REVISED DESIGN PROPOSAL

JAY RAJ SINGH(19111038) DHEERAJ KUMAR PANT(19111029)

September 1, 2019

1 Part I

1.1 User Data Structure (UDS)

User Name	*Password*	*Private Key*	*Argon Key*	*IV*

1.2 File Records

File Owner	*Size in blocks*	*Sharing Flag(bool)*	*Location of inode*	*Shared Key*	*IV*

1.3 Creating a new User(InitUser)

1. Name and Password to be provided by the user.
2. A 32 byte Argon-Key is generated by using Username,Password provided by user and 16 byte of it is stored in UDS.
3. RSA key-pair is generated and public key is stored in key-store server.
4. User Name , Password , Private key , Argon-Key and IV is stored in UDS.
5. To provide integrity we will calculate HMAC of UDS and append it after bytes of UDS.
6. Now we will encrypt UDS+HMAC using Argon-Key and create HMAC of the output and store this HMAC in data store , with key as UUID generated from 32 byte argon key.
7. We will store encrypted UDS+HMAC in data store using another UUID generated from 32 byte Argon key.

1.4 GetUser

1. A 32 byte Argon-key is generated by provided username and password.
2. Using this Argon-key we will encrypt User Name to get our key , using this key we will get UDS+HMAC from Data-Store Server.
3. Now HMAC of this UDS+HMAC is calculated and compared with the HMAC stored in data-store.
4. Now this UDS+HMAC is decrypted using Argon-Key , and to verify integrity we will calculate HMAC from UDS and compare it with given HMAC.

5. If verification succeded we will return UDS.
6. If at any stage any error occurred we will throw some error.

1.5 StoreFile

1. First we will verify whether the file size is in multiples of blocksize or not , if not we will throw an error.
2. To provide functionality of having two different user same file name we are adopting the naming convention like UserName.FileName.inode. We are using the same naming convention for each data blocks also.
3. We will break file into blocks and calculate HMAC for every block. Now we will append HMAC after the bytes of data-blocks itself.
4. We will create an entry of this file in File Record Structure(we are creating a seperate file record structure corresponding to every file).
5. Now we will encrypt inodes and each data block individually using Argon-Key+Counter stored in UDS.

1.6 AppendFile

1. First we will verify whether the appended data is in multiples of blocksize or not , if not we will throw an error.
2. We will divide appended data into blocks , calculate HMAC for each blocks , and create entry in respective inodes (as we were doing with StoreFile).
3. Now we will encrypt only the appended blocks with Argon-Key+Counter.

1.7 LoadFile

1. Using inode-location persent in file record we get inode.
2. From inode we get to the location of datablock just by seeing offset.
3. Now we will get data block , decrypt it with Argon-Key+Counter , calculate HMAC of decrypted block and compare it with HMAC(which was founded). If HMAC found equal we will return decrypted data-block.
4. If at any stage any error occurred we will throw some error.

2 Part II

2.1 ShareFile

1. LoadFile,decrypt it and encrypt it using randomly generated 16 byte key which is called as shared key , finally store it back.
2. Make entries in sharing record and update entries in file record (like shared flag and sharing record).
3. Calculate HMAC of sharing record and encypt sharing record + HMAC using public of user with whom we are sharing.
4. This encrypted object is our message that we will sent to user.

2.2 ReceiveFile

1. Now reciever will decrypt message compare HMAC of sharing record with given HMAC provided inside message.

3 Part III:RevokeFile

1. To revoke we will load file , decrypt it using shared key , then encrypt it using Argon-Key+Counter ,and store it back again.
2. We are also changing the location of inode , to make sure that revoked user will not be able accesss even the encrypted file.

4 File-Organization

For each file we are creating an inode index having 800 entries(as our file size limit is 800 blocks). In these entries we are storing the location(keys) of data-blocks stored in data-store. We are encypting our inode using Argon-key.