CS-628 Assignment 1
Design

Ayush Nagal, Ashish Kumar

September 3, 2019

A User data structure and a file directory is created for every user.

1. **User Struct**
   - **Username**
   - **Private RSA key :** It will be used later for sending encrypted text.
   - **Symmetric Key(K0) :** It will be used for encrypting file directory.
   - **SHA256(password)**
2. **File Directory Struct :** It is encrypted using CFBEncrypter with K0 as the key and stored at location HMAC(username) with SHA256(Password) as the key. HMAC(Encrypted File directory + HMAC(username)) is also stored along with the encrypted file for integrity check.
   - **SHA256(filename)**
   - **SHA256(username + SHA256(filename)) :** Location of meta-data
   - **Symmetric Key(K1) :** Key for encrypting meta-data
3. **Sharing Struct**
   - **SHA256(username + SHA256(filename)) :** Location of meta-data of the file to be shared
   - **Symmetric Key(K1) :** Key for decrypting meta-data of the shared file

Question 1: Simple Upload/Download

InitUser :

- Generate a key **KK** = Argon2Key(SHA256(password) + username, username, 16). Encrypt user data using CFBEncrypter with **KK** as the key. Generate the HMAC(Encrypted user data + location of user data) using the same key and store it at SHA256(SHA256(password) + username) and user data structure at SHA256 (username + SHA256 (password)).
- Also we create a file directory structure for this user and store it at HMAC(username) with SHA256(password) as the key.
- Generate an RSA key pair, random symmetric key(K0) and SHA256(password). Push the RSA public key to the keystore and store the remaining information in the user data struct.

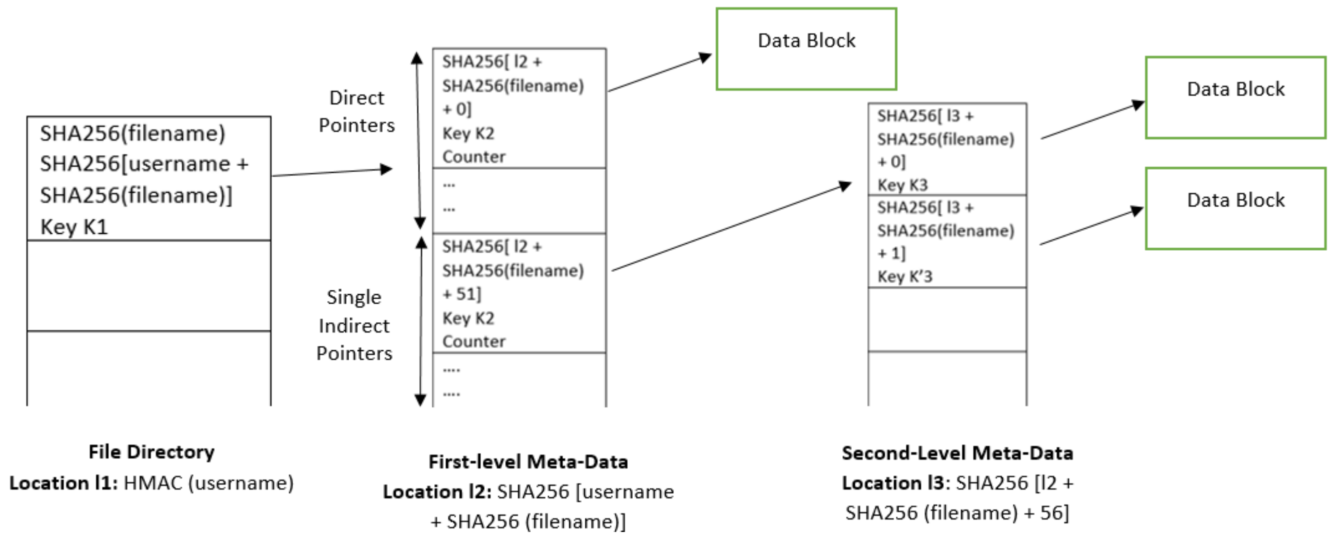
GetUser :

- Get the location of user data struct using SHA256 and key(KK) using the above invocation of Argon2. If password or username is incorrect, return an error.
- Generate HMAC as explained in InitUser and compare it with the stored HMAC. If all checks are satisfied, decrypt the user data using CFBDecrypter with key KK and return the user data.

