Ransomware-resistant remote documents
SIRS, A44

87687 - Miguel Coelho, 89513 - Nikoletta Matsur, 98675 - André Lopes

October 2020



1 Introduction

The aim of this proposal is to describe the design of a functional prototype of a system that allows to manage remote documents and that protects them from ransomware attacks.

2 Problem

Whenever there are documents that are to be accessed and edited by a group of people - sometimes simultaneously - there must be a platform similar to GoogleDrive, Overleaf or even GitHub to make collective work more productive and efficient. This need is common in different environments: from research documents to journalist reports, from corporate organizations to a group assignment from the local school.

Although each group of people may have the same motivation to use such platforms, their nature differ and so do their security requirements: a top secret file from a government's investigation unit has significantly higher demands in terms of security than does a student's weekly assignment. And this is true since the impact of the loss in each case is undoubtedly different. Therefore a system that protects remote documents from ransomware attacks is more oriented to support a more confidential environment. Curiously, these are the most prone to suffer attacks since they are more interesting and rewarding to the attackers.

Even though the main problem being solved is the loss of important documents (or the loss of monetary means in order to recover them) - hence the ransomware-resistance, it's also important to mitigate the risk of unauthorized access to the system, to hide the information from the files when the data is illegally accessed so that the confidentiality is preserved. Moreover it's best if the system can detect the attack to lower the damage and recover from attacks.

2.1 Requirements

- Users can create and edit files remotely.
- Documents should not be accessed by unauthorized parties.
- The owner of the documents must be able to give/revoke permissions other users.
- Documents can be synchronized between user devices and the remote server.
- Documents should be shared over a public network in a secure fashion.
- In case an attacker accesses the servers storing the documents, he must not be able to view the contents of the document.
- If an attacker tries to edit any document, there must be a way to detect the illegal modifications.
- There must be a way to recover the data in case of ransomware attacks.

2.2 Trust assumptions

Every entity in the system trusts the Certificate Authority that comes installed with the machine. If the attackers physically access the remote file server or the backup server, they can't turn off the machines or steal the hard-drives. The system administrators are fully trusted and they will manage the users' public keys in the main server. The server is agnostic to the content of each file and therefore isn't trusted.

3 Proposed solution

3.1 Overview

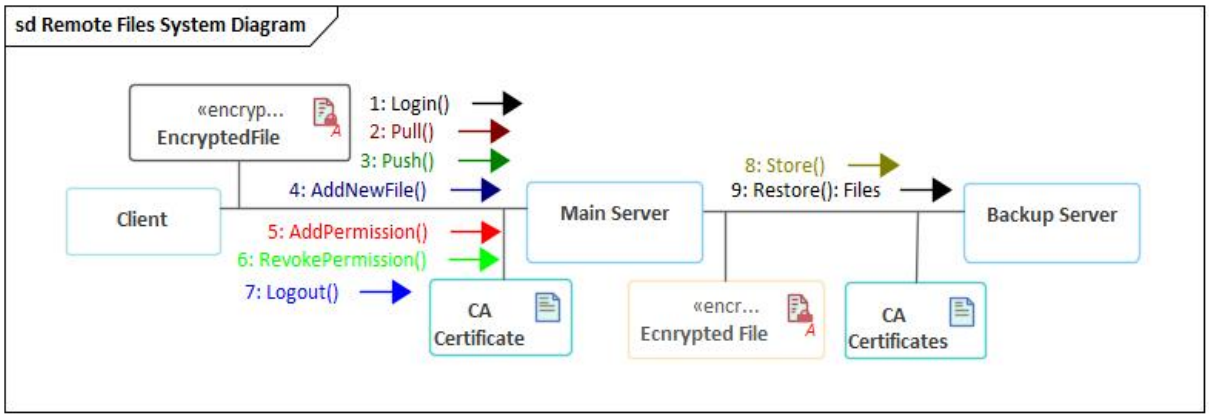

Figure 1: Diagram of the Remote Files System

Designing our solution we decided to create the simplest file-synchronization system so that we could focus on the security issues. We also considered to use an open-source system and to improve it's security but the developers of such systems put great effort in security as well so there wouldn't be much room for us to improve the system. This way, after an extensive search, we concluded that it would be best if we developed our system from scratch.

