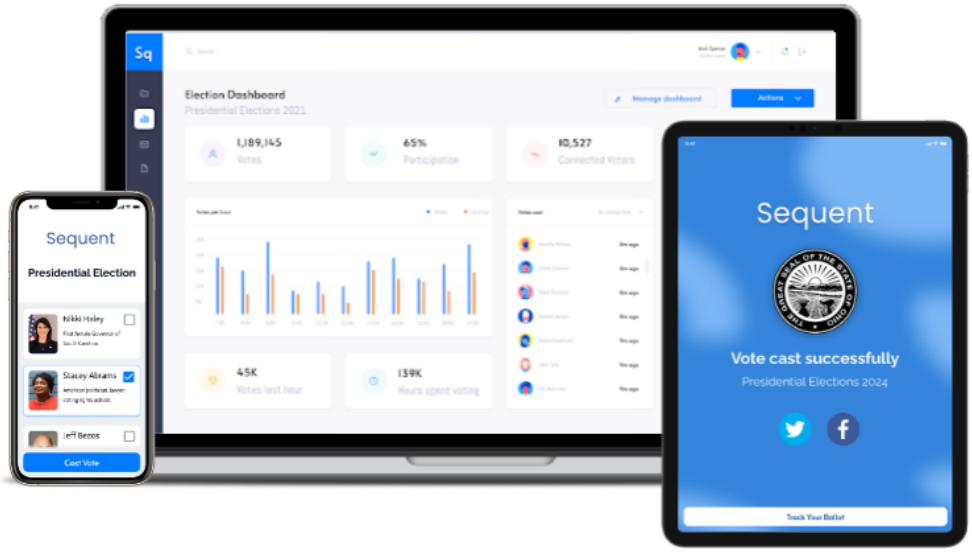



# Sequent Voting SystemArchitecture Overview

## (Work in progress)

Author: Eduardo Robles

| | |
|---|---|
| **Document identifier:** | 2022-04-10-arch-1 |
| **Date:** | 2022-04-10 |

Sequent

# Index

# 1. Introduction

## 1.1. Purpose and scope

This document provides an architectural overview of Sequent Voting Platform using multiple architectural views to depict different aspects of the platform.

