CS161 Project 2 Design Document

Authors: Karl Mattsson, Gavin Liu

1 Data Structures

We define inner and outer structs to store invitations, files and users, where outer structs always have pointer to its corresponding inner structs. Bascally outer struct is not encrypted and often used for checking integrety.

1.1 Outer User Struct

- Pointer to encrypted InnerUserStruct
- InnerUserStruct's HMAC
- username, which is a string
- rootKey, which is a byte string

1.2 Inner User struct

1. The user's private key for decrypting files
2. The user's private key for signing digital signatures

1.3 Outer File Struct

- the pointer to encrypted Inner File Struct
- the Inner File Struct's HMAC

1.4 Inner File Struct

- LinkListNumber
- the content of file, stored as byte

1.5 Outer Invitation Struct

- Pointer to the encrypted Inner Invitation Struct
- a digital signature

1.6 Inner Invitation Struct

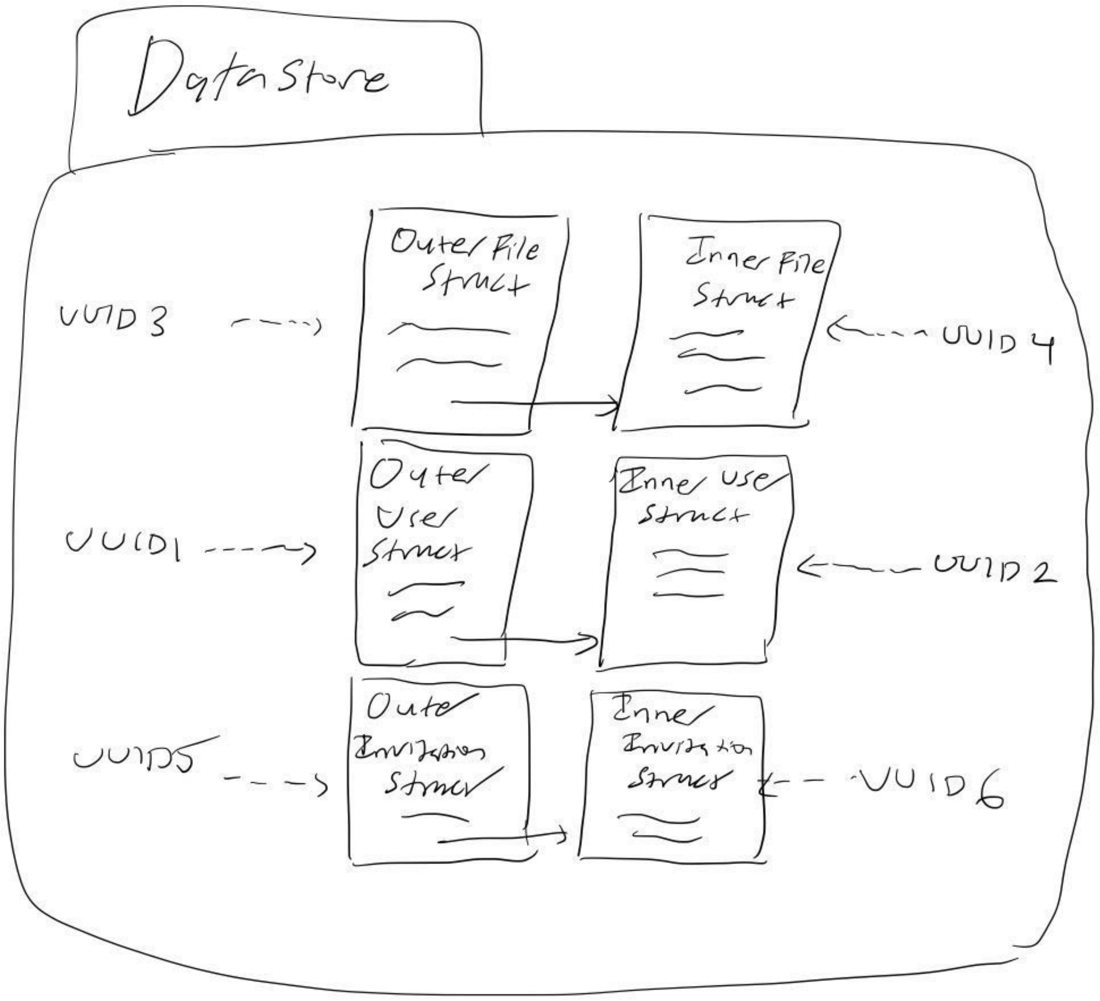
- Pointer to the sharedFileRootKey

1.7 RecipientUsernames

- list of hashed recipient user names, stored as a byte list

1.8 Files

We stored each file separately in datastore in its own UUID, as an address, where each file is stored as a double linked list with a sentinel node. We use the sentinel node to complete loadFile() and AppendToFile() operations. As the figure below,



2 User Authentication

We firstly check the validity of username, whether it is empty string or whether it already exists. When the username is valid, then start to init Outer

2.1 User initiation

1. check the validity of username
   (a) whther it's empty or not
   (b) whether it exists or not
2. call the helper function

   ```
   CreateAndStoreInnerAndOuterUserStructs
   ```

   (a) initate outer user struct and inner user struct
   (b) generate digital signature and RSA keys and store them at Keystore
   (c) return the pointer of outer user struct (We do not return pointer to inner user struct, though it is generated by the helper function)

2.2 Authenticate users

Each time we need to access an existing user account, we need to check the user password and his files' integrety.

1. check the validity of the username — if it exists in the datastore, if not return error
