CS 161 Project 2 Design Document

Data Structures

User Struct

```
type User struct {
  MyUUID uuid.UUID
  Username string
  Salt []byte
  PasswordHash []byte
  EncPrivateKey []byte
  EncSignKey []byte
  VerifiedSig []byte
  raw_password string
}
```

When InitUser() is called, we create a user struct using the information passed in, and initiate every field:

- MyUUID: A UUID generated from the user's username and used to identify the user struct in the datastore.
- Username: The username passed in by the user.
- Salt: A random salt generated for each user to be used in password hashing.
- PasswordHash: The hashed password generated using PBKDF.
- EncPrivateKey : The RSA private key encrypted with a symmetric key derived from the user's password.
- EncSignKey: The signature private key encrypted with a symmetric key derived from the user's password.
- VerifiedSig: The signature of all the user struct fields signed with the user's DSA private key.
- raw_password (Private field): The password passed in by the user, not marshaled.

File Structs

```
type ShareNode struct{
```

```
  MyUUID uuid.UUID // UUID generated from the user's username and filename
  ChildrenList []uuid.UUID // list of uuid's of children Invitations
  InvitationPtr uuid.UUID // uuid of the corresponding Invitation
  IsOwner bool // a boolean indicates whether the user corresponding to the ShareNode is the owner of the file
  Signature []byte // the signature that protects all the fields in the struct
}
type Invitation struct{
  MyUUID uuid.UUID // InvitationPtr in ShareNode
  SharePtr uuid.UUID // UUID of the corresponding ShareNode struct
  InviterName string // name of the user who sent the Invitation
  EncDecodeKey []byte // Use user's public key to encrypt the random generated 16-byte symmetric key
  EncInviteeName []byte // Use decode key to encrypt the Invitee Name
  EncFileRoot []byte // Use decode key to encrypt the FileRoot
  EncHMACKeyRoot []byte // Use user's public key to encrypt the HMACKeyRoot, which is a random generated 16-byte slice
  EncHMACInvKey []byte // Use user's public key to encrypt the HMACInvKey, which is a random generated 16-byte slice
  HMAC []byte // Take HMACInvKey as key to generate HMAC of fields in this struct
}
type FileMeta struct{
  MyUUID uuid.UUID // UUID generated from fileroot + number for this file
```

```
  EncContent []byte // Encrypted content of the file using dec
ode key
  HMAC []byte // Take HMACKeyRoot + "FileMeta" as key to gener
ate HMAC of fields in this struct
}
type FileCount struct{
  MyUUID uuid.UUID // UUID generated from fileroot + "FileCoun
t"
  Num int // number of FileMeta's of the file.
  HMAC []byte // Take HMACKeyRoot + "FileCount" as key to gene
rate HMAC of fields in this struct
}
```

User Authentication

`InitUser`: During user registration, a user struct is created using the provided information, and a deterministic UUID is generated by hashing the username and taking the first 16 bytes of the result. This UUID is used to identify the user struct in the datastore. Additionally, a random salt is generated for each user, which is used to encrypt the password. The password hash is generated using PBKDF with the salt, and a 16-byte key is derived from the hashed password.
To enable encryption, a pair of RSA asymmetric keys is generated using the key derived from the hashed password. The public key is stored in a keystore with the username as the key. The private key is encrypted using SymEnc with a randomly generated 16-byte initialization vector (IV), and the encrypted key is stored in the user struct.
To enable signature verification, a pair of signature keys is generated using DSKeyGen, and a signature is generated using DSSign with the private signature key and all the information in the user struct. This signature is stored in the user struct as "VerifiedSig," and the signature public key is stored in the keystore with the username as the key.
We also store the raw_password in the struct as a private field. The user struct is then serialized using json.Marshal and stored in the datastore with the UUID as the key.

`GetUser`: During login, the UUID is computed from the provided username, and the corresponding user object is retrieved from the datastore. The signature public key is retrieved from the keystore using the username as the key. The user struct is unmarshalled from the retrieved object, and the UUID is verified against the one stored in the user struct. DSVerify is called with the signature public key to ensure the integrity of the user struct. The password hash is computed using PBKDF with the salt stored in the user struct, and it is verified against the hash password also stored in the user struct. If both checks pass, the user is authenticated and granted access to the system.

