Design Report: PA1

CS628

Aditya Rohan(160053)
Paramansh Singh(160474)

InitUser: InitUser function initiates the User with at least the password and one private-public key pair. The username acts as a salt for the password which is then hashed and stored. Then this hash can act as a symmetric encryption key to store user details. Hash of username is also stored in the user struct.

GetUser: The GetUser function checks for the correctness of the username and the password at first. An error is raised is either of them fail to match, if the username and password are indeed correct the function proceeds to verify the checksum of credentials of the user, stored in the user struct along with other credentials. If the stored checksum doesn't match the freshly computed checksum, then the user data is either corrupted or has been tampered with, in either case we raise an error.

StoreFile: We maintain a dictionary which maps each filename to a random ID which is generated when the user first calls StoreFile. The location of the file is determined by this ID. This dictionary is itself encrypted and HMAC is used.
The user generates keys which are used to encrypt and decrypt the user's files. All the keys are further encrypted with the user's public key and an RSA signed message can be appended to ensure authenticity. This way any other person can't know the keys and hence the file contents unless shared with.
Every time a file is uploaded a new set of keys are also associated along with them. The file is encrypted using one of the keys and the checksum computed using HMAC is also appended to the file. The key used in HMAC is the file-id, which is unique and hence if the adversary swaps the (file, checksum) with some other file, we will get an error as the HMAC is computed using different keys.

LoadFile: The loadfile function checks for the integrity of the file in question and it also checks if it exists or not. For integrity check, we compute the checksum of the file data using the HMAC API. Now, whenever the file is accessed the appended checksum is matched with a freshly calculated checksum to verify the integrity of the file. The existence of the file can be verified using the array/list of the files owned by any user. If a user doesn't have a particular file in this list then he isn't allowed to access the file anyway.

AppendFile: The AppendFile function works along the lines of the inode structure present in normal filesystems. Each user maintains a list of files he has access to, along with this they also store the pointer to the file; And the size of the file after every append is stored in a list in the file itself. So whenever someone wants to append to the file they get the size of the file from the last element of the list, use that as an offset and append to that offset from the pointer and update the size of the file. The reason we're storing the size after every is that whenever someone does a loadfile, they decrypt the part after every size change and get the contents of the file. We also maintain a checksum of this list so that we can identify any cases of tampering with this list.

Sharing: Each file has an associated list of users (hashed usernames) who have access to the file. For performing any operations on the file and its associated contents (i.e. even modifying this list), a user's name must be in this list. This list will be encrypted (with the same key as the one used in file operations) and stored along with the HMAC of this list to ensure confidentiality and integrity.
When the owner or some other collaborator decides to share the file, he encrypts using the receiver's public key and shares file-id, the keys used to decrypt the files, and signs the contents with his private key. He also adds the name of the receiver in the list of allowed users.

Revocation: When the owner calls RevokeFile, the user modifies the list of allowed usernames corresponding to the file and removes names of all other collaborators, and updates the checksum. In this case, even if a non-owner user calls RevokeFile the list of allowed users is updated and all the other users (including the owner) can't access the file. After the revocation, the file-id is randomly generated again in order to potential attack by users who know the file-id.

Testing:

- Data Integrity Check: This checks whether LoadFile gives proper error messages.
  The adversary makes changes in the file data -> Error: checksum failed.
  The adversary replaces <data, checksum> with <data2, checksum2>
  Again checksum error since different HMAC keys as no unauthorised person has access to the file-id.
- File Sharing: Integrity - The adversary modifies the data shared. The signature of the sender's private key results in an error if some contents have been modified.
  Confidentiality - Signed using receiver's public key and hence no one else can decrypt.