CAA - Lab 02

Encrypted Vault

24 mai 2021

Student
Doran Kayoumi

Teacher
Alexandre Duc
Assistant
Nathan Séville

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Table des matières

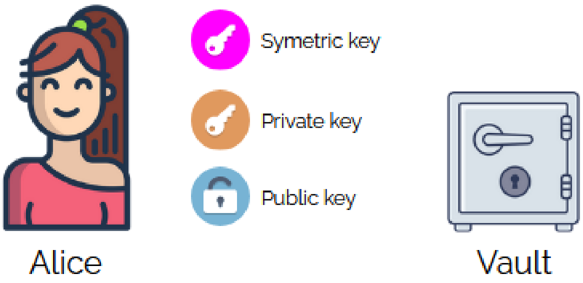
1 Architecture

The architecture follows a Client-Server model with a (imaginary) TLS 1.2/1.3 connection between them.

To make sure that the server knows nothing about the data the users upload to the vault, all the key generation and file encryption/decryption will be done client side. The only cryptography the server will be doing is to validate the tags sent by the client.

1.1 Legend

Here's the legend for all the following diagrams explaining the architecture.



1.2 Client-Server communication

All the requests made between the client and the server will have attached a tag of a session token (see [Authentication](authentication) for more info). The tag will be a **HMAC** of the session token using a **shared secret** as the key.

The shared secret is a derivation of the user' password done with **argon2id**.

1.3 Authentication

The authentication will be done using a simple challenge response protocol. Once a user authenticates her/him self, the client will send the **name** of the user that's trying to login. The server will send back a **challenge** and the **salt** needed to (re)compute the shared secret. The client will (re)compute the shared secret, calculate the tag of the **challenge** and send it to the server. The server will check if the tag received is valid, if it is, the server will send back a **session token** (and an encrypted list of the names of all the files owned by the user).

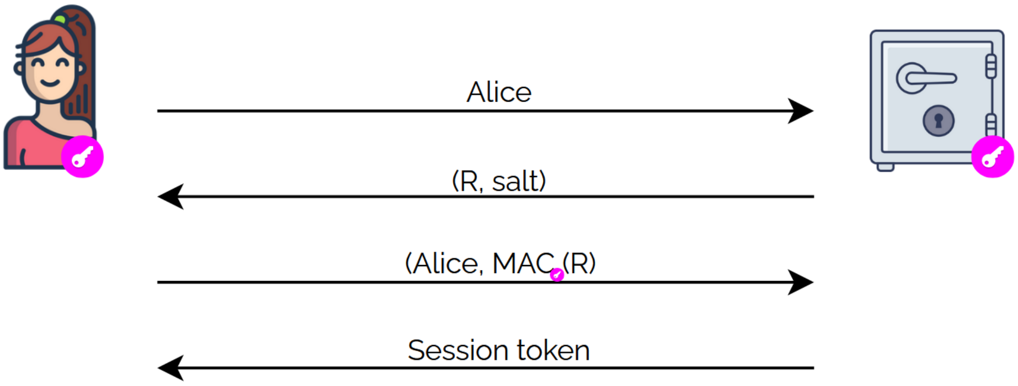


FIGURE 1 – Authentication protocol

1.4 File upload

When a user uploads a file to the vault, its first encrypted then sent alongside the key used for the encryption (which is also encrypted).

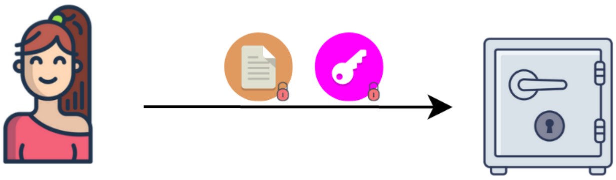


FIGURE 2 – File upload

1.4.1 Encryption

The file encryption will be done using **AES-GCM** and to guarantee long term security, the keys (a different key for each file) will have a length of **256 bits**. Since the key is also stored on the server, it will also be encrypted, but this using **ECIES**. The reason for using asymmetric cryptography for the key encryption is to simplify file sharing. **ECIES** was chosen to avoid having to also store huge keys (since we want to have long term security).

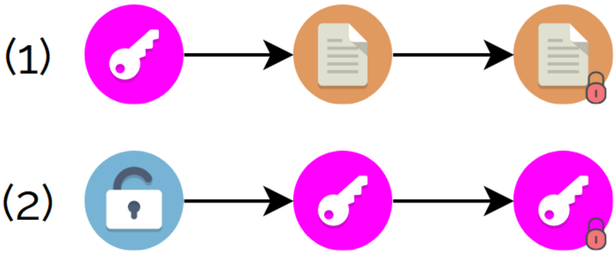


FIGURE 3 – File encryption

1.5 File download

For the download, the user gives the name of the file she/he wants, and the server will send back the encrypted file and encrypted key. And, Of course, the server will check that the user requesting the file actually owns it.

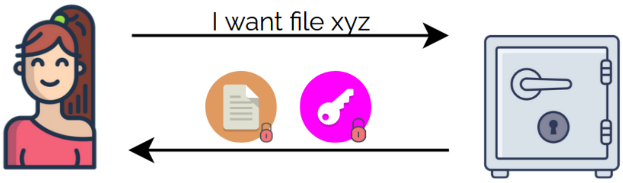


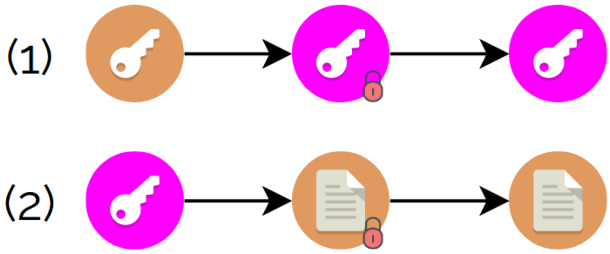
FIGURE 4 – File upload

1.5.1 Decryption

The decryption is pretty straight forward. The symmetric key is decrypted using the users private key and then the file is decrypted using the decrypted symmetric key.

1.6 File sharing

To share a file, the only thing to do is to encrypt the files encryption key with the public key of the user we want to share the file with.



)

Figure 5 – File upload