whether the UUID exists in the datastore by creating the UUID from the provided username. Since each username is unique, we will know if the user exists or not based on if the UUID exists in the datastore or not. Moreover, to verify that the user inputted the correct password, the correct symmetric key and mac key can only be recreated if the username and password match. Therefore the only way to decrypt the User Struct and get its data at the UUID is to have the correct credentials.

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

We implement a cloud-based data store that holds the latest state of user data. Whenever there is a change to a user object, we download the user object from the datastore, decrypt it and modify it to match the desired changes. Then we encrypt the updated user data and store it in a datastore in a struct. This way when a user logs in a different device, they can always access the updated User object by downloading it from datastore.

File Storage and Retrieval: How does a user store and retrieve files?

Users will store their files a FileNode structs which will be implemented in a linked lists formation. Each FileNode will have bytes decided for the contents of the file as well as the UUID of the next FileNode in the linked list and the UUID of the previous FileNode in the linked list. There will also be a FileLocator struct which will contain the UUID of the first and last FileNode in the linked list as well as the keys to decrypt and verify FileNodes. When a user first stores a new file, the first FileNode will be populated with the contents while the last FileNode will be a FileNode with no contents at all. To retrieve files, we locate the FileLocator associated with the file to locate the first FileNode. Then iterate through

until the end of the linked list, appending each contents of the linked list together along the way. The final content will be the contents of the file which will then be returned for the user.

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

The total bandwidth used in call of append will equal to the size of the data being appended to the file plus a constant (the size of an empty FileNode struct and the FileLocator struct). Because the FileNodeLoator has the location of the last FileNode in the linked list,, we will not have to download the whole file when appending. Instead, we will modify just the last FileNode and populate it with the contents being appended. Then we will need to add an extra empty FileNode struct at the end of the linked list since the previous empty linked list was just modified with the appended contents.

File Sharing: What gets created on CreateInvitation?

If the sender is the owner of the file, an intermediate struct will be created alongside with an invitation struct. The intermediate struct contains the UUID of the FileLocator struct as well as the keys to decrypt it. The invitation struct will contain the UUID of the intermediate struct as well as the keys to decrypt that. If the sender is not the owner of the file, only an invitation struct is created which instead holds the UUID of the intermediate struct of the sender. CreateInvitation will then return the UUID of the invitation struct in both cases.

…and what changes on AcceptInvitation?

The user will receive the UUID of the invitation struct as well the ability to decrypt it and access the information in it. Then a KeyFile struct will be created. This KeyFile struct is a struct that only the user can access which will hold the UUID of the intermediate struct and the keys to decrypt it. Since the intermediate struct points to the FileLocator struct, the user will now be able to access the contents of the file.

File Revocation: What values need to be updated when revoking?

The contents of our FileNodes will be downloaded and stored all in one new FileNode. Then our old FileLocator will be deleted and replaced with a new FileLocator with a new UUID which will have updated key values and point to the new FileNode as the firstFileNode and an empty fileNode as the lastFileNode. Since the locator of our FileLocator has changed and the keys have been updated, we will then update all of the intermediate nodes that point to it with the correct keys and UUID.. The revoked user’s intermediate Sturct will not be updated, but deleted instead.

Information about Structs in DataStore:

Struct	Enc/Mac	UUID	Contents	Description
User	Symmetric Key/ HMAC	uuid.FromBytes(Hash(Username)[:16])	Username, password, Private key, Signature Key, and an IntermediateUUID map	Contains information of User. “IntermediateUUID” is a map of a map which holds information of the invitation struct for direct invited user.
FileNode	Symmetric Key/ HMAC	uuid.New()	File contents, PrevUUID, NextUUID	FileNodes are stored in a link list data structure. The last FileNode will always be empty for efficient bandwidth append.
FileLocator	Symmetric Key/ HMAC	uuid.New()	FirstFileNodeUUID, LastFileNodeUUID, SymKeyFN, MacKeyFN	Has UUID of first and last FileNodes in the linked list as well as the keys to decrypt it.
Intermediate	Symmetric Key/ HMAC	uuid.New()	FileLocatorUUID, SymKeyFileLocator, MacKeyFileLocator	Only direct recipients of file will have an intermediate with UUID of the FileNodeLocator and the keys to decrypt it. Used for file sharing and revocation.
Invitation	HybridEncryption/HMAC	uuid.New()	IntermediateUUID, SymKeyInter, MacKeyInter	Created when inviting user. Contains the UUID of intermediate struct and keys to decrypt it.
KeyFile	Symmetric Key/ HMAC	uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "file" + filename))[:16])	IsFileOwner bool FileUUID SymKeyFile MacKeyFile	Each User with access to file will have KeyFile that has the information to decrypt Intermediate struct to access file contents.