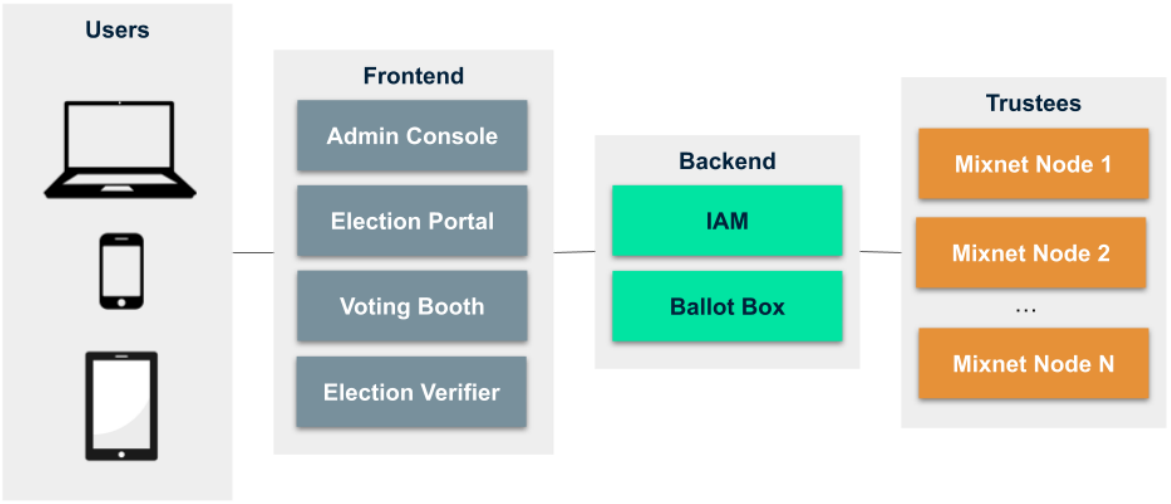
## 1.2. About

The Sequent Voting Platform is an open-voting platform developed by Sequent that makes voting more secure, transparent and accessible. It enables end-to-end verification (E2EV) of elections as well as supports the publication of results post tabulation. Under the hood, the Sequent Online Voting Platform leverages ElGamal encryption to ensure that votes recorded by electronic systems of any type remain encrypted, secure, and secret. Results can be published online or made available to third-party organizations for secure validation, and allow individual voters to confirm their votes were correctly counted.

# 2. Logical View

Presents a high level view including only the most important components and entities. Its main goal is to provide an understandable overall picture of the system.

## 2.1. Logical Architecture Diagram




## 2.2. Logical Components

### 2.2.1. Admin Console

The frontend web-based administrative component. It's used for set-up and configuration of an election, including all election metadata, performs credential management in connection to the IAM backend, manages the election lifecycle including start and stop of the election, generation of election and participation reports.

### 2.2.2. Election Portal

The frontend web-based component that serves as the public access to the election process for voters and observers. It can show election results, election information, election record

and allows voters to authenticate and access the verifiable election record and the different election verification methods.

### 2.2.3. Voting Booth

The frontend web-based software component used by voters to cast their ballots. This software runs on the voter's client device and displays a graphical user interface through which voters make their selections.

### 2.2.4. Election Verifier

Allows voters and third-parties to perform E2E verification on the election (cast-as-intended verifiability, recorded-as-cast and counted-as-recorded) using the election record provided through the election portal.

### 2.2.5. IAM

The IAM is the backend software component used to authenticate voters such that they can access the voting booth as eligible participants as determined by the Election Authority.

### 2.2.6. Ballot Box

The backend service that collects ballots. This service also interconnects with Trustee's Mixnet nodes to perform the anonymization and decryption of the votes, applying first vote cleansing, and performing the tally of the decrypted votes.

### 2.2.7. Mixnet Node

Backend service that jointly creates public keys, performs ballot shuffling and decryption. Most of the system cryptography is contained in this code. Since Mixnet nodes handle sensitive information, they are administered by multiple Trustees.

## 2.3. Actors

The most significant actors in an electoral process are listed below.

### 2.3.1. Election Organizer

The organization or institution wishing to carry out an electoral process. Election organizers use the Admin Console to manage the electoral process.

### 2.3.2. Voter

The people that the Election Authority has determined are eligible to participate in the electoral process as voters. Voters use the Voting Booth to cast their votes.

### 2.3.3. Trustee

A person or organization entrusted with sensitive information in order to safeguard ballot privacy. Trustees participate in key steps of the cryptographic protocol by managing Mixnet Nodes. Typically, trustees are either independent by themselves or independent among each other.