2. generate rootKey using PBKDF, based on user input password
3. call helper function

   `LoadOuterUserStructFromDatastore`

   (a) get the marshalled encrypted userstruct from datastore and unmarshal it
   (b) **check the HMAC of encrypted userstruct, since the HMAC generated based on rootKey and rootKey is generate based on password, once the password is incorrect, the HMAC won't match. At the same time, when the userstruct is tampered, the HMAC won't match either.** Therefore both situation is considered.

2.3 Multiple Client Sessions

To support multiple session of the same user, we compute file's UUID on the fly each time, based on the file names, so that everything Alice's laptop modifies something, her phone could also get the latest version of files by recomputing those UUIDs.

3 File Storage and Retrieval

3.1 StoreFile

As discussed in Data Struct part, different files are separated from each other, stored as different "file blocks" with a unique UUID. In each file block, contents of the file are stored as a double linked list, with a sentinel node in each block.

This function assumes the user just storing the first file, so in this function, we just do first file block initialization as well as sentinel node. The whole procecss are as follows:

1. using user's rootKey to generate a fileRootKey by calling helper function:

   `GetSharedFileRootKey(rootKey, filename)`

2. store the first file struct

   (a) initiate every variable in outer file struct
   (b) call the helper function to create outer file struct and inner file struct. which does following steps:

   ```
   CreateAndStoreInnerAndOuterFileStructs(filename, content, fileRootKey, revokeNumber,
   linkedListNumber, linkedListSize)
   ```

      i. generate UUID for each inner and outer file struct
      ii. create file outer struct given linkedlistNumber > 0.
      iii. create inner file struct by initiate LinkedListNumber and content variable
      iv. marshal, encrypt and then put the inner file struct in datastore.
      v. generate HMAC and create outer file struct and then store it in datastore (WITHOUT encryption)

   (c) create sentinel file struct using the same steps as creating other file structs, with the LinkListNumber $= 0$ instead some value > 0.

