HEIG-VD - CAA

Projet

Christophe Roulin

Contents

1 Description of the project

1.1 Project Overview

The project aims to develop a robust and secure shared encrypted network file system. The system will prioritize user access, confidentiality of file and folder names, protection against active adversaries, and efficient sharing mechanisms while ensuring usability and ease of interaction.

1.2 Project Goals and Objectives:

- **User Authentication:** Implement a user-friendly username/password authentication system requiring minimal logins for seamless user experience.
- **Security Measures:** Ensure robust protection against active adversaries while maintaining confidentiality of file and folder names.
- **Trust Model:** Assume an honest but curious server scenario, focusing on safeguarding data from potential breaches.
- **Device Flexibility:** Enable users to access the file system from various devices effortlessly.
- **Sharing and Access Control:** Facilitate folder sharing among users and implement access revocation.
- **Password Management:** Enable users to securely change their passwords.

1.3 Features and Functionalities:

The system will provide the following key functionalities:

- **Secure File Operations:** Support secure downloading and uploading of files within the system.
- **Folder Management:** Allow users to create folders and manage their structure securely.
- **Sharing Mechanism:** Enable users to share folders securely with other authorized users.
- **Access Revocation:** Implement a secure process for revoking access to shared folders.
- **Password Change:** Provide a secure mechanism for users to change their passwords.

2 System modelling

2.1 User management

2.1.1 User registration

The server will be responsible for user registration while most of the operations will be done by the client locally.

The registration process is as follows:

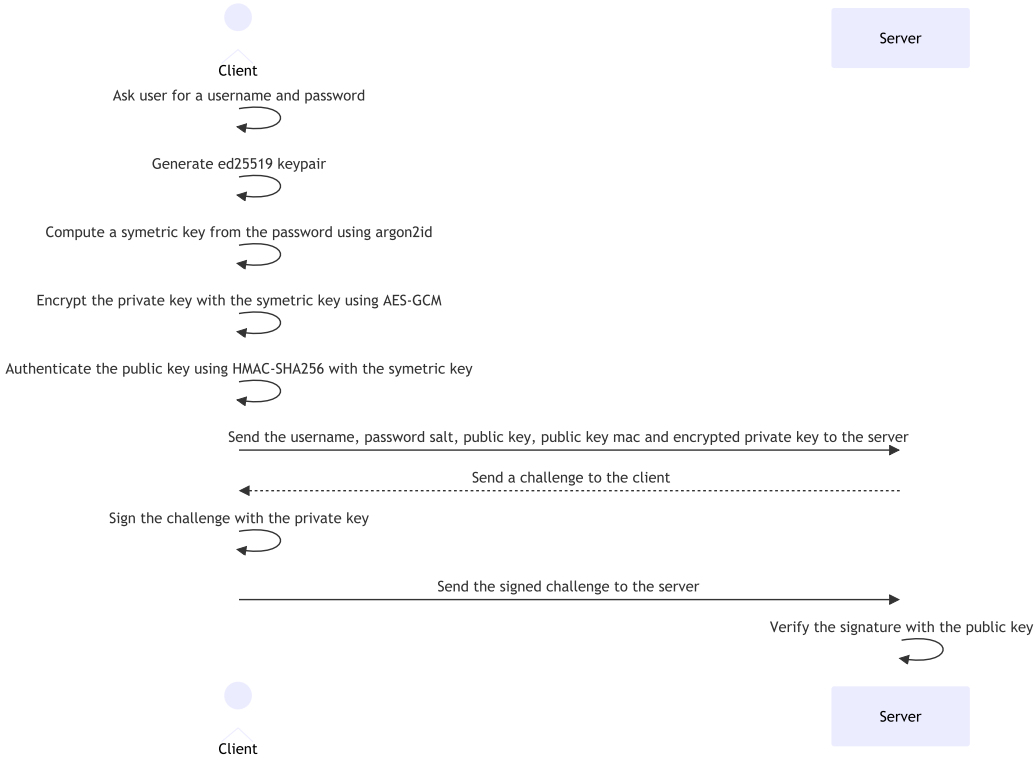

Figure 1: User registration

2.1.2 User login

The login process is as follows:

