Scytl

Innovating Democracy

Scytl sVote

Software Architecture description

Software version 2.1

Document version 1.0

Scytl - Secure Electronic Voting

STRICTLY CONFIDENTIAL

Table of contents

List of Figures

List of Tables

1 Introduction

This document provides a brief description of the architectural cornerstones of Scytl sVote. It highlights the relevant requirements and the driving forces that development teams must consider.

1.1 Requirements overview

The main goal of Scytl sVote is to provide voters with the possibility of casting their vote online in a secure manner. In this sense, it covers all the process from the configuration of the electoral event to the post-electoral activities, while following a rigorous and secure cryptographic protocol. The description of this protocol can be found in the document Scytl sVote Protocol Specifications.

1.2 Quality goals

Scytl sVote should fulfil for instance the following quality standards (not exhaustive):

- Scalability.
- Accessibility and responsiveness.
- Data separation.
- Universal verifiability and individual verifiability.
- Performance efficiency.

1.3 Constraints

The development process of Scytl sVote is subject to some organizational and procedural constraints. The development process and its constraints are described in the document Scytl Software Development Life Cycle.

2 Context and scope

Scytl sVote is used by different organizations (private or public) to provide internet voting to a subset or to all of its eligible voters.

The system context is described below:

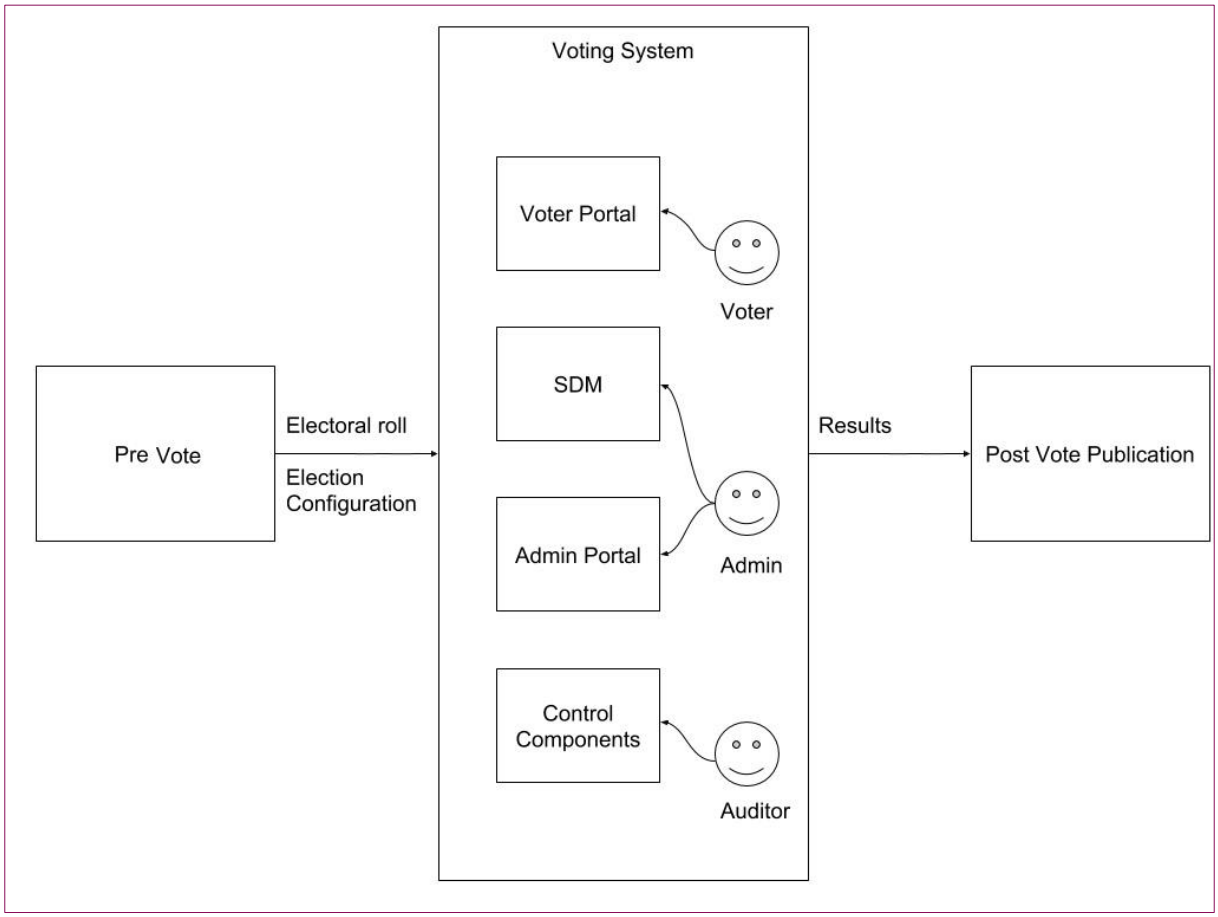

Figure 1 – Scytl sVote context

A pre-voting system is used to manage the electoral roll together with candidates and party lists in case of elections, and questions and answers in case of referendums. The pre-voting system delivers all the information about the electoral roll and the configured election to the voting system.

The post-voting system receives the results, i.e. the number of votes per voting option for a given counting circle and is responsible for the publication of results.

Scytl sVote is defined as a solution taking place between the pre-voting and post-voting system. It includes different roles:

- Voters who receive credentials and access to the Voter Portal to cast their votes.
- The election administrator (Admin), who is responsible for configuring the system.
- The administration board (Admin Board), who validates the configuration.

