Highly Dependable Location Tracker

Afonso Ribeiro - 86752
Pedro Lamego - 89498
Beatriz Martins - 89526

Stage 1

1 Problem

The aim of this project is to implement a location tracker system that works with time being split into epochs. That system has the users periodically sending their location to a server, accompanied with a number of proofs guaranteed by the users neighbours to validate that the user is in the position it says. Users can also consult the server in order to know where they were in each epoch and there is a special user, the ha client, that can consult the location of all users across any epoch.

2 Proposed Solution

2.1 Design choices

The project was implemented using Rust. For the communication technology we opted to use a crate called tonic which is a gRPC implementation for Rust and for the cryptographic functions we used a crate called sodiumoxide.

The cryptographic part of the system requires the generation of the keys. That is done when the program starts and the keys are stored in a directory inside the security one called keys. There is a private and a public key for the server and all the clients, including the ha client have a signature key.

As mentioned in the definition of the problem users are required to send their location periodically to the server. They do this by sending a number of proofs, which are part of a report in our design. Besides the proofs the report includes the epoch, the location of the user who is sending it and the id that identifies that user. On the other hand, the proof includes the id of the user who requested it, the id and location of the user that signed it and the epoch it relates to.

2.2 Possible threats and protection mechanisms

There can be up to f byzantine users in the entire system and there is a certain limit imposed (f' < f) on the number of byzantine users that can be (or appear to be) nearby a correct user. Then, f' is such that all clients have at least f' + 1. With that in mind, the protection mechanism implemented was: each client tries to obtain 2f' + 1 proofs and sends a report as long as it gets f' + 1 (which by definition it would). The server only needs to verify f' + 1 correct proofs.

To ensure non-repudiation all reports are signed by the user who created them, which means that a user is not able to repudiate any previously submitted location reports. The communication among users is not confidential since it does not involve any security mechanism except for the signature mentioned before.

To ensure confidentiality and to protect against man-in-the-middle attacks in the messages exchanged between users and the server the requests are encrypted with a generated symmetric key which is encrypted with the public key of the server, meaning only he can decode those messages. Then, the server uses that symmetric key on the response. That means that the reports, besides being signed, are also encrypted.

With our implementation a byzantine client can only pretend he is in the neighbourhood where he appears to be. To prevent byzantine clients from submitting too many false reports we created a blacklist. It works in the following way: the server can detect a byzantine client if he submits another report with a different location for the same epoch and if that happens the client is blocked from the system.

2.3 Integrity guarantees

The system should protect against replay attacks. For that, the server has a set of nonces for each of the client requests.

Since it is requested that if the servers crashes data loss or corruption can't happen, we use a crate(atomicwrites) which takes advantage of moves being atomic in the same filesystem and that first writes to a temporary file before doing a move.

To ensure synchronization and that the data will not get corrupted we use read and write locks. This means that multiple threads can read the data in parallel but an exclusive lock is needed for writing or modifying data.