File Storage and Retrieval

`StoreFile`: When a user stores a file calling StoreFile, we will create 4 structs: ShareNode, Invitation, FileMeta, FileCount.

- First, we create and initialize the ShareNode that is unique for every user/file pair: 1. Generate a UUID for the ShareNode based on the hash of the username and filename. 2. Initialize the ChildrenList to an empty slice. 3. Generate a UUID for the associated Invitation. 4. Set the InvitationPtr to the generated UUID. 5. Set the IsOwner flag to true. 6. Serialize the ChildrenList to JSON format. 7. Combine all fields into a string to be used for signature. 7. Generate a signature using the user's DSA private key and the combined fields. 8. Serialize the ShareNode struct to JSON format. 9.Store the ShareNode struct in the datastore using the generated UUID as the key.
- Secondly, we create and initialize the Invitation which is also unique for every user/file pair: 1. Set the SharePtr to the UUID of the associated ShareNode. 2. Set the InviterName to the username. 2. Generate a random decode key. 3. Encrypt the decode key using user's public key. 4. Encrypt user's name using the decode key. 5. Generate a random FileRoot and encrypt it using user's public key. 6. Generate a random HMACKeyRoot and encrypt it using the decode key. 7. Generate a random HMACInvKey and encrypt it using the decode key. 8. Calculate the HMAC of the Invitation using the HMACInvKey. 9. Store the encrypted Invitation struct in the datastore using the generated UUID as the key.
- Then, we create and initialize the FileMeta: 1. Generate a UUID for the FileMeta based on the hash of the FileRoot and a number initialized to 0. 2. Encrypt the content of the file using the decode key. 3. Calculate the HMAC of the FileMeta using the HMACKeyRoot and the string "FileMeta". 4. Store the encrypted FileMeta struct in the datastore using the generated UUID as the key.

- Lastly, we create and initialize the FileCount: 1. Generate a UUID for the FileCount based on the hash of the FileRoot and string “FileCount”. 2. Set the Num field to 0. 3. Calculate the HMAC of the FileCount using the HMACKeyRoot and the string "FileCount". 4. Store the encrypted FileCount struct in the datastore using the generated UUID as the key.

`AppendToFile`: The `AppendToFile` function is responsible for appending the given content to an existing file with the given filename. The function starts by retrieving the ShareNode struct from the Datastore using the username and filename. Then, it verifies the ShareNode to ensure that the user has access to the file. It also retrieves the Invitation struct from the Datastore using the ShareNode's InvitationPtr, and verifies that the invitation is valid and not tampered with.
Next, the function decrypts the encoded FileRoot using the user's private key and retrieves the FileCount struct from the Datastore using the decrypted FileRoot. It verifies the FileCount, and then iterates through all the FileMeta structs in the file to verify them as well. After verifying the ShareNode, Invitation, FileCount, and FileMeta structs, the function creates a new FileMeta struct and initializes it. It decrypts the encoded DecodeKey from the Invitation using the user's private key and uses it to symmetrically encrypt the given content. It then generates a new UUID for the FileMeta struct, calculates its HMAC, and updates the FileCount struct to indicate that a new file has been added. Finally, it calculates the HMAC for the updated FileCount struct and stores both the updated FileMeta and FileCount structs in the Datastore.

`LoadFile`: The `LoadFile` function takes in a filename and attempts to load the file content from the datastore for a given user. The function first retrieves the ShareNode struct for the file and verifies it for tampering. It then retrieves the Invitation struct associated with the ShareNode and verifies it as well. The function then retrieves the FileCount struct from the datastore, decrypts the file root with the user's private key and verifies the FileCount for tampering. It uses the decode key obtained from the Invitation to retrieve and decrypt each of the file's FileMeta structs one-by-one from the datastore, and appends the decrypted content of each file to a combined byte array.
Finally, the function returns the concatenated byte array containing the decrypted content of all the file's FileMeta structs. If any of the verification steps fail, the function returns an error indicating the specific issue.