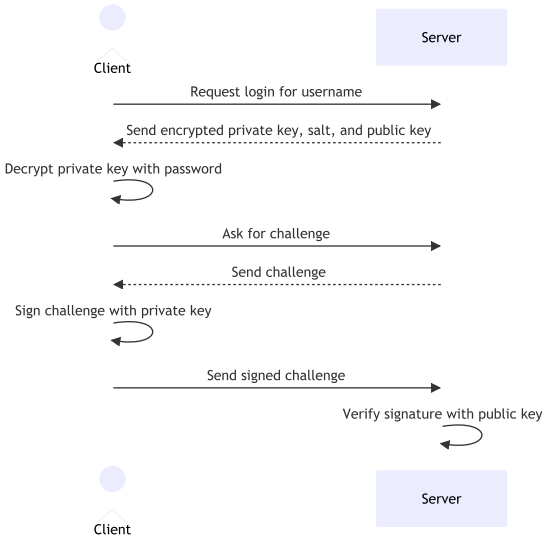

Figure 2: User login

2.1.3 Users list

The server will maintain a list of registered users. Anyone can query the server for the list of users. The server will then return the list of users along with their public keys.

2.2 Files and folders

The system will be based on a tree structure. Each user will have a root folder that will contain all of the user's files and folders. Each element inside a folder will have a key derived from the folder's key. The key derivation process will be done using a key derivation function (KDF), argon2id. This will allow us to easily share a folder with another user by simply sharing the folder's key.

2.2.1 Metadata

Each file and folder will have a metadata file that will contain the following information:

- **Nonces**
- **Encrypted keys**
- **Sharing list**: The sharing list will contain the list of users that have access to the file or folder.

2.2.2 Root folder

The root folder is a special folder that is created when a user registers. It is the only folder that has a key that is not derived from the folder's parent key since it has no parent.

The root folder creation process is as follows:

