SECURE FILE SHARING

Nitin Vivek Bharti
18111048

Yaddanapudi Sri Divya
18111073

March 4, 2019

Property1: InitUser

The user details are stored in the Data store with the key generated by the hash of the concatenation of username and password. Key in data store for user data : < H(username || password) >
We will generate RSA private key- public key pair for this user, and store the public key in the KeyStore with username as the key so that other users can get the public key of the user. The user structure will be stored in the json format which contains the following fields:
< random salt || H(password || random salt) || H(username || password) || private key >
In order to maintain integrity and confidentiality we store the hash of the entire value of the user data and then we encrypt the entire value with CFBEncrypter with key as the password and IV as password hash. Multiple users with same name are not allowed to create an account, an error would be issued in such cases as we can find if the username already exists in KeyStore.

Change:

The user structure will be stored in the json format which contains the following fields:
< Username || Argon2(password) || private key || Data store key || File Meta Data Locator || Hash of User structure >
The user structure is Encrypted using CFBEncrypter which uses Argon2 password hash as key and Random string as IV which is stored along with the cipher text at the start.

Property 2: GetUser

This takes two arguments username and password. We concatenate username and password to generate a string and hash it. Then we find if this hash is present in keys the DataStore map table. If this key does not exist then return error "Invalid Information Entered". If the key exists then value pointed by this key is fetched. Now using the password as key and hash of password with length as block length we generate key using CFBEncrypt and use the key returned by it to decrypt the data. Decrypt will give us a json structure that can be used to fill User Data structure and validate whether the data has been tampered. A check is made to ensure the password entered is correct for the user and not by collision of the hash function the value is retrieved .We do this by checking < H(password || random salt) == saved hash of the password || salt >. If this does not match then return error. If matches then verify data stored has not been tempered with by comparing hash of data with saved hash. If not tampered then return the User structure else return error.

Change:

We use Argon2 Password hash as key and first 16 bytes of value from Data Store to Decrypt using CFBDecrypt. We check if the password hash matches the saved Hash to verify the identity of User.

Property 4: StoreFile

Load file is explained later because it needs understanding of how File is stored on DataStore server.
To store a file we are going to use hash of username+filename as this will make sure two files with same name are not

possible and easy to find the key for the that file in DataStore. We are going to create metadata for each file and store it at this key location. This meta data will guide us to find and read data of the file. This meta data consists of IV and key(randomly generated for each file) used for CFBEncrypt, Shared address, File Blocks address, DataStore key and hash of all data combined. This file meta data structure will be encrypted using RSAEncrypt and then stored as value of the key for file.
Shared address key will be generated with file name along with some random number. Shared address will contain the names of user it is shared with along with the hash of the data. Initially at creation only the name of owner will be there. This will also be encrypted using IV and key used for the file and stored.
File Block Address will be generated with key as file name and some random number. This will contain the key for each File Part and each new file part is an append. Along with the key, hash of the value of the File Part is also stored along with each key. This combined data is hashed and encrypted using same IV and key pair and stored on the Data Store.
File part contains the data and its hash in encrypted format using IV and key pair for this file. The key is generated using hash of filename+"parts"+random number.
• Since meta data is encrypted by each user using his public key only he can decrypt the data. • Our model is safe from any swap operations performed on the values of the keys. We are not checking the case when two File Parts with same data of the same file are swapped as it will not be an issue.

Change:

The File Meta Consists the following data for each of the file, the UUID where the file part is stored, the key used to decrypt. The IV is stored in the start of the block of the encrypted data of the file meta data. The File Meta Data is stored against a random UUID in the File Meta data Locator with the filename as key in the user structure. The file is stored at a random UUID. The UUID is stored in the File Meta data along with the key. The IV is the start of the encrypted data of the file.

Property 3: LoadFile

Benign Setting :

Search the data store with key <H(username || filename>. If there is no value under key return nil. Decrypt the value obtained from Data store using the private key of the user which is stored in user data structure. Make the following checks on the value • Compare hash of the file meta data fields with the hash of the value to ensure integrity • Ensure the user requesting the file is the authorized(owner/shared with) user of the file • For shared address , file blocks address fields fetch the value stored under the keys and check for integrity after decrypting using the key and IV stored in the file meta data. File Block validation is two layered file storage system as mentioned. To check if File Part is not tampered with each File Part is checked after decrypting that its hash matches with hash stored in File Part block. Also it is checked that the key contains the data it is intended to by comparing the hash of the File Part block with the hash stored in File Block. If all the integrity checks are successful then concatenate the file stored in different File Parts and return content to the user.

Attack Setting :

If any of the above mentioned integrity checks fail then return null to the user as the file has been modified by an attacker or other user.

Changed:

Get the file meta data location from the user structure through the file meta data locator. Using file meta data find the UUID of the file parts and perform integrity checks on the data, file meta data if any of it fails return error. Else return the data present after decrypting using the key stored in the file data map of the meta data and IV is the starting block data.

Property 5: AppendFile

To append file first we verify if the meta data is not tampered by steps mentioned in LoadFile. Then we create a key for File Part to append new data as mentioned in StoreFile. This key will be included in the File Block along with the hash of value at the new File Part key. New hash of the file block data will be calculated and encrypted again to store. This

requires only updating File Block Address and new File Part to store the encrypted data.
Changed : The UUID and Key of the new file part are added in the meta data location of the file, the IV is stored in the start block of the encrypted text and stored with the key specified in the file meta data locator.

Property 6: Sharing and Receiving File

The sharing information will be constructed by decrypting the meta data block of the file and then encrypting using the public key of the receiver. In order to prevent man in the middle attack we sign the information being shared with signature of the sender. The sender passes this information to the receiver. When the receiver calls ReceiveFile integrity checks are made to compare that signature is that of the sender and the data has not been modified, if the integrity check is passed then a new key value pair is created in the data store where key is the H(username || filename) and the value is the information received from the sender encrypted in the public key of the receiver. Here the receiver gets all the meta data required to access the file and also he is accessing the file by a new file name not the one sender has.
Changed : The File Meta Data block is shared to the Receiver. On calling ReceiveFile the file is added to the meta data locator under the filename specified which contains a link to the meta data location of the file. The file is now not stored under H(username || filename)

Property 7: Revocation

If the authorized user of the file calls to revoke the file, new meta data will be created for the file. By this the keys are also changed so the entire file data needs to be encrypted with the new key and stored in new location.