Design Document

Arunkumar Vediappan (18111009), Supriya Suresh (18111077)

March 5, 2019

1 Problem 1

Symbols used : H() - SHA256 Hash Function, $\|$ - concatenation, un - username, pw - password, userDS - user Data Structure, [K:V] - Key Value pair, $E_k(X)$ - X encrypted using key k

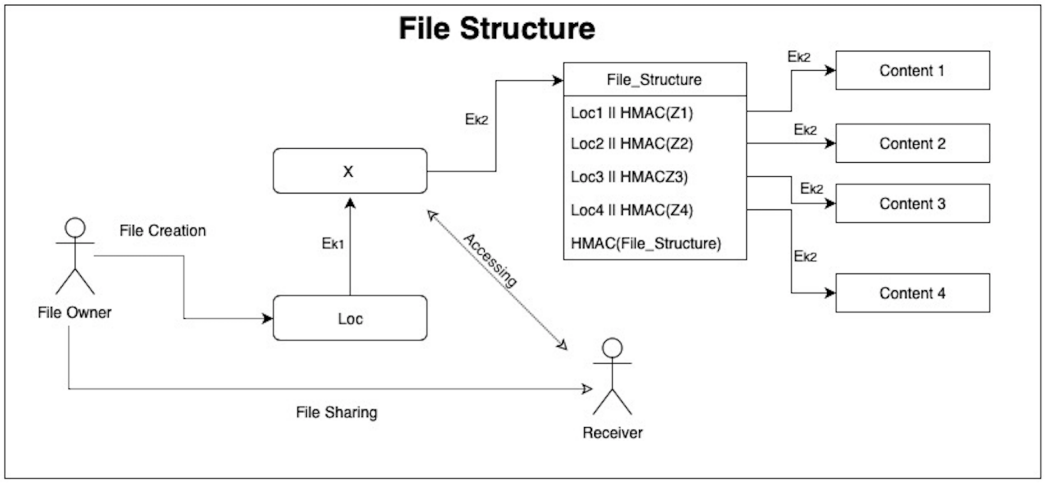
1.1 InitUser

- Compute **H1 = H(**$un\|pw$**)** and Generate RSA public private key pair**(pubKey, privKey)**
- Store pubKey in Keystore - [**un : pubKey**] and store privKey in userDS. The assumption here is that keystore is trusted so we have directly used username in plaintext form.
- Generate AES Key E_k **= Argon(pw, un)** and encrypt userDS with this key and place it in Datastore. - [**H1 :** E_k**(userDS + HMAC)**] The assumption here is that keystore is trusted so we have directly used username in plaintext form.

1.2 GetUser

- Compute **H1 =H(**$un\|pw$**)** and if this exists in the datastore map as a key in a key-value pair, implies that verification is successful. Else return error.
- Compute E_k (as done in InitUser). Decrypt value at the location H1 in the datastore using this key. Verify HMAC.
- Load userDS if verification is successful. Else return error.

1.3 StoreFile



New File Creation

- Generate two random numbers from /dev/random
- Compute **Loc = H(r1)** and generate AES Key E_{k1} = **Argon(pw, r2)**. E_{k1} is used to encrypt the actual location of the filestructure X.
- Generate actual location of filestructure X = H(r_2) and place its encrypted version in the data store, [Loc : E_{k1}(X+HMAC)]
- Generate AES Key to encrypt FileStructure E_{k2} = **Argon(X,**E_{k1}**)** and place it in data store. Both file content and file structure is encrypted using E_{k2}.
- Store filename and E_{k1} in userDS. Also make this change in datastore's version of userDS.

File already present

- Get Loc and E_{k1} from UserDS. Access Loc and Verify HMAC.
- Generate E_{k2} (as done in StoreFile) and use this to decrypt value at X to get FileStructure. Verify HMAC. Return error if unsuccessful.
- Change file content to new content passed in StoreFile function. Encrypt it and place it back in datastore.

FileStructure is actually a table of locations, each location is generated by H(some random number). Nth location in this table contains Nth append to the file, the contents of which are also encrypted using the same key E_{k2}. In the figure, HMAC(Z_i) refers to HMAC(Content) at Loc(i).

1.4 LoadFile

Load File follows the same steps as above to get the location of FileStructure. It goes through each Loc_i to get the content of the entire file, verifies HMAC(FileStructure) and also HMAC(Content) in each location. If unsuccessful, it returns nil.

1.5 AppendFile

In Append File, since the user has E_{k2} derived using the above mentioned steps, he decrypts FileStructure and verifies HMAC(FileStructure).If verification is successful, he adds a new row with the hash of a new Loc_i in the table where the appended content that user sent will be placed encrypted by E_{k2}. HMAC(FileStructure) is modified. There is a field of number of appends in the file which is saved in UserDS which is incremented.

2 Problem 2

2.1 ShareFile(f1,"B")

In "sharing" string we share loc and E_{k1} (symbols have their usual meaning as above). It is encrypted using receiver's public key, signed using sender's private key. Send it to B. Add the fact that you shared this with B in sender's userDS.

2.2 ReceiveFile(f2, "A", sharing)

- Decrypt "sharing" using B's RSA private key to get loc and E_{k1}. Verify signature using A's public key.
- Save Loc, E_{k1} and f2 to B's userDS and to its copy in datastore. So B can access the file under name f2.

2.3 RevokeAccess

In order to revoke access the key E_{k1} is changed by generating a new random number $r2'$. This will generate new X' and change E_{k2} as well. Use original X and decrypt filestructure using original E_{k1} and verify HMAC. If verification is successful, encrypt the file content using new E_{k2} and place it at new location X'. Change the value at Loc to the encrypted form of X' using new E_{k1}. Make all the modifications required in the userDS.

Note : We are assuming that the username and password entered by the client are being passed through a secure channel in the call to InitUser().

3 Modifications

- Generaton of AES Key E_{k1} has been revised to E_{k1} = **Argon(pwhash, r2)** , where pwhash corresponds to H(pw + salt). Salt is randomly generated on the first call to Init User.
- Number of Appends to a file is stored in the File Structure itself.