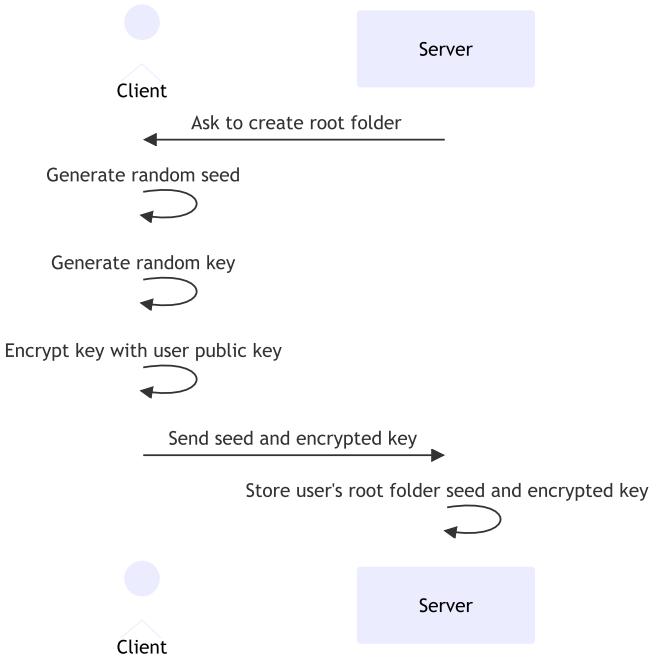

Figure 3: Folder creation

2.2.3 Folder creation

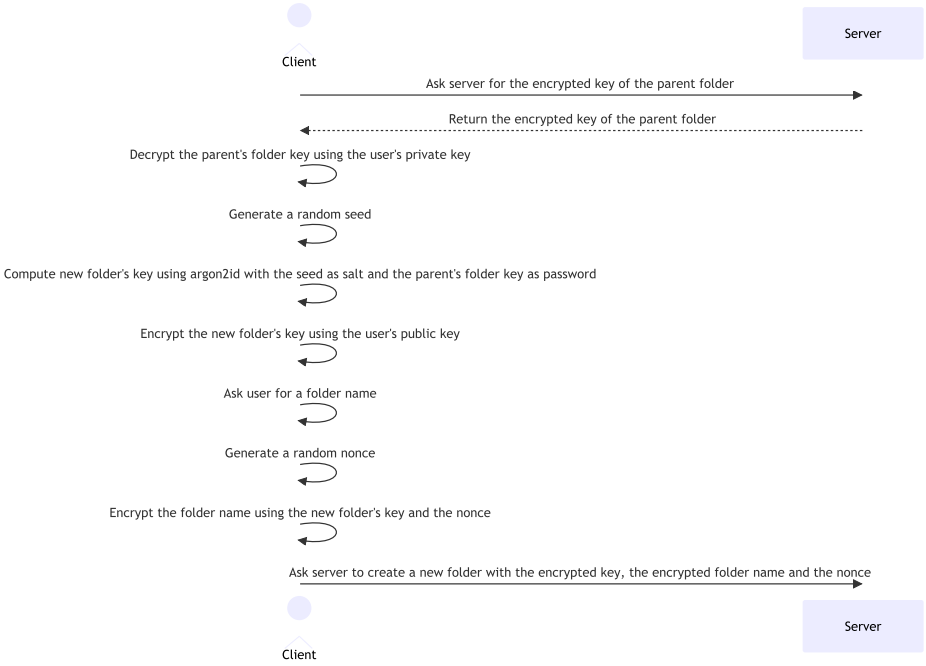

Figure 4: Folder creation

2.2.4 File creation

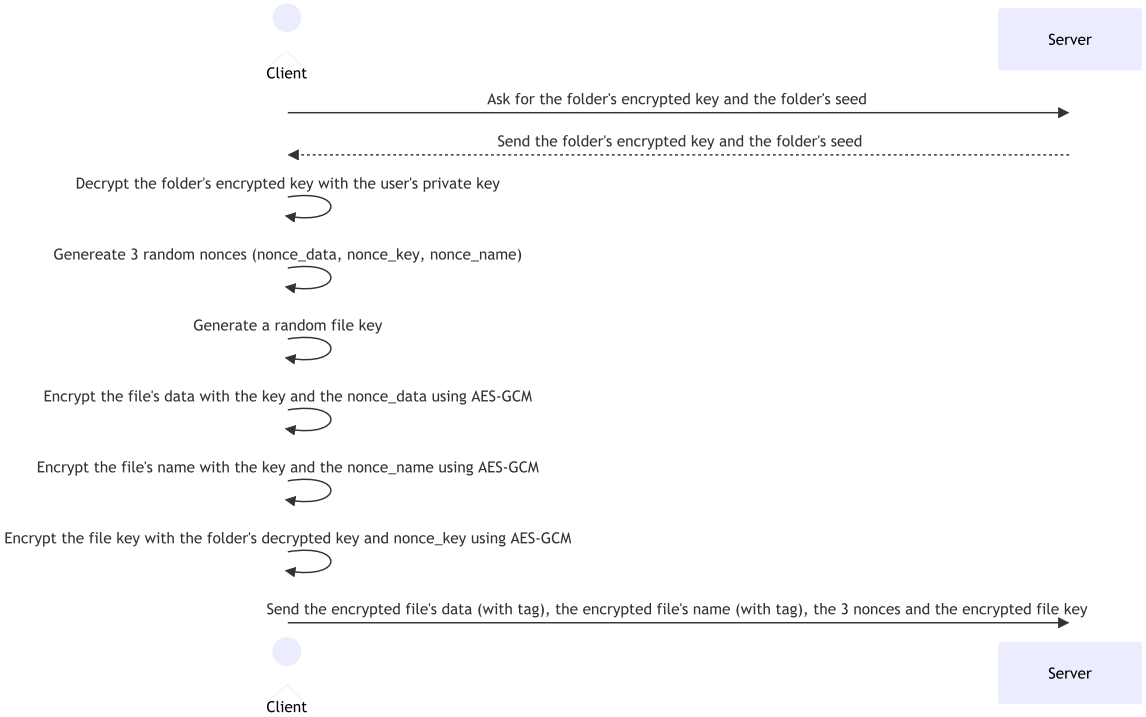

Figure 5: File creation

2.2.5 Access files and folders

The process of accessing a file or folder is as follows:

