Jonathan Saleh Eric Ma

Variables and Data

Struct User

- Username — string
- Password — string
- Private RSA key for Invitations. — userlib.PrivateKeyType
- Private key for Signatures — userlib.DSSignKey
- FileeMap UUID — uuid.UUID
- FileeMap Struct — FileMap

Struct File

- File_id — uuid.UUID
- Pointer to Content — uuid.UUID
- Next UUID — uuid.UUID
- Last UUID — uuid.UUID

Struct Invitation

- File id — uuid.UUID
- File salt — []byte

Struct FileeMap

- Map (HashedFilename, UUID) — map[string]uuid.UUID
- Map (HashedFilename, Salt) — map[string][]byte
- Map (HashedFilename, Shared Users) — map[string][]string
- Map (HashedFilename, Map(Username, UUID)) — map[string]map[string][]byte
- Map (HashedFilename, RecievedInvites) — map[string][]byte
- Map (HashedFilename, RecievedInvitesKey) — map[string][]byte

Init User(username, password) (CTR + HMAC)

- **Check** if username is **emptystring**
- **check if usernam**e is available. (Helper Function CheckExist(username))
- Generating **RSA key pai**r and **DS Keypair**, and upload **public key** to keystore
- Generating **sourceKey** using **Arg2Key, []byte(pw)** as pw and username as salt
- **HashKDF** SourceKey to get **fileMapMacKey, FileMapEncKey, macKey** and **EncKey,**
- Make an empty **FileeMap** struct and fill out necessary variables, **json.Marshal** the empty FileeMap struct, generate **uuid.New()** for filemap struct, set **userdata.Files** to the UUID for filemap
- Copy userdata to user struct, and User **json.Marshal** userdata
- Use **SymEnc** on FileeMap struct, with **random IV**, FileMapEncKey, generate it's **HMAC** tag with FileMapMacKey
- **SymEnc** user struct, with **new random IV**, EncKey, generate it's **HMAC** tag with macKey
- **Concatenate** both of the encrypted content with corresponding macTag using append, (**MacTag||Encrypted Conten**t) (first 64 is macTag)
- Generate User Struct **UUID** using **hashed username**, and upload user struct and FileeMap struct to data store
- Return address of user struct locally

GetUser (CTR decrypt + HMAC integrity)

- Check if username is emptystring
- Use **username password** to generate the same **symmetric keys and mac keys**. Put in the HashKDF function to get integrity and encryption keys.

- Get **UUID** from username and do datastoreGet(hash(username))
- Eval same original macTag **HMACEval**(macKey[:16], EncContent[64:]), and check if same as the the macTag EncContent[:64]
- SymDec(**symmetric key,** EncContent[64:]) and unmarshal to get the stored user struct
- get filemap id from userdata and get the encrypted struct from datastore
- Using the same to get the keys, sse same decryption process on FileMap struct, save to userdata.FileMap
- Return address of user struct locally

StoreFile (CTR + hmac)

- Check if user is nil, and if not Do **CheckUpdateUser()**
- Check if filename exists in the user database by getting the hashed filename and check in FileeMap
- If filename exists, get **filestruct UUID** from **userdata,** generate the same **SourceKey** by **Argon2Key** using the **saved salt,** and **file uuid** as password**,** and **hashKDF** to get **EncKey, MacKey.**
- **DatastoreGet** the origina**l fileStruct**, check fo**r hmac,** and do **symDec,** and get the original file **content uuid**
- Overwrite the old content with new content using **StoreContentToDataStore()**
-
- If filename doesn't exist, Create new File Structs for the first file and the last file in the file linked list scheme. Generate unique **filestruct UUID and lastFile UUID** by Hash(filename + username), set **contentUUID = uuid.New()**. Generate SourceKey by using Argon2key, random bytes as **salt** for and Hash th**e file UUID as password.** Use this key in the **HashKDF** function to get the **EncKey, MacKey**
- **Use StoreContentToDatastore** to store content to **contentUUID with** EncKey and MacKey.
- **Store the fileStruct to Datastore using StoreFileToDatastore with EncKey and MacKey at filestruct UUID**
- **Store the last fileStruct to DataStore using same method**
- Add Information to FileMap struct in userdata struct
- Do UpdateUser**()**

LoadFile (CTR decrypt + hmac integrity)

- Do **CheckUpdateUser()**
- Hash the filename**,** check if this is the **origina**l file owner. If not then the user is using the original invitation as the source of information to access the files. Do updateFilePath(strHashedFilename) to check for updates on file information.
- Get the UUID of the file from the FileeMap in the user struct
- Generate the same SourceKey by **arg2key** using the salt saved in user struct and hash(filestruct uuid) as password. Get the Encryption and HMAC Keys
- Get the file struct using **CheckMac()** to Authenticate file integrity and **DecryptFile()** to decrypt into local file variables.
- Get the content of the first file struct using **CheckMac()** and **DecryptContent()** which is similar to the functions before, check if the next file is the last file in the linked list.
- If not, go to next file, and decrypt again until **next is lastFile**, combine all content in the file and return (HMAC and symDec for each next file)
- UserUpdate() to update the user struct

