Dependable Public Announcement Server

MEIC-T T5

April 4th 2020

1 System Description

1.1 System Architecture

Our system was designed using:

- Programming Language - Java 13;
- Build Automation - Maven;
- Security Library - JavaCrypto;
- Distributed Remote Call System - gRPC;

We decided to implement the project using gRPC due to its simplicity and built-in multi-threading for handling multiple clients.

1.2 Implementation Decisions

1.2.1 Secure Channel

In this section, we will address possible security issues in the communication between the Library and the Server. Our system is designed assuming an attacker could influence the communication by either dropping, rejecting, manipulating or duplicating messages from the Library to the Server and vice-versa. The Library handles all security aspects of the communication with the Server.

When the Client initializes the Library, they must give the Library their Public and Private keys to be used for the security protocol. We chose this implementation because the Client Library is trusted and to lift the burden of handling security from the Client.

When the Client wants to communicate with the server, they must first register. To register, the Library communicates with the Server and proves the Client's identity by making a digital signature using SHA256 with RSA with the Client's private key. This combined with the Freshness guarantee that will be explained below, assures that the client is authenticated. The Client is identified by its public key in the system.

After registering, when a Client first sends a request, a new session between the Library and the Server is created using the Diffie-Hellman Protocol to generate a session key, which will be used in subsequent messages in order to avoid overusing the private keys from both sides.
For each of the abilities we assumed an attacker had, we have the following security counter-measures:

- **Dropping/Rejecting of messages**:
  Since gRPC uses TCP, we do not need to worry about attackers dropping packets, as the transport protocol will re-send them.
- **Manipulation of messages**:
  To prevent message manipulation by an attacker, we have decided to use HMAC with SHA256 using the Session key. For the creation of the session, we use the aforementioned digital signature to authenticate the parties involved.

- **Duplication of messages**:
  To avoid the duplication of messages, we use a combination of a Nonce with a Timestamp, being the Nonce a unique random number and the Timestamp working as a lease from the time when the message was created. The Server and the Client would check if the Timestamp is still valid by comparing with its own clock, calculating if the Timestamp is still in the window of the lease. If it was, they would compare with the previously seen Nonces if it was present.

  We then have a garbage collector, ran whenever twice the lease time has passed, which cleans the Nonces older than twice the lease time, in order to avoid clock synchronization problems. This way we avoid having to store nonces indefinitely, and thus overflowing the Server with Nonces.

  This implementation has some setbacks. First, by using physical clocks, we may encounter some problems with synchronization between the Client and Server, especially in the second phase of the project. Secondly, an attacker can change the order of requests made by the client, as long as all of them are made within the lease time.

  In the next phase we will implement logical clocks in order to avoid these problems.

1.2.2 Accountable Posts

We use Digital Signatures to assure the non-repudiation of posts, making clients accountable for their posts. All arguments of the announcement are signed together with the Client private key to assure integrity and non-repudiation. This signature is kept on the server for other Clients to verify. Additionally, the post is sent is encripted using RSA, to prevent an attacker from copying an announcement from a client and posting it before him.

1.2.3 Client Concurrency

Multiple clients may want to access the server and write or read from it at the same time. The gRPC implementation has its built in multi-threaded system for server connections, however two clients may still be writing or reading at the same time. In order to avoid concurrency issues, we use concurrency friendly structures, specifically a Concurrent Hash Map for the Client Boards and an Array List with synchronized locks for the General Board.

1.2.4 Persistence of Data

For data persistence, we save each data structure into a file on the server side. In order to avoid corruption from crashes in the middle of writing, we first save the structures in an auxiliary file, making sure the full structure is saved before substituting the old file atomically.

We save both boards and the Server generated IDs of the Announcements.