University of Pisa
School of Engineering

Distributed Systems and Middleware Technologies

DVoting - A distributed electronic voting system

Supervisors
Alessio Bechini

Students
Yuri Mazzuoli
Marco Pinna

September 10, 2022

Contents

Chapter 1

Introduction

DVoting is a distributed application for electronic voting, written in Erlang and Java. It was designed to be distributed among different nodes and to also guarantee the security aspects which are desirable in any election.
The work is organized as follows:

- In chapter 2 a general overview of the system is given, with a high level view of the application; functional and non-functional requirements are given for all actors involved in the system.
- In chapter 3 a more in-depth explanation of the application architecture is provided, together with the structure of the packets exchanged between the various nodes that compose the system.
- In chapter 4 all the Java components are listed and their respective functions are explained.
- Chapter 5 concerns the Erlang part of the codebase.
- Chapter 6 lists all the different databases in the system and their schema.
- Finally, chapter 7 shows some use usage examples of the app.

The entire codebase is available at
`https://github.com/MPinna/DVoting` .

Chapter 2

Overview

DVoting is a distributed electronic voting application that allows people to take part in a state election by going to the polling station, authenticating with personal ID and a smart-card, and expressing their vote on an electronic device in the polling booth.

Figure 2.1 show a high level view of the application.
When a voter enters the polling station, they authenticate and they enter the polling booth. In each polling booth there is an electronic device which is used to cast the vote. The voter authenticates with PGP in the polling booth and they are presented with a web page where they can express their voting preference. The vote is sent from the polling booth to the polling station, which takes care of marking that the voter has cast their vote.
The vote is then forwarded to the central station, where it is stored in an encrypted database until the end of the election. The central station also takes care of counting all the votes and computing the outcome of the election.

2.1 Functional requirements

The following functional requirements have been identified:

Voter

- An anonymous user should be able to login to the platform via PGP authentication.
- A logged user should be able to express their voting preference via web UI on the electronic device present in the polling station.

Polling station admin

- An admin user should be able to login to the admin dashboard via username and password.

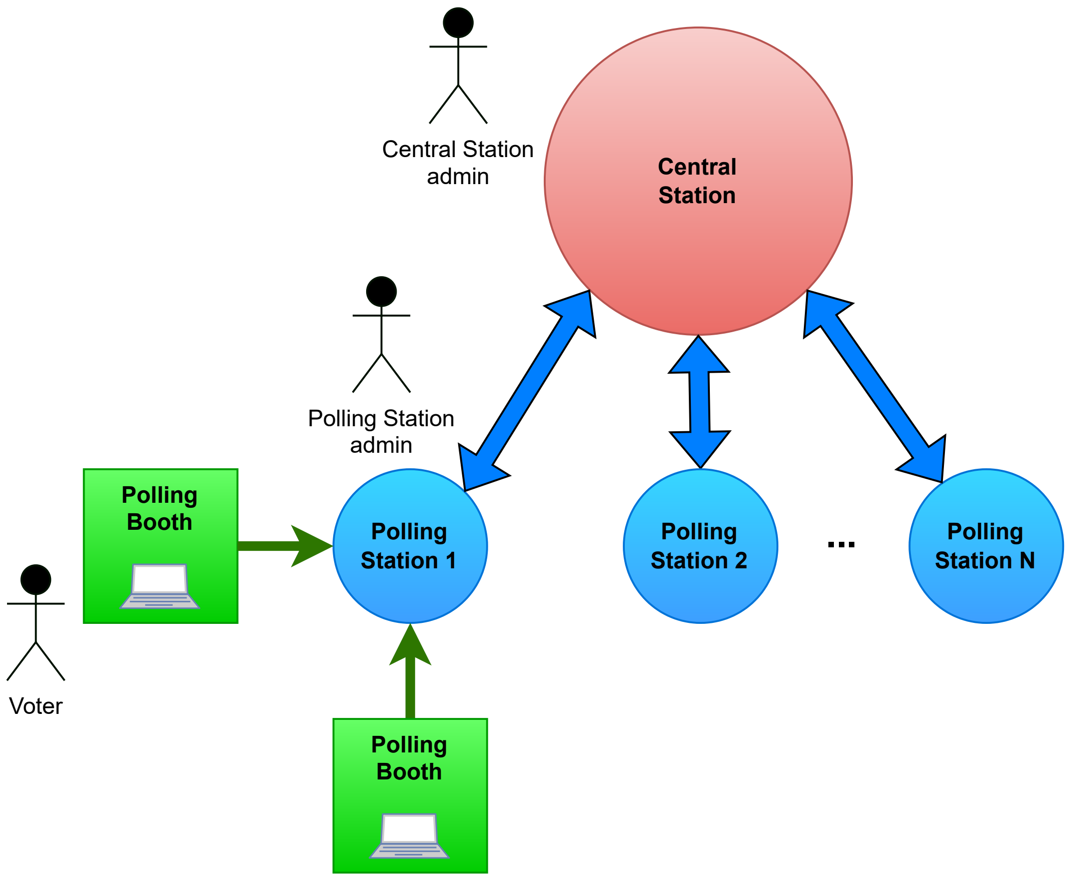


Figure 2.1: High level view of the application architecture.

