Remote Document Access

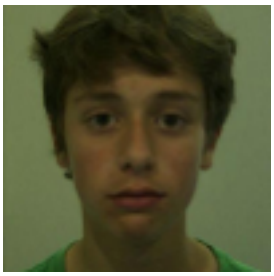
- Network and Computer Security 2018/2019 -

Project Proposal

Group T11

André Fonseca 84698
andre.filipe.alves.fonseca@ist.utl.pt

Leonor Loureiro 84736
leonor.filipa@ist.utl.pt



Sebastião Amaro 84767
sebastiao.amaro@ist.utl.pt

Instituto Superior Técnico
MSc. Engineering and Computer Science

Problem

We want to develop a cloud-based solution that allows documents to be shared over a public network in a secure fashion, meaning that a group of users can work together on the same documents.

Since communications are carried over a public network and the documents are stored on a remote device, the documents are left exposed to unauthorized third parties that might be able to view and tamper with them, without the user's permission.

Authentication, authorization, integrity and confidentiality must be ensured to control the accesses and detect illegal modifications of the documents by unauthorized users, even in the situation where the attacker might gain physical access to the storage devices.

The system must also be resistant to *ransomware* attacks, allowing a user to restore the information lost in the attack.

Requirements

- The servers must authenticate the user for every request.
- The system shall not allow any user to access the documents of another user that have not been shared with him.
- The system must detect attempted accesses and modifications to the documents by unauthorized users.
- The server must be able to restore the documents the users lost through a *ransomware* attack from a previous backed up version.
- The system must ensure that if an attacker accesses the servers storing the documents, he is not able to view the files.
- The system shall not allow unauthorized third-parties to access or corrupt the communication between the client and the servers.

Solution

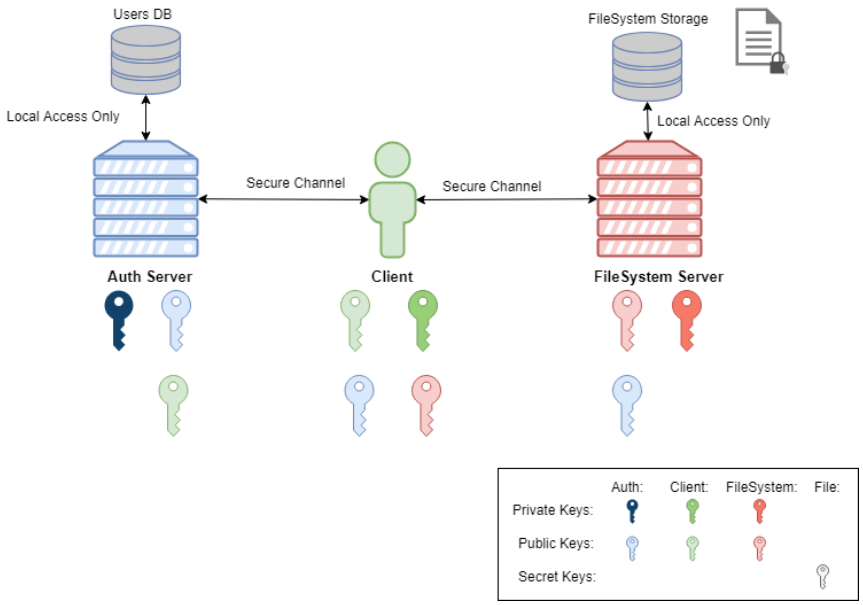


Figure 1. Architecture

The figure above describes the system's architecture and the distribution of the keys.

Assumptions:

- We trust the authentication server only to authenticate the users.
- We trust the file system for storage only.
- The server's certificates were installed with the application, in a secure way.
- A user is able to store his private key in a flash memory device, encrypted with his password. The security of this device is assured by the client.
- The public key of a user is validated when the user is registered.
- The user is responsible for the safety and confidentiality of all that is stored on his computer.
- The file system is immune to direct *ransomware* attacks (only through the client).

The servers authenticate themselves before the client using their public key certificate. The client authenticates himself before authentication server using it's username and password, to which he receives an authentication token that he uses to authenticate himself before the file system server.

When a user is registered, the authentication server stores a salted hash of the user's password, which later is used to validate the user's credentials during login.

The documents are stored in the file system, encrypted with a unique symmetric key K_{CS}. This key is generated by the user that created the file. The file system stores this key, encrypted with the public key of all the users that have access to it.

To share a file with another user Y, the user X will request the public key for user Y before the authentication server and encrypt the file encryption key K_{CS} with user's Y public key.

To ensure the files are not corrupted, a MAC(file, K_{CS}) is also stored with the file and sent with every upload request and validated with every download.

To provide protection against *ransomware* attacks to the client, the file system keeps the previous versions of the documents backed up so that the client might be able to restore the lost information from the system.

The figure below shows a possible sequence of interactions between the client and the servers.

