Mini-Project MA-ICR

A shared encrypted network file system

Titus Abele

MSE Computer Science
HES-SO Master
Lausanne, Switzerland
June 4, 2024

Contents

Listings

List of Figures

Introduction

Before jumping into the theory behind a shared encrypted network file system, it is important to lay a foundation about what a file system is, why it requires sharing capabilities and why security is paramount. A file system (FS) manages and provides access for resources. Generally, resources are composed of folders containing files or other folders. This way, an organizational hierarchy of resources can be established.

```
[drwxr-xr-x 4.0K]  .
  |--- [lrwxrwxrwx     7]  bin -> usr/bin
  |--- [drwxr-xr-x 4.0K]  boot
  |--- [drwxr-xr-x 3.5K]  dev
  |--- [drwxr-xr-x 4.0K]  etc
  |--- [drwxr-xr-x 4.0K]  home
  |--- [ rwxrwxrwx 2.0M]  init
  |--- [lrwxrwxrwx     7]  lib -> usr/lib
  |--- [lrwxrwxrwx     9]  lib32 -> usr/lib32
  |--- [lrwxrwxrwx     9]  lib64 -> usr/lib64
  |--- [lrwxrwxrwx    10]  libx32 -> usr/libx32
  |--- [drwx------   16K]  lost+found
  |--- [drwxr-xr-x 4.0K]  media
  |--- [drwxr-xr-x 4.0K]  mnt
  |--- [drwxr-xr-x 4.0K]  opt
  |--- [dr-xr-xr-x     0]  proc
  |--- [drwx------ 4.0K]  root
  |--- [drwxr-xr-x   680]  run
  |--- [lrwxrwxrwx     8]  sbin -> usr/sbin
  |--- [drwxr-xr-x 4.0K]  snap
  |--- [drwxr-xr-x 4.0K]  srv
  |--- [dr-xr-xr-x     0]  sys
  |--- [drwxrwxrwt   12K]  tmp
  |--- [drwxr-xr-x 4.0K]  usr
  |--- [drwxr-xr-x 4.0K]  var

23 directories, 1 file
```

Listing 1: Ubuntu root folder

Access should be handled so that some roles can access some resources, this is generally done by using dedicated roles such as administrators, owners, guests or even users. An example can be seen in Listing 1, it shows all folders and files contained within the root directory of a machine running Ubuntu, a popular Linux distribution. The root directory is the top-level directory in the file system's hierarchy. Left of the resource names (`boot`, `dev`, `etc`...) we can find their relative permissions. For instance, the `boot` directory is marked as `[drwxr-xr-x 4.0K] boot` which means:

- The `d` in first position indicates the nature of the resource, in this case a directory.
- The pattern `rwxr-xr-x` translates to the owner having read, write, and execute permissions, while the group and others have read and execute permissions only.

Each triplet (rwx) relates to a specific permission class. In a Unix-like file system such as the one depicted in Listing 1, these classes are defined in order as "Owner-Group-Others". This means that for the boot directory:

- The Owner has read (r), write (w) and execute (x) permissions.
- The Group that is associated with the directory and all other users only have read and execute permissions.

This way any access to the directory is dynamically limited and permissions can be revoked, approved and modified very easily.

```
seirios@T16:/$ touch foo.txt
touch: cannot touch 'foo.txt': Permission denied
```

Listing 2: Trying to create a file in the root folder without write permissions

The reason such strategies are put in place is not only related to security. The root directory, such as the one from the Linux distribution, contains all the resources necessary for the proper functioning of the operating system (OS). Modifying these files may cause irreversible configurations that may break the proper flow of the OS and cause failure. The user trying to create a text file using the touch[1] command in Listing 2 is being denied by the operating system. The current directory being the root directory, one needs so called root-privileges to create, modify or delete any resources[1].