File Sharing and Revocation

`CreateInvitation`: The function first attempts to load the specified file and returns an error if it got revoked. It then retrieves the `ShareNode` struct from the datastore based on the username of the current user and the filename. The function then verifies the integrity of the retrieved `ShareNode` using the `verify_shareNode` helper function. If the verification fails, the function returns an error. Next, the function retrieves the invitation from the datastore based on the `InvitationPtr` field of the retrieved `ShareNode`. The function then verifies the integrity of the retrieved invitation using the `verify_invitation` method of the current user. The function then creates a new `Invitation` struct and populates its fields with the appropriate values encrypted by the decode key or the public key of the recipient user. The RSA encrypted decode key is assigned to the `EncDecodeKey` field of the new `Invitation` struct.

`AcceptInvitation`: The function first checks whether the filename already exists in user's space by checking the UUID generated from the hash of the filename concatenated with the user's username against the data store. If the file exists, an error is returned. Next, the function retrieves the invitation from the data store using the UUID provided as an argument. If the invitation does not exist, an error is returned. The function then attempts to verify the invitation by calling the `verify_invitation` method on the **`User`** struct. If the invitation is not valid due to tampering, an error is returned. Next, the function attempts to decode the encrypted decode key in the invitation using the user's private key. The function then retrieves the `ShareNode` from the data store using the UUID provided in the invitation. The function then attempts to verify the `ShareNode` by calling the `verify_shareNode` function. If there is an error, it is returned. The function then creates a new `ShareNode` struct with the provided filename and the `MyUUID` field set to the UUID generated from the hash of the filename concatenated with the user's username. The `ChildrenList`, `InvitationPtr`, and `IsOwner` fields are initialized as an empty list, the invitation UUID provided, and `false`, respectively. The function then signs the fields in the new `ShareNode` using the user's DSA key and stores the new `ShareNode` in the data store. The function then updates the invitation with the UUID of the new `ShareNode` and generates a new HMAC key to encrypt the invitation's `EncHMACInvKey` field. The function then calculates the HMAC of the invitation and store it into datastore using the `calculate_inv_hmac` function and sets the `Signature` field of the invitation to the calculated HMAC. Finally, the function

attempts to load the file using the `LoadFile` to check whether the invitation is revoked. If it is, the newly created `ShareNode` is deleted from the data store and the error is returned.

`RevokeAccess`: The function first retrieves the ShareNode struct associated with the file from the datastore and verifies its authenticity. It checks if the user is the owner of the file and retrieves the invitation associated with the ShareNode. The function then decrypts the invitation using the user's private key and verifies its authenticity. The function proceeds to decrypt the invitation's encoded decode key and file root using the user's private key. It then retrieves the FileCount struct associated with the file from the datastore, decrypts it using the decoded key, and verifies its authenticity. The function then goes through all the FileMeta structs associated with the file and updates them using a new decoded key, file root, and HMAC keys. It verifies the authenticity of each FileMeta struct before updating it. The function then deletes the revoked user’s invitation from the children list and delete the FileCount and FileMeta structs associated with the revoked invitation using the helper function `delete_all_structs`. Finally, the function updates all the invitations left in the list and their children with the new keys using the helper function `update_sharenode_childrenlist`.

① Alice init User

DataStore

Alice
User struct

Keystore
RSA.PK (Alice)
DSA.PK (Alice)

② Bob init User

DataStore

Alice
User struct

Bob
User struct

Keystore
RSA.PK (Alice)
DSA.PK (Alice)
RSA.PK (Bob)
DSA.PK (Bob)

③ Alice stores file "foo" with content "1"

DataStore

Alice
User struct

Bob
User struct

Alice/foo
ShareNode

Alice/foo→Alice
Invitation

Alice/foo/0
Filename
"1"

