Notes:

Data Structures:

User struct:

- RKey - Deterministically derived key via argon2key()
- PrivateKey – Generated together with the public key through PKEKeyGen()
- SignKey – Generated together with public key through DSKeyGen()

File struct:

- FragmentsUUID
- FragSKey – sym key for fragments encryption
- FragHMACK – HMAC Key for fragments verification
- Children – a list of Invitation structs to track the people who the file was shared with directly

Fragments struct:

- These are the structs that will hold the information for encryption and decryption of the actual data
- It will hold OwnerUN as owner's username
- FName – file name
- HMACK – HMac key for making sure that the fragments has not been modified
- Cnt – counter that tracks how many fragments the file is broken into / number of appends
- EKey – Enc Key for the data

SharedFile struct:

- Fragments struct UUID
- UName – user name of the person the file was shared with
- IsOwner – indicates whether this shared file belongs to owner

- FragSKey – Sym key for Fragments struct
- FragHMACK – HMAC key for Fragments struct
- FileUUID – UUID of the file struct, only present for file owner
- FKey – File enc key, only present for file owner
- FHMACK – File HMAC key, only present for file owner

Invitation struct:
- SFUUID – SharedFile UUID
- SFK – Symmetric encryption key for SharedFile
- SFHMACK - HMAC key of SharedFile

User Authentication:
- When Alice creates a new user it is stored in the data store as key/value pair
- We can store each username in the datastore with a key that will be the uuid – hash(username + "_usr"), and its value wil be an encrypted User struct with a 64 byte HMAC for verification of authenticity
- The user root key RKey will be deterministically derived for each access whether Alice is creating a new user or logging in as an existing user. It will be derived through argon2Key(password, username, 64).
- This RKey can be used further on to deterministically generate HMAC and encryption keys using faster HashKDF.
- These structs can be converted to bytes using JSON helpers.
- To create a new user use InitUser
- To retrieve an existing user session use GetUser
    - If the username password pair is wrong the wrong key will be derived and the function will throw error

File Storage / Retrieval:
- The information needed to store and retrieve a file will be kept in the Fragments structure.'' Each File will have its own Fragments structure.
- When a user tries to access the file by either storing something into it or loading it, the system will check in the data store at key hash(username + filename + "F") which the user can decrypt with his/her private key. The value at these keys will be the Invitation struct which will have the necessary keys to get to the SharedFile struct which will contain the encryption key needed for decryption.

- **Upon a call to StoreFile**, we first check to see if the user has an existing invitation to the file. This would also indicate whether the file exist in user's file space
    - If yes, this means we are overwriting the file. So, we zero out Cnt – the counter for the fragments and call AppendToFile. This would overwrite the contents of the file to the first fragment.
    - If not, we initialize the file through initFile helper method and call AppendToFile
- **Upon a call to AppendToFile** we first check to see if the user has an invitation for this file. If not, we return an error
- Then we traverse Invitation -> Shared File -> Fragments
- Then we generate the id for the Fragment we are about to save through uuid.FromBytes(hash(username + filename + cnt + "frag"))
- Then we encrypt and HMAC the data, increase the counter and save fragments in datastore.
- **Upon a call to LoadFile** we first check if the user has an invitation for the file ensuring the user has the file in their filespace
- Then we traverse to fragments as mentioned above and call the helper glueFragments which zeroes out the counter retrieves all fragments in order and glues them together

File Sharing and Revocation:

- The access to the File struct is exclusive to the file's owner. The other users whom the file was shared with by the owner will only get access to their own versions of the SharedFile. Each person that gets access through a second degree sharing will be able to access the same SharedFile instance as the level 1 root of the subtree. Once the file access is revoked that SharedFile instance is destroyed, and the Fragments are re encrypted and relocated.
- **Upon a call to Create Invitation**, there are two cases:
    - If the person sharing the file is the file owner, we get owner's file struct add the recipient to the children list and create a new Shared File instance for this recipient. Now, the invitation created will contain the keys to this new Shared File.
    - Else, the new invitation created will contain the same keys as the level one descendant/ root user of this SharedFile branch.

    - Minor detail: the returned invitation from Create Invitation function will not be the final invitation stored by the recipient as it will have a different file name and is made to be deleted from datastore upon a call to Accept Invitation
    - The invitations that are created are encrypted with the recipients public encryption key which is stored in KeyStore, and signed with the sender's private signing key
- **Upon a call to Accept Invitation**, the signature on the invitation is verified using the sender's public verification key, then decrypted using the recipient's private key.
- After it is decrypted it is stored with the new filename at username+filename+"F" hashed, encrypted with user's private key and signed
- **Upon a call to RevokeAccess**, the invitation is fetched from DataStore ensuring that the user has the file in their file space.
- If the user is not the owner of the file the method returns with an error as only the file owner has the right to revoke access to a file
- After that, the fragments for this file are encrypted with new keys and relocated to a different random UUID.
- Finally, the SharedFile structs of all the users except the revoked access user are updated to reflect the new keys for fragments. The revoked access user's shared file on the other hand is deleted from datastore