TÉCNICO LISBOA

MEIC-A - Instituto Superior Técnico
Universidade de Lisboa

— Highly Dependable Systems —
(Sistemas de Elevada Confiabilidade)

Dependable Public Announcement Server

Stage 1

Students: Group 32
Emilio Basualdo Cibils
Nathan Edely
Florian Mornet

Professors:
Dr. Miguel Matos
Dr. Pedro Daniel Rogeiro Lopes

Repo: github.com/emiliobasualdo/Dependable-Public-Announcement-Server

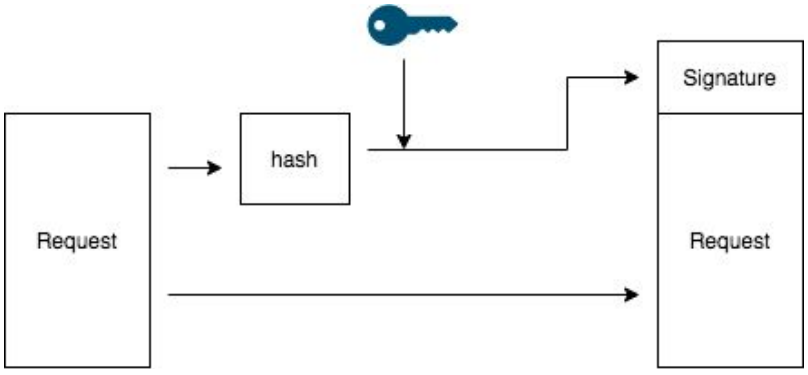
Contents

Introduction

We were requested to tackle a very basic yet not simple problem; maintaining resilience in an open and unencrypted communication between client and server while possibly being attacked. In this solution we cope with problems such as user accountability, information provenance and tapering of information during the communication.

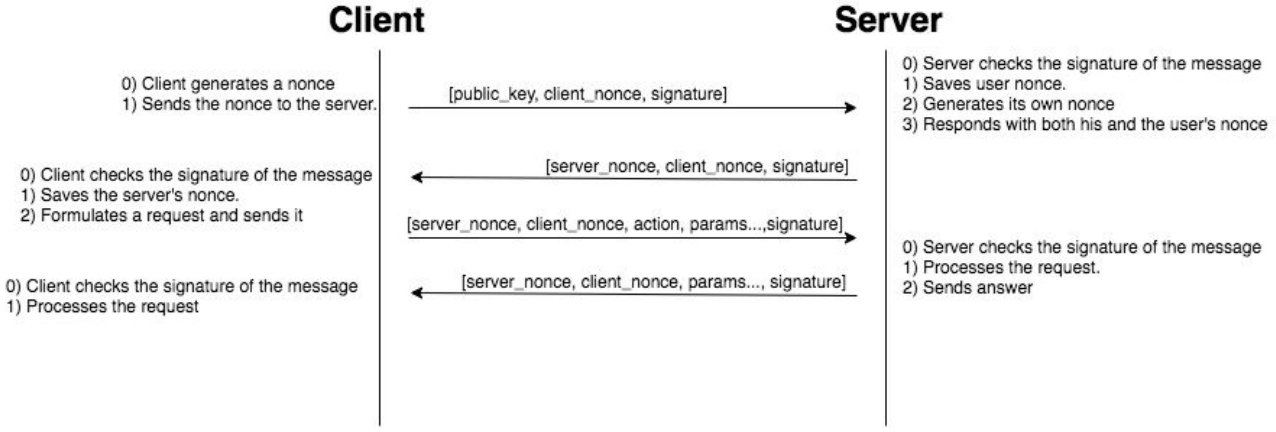
1. Design

In this protocol, every request is signed entirely before being sent by either party. This ensures that the receptor party is able to detect if the request was altered, intentionally or unintentionally, during the transport to destination.



The hash algorithm used is SHA-512. Though bigger and slower, it ensures near future collision-less hashes. The key used is the sender private key and therefore the algorithm used to sign is RSA.

The communication between server and client has two steps. During the initial one they transfer nonces amongst each other. During the second step, the user requests are attended.

