Grupo 9: Afonso Ribeiro - 89400, Duarte Santos - 89438, Sofia Carvalho - 89539

Highly Dependable Location Tracker

First Project for Highly Dependable Systems

The goal of this project was to develop a Highly Dependable Location Tracker that operates on a Client-Server architecture with multiple clients running simultaneously.

We chose to implement our application with RESTful services, using Spring Boot.

In order to implement a notion of time inside our project, we split time into epochs and designed, inside every client, a "step" command which increases the current epoch. This command can be called by any client and is propagated to every other client, so that the current epoch is synchronised amongst all clients. Each user automatically requests the location proofs regarding its current epoch before increasing it. As this is a simplification of the real world, we assume that the propagation of this command is always successful.

Threats and Protection Mechanisms:

To assure secure communication, a key pair for each user, server and health authority is generated before running the project. These are saved in a common directory for simplicity – we assume that each party can only retrieve public keys or their own private key.

- **Confidentiality, Integrity and Non-Repuditation:**

Concerning the communication between two clients, we assumed that the messages did not need to be confidential. We also assumed the server to be honest.

Regarding the communication between a client (including HA) and a server, we assumed that an attacker could read, drop, reject, manipulate and duplicate messages and, therefore, needed to assure confidentiality, integrity and non-repudiation while using this channel. To achieve this, every message sent is encrypted with an ephemeral symmetric key which is generated at the moment. The message is also signed using the sender's private key. The generated symmetric key is then encrypted with the receiver's public key and sent along with the encrypted message and signature.

For increased security when receiving messages from the server, we send information about an original request along with the server's response, so that it is possible for the client to verify that the response corresponds to what was originally requested.

- **Forged Location Proofs and Reports:**

Furthermore, there's the problem of having an attacker (or a byzantine user) submitting reports with false proofs. To mitigate this problem, a witness signs every proof that they send so that it cannot later be altered or repudiated – the proof contains the witness's information about the 'epoch', 'location', 'witness' and 'prover'. Furthermore, they check if they are effectively near the prover. Every time a report is submitted, the server discards any proofs that have a different 'epoch', 'location' or 'prover' than the ones alleged in the report. It also discards repeated proofs, proofs with a wrong signature, and proofs witnessed by the original prover. Finally, the server only accepts the report if it has a minimum of f+1 valid proofs (f is the number of tolerated byzantine users).

- **Freshness:**

Next, there's the challenge of assuring the messages' freshness to avoid replay attacks.

In the submission of location reports, since every report refers to a single epoch and user, if a second report with the same epoch and user is sent, it is refused by the server.

All other types of server requests are queries that don't change the server's internal state, but rather return back some sort of information. Since all requests are encrypted with a new symmetric key that is generated by the sender, the server should reuse it to encrypt the response. This way, if a response is replayed, the symmetric key used to encrypt it will be outdated, and the response is discarded. If a request is replayed, the attacker will not be able to read the response as the symmetric key is encrypted using the client's private key.

- **Server Persistence:**

Finally, to assure the integrity of the server's data, we are using a mysql database, which guarantees its persistence even in the case of server failure.