CS 161 Project 2 Design

Devun Amoranto

April 21, 2023

1 Data Structures

Basic Constants (Global) and Other Important Notes on Implementation

- SIXTEEN_BYTES = 16: A common length for IVs and randomly generated keys
- BLOCK_SIZE = 128: The maximum number of bytes stored in the 'Content' field of a file struct.
- "User derived key" will refer to any key that is derived from a user's password. This is done by invoking Argon2Key on the password, and HashKDF for any new keys that must be derived. Keys may be derived from other derived keys, as all keys are deterministically derived on-the-fly,
- To ensure the integrity and authenticity of stored structs, ANY time a value is pulled from Datastore, its signature is checked against the signing user's public key, and an error is returned if tampering has occurred.

User Authentication

```
type Credentials struct {
    #Public Information
    Username string #username
    Password []byte #salted slow hash of user password

    InvitationPK UUID #uuid of public key for encrypting Invitations
    FileSharePK UUID #uuid of public key for sharing file keys.
    SignaturePK UUID #uuid of public key for verifying the
        signatures of this user.
    UserStruct UUID #uuid of user struct in datastore

    #Private or semi-private information
    PasswordSalt []byte #salt for the password hash
    InvitationSK []byte #encrypted SK for InvitationPK
```

```
    FileShareSK []byte #encrypted SK for FileSharePK.
    SignatureSK []byte #encrypted SK for SignaturePK

    Signature []byte # hash of struct contents, signed with
        signatureSK
}

A public directory card for a user, accessible at any point in time
    by any user. Used to verify login information and store
    encrypted RSA keys.
```

The User struct

```
type User struct {
    #serialized values
    Username string #username
    pbk []byte #PBK of user
    Signature []byte #signed hash over struct contents
}

The user struct contains the user PBK for convenience, but is never
    stored in datastore.
```

Basic file storage.

```
type FileDesc struct {
    #metadata
    Identifier #HMAC of username and filename
    Parent string #Who invited this user, if applicable
    Children [string] #who this user has shared this file with

    #fields for access
    FileLocation UUID #uuid of file struct for this file
    Signature []byte #signed hash over the struct
}

The FileDesc struct is per-user, and contains information on files
    in a single user's space, and is only relevant to them

type File struct {
    FirstBlock UUID #first block of file content
    LastBlock UUID #last block of file content

    Owner string #who created this file in the first place
    AuthorizedAccess [SharedFileAccess] #see SharedFileAccess
    Integrity #HMAC over struct contents, using shared key
}

The File struct is shared among all users, and stores encrypted
    shared keys for each user.
```

```
type Block struct {
    Content string #Up to 128 bytes of encrypted content
    Prev UUID #previous block
    Next UUID #next block
    Integrity []byte #hmac over encrypted file contents, using
        shared key
}

File contents are stored as a doubly linked list of blocks, with
    content integrity being recorded on a per-block basis.

type SharedFileAccess {
    Username string #the corresponding user for this entry
    Children []string #list of users directly shared with
    DecryptionKey string #shared encryption key, encrypted with
        user's FileSharePK
    MacKey string #shared mac key, encrypted with user's FileSharePK
    Valid bool #usually true, but only set to false by Inviting
        users. (Set to true upon accepting invite).
    Signature []byte #signed HMAC over struct contents
    SignedBy string #the user whose SignatureSK was used to sign
        this struct. The signer must have valid access to the file.
}

The SharedFileAccess contains per-user credentials for accessing
    file contents.
```

User invitations.

```
type Invitation struct {
    EncKey []byte #symmetric key to decrypt payload
    Payload []byte #encrypted invitation contents (the juicy bits)
    Signature []byte #Signed with the inviting user's SignatureSK
}

The Invitation struct is a wrapper around a file sharing invitation,
    built to preserve the confidentiality and integrity of the
    actual payload's details

type InvitationPayload struct {
    Parent string #sharing user
    Child string #invited user
    FileLocation UUID #location of File struct
    SymKey []byte #symmetric key for the file
    MacKey []byte #hmac key for the file
}

The InvitationPayload struct contains relevant sharing information
    for the file, as well as the shared keys. There is no signature
```

```
over the InvitationPayload, because there is a signature on its
wrapper including the encrypted payload.
```

2 User Authentication

InitUser(*username*, *password*)
The Credentials Dictionary in Datastore is referred to as *credDict*. It is stored in Datastore under its UUID key as part of initialization.

- Derive a pbk from the user's password, salted. This increases entropy. Derive other keys from this pbk (this is done for every method). Also, generate random RSA keypairs.
- Create a Credentials struct for the user, checking that the user doesn't already exist. The Credentials struct will store the location of their public keys, as well as encrypted private keys that can only be decrypted with keys derived from the pbk.
- Sign off on the Credentials struct and store it in Datastore under a deterministic ID (Derived from a hash of "Credentials/Username").
- Create a User struct for this user, encrypt it, and store in datastore under a random ID. Return a pointer to this struct.

GetUser(*username*, *password*)

- Derive necessary keys, and obtain the relevant user credentials from Datastore. Error if none are found.
- Compute a pbk from *password* and $Credentials.PasswordSalt$, and assert that the two hashes match.
- Retrieve the user struct from Datastore, and decrypt it. Check the signature on the user struct, and return it.
- In this method, we let the attempted login generate keys from the provided password. If the password doesn't match, all the generated keys will result in failing signature checks, although the algorithm would already short circuit by then,

Additional notes on security:

- Multiple users with the same password will have unique derived keys because of *salt*. This also applies to empty passwords
- If an attacker determines the user's password, they are able to derive all keys that are derived from the password. However, they won't be able to tell that other users' passwords or derived keys.

