Helper Functions

1. FetchLockboxPtr(username, filename, privateKey, vk userlib.DSVerifyKey) -> Lockbox UUID and root key
2. FetchLockbox(Lockbox_UUID userlib.UUID, Lockbox_rootKey []byte) -> fields in the lockbox

3. SymEncMac(root_key, contentSerial) -> encrypted content serial with appended HMAC
4. SymDecMac(root_key, encryptedContent) -> decrypted content serial
    a. Sym functions: Generate sym_key and mac_key from HashKDF(root_key,), use HMACEqual and HMACEval(mac_key,) to verify integrity, Encrypt with SymEnc(sym_key, RandomBytes(16), contentSerial), decrypt with SymDec(sym_key, encContentSerial).

5. AsymEncSig(ek userlib.PKEEncKey, sk userlib.DSSignKey, plaintext []byte) -> ciphertext []byte
6. AsymDecVerify(dk userlib.PKEDecKey, vk userlib.DSVerifyKey, encContent []byte) -> plaintext []byte
    a. Asym functions: Ensure integrity with Sign with DSSign(sk, plaintext) and verify with DSVerify(vk,ciphertext, sig), Encrypt with userlib.PKEEnc(ek, plaintext), decrypt with userlib.PKEDec(dk, ciphertext)

User Initialization and Authentication

User Struct Fields:
(1) Username (string), (2) PrivateKey (userlib.PKEDecKey), (3) SignKey (userlib.DSSignKey)

Struct LockboxPtr: LB_rootKey, LB_UUID (used to decrypt a lockbox)

Struct Master Lockbox: File_rootkey []byte, EncFileRoot []byte, FileHeader_UUID, SharedUserLBs_UUID

Struct Lockbox: File_rootkey []byte, EncFileRoot []byte, FileHeader_UUID

InitUser

Function header: InitUser(username string, password string) (userdataptr *User, err error)
Errors: “username” exists in the keystore, “username” is empty

1. Create User Struct Object
2. For Asymmetric enc/dec (RSA), create public/private key with PKEKeyGen() and sign/verify key with DSKeyGen()
    a. Assign username: public_key into keystore, Assign user.Privatekey := private_key
    b. Assign username + ”_verify”: verify_key into keystore, Assign user.SignKey := sign_key
3. Generate random salt using RandomBytes(16), store uuid(hash(username + “_salt”)[0:16]): salt in Datastore
4. Create a root_key with Argon2Key(password, salt, 16), then serialize using json.Marshal(user struct)
5. Encrypt and add HMAC to the user object serial using SymEncMac(root_key,). Returns HMAC || encrypted user struct

6. Store (uuid(hash(username)[0:16]) : HMAC || encrypted user struct) into Datastore

GetUser

Function header: GetUser(username string, password string) (userdataptr *User, err error)
Errors: No initialized user for the given username (username doesn't exist in keystore), SymDecMac errors if HMAC of encrypted user struct is invalid

1. Check if user exists, then fetch user struct from Datastore
2. 2. Fetch salt from uuid(hash(username + "_salt")[0:16]): salt in Datastore
    a. Regenerate root_key = Argon2Key(password, salt, 16) to regenerate mac_key and sym_key.
3. Check integrity of encrypted user struct using helper function SymDecMac, returns decrypted serial
4. Deserialize (with json.Unmarshal) and return the encrypted user struct

File Operations

User.StoreFile

Function header: User.StoreFile(filename string, content []byte) (err error)