To further emphasize the importance of the mechanic surrounding permissions, we must talk about security. A machine such as the one depicted above, may be of use to multiple actors. Therefore, systems must be in place to regulate access across user sessions and resources. Alice may not want Bob to read, edit or delete some of her files or even worse create some in her name. This means that a file system must also be able to obfuscate files from eavesdroppers and make them unavailable, unreadable and uneditable to malicious attackers. This means that in order to consult her own files, Alice must be logged in otherwise her files are inaccessible. But there might be a file which Alice wants to share with Bob for a project, in this case she may create a group, add Bob to this group and using the appropriate triplet from earlier, specify that people of that group may read and write to this file.

Project goal

The goal of this project is to design a shared encrypted network file system. The most important aspect of this FS is the manner in which security is guaranteed. In this report, we will outline the general structure of the FS and for each component, specify the necessary cryptographic tools used to provide trust and security.

[1]There are many ways of obtaining some of these privileges notably the sudo command which will give the user elevated permissions. It is important to note however, that the user trying to sudo a command must be part of a sudoers group

1 Architecture

1.1 Web Server

A web server is an online machine that serves clients. Its purpose can be manifold. In our project, this web server will store files for users. They may connect to the server using some credentials (in our case a unique username and a password) and have access to their files as well as the files that have been shared with them by other users. In Figure 1 we outlined the general architecture of the server. The server prompts the client for credentials, the client provides them, the server can then verify the credentials and grant access to the storage. In this way, the client can connect from anywhere and on any device, without having to transport the entire storage. This is essentially what is commonly referred to as a cloud storage service.

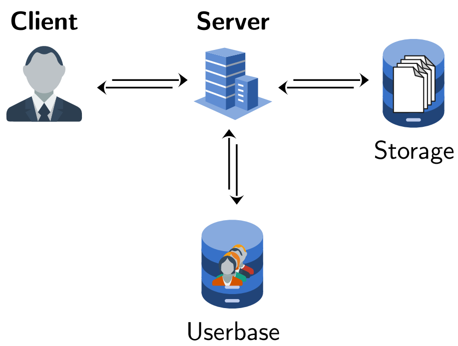

Figure 1: General architecture of a web server with file saving and sharing capabilities

The standard procedure to interact with the server is outlined in Figure 2. As depicted, there are four components that need to be conceptualized.

