Mikal Viga, Hans Erik Hum - CS161

Project 2 - Design

Data structures

Our solution has 5 main structs, a User , OwnerMetaFile, MetaFile, MetaMetaFile, FileInfo, File, and an Invitation struct.
The purpose of the User struct is to store the Username, UUID, RSA Private key, and the hash(password+salt (UUID)). The reason we would like to store the username and the password hash is to verify the user when a new instance is created. We would also like to store the RSA private key because this is not deterministically created, but randomly.

The purpose of the File-struct is to keep the contents of the File. This struct has some contents and a UUID that potentially points to the next file element. Also, it contains an HMAC. We have solved the linear runtime append problem by storing Files as linked lists.

The purpose of the FileInfo struct is to give information about where the head and the tail of the linked list are. This struct uses the same HMACKey and EncryptionKey as the File.

The purpose of the OwnerMetaFile struct is for the owner to have control over the File and the people he has shared the file with. The OwnerMetaFile struct contains information about the MetaFiles he has shared the File with. This information is the UUID, the EncryptionKey the HMACKey of the MetaFiles. It also contains the EncryptionKey, HMACKey, and the FilePointer for the File so the owner also has access to the File.

The purpose of the MetaFile-struct is to give access to the File to the people who have got an invitation. Each MetaFile contains the UUID of the File which it points to, the encryption key for the File, and an HMAC-key which is used to guarantee integrity for the File. The MetaFile is encrypted randomly. An HMAC-tag on the content is also added, which is also created with a random key. The MetaFile's UUID is based on the EncryptionKey of the MetaFile.

The purpose of the MetaMetaFile-struct is to give access to the MetaFile. This struct is created by the Receiver of an invitation. MetaMetaFile contains the UUID of the MetaFile, the key to decrypt the MetaFile, and the MetaFile's HMAC-key. An HMAC-tag is also added to the MetaMetaFile, and the Key for this is deterministically created from the Users MasterKey and a purpose. The same is done for encrypting the MetaMetaFile.

The purpose of the Invitation-struct is to pass information about the MetaFile to the Receiver. The Invitation struct contains the UUID of a MetaFile and the key to decrypt the content inside a MetaFile.

Initialization

When a user is made with the InitUser function first of all one public and one private key are made. They should be completely random and are made with the PKEKeyGen() function. These keys should persist the entire time, and should therefore be stored. We also create a SignKey and VerifyKey from DSKeyGen(). These keys should persist the entire time, and should therefore be stored. By using a concatenation of the user's username and the hash of the password we make a MasterKey by using the Argon2Key() function. Then, the HMACKey is created with the HashKDF() function and the MasterKey.

Argon2Key() and HashKDF() are deterministic, and these keys can therefore be derived from a user's password and username. We then store the PrivateKey, FilesOwned, and SignKey in the datastore. We include an HMAC-tag. The UUID is made by hashing the person's username. The contents are encrypted with the MasterKey. An HMAC-tag is then made on the contents by using the HMAC-key we made in the beginning. The new user's public key is sent and stored in the KeyStore, with UUID.FromBytes(username+"PublicKey") as UUID. The same is done when storing the VerifyKey, but this time we use username+"VerifyKey".

Logging in

When a user logs in, the UUID is generated the same way as in Initzialation, and the user then tries to retrieve something from the DataStore at that UUID. If the information that is retrieved is garbage, the user fails to log in, if it "makes sense" the user is logged in, and the MasterKey and HMACKey are generated. The HMAC-tag is also checked.

Multiple client instances

With our current approach, several clients of the same user can be "logged in" simultaneously. Before each action, the instance has to update its local User struct. And after each action, the information in the DataStore is updated to match the local User struct.