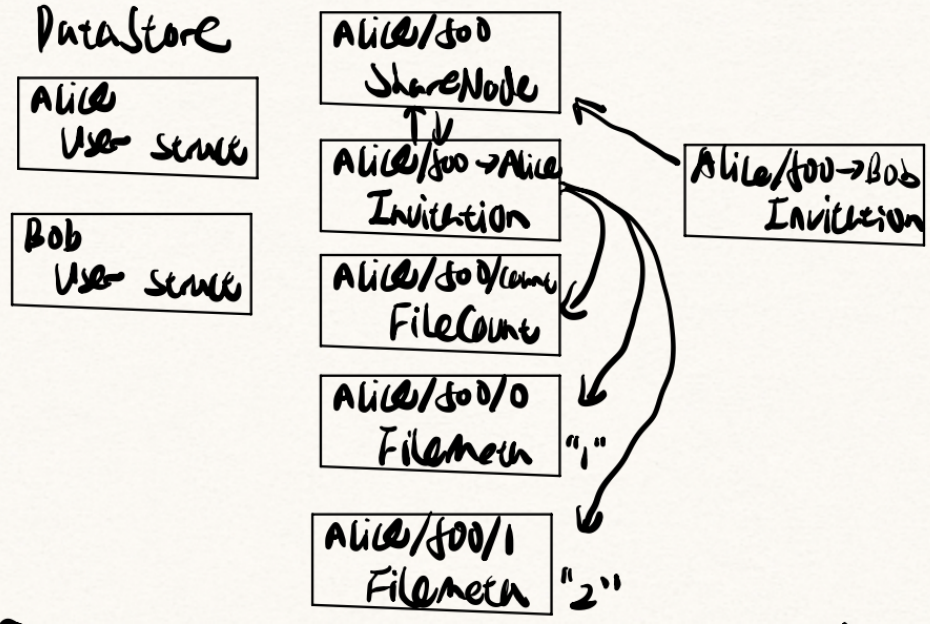
Alice/foo/count
FileCount
num=0 → num=1

Keystore
RSA.PK (Alice)
DSA.PK (Alice)
RSA.PK (Bob)
DSA.PK (Bob)

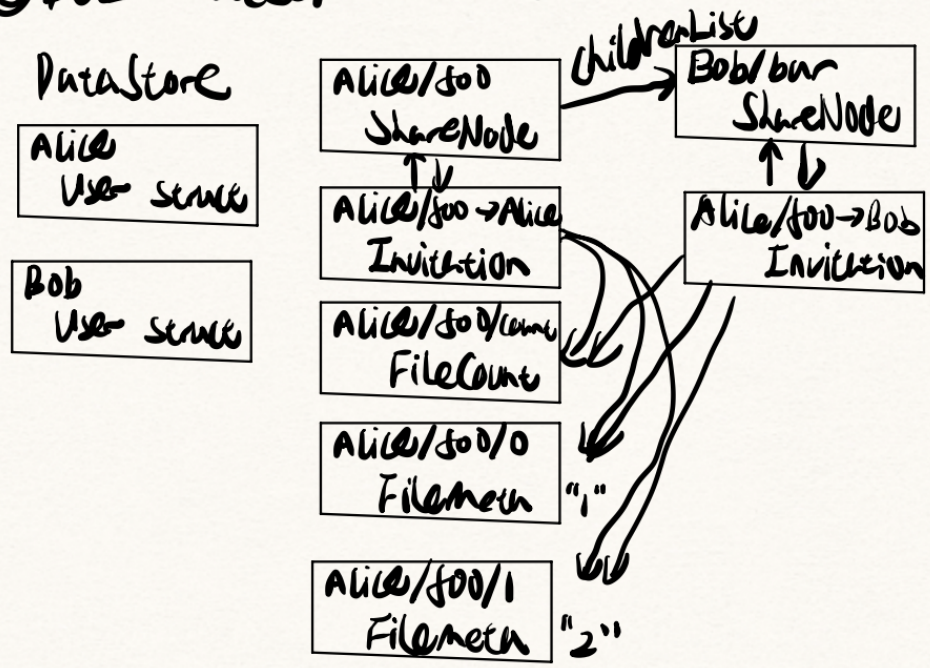
④ Alice appends file foo with 2

Alice/foo/1
Filename
"2"

