CS161 Project 2 Design Doc

Edward Chau, Zoe Zhang

InitUser(username, password)

The user struct contains the user's RSA private key, a FilesToEnc<filename, file enckey>, a FilesToHMAC<filename, file hmackey>, and a TokenMap<filename, UUID of ShareTree>. When creating a new user struct, we hash the password and username to generate a set of deterministic UUID, Encryption Key, and HMAC key, used to encrypt and authenticate the user struct to be stored in DataStore. We also store the user's RSA public key in KeyStore. The user struct is then JSON marshalled, encrypted using Encryption Key, tagged using HMAC key and stored in DataStore.

GetUser(username, password)

We regenerate the UUID, Encryption Key, and HMAC using Argon2 with the username and provided password to see if the stored user struct of the corresponding UUID can be verified and decrypted. If yes, both integrity and confidentiality are preserved. We then fetch the user data and unmarshall it.

StoreFile(filename, data)

We generate the UUID, Encryption Key, and HMAC key randomly, followed by adding the keys to the owner's FileKeyMap. If overwriting, we fetch file uuid from datastore and reuse the UUID and keys. We also store the file metadata containing the file uuid, number of appends, and the next/previous uuids of appends (in order to create a linked list of appends for each file, making storefile linear time to number of appends and content appended). We also create a sharetree whenever storing to enable multiple session synchronization. We encrypt file data/metadata using Encryption Key for confidentiality and tag with HMAC for integrity, and then store the UUID of the file with the encrypted file data to DataStore, followed by updating userdata.

AppendFile(filename, data)

We fetch the Encryption Key and HMAC Key of the file from either FileKeyMap, if owner, or from the share tree in DataStore, if not owner. We then generate a new UUID for the to-be-appended part of the file, encrypt and sign the new data of file using the file's Encryption key and HMAC key as we did in StoreFile(), and add the new entry to datastore. Since we are using a linked list, each append will directly access the last node of the linked list, and update only the first, second last and last node of the linked list. This gives a runtime linear to the size of the newly

appended part of the file and number of users sharing the file since we are only encrypting and signing the new parts of that particular file.

LoadFile(filename)

For the owner, retrieve file uuid and keys from their own maps. For a non-owner, retrieve from the share tree in DataStore. To retrieve keys from DataStore, retrieve the UUID of the share tree node struct from TokenMap, followed by a tree traversal through the share tree and getting the uuid of the metadata and filedata. Then we iterate through the linked list of metadata to collect the filedata.

ShareFile(filename, recipient) (accessToken UUID)

For the owner, we retrieve the information to be shared from local maps. For a non-owner, we retrieve the same info from the share tree in datastore. We then retrieve the recipient's RSA public key from KeyStore, encrypt the file keys using the RSA public key, sign with the sender's RSA private key and put these along with UUID of the file in ShareRecord struct, which will be stored in DataStore. We generate a new node for the recipient and store it in the share tree. We then generate the accessToken struct containing the UUID of the rootnode of the share tree, encrypted and signed to be shared with the receiver. Both the owner and the sharers will update the share tree whenever sharing a file to include their descendents.

ReceiveFile(filename, sender, accessToken UUID)

We first verify the encrypted accessToken using the sender's RSA public key, and then decrypt it using the receiver's private key to get the UUID of the share tree. We retrieve the UUID of the ShareRecord struct by traversing through the tree using username, decrypt the struct using the receiver's RSA private key to get the UUID, the Encryption Key and HMAC Key for the file. We then retrieve the file from DataStore using UUID of file, verify the file using decrypted Encryption key and HMAC keys. Lastly, we store the filename, and the UUID of the root node in the receiver's TokenMap.

RevokeFile(filename, targetUsername)

The owner retrieves the share tree of the file from TokenMap and then DataStore. We then traverse through the tree to delete the UUID of the SharedFile struct of all instances of the target user's node and all its descendents.