1. Randomly generate file_rootKey = RandomBytes(16). Generate sym_key and mac_key with HashKDF(root_key,)
2. Create filename_UUID from Hash(filename + "_" + userdata.Username).
3. Generate new file_rootKey, fileHeader_UUID, encFileRoot, sharedUserLBs_UUID, LB_UUID, and LB_rootKey using uuid.New() and userlib.RandomBytes(16)
4. Create LockboxPtr and MasterLockbox objects using previous step variables
5. Serialize and encrypt both numSegments = 1, content (into encFileSegmentContent), and sharedUserLBs = make(map[string][]byte) using SymEncMac(file_rootKey,). This dict will map usernames to unencrypted lockboxPtr structs, and will give the owner of the file the ability to overwrite other shared users' lockboxes for this file.
6. Serialize and encrypt MasterLockbox with SymEncMac(LB_rootKey, masterLBSerial)
7. Serialize and encrypt LockboxPtr using AsymEncSig(user's public key, user's signkey, LB_ptrSerial)
8. Generate encFile_UUID using uuid.FromBytes(HashKDF(encFileRoot, intByte=json.Marshal(0)))
9. Store encFileHeader at fileHeader_UUID, encFileSegmentContent at encFile_UUID, encMasterLB at LB_UUID, encLB_ptr at filename_UUID, and encSharedUserLBs at sharedUserLBs_UUID

User.AppendToFile

Function header: User.AppendToFile(filename string, content []byte) (err error)

1. Fetch LockboxPtr using FetchLockboxPtr(), then Lockbox from FetchLockbox(LB_UUID, LB_rootKey)
2. Add 1 to # of file segments at fileHeader_UUID serializing/deserializing and encrypting/decrypting with SymDecMac(file_rootKey, encFileHeaderContents), SymEncMac(file_rootKey, numSegmentsSerial)
3. Generate appendUUID to store content using uuid.FromBytes(HashKDF(encFileRoot, intByte = json.Marshal(numSegments - 1))). Each AppendToFile call, the new content will be at a different appendUUID.
4. Store & encrypt content serial using SymEncMac(file_rootKey, content) at appendUUID. Store encFileHeader at fileHeader_UUID

User.LoadFile

Function header: User.LoadFile(filename string) (content []byte, err error)

1. Fetch LockboxPtr using FetchLockboxPtr(), then Lockbox from FetchLockbox(LB_UUID, LB_rootKey)
2. Fetch and SymDecMac fileHeaderContents from fileHeader_UUID
3. Iterate though file segment UUIDs at uuid.FromBytes(HashKDF(encFileRoot, intByte)), where intByte goes from 0 to numSegments - 1. Deserialize and SymDecMac(file_rootKey,) the content at each UUID, return []byte of all content.

Sharing and Revocation

CreateInvitation

Function header: CreateInvitation(filename string, recipientUsername string) (invitationPtr UUID, err error)

1. Fetch LockboxPtr using FetchLockboxPtr(), then Lockbox from FetchLockbox(LB_UUID, LB_rootKey)
    a. If sender is the owner (if lockbox is a Master lockbox)
        i. Create a new shared lockbox for the recipient with the same file_rootKey, encfile_UUID, and fileHeader_UUID
        ii. Generate new lockbox root key for the recipient, then put lockbox root key and lockbox UUID in an lockboxPtr struct.
        iii. Asymmetrically encrypt LBPtr struct with recipient's public key, and digitally sign with sender's private key (using AsymEncSig). Generate a UUID for the invitation (= invitationPtr) and store it in the datastore at invitationPtr.
        iv. Put LBPtr struct into MasterLockbox dictionary sharedUserLBs at key recipientUsername. Get dictionary by marshaling and SymDecMac(file_rootKey, serial)/SymEncMac(file_rootKey, serial). Store at sharedUserLBs_UUID
    b. Else, if sender isn't owner
        i. Put sender's lockbox rootKey & UUID in a new lockboxPtr struct, generate a UUID for the LBPtr struct
        ii. Asymmetrically encrypt LBPtr struct with recipient's public key, and digitally sign with sender's private key (using AsymEncSig), put at a generated UUID invitationPtr.
2. Return invitationPtr

