Assignment 1

Siddharth Chinmay(160685)
Yogendra Singh(160829)

16 August 2019

1 A simple, but secure client

1.1 A simple Upload/ Download

1.1.1 InitUser(username string, password string)

First, a key is generated from the given username + password using Argon2Key. Then, the RSA key-pair is generated. The public key is registered in the Keystore. The username , password and private key are used to populate the User Data Structure. The hash for the User structure is generated and the user structure is encrypted using the password key. Then, the username + password hash is generated to get the location (key) to store the user structure in the Datastore. The encrypted user structure and user structure hash is stored in the location of the username + password hash.

1.1.2 GetUser(username string, password string)

Username + password hash is generated and user encrypted data is fetched from data server using DatastoreGet(key string). Then Argon2Key is generated using username and password to decrypt the user data. Then the integrity of the user data is checked by computing the hash of decrypted data against already stored hash in user data. If the user data is corrupted or, either the username or the password is incorrect, then the hashes will not match, resulting in an error.

1.1.3 StoreFile(filename string, data [ ]byte)

Each user structure has a map which stores the file data. Filename hash acts as key which maps to the SharingRecord struct which stores the inode address and CFB key. StoreFile function first searches if the Filename hash is already present in the map of the user structure.

If it is, then we overwrite the data (Update Inode and Data Blocks), else we create a new entry in the map with key as the Filename hash. Then we generate the file (Inode) address and the CFB key randomly. The Inode contains the number of blocks followed by single indirect pointers and double indirect pointers (along with the hash of the data blocks appended to each pointer). In StoreFile, the number of blocks is equal to the DataSize/BlockSize. Data is divided into blocks and encrypted using the CFB key, then stored in data blocks pointed to by the single and double indirect pointers. Their hashes are generated and appended to the respective blocks. The Inode is encrypted and its hash is generated. The encrypted Inode with its hash and the encrypted data blocks are then stored in the Data Store.

1.1.4 AppendFile(filename string, data [ ]byte)

The Filename hash is searched in the file map of the user to get the SharingRecord struct which has Inode address and the CFB key. We receive the Inode from the Data Store and decrypt it using the key. Then, the hash is generated and matched with the old hash to check Integrity. The number of blocks is updated to the old number of blocks + DataSize/BlockSize. The data is divided into blocks and encrypted using the key, then stored in Data Blocks starting from the first empty data block (can be calculated using the old number of blocks). Their hashes are generated and appended to the respective blocks. The updated Inode is encrypted and its new hash is generated. The updated encrypted Inode with its hash and the encrypted new data blocks are then stored in the Data Store.

1.1.5 LoadFile(filename, blockOffset integer)

First, we calculate the block position using the blockOffset (This depends on the number of single and double indirect pointers). Then the Filename hash is searched in the file map of the user to get the SharingRecord struct which has Inode address and the CFB key. We receive the Inode from the Data Store and decrypt it using the key. The hash of the Inode is generated and compared with the old hash. Also, hashes of all Data Blocks are generated and compared to the old hashes appended to the data blocks. If all hashes match, then Integrity is preserved, otherwise we raise an error. If no error, then get the encrypted data block from the Data Store using the block position. We decrypt it using the key and return the data. If no data is found, then nil is returned.

1.2 Sharing and revocation

1.2.1 Sharing

m = a.ShareFile(n1,“b”): File n1 is first searched in the map list of user ‘a’, if found then we get the address and key value of the file. Then we encrypt the address and the key using b’s public key and signed it using a’s private key for authentication. This encrypted data and the signature forms the message ‘m’.
b.ReceiveFile(n2, “a”, m): ‘b’ will decrypt the message ‘m’ using his private key and checks the authenticity of the message received with signature check using a’s public key. Once the address and the key are authenticated, ‘b’ will check for n2 in his file map. If found, then we overwrite the address and key mapped to by n2 hash with the received address and key. Otherwise, we create a file with filename n2 and map the received address and key to it.

1.2.2 Revocation

New address and key pair will be generated randomly. Find the file in the file map. Decrypt the file using old encryption key at old address. Then store the file data at newly generated address and encrypt it using newly generated key. Old data blocks and inode structure are deleted from datastore which were stored at old inode address.