- The Electoral Authority[1], who is responsible for holding the election's private key to decrypt votes.

These actors are explained in greater detail in the figure below.

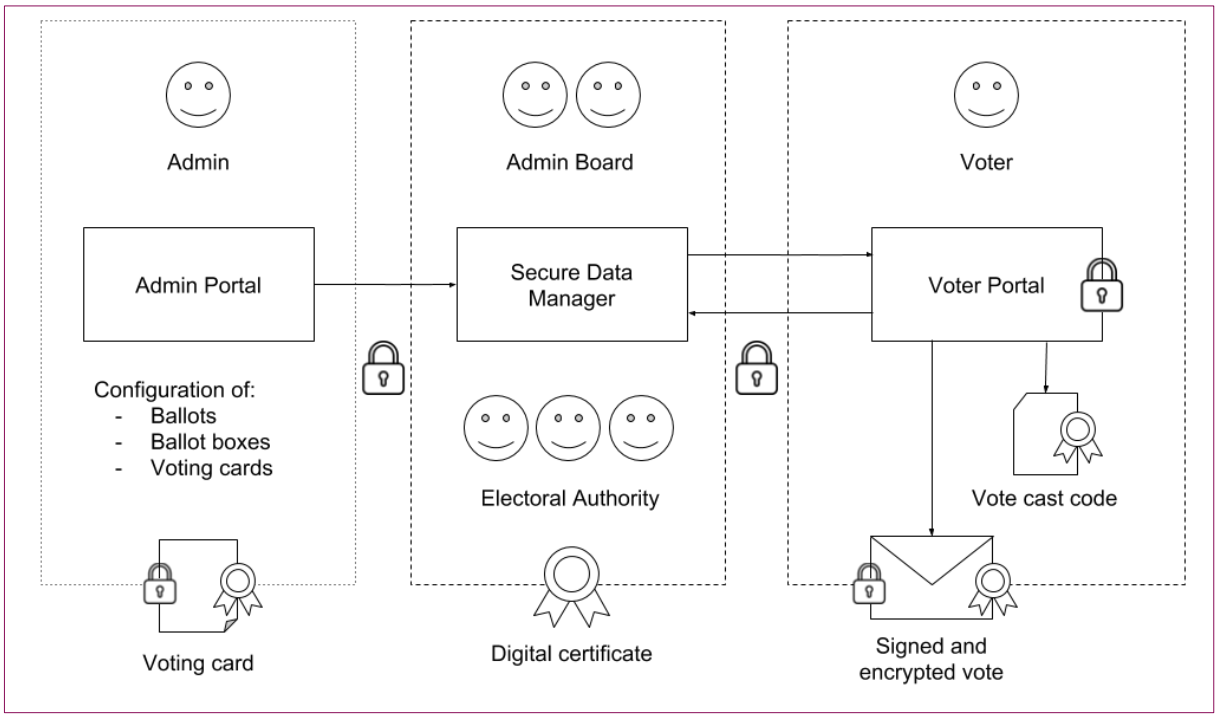

Figure 2 – Scytl sVote main actors

The administration role is split into three different actors with diverse duties within the process:

- A member of the Admin Board.
- The Admin, who is usually part of the Admin Board as well.
- A member of the Electoral Authority.

3 Solution strategy

3.1 Scalability

Scalability is a key feature of the solution, as it means a high level of adaptability in different elections sizes and types. For instance, some elections can include only a few thousand people, and using a referendum, while others can reach millions with a combination of referendums and open lists with several candidates and parties involved. Hence the solution must be fully adaptable.

[1] In some parts of the sVote code, this concept might be referred to as “Electoral Board” instead of “Electoral Authority”.

The scalability is achieved by using a microservice architecture, which enables the creation of available resources, according to the estimated needs of a specific submodule.

With this solution, the resources allocation can be done in an economically feasible manner, as opposed to a monolithic architecture, where needs are defined for the whole platform altogether, which can thus result in over or insufficient resources allocation. All components have different needs in terms of processing power, memory requirements or database size, and the fine-grained handling and sizing of these requirements can reduce the overall operating cost of the solution during a running election.

3.2 Accessibility and responsiveness

Considering that Scytl sVote is meant to be used by a wide range of users, special attention must be paid to the usability of the platform. People with different economic backgrounds, social status and with diverse devices will be participating in the voting process. Therefore, the user interface should be appropriately displayed on an extensive variety of computers, mobile phones and tablets. This is achieved by using the Material Design (https://material.io/design/) guidelines for the improvement of the Admin and Secure Data Manager portals, by creating a fully responsive Voter Portal and in general, by providing a unified voter experience across all supported platforms.

In addition, making the frontend readable by most of the screen-readers helps visually impaired citizens to easily participate in running elections. In this sense, Scytl sVote fully complies with the Web Content Accessibility Guidelines 2.0 and was certified level AA by an independent web accessibility certification authority (Stiftung Zugang für Alle) in Switzerland.

3.3 Data separation

The microservice architecture allows data separation. Hence the services have only access to the information they need. In this sense, the microservices can use either completely isolated databases and separate schemas, or the same database, being more an economic and security decision than a restriction imposed by the platform.

The main reasons for the implementation of data separation in the solution, are scalability and security related.