AcceptInvitation

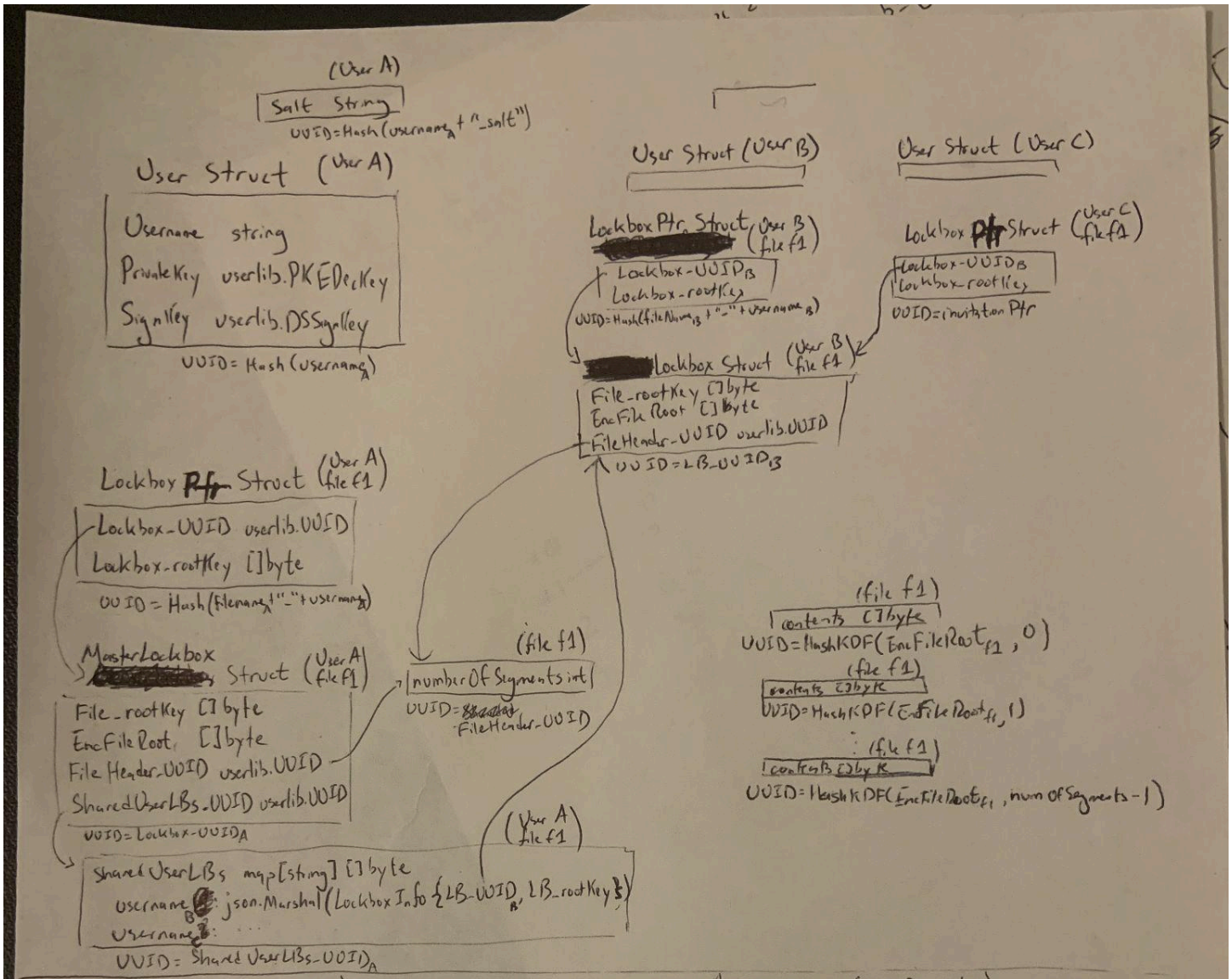
Function header: AcceptInvitation(senderUsername string, invitationPtr UUID, filename string) (err error)
Errors: filename in use: uuid.FromBytes(userlib.Hash([]byte(filename + "_" + userdata.Username))[:16]) in datastore,
invitationPtr doesn't exist / is garbage, signature of LockboxPtr object is invalid

