Project 2 Design Doc

Storage

Keystore: public keys
Datastore: everything else

Data structures

User struct

- Datastore ID: UUID generated from username + password + salt
- username and password: only stored on client, not uploaded to Datastore
- **InvitationPrivateKey**: private key used to sign invitations (public key for verify is stored in Keystore)
- **AccessPrivateKey**: private key used to decrypt invitations (public key for encrypt is stored in Keystore)
- **OwnedFiles**: used to track session sign/encrypt keys for an owned file
    - Map of format {personal filename: {'encryptKey': key, 'signKey': key}, …}
- **OwnedFilesUserManagement**: used to track sign/encrypt keys for session key structs that are distributed to other users.
    - Map of format {personal filename: {user id: {'encryptSessionKeyStructKey': key, 'signSessionKeyStructKey': key}}}
- **OwnedSessionKeyStructIDs**: used to track the uuid that a specific session key struct is stored at in Datastore for owned files
    - Map of format {personal filename: {user id: uuid of user specific session key struct}
- **AccessibleFiles**: used to track user-specific session key struct sign/encrypt keys for files that are accessible but not owned
    - Map of format {personal filename: {'encryptSessionKeyStructKey': key, 'signSessionKeyStructKey': key}}
- **AccessibleSessionKeyStructIDs**: used to track the uuid that a user-specific session key struct is stored at in Datastore for files that are accessible but not owned
    - Map of format {personal filename: uuid of user specific session key struct}

SessionKey Struct

- Datastore ID: UUID - random
- **EncryptSessionKey** []byte: used to encrypt a file struct
- **SignSessionKey** []byte: used to sign a file struct
- **FileID** uuid.UUID: the UUID of the corresponding File Struct
- → This struct is encrypted and signed with user-specific keys stored in AccessibleFiles

- → This struct is created by the owner of a file each time the owner shares with another user

File struct

- Datastore ID: hash created with original filename + owner's username
    - (uuid.FromBytes(userlib.Hash([]byte(filename + userdata.username))[:16]))
- **NextFileBlockID** uuid.UUID: points to start of chain of file blocks for the file
- → Using the session keys found in a session key struct, encrypt with symEnc, sign with HMAC

FileBlock Struct

- Datastore ID: UUID - random
- **FileContentBlockID** uuid.UUID: points to the actual file content in this block
- **NextFileBlockID** uuid.UUID: points to the next file block for the file (last block points to uuid.Nil)
- → Using the session keys found in a session key struct, encrypt with symEnc, sign with HMAC

FileContentBlock Struct

- Datastore ID: UUID - random
- **FileContent** []byte: the actual byte content of this file block
- → Using the session keys found in a session key struct, encrypt with symEnc, sign with HMAC

DatastoreValue struct

- **Ciphertext** []byte
- **Tag** []byte
- → This struct is used as a wrapper around other structs to store both the encrypted struct and the signature of that encrypted struct onto the datastore.