- From the scalability point of view, if each microservice can scale on its own, it is advisable that the used database has also the same possibility.
- The security concern was that in the case that a single microservice was compromised, the attacker would only have access to the data used by that microservice and would not be able to put the whole platform at risk. The compromised microservice can then be replaced seamlessly, without affecting the rest of the microservices, thus affecting the availability of the election portal.

3.4 Security and transparency

The 100% electorate authorization requirements from the Swiss Federal Chancellery (FCh) ordinance demands that the voting system should be completely verifiable. That means that, in addition to being

individually verifiable, the system must also be universally verifiable. These complete verifiability requirements, and more concretely those related to universal verifiability, are compiled in Article 5 of the Federal Chancellery Ordinance 161.116 1 and related Annex 2 4.3 and 4.4. The Individual verifiability requirements required for 50% electorate qualification are compiled in Article 4 of the ordinance and related Annexes 4.1 and 4.2. and should be also implemented.

In this document, we describe how Scytl achieves the universal verifiability requirements from the point of view of the architecture. This approach is the basis used to implement on the Swiss Post/Scytl voting solution to achieve 100% electorate authorization.

The starting point for this specification is the voting system implemented for the 50% authorization; a system that already implements the individual verifiability requirements described in the FCh regulation. The general voting protocol of Scytl sVote is not within the scope of this document as it is described in the Scytl sVote Protocol Specifications document.

3.4.1 End-to-end encryption

End-to-end encryption is a feature of the solution, covering the whole voting process from the casting phase to the final tally of results.

During the voting stage, the encryption takes place in the voter's browser (by using a public key, the private part of which should have previously been split in shares and distributed among different electoral authorities), thus ensuring that no sensitive unencrypted information reaches the voting platform in a way that could potentially determine the selected voting option(s).

3.4.2 Anonymous preserving decryption

To preserve the voter's privacy, the decryption process implements cryptographic mechanisms which prevent correlation between decrypted and encrypted information.

3.4.3 End-to-end verifiability

Voters should be able to verify that their vote has been recorded-as-cast and cast-as-intended; and both observers and independent auditors should be able to check that votes are counted as recorded without compromising voters' privacy.

- Recorded-as-cast verifiability: This verifiability level is achieved by means of vote confirmation receipts which are displayed to voters after their vote has been cast and can be looked for once the election is closed on a Receipts Portal or a Receipts List made available to voters.
- Cast-as-intended verifiability: This verifiability is achieved by means of Choice Return Codes[2], which are sent by mail to the voter before the election starts and univocally represent voter's valid options. The server can regenerate these Choice Return Codes and send them back to

[2] In some parts of the sVote code, this concept might be referred to as "Choice Code" instead of "Choice Return Code".

the voter while voting without knowing their real option representation. This way, the voter can check if these codes match with the ones contained in the paper voting card.

These two capabilities provide full individual verifiability to sVote.

- Counted-as-recorded verifiability: This verifiability level is achieved by using a Verifiable Mixing and Decryption, which guarantee voter's anonymity and disclosing the content of the vote. Both processes provide mathematical proofs to demonstrate that all the votes contained in the ballot box have been considered.

This last capability provides universal verifiability to the solution; which together with full individual verifiability, makes sVote an end-to-end verifiable solution.

3.4.4 Control Components

The Control Components requirements are introduced in the Article 5 of the Federal Chancellery Ordinance and further extended in annex sections 4.3 and 4.4. According to the regulation, Control Components can be implemented as:

- A group of people (4.4.10): People are considered only for protecting voter privacy during the vote decryption process. In this case, it is possible to set up a group of at least 4 people for keeping shares of the secret key in smartcards.
- Standard computers (4.4.16): At least 4 computers with different OS to ensure that they do not share a common threat.
- Hardware Security Modules - HSM (4.4.16): At least 2 HSM from different vendors with CC EAL4 or FIPS 140-2 level 3 certification.
- Control Components can be combined in one or more groups. All the Control Components in a group must collaborate to perform their assigned voting protocol function and any attempt to abuse the system should always be detected if at least one component of the group is honest.

This document describes the implementation of the Control Components based on standard computers (4.4.16).

3.4.4.1 Control Component concept

In the context of Scytl sVote, the Control Components are defined as:

- A Choice Return Codes Control Component, where Choice Return Code generation occurs in 4 execution environments, and Choice Return Code verification is split into 4 execution environments.

