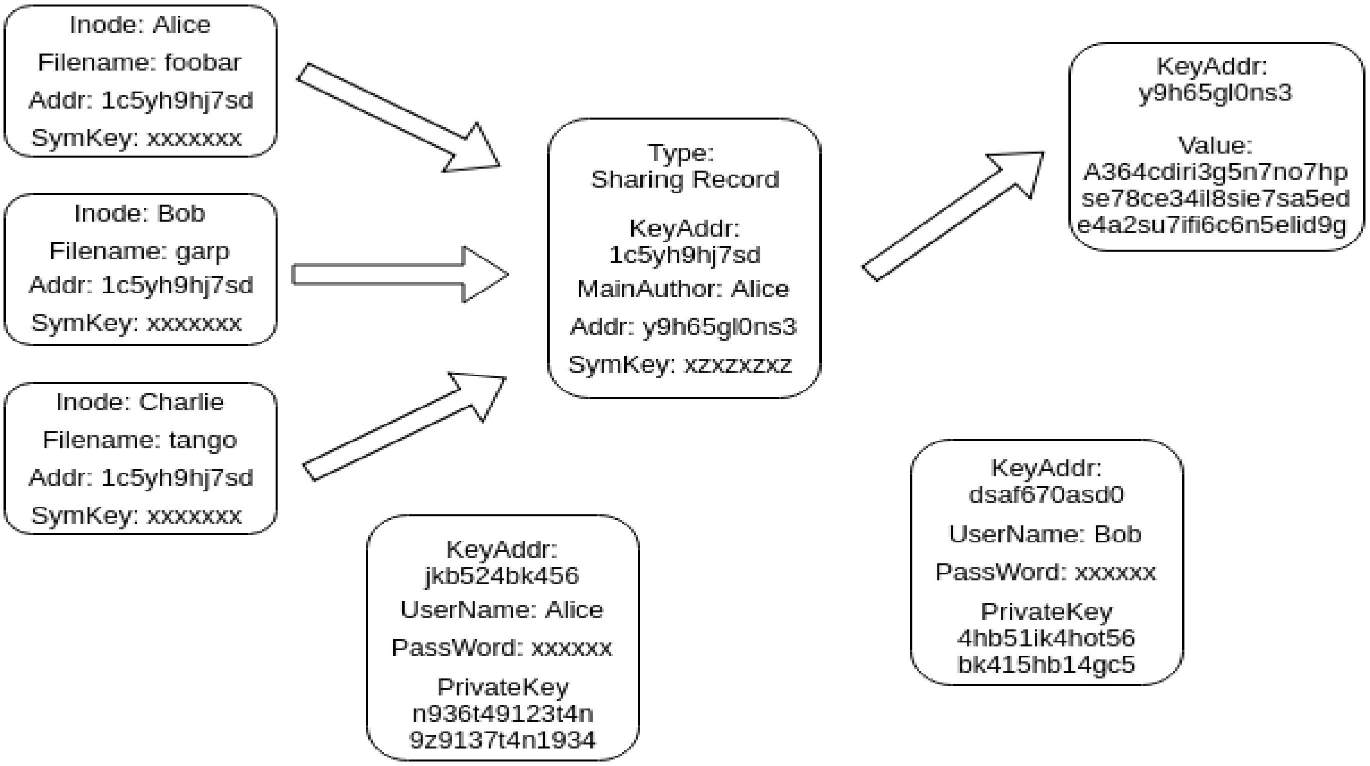
# CS628 Assignment 1
## 2018-19 **II Semester**

# Design Report for Secure Key-Value File Sharing



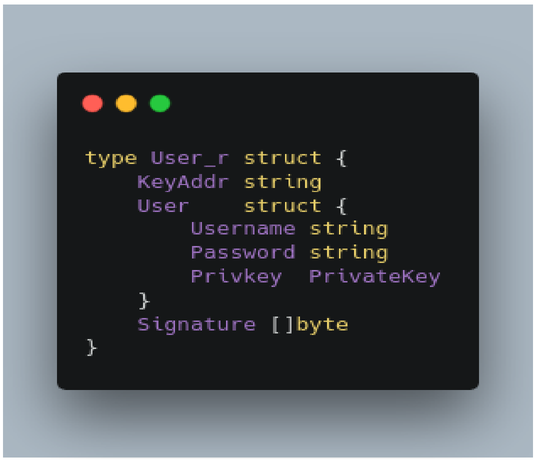
Aniket Pandey (160113) · Ashish Kumar (160160)
*B.S MTH* *B.Tech CSE*

February 15, 2019

# 1 A simple, but secure client

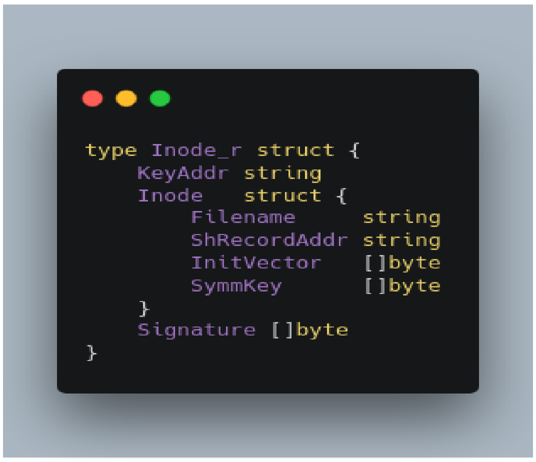
**Purpose:** To maintain Confidentiality and Integrity of data without any regards to Availability. **NOTE:** *KeyAddr* field in every struct to protect against key-value-swap attack.

