TÉCNICO LISBOA

MEIC-A - Instituto Superior Técnico
Universidade de Lisboa

— Highly Dependable Systems —
(Sistemas de Elevada Confiabilidade)

Dependable Public Announcement Server

Stage 2

Students: Group 32
Emilio Basualdo Cibils
Nathan Edely
Florian Mornet

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

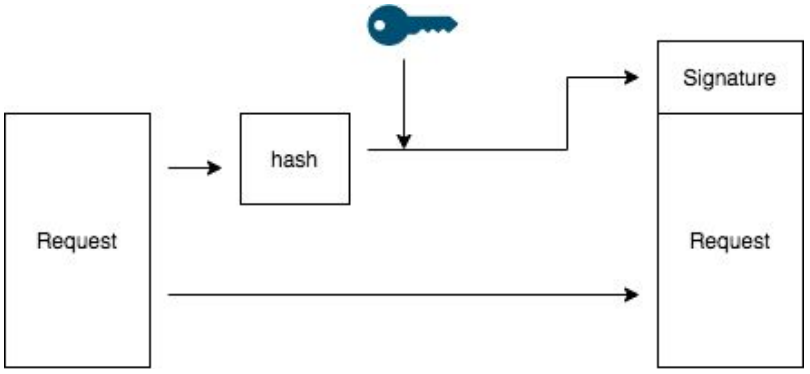
Repository: github.com/emiliobasualdo/Dependable-Public-Announcement-Server

Introduction

We were requested to implement (1,N) Byzantine Atomic Registers and a (N,N) Byzantine Regular register to prevent Byzantine clients and servers illegally altering the state of the distributed system. We maintained some of the protocol specifications designed for the first submission and changed others, as we are going to explain.

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 the destination.

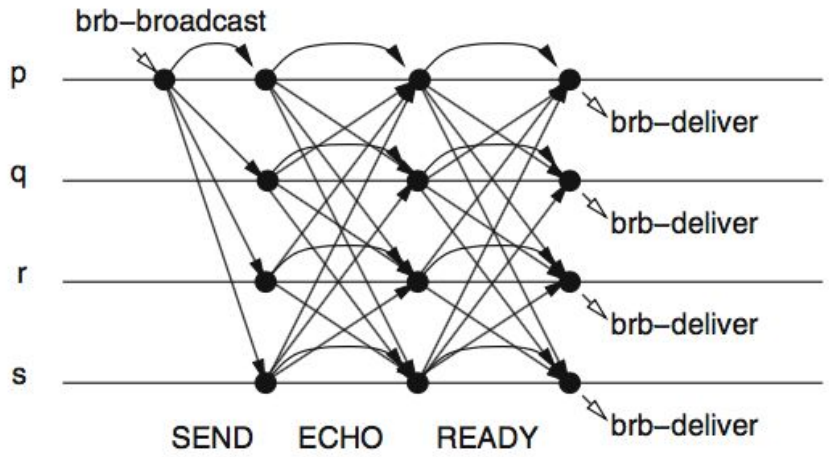


The hash algorithm used is SHA-512. Though bigger and slower, it ensures near future collision-less hashes. The key used is the sender private key and therefore the algorithm used to sign is RSA.

(1,N) Byzantine Atomic Register: To create such register we implemented Read-Impose Write-Majority (1,N) Atomic Register with digital signatures, as suggested in section 4.8.2 of the book, to create a Byzantine Atomic Register.

The communication between servers and clients has three steps.

1. During the initial one, the client broadcasts a request to all servers in which he adds a session nonce.
2. The second step ensures that all servers were requested to write the same message. To do this, the algorithm chosen is presented on page 118 of the course book: *Authenticated Double-Echo Broadcast*, a very basic and correct execution of this algorithm can be seen in the image below.



The authentication is done by means of a digital signature that, although being more computationally expensive than a simple MAC, is used in the Read-Impose Write-Majority as suggested in section 4.8.2 of the book.
Checks are being done, for example, the protocol first ensures that N>3f is verified (N being the total servers count, and f being the maximum faulty servers count). During the broadcast, if there are more errors received than the maximum allowed, an exception is raised.

3. The third step of the communications occurs when each server responds to the client with either an error or a success code. The client then checks to see that the answers differ in less than ⅓ of the total number of servers.

