Highly Dependable Location Tracker

Highly Dependable Systems

Felix Saraiva 98752; Bruno Freitas 98678; Alexandru Pena 98742

Group 27

1. Design

1.1. System Overview:

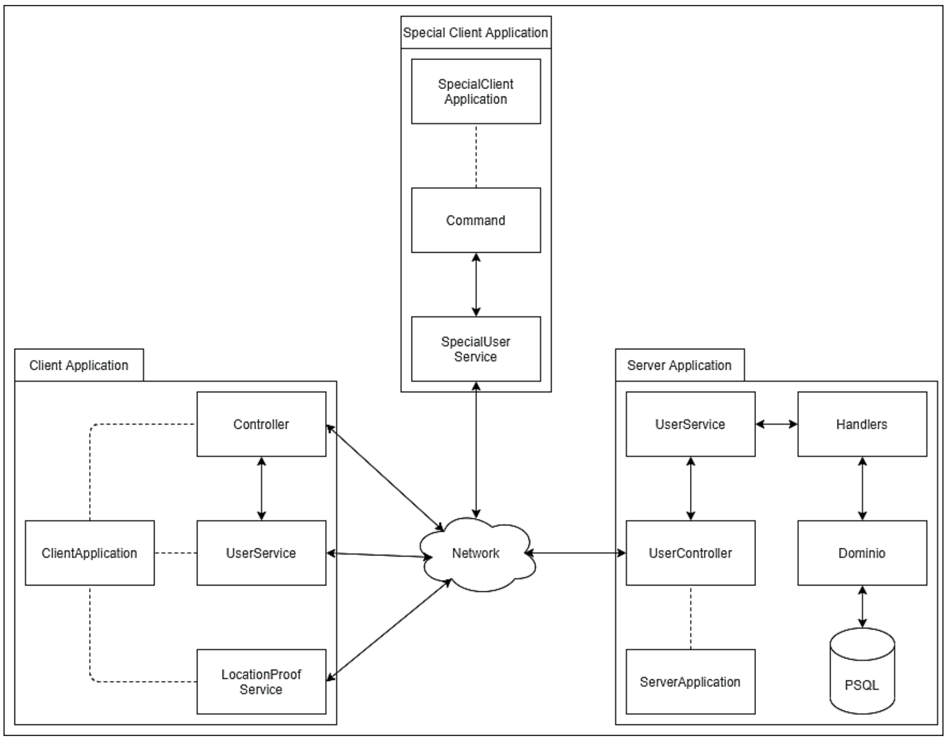

Figure 1 - System Architecture

Our system is divided into three main modules: the Special Client Application, Client Application and Server Application.

The Client Application is composed by a *Controller* that deals with the incoming Proof Requests and dispatches them into the appropriate services. The *LocationProofService* module is responsible for requesting the requested proofs and sending them to the Witness's *Controller*. The *UserService* module is responsible for the communication between Client and Server.

The Special Client Application provides a command line interaction where each input triggers a command that communicates with the *SpecialUserService* that establishes the communications with the Server (and with a Client to execute the respective Test Cases).

In the Server Application all the packets arrive to the *UserControler* which will then forward them to the appropriate service. The respective service contacts the necessary Handlers for the specific use case, (ex. CryptoHandler for Security verifications) and the Handler manipulates the Domain that interacts with the PSQL Database trough Hibernate framework and JPA API.

The Entire system is composed in an asynchronous way and using a REST based communication Technology (SPRING Boot).

