CS161 Project 2 Final Design

Structs

- `User`
    - `Username string //user's username`
    - `Password_key []byte //key generated from PBKDF`
    - `Priv_rsa PKEDecKey //private key for rsa decryption`
    - `Priv_sig DSSignKey //private key for signatures`
- `FileInfo`
    - `Content_head UUID //where first content block is located`
    - `Content_tail UUID //where last block is located`
    - `File_key []byte //how to decrypt the file`
- `FileContentBlock`
    - `UUIDafter     userlib.UUID //uuid of the block after`
    - `BlockContents []byte      //contents`
    -
- `Shared`
    - `FileInfoUUID userlib.UUID //uuid of fileinfo`
    - `FileInfoKey  []byte //key to decrypt fileinfo`
- `Invitation`
    - `SharedUUID userlib.UUID //uuid of shared`
    - `SharedKey  []byte //key to decrypt shared`
    - `Owner      bool`
    - `Sharedlist []string //List of users with access`

Helper Methods

- EncThenMac
    - `func EncThenMac(thingToEncrypt []byte, secretkey []byte, purposeEnc string, purposeMAC string, iv []byte) (macResult []byte, err error)`
    - Returns the HMAC appended to the symmetrically encrypted ciphertext and an error.
    - How it works: Compute two HashKDFs on the secret key for the encryption and HMAC keys using the purpose strings. Symmetrically encrypt the plaintext and compute an HMAC over the ciphertext. Append the HMAC and the encrypted byte array and return this result with an error.
- DemacThenDecrypt

- `func DemacThenDecrypt(thingToDecrypt []byte, secretkey []byte, purposeDec string, purposeDeMAC string) (decrypted []byte, err error)`
    - Returns the decrypted plaintext and an error.
    - How it works: Recompute the HashKDF keys used for encryption with the purpose strings and the secret key. Recompute an HMAC on the first 64 bytes of the ciphertext (because the ciphertext is passed in as HMAC||cipertext). If the recomputed HMAC is equal to the first 64 bytes of the ciphertext, symmetrically decrypt the last 64 bytes of the ciphertext and return the decrypted plaintext and an error.
- RetrieveFileInfo
    - `func (userdata *User) RetrieveFileInfo(filename string) (info *FileInfo, shared *Shared, owner *Invitation, err error)`
    - Returns pointers to the FileInfo, Shared, and Invitation structs associated with the user's file and an error.
    - How it works: Recompute the user's deterministic invitationUUID from the first 8 bytes of the hashed username appended to the first 8 bytes of the hashed filename. Retrieve the Invitation struct from Datastore and then DemacThenDecrypt and unmarshal it. From the unmarshaled Invitation, retrieve the Shared struct with the Invitation's SharedUUID field. After we DemacThenDecrypt and unmarshal it, we retrieve the FileInfo struct with the Shared struct's FileInfoUUID field and then DemacThenDecrypt and unmarshal it. We then return pointers to each of these structs.

User Authentication

- Creating a user (InitUser)
    - Call GetUser and throw an error if the user already exists.
    - Generate a deterministic UUID from the first 16 bytes of the hashed username.
    - Generate a 16 byte secret key with PBKDF on the first 8 bytes of the hashed username concatenated with the first 8 bytes of the hashed password and a randomly generated salt.
    - Generate public and private key pairs for RSA encryption and digital signatures. Store the public keys in Keystore and the private keys in the User struct.
    - Marshal and EncThenMac the user struct and store it in Datastore with the deterministic username UUID.

    - Compute a deterministic UUID for the salt computed for the PBKDF and store the salt in Datastore.
- Authenticating a user (GetUser)
    - Recompute the deterministic username UUID and get the marshaled user from Datastore. Also check Keystore for their RSA public key to make sure they're a real user.
    - Recompute the string used for PBKDF with the user provided username and password and get the salt from Datastore.
    - DecThenDemac the marshaled user with the PBKDF key, which should verify the HMAC and decrypt the user.
    - Unmarshal the user and return the pointer to the user.

