Andrew Huynh, SID 25607104, cs161-ahd

Design Document

To begin, I chose to add fields to the User struct. These are: the user's username, the user's password, the user's private key, a map that maps what the user calls a file to what the initial creator called the file (for when users give shared files a name) both HMac'd using the respective user's private key, and a map that maps file names to the keys used to encrypt them (these will be symmetric keys used when sharing files).

We can assume that username and password combinations are unique and that passwords have high entropy. So I used the username and password as inputs to PBKDF2Key to create unique User struct storage locations. Before storing the User struct I CFB encrypt it using the password as the key and also HMac it using the password as the key. This way when we load a User struct for a user that gives a correct username and password we can check if the User struct has been corrupted by comparing HMac values and return the decrypted User struct if is not.

I also introduced a File struct, a structure that holds data for a file and any appends that it may have. It has fields for: an array of creators (keeps track of who made what addition to the file to check signatures), an array of file data (to keep appends sequential), an array of RSA signatures, and a map that maps users' names that the file is shared with to 1.

We can assume that a user's private key and filename combinations are unique. So I used the user's private key as the key for HMac and HMac'd the filename to create unique File struct storage locations.

For a new file, before storing the File struct the data is encrypted using a hashed randomly generated key which is used as the symmetric key when sharing files. For appending to an existing file, the append data is encrypted using the symmetric key for the file. To ensure

integrity, the encrypted data for both is signed using the user's private key. Using the array of creators from the File struct, we can retrieve each data's creator's public key and check if all the signatures are valid and only decrypt the file if they are. This allows multiple users to append to a file and check for the file's integrity.

To share files, I store a sharingRecord struct and send its location in memory to the recipient with RSA encryption and sign the encrypted location. I gave the sharingRecord fields for: the file location, and the symmetric key RSA encrypted. The recipient checks if the signature is valid and decrypts the location for the sharingRecord if it is. Then the recipient maps the HMac'd name they give the file, using the user's private key to create unique maps, to the file's location in memory. The recipient also maps the HMac'd name they give the file, again using the user's private key to create unique maps, to the symmetric key. This allows multiple users to be able to encrypt and decrypt the same data.

To revoke a file, you need to have the file name that the initial creator gave it and the initial file creator's private key. If not, then the revoke is denied. This means that only the initial file creator can revoke access to a file. When a file is revoked, I generate a new symmetric key and re-encrypt all the data using this new symmetric key. Therefore shared users that could previously decrypt the file can no longer decrypt the file.

The testing methodology is simple. I create a few users and files and attempt valid file loads, invalid file loads, valid file appends, and invalid file appends. Then I share a file between all of them and again attempt valid file loads, invalid file loads, valid file appends, and invalid file appends. I finish with attempting valid and invalid file revokes and again attempt valid file loads, invalid file loads, valid file appends, and invalid file appends for the now revoked file.

Security Analysis

One attack my design protects against is data corruption from the server and other users. Important data is stored with an HMac with varying keys: User structs use the user's password as the key, file data uses a hashed randomly generated symmetric key as the key. The location of sharingRecord structs that we send when sharing files are RSA signed. A user's password should only be known to the user. The symmetric key is only known to the system. It is stored inside the encrypted User struct and stored RSA encrypted in its sharingRecord structs. The symmetric keys stored inside the encrypted User structs cannot be discovered unless the attackers know the user's password, again which should only be known to the user. The symmetric key in the sharingRecord struct cannot be discovered unless the attacker knows the user's private key, which is known only to the system and is stored inside the encrypted User struct. Before decrypting any data, my design compares HMac values and raises an error if they are different and continues decrypting if they are the same. The recipient of a file share can confirm whether msgid, the location of the sharingRecord struct when sharing files, came from the sender or not by RSA verifying.

A second attack my design protects against is attackers attempting to brute force User struct locations and File struct locations even if they know user names and file names. User structs are stored at a location based off of the user name and the password using PBKDF2Key. We assume that the passwords are of high entropy. Therefore attackers cannot feasibly guess User struct locations unless they know a user's password which should only be known to the user. The case is similar for File struct locations. File structs are stored at a location based off of the user name and user's private key using HMac with the private key as the key for HMac. The private keys are of high entropy. Therefore attackers cannot feasibly guess File struct locations

unless they know a user's private key which is only known to the system. The private keys are stored inside the encrypted User structs so they cannot be discovered unless the attackers know the user's password, again which should only be known to the user.

A third attack my design protects against is attackers gaining access to the entire data store. All important data in the data store is encrypted. This includes: the User structs, the file data in File structs, and the symmetric keys in sharingRecord structs. A User struct is CFB encrypted with the user's password which should only be known to the user. A file's data are CFB encrypted using a symmetric key that should only be known to the system. The symmetric keys are stored inside the encrypted User structs so they cannot be discovered unless the attackers know the user's password, again which should only be known to the user. The symmetric keys are also stored in its sharingRecord struct RSA encrypted. An attacker cannot decrypt this without the user's private key which is stored in the CFB encrypted User struct. The file storeage locations are created using HMac on the filename with the user's private key as the key for the HMac. The private keys are stored inside the encrypted User structs so they cannot be discovered unless the attackers know the user's password, again which should only be know to the user. Therefore knowing where files are stored does not tell attackers anything about file names.