Design Document

Data Structure

```
type User struct {
    Username     string
    UserStoreKey []byte                  // to generate user uuid in datastore
    HmacKey      []byte                  // for user data generate by username
and password
    EncKey       []byte                  // for user data generate by username
and password
    DecKey_PKE   userlib.PKEDecKey       // private Decrypt Key pair to
KeystoreGet(username)= EncKey
    SignKey      userlib.DSSignKey       // private RSA sign Key pair to
KeystoreGet(username+"DS")= VerifyKey
    own          map[string]userlib.UUID // file uuid owned by the user
    FileKeys     map[string]FileKey

    share map[string]InvitationIDPair //invitation received
}

type File struct {
    Owner            string
    FileID           userlib.UUID
    Data             []byte
    AccessUserList   []string
    UserToInvitation map[string]Invitation
}

//FileKey can be used to get complete file
//Own files: can be found in user struct
//Invitated by others: can decrypted by keys in invitation struct
type FileKey struct {
    FileUUID userlib.UUID
    FileEncKey  []byte // to Decrypt file from datastore
    FileHmacKey []byte // to Decrypt file from datastore
}

type InvitationIDPair struct {
    InvitationPtr  userlib.UUID // to get invitation from datastore
    SenderUsername string       // to get sender's RSA public key from keystore
}

//invitation encrypted by sender's RSA private sign key and receiver's EncKey
//invitation decrypted by sender's RSA public verify key and receiver's
DecKey_PKE
type Invitation struct {
    MBEncKey      []byte
    MBKeyFileUUID    userlib.UUID
    MBFileHmacKey []byte
}
```

User Authentication

1. For the user to initiate and login: Use username and password to generate UserStoreKey and then get random bytes from this key to set UUID for this user struct. Use EncKey and HmacKey, which are also generated by username and password, to encrypt the user struct and store it in Datastore. So, when the uswantsant to login in, we can use UUID to download the user struct bytes from DataStore and decrypt it.
2. For file sharing between users: When an invitation is created, the receiver's Public Encrypt Key of PKE will be used to encrypt this invitation so that only the receiver can decrypt this invitation by using its Private Decrypt Key. Invitation Receivers can use the Public Verify Key which can be found in KeyStore to verify the sender of the invitation.

File Storage and Retrieval

To store a new file, the user creates fileEncryptKey and fileHmacKey from random bytes to encrypt and add tag to the file bytes. And store the file with an uuid in DataStore. These Keys will be stored in FileKeys within the user struct.

To retrieve the file, we need to get the UUID in the user's own files' filename or from others' invitations. Then download the file from DataStore. Get EncKey and HmacKey from the user struct or invitation to decrypt the file.

My design does not support efficient file append yet.

File Sharing and Revocation

To share files with another user, the user sends an invitation to the receiver. The sender needs to encrypt this file with the Public Encrypt Key from the receiver in KeyStore and sign the file bytes with his Private Sign Key. So only the receiver can decrypt it with his own key and verify the sender's signature by public verify key from KeyStore.

To revoke a different user's access to a file, we change the File Key struct in Datastore and use new File Keys to encrypt the file and update the file. So the revoked user cannot access this file by previous fileKeys. Since we changed FileKeys we need to update all other invitations by AccessUserList in the file struct and update these invitations's FileKey.

Helper Methods

```
//************************** self defined functions
*******************************//
//Encrypt struct to Bytes: SymEnc and add Hmac tag ; PKE Enc and RSA Sign
func (userdata *User) EncryptFile(fileEncryptKey []byte, fileHmacKey []byte,
file_struct File) (EncryptedFile []byte, err error) {...}
func (userdata *User) EncryptMB(fileEncryptKey []byte, fileHmacKey []byte,
file_struct FileKey) (EncryptedFile []byte, err error) {...}
func (userdata *User) EncryptInvitation(recipientUsername string,
invitation_struct Invitation) (EncryptedInvitationBytes []byte, err error) {...}

//Decrypt file Bytes to struct
func (userdata *User) DecryptFile(fileEncryptKey []byte, fileHmacKey []byte,
EncryptedFileBytes []byte) (file_struct File, err error) {...}
func (userdata *User) DecryptMB(fileEncryptKey []byte, fileHmacKey []byte,
EncryptedFileBytes []byte) (file_struct FileKey, err error) {...}
```

```
func (userdata *User) DecryptInvitation(senderUsername string,
SignedEncryptedInvitationBytes []byte) (invitation_struct Invitation, err error)
{...}

//update user
func (userdata *User) UpdateUser() (err error) {...}
```

Test Proposal

Test1 - Design Requirement: The client MUST ensure confidentiality and integrity of file contents and file sharing invitations.

1. Create two users and create a file owned by one user and send invitation to another user
2. Define ChangeDataStore(), a method to change data in DataStore by any uuid
3. No matter changed FileKey, Invitation, File, or user struct. Expect errors triggered when we try to reload files, invitations, or users again.

Test2 - Design Requirement: The Client MUST allow users to efficiently append new content to previously stored file.

1. Create one user who owns two files, one of the size is 1GB, and the other size is 10Bytes.
2. Define CalculateAppendTime(), a method to calculate the time to append 10 Bytes to these two files.
3. Expect there is no significant time difference between them.

Test3 - Design Requirement: All file changes MUST be reflected in all current user sessions immediately.

1. Initialize two users A and B with the same password and username locally from DataStore.
2. Store a file by user A's method.
3. Expect to download the same file by user B's loadfile method.

Test4 - Design Requirement: Changes to the content of a file MUST be accessible by all users who are authorized to access the file.

1. Create two users A and B. Create a file owned by A and send an invitation to B. B accepts the invitation.
2. Change the content of the file to a new file.
3. Download the new file by B's loadfile method and expect the file is the same with the new one.

Test5 - Design Requirement: The client MUST prevent any revoked user from using the client API to take any action on the file.

1. Create two users A and B. Create a file owned by A and send an invitation to B. B accepts the invitation.
2. A revokes B's access to the file.
3. Expect B cannot load or modify the file.

Test6 - Design Requirement: When the owner revokes a user’s access, the client MUST enforce that any other users with whom the revoked user previously shared the file also lose access.

1. Create three users A and B. Create a file owned by A and send an invitation to B. B accepts the invitation.
2. B send the file invitation to C, and C accepts the invitation.
3. A revokes B's access to the file.
4. Expect C cannot load or modify the file.