### 2.3.4. Auditor

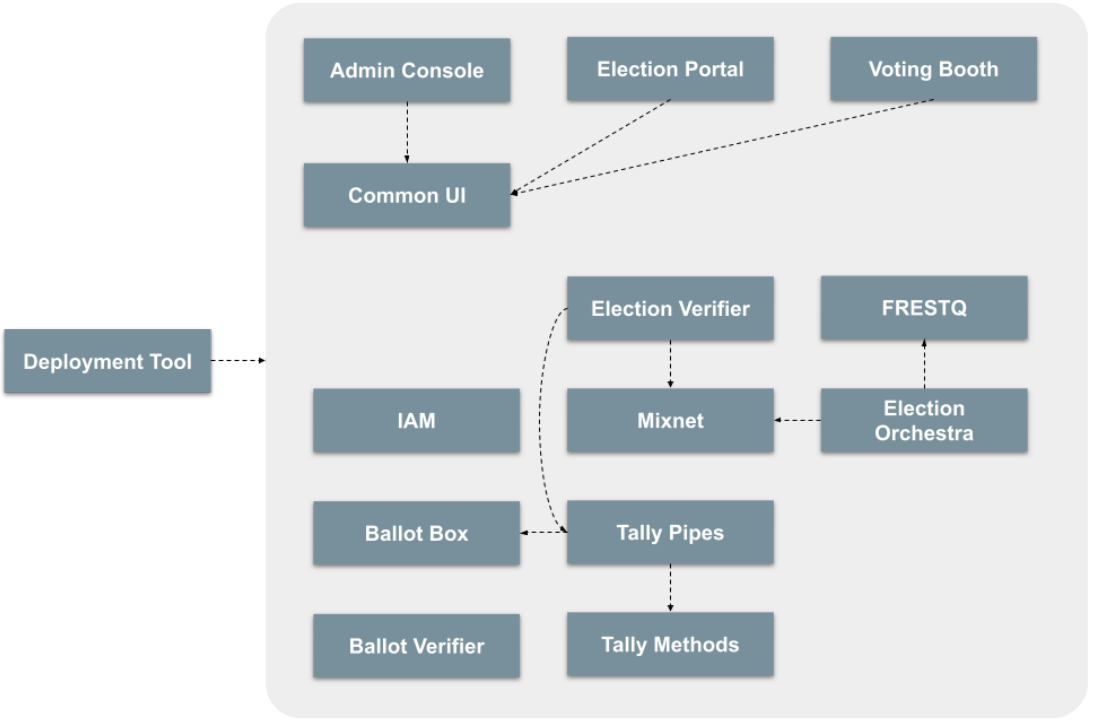
A person or organization that will perform different integrity and compliance verification procedures on the system and the election artifacts, including verification of election results, system logs or end-to-end verifiable proofs. Accredited independent auditors, researchers or even voters or members of the public can all perform some audits on Sequent's E2EV system. Auditors use the Election Verifier component to perform these verifications, and also the Election Portal to access the election record.

# 4. Implementation View

This view illustrates a system from a programmer's perspective, dividing the system into different software components. In our case, each distinct component has a corresponding Github Repository. This view only includes components directly developed by Sequent and excludes third-party components.

## 4.1. Components Diagram

The connections between the components depict the software dependency of one component of another.



## 4.2. Implementation Components

### 4.2.1. Admin Console

Repository: https://github.com/sequentech/admin-console

The frontend web-based administrative component. It's used for set-up and configuration of an election, including all election metadata, performs credential management in connection to the IAM backend, manages the election lifecycle including start and stop of the election, generation of election and participation reports.

### 4.2.2. Ballot Box

Repository: https://github.com/sequentech/ballot-box

The backend service that collects and stores cast ballots, written in Scala. This service interconnects with Trustee's Mixnet nodes to perform the anonymization and decryption of the votes applying first vote cleansing, and performs the the tally of the decrypted votes.

### 4.2.3. Ballot Verifier

Repository: https://github.com/sequentech/ballot-verifier

Reference cast-as-intended verifier written in C++. Allows voters to audit spoiled ballots, and performing the audit part of the Benaloh cast-or-audit mechanism.

### 4.2.4. Common UI

Repository: https://github.com/sequentech/common-ui

Common user interface library written in Javascript and based in AngularJS. Includes functions and components shared by the other three user-interface components: the Voting Booth, the Election Portal and the Admin Console.

### 4.2.5. Deployment Tool

Repository: https://github.com/sequentech/deployment-tool

Deployment ansible scripts. It serves to deploy the complete system for both master and slave machines, and also for trustee servers.