2.3 Access file

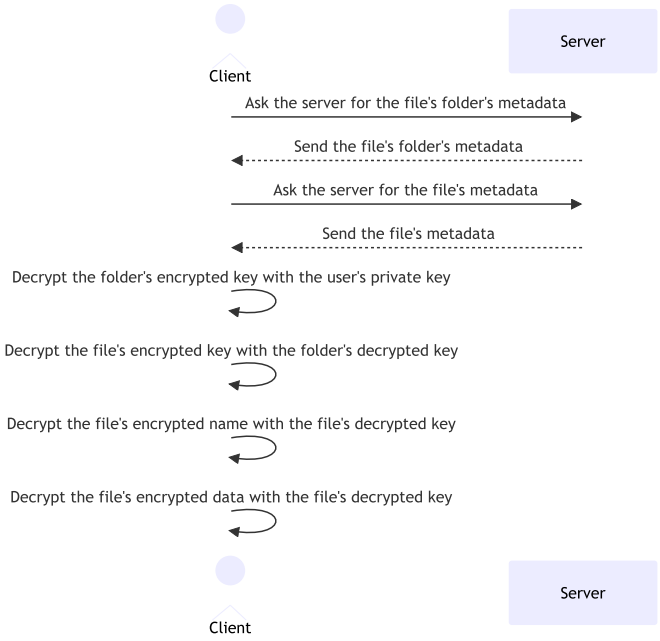

Figure 6: Accessing a file

2.4 Access folder

TODO : Complete this

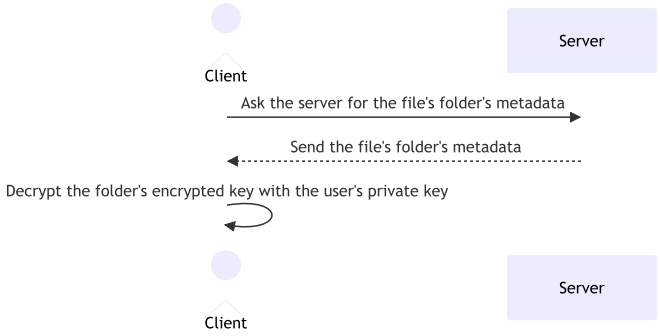

Figure 7: Accessing a folder

2.5 Sharing

2.5.1 Folder sharing

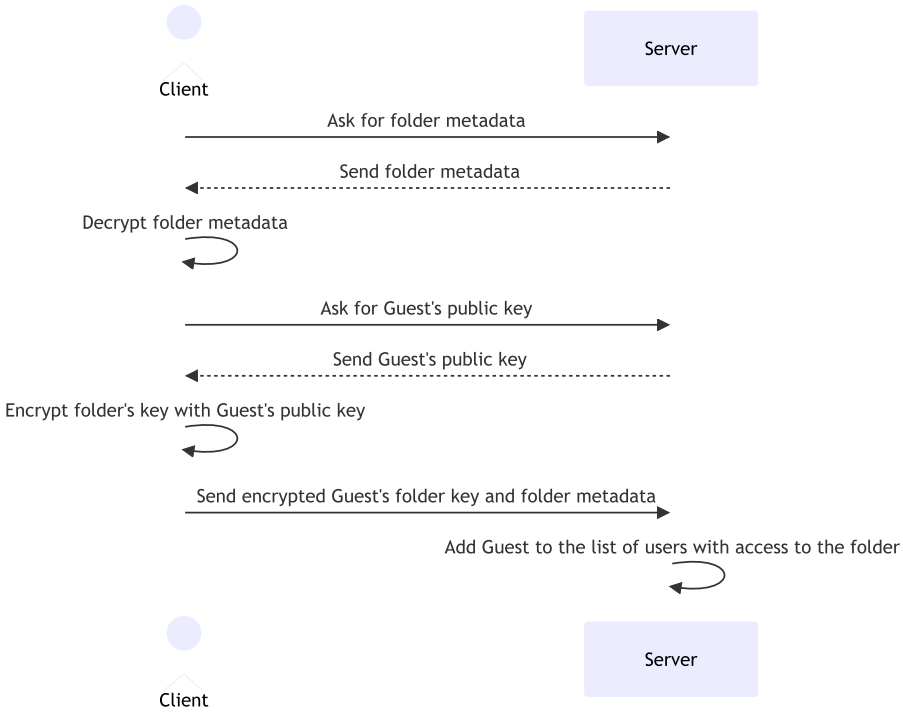

Figure 8: Folder sharing

2.5.1.1 Root folder special case

The root folder cannot be shared or revoked. If a user wants to share their root folder, they will need to create a new folder inside their root folder and share that folder instead.

