

# An End-to-End Encrypted File Sharing System

## Data Structures

```
type User struct {
    Username string
    Password string
    OwnedFileMap map[string]userlib.UUID //Map of file name to file pointer only holds owned files
    SentSharedInvitation map[string]userlib.UUID //Map of file name+username to invitations we have sent
    AcceptedSharedInvitations map[string]userlib.UUID // Map of file name to invitiation only files not owned
}
type AuthenticatedEncItem struct {
    Ciphertext []byte
    Tag []byte
}
type FileBlob struct { // works like a linked list
    Content []byte
    NextContentPointer userlib.UUID
}
type FilePointer struct {
    FrontPointer userlib.UUID //pointer to first FileBlob linkedList
    EndPointer userlib.UUID //pointer to last FileBlob linkedList
}
type Invitation struct {
    Owner string
    FilePointerUUID userlib.UUID
    Senders []string
}
```

## User authentication

### InitUser

```
// Generate and store salt. unique for each user
// Derive keys for user encryption
// Encrypt user using symmetric encryption and store user to the datastore
// return a pointer to the user
```

### GetUser

```
// Derive keys for user decryption
// Check mac of User and decrypt
// return a pointer to the user
```

## File Storage and Retrieval

### User.StoreFile

```
// Get the updated user from datastore
// Case 1: filename is not owned; shared by someone else
//This means filename is in AcceptedSharedInvitations map

//Get invitation from datastore
//Derive keys for invitation decryption
// Check mac of invitation and decrypt

//Make new file Blob
//Derive Keys for fileBlob encryption
//Encrypt fileBlob using symmetric encryption and store fileBlob to the datastore

// Get filePointer UUID from the invitation
//Get encrypted filePointer using filePointerUUID from datastore
```

```

//Derive keys for filePointer decryption
// Check mac of filePointer and decrypt

// Update the head and tail of the filePointer's linked list

//Encrypt updated filePointer using symmetric encryption and store updated filePointer to the datastore

// Case 2: filename is owned by the user
// Make a new file Blob
//Derive Keys for fileBlob encryption
//Encrypt fileBlob using symmetric encryption and store fileBlob to the datastore

// Case a: Filename already exists
//This means filename is in ownedFileMap

//Get current encrypted filePointer
//Derive keys for filePointer decryption
// Check mac of filePointer and decrypt

// Update the head and tail of the filePointer's linked list
// Case b: Filename doesn't exists
// File is not in ownedFileMap

// Make a new filePointer struct
// Derive filePointer UUID and add it to ownedFileMap
// Derive keys for filePointer encryption
// Encrypt filePointer using symmetric encryption and store filePointer to the datastore

// Generate keys to encrypt the updated user
// Encrypt updated user using symmetric encryption and store encrypted updated user to the datastore

```

#### User.LoadFile

```

// Get the updated user from datastore
// Case 1: filename is not owned; shared by someone else
//This means filename is in AcceptedSharedInvitations

// Get invitation from datastore
// Derive keys for invitation decryption
// Check mac of invitation and decrypt

// Get filePointer UUID from the invitation
// Get encrypted filePointer using filePointerUUID
//Derive keys for filePointer decryption
// Check mac of filePointer and decrypt

// initialize content_to_return to an empty byte array
// Loop through file blob linked list until we reach the end of linked List
// Get encrypted fileBlob
// Derive Key for fileBlob decryption
// Check mac of fileBlob and decrypt

// Append the current blob's content to content_to_return
// Update the current fileBlobPointer to point to nextContentPointer

// return content_to_return

// Case 2: filename is owned by the user
//This means filename is in OwnedFileMap

// Get current encrypted filePointer
// Derive keys for filePointer decryption
// Check mac of filePointer and decrypt

```

```

// initialize content_to_return to an empty byte array
// Loop through file blob linked list until we reach the end of linked List
// Get encrypted fileBlob
// Derive Key for fileBlob decryption
// Check mac of fileBlob and decrypt

// Append the current blob's content to content_to_return
// Update the current fileBlobPointer to point to nextContentPointer

// return content_to_return

```

## User.AppendToFile

```

// Get updated user from datastore
// Case 1: filename is not owned; shared by someone else
//This means filename is in AcceptedSharedInvitations

// Get invitation from datastore
// Derive keys for invitation decryption
// Check mac of invitation and decrypt

// Get filePointer UUID from the invitation
// Get encrypted filePointer using filePointerUUID
// Derive keys for filePointer decryption
// Check mac of filePointer and decrypt

// Make a new file blob
// Derive Keys for fileBlob encryption
// Encrypt fileBlob using symmetric encryption
// Store fileBlob to the datastore

// Update the tail of the filePointer's linked list

// Encrypt the updated filePointer and store the updated filePointer to the datastore
// Case 2: filename is owned
//This means filename is in OwnedFileMap

//Get current encrypted filePointer
// Derive keys for filePointer decryption
// Check mac of filePointer and decrypt

// Make a new file blob
//Derive Keys for fileBlob encryption
//Encrypt fileBlob using symmetric encryption
//Store fileBlob to the datastore

// Update the tail of the filePointer's linked list

// Encrypt the updated filePointer and store the updated filePointer to the datastore