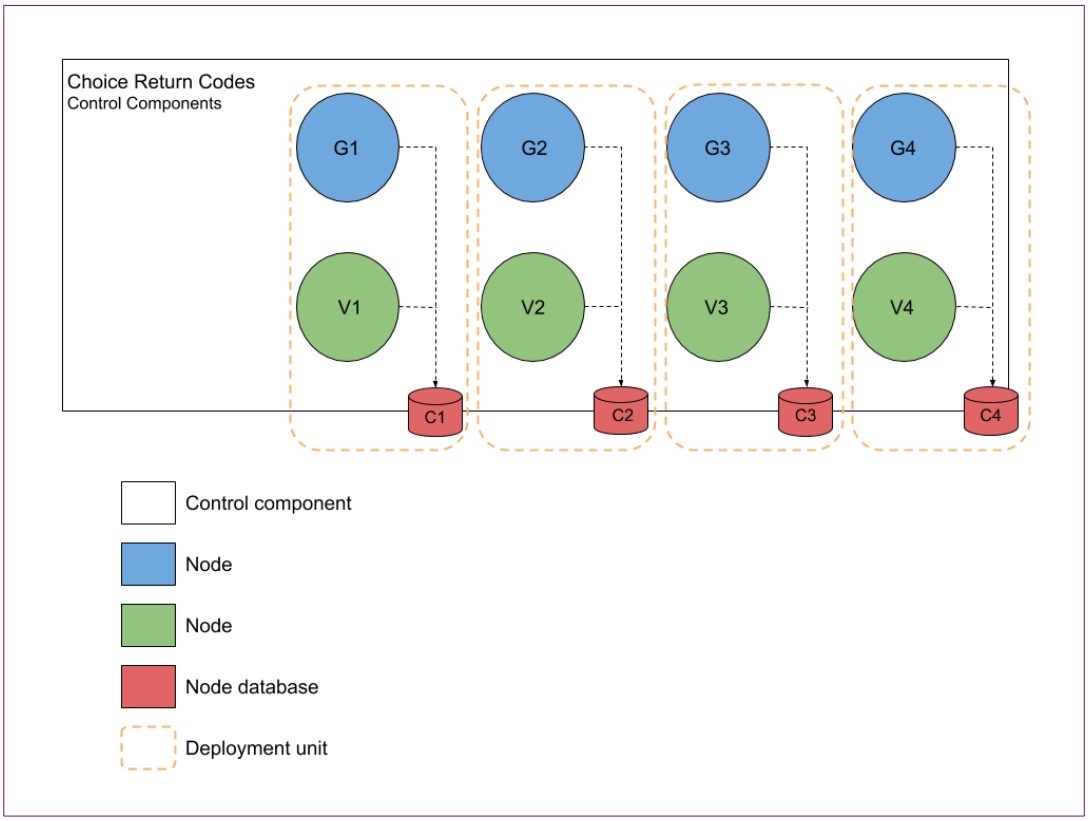

Figure 3 – Choice Return Codes Control Component

- A mixing and decryption Control Component, which splits the mixing and decryption of ballot boxes into 4 execution environments, one of it being the SDM.

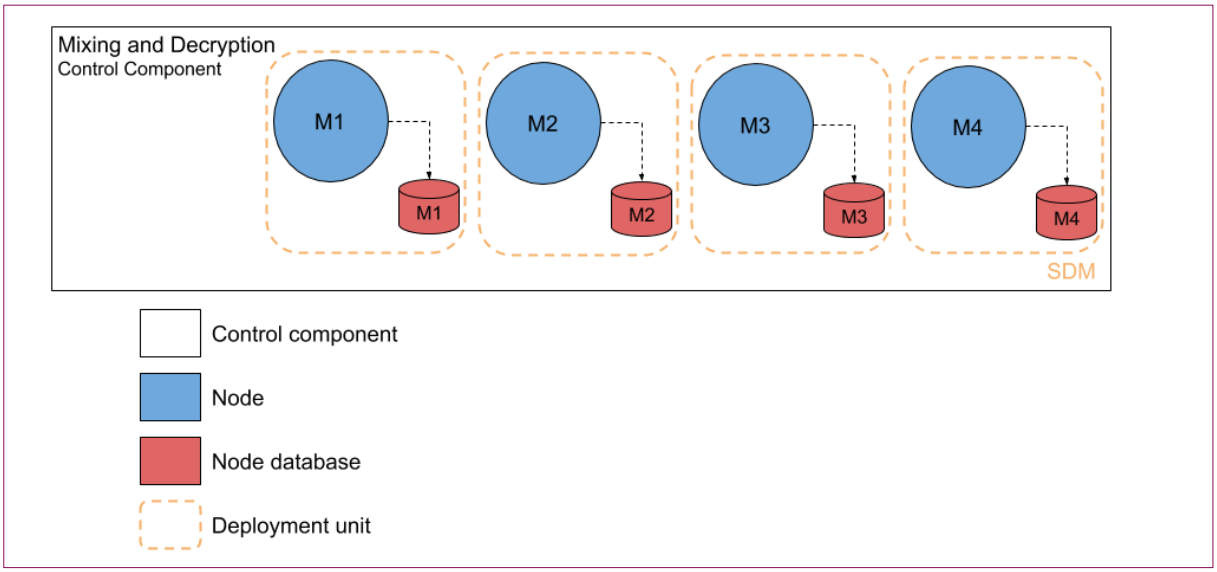

Figure 4 – Mixing and Decryption Control Components

- A Bulletin Board Control Component, which consists of all the microservices' aggregated logs.