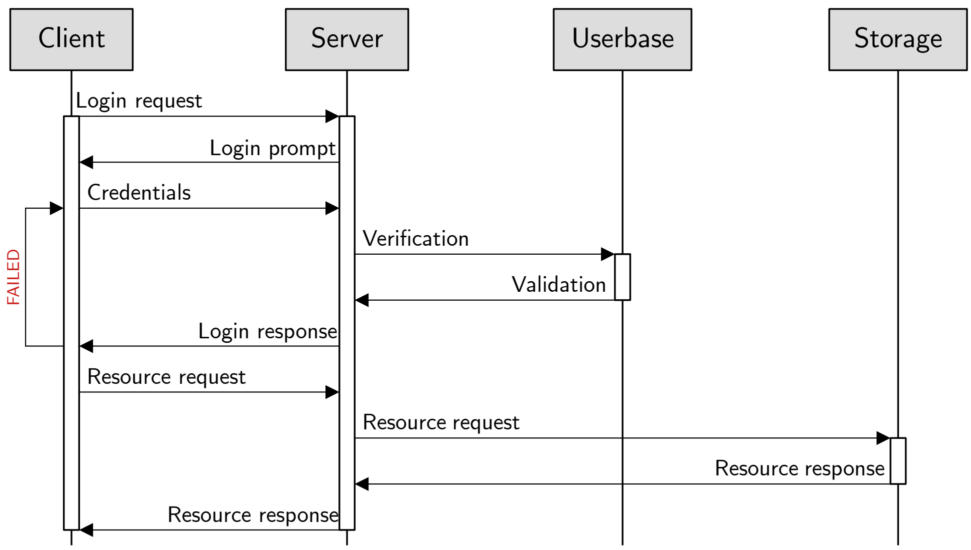

Figure 2: General interaction graph

1.2 Storage

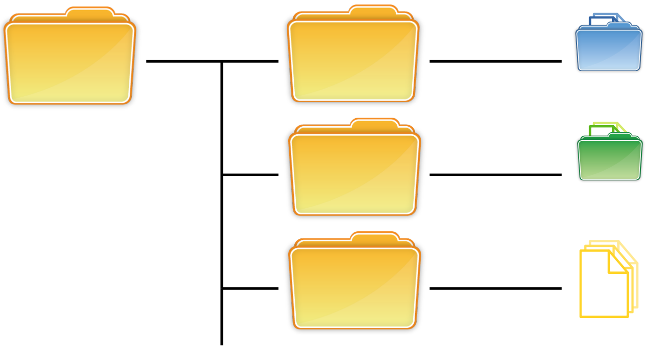

Figure 3: A basic filesystem

The building blocks for a simple file system are as follows:

- Folders are containers for other folders and files.
- Files are bundles of information.
- A root folder is the root node of the hierarchy that illustrates the file system. In Figure 3 the root folder is the left most folder.

Storage is handled by a server module which implements all the necessary tools for our model. It also provides types corresponding to the architecture we described above. The file structure is composed of a file name, an owner and some content. Added to the simple structure is a "signature" field, it will be crucial for ensuring trust in the provenance and the integrity of the file, more on that later. Secondly, the folder structure, in addition to name, owner and signature it also contains a list of folders, and a list of files.

1.2.1 Storage encryption

The filesystem is stored on a server. In this project, the server is exposed to active adversaries which means that the filesystem must be fully encrypted. The constraints of the project allow for the filesystem's hierarchy to leak which means that only file contents, file and folder names as well as owners must be encrypted. The signature must also be encrypted, this may seem strange but if this weren't the case, an attacker could potentially try all public keys on a resource until the signature verification algorithm verifies the given public key. This would tell the attacker whose file they just verified and may lead to unique id leakage, linking unique ids to usernames. Each folder additionally contains two separate lists: a list containing each folder's (that is inside the current folder) symmetric key and a list containing each file's (also contained in the current folder) symmetric key. These are used to encrypt and decrypt the files.

Because only the owner of a root folder on the server should be able to access all the contents of their root folder, symmetric encryption seems to be the ideal choice. It also provides no way for an active adversary to find a way of sneaking between the user and the server to eavesdrop on the communication or perform a man in the middle attack. Symmetric encryption dictates that a single master key encrypts the root folder

To achieve symmetric encryption on this scale, we use a password. This password is not stored on the server. The user must remember it well. The password will be digested client-side to create a password hash. This hash can be used for two processes:

- First, the login challenge.
- Second, the fetching of the encrypted root folder (containing all their files).

The process is divided into two separate processes to make it impossible for an attacker to get a free offline version of the encrypted resources. If there were no challenge and since we do not store passwords on the server, we would immediately send back the encrypted root folder without authenticating the user. Therefore, an attacker has an offline version of the encrypted root folder and can perform brute-force attacks without us ever knowing. The two parts will be explained in detail in the next section.