2.5.2 File sharing

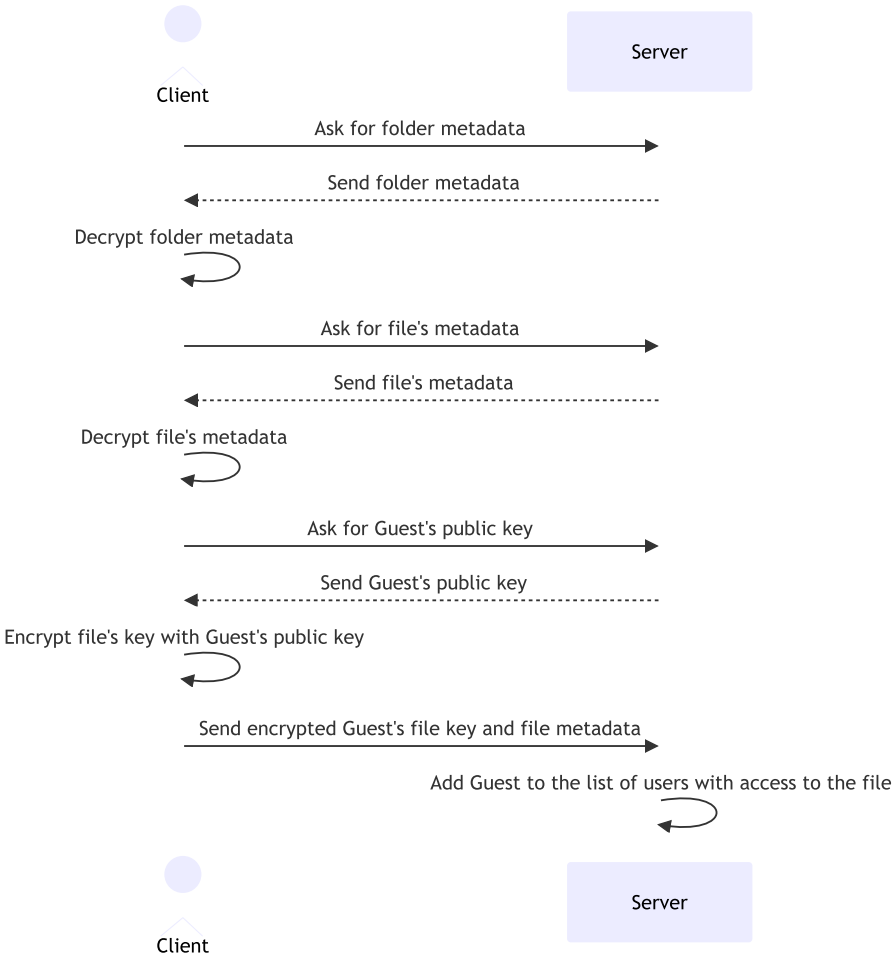

Figure 9: File sharing

2.6 Revoking access

The process of revoking access to a file or folder is as follows:

2.6.1 Folder

Note: The revokation process is done recursively. If a folder is revoked, all of its subfolders and files keys need to be updated as well. If any subfolder or file was shared with another user, we'll need to send the server the new keys for those files and folders for those users to be able to access them.