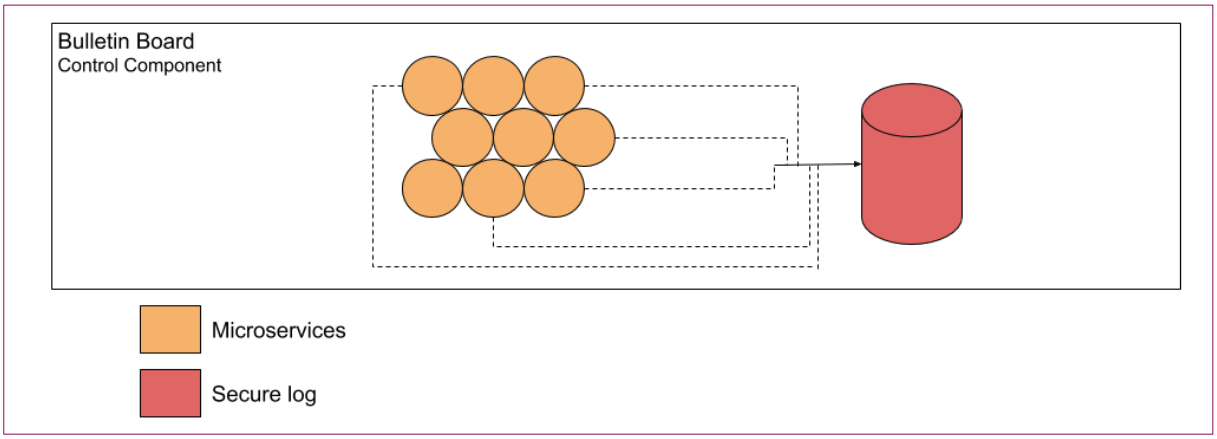

Figure 5 – Bulletin Board Control Component

- The Share Splitting Control Component, which splits the Electoral Authority private key into multiple shares.

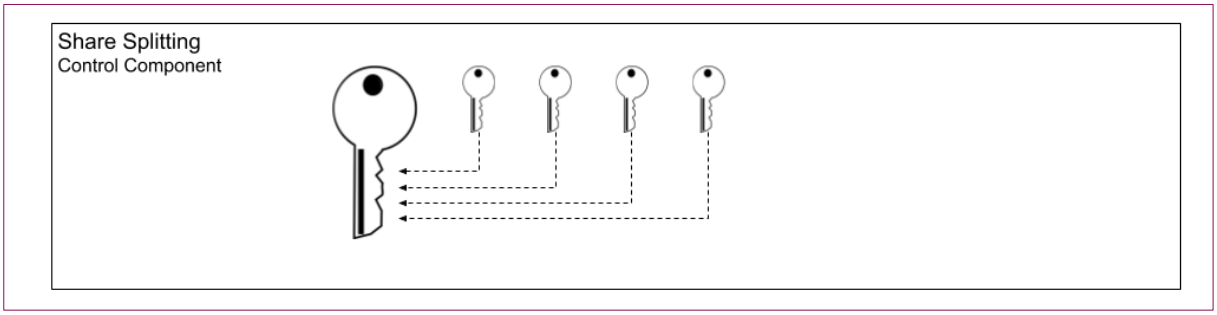

Figure 6 – Share Splitting Control Component

3.4.4.2 General remark

For the Choice Return Codes generation, the execution sequence of the data processing (the order the data is sent to the different Control Component nodes) is not required to be sequential, nor for the mixing. However, by choosing a sequential approach, the recoverability of the system can be provided with less implementation complexity.

For the verification of the Choice Return Codes, since it introduces at least one extra step in the voting process, it will have an impact on the user experience. Thus, the sequential execution is not the optimal solution in the verification phase. For verifying the Choice Return Codes, the Control Component nodes should be working in parallel, to mitigate the impact of the communication delays that arise from using Control Component nodes in different, physically separate locations.

3.5 Performance efficiency

The time to complete an electronic ballot should not take longer than filling a paper-based ballot. And although ideally, it should be even faster, the highly time-consuming of the cryptographic operations behind the verifiability and encryption activities, make the process slightly slower.

To reduce times in casting a vote, several optimizations can be implemented. These are focused on reducing the number of modular exponentiations to be computed once the voter is ready to submit its vote.

The modular exponentiations take much more time than any other operation executed by the voting client, and therefore is where all the efforts are oriented.

Three strategies can be followed:

- Pre-computation at the voting client: Some modular exponentiations can be computed while the voter is navigating through the application, and before the send button is clicked.
- Pre-computation at configuration: Some modular exponentiations are already being computed during the configuration. Their results can be stored in the password-sealed KeyStore sent to the voter at the voting phase, so that these can be recovered, and used to construct the vote to be sent.
- Use of short exponents: For some operations, exponents may be shorter than required by the mathematical group where the operations are done, without being a security risk.

3.6 Vote correctness

3.6.1 Vote correctness types

The vote correctness is checked at two levels:

- The first level is when the vote is in plaintext, and validations that need the full context of the election can be applied to it. For example, if the vote contains an invalid voting option, that will fail in the plaintext vote validation, as it can check against the available voting options in the election. This validation cannot be performed when the vote is encrypted, because that could break the voter privacy if such information would be deducible from the encrypted vote.
- The second level is the encrypted correctness check.The different types of validations that are executed in the different vote correctness checks are the following:

3.6.1.1 Plaintext vote correctness validations

- The minimum selected options, excluding blank selections, is equal to, or greater than the minimum established (except if the property to allow blank votes is set to true and the whole vote is blank).
- The maximum selected options, including blank selections, is equal to, or less than the maximum established.
- The number of times that a candidate has been selected, including candidates that belong to more than one list, is not greater than the contest's accumulation value.
- There is no option representation (different from real candidates) that has been selected more than once.
- The text used for write-ins does not include the special character #.
- The representation of the write-in is included (by the system, not by the voter) next to the written-in text, separated by #.
- If write-ins are not enabled, the vote does not contain write-ins.
- If write-ins are enabled, the vote contains write-ins (if the voter has not included any, dummy data will be included by the system). In addition, it verifies if the number of write-in fields (which are padded by dummy data if not all are used) are exactly matching the required number of write-ins in the vote.