### 4.2.6. Election Orchestra

Repository: https://github.com/sequentech/election-orchestra

Election process orchestrator to create election keys and launch the tally. It's written in Python and Flask. It's the backend service being run by Trustee's Mixnet Nodes that interfaces between the Ballot Box and the Mixnet component. The Ballot Box requests the creation of the election keys or the launch of the anonymization and decryption of the votes of an election, and the Election Orchestra communicates one Mixnet Node with another to launch the Mixnet.

### 4.2.7. Election Portal

Repository: https://github.com/sequentech/election-portal

The frontend web-based component that serves as the public access to the election process for voters and observers. It's written in Javascript using AngularJS. It's a static SPA (Single Page Application) that interfaces through API calls with the Ballot Box and the IAM components.

It can show election results, election information, election record and allows voters to authenticate and access the verifiable election record and the different election verification methods.

### 2.2.8. Election Verifier

Repository: https://github.com/sequentech/election-verifier

Reference implementation of the election verifier. Allows voters and third-parties to perform the universal verification of the recorded-as-cast and counted-as-recorded properties of an election using the election record provided through the election portal. It uses the Mixnet code for some cryptographic primitives and the Tally Pipes to perform verification of the election results. It's a mix of Scala, Python and Bash code.

### 4.2.9. FRESTQ

Repository: https://github.com/sequentech/frestq

FRESTQ implements a federated rest task queue. It allows the orchestration of tasks with different peers with no central coordination authority. It's the low-level library used internally by election-orchestra to operate and communicate between Mixnet nodes.

It's developed in Python with Flask and SQLAlchemy with PostgreSQL. It uses its own message protocol for communication of tasks and updates between any two peers.

### 4.2.10. IAM

Repository: https://github.com/sequentech/iam

Backend component that provides authentication and authorization primitives. It's used to authenticate and authorize any user within the system and manage what permissions those users have. It's implemented in Python using Django and PostgreSQL. It also provides a Task Queue for slower tasks using Celery and RabbitMQ, for example to send email and SMS messages with OTP tokens.

### 4.2.11. Mixnet

Repository: https://github.com/sequentech/mixnet

A verifiable re-encryption mixnet, written in Java. Backend service that jointly creates public keys, performs ballot shuffling and decryption. Most of the system cryptography is contained in this code.

### 4.2.12. Tally Methods

Repository: https://github.com/sequentech/tally-methods

Library that provides the implementation of different voting systems supported by the platform, like STV or Borda. Written in Python. Used by the Tally Pipes module to calculate election results.

### 4.2.13. Tally Pipes

Repository: https://github.com/sequentech/tally-pipes

Python application that calculates election results using a pipeline. It's used by Ballot Box to perform the tally. It allows for complex tallying including vote consolidation, application of tie-breaker rules, incorporation of paper-ballot tally sheets into hybrid election results and application of gender or other parity constraints.

### 4.2.14. Voting Booth

Repository: https://github.com/sequentech/voting-booth

The frontend web-based software component used by voters to cast their ballots. This software runs on the voter's client device and displays a graphical user interface through which voters make their selections. It's written in Javascript using AngularJS. It's a static SPA (Single Page Application) that interfaces through API calls with the Ballot Box and the IAM components.