We decided to use a client-server architecture where the clients interact with the server to get, add and edit remote documents, and in order to do so they Add, Pull and Push a document to make remote changes to it. Client verifies server's entity in the TLS handshake through the server's certificate and public key.

The users authenticate to a client via operating-system username and password, each user can have multiple devices (Clients) and a device can have multiple users (logged one at a time). The users will also authenticate to server via username and password, which they can do from whatever client machine they could login. The permission to the documents will be managed by the owner of each document. There will be a Backup Server that serves the Main Server in storing the encrypted edited files and restoring files from in case of an attack. The periodicity of the backups is defined by the system administrator.

The intrusion detection will be executed in the Main Server through canary files in each folder and the integrity of the documents will be done by the server in Push and Pull commands with a signature appended to each document.

3.2 Deployment

In order to implement the described solution we will set up one server that is responsible for the secure storage of the documents and three clients pushing and pulling the remote files from the server to their devices and vice-versa. In addition to that we'll have one backup server to recover from possible attacks. The different machines will be interconnected through the use of remote procedure calls for the communication between clients and server, as well as the communication between the server and the backup server.

This solution is going to span across four networks, one of which is going to be divided into two virtual networks, where the main server and the backup server are located. The other three networks connect each client to the server.

There will be 3 VMs simulating client machines. Each of this machines may contain N operating system sessions. Each of which will be created by a system administrator and is going to contain a private and public key associated to the user for which the session was created. Is also important to mention that

a given user may have sessions in different machines with different credentials but only one user in the server. So the server will store, for each server user, multiple public keys.

NOTE: We are aware that centralized systems with one server granting all the functionality to it represents a single point of failure, but since the goal of this project is not the replication but the security, we are going to assume that the attacker cannot physically turn off the machine. One solution to this might be the Primary-Backup replication with at least 2 servers.

3.3 Secure channel(s) to configure

In our solution, the clients will communicate with the remote file server to do file synchronization and file permission management. The remote file server will communicate with the backup server to make periodical backups of the remote files.

For the client-server connection, the server will need to be authenticated using TLS and for the remote-backup server communication, both servers will be authenticated using TLS. The TLS connections will be implemented by a gRPC channel on top of HTTPS.

A fictional CA will be created to sign the remote and backup server certificates that will be used to authenticate the parties. We assume all machines have the CA certificates pre-installed.

