Jared Spickard
McClain Thiel

CS 161 Project 2 Design Doc (Revised)

Section 1: System Design

Our design is best understood via our structs. For each of our basic structs (User, Sentinel, FileNode), we have a wrapper class that holds an encrypted version of the inner version in addition to the reported HMAC of the inner struct calculated at the time of storage. Because all of these types are stored in the untrusted data store, our wrapper structs allow us to ensure file integrity, by checking the reported HMAC vs a new calculated HMAC based on the data retrieved from the datastore. When a user loads or stores a file, they first interact with the sentinel node which contains the location of the wrapped file on the database and an access tree that keeps a database of users approved to access the file the sentinel guards. If the sentinel approves a user to access a file, a helper function retrieves the file wrapper from the database and checks the HMAC. If the HMAC is approved it proceeds to decrypt the file and return it to the user.

Sharing and revoking are also handled using the sentinel structure. If a user wants to share a document they can add the username of the target user to the sentinel's access tree and then give the target user a magic string that can be deciphered to reveal all of the information necessary for the target user to find the file in the datastore. This magic string is both encrypted and digitally signed during transfer and processing. Revoking access is primarily handled by removing the target user's username from the access tree.

1. How is a file stored on the server?

Files are stored in a nested structure where the inner struct is the encrypted (and marshaled of course) data and the outer structure contains an HMAC in addition to the inner structure.

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

First, the target user's username is added to the corresponding sentinel's access tree. Next, a unique magic string is computed using all of the information required for the user to find the file and the target user's data. This key is then sent to the user who will store it. When they want to access the file, they can present the token to the sentinel which will grant or deny access.

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

The target user's name is simply removed from the sentinel's access tree. The sentinel will deny access the next time they try to load the file and they will remove the file from their list of files shared with them.

4. How does your design support efficient file append?

Our program stores the appended block in a separate location in the datastore pointed to by the file it was appended to. When that file is next loaded after both chunks are decrypted, but before

the file is returned to the user, the files are appended. This is efficient from both a bandwidth and an encryption time perspective.

Section 2: Security Analysis

Attack 1

We know our adversary has complete access to the data store, so it's quite reasonable to expect that they might attempt to read or tamper with our data.

Defense 1

Our system defends against an adversary readying our files by encrypting all data on the data store and generating the location via hashing. This should prevent them from reading the file and make it at least difficult to find the file. If they do manage to find it and try to tamper with it our nested HMAC structure will at least alert the user that the data has been tampered with.

Attack 2

Another attack an adversary might attempt is to counterfeit an access token such that the target is writing to the adversary's file when they think they are writing to someone else file.

Defense 2

We defend against this by both encrypting and signing the access tokens. The encryption is done using public key encryption with the target user's public key as the encryption key. When the encrypted key is sent from one user to another, a digital signature is appended and when the target user gets it, they check the signature to ensure it came from who they thought it did.

Attack 3

We currently assume the encryption is safe from cracking via traditional methods, but if an adversary did manage to gain unauthorized access they might try to recover passwords

Defense 3

Our defense against this is two-fold. First and foremost, passwords aren't stored anywhere. We store a hash of the password for comparison when logging in, but a user could never recover the password itself. Another level of defense is that each user's data is encrypted separately and using keys unique to their account. There is no central point an adversary could gain access to where they could cause damage to more than one user.