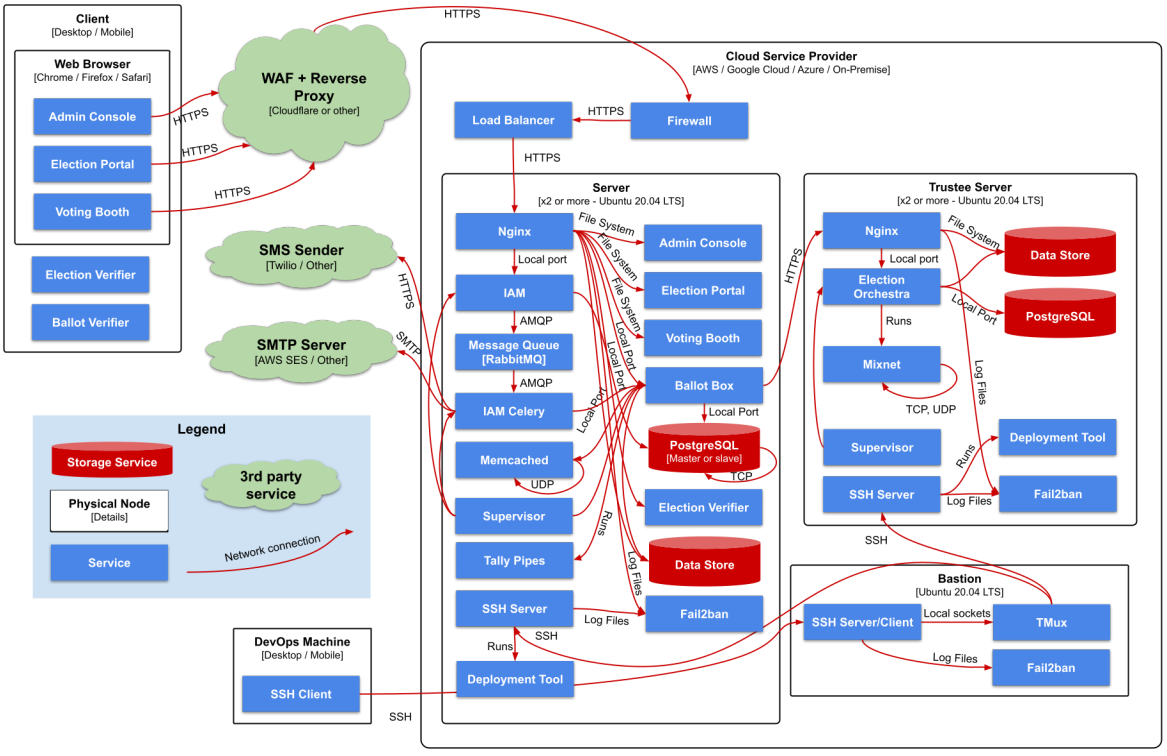
# 4. Physical View

Infrastructure point of view of the application. It describes the deployment of application modules in production and the technical components involved, including third party components and services.

## 4.1. Deployment Diagram

The deployment diagram shows:

- **Physical nodes**: the hardware or physical components. They can be virtual machines, a datacenter or a cloud provider.
- **Services**: Software components that run as a service inside each node.
- **Storage Services**: Database or filesystem storage device inside a physical node.
- **Connections**: How the different components are connected.
- **3rd-party services**: Services provided by third parties that are part of the deployment.



## 4.2. Physical Components

### 4.2.1. Client

The hardware device of the final user. The final user might be a Voter, an Election Organizer, an Auditor or any Third-party Observer. The hardware device can be a PC, Laptop, Tablet or Mobile Phone.

### 4.2.2. Web Browser

The browser agent of the final user. This can be Google Chrome, Safari or Mozilla Firefox, among others.

### 4.2.3. Admin Console

The frontend web-based administrative component. It's used for set-up and configuration of an election, including all election metadata, performs credential management in connection to the IAM backend, manages the election lifecycle including start and stop of the election, generation of election and participation reports.

This component appears depicted both inside the Client's Web Browser, where it's run, and within the Server, where it's provided and served from through Nginx.

### 4.2.4. Election Portal

The frontend web-based component that serves as the public access to the election process for voters and observers. It can show election results, show election information, election record and allows voters to authenticate and access the verifiable election record and the different election verification methods.

This component appears depicted both inside the Client's Web Browser, where it's run, and within the Server, where it's provided and served from through Nginx.

### 4.2.5. Voting Booth

The frontend web-based software component used by Voters to cast their ballots. This software runs on the voter's client device and displays a graphical user interface through which Voters make their selections.

This component appears depicted both inside the Client's Web Browser, where it's run, and within the Server, where it's provided and served from through Nginx.

