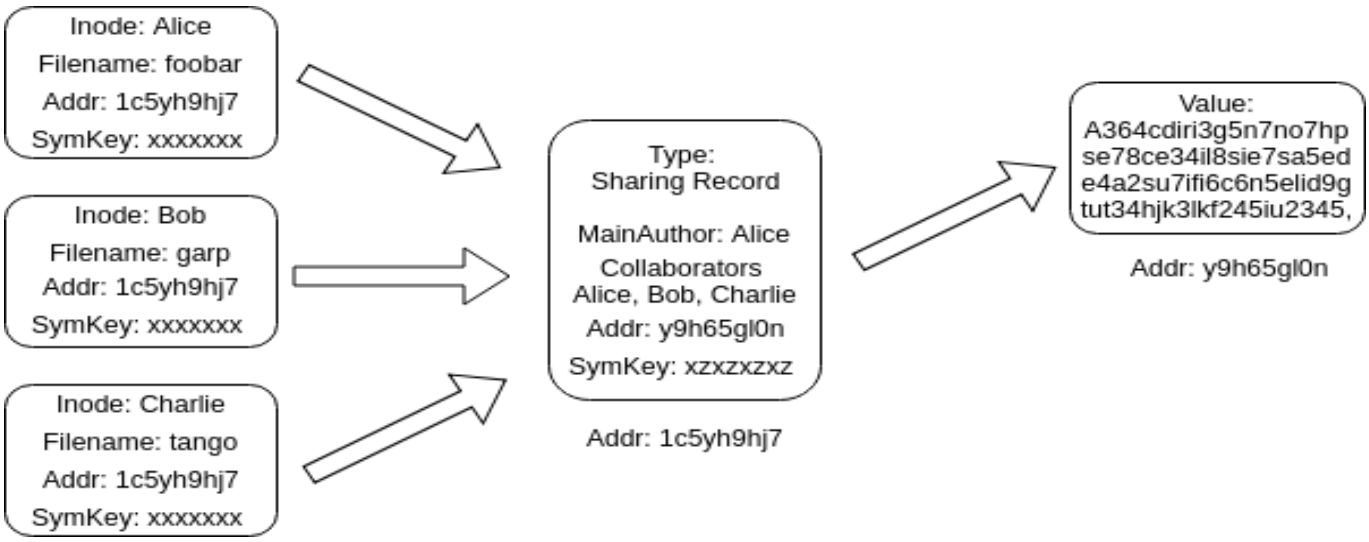
CS628 Assignment 1

2018-19 **II Semester**

Design Report for Secure Key-Value File Sharing



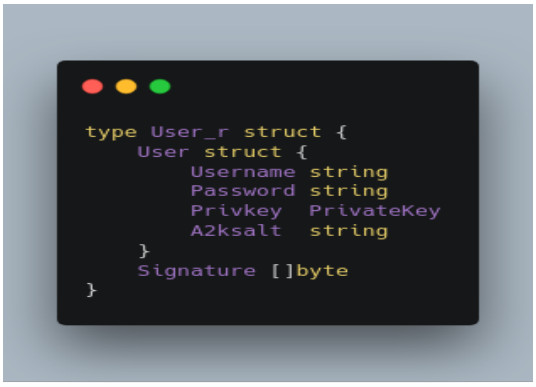
Aniket Pandey (160113) · Ashish Kumar (160160)

February 13, 2019

1 A simple, but secure client

Main Purpose: To maintain Confidentiality and Integrity of data without any regards to Availability.

USER Structure



INODE Structure

1.1 InitUser (username string, password string)

1. Obtain $key = Argon2Key(" < username > +user"," < password > +pass", 10)$. This will generate a 36 character key. This is where the value (User data) will be stored.
2. Generate an RSA Key-pair. Push the public key to the Public Key Server. Private key stored.
3. Obtained the HMAC signature of User struct. Encrypt the json-marshalled User_r struct data with AES-Cipher Feedback Mode. The "key" for CFBEncrypter() is obtained via Argon2key(), with salt="<password>+ssap" while password is same as above.
4. Call $DatastoreSet(key, ciphertext)$ to publish the encrypted User information in Data Server.