3.6.1.2 Encrypted vote correctness validations

- The number of selected options, including blank selections, is equal to the maximum established.
- The Choice Return Codes corresponding to the option representations exist.
- If write-ins are not enabled, the vote does not contain write-ins.

- If write-ins are enabled, the vote contains write-ins (if the voter has not included any, dummy data is included by the system).

3.6.2 Vote correctness architecture

During the creation of the Election Event when the data is synchronized with the Secure Data Manager, the vote correctness rules are added to the downloaded information when the election data is exported from the Admin Portal. The Secure Data Manager distributes the vote correctness rules with the help of the `ballot.json` that is signed, and is made available throughout the whole electoral process, including the voting phase and the post-electoral steps as well.

The `ballot.json` contains two attributes in each contest, that are the `encryptedCorrectnessRule` and the `decryptedCorrectnessRule` fields that contain a JavaScript function for evaluating the compliance of the rules configured in the Admin Portal.

The `decryptedCorrectnessRule` contains the following validations:

- The minimum selected options, excluding blank selections, is equal to, or greater than, the minimum established (except if the property to allow blank votes is set to true and the whole vote is blank).
- The maximum selected options, including blank selections, is equal to, or less than the maximum established.
- The number of times that a candidate has been selected, including candidates that belong to more than one list, is not greater than the contest's accumulation value.
- There is no option representation (different from real candidates) that has been selected more than once.

The rest of the validations (which are about the write-ins) are not handled by the JavaScript function in the attribute `decryptedCorrectnessRule` but are handled by the decryption module.

The `encryptedCorrectnessRule` contains the following validations:

- The number of selected options, including blank selections, is equal to the maximum established.

The rest of the validations (which are about the write-ins and the existence of the return codes) are not handled by the JavaScript function in the attribute `encryptedCorrectnessRule` but are handled by the election information module.

3.6.3 Write-in support

Write-ins are validated in several stages.

The first validation is imposed by the voter portal that checks that the write-in fields contain only the characters that are specified in the ballot in the `writeInAlphabet` field. The alphabet contains a

separator character as the first character (#) which is omitted as a valid character for input but is used internally.

The second validation is imposed by the ov-api, that is the JavaScript client of sVote, which makes sure that exactly the required number of write-ins will be submitted during the voting process. It also makes use of the "#" character as it prefixes the write-in value with the representation of the voting option plus the separator character before encoding the field.

A similar validation is executed during the submission of the vote by the election information service.

The last stage is after decryption, where the rest of the validations are performed after decrypting the write-in field(s).

3.6.4 Other validations

Those validations that are closely tied to the protocol (proofs, signatures, tokens) are performed across the voting process. These validations will be listed if they are executed next to any of the previous vote correctness validations.

3.6.5 Setup

During the synchronization phase, the Admin Portal includes the `writeInAlphabet`, `decryptedCorrectnessRule` and the `encryptedCorrectnessRule` attributes in the downloaded configuration, so the Secure Data Manager can distribute this information, and include them in the respected ballots for each contest that is in the ballot. This information is then signed and uploaded to the ballot information table in the database that is handled by the election information service.

3.6.6 Validations during election phase

3.6.6.1 Voter Portal Frontend

- Decrypted vote correctness rule is executed.

3.6.6.2 Ov-api (ov-client)

- Rules for the write-ins are executed.

3.6.6.3 Election information

- Encrypted vote correctness rule is executed.
- The service checks whether the ballot box the vote belongs to is available (not blocked).
- Vote certificate is validated.
- Vote certificate chain is validated.
- Validation is performed on the election dates, so votes outside of the election validity are rejected.
- A check is performed whether the voter is permitted to store the vote and the storage of the vote will not exceed the maximum allowed votes per election.

- The Schnorr proof of the vote is validated.
- The vote signature is validated.
- The authentication token is validated.

3.6.6.4 Vote verification (called from Election Information)

- Validations are executed on the partial Choice Return Codes, to make sure that the choice can be retrieved.

3.6.6.5 Vote verification in the Control Components

- Exponentiation proof validation.
- Schnorr proof validation.
- Plaintext equality proof validation.

3.6.7 Validations in the post electoral process

3.6.7.1 Decryption

- Validation of the write-ins (vote contains correct number of write-ins, write-ins are of the valid format, write-ins are not associated to the same representation multiple times, write-ins values associated only to write-in representations).
- Validation of the vote against the decrypted vote correctness rule.

4 Building block view

4.1 Mapping of contexts and microservices

Voter actions modify data in contexts and each context maps to a particular microservice. In this aspect, the term “Context” is used to identify a set of responsibilities that have clear boundaries, and the data the contexts are operated on are isolated, so that it can be used to define the boundaries of the different microservices resulting in a one-to-one mapping between the microservice and the context.