3. store the sentinel node file struct by the same steps

3.2 File Retrieval

3.2.1 LoadFile

1. get all the keys needed

   (a) get rootKey at first
   (b) calling helper function based on user's rootKey

   `GetSharedFileRootKey()`

2. initiate revokeNumber and linkedListnumber, which is for revoke user and appendToFile() respectively.
3. calling the helper function to access the sentinel node, which returns

   `LoadInnerFileStructFromDatastore()`

4. loading file content, stored as double linked list, using sentinel node and for loop accessing each content

3.2.2 AppenToFile

1. get necessary keys
    (a) firstly get user's rootKey.
    (b) then get fileRootKey based on the rootKey.
2. singly load the sentinel struct, and update the LinkListSzie and LinkedListNumber, and store it back
3. call the helper function:

    ```
    CreateAndStoreInnerAndOuterFileStructs(filename, content, fileRootKey, revokeNumber,
    linkedListNumber, linkedListSize)
    ```
    with updated LinkedListNumber, then the appended content could go into the file.

For bandwith consideration, since we only load and store the sentinel which has no content. Then we only store the new content to be appended to the datastore. Therefore the runtime of AppendToFile only is concerning size of the content to be appended, which conforms the requirement.

4 File Sharing and Revocation

We define *Alice* as sender, *Bob* and others are receivers

4.1 Sending Invitation

Alice wants to invite Bob to work on file foo.txt, so she needs to share with Bob the symmetric key that is used to encrypt and decrypt foo.txt. Steps:

1. check the file's integrety before sending invitation. We are doing this by loading the file. If the file could not be loaded, then it is tampered.
2. create a shared fileRootKey based on fileRootKey, in order to enable users invited to invite other users.
3. create the outer invitation struct and store it in the datastore
4. add the recipient name to the list called

    ```
    recipientUsername
    ```

4.2 Accepting Invitation

Bob wants to accept the invitation from Alice on foo.txt. Steps:

1. Bob goes to the Datastore at the invitationPtr UUID. This UUID is given in the arguments to AcceptInvitation()
2. Bob gets the serialized struct using DatastoreGet(invitationPtr), and then deserializes the Outer Invitation Struct: P = json.Unmarshal(Outer Struct).
3. Bob checks signature using Outer Invitation Struct to see if invitation is tampered. To do this, he fistly calls helper function getPublicKey() to get DSSVerifyKey then call DSVerify().
4. If signature is valid, Bob starts to decrypt Inner Invitation Struct. He firstly get
5. Bob needs to check:
    (a) if the signature of, in the Outer Struct, the Inner Struct is correct using DSVerify(Alice's DSVerifyKey, E, S); if the digital signature is valid; if not, it means the invitation not from Alice or files being tampered or invitation being revoked;
    (b) if Bob's personal namespace has the same file name as the one being shared; if so, return err.

4.3 Revoking

1. check if the user being revoked valied, if one of following conditions do not satify, then it is invalid.
   (a) username exists in the recipient name list.
   (b) The given filename is not currently shared with recipientUsername.
2. Get locations of the invitation structs and the sharedFileRootKey
3. Set the invitation to garbage
4. put the copy of the file in the datastore using

   ```
   StoreFile()
   ```

5. change the sharedFileRootKey of the file for other users not being revoked

Helper Functions

1. IsEmptyUsername: takes in a string and see if it's a username

   ```
   func IsEmptyUsername(username string) (isEmpty bool)
   ```

2. CheckUserExists: takes in a string as a username and see if the username already exists

   ```
   func CheckUserExists(username string, password string) (exists bool)
   ```

3. CreateAndStoreInnerAndOuterFileStructs:

   ```
   func CreateAndStoreInnerAndOuterFileStructs(filename string, content []byte, fileRootKey []byte,
   revokeNumber int, linkedListNumber int, linkedListSize int) (err error)
   ```

4. CreateAndStoreInnerAndOuterUserStructs: it takes in a username and password, and creates outer user struct and inner user struct. It return pointers to each struct.

   ```
   func CreateAndStoreInnerAndOuterUserStructs(username string, password string)
   (innerUserStructPtr *InnerUser, outerUserStructPtr *User, err error)
   ```

5. CreateAndStoreInnerAndOuterFileStructs:

   ```
   func CreateAndStoreInnerAndOuterFileStructs(filename string, content []byte, fileRootKey []byte,
   revokeNumber int, linkedListNumber int, linkedListSize int) (err error) {
   ```

6. LoadOuterUserStructFromDatastore: it takes in a rootKey, and returns a pointer to its corresponding outer user struct.