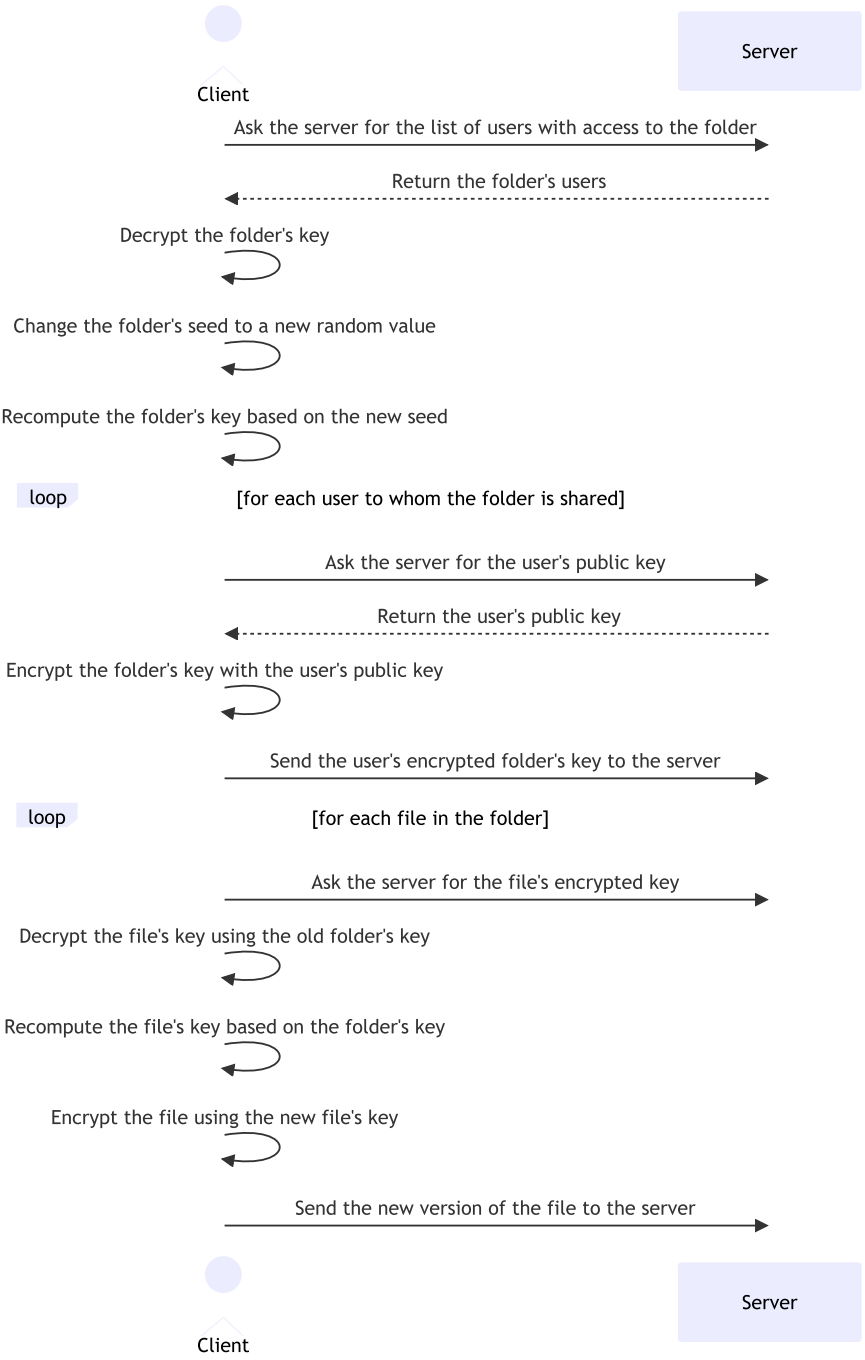

Figure 10: Folder revokation

2.6.2 File

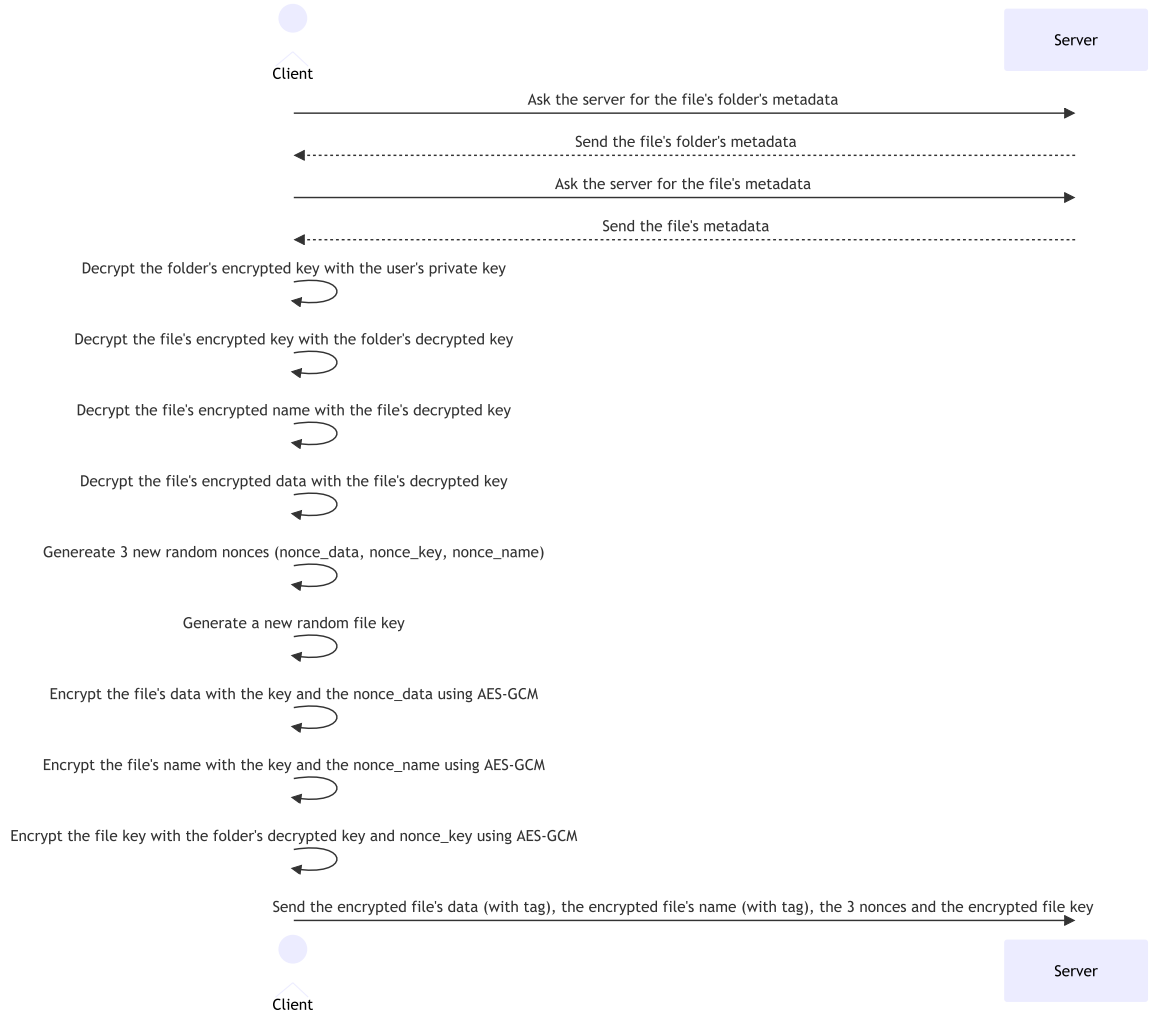

Figure 11: File revokation

2.7 Server role

In our model, the server is only acting as a storage server. It does not have access to the user's private keys and cannot decrypt any of the user's files or folders. The server is only responsible for storing the user's files and folders and for sharing the user's public keys with other users.

The server is also responsible of verifying each user identity when they register or login. This to ensure that no one tries to update something that he's not allowed to.

2.8 Client-Server communication

TLS 1.3 will be used for client-server communication. The server will be authenticated using a valid X. 509 certificate. The client will verify the server's certificate and will only communicate with the server if the certificate is valid.

When connecting to the server, the client will ask the user for the server's hostname. The client will then try to connect to the server using the hostname and will verify that the server has a valid certificate for the hostname.

