CS628 Assignment 1 Design Document

Arpit Gupta(160149) Hritvik Taneja(160300)

February 2019

Introduction

```
struct User {
    Username string;
    Key string; //To encrypt MetaFile struct
    Password_Hash string;
    RSA_Private_Key string;
    UUID uuid;
}
```

Listing 1: User Data Structure

```
struct MetaFile {
    Key string; //To encrypt File struct
    File_Pointer string; //Pointer to the corresponding File struct
}
```

Listing 2: File PreMetadata Data Structure

```
struct File {
    Type string; //whether this is a file or just the metadata
    Owners_UUID uuid;
    Key string; //To encrypt file data
    Data_Pointer string; //The head of the file data
}
```

Listing 3: File Metadata Data Structure

```
struct FileData {
    Type string; //whether this is a file or just the metadata
    Value string; //encrypted data
    Length int; //whether this file is shared or not
    Next_Pointer string; //Hash to the next pointer
    Next_Key string; //encryption key for the next block in the chain
}
```

Listing 4: FileData Data Structure

Question 1

Property 1: InitUser(username string, password string)

- Check for duplicate user using KeystoreGet(username).
- Generate RSA key pair.
- Populate the User data structure.
- Use `hash(username + password)` as the key for this User in Datastore.
- Use `password` as the encryption key for this User structure.

Property 2: GetUser(username string, password string)

- Use `hash(username + password)` as the key in DatastoreGet.
- If nothing is found return `nil`.
- Otherwise decrypt using the password.

Property 3: (userdata* User) LoadFile(filename string)

- Use `hash(userdata.uuid + filename)` as the key in DatastoreGet to get the encrypted `MetaFile` structure.
- Decrypt this using `userdata.Key`
- Go to `MetaFile.File_Pointer` and decrypt using `MetaFile.Key` to get the `File` data structure
- If `File.Type` is not "shared" use `File.Key` to decrypt the data present at `File.Data_Pointer` and traverse the linked list and decrypt along the way using `FileData.Next_Key`, if needed.

- If `File.Type` is "shared" use `File.Data_Pointer` as the key in DatastoreGet to get the shared encrypted File structure.
- Decrypt this shared encrypted file using `File.Key`
- Once decrypted use similar technique to extract data as used above.

Property 4: (userdata* User) StoreFile(filename string, data []byte)

- Use `hash(userdata.uuid + filename)` as the key to store `MetaFile` data structure in DatastoreGet.
- Populate the `MetaFile` data structure and encrypt this using `userdata.Key`
- Use `MetaFile.File_Pointer` as the address to store `File` data structure.
- Populate the `File` data structure and encrypt this using `MetaData.Key`
- Use `File.Data_Pointer` as the address to store `FileData` data structure.
- Populate the `FileData` data structure and encrypt this using `File.Key`

Property 5: (userdata* User) AppendFile(filename string, data []byte)

- Read the `MetaFile` and `File` data structures as done in LoadFile.
- Make a new `FileData` node and place it at the head of the linked list.
- Store the encryption key of the previous head(`File.Key`) in new `FileData.Next_Key` and Data Pointer of the previous head(`File.Data_Pointer`) in new `FileData.Next_Pointer`

Question 2

Property 6: (userdata* User) ShareFile(filename string, receiver string)

- First decrypted `MetaFile` data structure and pack this data structure in an object.
- RSAEncrypt this object using the public key of the receiver from KeyStore
- Sign this encrypted object with senders private key.
- Send the encrypted object and the signature as a marshalled object.

Property 7: (userdata* User) ReceiveFile(filename string, sender string, sharing string)

- Verify the signature using senders public key from KeyStore.
- RSADecrypt message using receivers Private Key to get an `object` of type `MetaFile`.
- Otherwise Populate a new `MetaFile` and `File` data structure with `File.Type = 'sharing'`, `File.Data_Pointer = object.File_Pointer` and `File.Key = object.Key`

Question 3

Property : (userdata* User) RevokeFile(filename string)

- Load and decrypt the `MetaFile` and `File` data structures.
- Generate a new `MetaFile.Key` and encrypt `File` using this new key.
- Save both the modified data structures back.

Integrity check for all the data structures

Before encrypting any of the data structures(`User, MetaFile, File, FileData`), we should double hash it and store it in the same key by prepending it to its value and, whenever we load the value for any key we should check whether the double hash of the data structure after decryption is equal to this prepended hash.