CS161 Project 2 Design Document
Groupmates: Ko Tsun Leung(3037588055), David Goldstein(3034119326)

1 Data Structures

a. User
   i. User Struct
      1. Username: username string from user input
      2. Salt: 64-bit random, non-secret nonce generated by RandomBytes(8)
      3. PasswordKey: Argon2Key(password: password, salt: Salt, keyLen: 32)
      4. UserID: uuid.FromBytes(Hash(username)[:16]))
      5. privKey1(private var): RSA Sign private key by DSKeyGen() - use for encrypting file certificate
      6. privKey2(private var): RSA private key by PKEKeyGen() - use for invitation signature
      7. HMACUser: HMACEval(key: password, msg: username)
      8. CreatedFiles: map<string, File>
      9. SharedFiles: map<string, File>
b. File
   i. File Struct
      1. Filename(private var): string
      2. Content: []bytes
      3. macKey: RandomBytes(16)
      4. Creator: string
      5. FileID: uuid.FromBytes(Hash(username||filename)[:16]))
   ii. FilePackage Struct
      1. FileJSON: []bytes
      2. HMACFile
c. Invitation
   i. Invitation Struct
      1. CreatorID: the UserID of the invitation creator
      2. AcceptorID: the UserID of the invitation acceptor
      3. FileID: the fileID of the invitation
      4. isAccepted: boolean
   ii. InvitationPackage Struct
      1. InvitationFile: JSON Serialized String of Invitation Instance
      2. Sign: RSA Digital Signature with the message and signing key

2 User Authentication

a. Define a helper function SendUserTostore(userdata User) which converts userdata into JSON string and encrypts using first half of PasswordKey, then calls DataStoreSet with UserID as the storage key.
   i. Returns an error if JSON serialization has problem {JSON serialization error}
b. InitUser(username string, password string)
   i. Returns an error if an empty username is provided{empty username provided}
   ii. Create a new User instance - userdata
   iii. userdata.username field is set to be username
   iv. Userdata.salt field is set to be RandomBytes(8) (it is a 64 bit random nonce)
   v. userdata.PasswordKey field is a 32 bit cryptographic key generated by Argon2Key(password: password, salt: salt, keyLen: 32)
   vi. userdata.UserID field is set to be uuid.FromBytes(Hash(username)[:16]))

        vii. Returns an error if datastore already includes UserID as storage key{user already exists}
        viii. userdata.HMACUser is set to be HMACEval(key: password, msg: username)
        ix. Generate privKey1, PubKey1 by PKEKeyGen()
        x. Generate privKey2, PubKey2 by PKEKeyGen()
        xi. Call KeystoreSet(userdata.UserID, PubKey1)
        xii. Call KeystoreSet(userdata.UserID, PubKey2)
        xiii. Populate userdata with name, passwordKey, salt, privKey1, privKey2, HMACUser, shared, owner
        xiv. Call SendUserToStore with the created user struct
        xv. If the above process is successful without error, return the created userdata

    c. GetUser(username string, password string)
        i. Compute UserID by uuid.FromBytes(Hash(username)[:16]))
        ii. Set userdata(Type: User), success = DatastoreGet(UserID)
        iii. Returns an error if success returns false {User is not initialized}
        iv. Returns an error if the above instruction raise error {Error on retrieving user from database}
        v. Verify if HMACEqual( HMAC(password, username), userdata.HMACUser) is true, otherwise returns an error {integrity of user struct has been compromised}
        vi. Compute 32-bit localPasswordKey by Argon2Key(password: password, salt: userdata.salt, keyLen: 32)
        vii. Verify if HMACEqual( Hash(localPasswordKey), Hash(userdata.PasswordKey)) is true, otherwise returns an error {User credentials are invalid}
        viii. Create a local variable called SessionID by uuid.fromByte(HMAC(key: RandomBytes(16), msg: length of ActiveSessionIDs + 1)[:16])
        ix. Append ActiveSessionIDs array with the created local SessionID
        x. Populate updated userdata with name, passwordKey, salt, PrivKey1, PrivKey2, ActiveSessionIDs
        xi. Call SendToDatastore with the updated user struct
        xii. If the above process is successful without error, return the pointer to updated userdata, otherwise return error

