CS161 Project 2 Final Design Document

Charlotte Chen

August 4, 2025

Overview

This document outlines the design for a secure file sharing and storage system that satisfies the requirements described in the Project 2 specification. The design provides user authentication, secure file storage and append, file sharing, and revocation, while defending against both the Datastore Adversary and the Revoked User Adversary.

1. Data Structures and UUID Strategy

The local data is stored as UUID and marshaled data pairs. While UUID is derived from either random generation or a deterministic key derivation with HashKDF. Confidentiality is preserved through symmetric encryption and public-key encryption, and integrity is verified by HMAC and digital signatures.

2. File Storage and Metadata

In the Datastore, each file is represented by a `file pointer`, which includes a UUID pointing to the file metadata, a flag indicating whether the file is shared from another user, a root hash key used to derive the file key, and related pointers used to track recipients and help recipients decrypt the file. `FileMetadata` is the central node for each file, which contains head and tail chunk UUIDs that point to a list of file chunks. `FileChunk` is the basic unit of file storage, containing the encrypted content and a pointer to the next chunk.

3. User Authentication

User authentication is handled by deriving the root key from the password and using it to access encrypted user data. The root key is derived from the password using `Argon2` with a combination of username and password as the input. The user data is stored in a `User` struct, which includes the user's public and private keys, and a root key randomly generated for file pointer encryption.

4. Multiple Devices Synchronization

All persistent user and file data resides in Datastore. Thus, all `User` objects generated via `GetUser` will pull lastest data from Datastore. Changes on one device are immediately reflected on another as they reference the same UUIDs.

5. File Storage and Retrieval

Each user maps a local filename to `FilePointer` (encrypted and HMAC'd with user root key). `FilePointer` then points to the file metadata UUID. (encrypted and HMAC'd with file root key). For all shared files, while the file pointer is encrypted with the user's key, the file metadata and file nodes are encrypted with the file keys that are available upon invitation acceptance and are needed to be downloaded each time the file is accessed.

6. Efficient Append Bandwidth

AppendToFile downloads only the tail chunk and appends a new encrypted chunk. It uploads a file node containing the new chunk and updates the file metadata with the new tail chunk UUID. On the other hand, it downloads the file metadata, tail chunk, and a shared information pack only if the file is shared. Thus, the total bandwidth scales only with the size of appended content (n + small constant).

8. File Sharing Semantics

On `CreateInvitation`, the sharer:

- Creates a new `SharedInfoPointer` struct, which points to `SharedInfo`, encrypted and signed with recipient's public key and sign key respectively and `SharedInfo`, which contains the location of the file metadata and the file key, encrypted with a unique file key generated by the sharer.
- Stores a new recipient struct containing the location of the share information pack and the name of the recipient. The recipient struct is then linked to `RecipientListTail` at the `FilePointer`.

Recipient retrieves and decrypts the invitation in `AcceptInvitation`, access the `SharedInfo`, and store the encryption key, HMAC key, and file metadata UUID in the `FilePointer` created for the file with share flag set to true.

9. File Revocation Mechanism

- Owner re-encrypts all file chunks with a new file key
- Generates new UUIDs for each chunk and metadata
- Updates metadata and access list, invalidating old passwords and `SharedInfo` structs
- Copy the recipient list, update `SharedInfo` struct for remaining users, and regenerate new file root key.

Revoked users cannot access new data due to loss of valid keys and UUIDs. They cannot observe updates since UUIDs and encryption keys change.

Note: A visual diagram of these data structures and their relationships will be included in a separate page.

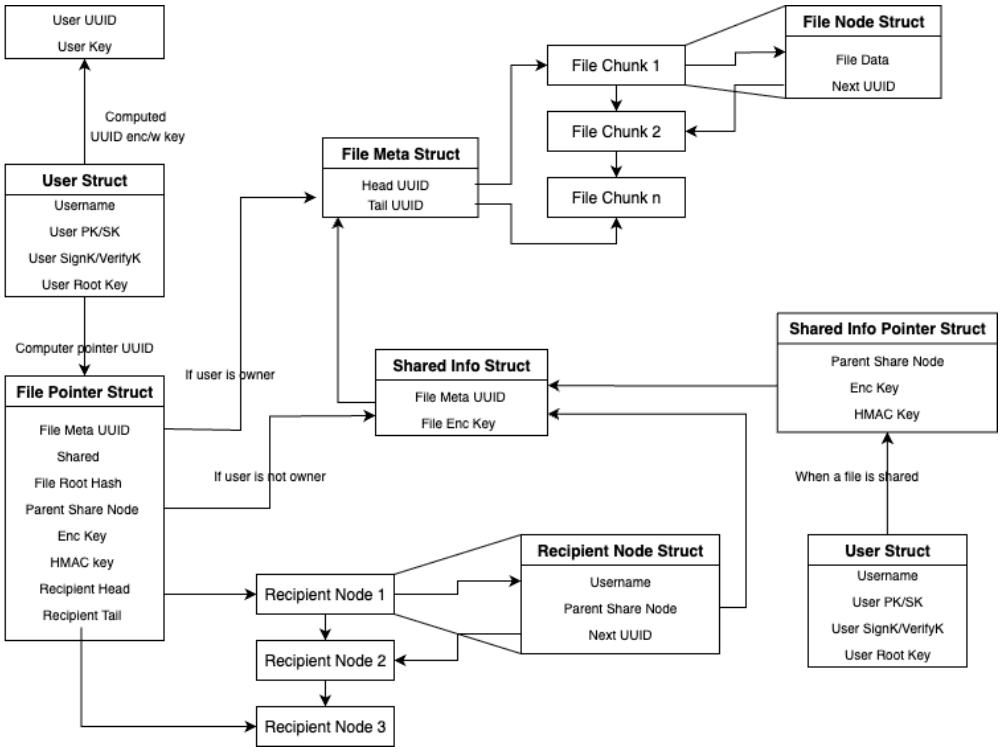

Figure 1: Data Structures Overview