2.9 Threat model

The following are the various parts of our threat model:

- **Honest but curious server.** The server is assumed to be honest but curious. This means that the server will not try to attack the system but will try to learn as much as possible about the system. We can assure a perfect security against this type of adversary by encrypting everything that is sent

to the server. This includes the user's files, folders, and metadata. The server cannot decrypt any of the user's data since it does not have access to the user's password.

- **Active adversary.** An adversary that can intercept, modify, and inject messages. We consider this adversary to be installed in between the client and the server. We are protected against this type of adversary by using TLS 1.3 for client-server communication. This ensures that the adversary cannot intercept, modify, or inject messages.
- **Stolen client device with closed session.** We assume that the client device can be stolen. This is possible because once a session is closed and the user logs out, nothing stays on the device.

The following are not considered part of our threat model:

- **Offline attacks.** We do not consider offline attacks on the user's private key. Anyone can ask the server for the user's encrypted private key and try to decrypt it. Since the key is encrypted with a KDF of the user's password, it would be equivalent to bruteforce the user's password. This would be possible to prevent by asking the user to store the private key himself but this would be less user-friendly than just using a password to login.
- **Stolen device while openned sessions.** We do not consider the case where the device is stolen while the user is logged in. This is because the user's private key is stored in memory and can be accessed by anyone with access to the device.

2.10 What could we do to protect against compromised server ?

TODO : What could we do to protect against a compromised server ? MACs or signatures everywhere

2.11 Storage overhead

TODO : Compute the storage overhead for a file and for a folder.

3 System architecture

3.1 Session handling

A session is created for each user when they login. The session is used to remind ourself that the user is logged in and authenticated. The session is closed when the user logs out.

This is done by sending the client a session token when the user logs in. The client will then send this token with each request to the server. The server will then verify the token and will only respond to the request if the token is valid.

3.2 Protocol design

Data between the client and sever is sent in JSON format. Each request will contain the following fields:

```
{
  "session_token": "session token", # Optionnal, not needed for login and register
  "request": "request type",
  "data": "request data"
}
```

3.2.1 Request types

The request type can be one of the following

3.2.1.1 User management

- **register_user:** Used to register a user. The data field will contain the username, encrypted private key and public key.
- **prepare_login:** Used to prepare a login. The data field will contain the username. The server will then send the user's encrypted private key.
- **login:** Used to login a user. The data field will contain the username. The server will then send the challenge for the user.
- **verify_login:** Used to verify a login. The data field will contain the username, the challenge and the challenge response.
- **logout:** Used to logout a user. The data field will be empty.
- **get_users:** Used to get the list of users. The data field will be empty.
- **get_user_public_key:** Used to get a user's public key. The data field will contain the username.
- **change_password:** Used to change a user's password. The data field will contain the new encrypted private key.

3.2.1.2 Files and folders

- **create_folder:** Used to create a folder. The data field will contain the folder's name and the parent folder's path.
- **create_file:** Used to create a file. The data field will contain the file's name and the parent folder's path.
- **get_file:** Used to get a file's content. The data field will contain the file's path.
- **get_folder:** Used to get a folder's content. The data field will contain the folder's path.
- **update_file**: Used to update a file. The data will contain the file's path, the new content and the new file's metadata.
- **update_folder**: Used to update a folder. The data will contain the folder's path and the new folder's metadata.

3.2.1.3 Sharing

- **share_folder:** Used to share a folder. The data field will contain the folder's path, the user to share the folder with and the encrypted folder's key. If the folder is already shared with the user, the encrypted folder's key will be updated.
- **share_file:** Used to share a file. The data field will contain the file's path, the user to share the file with and the encrypted file's key. If the file is already shared with the user, the encrypted file's key will be updated.

3.2.1.4 Revoking access

- **revoke_folder:** Used to revoke a folder. The data field will contain the folder's path and the user to revoke the folder from. This will have the effect of removing the user's encrypted folder key.
- **revoke_file:** Used to revoke a file. The data field will contain the file's path and the user to revoke the file from. This will have the effect of removing the user's encrypted file key.