### 4.2.6. WAF + Reverse Proxy

Third-party service that implements a Web Application Firewall and Reverse Proxy Cache. It provides DDoS attack mitigation, being the first line of defense. In many instances this third-party service is used to provide TLS session termination. It is typically provided by services like CloudFlare, Akamai or Amazon CloudFront.

### 4.2.7. Cloud Service Provider

The cloud service provider is a third-party service provider that provides access and management services to Virtual Machines and other physical hardware in which the deployment will be performed. This can be Amazon Web Services, Google Cloud, Microsoft Azure, Deutsche Telekom Cloud or even a service for a datacenter provided directly by the client.

### 4.2.8. Firewall

The Cloud Service Provider provides a managed firewall to restrict access by port and protocol to specific machines.

### 4.2.9. Load Balancer

A Load Balancer service managed by the Cloud Service provider. Depending on the requirements of the deployment the firewall might be directly to the "master" Server and high availability is provided in a 100% manual fashion, for small-scale elections where load balancing is not required.

### 4.2.10. Server

Server physical nodes run all the backend services to which the final users connect directly to. The deploy-tool ansible scripts are run in these nodes to install, configure and deploy all these services. For any deployment at least one master server is required, plus any number of slave servers. Servers are typically virtualized Ubuntu 20.04 LTS machines.

### 4.2.11. Nginx

Web server that acts as an intermediary between HTTPS traffic and the backend services such as the IAM and the Ballot Box. It also serves static content such as the election artifacts and the front-end Single Page Applications: the Admin Console, the Voting Booth and the Election Portal. Nginx is also configured for reverse-caching and rate-limiting requests as additional measures for DDoS attack mitigation and increased load performance.

### 4.2.12. IAM

The IAM is an API backend service used to authenticate voters such that they can access the voting booth as an eligible participant as determined by the Election Authority. It only executes tasks that can be executed immediately, deferring slower and more complex tasks to the IAM Celery background service. The IAM connects to the PostgreSQL database where all the data models of this service are stored. If the node running the IAM is a slave node, it connects to the master node to allow read-write queries.

### 4.2.13. Message Queue

The celery message queue used to send tasks from the IAM to the IAM celery background service is provided by RabbitMQ. The queues are local to each Server node, which allows for better scaling.

### 4.2.14. IAM Celery

Service providing the execution of longer-running tasks related to the IAM. It's an integral part of the IAM software component and it's also written with Django, Python and PostgreSQL, but it is run as another background backend service, connected to the IAM through the usage of a local Celery message broker. It interacts with third-party services such as an SMS Sender

service or an Email Sender service, and also connects to the Ballot Box to launch some tasks there.

### 4.2.15. Memcached

Memcached is a general-purpose distributed memory-caching system used by the Ballot Box service. The memcached instances of the different Servers are connected through each other using UDP to maintain their local caches in sync.

### 4.2.16. Ballot Box

The API backend service that collects ballots. This service also interconnects with Trustee's servers Election Orchestra service to perform the anonymization and decryption of the votes applying first vote cleansing, and launches the the tally of the decrypted votes using the Tally Pipes application. Once the tally is complete, it generates the election record and stores it in the Data Store. Election configuration and cast ballots are stored in a PostgreSQL database, and API queries are cached using memcached.

### 4.2.17. PostgreSQL

Relational database management service used within Server and Trustee physical nodes to reliably store the data of other services like the Ballot Box, the IAM and Election Orchestra. In Server nodes, PostgreSQL is typically used in a master-slave configuration, in streaming replication mode using repmgr. Three different kinds of backups are configured to be performed within each physical node: archived SQL Dump snapshots, WAL backups and continuous archiving of the WAL files.

### 4.2.18. Election Verifier

Reference implementation of the election verifier. Allows Voters and third-parties to perform the universal verification of the recorded-as-cast and counted-as-recorded properties of an election using the election record provided through the election portal. It uses the Mixnet code for some cryptographic primitives and the Tally Pipes to perform verification of the election results. It's a mix of Scala, Python and Bash code.