1. AsymDecVerify and Unmarshal content at invitationPtr to get LockboxPtr obj
2. Check if invitation revoked by decrypting (with SymDecMac) lockbox associated with LockboxPtr obj
3. AsymEncSig LBPtr struct and store to Datastore at uuid.FromBytes(userlib.Hash([]byte(filename + "_" + userdata.Username))[:16]), encrypt with user's privateKey
4. Destroy invite by overwriting information at invitationPtr with garbage

RevokeAccess

Function header: RevokeAccess(filename string, recipientUsername string) (err error)

1. Generate a new file_rootKey, encFileRoot, fileHeader_UUID, and sharedUserLBs_UUID for the file
2. For each user that still has access to the file: only update lockboxes of non-revoked users
    a. Get their LB_UUID and LB_rootKey in master lockbox's sharedUserLBs
    b. Use LB_rootKey to decrypt the lockbox at LB_UUID
    c. Replace old contents with new contents (from step 1), overwrite old LB with garbage
3. Get numSegments from fileHeader, iterate though file segment UUIDs at uuid.FromBytes(HashKDF(oldEncFileRoot, intByte)), where intByte goes from 0 to numSegments - 1.
4. Deserialize and SymDecMac(oldFile_rootKey, encContent). SymEncMac(file_rootKey, content).
5. Re-serialize at uuid.FromBytes(HashKDF(encFileRoot, intByte)).
    a. Overwrite both numSegments at fileHeader_UUID and file contents at old UUIDs with garbage


(User A)
Salt String
UUID = Hash(username_A + "_salt")
User Struct (User A)
Username string
PrivateKey userlib.PKEDecKey
SignKey userlib.DSSignKey
UUID = Hash(username_A)
User Struct (User B)
Lockbox Ptr Struct (User B file f1)
Lockbox-UUID_B
Lockbox-rootKey
UUID = Hash(fileName_B + "_" + username_B)
User Struct (User C)
Lockbox Ptr Struct (User C file f1)
Lockbox-UUID_B
Lockbox-rootKey
UUID = invitation Ptr
Lockbox Struct (User B file f1)
File-rootKey []byte
EncFileRoot []byte
FileHeader-UUID userlib.UUID
UUID = LB-UUID_B
Lockbox Ptr Struct (User A file f1)
Lockbox-UUID userlib.UUID
Lockbox-rootKey []byte
UUID = Hash(filename_A + "_" + username_A)
MasterLockbox Struct (User A file f1)
File-rootKey []byte
EncFileRoot []byte
FileHeader-UUID userlib.UUID
SharedUserLBs-UUID userlib.UUID
UUID = Lockbox-UUID_A
(file f1)
numberOfSegments int
UUID = FileHeader-UUID
(file f1)
contents []byte
UUID = HashKDF(EncFileRoot_f1, 0)
(file f1)
contents []byte
UUID = HashKDF(EncFileRoot_f1, 1)
(file f1)
contents []byte
UUID = HashKDF(EncFileRoot_f1, numOfSegments - 1)
(User A file f1)
SharedUserLBs map[string] []byte
username_B: json.Marshal(LockboxInfo{LB-UUID_B, LB-rootKey_B})
username_C: ...
UUID = SharedUserLBs-UUID_A
Datastore (UUID: value)
hash(username): HMAC || SymEnc(marshal(user struct))
hash(username + "_salt"): salt
hash(fileName + "_" + username): Signature || AsymEnc(marshal(Lockbox Ptr struct))
HashKDF(EncFileRoot, int Byte): HMAC || SymEnc(marshal(contents (of file)))
Lockbox-UUID: HMAC || SymEnc(marshal(lockbox struct))
FileHeader-UUID: HMAC || SymEnc(marshal(numOfSegments (int)))
SharedUserLBs-UUID: HMAC || SymEnc(marshal(sharedUserLBs (dict)))
invitation Ptr: Signature || AsymEnc(marshal(Lockbox Ptr struct))
Keystore (UUID: Value)
username: PublicKey (userlib.PKEEncKey)
username + "_verify": VerifyKey (userlib.DSVerifyKey)