File Storage and Retrieval

- Storing files:
    - Each time a file is stored, we create its corresponding FileInfo, Invitation, Shared, and FileContentBlock structs. The FileContentBlock contains the contents and the UUID of the last block of the file. The FileInfo struct contains the key to decrypt the FileContentBlocks associated with the file and the UUIDs of the first (the block we just made) and last content blocks. The Shared struct contains the UUID and encryption key for the FileInfo struct. The Invitation struct contains the UUID of and key to decrypt the Shared struct, along with a boolean field to indicate if the user is the owner of the file, so it is true each time StoreFile is called.
- Loading files:
    - RetrieveFileInfo on the filename to get the 3 corresponding structs.
    - Follow pointers from the Content_head of the FileInfo to the Content_tail. For each pointer, we search Datastore for that uuid and then DemacAndDecrypt and unmarshal it, and append the contents to a byte array to be returned and then set the next pointer to the UUIDafter field of the decrypted file block.
- Appending to files:
    - RetrieveFileInfo on the filename
    - Create a new FileContentBlock struct
        - Set ContentBlock.BlockContents to the contents
        - Save the current FileInfoStruct.Content_tail UUID
        - Compute a new UUID for FileInfoStruct.Content_tail UUID
        - Set BlockContents.UUIDafter to the new UUID

    - EncThenMac the FileInfo and FileContentBlock, then restore the FileInfo to its previous location and store FileContentBlock at the previously saved UUID.

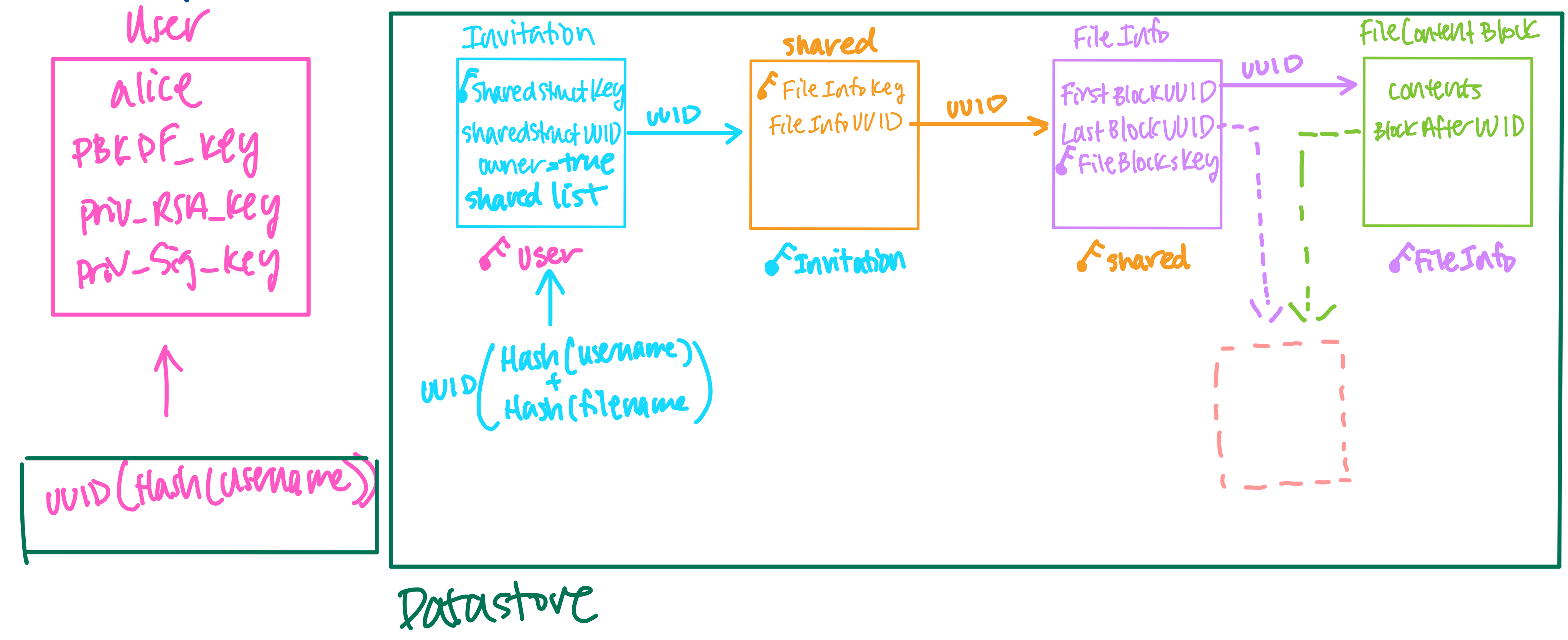
File Sharing and Revocation

How will a user share files with another user? How does this shared user access the shared file after accepting the invitation? How will a user revoke a different user's access to a file? How will you ensure a revoked user cannot take any malicious actions on a file?