- An admin user should be able to open the vote.
- An admin user should be able to temporarily suspend the vote.
- An admin user should be able to close the vote.
- An admin user should be able to view statistics related to the vote such as the turnout.
- An admin user should be able to manually add people to the list of the voters in particular cases (military, etc.)

System

- The system should remember which voters already expressed a vote.
- The system should aggregate the votes expressed in the single polling station.
- The system should aggregate the total counts of vote for each candidate coming from each of the polling stations.

2.2 Non-functional requirements

The application was designed to ensure security aspects which are to be guaranteed during any election, namely:

- Privacy and vote secrecy
- Double voting prevention
- Anonymity
- Authentication
- Authenticity
- Unlinkability

These security aspects are achieved through the use of different means. More in detail:

- Privacy and vote secrecy have been ensured by means of asymmetric encryption
- Double voting prevention is ensured by keeping track in a database of who voted and who did not.
- Anonymity and unlinkability are guaranteed by “splitting” the vote and the voter identity once the former is sent to the central station

- Authentication is ensured by face recognition upon entering the polling station and via PGP authentication.
- Authenticity is achieved via digital signature algorithm

Chapter 3

Architecture

This chapter analyses in greater detail the architecture of the application, showing which modules are present where and how they interact with each other. In the second part the structure of the packets exchanged between modules is presented.

When a voter enters the polling station, the personnel in charge authenticates them via face recognition and personal documents. After checking that the voter is registered to that polling station, they are allowed to enter the polling booth.
The voter authenticates themselves again inside the voting booth with an electronic document, such as an electronic identity card. On each polling booth there is a Tomcat instance running, which will serve the web page on which the voter will express their vote.
The voters' electronic document will also contain the private key of the voter ($Priv_{vot\_i}$), which will be used to sign the packet containing the vote; the packet signature will be verified on the polling station upon reception with the corresponding voter's public key (Pub_{vot_i}) contained in the *voter* database, to guarantee authenticity and integrity of the vote.
The vote, before being sent over the local network to the polling station server, is encrypted with the public key of the central station (Pub_{CS}). The polling station, therefore, is not able to decrypt the vote cast by the voter, and vote secrecy is thus ensured.
Once the polling station receives the voter's vote, it sets their corresponding flag `has_voted` in the *voter* database to `true`, to prevent double voting. At this point, the polling station node "splits" the voter id from the vote and only sends the latter, after signing it with its own private key ($Priv_{PS\_j}$). This ensures the unlinkability of the vote.
The central station receives the packet containing the vote, signed by the polling station, and verifies said signature using the corresponding polling station public key (Pub_{PS_j}) which is contained in the *pollingStation* database.
Lastly, the vote is saved in the *votes* database in encrypted form. When the voting is over (and only then), the votes will be retrieved from the database and decrypted.

