HDS Notary - Stage 1

Mafalda Ferreira
81613

Leonardo Epifânio
83496

Pedro Lopes
83540

Group 11 - Alameda

1 Introduction

The goal of this project is to create a notary system that allows users to trade goods in a secure and reliable way. The system was implemented in Java and the communication is made via Java RMI.

2 Protocol

The main task of the notary is to legitimize a trade made between two users. This is achieved by performing a reliable **transaction** which is then logged and certified, using the Portuguese Citizen Card (CC from this point on) of the notary. A **transaction** is composed by the transaction ID, the user IDs of both participants (referred to as **buyer** and **seller** from this point on), the exchanged **good** ID, both digital signatures of the participants, and the digital signature of the **notary**, using the CC.

In order to perform the **transaction** T of **good** G, from **user** A to **user** B, certified by **notary** N, the following steps must be taken:

1. A sends ***intentionToSell*** request, for G, to N.
2. N sends positive response to A.
3. B sends ***getStateOfGood*** request to N.
4. N sends G's data to B, with ***isOnSale*** boolean valued true.
5. B sends ***intentionToBuy*** request to N.
6. N creates a pending transaction T for the good G. These pending transactions are saved in a Map that maintains a list of transactions for each good. When one of the pending transactions for a good is fulfilled, the other transactions for that same good are cancelled, and the good's pending transactions list is cleared.
7. N sends T object to B, containing a new transactionId, which will be used to uniquely identify the transaction and guarantee that B really wants to buy the good G.
8. B sends a ***buy*** request to A.
9. A sends a ***transferGood*** request to N.
10. N verifies that T is in the pending list, and verifies the participants. After that, G is transferred, T is certified, and the state of the notary is saved persistently.
11. N sends the new certified transaction T to A, which then sends it to B.

For each request and response in the system, the receiver verifies the **signature** of the sender. Before each request to the server, the sender asks for a **nonce**, which is used to identify the next request, and then verified by the server. We can see a representation of the protocol in Figure 1.

Request to Notary	$M \mid\mid n \mid\mid \{H(M \mid\mid n)\}_{K_C}$
Request to User	$M \mid\mid n \mid\mid \{H(M \mid\mid n)\}_{K_S}$

Table 1: Protocol messages

The structure of the messages traded in the protocol, for the ***intentionToSell***, ***getStateOfGood*** and ***intentionToBuy*** requests, is represented in Table 1, with M representing the message sent, n the request's nounce, H a hash function, K_C the client's private key, and K_S the server's private key.

Consider a transaction T. Let TID be T's ID, SID the seller's ID, BID the buyer's ID, GID the good's ID, $SSIG$ the seller's signature, $BSIG$ the buyer's signature, K_{Seller} the seller's private key, K_{Buyer} the buyer's private key, K_{Notary} the notary's CC private key, and H a hash function. The signatures involved in the transaction operation T are represented in Table 2.

When a client starts up, it requests the Notary's public key (from CC), which is sent to the client, signed with the server's private key.

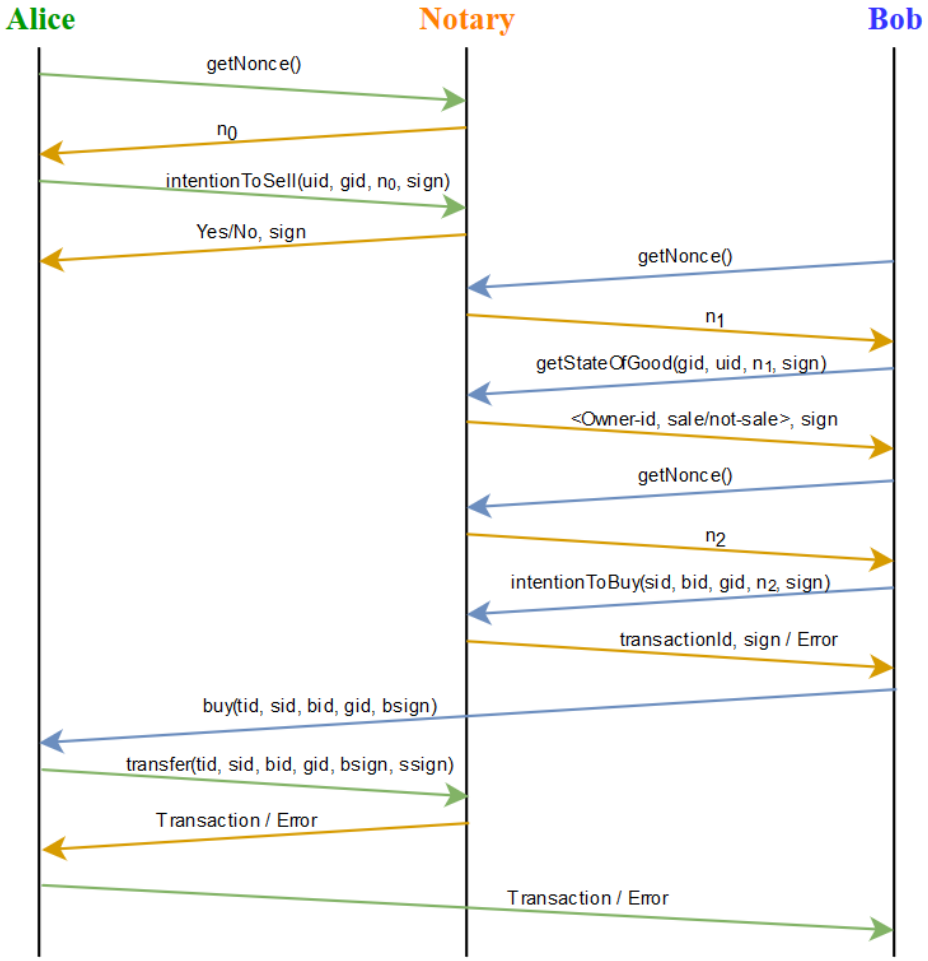

Figure 1: HDS Notary scheme

Seller Signature	$\{H(TID \,\|\|\, SID \,\|\|\, BID \,\|\|\, GID)\}_{K_{Seller}}$
Buyer Signature	$\{H(TID \,\|\|\, SID \,\|\|\, BID \,\|\|\, GID)\}_{K_{Buyer}}$
Notary Signature	$\{H(TID \,\|\|\, SID \,\|\|\, BID \,\|\|\, GID \,\|\|\, SSIG \,\|\|\, BSIG)\}_{K_{Notary}}$

Table 2: Signatures in a Transaction operation

3 Attacks and Assurances

The system ensures the dependability and the implementation requirements declared in the statement.

Spoofing - Every message has hashed information encrypted with the sender's private key, creating a verifiable digital signature.
Tampering - Every message contains hashed information encrypted with the sender's private key, alerting the receiver if the data is changed.
Non-Repudiation - Every message contains hashed information encrypted with the sender's private key, constituting a digital signature which authenticates the sender. It is highly unlikely that a user can create a digital signature for another person.
Replay Attack - Each time a user wants to make a request, it first requests a nonce to the server. A randomly generated number, using UUID, is created in the server and assigned to the user. In the next request, the user must use that nonce along with the message and the nonce cannot be used again.
Information Disclosure - There is information disclosure. For transparency, all communications are done in the clear, as stated in the dependability requirements.
Persistence - After every request, the server, saves its state in two files. The second file is meant as a backup, in order to prevent corruption. Since the save operation is sequential, if the server crashes, at maximum only one of the files will be corrupted.
Atomic Persistence - The system ensures synchronization, there are never two threads saving or changing the state concurrently.