    d. Datastore
        i. [UserID(uuid.FromBytes(Hash(username)[:16])))]: (User struct with name, passwordKey, salt, PrivKey1, PrivKey2, ActiveSessionIDs)
        ii. [FileID(uuid.FromBytes(Hash(username||filename)[:16]))]: (File Struct)

    e. Keystore
        i. For each username, store the following information:
            1. PubKey1, PubKey2

3 File Storage and Retrieval

    a. Helper function SendFileToStore(file File) to converts a File struct instance to JSON string, encrypted by certificate with message: {“File is encrypted by [username]”}user.privKey^-1, or, append to messages if there are existing messages: {“File is encrypted by [fileOwner], then encrypted by[userA], then encrypted by [userB]}fileOwner.privKey^-1,userA.privKey^-1,userB.privKey^-1 (Hierarchy of trust)
        i. After calling this function, it should return a JSON string, then append a HMAC computed by HMAC(MacKey, JSONstring) to be FilePackage, and JSON serialize again for FilePackage
        ii. and store in datastore using file.fileID as storage key
        iii. Returns an error if JSON serialization has problem {JSON serialization error}

b. LoadFile(filename)
    i. Compute FileID by uuid.FromBytes(Hash(username||filename)[:16]))
    ii. file, success = DatastoreGet(uuid: uuid.FromBytes(Hash(username)[:16])))
    iii. If the success is false, meaning that the given filename does not exist in the database, returns an error {given filename does not exist}
    iv. If the above statement has error, return the error {Loading the file cannot succeed}
    v. Verify if the chain of certificates from oldest to newest by each public key of the users, throw an error if one of the users cannot be verified {integrity of the downloaded content cannot be verified}
    vi. Return file.content

c. StoreFile(filename string, content []byte)
    i. Initialize a file struct and set the following variable
        1. filename(private var): string
        2. content(private var): []bytes
        3. macKey(private var): RandomBytes(16)
        4. fileID(private): uuid.FromBytes(Hash(username||filename)[:16]))
    ii. DatastoreSet(storageKey: file.fileID), value: sendFileToStore(file))
        1. Before sending it to database,
        2. Returns an error if the above instruction fails {Write cannot occur}
        3.

d. AppendToFile(filename string, content []byte)
    i. Compute FileID by uuid.FromBytes(Hash(username||filename)[:16]))
    ii. file, success = DatastoreGet(uuid: uuid.FromBytes(Hash(username)[:16])))
    iii. If the success is false, meaning that the given filename does not exist in the database, returns an error {given filename does not exist}
    iv. If the above statement has error, return the error {Loading the file cannot succeed}
    v. Create a new file with content appended to file.content, and regenerate macKey, encKey, and fileHMAC
    vi. DatastoreSet(storageKey: file.fileID), value: sendFileToStore(newfile))
    vii. Returns an error if the above instruction fails {Appending files cannot occur}

4 File Sharing and Revocation

a. CreateInvitation
    i. Create an Invitation instance, with the following variables initialized
        1. creatorID = uuid.FromBytes(Hash(user.username)[:16]))
        2. acceptorID = uuid.FromBytes(Hash(recipientUsername)[:16]))
        3. fileID = uuid.FromBytes(Hash(user.username||filename)[:16]))
        4. invitationID = uuid.FromBytes(Hash(user.username||recipientUsername||filename)[:16])
)
        5. isAccepted = False
    ii. Create an invitationPackage, with the following variables initialized
        1. InvitationFile = JSON serialization of Invitation instance
        2. Sign = DSSign(user.privKey, invitationFile)
    iii. Compute FileID by uuid.FromBytes(Hash(username||filename)[:16]))
    iv. Check if given filename exists in database
        1. file, success = DatastoreGet(uuid: uuid.FromBytes(Hash(username||filename)[:16])))

