CS 161 - Project 2

Design Document

Wayne Li [3032103452]

Somi Yi [26233583]

15th October 2018

System Design

User Data:

When a new user, Alice, is registered, we create Alice's personal information - including randomly generated identity code (UUID), multiple keys (EncryptionKey, Nonce, HMACKey), and an RSA PrivateKey. We create the keys and the address (the "key" in our key-value pair) for Datastore with Argon2Key over Alice's username and password so that Alice can retireve her information on future logins. **Second,** we save the Alice's public key to Keystore. **Third,** we encrypt Alice's user data with her own encryption key, and create an HMAC tag for the encrypted data. The encrypted data and HMAC tag are stored in a UserData struct which uploaded to the Datastore. **When Alice wants to login again,** our system will once more use Argon2Key on her username and password to calculate the address of her info in DataStore and will only decrypt and return her user data if the HMAC tag check passes.

File Data:

Storage & Append & Load:

When Alice wants to store a file, our system **first** creates a random address to store the file and also randomly generates EncryptionKey, Nonce, and HMAC Key. **Second,** our system uses her keys to encrypt and HMAC the file and stores this data into a FileData struct. **Third,** the file-storage-address is stored into a struct named FileLink which functions as a LinkedList where that stores the current file's storage address and the next FileLink (nil if it's end of file). **To append to a file,** the new content is stored in a random address in Datastore and a new FileLink end-node struct is constructed (nextFileLink=nil) to replace current FileLink's end-node's nextFileLink. **To Load a file,** the user can iterate through the FileLink, reading currentAddress then nextFileLink, and finally concat the contents together to get the full content of the file. **Fourth,** we encrypt and HMAC the FileLink into a FileLinkStorage struct and store this in a random address in DataStore**. Fifth,** we create an entry point for Alice to actually access this file by first creating an Access Address (using Argon2Key over Alice's UUID and filename) to access the information about the file (we call it metadata). For the metadata, we define a FileMeta struct to contain the address of FileLinkStorage and the keys to access the file content. **Lastly,** we use Alice's encryption key to encrypt the FileMeta into EryMarFileMeta, generate an HMAC tag for integrity and store these to a struct named FileMetaStorage that will be stored in Datastore with the address as Alice's access address.

Share & Receive:

When Alice wants to share a file to Bob, after calculating her access address (key) and passing all HMAC tag checks, we retrieve her metadata (keys and FileLinkStorageAddress). Then, we encrypt the metadata with Bob's public RSA key and create a signature using Alice's private RSA key and store both data and signature in a ShareMsg struct which gets sent to Bob.

When Bob receives the file from Alice, if the signature is valid, the metadata is decrypted with Bob's private key. Then, similar to the process for file storage, Bob creates his own access address with his UUID and the filename he gives the shared file. This metadata is stored into a new FileMeta struct and encrypted and HMAC'd into a FileMetaStorage struct. This FileMetaStorage struct gets sent to DataStore with Bob's access address as the key.

Revoke:

When Alice tries to revoke file access, the basic idea is that she treats the revoked file (content, keys, and address included) as if it's a new file in a new address with new Encryption, Nonce, and HMAC keys. **Then** she re-creates a FileLinkStorageAddress storing FileLinkStorage which has a new FileLink containing only the new file's address. **Then** she updates her metadata for the file, to reflect new keys and address. **Lastly,** Alice deletes the original files' addresses and their contents from the Datastore. This way, all others cannot find nor decrypt the files

Security Analysis

Man-In-The-Middle for DataStore:
We know the channel used to store into and retrieve user data from DataStore is insecure and so it's possible for a man-in-the-middle, Mallory, to read or manipulate user data struct, which contains keys for encryption and HMAC, and a nonce that must stay secure to Alice. As such, when we create and store the user, we provide confidentiality by encrypting data with Cipher Feedback (CFB) encryption. Integrity is provided with an HMAC tag on the user's encrypted data; and if some information is modified, the user can detect the error.

Man-In-The-Middle for Sharing Channel:
We know the channel to send messages for sharing is insecure, which would potentially allow for a man-in-the-middle, Mallory, to read or manipulate the message to leak information. Our system protects against this by having a user, Alice, encrypting her message using Bob's RSA public key for confidentiality, HMAC for integrity, and signing the message with her RSA private key for authentication.

Attack after Revoked:
In our system, it would be possible for Bob, while having access to Alice's file, to remember all the encryption and HMAC keys, the nonce, and the addresses of all the file nodes, potentially leading him to have access to a file even after Alice revokes access. Or Bob may call receiveFile again to re-gain the metadata Alice shared to him. Our design protects against this because when access is revoked, we generate completely new keys for the file contents, then modify Alice's metadata to reflect new encryption and HMAC key and nonce, as well as the new file address. We delete the file contents at the old address and thus, Bob would be unable to access the file with the data he saved previously.