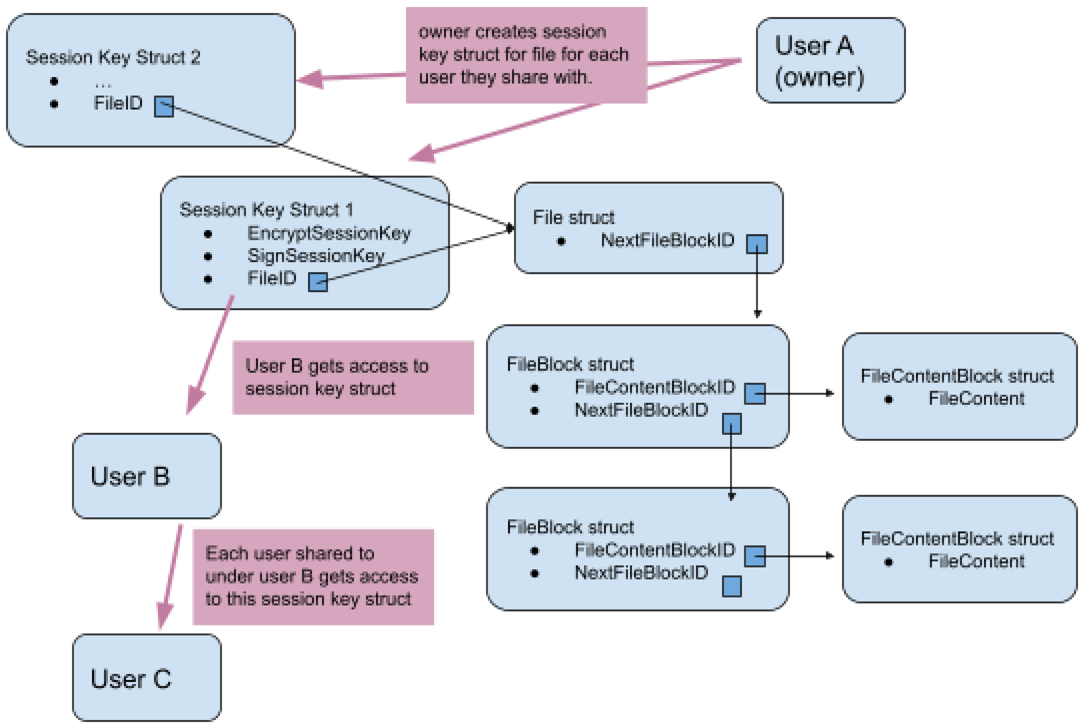
Invitation struct

- Datastore ID: random UUID
- **EncryptSessionKeyStructKey** []byte: the user specific encrypt key for the session key struct (every other user who this recipient shares the file to will also receive this key)
- **SignSessionKeyStructKey** []byte: the user specific sign key for the session key struct (same as above)
- **SessionKeyStructID** uuid.UUID: the UUID pointing to where the session key struct (which contains the actual file session keys) is stored in Datastore
- → Using 2 randomly generated keys, encrypt with symEnc, sign with HMAC

InvitationWrapper struct

- **EncryptedInvitation** []byte - the symmetrically encrypted invitation struct
- **EncryptedInvitationSignature** []byte - the HMAC signature corresponding to the encrypted invitation struct
- **EncryptedEncryptInvitationKey** []byte - the randomly generated key used to encrypt the invitation struct, encrypted using the recipient's AccessPublicKey
- **EncryptedSignInvitationKey** []byte - the randomly generated key used to sign the invitation struct, encrypted using the recipients AccessPublicKey
- **EncryptedEncryptInvitationKeySignature** []byte - the digital signature on the above encrypted encrypt key, created using the sender's InvitationPrivateKey with DSSign
- **EncryptedSignInvitationKeySignature** []byte - the digital signature on the above encrypted sign key, created using the sender's InvitationPrivateKey with DSSign
- → This struct is exclusively used (instead of DatstoreValue struct) to wrap invitations to store in Datastore, as hybrid encryption is needed to pass invitations between users

Overview of How Data Structures Are Related



User Authentication

```
InitUser(username string, password string) (userdataptr *User, err error)
```

- Verify valid fields entered

    - Check that username is unique
        - See if userUUID from that username is already stored in Datastore
    - Check lengths valid (not length 0)
- Generate relevant keys
    - access key pair - PKEKeyGen() → private key stored in User struct, public key stored in Keystore
    - invitation key pair - PKEKeyGen() → for signing the Invitation struct, private key stored in User struct, public key stored in Keystore
- Store User struct in Datastore
    - Key: uuid.FromBytes(hashedUsername)
    - Value: new datastoreValue struct
        - For the user struct:
            - First marshal the user struct, then encrypt marshaled user struct with SymEnc, using key generated from PBKDF on hashed username + password
            - Sign user struct with HMAC
        - Store encrypted marshaled user struct as the ciphertext
        - Store the HMAC as the signature tag

```
GetUser(username string, password string) (userdataptr *User, err error)
```

- Retrieve the datastoreValue struct from Datastore using uuid.FromBytes(hashedUsername): this contains the encrypted User object and the HMAC
- Verify retriever's identity with HMAC
- Generate key using hashed username and password, input to PBKDF to get key, use key and attempt to decrypt User object
    - If successful, store username and password onto client, return userdataptr (&userdata)

File Storage and Retrieval

```
StoreFile(filename string, content []byte) (err error)
```

- Fetch user data to client to support multiple sessions
    - Use helper function: userdata.FetchUserDataToClient()
- Validate file fields
    - Filename (Check if File exists in either user's OwnedFiles dictionary or AccessibleFiles dictionary)
- If filename does not exist (i.e. need to create new file):
    - Create 2 session keys and fileID, store them to the 3 owner dictionaries (OwnedFiles, OwnedFilesUserManagement, OwnedSessionKeyStructIDs)