3.4 Secure protocol(s) to develop

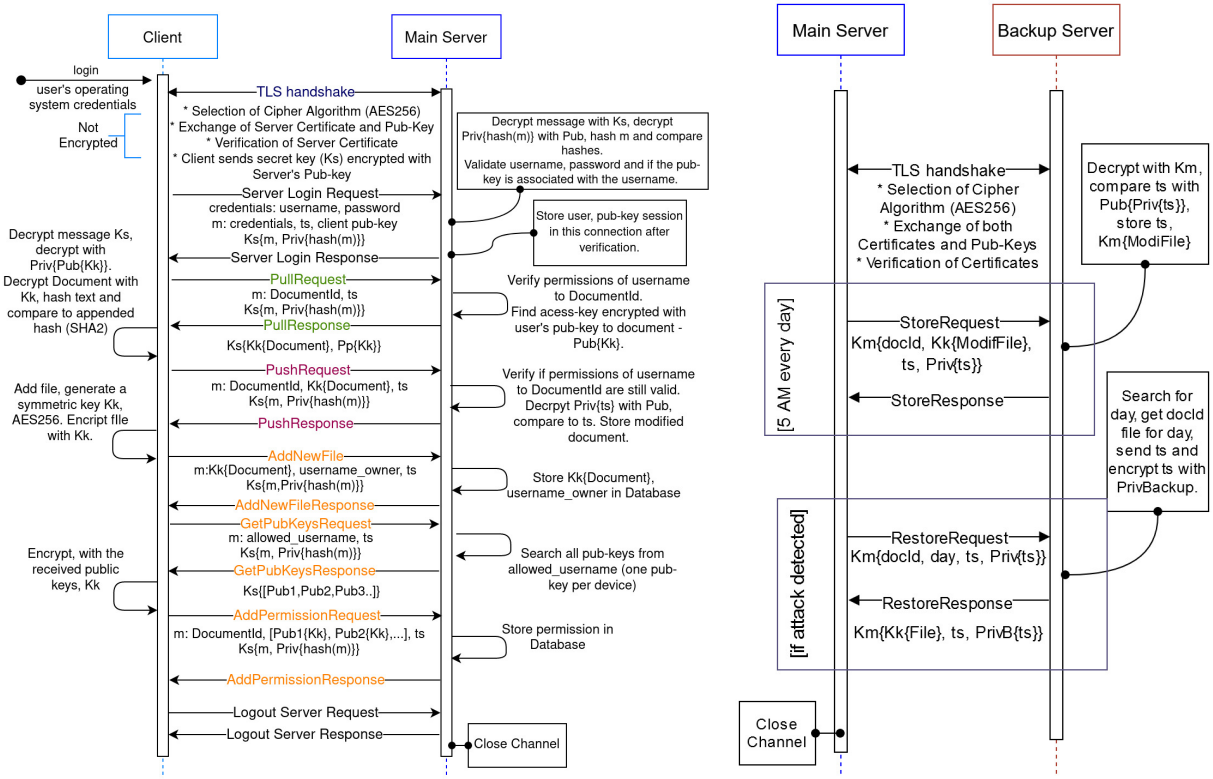

Figure 2: Sequence Diagrams of the Remote Files System

In Figure 2 sequence diagram we present, to the left, usual client-server communication and to the right the server-backup communication.

The secure properties to be ensured are user-client authentication: operating system username and password, client-server authentication - password and username to the server, client authorization - sym-

metric key (Kk) to decrypt file is encrypted with user's public keys (Pub{Kk}) for each allowed document, confidentiality - symmetric key encryption both ways agreed in TLS (valid for client-server (Ks) and server-backup (Km)), data (file) integrity - with a SHA2 text-hash appended to the end of each file (Ks{file+hash(file)}), message non-repudiation - each request is hashed and the hash signed with sender's private key (Priv{hash(m)}), freshness - use of timestamps in each request and message integrity - TLS with HMAC that was not represented in diagrams to simplify (same to the key revocation).

Our system will be developed in Java 11 as it's the latest LTS (Long Term Support) released version of Java, supporting the free public updates until 2027. Along with Java we'll use the SpringBoot Framework with JPA to make the persistent object management less laborious. To build the system we'll use Maven and the storage will be done in a MySQL database. The references to the tools are linked the name of the tool.

4 Plan

4.1 Versions

The basic version of the system allows users to login, both into the physical machines through a operating system session and to the server through a login and password. In addition, this version is also going to allow documents being shared over a public network in a secure fashion due to the use of TLS. Finally a user that created a document (owner) may give/revoke permissions to other users, which allow them to pull and push the document in question.

The intermediate version is going to add data recovery in case a ransomware attack is successful by backing up the documents to a backup server. The system administrator can define when this backups happen. This version is expected to be achieved by the week of 30 of November to 4 of December.

The advanced version, on top of recovering from ransomware attacks, also tries to detect and stop them in real time. This is achieved by using canary files, that allow the detection of attacks an consequent preventive actions such as turning off the connection to the attacker. We expect to conclude this version by the end of the week of 7 to 11 of December.

4.2 Effort commitments

Week	Miguel	Nikoletta	André
Nov 16 - Nov 20	VMs, network creation, CA certificates	Server: password management, database structure, sysadmin functions	Implement the client's cryptographic functions
Nov 23 - Nov 27	Connect different components and verify them		
Nov 30 - Dec 4	Backup-Server code	Server communication with client and backup	Client communication with server
Dec 7 - Dec 11	Canary Files, Debug, Report		
Dec 12. 17:00	Code/Report Deadline		

5 Conclusion

In this proposal we aimed to rigorously describe our design of a ransomware-resistant remote documents system. We are perfectly aware that we may commit some minor mistakes that will be obvious once we start developing it. Until then, we think we quite clearly expressed our ideas and plans to this project.