AppendFile (File Store)

- Get the **hashed filename** for the mapping key and get **UUID** of the file. Generate encryption and hmac **keys** with same method as LoadFile and StoreFile from salt stored in the FileeMap struct.
- Check if our information on the file is valid, if not, do **CheckUpdateUser()** and **updateFilePath()** if we are not the original user to update information on file. If we still are getting HMAC errors then we have been revoked.
- Create File struct variable**s file, tempfile, lastfile**, and a **new file** struct and random UUID.
- If there hasn't been an append yet, change the **file.next** to the new file UUID, restore the first file with **StoreFileToDatastore()**, update th**e lastFile.last** to the newFile UUID and also store that lastFile back to datastore.
- If there has been an append, Verify and Decrypt the **lastFile** and the **second to last file** to **tempFile** with the same helper functions on **file.last** and **lastFile.last**. Set **tempFile.next** to the **newFileUUID** and se**t lastFile.Last** to the **newFileUUID**.
- Then create and update the new File Struct. Using the same method from storeFile, store both the new file struct and file content to datastore. All of these with the same encKey and macKey
- **UserUpdate()**

CreateInvitation (Hybrid Encryption)

- Check if recipientUsername exist, and check if it's empty
- hash(filename), and check if filename exist in user's struct
- Check if user is the owner of file,
- If yes, generate 2 randomBytes for InvitiestructUUID and keyUUID
- Generate source key by Arg2Key using hash(userPassword || InvitiestructUUID) as password and username as salt, and HashKDF to the EncKey and MacKey for InviteStruct
- Put the shared file salt and shared file UUID in invite struct
- Get recipient public key from keyStore
- Put recipient name in user's[filename] shared userlist
- Encrypt and HMAC an invitation Struct and store it in datastore
- **Data** = Append(SourceKey for decrypting invite struct, UUID for invite struct)
- Use recipient's Public Key to RSA encrypt(**Data**) and digital sign on it
- Store Data on datastore
-
-
- If not the original owner of the file, user the recipient's public key to RSA encrypt the symmetric key and uuid of the invitation struct. (We are sending the data to access of the original invitation struct)
- Return uuid of **Data** and Update user at the end

AcceptInvitation (RSA)

- **RSA decrypts** the invitePTR with the user's private Key to get the symmetric key and struct UUID
- Check the digital signature on the invite struct.
- Use the symmetric key to decrypt the invitation struct stored in Datastore.
- Store file struct information in FileeMap struct
- UpdateUser()

RevokeAcess (RSA)

- Check if if filename and username is valid
- Remove the **username** from the list of file's shared user
- Copy content from the old file and creating new file, copy the content to new file. Generate a new random uuid and salt for the new file. Arg2Key to Hash the source key from the new file uuid, and random byte as salt. HashKDF the source key into macKey and encKey. Store the file to Datastore after symEnc and HMAC using macKey and encKey
- Update the information in each user's inviteStruct, except for the revoked user.
- UpdateUser()

HELPER FUNCTIONS

CheckExist(username)

- Call Datastore(username), if ok == True means user exist

UpdateFilePath(strHashedFilename)

- Generate the sourceKey using Arg2Key and HashKDF to macKeyInvite, encKeyInvite
- Get the uuid of the invite struct from datastore
- checkMac and symDec the invite struct
- Update the new file struct UUID and salt to user's data

CheckUpdateuser()

- Generate the sourceKey using Arg2Key and HashKDF to FilemapmacKey and filemapencKey, datastore get the latest userdata.File struct. Check hmac and symdec
- Copy file's data to userdata

updateUser()

- Generate the sourceKey using Arg2Key and HashKDF to FilemapmacKey and filemapencKey
- symEnc the latest userdata .file and hmac it. Upload to datastore

StoreContentToDataStore(UUID, data, encKey, macKey)

- Marshal data, generate randombyte as IV. symEnc data using encKey and IV. Generate Hmac tag by macKey and put tag on the encrypted data.

- Upload to datastore

DecryptContent(uuid, encKey, data)

- Datastore get the data from the uuid address.
- Using the given keys, symDec the data to &data
- Return data

CheckMac (UUID, mackKey)

- Datastore get the daata
- Using the given keys, Check HMAC is valid
- Return data

DecryptContent(uuid, encKey, file)

- Datastore get the encrypted file from the uuid address.
- Using the given keys, symDec the file to &file
- Return file

StoreFileToDataStore(UUID, data, encKey, macKey)

- Marshal data, generate random byte as IV. symEnc file using encKey and IV. Generate Hmac tag by macKey and put tag on the encrypted file.
- Upload to datastore