1.2. System Communication:

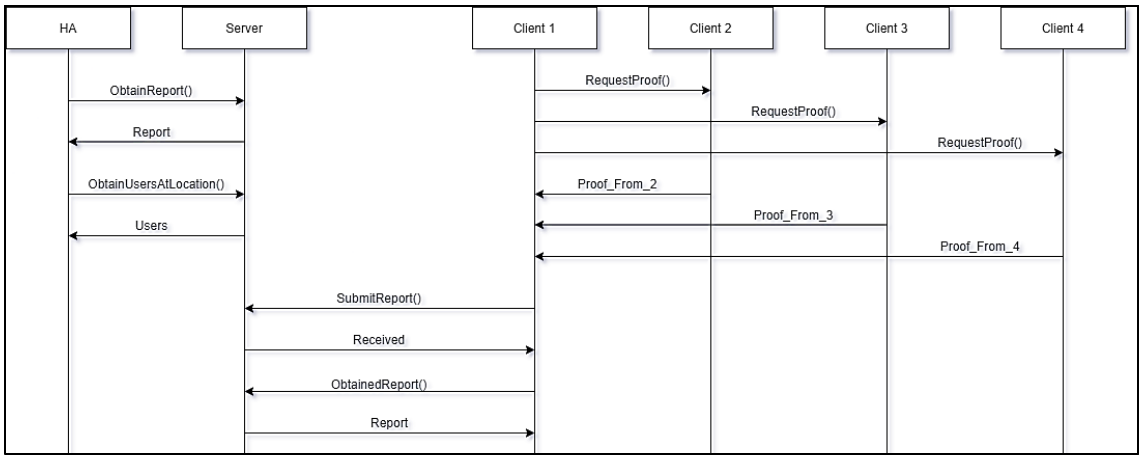

Figure 2 - System Sequence Diagram

The system communication starts with a RequestProofDTO by a Prover to the nearby

witnesses, which respond with a ClientResponseDTO encapsulating a ProofDTO or an error message. The Prover builds a ReportDTO with all the ProofDTOs provided plus the RequestProofDTO. The ReportDTO is then encapsulated into a SecureDTO, which encrypts it and sends it to the Server. All messages between Server and Client are encrypted.

After submitting the SecureDTO, the Server decrypts it and validates the ReportDTO, responding with an OK or an error message. When a Client requests a report, it sends a RequestLocationDTO, encapsulated in a SecureDTO, with all the necessary parameters. After the Server receives the RequestLocationDTO, it validates it and responds with a ReportDTO, again encrypted in a SecureDTO.

In a report request, the Server verifies if it comes from a regular user or from the HA. The HA works similarly to a regular Client with the difference that it can ask for any Report. It can also request, with a RequestLocationDTO, all the users at a specific location an epoch and obtains a SpecialUserResponseDTO.

2. *Protections*

2.1. Cryptography Implementations

The communication between Clients is not encrypted, although their messages are signed, adding nonrepudiation and integrity, preventing man in the middle attacks. To prevent Replay attacks the messages also contain Nonces. The Client also verifies if the respective request is in its range to prevent fake Proofs.

For Confidentiality, between Client-Server and HA-Server, a particular packet, SecureDTO, is exchanged. This packet's data is encrypted using AES/CBC with a unique session key per packet, generated from a randombytes field. These randombytes are protected using the Server's Public Key that encrypts the randombytes using RSA. For Non-Repudiation and Integrity, every packet is then signed using SHA256RSA.

From Client to Server each packet contains a unique nonce that prevents Replay Attacks. From Server to Client a Replay attack will not work because the packet will have a different session key per packet, as mentioned previously, so the victim will not be able to decrypt the replayed packet.

All Communications prevent Message Stealing because the Sender's ID is also signed and sent within the message. Forwarding Attacks fail because a user will drop the forward packet, since that same user did not start a communication with the attacker.

2.2. Logic Implementations

The Server also verifies if in the report submitted are duplicated proofs, if the proof belongs to submitted report and if the proofs and the request proofs are from the same epoch (a proof contains the request proof inside of it). These verifications eliminate the non-valid proofs and check if it has enough valid proofs to apply our byzantine rule, if the rule is verified then the report is valid.

To be accepted it still needs to be check if there is a report in the submitted report epoch.

2.3. Byzantine Rule:

There can be up to *f* byzantine users in our system, which implies that there are *3f+1* users. Following this logic, it is necessary to have a quorum of *2f+1* for a report to be accepted. For our system, the group assumed that there is *f = 1* byzantine users, therefore are needed 3*f+1 = 4* users, and the quorum must be *(2 * 1) + 1= 3*.

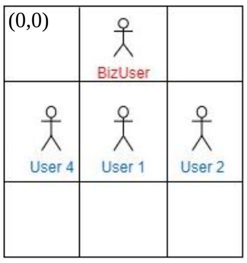

Figure 3 - Grid Example

In this example, let us assume that the BizUser states that he is at location (2,2) and sends a Proof request to all nearby users. All users will reply with a Proof because the BizUser is in range. However, both User 4 and User 2 will not have enough proofs to verify their location, and User 1 will only be verified if the BizUser behaves correctly.