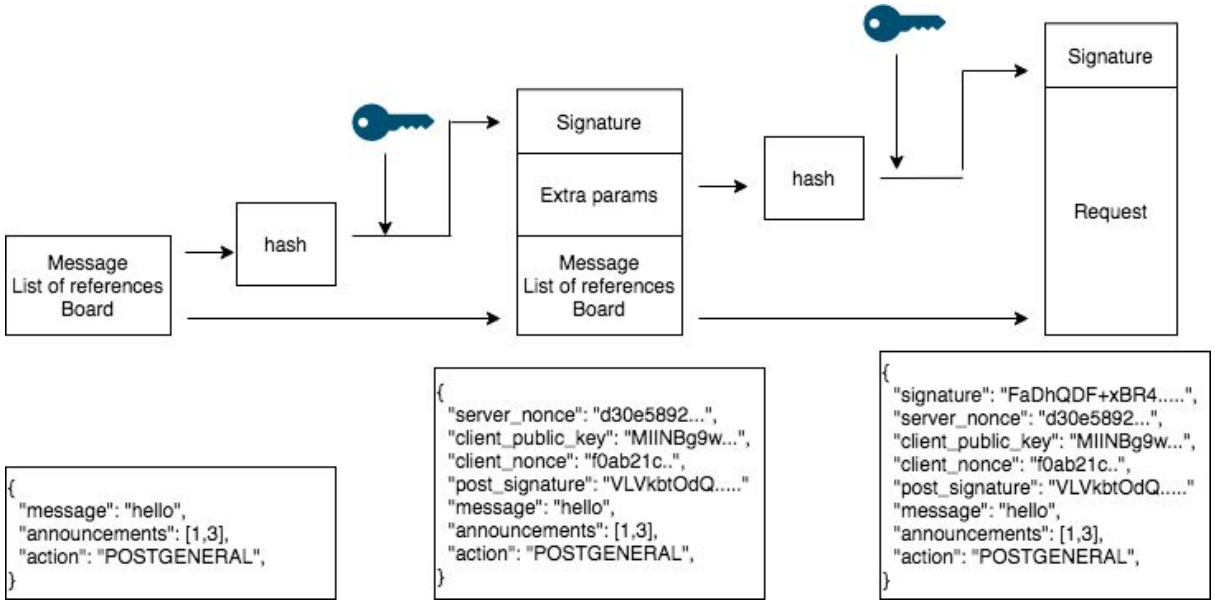


The nonces were implemented to prevent an attacker from sending a perfect copy of a package in the future. This is because both sides save each other's nonce and validate that they are contained in every package received. If the nonces don't match, it means that the package was not sent in the current communication, or sent at all, by the server. The nonces are generated with the Java UUID generator to ensure uniqueness of the communication's id.

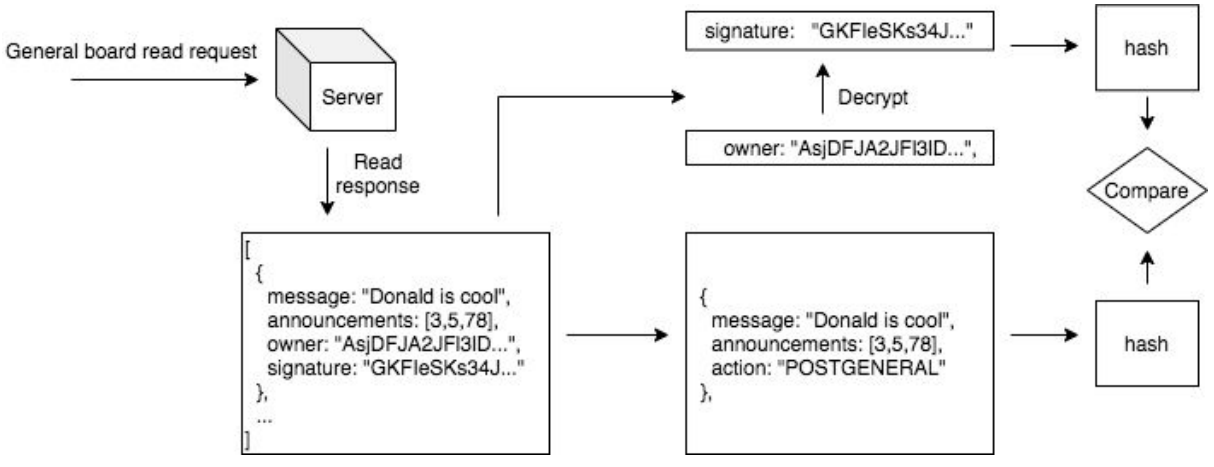
2. Integrity guarantees

Regarding announcement integrity, if a malicious client tries to repudiate a message sent by himself or in case a malicious attacker edits the contents of an announcement being read by a third client, this last client can always verify, by himself, if the contents of the announcement are authentic.

When a client posts an announcement, he must send a second signature inside his package. This signature is generated with the combination of: the message, the list of reference announcements and the board he is publishing to.



These three pieces of information are of public knowledge and can be hashed by anyone to verify its authenticity against the signature, also of public knowledge, using the writer's public key, also of public knowledge.



3. Other types of dependability guarantees

The protocol is also resilient to replay attacks because each part only expects to receive two packages. As the first package regards nonce checks, the second package can only be a client request (server point of view), or a server response (client point of view). After the second message, the protocol determines that the TCP channel must be closed.

We use TCP sockets for communication as it helps to deliver packages when there are possible network failures or intentional package drops (in comparison with HTTP, which is heavier and therefore less fast and efficient). We send the data inside the package as JSON strings using UTF-8 character set and Base64 to encode byte information, for simplicity when parsing and to ensure compatibility with most clients.

If higher level package drops occur, like a malicious attacker dropping the communication, the protocol determines a timeout of 50 seconds before closing the TCP channel to ensure consistency as both parties know what to do in such case.

The server's DB is SQLite and it runs a maximum of 10 threads to attend 10 different clients. Each thread is almost stateless and therefore a crash at any moment will ensure no data loss nor corruption. Recovery from a crash only requires the DB file and the key-store.

4. Problems

The protocol constantly signs with the sender's private key. We know that repeated use of the private key can lead to an attacker eventually reproducing such key, but this problem is out of the scope of this project and can be solved by generating symmetric keys on handshake.