        2. If the success is false, meaning that the given filename does not exist in the database, returns an error {given filename does not exist}
    v. Check if given recipientUsername exists
        1. Returns an error if DatastoreGet(acceptorID) returns false {Given recipientUsername does not exist}
    vi. Define a helper function SendInvitationToStore(Invitation) to convert Invitation instance into JSON string, DSSign it using acceptor's publicKey2, and store it on datastore using invitationID
    vii. Get JSON serialized InvitationFile by calling SendInvitationToStore(Invitation), then call DSSign(user.privKey, InvitationFile), and JSON serialize it, finally calls DatastoreSet the serialized file with the invitationID as storage key.
    viii. Returns an error if the above instruction fails {Sharing cannot complete due to error}

b. AcceptInvitation
    i. Check if given filename exists in caller's personal file namespace
        1. file, success = DatastoreGet(uuid: uuid.FromBytes(Hash(user.username(caller's username)||filename)[:16])))
        2. If success is false, then returns an error {the caller already has a file with the given filename in their personal file namespace}
    ii. Invitation, success = DatastoreGet(uuid: invitationPtr)
        1. If success is true, then returns an error {invitation is not found, or it is revoked)
    iii. Decrypt the invitation using user.privKey2(recipient's private Key)
        1. If the decryption fails, then returns an error {given invitationPtr was created by senderUsername}
    iv. Verify the signature in InvitationPackage by DSVerify using the sender's public key , and throw an error if error occurs {RSA Signature failed, tampering detected.}
    v. If the above statements are executed successfully, then change the isAccepted variable in InvitationFile and then JSON serialize it, use DatastoreSet to send it to DB

## Draft Test Proposal

Groupmates: Ko Tsun Leung(3037588055), David Goldstein(3034119326)

Test 1: This tests whether initUser ran successfully and that getUser only returns this newly created user with the correct password

Bob = InitUser(username, password)
userA, error = GetUser(username, correctPassword)
Expect that no error will be thrown, and userA is same as Bob
userB = GetUser(username, incorrectPassword)
Expect an error to be thrown as the wrong password is provided

Test 2: This tests whether loadFile verifies a file exists

Alice =  InitUser(username1, password1)
Alice calls loadFile(name)
Expect to throw an error since file does not exist

Test 3: This tests whether unauthorized users have access to a file

Alice =  InitUser(username1, password1)
Bob = InitUser(username2, password2)
Alice calls storeFile(test, contents)
Bob calls loadFile(test)
Expect to throw an error since does not have access

Test 4: This tests whether users can be authorized and deauthorized to access a file

Alice =  InitUser(username1, password1)
Bob = InitUser(username2, password2)
Alice calls storeFile(test, contents)
Alice calls createInvitation(test, username2);
Bob callsAcceptInvitation
Bob calls loadFile(test)
Expect the file contents to be loaded
Alice calls revokeInvitation
Bob calls loadFile(test)
Expect to throw an error since Bob no longer has access

Test 5: This tests that a file content is the same across multiple sessions by having the original file creator modify a file

Alice =  InitUser(username1, password1)
Bob = InitUser(username2, password2)
Alice calls storeFile(test, contents)
Alice calls createInvitation(test, username2);
Bob callsAcceptInvitation
File1 = Bob calls loadFile(test)

Alice calls appendFile
File2 = Bob calls loadFile(test)
Expect File1 and File2 to be different

Test 6: This tests whether an invitee can modify a file and its contents be reflected across all sessions

Alice =  InitUser(username1, password1)
Bob = InitUser(username2, password2)
Alice calls storeFile(test, contents)
File1 = Alice calls loadFile
Alice calls createInvitation(test, username2);
Bob callsAcceptInvitation
Bobs calls storeFile(test, contents)
File2 = Alice calls loadFile
Expect File1 and File2 to be different