3 File Storage and Retrieval

User.StoreFile($filename$**,** *content***)**

- Compute necessary keys from the User struct's pbk.
- Generate a unique file identifier by hashing the username with the filename, plus an extra unique string for file identifiers. Generate a deterministic UUID from this identifier.
- Attempt to find an existing file descriptor stored at the generated UUID.
- If such a file descriptor is found:
  - Store *content* as a linked list of *Blocks* stored at random UUIDS, and keep track of the first and last block.
  - Retrieve the File struct and shared keys for the file, and overwrite the $File.FirstBlock$ and $File.LastBlock$ fields with the uuids of the corresponding newly generated blocks.
  - Rehash the contents of the File struct and add a digital signature.
  - Store the File struct in Datastore
- If no file descriptor is found:
  - Generate a new FileDesc struct, and store it at the generated UUID. Also generate random shared keys for decryption and MACs for the new file.
  - Generate a new File struct, and store the content as above; initialize other fields, adding the current user to the Authorized User's list, as well as the encrypted keys.
  - Store the File struct in Datastore under a random UUID, and have $FileDesc.FileLocation$ point to it
  - Store the FileDesc struct in Datastore under the deterministically generated ID from above.

User.LoadFile(*filename*)

- Generate the file identifier as in $User.StoreFile$.
- Retrieve the corresponding FileDesc from Datastore, and the corresponding File struct. (Remembering to always check signatures)
- Find the user's SharedFileAccess struct within the File struct, and decrypt the shared encryptionKey and macKey. Return an error if they are not authorized to access this file (may be a revoked user).
- Retrieve the File's first block, and follow the linked list to reassemble content. First check the integrity of contents with macKey, then decrypt with encryptionKey.
- Return the reasssembled contents.

User.AppendToFile(*filename*, *content*)

- As above, generate the file identifier, retrieve the FileDesc and File struct from Datastore, and check SharedFileAccess is valid.
- Retrieve the LastBlock of content.
- Store new content in newly generated blocks (at randomly generated UUIDS), and update LastBlock so it points to the newly appended linked list blocks. Note that macs over content are only computed per block, so we only need to retrieve the final block to start things off.
- Update $File.LastBlock$ so it points to the new last block of content.
- Update the File struct, sign it, and update its datastore value.

4 File Sharing

User.CreateInvitation($filename$**,** $recipientUsername$**)**

- Check that all input is valid, and derive keys. Create an InvitationPayload struct with the sender, and recipient.
- Obtain the File Descriptor and File structs from Datastore corresponding to this file, and add the File struct's UUID to the InvitationPayload struct.
- Add the shared keys to the payload, and encrypt the payload with a randomly generated key. Encrypt the randomly generated key with the $InvitationPK$ for $recipientUsername$
- Add the encrypted payload to the wrapping Invitation struct, and add a signed hmac over the wrapper to protect integrity.
- Add the recipientUser to the list of Children for the File struct, and an INVALID AuthorizedAccess struct for the user (they change the VALID tag upon accepting). The "Children" list keeps track of invited users, but doesn't reflect users who have Accepted their invitations (their acceptance is reflected in the AuthorizedAccess list of the File). This is for the special case of inviting a user, then revoking them before they accept.
- Return the UUID of the wrapper Invitation struct.

User.AcceptInvitation($senderUsername$**,** $invitationPtr$**,** $filename$**)**

- Check that the inviting user still has access to the file. If not, short circuit and return an error (this user's pending SharedFileAccess struct will have already been deleted).
- Check the signature over the invitation with the PK for $senderUsername$. If this is the correct sender, their SK will be the key to the signature, and DSVerify will check out.
- Decrypt the invitation's encryption key with this user's InvitationSK, and decrypt the invitation contents.
- Create a file descriptor for $filename$ just like $User.StoreFile$, but instead have $FileLocation$ point to the location of the File struct given by the invitation.
- Find this user's entry in the AuthorizedUsers list, as given to them by their inviter. They will validate the entry upon accepting. Update the File struct with this user's valid entry in the AuthorizedUsers list.

5 File Revocation

User.RevokeAccess($filename$**,** $recipientUsername$**)**

- Find the corresponding FileDesc and File structs. Do a depth first search of the Children field of the SharedFileAccess struct $recipientUsername$, to compile a list of all names to be revoked.
- Nullify the SharedFileAccess struct for all revoked users, including structs with the VALID tag marked false (these are pending invites that are not yet accepted).

6 Helper Methods

These methods make it easier to implement some heavily-reused segments of code in the design.

- DeriveKeys(pbk []byte, creds *Credentials) -¿ Returns a list of keys derived from the pbk, and the decrypted secret keys from the credentials struct
- UUIDFromFileID, GenerateFileId, RetrieveCredEntry, FindFileById, etc -¿ Generic methods that create deterministic output, as well as helper methods to store and get specific structs from Datastore, and check their signatures,
  - Most of the helper methods in my code are like this, built to reduce the need for reduntant checks and easier code reusability. They aren't really algorithms.
- Sign, CheckSignature, etc –¿ Methods that marshal structs, hash them, and either compute signatures/hmacs or verify them against a signing key.
- StoreContent (And ReassembleContent) –¿ Store files in a linked list and return the UUIDS of the first and last block. ReassembleContent is the reverse of this
- RetrieveFileAndKeys –¿ Given the correct credentials (derived keys), this function retrieves a file struct AND returns the decrypted shared keys for ease of access
- UpdateKeys and NamesToRevoke –¿ Revoke necessary users first, then update the keys for everyone else to ensure they still have valid access.