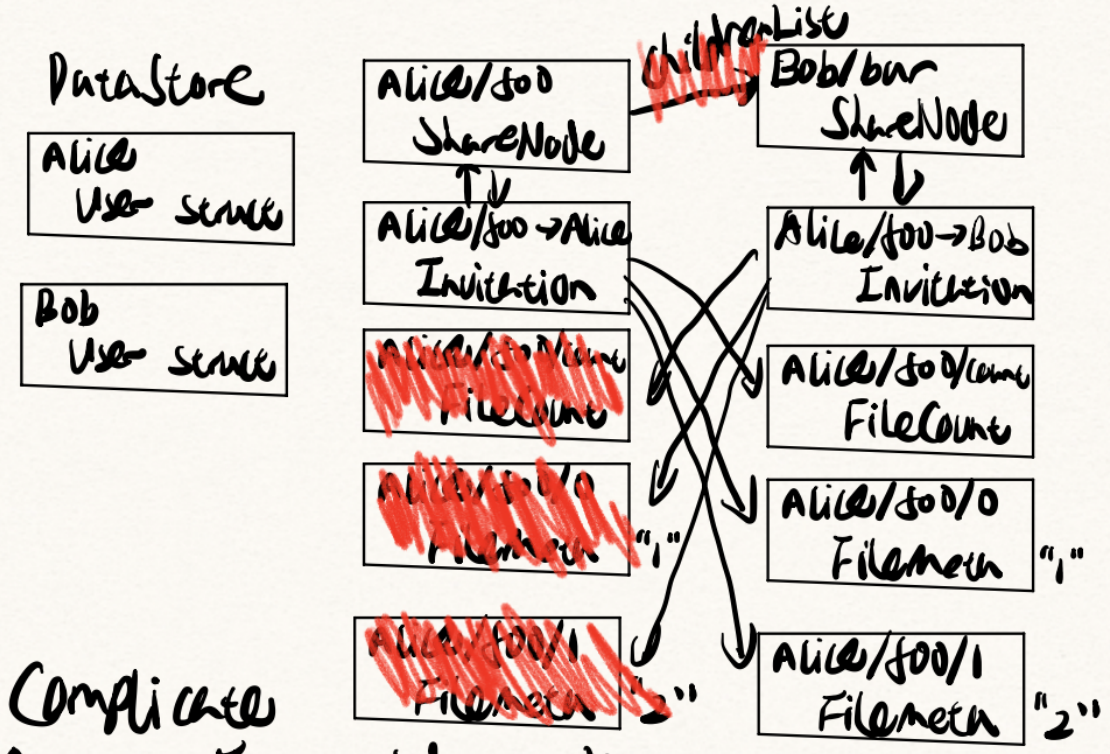
⑤ Alice shares file "foo" with Bob



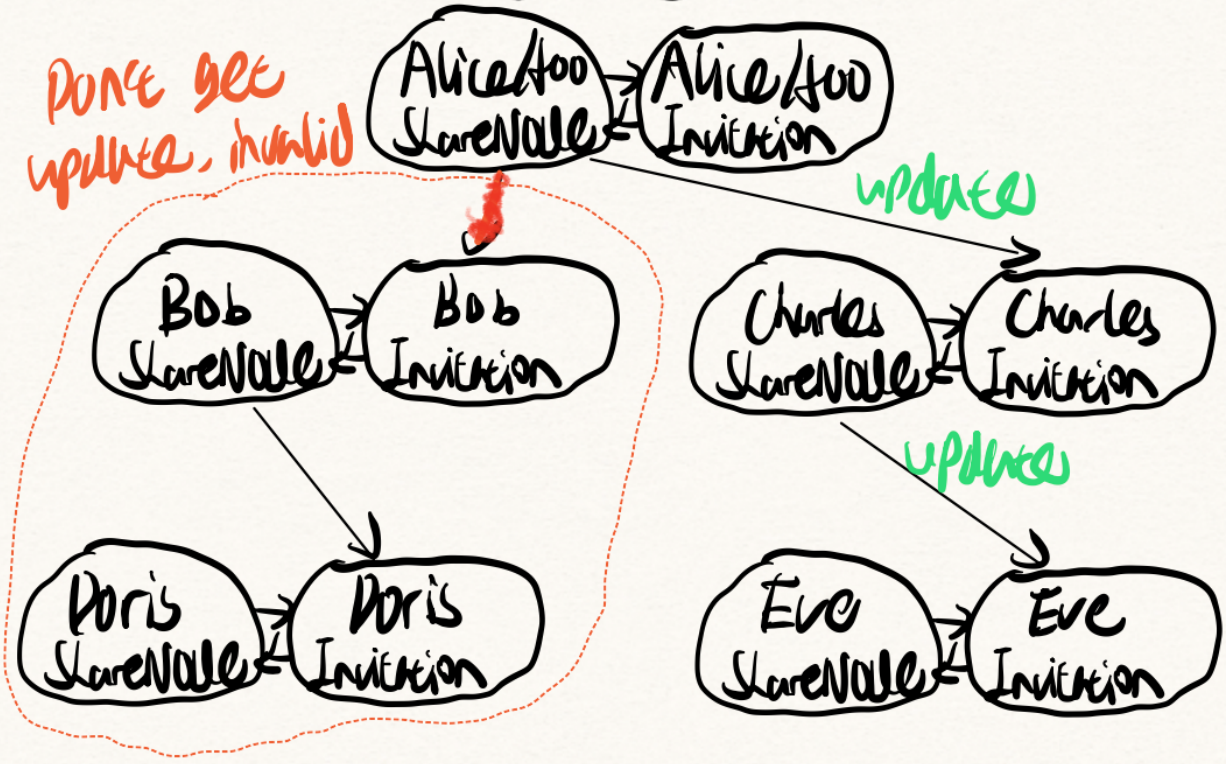
⑥ Bob accepts the invitation, and name it "bar"