        - fileID is generated by: `uuid.FromBytes(userlib.Hash([]byte(filename + userdata.username))[:16])`

- Regardless of existing or new file:
    - Create a new file content block, file block, and file struct and store in Datastore.
        - Key: random UUID, Value: Struct.
        - For each struct, set each field to the necessary value
            - Ex. FileContentBlockID for the file block struct is set to the UUID of the new file content block struct
        - Encrypt with & Sign before storing
            - Use DatastoreValue struct to store both the struct and signature
    - Update new user data to server to support multiple sessions
        - userdata.UpdateUserDataToServer()

```
LoadFile(filename string) (content []byte, err error)
```

- Fetch user data to client to support multiple sessions
- Check that file exists in either OwnedFiles map or AccessibleFiles map
- If file exists:
    - Retrieve relevant user-specific key and file's session key struct id from file dict
    - Retrieve session key struct from id
    - Use key to decrypt and retrieve session keys and file id for file
    - Load encrypted File struct and decrypt/verify signature using session keys
    - Validate user access
        - If session key doesn't work to decrypt / HMAC not valid then error immediately
    - Traverse and decrypt file blocks pointed to by File struct and following File Block structs
        - For each File block structs, decrypt the pointed File Content Block structs and append all file contents together
        - Continue until NextFileBlockID for a FileBlock struct is uuid.UUIDNil
    - Return the appended content

```
AppendToFile(filename string, content []byte) (err error)
```

- Fetch user data to client to support multiple sessions
- Check if file exists in either OwnedFiles map or AccessibleFiles map
- If so:
    - Retrieve encrypt and sign session keys, as well as the UUID for the File struct
    - Download and decrypt the outermost File struct, and retrieve the first FileBlock struct
    - Traverse to the end of the FileBlock struct chain
        - for fileBlockStruct.NextFileBlockID != uuid.Nil

    - Create new FileBlock and FileContentBlock structs with relevant fields, store them to Datastore, and update the NextFileBlockID of the last FileBlock struct (found in the previous step).

File Sharing

```
CreateInvitation(filename string, recipientUsername string) (invitationPtr
UUID, err error)
```

- Fetch user data to client to support multiple sessions
- Check if file exists in either OwnedFiles map or AccessibleFiles map
- If so:
    - First verify that the calling user is not revoked by trying to download and decrypt outermost file struct
    - Verify recipientUsername exists
    - Check if the calling user is the owner of the file
    - If owner:
        - Generate new user-specific keys, create, encrypt and sign new SessionKey struct and store to Datastore.
        - Update OwnedFilesUserManagement and OwnedSessionKeyStructIDs dictionary
    - If not the owner:
        - Retrieve relevant keys and structs
            - sessionKeyStructID, sessionKeyStructKeys, encrypt/sign sessionKeyStructKeys
    - Create and Marshal an Invitation struct
    - Generate new symmetric keys (for hybrid encryption)
        - encryptInvitationKey, signInvitationKey
    - Encrypt marshaled Invitation struct with encryptInvitationKey, sign with signInvitationKey
    - Encrypt the both symmetric keys (encryptInvitationKey and signInvitationKey) with access public key of the recipient
    - Sign the symmetric keys using calling user’s invitation private key
    - Create new InvitationWrapper struct, store all the relevant data in its fields
    - Marshal the new InvitationWrapper struct, store it into Datastore with new invitationPtr UUID (random).
    - Update new user data to server to support multiple sessions
    - Return the invitationPtr

```
AcceptInvitation(senderUsername string, invitationPtr UUID, filename
string) (err error)
```

- Fetch user data to client to support multiple sessions
- Check if file exists in either OwnedFiles map or AccessibleFiles map

- If not:
    - Get the invitation object from the marshaled InvitationWrapper struct stored in Datastore at key invitationPtr
    - Using sender’s invitation public key, verify integrity of encryptedEncryptInvitationKey and encryptedSignInvitationKey
    - Decrypt encryptedEncryptInvitationKey and encryptedSignInvitationKey using own access private key
    - Verify HMAC of invitation was created by senderUsername with signInvitationKey
    - Get session key struct keys
        - Decrypt invitation object with encryptInvitationKey
    - Validate that the senderUser was not revoked from the file
        - ValidateAccessToFile(accessibleFileInfo, sessionKeyStructID)
    - Store in user's AccessibleFiles and AccessibleSessionKeyStructIDs map
    - Update new user data to server to support multiple sessions

