Encrypted File Store
Assignment 1

Ashwani Bhat Manikant Singh

September 2, 2019

Design Document

1. **The Program will start with *InitUser*:**

   - User signs up with a username and password. We have ensured that the user sign up takes place only once for a particular user.
   - We have generated CFB key to encrypt the user data using **Argon2key** with *password* as its salt.
   - Integrity of the user data is maintained using HMAC with HMAC key as *(Username + password)*. This HMAC is prepended to the user data and then the entire structure is encrypted.
   - Generate RSA private key and public key using *GenerateRSAKey()*.

User data struct has the following fields: $\left\{\begin{array}{c} username \\ password \\ RSA\ privatekey \end{array}\right\}$

The user data will be stored in datastore so that it can be retrieved later on incase of system restart. RSA Public key generated above will be stored in the trusted key store with *username* as the key of the KeyStore.

2. **Retrieving User information using *GetUser:***

   - When User logs in, his entry is first checked in the datastore.
   - HMAC will be computed for the user data in the datastore which is then checked with the previous HMAC to ensure that the integrity is maintained.

3. **Storing file in the dataStore using *StoreFile***:

   - We will have following data structures to store file.

     File$\{RootIndexUUID\ uuid.UUID, BlockCFBKey\ []byte, StackPointer\ int\}$
     Root $\{DP\ []byte,\ SIP2Block\ [800][]byte,\ Top\ int\}$
     Block $\{Data\ []byte, Hmac\ []byte\}$

   - Generate a FileCFBKey to encrypt File structure using *(Username + Filename)*.
   - Generate a Random BlockCFBKey which is later stored in the file structure.
   - The Hmac of block is stored within the block structure, which is then encrypted with the blockCFBKey.
   - We have used a pointer to the last stored block.

4. **Sharing files between Users using *ShareFile:***

   - We will have following data structures to share file.

     SharingRecord$\{RootIndexUUID\ uuid.UUID, BlockCFBKey\ []byte, StackPointer\ int\}$
     Message $\{\ EncryptedMessage\ []byte,\ Sign\ []byte\ \}$

   - To share a file we are sharing the file information with the other user. Other user will get the shared information and create his own file structure.
   - The basic idea of our share implementation is that every authorized user has a pointer to the root structure and every authorized user has its own copy of the file structure.
   - The message id which is created by the sender is encrypted using the receiver's public key and signed using sender's private key.

5. **Receiver receives the shared file using *ReceiveFile:***

   - Receiver gets the message id from the sender.
   - Receiver verifies the message id using sender's public key which he gets from the Keystore.
   - Then he decrypts the message id using his own private key.
   - Then the user creates his own file structure which contains the root uuid of the actual data blocks present.
   - File structure is different for each user sharing the same file but the root structure remains same. This step is same as original user's *Store file* except we do not perform any actual file movement (copy/read/write) operations separately to make file accessible to the receiver.

6. **Adding extra blocks of data using *AppendFile:***

   - Every user with the access privileges can append new data blocks to existing file.
   - The user first accesses the file structure of the file from the datastore using *Username + Filename*.
   - For the last block we are maintaining a pointer. This pointer helps in appending a new block next to the last appended block.
   - The encryption of this block and calculating the MAC is same as done in the store file method.
   - Since, there is only one copy of original file all the changes will be available to all other authorized user immediately.

7. **Revocation of the collaborators using *RevokeAccess:***

   - We are ensuring that the revoked user even doesn't get to know the location of the file i.e., even the partial information is hidden from the revoked user.
   - This is ensured by changing the Root uuid. We are also changing the BlockCFB.
   - We are also deleting the entry of the old root in the datastore by deleting the key from the datastore. This way the revoked user will never have access to any previous data if he hasn't stored a copy of that data.
   - Since the BlockCFB is generated afresh, encryption of the blocks with this newly generated key is performed.