CS1660: Dropbox Project

Design Plan

Table of Contents

[Introduction 3](#_Toc166186040)

[User Authentication 3](#_Toc166186041)

[Creating Users 3](#_Toc166186042)

[Authenticating Users 3](#_Toc166186043)

[File Storage 3](#_Toc166186044)

[Uploading Files 3](#_Toc166186045)

[Downloading Files 4](#_Toc166186046)

[Appending Files 4](#_Toc166186047)

[File Sharing 4](#_Toc166186048)

[Sharing Semantics 4](#_Toc166186049)

[Sharing Methods 5](#_Toc166186050)

[Revoking Access 5](#_Toc166186051)

[Security Analysis 5](#_Toc166186052)

[Attack 1: Unauthorized File Access 5](#_Toc166186053)

[Attack 2: Replay Attack on File Sharing 5](#_Toc166186054)

[Attack 3: Impersonation Attack 5](#_Toc166186055)

[Attack 4: Rollback Attack on File Updates 5](#_Toc166186056)

[Testing 6](#_Toc166186057)

# Introduction

In this project, we aim to implement a secure client for file storage and sharing across untrusted and trusted servers. We will achieve confidentiality and integrity using appropriate encryption, key management, and efficient file manipulation techniques. Our design addresses the stateless operation requirements, ensuring no reliance on local storage or global variables for maintaining user state. We aim to provide eight key functions, ranging across authentication, file storage, and file sharing.

# User Authentication

## Creating Users

* User creation will involve generating a new asymmetric key pair (encryption key and decryption key) for the user using AsymmetricKeyGen().
* The user's password will be used to derive a symmetric key using PasswordKDF(). This symmetric key will be used to encrypt the user's private decryption key.
* The encrypted private key and the user's public encryption key will be stored on the data server, indexed by the user's username.
* The user's public key will also be stored on the trusted key server, indexed by the user's username.

## Authenticating Users

* During authentication, the user provides their username and password.
* The client retrieves the user's encrypted private key and public key from the dataserver and keyserver, respectively, using the provided username.
* The user's password is used to derive the symmetric key with PasswordKDF().
* The derived symmetric key is then used to decrypt the user's private key retrieved from the dataserver.
* If the decryption is successful, the user is authenticated, and the client now has the user's private key and public key for future operations.

## File Storage

### Uploading Files

* To upload a file, the client generates a new random symmetric key using SecureRandom().
* The file contents are encrypted using SymmetricEncrypt() with the generated symmetric key.
* The encrypted file contents are stored on the dataserver at a new random location using memloc.Make().
* The symmetric key used to encrypt the file is encrypted using the user's public key with AsymmetricEncrypt().
* The encrypted symmetric key and the file's location (memloc) are stored on the dataserver, indexed by the user's username and the chosen filename.

### Downloading Files

* To download a file, the client retrieves the encrypted symmetric key and the file's location from the dataserver, indexed by the user's username and the requested filename.
* The encrypted symmetric key is decrypted using the user's private key with AsymmetricDecrypt().
* The decrypted symmetric key is then used to decrypt the file contents retrieved from the dataserver using SymmetricDecrypt().

### Appending Files

* To append data to an existing file, the client first downloads the file as described above.
* The client then appends the new data to the downloaded file contents.
* A new symmetric key is generated, and the updated file contents are encrypted using SymmetricEncrypt() with the new key.
* The encrypted updated file contents are stored on the dataserver at a new random location.
* The new symmetric key is encrypted using the user's public key and stored on the dataserver, overwriting the previous entry for that filename.
* This approach ensures that appending data to a file is efficient, as only the new data needs to be uploaded, and the previous file contents do not need to be re-encrypted or re-uploaded.

## File Sharing

### Sharing Semantics

* File sharing is implemented using a shared symmetric key approach.
* When a user (Alice) shares a file with another user (Bob), a new random symmetric key is generated using SecureRandom().
* The file contents are re-encrypted using SymmetricEncrypt() with the new shared symmetric key.
* The re-encrypted file contents are stored on the dataserver at a new random location.
* The shared symmetric key is encrypted separately using Alice's public key and Bob's public key, obtained from the keyserver.
* The encrypted shared keys and the new file location are stored on the dataserver, indexed by Alice's username, Bob's username, and the filename.

### Sharing Methods

* ShareFile(filename, recipient): Implements the sharing process described above.
* ReceiveFile(filename, sender): Retrieves the shared file information from the dataserver, decrypts the shared symmetric key using the user's private key, and uses it to decrypt and retrieve the shared file contents.

### Revoking Access

* RevokeFile(filename, recipient): To revoke access to a shared file, the client generates a new random symmetric key and re-encrypts the file contents using this new key.
* The re-encrypted file contents are stored at a new random location on the dataserver.
* The new symmetric key is encrypted using only the owner's (Alice's) public key and stored on the dataserver, overwriting the previous entry for the shared file.
* This effectively revokes access for the recipient (Bob), as the shared symmetric key encrypted with Bob's public key is no longer available.

## Security Analysis

### Attack 1: Unauthorized File Access

* An adversary (Eve) may try to access a file owned by Alice by guessing the filename and attempting to download it using DownloadFile().
* Protection: The file contents are encrypted with a symmetric key, which is further encrypted with Alice's public key. Without Alice's private key, Eve cannot decrypt the symmetric key and access the file contents.

### Attack 2: Replay Attack on File Sharing

* An adversary (Eve) may try to replay an old shared file entry to regain access to a revoked file.
* Protection: Each file sharing operation generates a new random symmetric key and re-encrypts the file contents. Old entries become obsolete and cannot be replayed successfully.

### Attack 3: Impersonation Attack

* An adversary (Eve) may try to impersonate a legitimate user (Alice) by providing Alice's username during authentication.
* Protection: Authentication requires knowledge of Alice's password to derive the symmetric key necessary for decrypting Alice's private key stored on the dataserver. Without the correct password, Eve cannot impersonate Alice.

### Attack 4: Rollback Attack on File Updates

* An adversary (Eve) may try to rollback a file to an older version by replaying an old encrypted file entry stored on the dataserver.
* Protection: Each file update operation generates a new random symmetric key and stores the re-encrypted file contents at a new random location on the dataserver. Old entries become obsolete and cannot be replayed successfully.

## Testing

* For each proposed attack, tests will be implemented to verify the robustness of the system against these attacks.
* Tests will simulate the adversary's actions and attempt to execute the described attack scenarios.
* Successful attacks will be detected by monitoring violations of confidentiality (unauthorized file access) or integrity (unauthorized file modifications) guarantees.