7. Alice Revoke Bob's Access



Complicated Revoke Tree: Using DFS, Revoke Bob



Helper Methods

- func **`find_revoke_uuid`**(childrenList []uuid.UUID, recipientUsername string, decode_key []byte) uuid.UUID

The function loops over the `childrenList` and for each UUID in the list, it retrieves an invitation associated with it from the datastore using `DatastoreGet` function. If an invitation is not found, it returns a nil UUID. The retrieved invitation is then unmarshalled from bytes to an `Invitation` struct, and the encrypted invitee name is decrypted using `SymDec` function from the `userlib` library with the `decode_key` provided. If the decrypted invitee name matches the `recipientUsername`, the function returns the UUID of the invitation. If no match is found after iterating through all the UUIDs in the `childrenList`, the function returns a nil UUID.

- func (userdata *User)**`update_sharenode_childrenlist`**(shareNode *ShareNode, child_uuid uuid.UUID) error

First, the function loops through the children list of the ShareNode and removes the child with the given UUID. Then, the function marshals the updated children list into a byte array, combines it with other fields from the ShareNode (including its UUID, invitation pointer, and owner status), signs the combined fields using the user's DSA private key, and stores the updated ShareNode in the datastore. If successful, the function returns nil. If there is an error during marshaling or signing, it returns the error.

- func **`delete_all_structs`** (decode_key []byte, file_root []byte) error

First, the function retrieves the `FileCount` struct from the datastore using a UUID derived from the file root and the string "FileCount". If the `FileCount` struct does not exist, an error is returned. The function then iterates through the number of files stored in the `FileCount` struct and retrieves each corresponding `FileMeta` struct from the datastore using a UUID derived from the file root and the file number. If a `FileMeta` struct does not exist, an error is returned. The function then deletes each `FileMeta` struct from the datastore. Finally, the `FileCount` struct is deleted from the datastore.

- func **`update_invitation`**(invitation *Invitation, new_decode_key []byte, new_file_root []byte, new_hmac_key_root []byte, decode_key []byte, shareNode_uuid uuid.UUID) error

This function is used to update an invitation object with new keys and recursively update the invitations of its children in a hierarchical structure. First, the function

decrypts the invitee's name from the invitation object and then updates the invitation object with the new keys and UUIDs.
Next, the function retrieves the ShareNode object associated with the invitation and verifies its authenticity. If the ShareNode is valid, the function recursively updates the invitations of its children. The function retrieves each child invitation and verifies its authenticity by calling the `rev_verify_invitation` function. If the child invitation is valid, the function calls itself on the child invitation object to update it recursively. The function returns an error if any of the necessary keys or UUIDs are missing or if any of the invitations are not valid due to tampering. If the ShareNode UUID provided in the parameters matches the one in the invitation object, the function does not update the ShareNode object, because it indicates that the the user hasn't accept the invitation yet, thus the new ShareNode struct is not created.

- func **`rev_verify_filecount`**(fileCount *FileCount, fileCount_uuid uuid.UUID, invitation *Invitation, decode_key []byte) error

This function is only used in revoke function to check the authenticity of the fileCount struct. It first checks if the UUID of the `FileCount` struct matches the `fileCount_uuid` input parameter. If it does not match, an error is returned. It then calculates the `keyroot` by decrypting the `EncHMACKeyRoot` field of the `Invitation` struct using the `decode_key`. Using the `keyroot`, the function appends the string "FileCount" to get the `filecount_hmackey` which is used to calculate the HMAC of the `FileCount` struct. It compares the calculated HMAC with the HMAC stored in the `HMAC` field of the `FileCount` struct. If they do not match, an error is returned indicating that the `FileCount` struct has been tampered with. If the `FileCount` struct passes all the verification checks, the function returns `nil`. Otherwise, an error is returned.