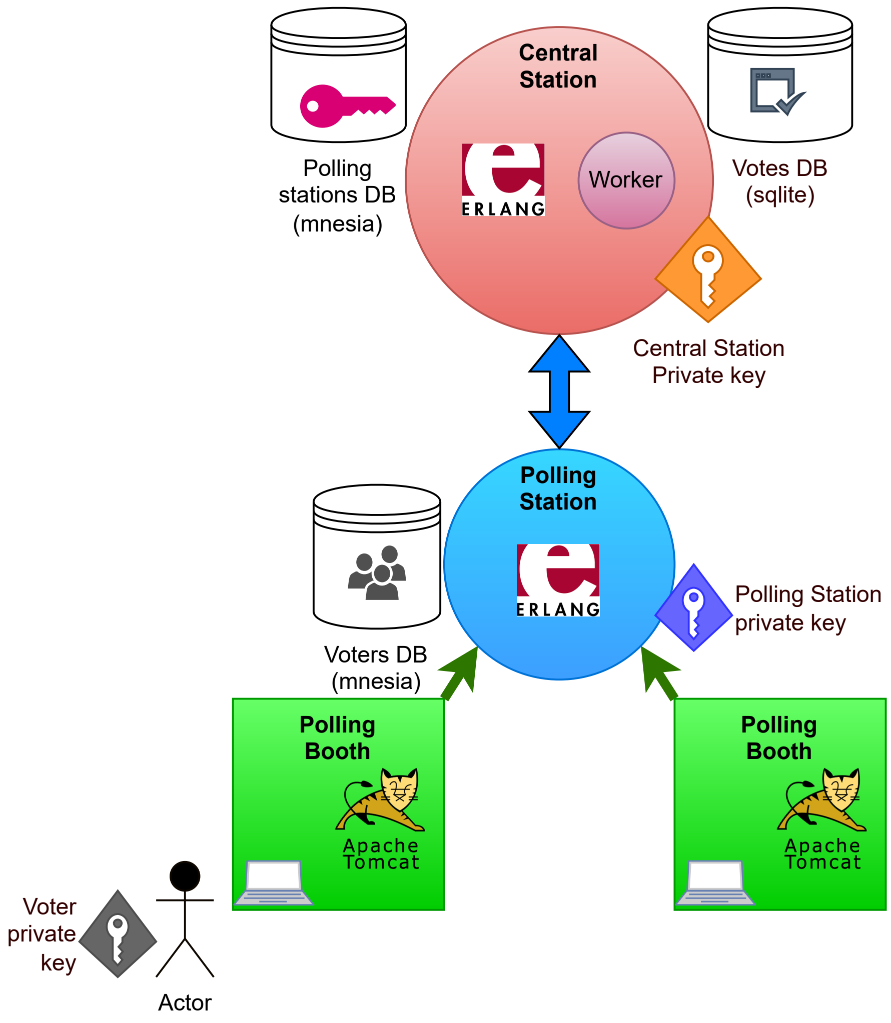


Figure 3.1: A more detailed view of the application architecture.

3.1 Packets structure

Chapter 4

Java

The application comprises four Java modules with the following structure:

```
DVoting
 ├─CentralStation
 │  └─src/main/java/it.unipi.dsmt.dvoting.centralstation
 │     ├─CentralStationDaemon
 │     ├─CentralStationDashboard
 │     ├─DatabaseManager
 │     └─VotesIterator
 ├─network
 │  └─src/main/java/it.unipi.dsmt.dvoting.network
 │     └─Network
 ├─WebApp
 │  └─src/main/java/it.unipi.dsmt.dvoting
 │     ├─AccessServlet.java
 │     ├─AdminServlet.java
 │     ├─BoothServlet.java
 │     ├─Candidates.java
 │     ├─Voter.java
 │     └─WebAppNetwork.java
 └─crypto
    └─src/main/java/it.unipi.dsmt.dvoting.crypto
       └─Crypto
```

CentralStation

The **CentralStation** module runs on the central station and handles the sending and receiving of messages and the interaction with the `votes` database.

- The `CentralStationDaemon` class is always running on the central station for the whole duration of the election. It takes care of receiving from the messages containing the encrypted votes from the polling stations and store them in the database.
- The `CentralStationDashboard` class provides a dashboard meant to be used by the central station admin. It allows them to start/stop the central station daemon and to also get the turnout of the election.
- The `DatabaseManager` class handles the interaction with the `Votes` database (cfr. 6).
- The `VotesIterator` class extends the `Iterator` Java util class and it is used when the election is closed and all the votes have to be retrieved from the `Votes` database.

Network

The **network** module is a utility module which provides to the Java classes the interface to interact with the Erlang modules. It uses the `com.ericsson.otp.erlang` Java package.

WebApp

The **WebApp** module runs on each polling station server.

- The `AccessServlet` class handles the authentication of the voter when they enter the polling station.
- The `AdminServlet` class implements all the administration-related functionalities: it authenticates and logs in the admin and allows them to perform management actions such as suspending, resuming or stopping the vote, search for a specific voter in the database or get the polling station turnout.
- The `BoothServlet` class provides the web page that will be server to the voter when they enter the polling booth.
- The `Candidates` class takes care of asking the official list of candidates to the central station and providing it to the voter in the polling booth.
- The `Voter` class is used to retrieve the voter information stored in the `voter` Mnesia database (cfr. 6).
- The `WebAppNetwork` class extends the aforementioned `Network` class, adding functionalities necessary to the web app.

Crypto

The **crypto** module is a utility module which contains all the cryptography related function use to generate keys, encrypt, decrypt, sign, verify messages, etc.

