Siddharth Bansal
Kanak Garg

CS 161: An End-to-End Encrypted File Sharing System

Section 1: System Design

1. How is a file stored on the server?

We first make a File struct which stores the UUID of the File struct that holds an append to the file, the data of the file, and a ShareTree struct that holds the hierarchy of which users the file is shared with. Then we marshal that File struct and encrypt it with a symmetric key. Finally we take that encrypted File struct and generate a MAC for the ciphertext. We then append the MAC and the ciphertext which is the encrypted File struct and store it on the Datastore server under the UUID of the File Struct, which the user stores in their mapping of filenames to UUIDs.

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

In order to share a file with a user, you would need to send three pieces of information The other user would need a UUID, in order to know where to access the file in our server. They would also need the proper keys to decrypt the file information and verify that the file has not been tampered with. Finally the other user would also need a signature to make sure that the magic_string that the user is receiving is coming from the correct person. The sharer uses the receiver's public key from Keystore to encrypt the UUID and key, and then signs the encrypted message and sends it as well. The other user can thus verify the signature, take the packet and finally have all the information to access the file. Furthermore, the sharer also modifies the File struct for the specific file so that the other user is added to the ShareTree struct.

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

To revoke a user's access to the file we remove their user name from the ShareTree struct, which is used to verify that all accesses to a file are by shared users, in the File Struct for the relevant file. Since the struct is a type of tree, when we delete a node which represents a User, all of the node's children, which are users that were shared to the file by the specific user are also deleted from the ShareTree.

4. How does your design support efficient file append?

The File structs that we implement have a linked list like structure where each struct has a field nextAppend which stores the UUID of where the next append should be stored on the server. Therefore, when appending we simply just encrypt the File struct to append, generate a MAC and then store it on the Datastore server at the UUID indicated by nextAppend.

Section 2: Security Breaches

Man in the Middle Attack:
A malicious attacker could intercept the communication that is sent while sharing a file with another user and then change it to a message of their own choice. To prevent this attack, all messages are sent with RSA encryption using the public key of the person that the message is being sent to so that an attacker can not decrypt the message, along with a signature signed by the user sharing the file so that the user receiving the message can verify the source.

File Manipulation Attack:
In the case that Eve gets her hands on the UUID of the file the User A uploads, Eve can rewrite or swap files onto that location in the server. Thus to prevent User A from accessing a file that is different from what he or she had stored, we store a MAC on the server with each encrypted message so that the user knows the file data was not tampered with. Verification of HMACs provides integrity and authenticity, thus preventing A from being deceived by Eve's malicious payload.

Changing User Struct:
An attacker could try to access the DataStore and change or read the parts of the User struct that hold the username, keys, and file UUIDs in order to mess with the integrity of the data sharing system This attack is prevented because the information is stored and encrypted using a combination of the password based key derivation function and the hash-based key derivation function so the attacker cannot reveal any information about the stored data. Since the functions above are deterministic, a user's username and password is enough for us to derive the UUID of the data and the relevant encryption and HMAC keys. Using these keys we can keep the data secure through symmetric encryption and also verify it against malicious activity with the MAC key.