   ```
   func LoadOuterUserStructFromDatastore(rootKey []byte) (outerUserStructPtr *User, err error)
   ```

7. LoadOuterUserStructFromDatastoreLoadInnerUserStructFromDatastore

   ```
   func LoadInnerUserStructFromDatastore(rootKey []byte) (innerUserStructPtr *InnerUser, err error)
   ```

8. LoadOuterFileStructFromDatastore

   ```
   func LoadOuterFileStructFromDatastore(filename string, fileRootKey []byte, revokeNumber int,
   linkedListNumber int) (outerFileStructPtr *OuterFile, err error)
   ```

9. LoadInnerFileStructFromDatastore:

```
        func LoadInnerFileStructFromDatastore(filename string, fileRootKey []byte, revokeNumber int,
        linkedListNumber int) (innerFileStructPtr *InnerFile, err error)
```

10. LoadInnerUserStructFromDatastore: it takes in a rootKey and returns a pointer to its corresponding inner user struct.

```
    func LoadInnerUserStructFromDatastore(rootKey []byte) (innerUserStructPtr *InnerUser, err error)
```

11. GenerateUUID: it takes in a byte argument and returns a UUID generated based on it.

```
    func GenerateUUID(key []byte) (location UUID, err error)
```

12. ComputeUserFileRootKey: compute Alice's userFileRootKey (the fileRootKey in her personal namespace)

```
        func ComputeUserFileRootKey(rootKey []byte, filename string) (userFileRootKey []byte, err error)
```

13. GenerateUserStructKeys: it takes in a rootKey, and returns outerUserStructKey, innerUserStructEncryptDecryptKey and innerUserStructHMACKey; which are used in generating UUIDs and serve as arguments for HashKDF for HMAC key, encrypt and decrypt the InnerUserStruct (Symmetric Key), and generate HMAC for InnerUserStruct.

```
    func GenerateUserStructKeys(rootKey []byte) (outerUserStructKey []byte, innerUserStructEncryptDecryptKey
[]byte, innerUserStructHMACKey []byte, err error)
```

14. GetDatastorePointersToUserStructs: it takes in a rootKey, and returns pointers to OuterUserStruct and InnerUserStruct.

```
    func GetDatastorePointersToUserStructs(rootKey []byte) (datastorePointerToOuterUserStruct UUID,
    datastorePointerToInnerUserStruct UUID, err error)
```

15. GenerateFileStructKeys: it takes in a byte string as fileRootKey, a string as filename, a revoke number and a linkedListNumber. It returns a outerFileStructKey, an innerFileStructEncryptDecryptKey and an innerFileStructHMACKey.

```
    func GenerateFileStructKeys(fileRootKey []byte, filename string, revokeNumber int, linkedListNumber int)
(outerFileStructKey []byte, innerFileStructEncryptDecryptKey []byte, innerFileStructHMACKey []byte,
    err error)
```

16. MarshalEncryptStoreInnerStruct: it takes in an interface of InnerFileStruct, an innerStructEncryptKey and a pointer to an InnerStruct. It returns a marshalled innerStruct.

```
func MarshalEncryptStoreInnerStruct(innerStructI interface{}, innerStructEncryptKeyI interface{},
datastorePointerToInnerStruct UUID) (marshalledInnerStruct []byte, encryptedInnerStruct []byte, err
```

17. MarshalStoreOuterStruct: it takes in an outerFileStruct interface and a pointer to an outerStruct. It returns the marshalled outerStruct.

```
        func MarshalEncryptStoreInnerStruct(innerStructI interface{}, innerStructEncryptKeyI interface{},
        datastorePointerToInnerStruct UUID) (marshalledInnerStruct []byte, encryptedInnerStruct []byte,
        err error)
```

