CS 161 Project 2 Design Doc

Data Structures

```
type User struct {
  Username  string
  DSSignKey userlib.DSSignKey
  PKEDecKey userlib.PKEDecKey
  encKey    []byte
  macKey    []byte
}
```

```
type FileNode struct {
  Username      string
  Filename      string
  FileHead      uuid.UUID
  Children      []uuid.UUID
  ChildrenNames []string
}
```

```
type Invitation struct {
  Owner      string
  ParentNode uuid.UUID
  FileKey    []byte
}
```

```
type FileHead struct {
  FirstNode uuid.UUID
  LastNode  uuid.UUID
}
type ContentNode struct {
  Contents uuid.UUID
  NextNode uuid.UUID
}
```

User Authentication

Account Setup

When `InitUser` is called, we first need to generate two key pairs: one for digital signatures and one for RSA encryption. We add the public keys to the keystore under UUIDs generated from `getUUID("ds", username)` and `getUUID("pke", username)` respectively. Next, we generate a 16-byte root key using `Argon2Key(password, salt, 16)`, where `salt` is 32 random bytes. From this, we can derive keys for symmetric encrypting and authenticating using `HashKDF(rootKey, []byte("enc-key"))` and `HashKDF(rootKey, []byte("mac-key"))`. We can finish by encrypt/tagging the user struct with the private keys before saving in the datastore under `getUUID("struct", username)`. The last step is to store the salt under `getUUID("salt", username)`.

Logging In

Now, when a user wants to authenticate, they will call to `GetUser` with their username and password. We re-derive their root key using their password, the salt stored in datastore, and `Argon2Key`. We then re-derive the user MAC key with `HashKDF`, and use `HMACEqual` with the user tag to verify that the entry in the datastore has not been compromised. Finally, we re-derive the user encryption key with `HashKDF` and decrypt the user struct!

File Storage and Retrieval

Storing A File

To start, we create a new `FileNode` struct to be stored at `getUUID(filename, username)`. We also need to create a `ContentNode` to point to our file content and a `FileHead` that points to our `ContentNode`. Once we have initialized all the necessary structs with appropriate arguments, we can generate the file key and file MAC key to encrypt/verify them. The file key can be created with 16 random bytes, and the file MAC key can be derived from it with `HashKDF(fileKey, []byte("mac-key"))`. We then use these keys to encrypt/tag all the structs. Finally, we encrypt/tag the file key and username and store them at `getUUID(filename+"key", username)` and `getUUID(filename+"owner", username)` respectively.

If the file already exists, we want to overwrite it. We first go to the file key and try to verify with the user's MAC key. If that doesn't work, we get the owner's username and try their DSVerifyKey as well (in case the owner had to change the file key). Using the key, we can recursively delete our linked list of `ContentNode`'s. Finally, we

create a new `ContentNode` using the same process as above and set the `FirstNode` and `LastNode` in `FileHead` to its UUID.

Loading Files

To load files, we retrieve the file key as above, verify/decrypt the `FileNode` and `FileHead`, then go to our linked list of `ContentNodes`. Once set up, we can simply loop through the list, verify/decrypt the nodes, aggregate the contents of each one, and return in one package.

Appending to Files

To append to a file, we first create a new `ContentNode` with the provided content and a nil `NextNode`, then encrypt/verify with the file key at a random UUID. Next, we retrieve the `FileHead` and hop to the last node. Here, we set the `NextNode` UUID to the UUID of our new `ContentNode`, set the `LastNode` value of the `FileHead` to the new UUID, encrypt/tag everything, and return.

File Sharing and Revocation

Sharing Files Through Invitations

We can create the `Invitation` struct with the fileowner's username, the file key, and the UUID of the user's `FileNode`. We then marshal the struct, encrypt with the recipient's PKEEncKey, sign with the user's DSSignKey, and store it at a random UUID. We also add the recipient to a list in the current user's `FileNode` to keep track of who was shared with.

Accepting Invitations

To accept an invitation, we will need to create a new `FileNode` struct in the current user's file namespace using info from the parent node. After we verify/decrypt the invitation, we can go to the parent's `FileNode`, verify/decrypt using the file key, and add the new `FileNode` UUID to the list of children nodes. The last step is to store the file key and the owner's username separately as described above (see Storing A New File).

Revoking Access

To revoke access from a user and their children, we first need a new file key. Using it, we encrypt/tag each content-related entry at a newly generated location. Next, we cut the revoked user out of the children list. We also need to traverse the tree and update the file nodes with the address of the new file head, and finally overwrite each user's file key with the new one.

Helper Methods

```
// Here we hash the query and username separately then generate a uuid from their bytes
func getUUID(query string, username string) (userID string)

func symEncThenTag(encKey []byte, macKey []byte, content interface{}, id uuid.UUID) (err error)

func symVerifyThenDec(encKey []byte, macKey []byte, id uuid.UUID) (content []byte, err error)

func asymEncThenTag(username string, signKey userlib.DSSignKey, content interface{}, id uuid.UUID) (err error)

func asymVerifyThenDec(username string, decKey userlib.PKEDecKey, id uuid.UUID) (content []byte, err error)
```

Diagram

UUID Derivation (args to GetUUID)	Encryption Method	Enc/Tag Key	Value at UUID	Description
“struct”, username	SymEnc	encKey, macKey	User struct, plus the salt and tag	Username + private keys. Needs salt to rederive root key.
filename, username	SymEnc	fileKey	FileNode struct plus tag	Owner + FileHead + parent/children. All users who can access the file have the file key, so they can derive the file node key and decrypt.
N/A (random)	SymEnc	fileKey	FileHead struct plus tag	FirstNode + LastNode. Start and end of linked list that stores file contents
N/A (random)	SymEnc	fileKey	ContentNode struct plus tag	Node contents + UUID of next node in list.
filename + “key”, username	SymEnc/AsymEnc	encKey, macKey / pkeEncKey, signKey	File Key plus tag	Base file key used to encrypt/tag file related values. Shared with all authorized users. If verify/decrypt doesn’t work using personal keys, try RSA scheme with owner’s verify key.
N/A (random)	AsymEnc	pkeEncKey, dsKey	Invitation Struct plus tag	Invitation struct, UUID is passed to the recipient
filename + “owner”, username	SymEnc	encKey, macKey	Owner’s username	The given file’s owner’s username, in case the owner needs to change the file key when revoking.