1.2 GetUser (username string, password string)

1. Get the "key" using above Argon2Key invocation. Errors suggest either incorrect username or password. Calculate the key for CFBDecrypter(), and obtain the decrypted User_r structure.
2. Check HMAC hashes of $User\_r.User$ and $User\_r.Signature$ for any tampering. **Confirm** if the passwords match.
3. If all checks are satisfied, return the $User\_r.User$ structure.

1.3 (User) StoreFile (filename string, data []byte)

1. Obtain $key = Argon2Key(" < username > + < filename > "," < password > + < filename > ", 10)$. This will generate a 36 character key. This is where the Inode structure for "Username"-"Filename" will be stored (Refer to the figure).
2. Generate a random address (key) and AES-CFB key for storing and encrypting SharingRecord_r structure respectively. Fill the Inode structure. Sign it using Author's Private key and store RSA signature in $Inode\_r.Signature$. Encrypt $Inode\_r$ with Author's Public key. Push (key, encrypted Inode_r) to Data Server.
3. Initialize a SharingRecord_r structure. Fill up all values. Add original author to Collaborators. **NOTE:** The List of Collaborators is most important construct in this design. Explained ahead.
4. Again generate a random address (key) and AES-CFB key for storing and encrypting the $Data\_r$ structure. Store these in the relevant fields of SharingRecord structure. Sign with a predecided HMAC key and store in $SharingRecord_r.Signature$ field. Encrypt the Structure with the key decided at $Inode_r$ and push to Data Server.

5. Store the "data" at The key generated above, Encrypt it with CFBEncrypter() method. Sign it and store the data at the "key". Push to Data Server and return.

SharingRecord Structure

User Structure

1.4 (User) LoadFile (filename string)

1. Follow the method given in StoreFile() to reach, decrypt and verify the signature of the Inode structure corresponding to "filename".
2. **IMP:** Check if $User_r.User.username$ is in Collaborators. If so, continue following the method in StoreFile to get to the Data location, decrypt the data, verify the signature and return it.

1.5 (User) AppendFile (filename string, data []byte)

1. Follow the method given in LoadFile() to reach, decrypt and verify the signature of the $Data\_r$ structure corresponding to "filename".
2. **NOTE:** Using the property of Cipher Feedback Mode, we don't need to decrypt the entire data for append operation. Instead, using the stored CipherText as Initialization vector, we can continue the encryption operation more efficiently.
3. Recalculate the signature and push the modifed data to Data Server.

2 Sharing and revocation

2.1 (User) ShareFile (filename string, recipient string)

1. Follow the method given in LoadFile() to reach, decrypt and verify the signature of the $SharingRecord\_r$ structure corresponding to "filename". **IMP** Verify if the user is present in Collaborators.
2. Collect the address and AES-CFB encryption key of $SharingRecord\_r$ structure from the respective Inode. Recieve the Public key of receiver from the Public Key Server.
3. $sharing = PubKey_{recipient}(PrivKey_{user}(Hash(CollectedInfo)) + (CollectedInfo))$, to maintain confidentiality and integrity in case of a **Man in the Middle attack** while sharing the message offline.

2.2 (User) ReceiveFile (filename string, sender string, msgid string)

1. Decrypt "msgid" using Private Key of User, verify the integrity using Public Key of Sender. Obtain the Address and "key" for CFB-Decryption of SharingRecord structure of the concerned data(value). **IMP** Check if the sender is in Collaborators. If so, proceed ahead.
2. Create an Inode for the receiver user using Argon2Key, with the method described in Store-File(). Store the info in the Inode, store the RSA signature and encrypt $Inode\_r$ structure with Public key of User. Return.

2.3 (User) RevokeFile (filename string)

1. Go to the SharingRecord structure corresponding to "filename". **IMP** After verifying the integrity, check if the User is **Original-Author** of the "filename".
2. If so, clear the list of collaborators except the original-author. From the Inode of original author, change the encryption key of SharingRecord structure and re-encrypt it with a new key. **NOTE:** This is to prevent any further invocation of "RevokeFile()" method by a distrusted user.