StoreFile :

- Fetch the file directory from HMAC(username). Generate HMAC(Encrypted File directory + HMAC(username)) and compare. If no error then, decrypt the file directory using key K0 which is stored in user data. If the size of data is not a multiple of blocksize then simply return error.
- Generate a random symmetric key(K1) and store it in file directory along with other details(filename and meta-data location). Encrypt the file directory using key K0 again and update HMAC as well.

- Meta-data contains 50% direct pointers and 25% each single and double indirect pointers. We will encrypt it like we did for file directory but the key will be K1 and generate HMAC(encrypted meta-data + SHA256(username + SHA256(filename))) as well. Store these two at location SHA256(username + SHA256(filename)).
- At each level a key is stored, which is used to encrypt data stored at the next level. HMAC is also stored using the same key and stored. A detailed figure is given below:


SHA256(filename)
SHA256[username + SHA256(filename)]
Key K1
Direct Pointers
Single Indirect Pointers
SHA256[l2 + SHA256(filename) + 0] Key K2 Counter
...
SHA256[l2 + SHA256(filename) + 51] Key K2 Counter
....
Data Block
SHA256[l3 + SHA256(filename) + 0] Key K3
SHA256[l3 + SHA256(filename) + 1] Key K'3
Data Block
Data Block
File Directory
Location l1: HMAC (username)
First-level Meta-Data
Location l2: SHA256 [username + SHA256 (filename)]
Second-Level Meta-Data
Location l3: SHA256 [l2 + SHA256 (filename) + 56]

LoadFile :

- Use the method given in StoreFile() to reach, decrypt and verify each level of meta-data for the given filename. If no error then decrypt and return the file.

AppendFile :

- Use the method given in LoadFile() to reach, decrypt and verify each level of meta-data for the given filename. If the size of data is not a multiple of blocksize then simply return error.
- Generate a new key to encrypt the data block and for generating HMAC as well. Increment the counter and store the encrypted data block at SHA256(current location + SHA256(filename) + counter) along with the HMAC.

Question 2: Sharing

ShareFile :

- Fetch the file directory and decrypt it. Create a sharing data structure to store the location of file and its key.
- RSASign the sharing data structure using sender's private key followed by encrypting the sharing structure using receiver's public key. Then generate a key ks = Argon2Key (SHA256 (SHA256 (username))), username, 16). Use ks to generate a HMAC(encrypted sharing struct + SHA256(sender + receiver + SHA256(filename))).
- These two(encrypted sharing structure and its HMAC) are stored at location SHA256(sender + receiver + SHA256(filename)). RSASign this location using the sender's private key and encrypt this location using receiver's public key and return it as the sharing message.

ReceiveFile :

- Decrypt the message ”sharing” using the private key of the receiver and verify the RSA signature using sender's public key. Generate the HMAC as explained in ShareFile and compare. If no error occurs, decrypt the sharing struct using receiver's private key and verify the RSA signature using sender's public key.
- Copy the file location and its key from the sharing struct to the receiver's file directory and save the updated file directory.

Question 3: Revocation

RevokeFile :

- Fetch the file directory and decrypt it as explained in StoreFile().
- Fetch the first level meta-data and decrypt it using the key(K1) stored in file directory. Now generate a new location by SHA256(old address) and a new key to encrypt it and store this key at the previous level(i.e., file directory), as explained in StoreFile(). Update the HMAC as well and delete the old data block.
- Likewise do the same for next level meta-data and after that for the data blocks as well. So that all the old encryption keys get replaced by new ones.