- Creating an invitation:
    - RetrieveFileInfo on the file.
    - Append the recipient's username to the inviting user's Invitation's shared list, which contains the users that have access to the file. Create a new Invitation struct and set the Owner field to false.
    - If the inviting user is the owner:
        - Create a new Shared struct and set its fields to the user's Share struct returned by RetrieveFileInfo.
        - Compute a deterministic UUID from the hashed username and file name and then marshal and EncThenMac the new Share struct. Store it in Datastore at the computed UUID.
        - Set the new Invitation.SharedUUID to the new Share's computed UUID and the Invitation.SharedKey to a new randomly generated key.
    - If the inviting user is not the owner of the file:
        - Directly set the new Invitation.SharedUUID and Invitation.SharedKey to the same fields found in the decrypted Invitation struct returned by RetrieveFileInfo.
    - Encrypt the invitation with the recipient's public key and then sign the invitation with the sender's private signature key.
    - Create a new UUID for this invitation and store it in Datastore along with signature||encryption of the invitation.
- Accepting an invitation:
    - Retrieve the Invitation and decrypt it and verify the signature. Create a new UUID for the Invitation based on the user provided filename, and then use EncThenMac on the decrypted invitation. Store the re-encrypted Invitation at the new UUID. This will now serve as this user's Invitation struct for the file.

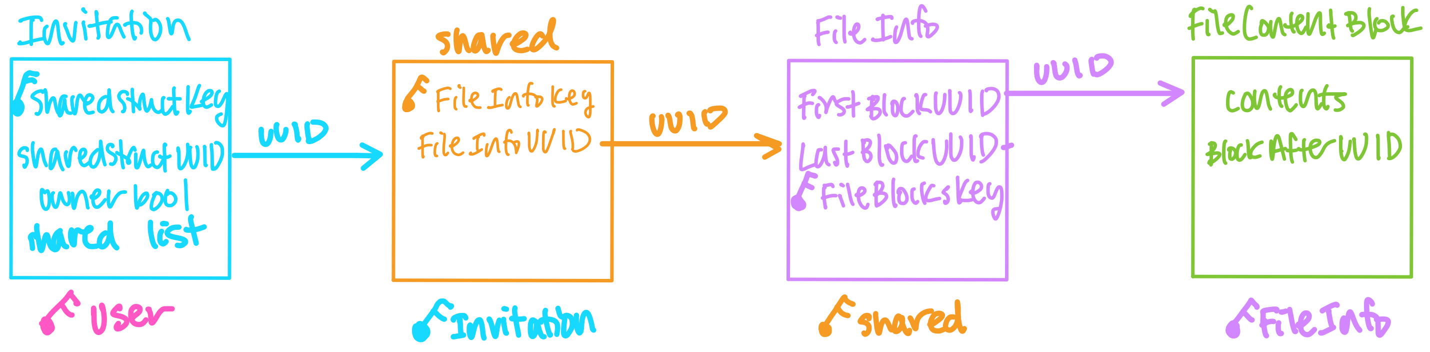
- Revoking Access
    - Retrieve the 3 structs from RetrieveFileInfo.
    - Remove the recipient’s username from the Invitation’s shared list.
    - Change the UUIDs of each FileContentBlock associated with the file by following pointers from FileInfo.
    - Change the UUID of the FileInfo struct and compute a new key
    - Go through everyone who still has access in the shared list, get their shared structs, and update the FileInfoKey and FileInfoUUID to the new ones computed above. Restore these in Datastore.
    - Store the updated FileInfo struct at the new UUID.
    - With this approach, the revoked user no longer has any information about the file.

① Creating a user   ② Storing a file

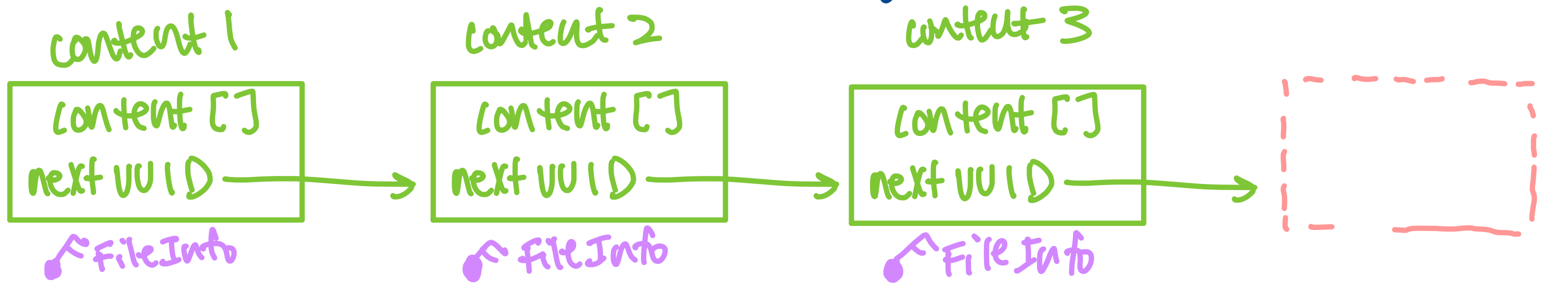


③ Loading a file

→ retrieve first content block after computing uuid(Hash(username) + Hash(filename))



