CS161: Computer Security
Project 2 : Design Document Final
Version 1.3
July 29, 2020

INTRODUCTION

In the project, we design a secure file sharing system. In particular, we build a client for a file sharing system, whereby the client allows users to store and load files, share files with other users, and revoke access to a shared file from other users.

SYSTEM DESIGN

First, for user verification, we use digital signatures to address this problem. We store the hash value of the user's password and its digital signature in the datastore and the corresponding public key in the keystore for future authentication. When a user logs in, after checking the integrity of the password hash value in the datastore, the client will compare the hash value of the one that the user provided to verify the user.

Second, for all the other data owned by a user in the datastore, including all files, file metadata, user metadata and struct used to support file sharing, we use symmetric encryption and HMAC to guarantee their confidentiality and integrity. The generation of the keys used to encrypt/decrypt and sign/verify these data forms a key hierarchy as in *figure 1*. The origin of all keys is the hash value of the password(different from the one used in verification), since when bootstrap the password is the only secret thing we can use. Those keys that relate to the metadata of the file and the user is in the second level of the hierarchy, while the keys of the data that can be shared with other users such as file data the file location record are in the third level. By this manner, metadata keys of each file in the second level remain constant, however, the sharable keys in the third level are changeable and generated based on a variable purpose string for HashKDF() in order to support file permission revocation.

Third, in order to support file sharing and permission revocation, the file sharer also stores a shareable struct called *FileSecrecy* in the datastore. Based on the secret key(i.e the second level key used to generate third level keys) in that struct, the file sharee is able to encrypt/decrypt and sign/verify the shared data. The UUID and encryption/verification keys of that struct are transmitted to the sharee by access token. We use public encryption and digital signatures to maintain the access token’s confidentiality and integrity.

SYSTEM DESIGN PROBLEMS

1. How is a file stored on the server?

File Data A file is stored on the datastore as a series of encrypted and maced data sections with different uuids, generated randomly when StoreFile() or AppendFile() is called.

File Metadata UUIDs of every data section will be recorded in another struct called *RecordBook,* which is used to obtain the whole file by users with permission. The UUID of the *RecordBook* will be stored in the metadata of the file called *DataStoreFile* which solely belongs to the file creator. Meanwhile, *DataStoreFile* will also maintain a map to record the file shared users.

2. How does a file get shared with another user?

In order to share a file with another user, we provide the UUID of the struct that records each file section's location and the secret key(i.e the *Sk2'* in Figure 2) that the sharee can use to generate file data decryption/verification keys. Therefore, we wrap these informations in a single struct called *FileSecrecy* and store it in the datastore. Meanwhile, we use an access token to pass the UUID and the decryption/verification keys of this struct to the sharee. After the sharee receives the access token, he will register that UUID and *Sk2'* in his metadata. Then when he later wants to access the file data, he can use this information. Also, when the sharer updates the file with a new version, he will also update this struct in the datastore so that users with permissions can still access the new version.

3. What is the process of revoking a user's access to a file?

First, we check whether the user is a direct sharee of the file creator. Then we delete him from the file metadata, which records all the shares. Second, we generate a random purpose string in order to generate new third level keys related to the encryption/verification of the file shareable data. We use these new keys to encryption and mac every file section and file location record and store them back into the datastore. In other words, we construct a new encrypted version of the file. Third, for every other shares in the file metadata struct, we update their corresponding FileSecrecy struct in the datastore so that they are able to get the new UUID of the file location record and generate new third level keys to decrypt/verify the new version of the file.

4. How does your design support efficient file append?

We regard the data appended in each call of *FileAppend*() as a single storage unit while at the same time recording its UUID to the *RecordBook* struct. Therefore, when later calling *LoadFile*(), the user is able to obtain the whole file by combining file data related to each UUID in the *RecordBook* struct.

KEY HIERARCHY

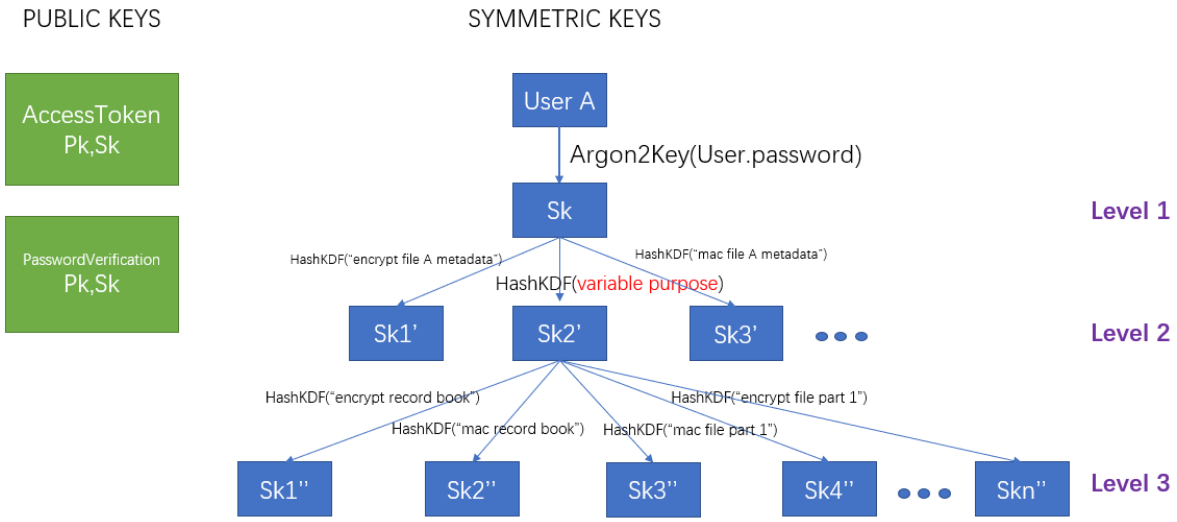

Figure 1 *Key Hierarchy*

SECURITY ANALYSIS

Attack 1: An attacker may conduct a possible dictionary attack on our password in the datastore. We use Algo2key() and different salts for different users to slow down that process.

Attack 2: An attacker can modify anything in the data store, but since for every data in the datastore we use encrypted version and also store its mac value or digital signature value, so the user can discover malicious modification. For instance, when A shares a file with B, A stores a secrecy in the datastore so that B can get the secret key to decrypt the file and when A changes the secret file for that file, A modifies the secrecy to notify B. An attacker may modify the secrecy in the datastore. Since we encrypt/mac the secrecy using SYMENC() and HMAC(), any malicious behavior will be detected.

Attack 3: An attacker may modify/read users' metadata in the datastore, however, since we use symmetric encryption for user metadata and mac for integrity check. The attacker's modification can be detected and also he cannot get any information from the encrypted version of that struct in the datastore.

Attack 4: When User A sends access token to user B, an attacker may eavesdrop the access token and modify it, however since we use B's public key to encrypt the access token, the

attacker cannot get any information about the access token. Also, since we use A's secret key to sign the access token, the modification that the attack made can also be detected when B verifies the access token using A's public key.

Attack 5: An revoked user may try to overwrite the shared file with the origin data(i.e old version encrypted file and mac value), however in our design the file creator will not only change the UUID of the new version file but also use new secret key to encrypt and mac the data. Therefore, this Dos attack will be detected by the file owner.