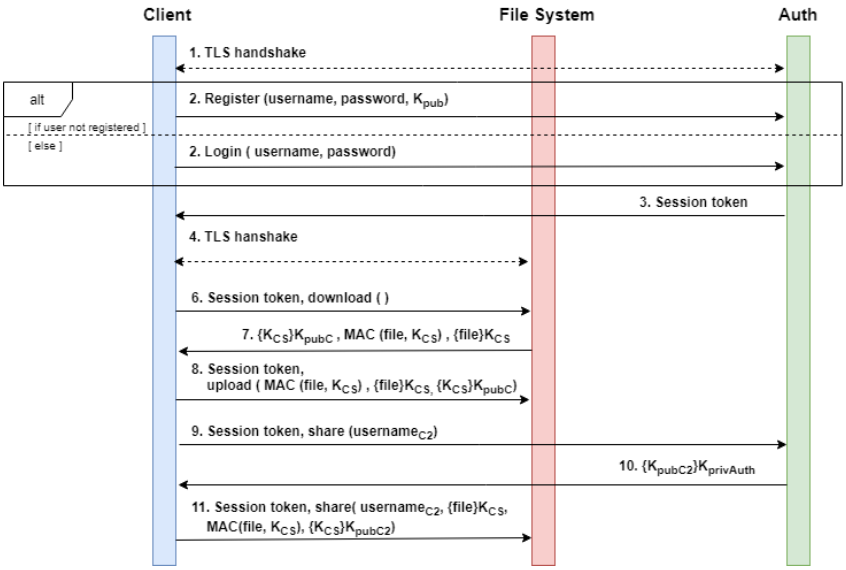


Figure 2. Possible interaction diagram

Basic Version:

- basic implementation of the download and upload functionalities,
- establishes a secure connection between the client and the servers, and
- authentication mechanisms: register, login, generate and validate authentication token.

Intermediate Version:

- file encryption and integrity validation,
- generation and storage of the client's private and public key, and
- file sharing.

Advanced Version:

- version control, and
- true sharing (propagates the changes one user makes to the document to the copies the other user share).

Results

We implemented all of the proposed solution, with exception of the true sharing. We were only able to do a simple sharing system that shares a copy of the document but does not propagate the subsequent changes each user makes

Evaluation

Weaknesses

The sharing system implemented is vulnerable as it does not request the permission of the user a document is being shared with for the operation, which might allow a malicious user to infect the other user's file system. This problem is especially aggravated as our system does not allow a user to delete a file.

Our version control system does not limit the number of versions stored. This solution is not only not scalable, as it might incur in high memory overhead, also represents a vulnerability, allowing a user to exceed the file system's memory capacity by sending a large number of upload requests for each of its file.

The login mechanism is susceptible to brute force attacks as it does not limit the number of attempts a user can make in certain period of time.

Neither the implemented solution nor the proposed solution provide perfect forward secrecy, as it relies on long lasting secrets that aren't ever renewed.

We also did not implement a password recovery system. As a user's password is the only means of authentication and of accessing it's keys, if the user were to lose this password, he would no longer be able to access the system and all his files would be lost, except for those that were locally stored.

Strengths

Our solution satisfies the confidentiality, authentication and integrity requirements of the project.

Our architecture strictly separates the authentication service from the storage. This not only allows for better load balancing of user requests between the servers but it also provides an extra protection by separating the documents from the users information.

The implemented project's structure is divided in lowly-coupled and highly-cohesive modules, meaning it's easier to maintain and improve its quality. It also means that the modules are sufficiently independent among themselves that they allow a module to be modified or replaced, without having much impact on the remaining modules.

Besides using wide-spread, well tested technologies, we also have a large coverage of tests, especially for the module that implement the cryptography functions, providing us with a high level of confidence in the quality of the system.

We verify the validity of the inputs at ever interface, reducing the probability of a misunderstanding about the behavior of other modules.

Design Decisions

We decided to cut back on functionality in order to develop a more robust and well tested system.

Our choice of technologies was based on the fact the, for most of the used technologies, we had worked with them before, and those we hadn't were simple and well documented.

Even though the client already had a private key, which he could have used to authenticate himself, we choose to implement an authentication system based on a password, because this the most widespread approach, which in turn would allow the authentication module implemented to be used with other services.

We chose to have the client communicate directly with each of the server, instead of having one of the servers facilitate the communications to reduce request overhead.

Even though we could identify the vulnerabilities described above in the sharing and version control systems implemented, we choose to do it this way because of its simplicity, as it satisfy the minimum security requirements and, in the way it was developed, could be improved to cover those vulnerabilities.

Conclusion

The design and implementation of the Remote Document Access was a challenging and interesting experience due to the fact, that it was the first time we had to design a secure architecture, and also the fact that during the implementation of the project we had to think about the implications of the code we were writing on the overall security of our project.

References

MySQL: was used because it provides a simple, fast, reliable and stable open source database management system.
Spring Boot: we used Spring RestFul web services to create secure channels between the client and the servers. This was possible because spring provides a simple way to enable HTTPS.
Bouncy Castle: was used to implement the cryptographic functions of the system. We choose this provider because it is well tested and documented and is also FIPS 140-2 Level 1 certified.
JDBC: Was used to establish a connection between the database and the authentication server.
JWT(Json WebToken): is a library that provides token signing/verification functionalities.