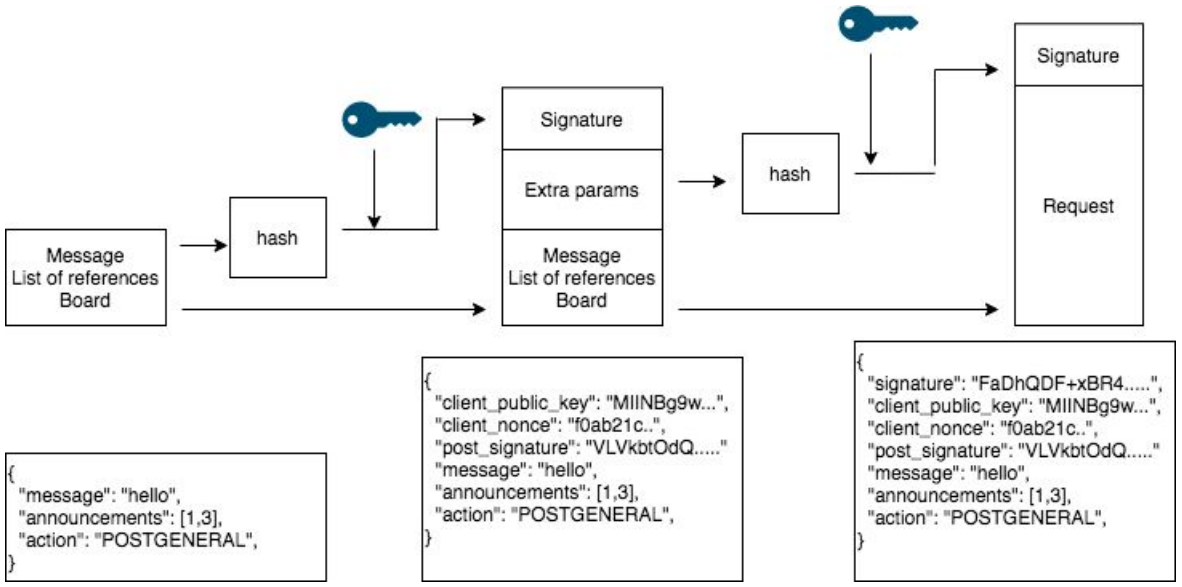
(N,N) Regular Register: This register is implemented based on the (1,N) Atomic Registers. Each server requests for all other servers to agree on which *write* will be executed first.

The nonces are implemented to ensure uniqueness of the communication ID and thus prevent an attacker from sending a perfect copy of a package in the future.

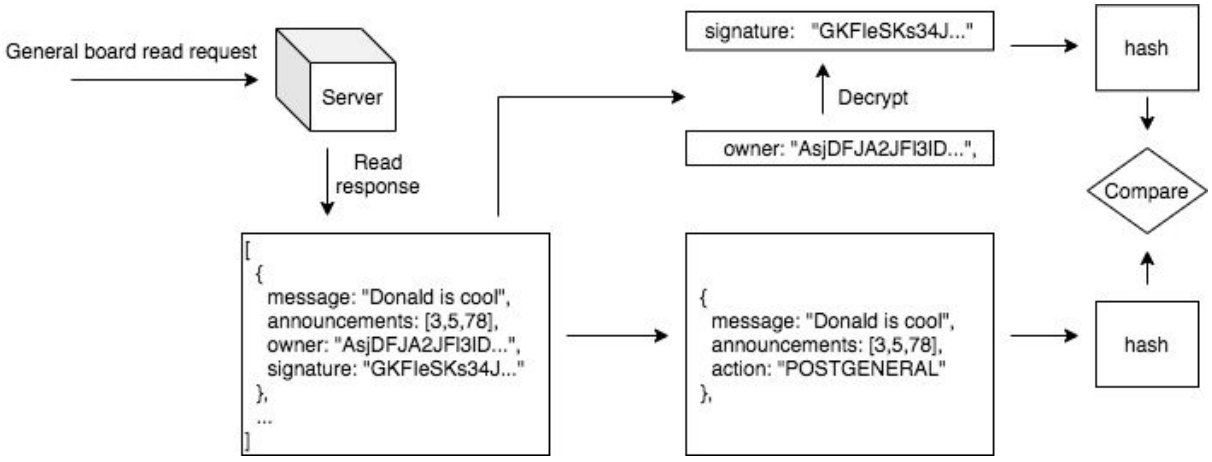
2. Integrity guarantees

Regarding announcements 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.

Server persistence is based on an SQLite database which provides ACID transactions. Each thread is almost stateless. Recovery from a crash only requires the DB file and the keystore.

All keys are created in a specific file which is directly setup in the creation of the different entities, in such a way that only the concerned entity can have access to its keys. It prevents the fact that private keys are personal and cannot be accessible to a stranger.

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.

Unfortunately, we did not manage to complete all the steps, mainly because we did not have the time to. The algorithms were quite hard to understand, thus making their implementation arduous.

This kind of implementation is thought to be bug-proof, having said that we have found a bug in which we did not check the public key published by the client in each request and hence was prone to duplication and edit attack. Nevertheless, we do not assure that our code is bug proof.