- func **`rev_verify_filemeta`**(fileMeta *FileMeta, fileMeta_uuid uuid.UUID, invitation *Invitation, decode_key []byte) error

This function is only used in revoke function to check the authenticity of the fileMeta struct. It first checks if the UUID of the `FileMeta` struct matches the `fileMeta_uuid` input parameter. If it does not match, an error is returned. It then calculates the `keyroot` by decrypting the `EncHMACKeyRoot` field of the `Invitation` struct using the `decode_key`. Using the `keyroot`, the function appends the string "FileMeta" to get the `filemeta_hmackey` which is used to calculate the HMAC of the `FileMeta` struct. It compares the calculated HMAC with the HMAC stored in the `HMAC` field of the `FileMeta` struct. If they do not match, an error is returned indicating that the `FileMeta` struct has been tampered with. If the `FileMeta` struct

passes all the verification checks, the function returns `nil`. Otherwise, an error is returned.

- func **`rev_verify_invitation`**(invitation *Invitation, inv_uuid uuid.UUID, decode_key []byte) bool

This function is only used in revoke function to check the authenticity of the invitation struct. First, the function checks whether the UUID of the invitation object matches the given UUID. If they don't match, the function returns false indicating that the invitation is invalid. Then, the function decrypts the encrypted HMAC key of the invitation object using the decoding key. If the HMAC key cannot be decrypted, the function returns false indicating that the invitation is invalid.
Next, the function calculates the HMAC of the invitation object using the HMAC key and a concatenation of various fields of the invitation object such as its UUID, share pointer, inviter name, invitee name, file root, and various encrypted fields. If there is an error while calculating the HMAC, the function returns false indicating that the invitation is invalid. Finally, the function compares the calculated HMAC with the HMAC value stored in the invitation object. If the two HMAC values do not match, the function returns false indicating that the invitation is invalid. Otherwise, the function returns true indicating that the invitation is valid.

- func **`calculate_inv_hmac`**(invitation *Invitation, inv_uuid uuid.UUID, hmacinvkey []byte) bool

The purpose of the function is to calculate the HMAC of the `Invitation` struct using the provided HMAC key and update the `Invitation` in the datastore with the new HMAC value. To calculate the HMAC, the function first marshals the `SharePtr` field of the `Invitation` struct into a byte array. It then concatenates all the relevant fields of the `Invitation` struct (including the marshaled `SharePtr`) into a single string. This string is then passed along with the HMAC key to the `userlib.HMACEval` function, which returns the calculated HMAC value. If any errors occur during the process of calculating the HMAC or updating the `Invitation` in the datastore, the function returns `false`. Otherwise, the function sets the `HMAC` field of the `Invitation` struct to the newly calculated HMAC value, marshals the updated `Invitation` struct into a byte array, and stores it in the datastore with the provided UUID. Finally, the function returns `true` to indicate that the operation was successful.

- func **calculate_filemeta_hmac**(filemeta *FileMeta, filemeta_uuid uuid.UUID, hmac_key []byte) bool

It calculates the HMAC of the `FileMeta` by concatenating the `MyUUID` field and the `EncContent` field (which is a byte slice representing the encrypted content of the

file) and computing the HMAC. The calculated HMAC is then updated in the `FileMeta`. Finally, the updated `FileMeta` is stored in the data store using the `DatastoreSet` function. The function returns `true` if all operations are successful, otherwise `false`. This function helps ensure the integrity of the `FileMeta` data by verifying that it has not been tampered with.

- func **calculate_filecount_hmac**(filecount *FileCount, filecount_uuid uuid.UUID, hmac_key []byte) bool

It calculates the HMAC of the `FileCount` struct using the provided HMAC key and updates the HMAC field in the struct. It then marshals the updated `FileCount` struct into bytes and stores it in the datastore using the provided UUID. If any errors occur during the process, the function returns false. Overall, this function is responsible for ensuring the integrity of the `FileCount` struct by calculating and updating its HMAC before storing it in the datastore.