File storage

StoreFile

When StoreFile is called the first thing is to check if the User already has a File with that filename. If that is the case, the user opens the OwnerMetaFile on an address that is created from the hash of the username, the hash of the password, and the hash of the filename. This is decrypted and HMAC checked with Keys that are created from the MasterKey and filename using HashKDF(). With access to the OwnerMetaFile, the User can open the FileInfo, because the OwnerMetaFile contains information and keys to open FileInfo for the given filename. FileInfo is then overwritten with a new FilePointerStart and FilePointerEnd which are randomly generated. The new contents from StoreFile are stored in a new File struct at the FilePointerStart. This File has a tail equal to the FilePointerEnd, where an empty File Struct is created.

If the User does not have the filename in his personal space. The User then creates a new UUID based on the hash of the username, the hash of the password, and the hash of the filename. It then tries to retrieve information at this UUID. If it finds nothing, it created an OwnerMetaFile. This is filled with a random EncryptionKey, random HMACKey, and a random pointer to the FileInfo. The OwnerMetaFile is encrypted and HMACed the same way as under decryption. After this, a FileInfo is created with the information in the OwnerMetaFile. It is filled with two random UUIDs as FilePointerStart and FilePointerEnd. The new contents from StoreFile are stored in a new File struct at the FilePointerStart. This File has a tail equal to the FilePointerEnd, where an empty File Struct is created. The Files are encrypted the same way as InfoFile.

If the user finds a MetaMetaFile at the created UUID in the last paragraph, the user then opens this MetaMetaFile with the same keys as it would open an OwnerMetaFile. A MetaMetaFile contains the information to open a MetaFile. The MetaFile gives information to open a FileInfo and then we do the same procedure as in the last two paragraphs.

LoadFile

When LoadFile is called upon, we must first check if the User is the owner of the File. This is done by checking the FilesOwned map in the User struct. If this is the case, the user then opens the OwnerMetaFile at UUID.FromBytes(Hash(Username)+Hash(Password)+Hash(Filename)), and decrypts and checks the HMAC with keys derived from the MasterKey and the filename. With access to OwnerMetaFile, its information and its keys, the User can open the FileInfo. Furthermore, the user can open the linked File list that starts at FileInfos FilePointerStart. The Files and FileInfo are decrypted and HMAC-checked with keys in OwnerMetaFile.
If the User is not the owner, it checks if something exists at UUID.FromBytes(Hash(Username)+ Hash(Password)+Hash(Filename)). If that is the case, it then opens the MetaMetaFile at this address with keys derived from the MasterKey and the filename. The MetaMetaFile contains the necessary information to open and decrypt the MetaFile for the File. This MetaFile contains the information to open and decrypt the FileInfo. From here FileInfo gives the information to iterate over the linked file list. The Files are decrypted and HMAC-checked with the keys in MetaFile.
If nothing is found at that address, an error is returned.

AppendToFile

When AppendToFile, we follow the same steps as in LoadFile until you have access to the FileInfo. At this stage, you take the FilePointerEnd in FileInfo and create a File at this address with its contents. You then put a random UUID at the File's tail. You also overwrite the FilePointerEnd in FileInfo with this random UUID. Everything you do is done with the keys in the MetaFile or OwnerMetaFile, depending on if you are the owner or not.