**USER Structure**



```
type User_r struct {
    KeyAddr string
    User    struct {
        Username string
        Password string
        Privkey  PrivateKey
    }
    Signature []byte
}
```

**INODE Structure**



```
type Inode_r struct {
    KeyAddr string
    Inode   struct {
        Filename     string
        ShRecordAddr string
        InitVector   []byte
        SymmKey      []byte
    }
    Signature []byte
}
```

## 1.1 InitUser (username string, password string)

1. Obtain $key = Argon2Key('' < password > + < username > '','' < username > +user'', 10)$. 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. Store the Private Key along with other fields in User_r struct.
3. Generate a new $key = Argon2Key('' < username > + < password > '','' < username > '', 10)$ for symmetric encryption. Use this key to sign $User\_r.User$ with HMAC scheme, store the signature in $User\_r.Signature$. Encrypt the json-marshalled $User\_r$ struct data with AES-Cipher Feedback Mode using the same key as above.
4. Call $DatastoreSet(key, ciphertext)$ to publish the encrypted User information in Data Server.

## 1.2 GetUser (username string, password string)

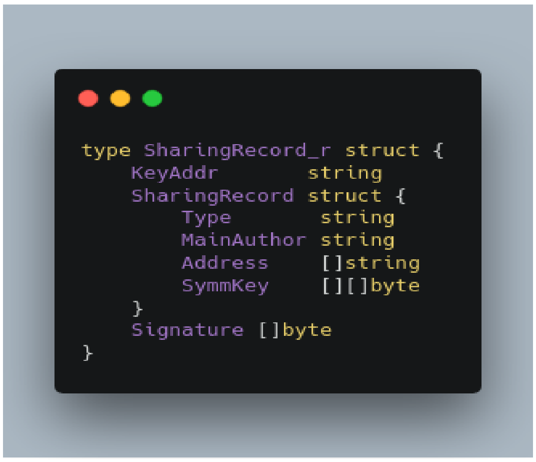
1. Get the "key" using above Argon2Key invocation. Errors suggest either incorrect username or password. Calculate the key for CFBDecrypter(), and obtain the decrypted $User\_r$ structure.
2. Check HMAC hashes of $User\_r.User$ and $User\_r.Signature$ for any tampering. If all checks are satisfied, return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

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

1. Obtain $key = Argon2Key( < password > + < filename > '','' < username > + < filename'', 10)$. 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. Check if a $SharingRecord\_r$ structure exists. If so, delete all the existing data blocks and write the new data in iterations. Else, initialize a new $SharingRecord\_r$ object.

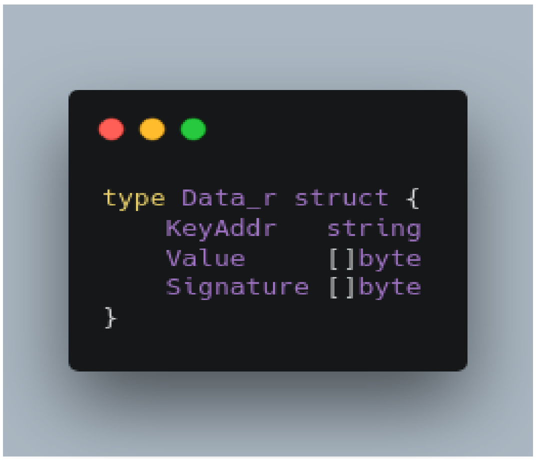
4. Again generate a random address (key) and AES-CFB key for storing and encrypting the *Data_r* structure. Store these in the relevant fields of SharingRecord structure. Sign with a predecided HMAC key and store in *SharingRecord_r.Signature* field. Encrypt the Structure with the key decided at *Inode_r* 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**



```
type SharingRecord_r struct {
    KeyAddr       string
    SharingRecord struct {
        Type       string
        MainAuthor string
        Address    []string
        SymmKey    [][]byte
    }
    Signature []byte
}
```

**DATA Structure**



```
type Data_r struct {
    KeyAddr   string
    Value     []byte
    Signature []byte
}
```

### 1.4 (User) LoadFile (filename string)

1. Follow the method given in StoreFile() to reach, decrypt and verify the signature of the *SharingRecord_r* structure corresponding to "filename".
2. Loop over the list of addresses of data chunks (via indirect pointers), decrypt and verify the HMAC signatures of each, reconstruct the entire data as a single byte array 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 *SharingRecord_r* structure corresponding to "filename".
2. Create a new *Data_r* block and append its generated encryption key and address to the appropriate list of keys and signatures in SharingRecord structure. Push the block to DataStore and return.
3. **NOTE:** We are not verifying the integrity of previous data blocks during AppendFile(), considering the unnecessary overhead of re-encryptions/decryptions.

## 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".
2. $collected\_info = Inode\_r.Inode.(Symmkey + ShRecordAddr)$. Recieve the Public key of receiver from the Public Key Server.
3. $sharing = PubKey_{recipient}(PrivKey_{user}(Hash(collected\_info)) + (collected\_info))$, to maintain confidentiality and integrity in case of a **Man in the Middle attack** while sharing the message offline. Where $PrivKey_{user}(Hash(collected\_info))$ is the signature of $collected\_info$.

## 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) and proceed ahead.
2. Create an Inode for the receiver user using Argon2Key, with the method described in StoreFile(). 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”. If not, return an error.
2. If so, from the Inode of original author, change the encryption key and the address of $SharingRecord\_r$ structure and re-encrypt it with a new key. Similarly, change the address of the actual data ($SharingRecord\_r.Address$).
3. Iterate over all data-blocks and re-encrypt them with fresh symmetric keys. Also, store each of them at new addresses. Store these new keys and addresses in the corresponding $SharingRecord\_r$ structure. **NOTE:** This is to prevent any further misuse by a distrusted user who knows the original keys and addresses of data blocks.