Chapter 5

Erlang

The application also includes the following Erlang modules:

```
DVoting
└─ Erlang
   ├─ centralStation.erl
   ├─ pollingStation.erl
   ├─ monitor.erl
   ├─ util.erl
   ├─ seggio.erl
   └─ voter.erl
```

- `centralStation.erl` runs on the central station. It handles the messages received from the polling stations and verifies their signatures. It also replies to requests coming from the polling stations for the list of candidates.
- `pollingStation.erl` runs on each polling station. Its main functionality is to receive the votes from the polling booths, verifying their integrity and authenticity with the voter's public key and updating the *voter* database by setting the corresponding flag.
  It also implements some of the administration functionalities by handling admin commands such as suspend/resume/stop the vote etc.
- `monitor.erl` acts as a supervisor for the other modules TODO complete
- `util.erl` contains constants and utility functions.
- `seggio.erl` contains the functions necessary to interact with the polling station database.
- `voter.erl` contains the functions necessary to interact with the voters database.

Chapter 6

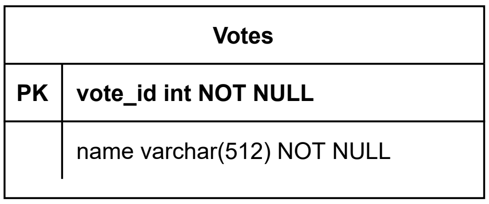
Database

A total of three databases have/has been created for the application:

- the `votes` database, which is located in the central node and stores all the votes in encrypted for the whole duration of the election;
- the `polling stations` database, which is also located in the central node and stores the information related to each polling station, included their public keys;
- the `voter` database. Each polling station has its own voter database, containing the information related to each voter who is registered to said polling station. It contains the voters' personal details, necessary for the identification, as well as each voter's public key.

6.1 Votes database

The `votes` database is a SQLite database located on the central node. It receives and stores in encrypted form all the votes cast by the voters during the election.
The database only consists of one table, `votes`, whose schema is the following:



Votes	
PK	**vote_id int NOT NULL**
	name varchar(512) NOT NULL

Figure 6.1: Schema of the `votes` table.

The interaction with this database is handled by the `DatabaseManager` Java class which offers an interface to perform CRUD operations.
The module that interact with the `votes` database are the `CentralStationDaemon` to

add the votes to the database and the `CentralStationDashboard` to obtain the election turnout. An example of how the table might appear during the election is the following:

TODO: add example of database

6.2 Polling stations database

The `polling stations` database is a Mnesia database located on the central node. It stores the information related to each polling station.
The database only consists of one table, `pollingStation`, whose schema is the following:

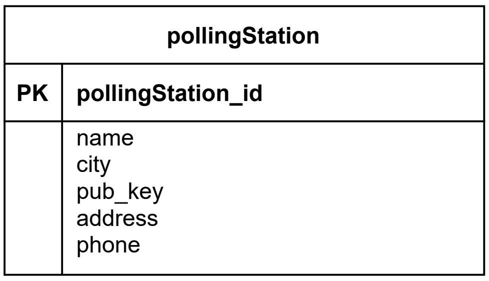


Figure 6.2: Schema of the `pollingStation` table.

It is worth noting that “Mnesia tables *have no built-in type constraints*, as long as you respect the tuple structure of the table iteslf.”[1].
The schema of this table is specified in the Erlang header file `pollingStation.hrl` while `pollingStation.erl` offers functions to interact with it, e.g. to add a polling station to the database or to retrieve a polling station public key given its id.
An example of how the table might appear during the election is the following:

TODO: add example of database

6.3 Voter database

A `voter` Mnesia database is located on each one of the polling station nodes. Each of them stores the information related to every voter who is registered to that polling station.
The database only consists of one table, `voter`, whose schema is the following:

[1] *Learn you some Erlang for great good*,Fred Hébert, 2013, p. 514.

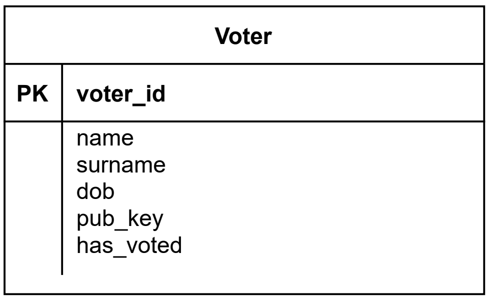


Figure 6.3: Schema of the `voter` table.

The schema of this table is specified in the Erlang header file `voter.hrl` while `voter.erl` offers functions to interact with it, e.g. to add a voter to the database, to retrieve a voter's public key given their id or to set the flag `has_voted` to *true* once the voter has cast their vote.
An example of how the table might appear during the election is the following:

TODO: add example of database

Chapter 7

Usage

TODO