File sharing and revocation

CreateInvitation

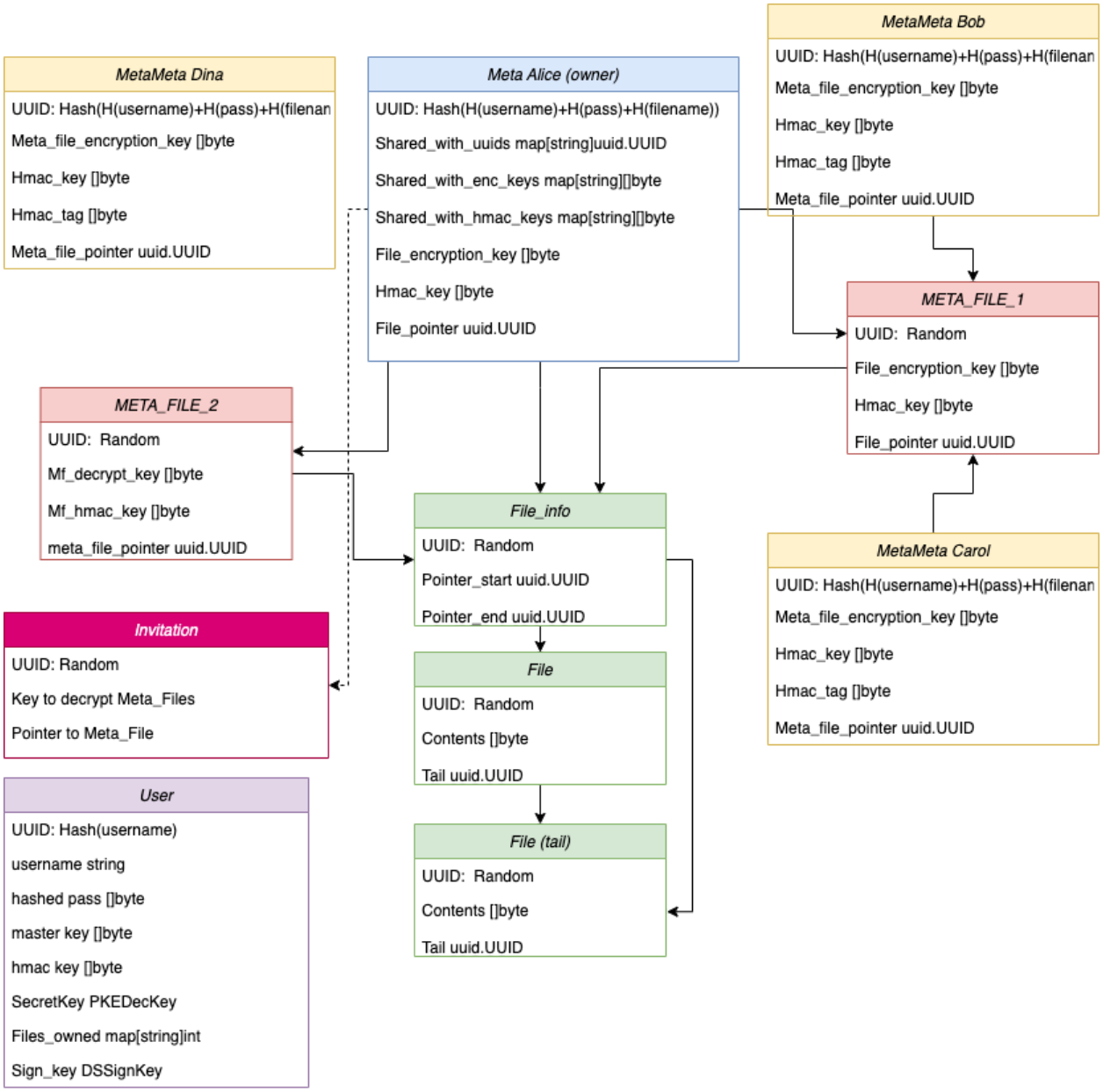
When CreateInvitation is called, after updating all changed information of the user that is logged in (changes made by alicePhone for example), the first thing to check is whether the user has access to the File he is trying to share. This is done by making the correct UUID where the file would be stored. The correct UUID is a concatenation of the user's hashed(username), hashed(hashed(password+salt)) and the hash of the filename. If the user retrieves content from this UUID in the DataStore, the file exists. Afterwards, we retrieve and re-calculate the hmac-tag for the content, check if they are similar to ensure integrity and confidentiality. If the current user has access and is the owner of the file, the user will retrieve encryption and hmac key of the file he wants to share. He then creates a new MetaFile and fills it with encryption key, hmac key and a pointer to the file. An encryption key and an hmac-key are randomly created to encrypt and HMAC-tag the meta-file. The pointer (UUID) to the MetaFile is made from a hash of the recently made encryption key. The new MetaFile is then published to the DataStore. The owner's owner_meta_file is also updated with information about the MetaFile (who the File is shared with, and the Metafiles encryption and hmac-key). An invitation, containing the MetaFile's encryption and hmac-key is made. This one is encrypted with the Public Key of the user we are sharing to. The encrypted invitation is signed with the file owner's private signature key. The encrypted and signed invitation is then sent to the other user.