18. DecryptUnmarshalInnerStruct: it takes in an encrypted innerStruct and the Symmestric key to it, and returns the decrypted, unmarshalled innerStruct.

```
    func DecryptUnmarshalInnerStruct(encryptedInnerStruct []byte, innerStructDecryptKey []byte)
    (plainTextInnerStruct interface{}, err error)
```

19. GetUserPublicKeys

```
        func GetUserPublicKeys(rootKey []byte) (PKEEncKey PKEEncKey, DSVerifyKey DSVerifyKey, err error)
```

20. GetUserPublicKeysFromUsername:

```
        func GetUserPublicKeysFromUsername(username string) (PKEEncKey PKEEncKey, DSVerifyKey DSVerifyKey,
        err error)
```

21. GetUserPrivateKeys: it takes in a rootKey, and returns the user's private key for decrypting public encrypted stuff, as well as private for digital signature signing stuff.

```
    func GetUserPrivateKeys(rootKey []byte) (PKEDecKey PKEDecKey, DSSignKey DSSignKey, err error)
```

22. GetDatastorePointersToInvitationStructs: it takes in a rootKey, sharedFileRootKey, recipientUsername (the invitation to whom) and a filename (the invitation to which file). It returns a pointer of OuterInvitationSTruct and one to the InnerInvitationStruct.

```
    func GetDatastorePointersToInvitationStructs(rootKey []byte, filename string, recipientUsername string)
(datastorePointerToOuterInvitationStruct UUID, datastorePointerToInnerInvitationStruct UUID, err error)
```

23. CreateAndStoreInnerAndOuterInvitationStructs: it takes in everything GetDatastorePointersToInvitationStructs does and return the same, but is different in process.

```
    func CreateAndStoreInnerAndOuterInvitationStructs(rootKey []byte, sharedFileRootKey []byte,
    filename string, recipientUsername string) (datastorePointerToInnerInvitationStruct UUID,
    datastorePointerToOuterInvitationStruct UUID, err error)
```

24. GenerateDatastorePointerToEncSharedFileRootKey: it takes in a rootKey, a file name and recipientUsername and return a poiter to sharedFileRootKey.

```
        func GenerateDatastorePointerToEncSharedFileRootKey(rootKey []byte, filename string,
        recipientUsername string) (datastorePointerToEncSharedFileRootKey UUID, err error)
```

25. StorePointerToEncSharedFileRootKey: Only use this function when accepting an invitation. The receiver stores the content of the InnerInvitation Struct in his own "personal namespace".

```
    func StorePointerToEncSharedFileRootKey(rootKey []byte, filename string, pointerToEncSharedFileRootKey
UUID) (err error)
```

26. GetSharedFileRootKey(): Only use this function after accepting an invitation (for example, in StoreFile, LoadFile, AppendToFile). This function takes in the rootKey of the RECEIVER (Bob), and returns the pointer to sharedFileRootKey If the sharedFileRootKey does not exist for the file, return the userFileRootKey. The owner will not have his own fileRootKey stored in the Datastore.

```
        func GetSharedFileRootKey(rootKey []byte, filename string) (sharedFileRootKey []byte, err error)
```

27. FileExistsInUsersPersonalNamespace: Function to check if a file exists in the User's personal namespace already. If it does, return true. Otherwise, return false. If an error occured, the exists bool is false by default.

```
func FileExistsInUsersPersonalNamespace(rootKey []byte, filename string) (fileExistsInUsersPersonalNamespace
bool, err error)
```

28. FileIsSharedWithUser: Returns true if the file with filename, created by a user with the provided rootKey, is shared with the recipientUsername (the invitation struct exists and is not garbage). If an error occurs, fileIsSharedWithUser is set to false by DEFAULT (but could be set to true)

```
        func FileIsSharedWithUser(rootKey []byte, filename string, recipientUsername string)
        (fileIsSharedWithUser bool, err error)
```