```

## File Sharing and Revocation

### User.CreateInvitation

```

// Error if recipient doesn't exist

// Get the updated user from datastore
// Case 1: filename is not owned; shared by someone else
//This means filename is in AcceptedSharedInvitations

// Get encrypted invitation from datastore
//if invitation does not exist

```

```

        // Delete invitation from user's acceptedSharedInvitations map
        // Generate keys to encrypt the updated user
        // Encrypt the updated user and store encrypted updated user to the datastore
        //exit function

// Get invitation from datastore
// Derive keys for invitation decryption
// Check mac of invitation and decrypt

// Add username to invitation's sender list
// Encrypt updated invitation and store updated invitation

// return updated invitation pointer

// Case 2: filename is owned
//This means filename is in OwnedFileMap

// Make a new sender list with user's username
// Create new invitation struct
//Derive keys for invitation encryption
//Encrypt invitation using symmetric encryption and store invitation to datastore

// Update user's sentSharedInvitation map
// Generate keys to encrypt the updated user
// Encrypt updated user and store encrypted updated user to the datastore

//return new invitation pointer

```

#### User.AcceptInvitation

```

//Get invitation from datastore
// Error if invitation does not exist

// Get the updated user from datastore
// Error if user already has file with the filename in their personal file namespace.
// Derive keys for invitation decryption
// Check mac of invitation and decrypt

//Check that the invitation's sender list has senderUsername to verify that the invitation was created by senderUsername
//error if check fails

//Update user's acceptedSharedInvitation map with new invitation pointer

// Generate keys to encrypt the updated user
// Encrypt updated user using symmetric encryption and store encrypted updated user to the datastore

```

#### User.RevokeAccess

```

// Get the updated user from datastore

//if filename is owned by user
//This means filename is in OwnedFileMap

// Error if recipientUsername does not have filename shared to them

// Get invitation pointer from SentSharedInvitation map
// Delete invitation pointed to by invitation pointer from datastore
//Delete invitation pointer from user's SentSharedInvitation map

// Generate keys to encrypt the updated user
// Encrypt updated user and store encrypted updated user to the datastore

```

## Datastore

Key	Value	Key Derivation
saltUUID	random salt	$\text{saltHash} = \text{Hash}(\text{username} + \text{'salt'})$ $\text{saltUUID} = \text{fromBytes}(\text{saltHash}[:16])$
userUUID	{ symmetrically encrypted user, mac }	$\text{sourceKey} = \text{Argon2Key}(\text{password}, \text{salt}, 16)$ $\text{userUUIDKey} = \text{HashKDF}(\text{sourceKey}, \text{username})$ $\text{userUUID} = \text{fromBytes}(\text{userUUIDKey}[:16])$
filePointerUUID	{ symmetrically encrypted filePointer, mac }	$\text{filePointerUUID} = \text{uuid.New}()$
fileBlobUUID	{ symmetrically encrypted fileBlob, mac }	$\text{fileBlobUUID} = \text{uuid.New}()$
invitationUUID	{ symmetrically encrypted invitation, mac }	$\text{invitationUUID} = \text{uuid.New}()$

## Symmetric Encryption Key Derivation

Authenticate/ Encrypted Item	Key Derivation
User	$\text{sourceKey} = \text{Argon2Key}(\text{password}, \text{salt}, 16)$ $\text{encKey} = \text{HashKDF}(\text{sourceKey}, \text{username} + \text{"encKey"})$ $\text{macKey} = \text{HashKDF}(\text{sourceKey}, \text{username} + \text{"macKey"})$
FilePointer	$\text{sourceKey} = \text{Argon2Key}(\text{filePointerUUID.String()}, \text{salt}, 16)$ $\text{encKey} = \text{HashKDF}(\text{sourceKey}, \text{"encKey"})$ $\text{macKey} = \text{HashKDF}(\text{sourceKey}, \text{"macKey"})$
FileBlob	$\text{sourceKey} = \text{Argon2Key}(\text{fileBlobUUID.String()}, \text{salt}, 16)$ $\text{encKey} = \text{HashKDF}(\text{sourceKey}, \text{"encKey"})$ $\text{macKey} = \text{HashKDF}(\text{sourceKey}, \text{"macKey"})$
Invitation	$\text{sourceKey} = \text{Argon2Key}(\text{invitationUUID.String()}, \text{salt}, 16)$ $\text{encKey} = \text{HashKDF}(\text{sourceKey}, \text{"encKey"})$ $\text{macKey} = \text{HashKDF}(\text{sourceKey}, \text{"macKey"})$