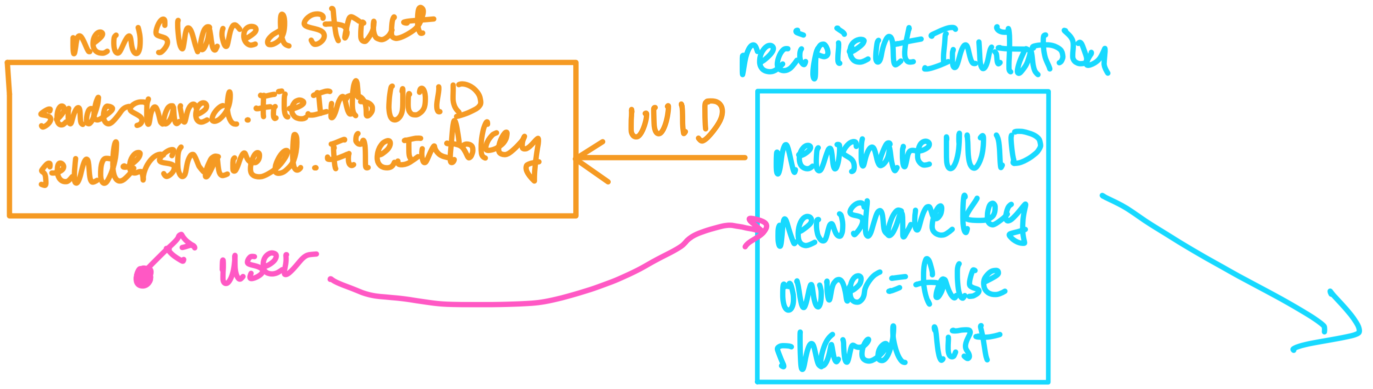
→ Follow pointer to load each file content block



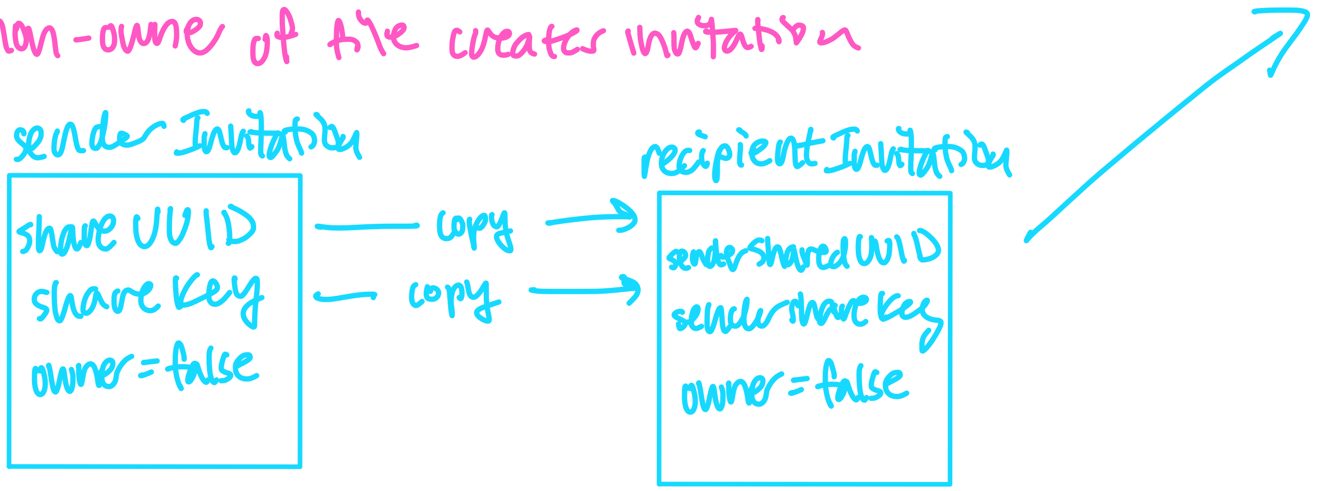
④ Creating Invitations

→ Retrieve corresponding senderInvitation, senderShared, FileInfo structs.