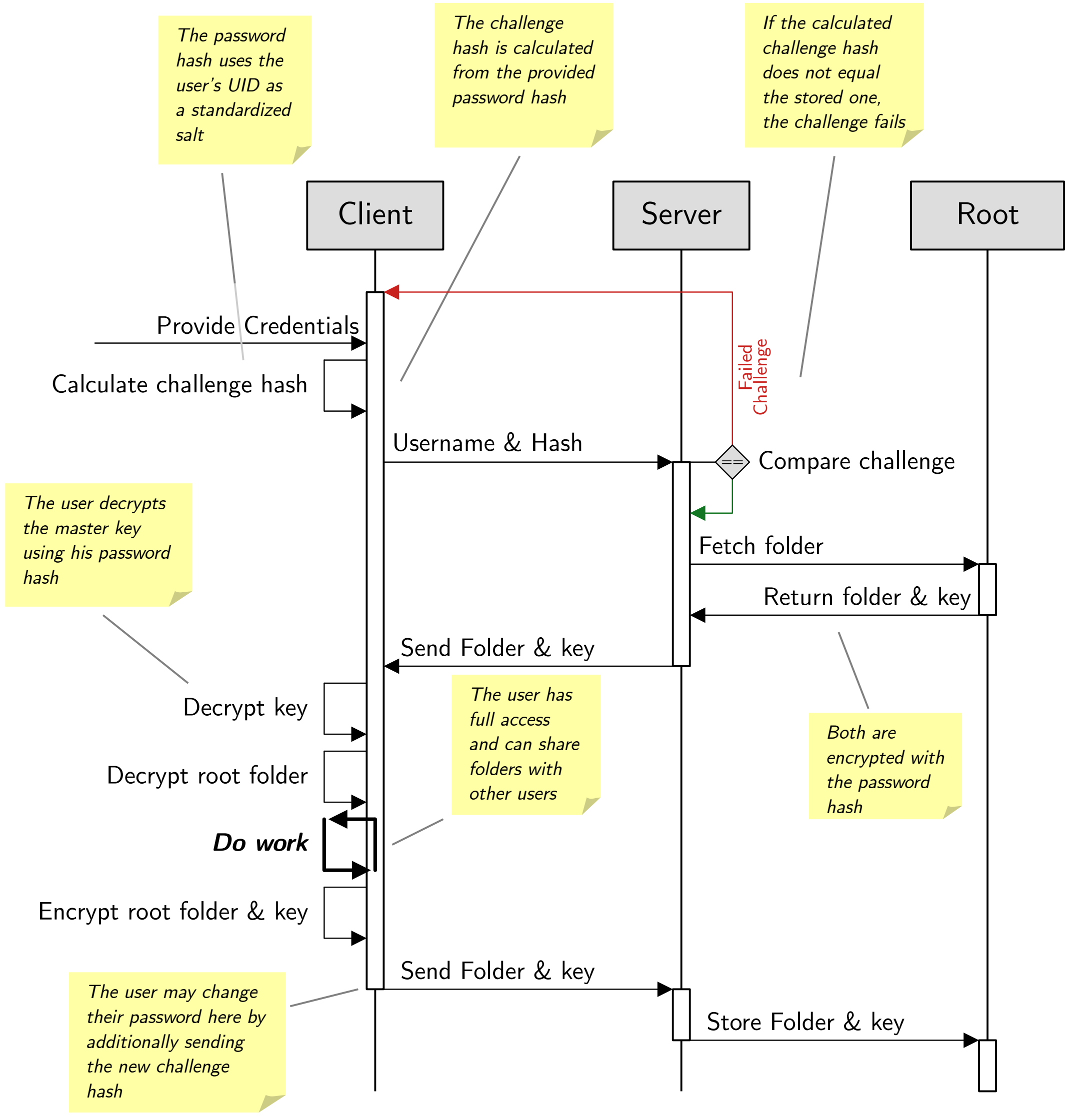

Figure 4: The sequence of our model

1.3 Authentication

The authentication part of the model relies on the User structure. A user can be created and added to the server's user list. When creating a user, we must also create its root folder which must be provided when adding the user to the server. This root folder carries the unique user id as a name and is therefore closely related to this user. We are willing to make unique identifiers public because they are nothing more than basic resource locators so we don't mind leaking them. No information can be extracted from them.

1.3.1 Root folder decryption and encryption

The sequence in Figure 4 shows the basic operation of the filesystem. Once a user has been created and provided to the server with a root folder, an additional challenge hash is required to make the two-step authentication possible. The user is not aware of this two-step process, he only witnesses the credential providing step and the receiving of data or failing (indicated by the red path).

The challenge hash is required because it allows us to avoid storing passwords or their hashes on the server. This means that the server is never in contact with something directly derived from the password or more crucially, the password itself. The whole process of creating a user by providing root folder and a challenge hash ensures this ignorance.

References

[1] The `touch` Command Manual Page - Linux manual page. (n.d.). https://man7.org/linux/man-pages/man1/touch.1.html