File Revocation

```
RevokeAccess(filename string, recipientUsername string) (err error)
```

- Fetch user data to client to support multiple sessions
- Check if file exists in OwnedFiles map
- If so:
    - retrieve original session keys for decrypting file
    - Check that the file is currently shared with recipientUsername
        - Using OwnedFilesUserManagement map
    - Delete the recipientUsername from owner dictionaries
    - Generate new session keys, and update the OwnedFiles dictionary
    - For all users who still have access:
        - Edit the session key structs by:
            - 1. Get session key struct for user
            - 2. Get user-specific keys
            - 3. Check integrity of the session keys struct
            - 4. Update session key struct
            - 5. Sign and encrypt session key struct
    - Traverse through FileBlock structs and FileContentBlock structs and decrypt/re-encrypt them with new session keys
    - Update new user data to server to support multiple sessions

Helper Functions

```
func GetUserUUID(username string) (uuid.UUID, error)
```

- Used to return the UUID at which the User struct is stored at
- Hash the username and call uuid.FromBytes(hashedUsername)

```
func GetUserKeys (username string, password string) ([]byte, []byte)
```

- Used to returns userEncryptKey and userSignKey, which are used to encrypt and sign the User struct
- Hash the username, and use the first 16 bytes as userEncryptKey and next 16 bytes as userSignKey

```
func ParseAccessibleFileInfo (sessionKeyStructID uuid.UUID,
sessionKeyStructEncryptKey []byte, sessionKeyStructSignKey []byte)
(encryptSessionKey []byte, signSessionKey []byte, fileID uuid.UUID, err
error)
```

- Used to find the encryptSessionKey, signSessionKey, and fileID(UUID of the File struct), given the sessionKeyStructID, sessionKeyStructEncryptKey, and sessionKeyStructSignKey
- Verify integrity of the session key struct by evaluating HMAC using sessionKeyStructSignKey
- Note that function does not attempt to use retrieved session keys to decrypt the actual File struct - keys may no longer be valid if owner has revoked access

```
func ValidateAccessToFile (accessibleFileInfo map[string][]byte,
sessionKeyStructID uuid.UUID) (err error)
```

- Given an accessibleFileInfo map - which stores encryptSessionKeyStructKey and signSessionKeyStructKey, validate access to the file
- Calls ParseAccessibleInfo to obtain signSessionKey and storageKey
- Download and verify integrity of the outermost File struct using the signSessionKey

```
func (userdata *User) CheckFileExists (filename string) (encryptSessionKey
[]byte, signSessionKey []byte, storageKey uuid.UUID, exists bool, err
error)
```

- Convenience function to check whether the filename exists in the user's personal namespace, whether as an owned file or just an accessible file
- Returns corresponding encrypt and sign session keys for the file (calls ParseAccessibleFileInfo)

```
func RetrieveDecryptedFileStruct (fileID uuid.UUID, signSessionKey []byte,
encryptSessionKey []byte) (fileStruct File, err error)
```

- Download encrypted and signed File struct from Datastore
- Verify integrity by checking HMAC signature using passed in keys
- If succeed, return File struct as an object (not JSON)

```
func RetrieveDecryptedFileBlockStruct (fileBlockID uuid.UUID,
signSessionKey []byte, encryptSessionKey[]byte) (fileBlockStruct FileBlock,
err error)
```

- Download encrypted and signed FileBlock struct from Datastore
- Verify integrity by checking HMAC signature using passed in keys
- If succeed, return FileBlock struct as an object (not JSON)

```
func RetrieveDecryptedFileContentBlockStruct (fileContentBlockID uuid.UUID,
signSessionKey []byte, encryptSessionKey []byte) (fileContentBlockStruct
FileContentBlock, err error)
```

- Download encrypted and signed FileContentBlock struct from Datastore
- Verify integrity by checking HMAC signature using passed in keys
- If succeed, return FileContentBlock struct as an object (not JSON)

```
func (userdata *User) FetchUserDataToClient() (err error)
```

- Used to support multiple user sessions, used at the beginning of some client API calls
- Refetch user data from Datastore using username and password stored on the client
- Verify integrity of user data, then update client’s local user data with fetched user data

```
func (userdata *User) UpdateUserDataToServer() (err error)
```

- Used to support multiple user sessions, used at the end of some client API calls
- Save changes to user data from client to Datastore
- Reencrypt and sign data using username and password stored on the client