① owner of file creates invitation



② non-owner of file creates invitation



PKE Enc w/ recipient's public key

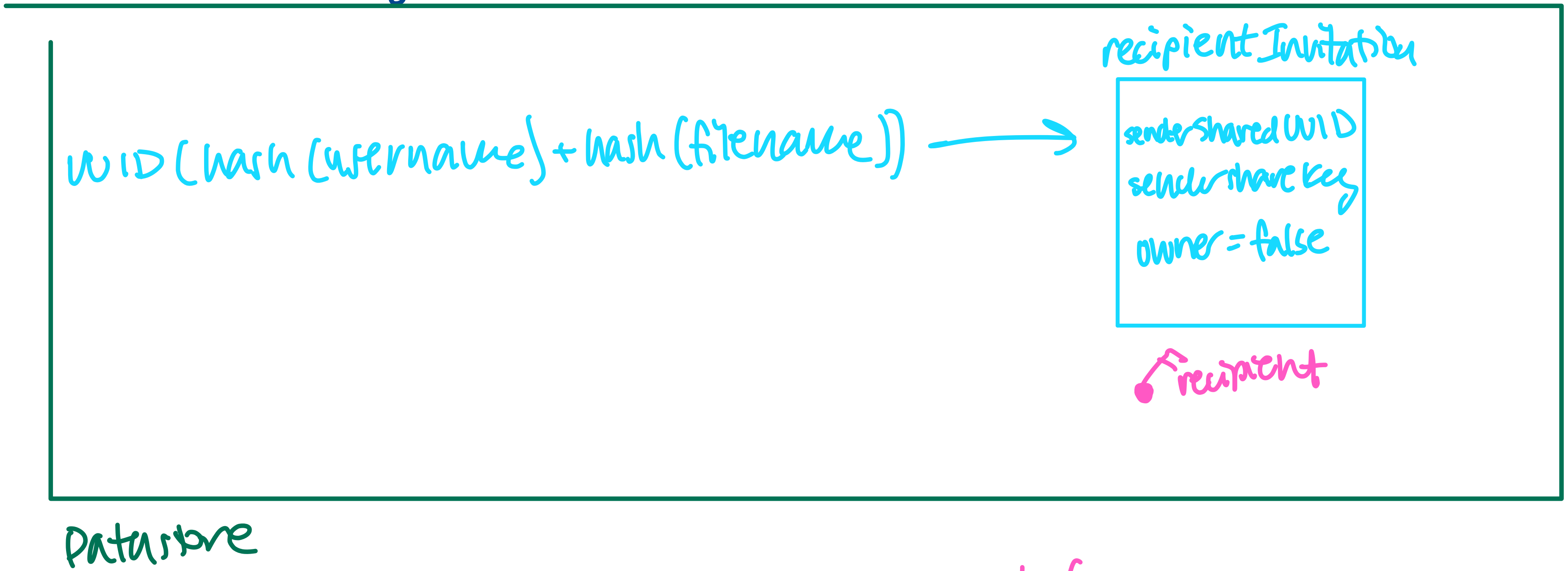
sign inv w/ sender's priv. signature key

④ Accepting Invitations

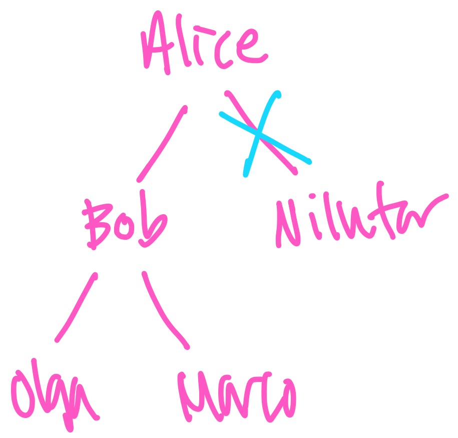
→ verify signature and decrypt invite

→ compute new UUID: UUID(hash(username)+hash(filename))

→ re-encrypt symmetrically, store at new location



⑤ Revoke Access



Alice revokes Nilufar

→ compute new UUIDs for fileblocks

→ Update Alice's File Info, generate new key

→ update Bob, Olga, Marco's structs w/ Alice's new FileInfo UUID, FileInfo key

→ store Alice FileInfo @ new FileInfo UUID