This component appears depicted both inside the Client, where it can be run, and within the Server, where it's provided and served from through Nginx.

### 4.2.19. Ballot Verifier

Reference cast-as-intended verifier written in C++. Allows Voters to audit spoiled ballots, performing the audit part of the Benaloh cast-or-audit mechanism. It's executed in the Client device.

### 4.2.20. Data Store

Both the Server and the Trustee Machine make use of a Data Store. These data stores are simply directories within the filesystem of the machines. In the case of the Server physical nodes, the datastore stores the election record, the election results in different formats like JSON, text and PDF, and also the dump of the encrypted ballots that is sent to the Trustee Machines to perform the Shuffling and Decryption process within the Mixnet. As Server nodes are typically configured in a Master-Slave setting, this Data Store is replicated periodically using rsync to the slaves. In the case of the Trustee Server, the Election Orchestra uses the Datastore to store both the private keys and all the binary information generated by the Mixnet.

### 4.2.21. Tally Pipes

Python application that calculates election results using a pipeline. It's used by Ballot Box to perform the tally. It allows for complex tallying including vote consolidation, application of tie-breaker rules, incorporation of paper-ballot tally sheets into hybrid election results and application of gender or other parity constraints.

### 4.2.22. Supervisor

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. It's used to manage and launch the unix processes related to Election Orchestra, the Ballot Box, the IAM and IAM Celery within the Server and Trustee's physical nodes. Supervisor is responsible for log rotation and log

management of these applications and also configures the number of unix processes to be launched for each of these services.

### 4.2.23. Fail2ban

Intrusion prevention service being run within the physical nodes that protects computer servers from brute-force attacks. Processes server logs from Nginx and SSH and is configured to manage iptables rules to block connections stemming from attackers.

### 4.2.24. SSH Server

Service that provides remote login and command-line execution within the Physical nodes. Direct access to production machines (Servers and Trustee Servers) from the Internet is not allowed. Only access through the Bastion physical node is provided to specific personnel and using secure cryptographic keys.

### 4.2.25. Bastion

The Bastion physical node is the only server that accepts connection from the outside. These connections are rate-limited using Fail2ban and only the DevOps team has access to the Bastion node. SSH access is configured to only allow the usage of approved secure public keys.

### 4.2.26. TMux

Terminal multiplexer used in the Bastion node to automate having different sessions to the different nodes for easy management and review, and saving long standing sessions without losing the state when disconnecting from an ssh session.

### 4.2.27. Election Orchestra

Election process orchestrator to create election keys and launch the tally. Runs in the Trustee nodes. It's written in Python and Flask. It's the backend service being run by Trustee's Mixnet Nodes that interfaces between the Ballot Box and the Mixnet component. The Ballot Box requests the creation of the election keys or the launch of the anonymization and decryption

of the votes of an election, and the Election Orchestra communicates one Mixnet Node with another to launch the Mixnet.

### 4.2.28. Mixnet

A verifiable re-encryption mixnet, written in Java. It's a backend service that runs in the Trustee physical nodes launched and managed by the Election Orchestra service to jointly create public keys and perform ballot shuffling and decryption. Most of the system cryptography is contained in this code. Generated artifacts are stored in the local filesystem using the Data Store. The Mixnet service of the trustees needs to communicate with each other. The Mixnet does this intra-communication using both a TCP port to communicate all the data and an UDP port for fast signaling.

### 4.2.29. DevOps machine

This is the PC or laptop machine of the DevOps operators managing the deployment.

### 4.2.30. SSH Client

The SSH Client is used both in the DevOps machine and in the Bastion physical node.

### 4.2.31. Deployment tool

Deployment ansible scripts. It serves to deploy the complete system for both Server and Trustee Server physical nodes.