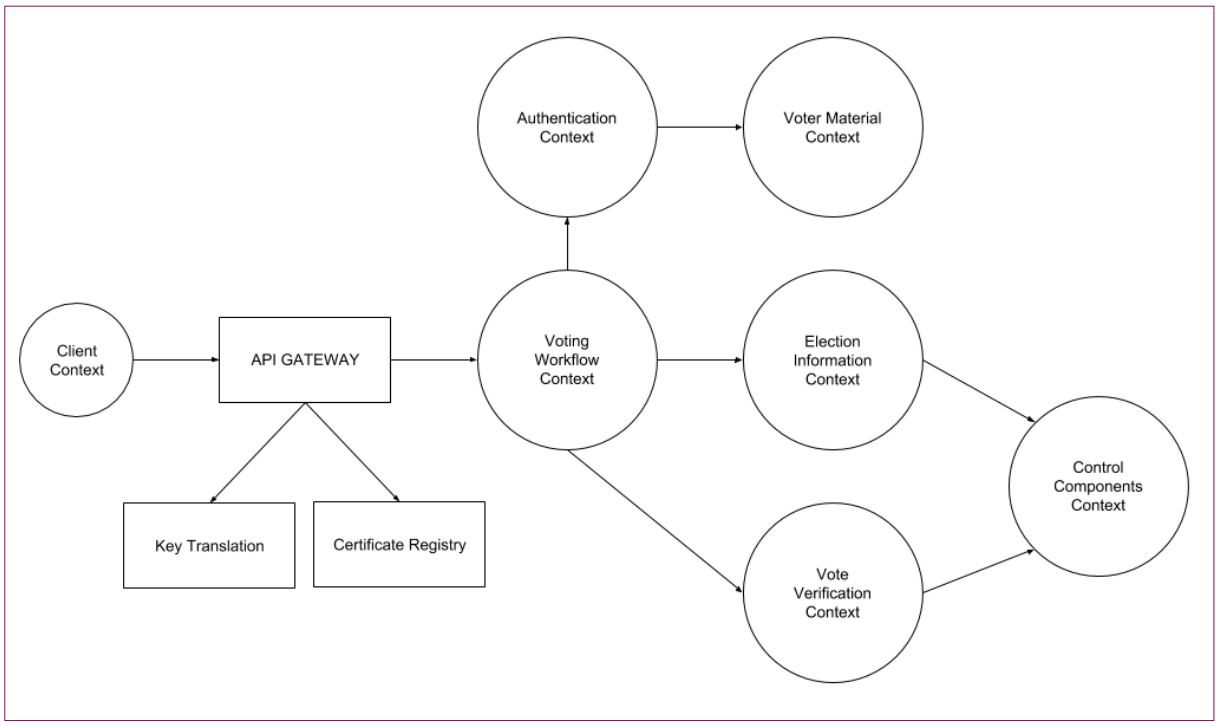

Figure 7 – Scytl sVote contexts and microservices

Context	Microservice	Responsibilities
Voting Workflow Context	Voting Workflow	Receives and manages client petitions. It contains the possible workflows per different Election Events. It contains the status of the voting cards.
Authentication Context	Authentication and Extended Authentication	Performs the server-side of the challenge-response with the voter. Generates Authentication Token to be sent to the client. Performs token validations every time the voter sends a vote.

Context	Microservice	Responsibilities
Voter Material Context	Voter Material Service	It stores the identifiers of different resources that are associated to the voter and the voter's cryptographic material.
Election Information Context	Election Information	Stores election information: • Ballots • Ballot boxes • Elections • Election Events Returns a Ballot given its Ballot ID. Performs vote validations. Given a valid vote and a ballot box ID, performs the following: • Generates and stores signed receipts. • Stores the vote (Recorded as cast). • Stores the confirmation proof of the vote. Performs validations of the number of allowed votes per Voting Card ID and per Authentication Token. Returns the complete ballot box when requested. Returns the anonymized ballot box content for votes confirmed for mixing (cleansed) when requested.
Vote Verification Context	Vote Verification	Verifies a vote is cast as intended.
Client Context	Does not map to microservice, but to the OV JavaScript library	Manages the interaction with the User through the User Interface. Communicates with the server-side through the Voting Workflow Context. Performs all the client cryptographic operations required by the protocol.

Context	Microservice	Responsibilities
Control Components Context	Does not map to a microservice, but to the Distributed Mixing, Return Codes Generation and Return Codes Verification applications	Participate of sensitive operations like the generation of the necessary data for the cast as intended verification, the verification of the vote being cast as intended and the ballot box mixing.

Table 1 – Scytl sVote contexts and microservices

4.1.1 The key translation

The key translation module is the component responsible for converting:

- From internal IDs (like ID of the election that is a UUID in the format of xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx) to aliases (like alias of the election, e.g. “test-election”).
- From aliases to internal IDs.

The following entities are supported by key-translation:

- Voting card
- Election Event
- Ballot box

4.1.2 Certificate registry

This service contains the different certificates (Platform Root, Tenant and Admin Board), checks their validity when uploaded, and returns them when requested.

The certificate hierarchy completely relies on the fact that Scytl sVote is multi-tenant, which means that the same infrastructure is shared among different tenants.

To protect the data to be tampered or to prevent a tenant to configure elections from another tenant, a proper certificate hierarchy has been designed. Thanks to it, the platform owner can issue certificates for each specific tenant.

Note: In a non-multi-tenant installation (an infrastructure that it is only used by one tenant), there are no functional differences between the Platform Root CA and the Tenant Authorities CA, since both will be managed by the Platform owner.

The following diagram outlines the voting platform certificate hierarchy.