If the user that is sharing the File is not the owner of the File a MetaFile is already created. Therefore the user that is sharing must retrieve information about the MetaFile. He loads the MetaMetaFile from DataStore, checks the integrity through his encryption and hmac-key, and now has a

pointer to the MetaFile. This must also be loaded and checked for integrity, before we make an invitation to the given user with the same approach as mentioned above.

AcceptInvitation
When an invitation is received we start by checking if we already have a file with the chosen filename. We then check if we can retrieve the sender's public sign key from the KeyStore. Then we load the invitation and verify its content by using the sender's public signkey. Afterwards, we decrypt the invitation with the user's private key. Then we check if the MetaFile (that points to the file) still exists, to ensure we have not been revocated. This MetaFile is calculated based on the encryption key that's located in the invitation. Afterwards, we make the new meta_meta_file struct that points to the metafile. This is encrypted with a key made by the user's masterkey and the filename, it is then added an hmac tag calculated with an hmac- key made with the same approach. The encrypted meta_meta_file is then published to DataStore.

RevokeAccess
When RevokeAccess is called, the first thing that is checked is if the owner has access to the File. If that is the case, he will load the owner_meta_file that points to the file and has information regarding all the users he has shared the File with. Its HMAC-tag is retrieved and recalculated to ensure integrity. Afterwards, we check if the user we are revocing access to really have access to the file. If he has, we delete all information about the user from the owners_metafile (the UUID of the metafile, metafiles encryption key and its hmac-key). Then we create a new encryption key, hmac-key and UUID for the file, and updates every other users' information to the new correct one. Then we iterate over the whole file, and retrieve all the contents before deleting the whole file. We then make a new file (that the other user's MetaFiles now points to), and write all the old content in this new file. This content is encrypted with an added hmac-key.


MetaMeta Dina
UUID: Hash(H(username)+H(pass)+H(filenan
Meta_file_encryption_key []byte
Hmac_key []byte
Hmac_tag []byte
Meta_file_pointer uuid.UUID
Meta Alice (owner)
UUID: Hash(H(username)+H(pass)+H(filename))
Shared_with_uuids map[string]uuid.UUID
Shared_with_enc_keys map[string][]byte
Shared_with_hmac_keys map[string][]byte
File_encryption_key []byte
Hmac_key []byte
File_pointer uuid.UUID
MetaMeta Bob
UUID: Hash(H(username)+H(pass)+H(filenan
Meta_file_encryption_key []byte
Hmac_key []byte
Hmac_tag []byte
Meta_file_pointer uuid.UUID
META_FILE_1
UUID: Random
File_encryption_key []byte
Hmac_key []byte
File_pointer uuid.UUID
META_FILE_2
UUID: Random
Mf_decrypt_key []byte
Mf_hmac_key []byte
meta_file_pointer uuid.UUID
File_info
UUID: Random
Pointer_start uuid.UUID
Pointer_end uuid.UUID
File
UUID: Random
Contents []byte
Tail uuid.UUID
File (tail)
UUID: Random
Contents []byte
Tail uuid.UUID
MetaMeta Carol
UUID: Hash(H(username)+H(pass)+H(filenan
Meta_file_encryption_key []byte
Hmac_key []byte
Hmac_tag []byte
Meta_file_pointer uuid.UUID
Invitation
UUID: Random
Key to decrypt Meta_Files
Pointer to Meta_File
User
UUID: Hash(username)
username string
hashed pass []byte
master key []byte
hmac key []byte
SecretKey PKEDecKey
Files_owned map[string]int
Sign_key DSSignKey