- func **verify_shareNode**(username string, shareNode *ShareNode, shareNode_uuid uuid.UUID) error

The function first checks whether the `MyUUID` field of the `ShareNode` struct matches the provided `shareNode_uuid`. If the UUIDs do not match, an error is returned. Next, the function retrieves the public key of the user specified by `username` from the keystore. If the public key cannot be found, an error is returned. Then, the function combines all the fields of the `ShareNode` struct into a single byte array, `fields_combined`. This includes the UUID, the `ChildrenList` field (which contains a list of UUIDs representing the child nodes of this `ShareNode`), the UUID of the `Invitation` object associated with this `ShareNode`, and a boolean `IsOwner` flag indicating whether the user is the owner of the shared file. Finally, the function uses the `DSVerify` function from the `userlib` library to verify the signature of the `ShareNode` using the user's public key and the `fields_combined` byte array. If the signature is invalid, an error is returned. If the signature is valid, the function returns `nil`.

- func (userdata *User) **verify_invitation**(invitation *Invitation, invitation_uuid uuid.UUID) bool

First, the function checks if the `MyUUID` field of the `Invitation` matches the `invitation_uuid` passed as an argument. If they do not match, the function returns false. Then, the function retrieves the private key of the user from the `User` struct, and decrypts the `EncHMACInvKey` field of the `Invitation` struct using the `EncDecodeKey` field. Next, the function combines all the fields in the `Invitation` struct and calculates the HMAC using the decrypted `hmacInvKey`. The calculated HMAC is compared to the `HMAC` field of the `Invitation` struct, and if they do not

match, the function returns false. If all the checks pass, the function returns true, indicating that the invitation is valid.

- func (userdata *User) **verify_filemeta**(fileMeta *FileMeta, fileMeta_uuid uuid.UUID, invitation *Invitation) error

The function verifies the authenticity of the fileMeta by decrypting `EncHMACKeyRoot` in the `Invitation` using the user's private key and then deriving the HMAC key for the `FileMeta`. The function then calculates the HMAC of the fileMeta using this key and compares it with the HMAC stored in the fileMeta. If the HMACs match, the function returns nil indicating the fileMeta is authentic. If there is an error during the verification process, an error message is returned. If the `MyUUID` field in the `FileMeta` does not match with the `fileMeta_uuid` argument, the function returns an error indicating the UUIDs don't match.

- func (userdata *User) **verify_filecount**(fileCount *FileCount, fileCount_uuid uuid.UUID, invitation *Invitation) bool

First, the function checks if the `MyUUID` field of the `FileCount` struct matches the `fileCount_uuid` input parameter. If not, it returns `false`. Next, the function retrieves the private key of the user and uses it to decrypt the `EncHMACKeyRoot` field of the invitation using `PKEDec`. The resulting plaintext is then used to decrypt `hmacKeyRoot` using `SymDec`. The function then generates an HMAC key for the `FileCount` struct by appending the string "FileCount" to `hmacKeyRoot` and taking the first 16 bytes of the resulting hash. It then computes the HMAC of the `FileCount` struct by concatenating its `MyUUID` field and `Num` field converted to a string, and passing the result and the HMAC key to `HMACEval`. Finally, the function returns `true` if the computed HMAC matches the `HMAC` field of the `FileCount` struct, and `false` otherwise.

- func (userdata *User) **`get_privateKey`**() userlib.PKEDecKey

The function `get_privateKey` is a method of the `User` struct and it returns the private key of the user for use in encryption and decryption operations. It does this by first generating a symmetric encryption key derived from the user's raw password using a hash-based key derivation function. This key is then used to decrypt the encrypted form of the user's private key, which is stored in the `EncPrivateKey` field of the `User` struct. The decrypted private key is then unmarshalled from its JSON representation into a `PKEDecKey` struct and returned. If any error occurs during the process, an empty `PKEDecKey` struct is returned.

- func (userdata *User) **get_dsaKey**() userlib.DSSignKey

The function `get_dsaKey()` is a method of the `User` struct. It returns the DSSignKey of the user, which is used to sign messages. The function first derives a key from the

user's password using a key derivation function (KDF) and the string "EncSign". This key is used to symmetrically decrypt the user's encrypted DSSignKey stored in the `EncSignKey` field of the `User` struct. The decrypted DSSignKey is then unmarshaled from its JSON representation and returned. If any error occurs during the process, an empty DSSignKey is returned.