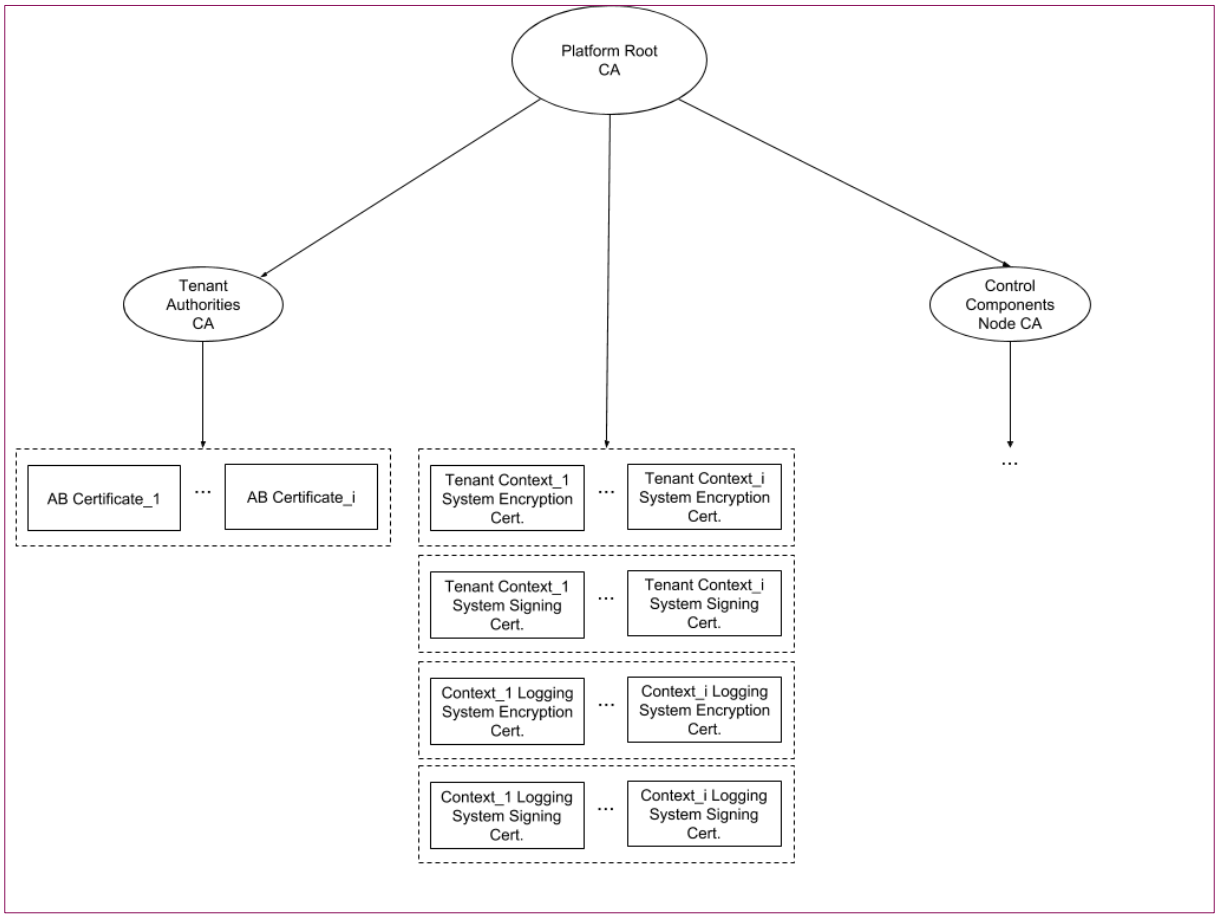

Figure 8 – Scytl sVote certificate hierarchy

Certificate registry items	Description
Platform Root CA	Top CA in the hierarchy. It issues the certificates for its three branches. Created only once by the platform owner. Needs to be installed in the platform by using the provided solution system tools.
Control Components Node CA	Intermediate CA that represents a Control Component deployment unit. There is one per execution environment and Control Component type. Issues the certificates used by Control Component nodes. It must be installed in the execution environment before the component nodes are started.
Tenant Authority CA	Intermediate CA that represents a tenant in the infrastructure, it is responsible for issuing Admin Board certificates in the SDM. There is one per registered tenant in the infrastructure. They need to be requested by the tenant (by means of a certificate signing request, CSR), signed by the canton, and installed in the platform by using the provided solution system tools.
Admin Board Certificate	Digital Certificate for which its private key is split among different members constituting the Admin Board responsible for signing and therefore approving an election configuration from the SDM before being uploaded to the platform. There could be as many as desired for every tenant.
Context_X Logging Encryption/Signing Certificate	Digital certificates used by the secure logging, there is one per context (system service component) and purpose (cipher/signing) in the infrastructure. They need to be created by the platform owner and installed in the platform by using the provided solution system tools.

Certificate registry items	Description
Tenant Context_X System Encryption/Signing Certificate	These digital certificates are created at the same time as the Tenant Authority CA, so they follow the same generation workflow. They need to be installed in the platform by using the provided solution system tools. The public encryption certificates need to be installed together with the SDM and are referred to as “system keys”. They are intended to cipher from SDM so the OVP contexts (system service components) can decipher all the passwords of the private KeyStores created from every Election Event certificate hierarchy. There is one per tenant and context.

Table 2 – Scytl sVote certificate hierarchy

4.1.3 API Gateway

The API Gateway is the main entry point for all communication and integration of the services. It is a single point of contact for any API call, and provides critical features such as service discovery, routing and aggregation, rate limiting, monitoring and alerts, authentication and authorization, exception handling and fail-over mechanisms.

The following communication goes through the API Gateway:

- The communications with the Voter Portal backend.
- The communications with the Control Components.

4.2 Level 1

Global view of the system, with the interactions between the major applications.

Note: The arrows represent the requests (origin and target) that occur during the electoral process (not the dataflow).

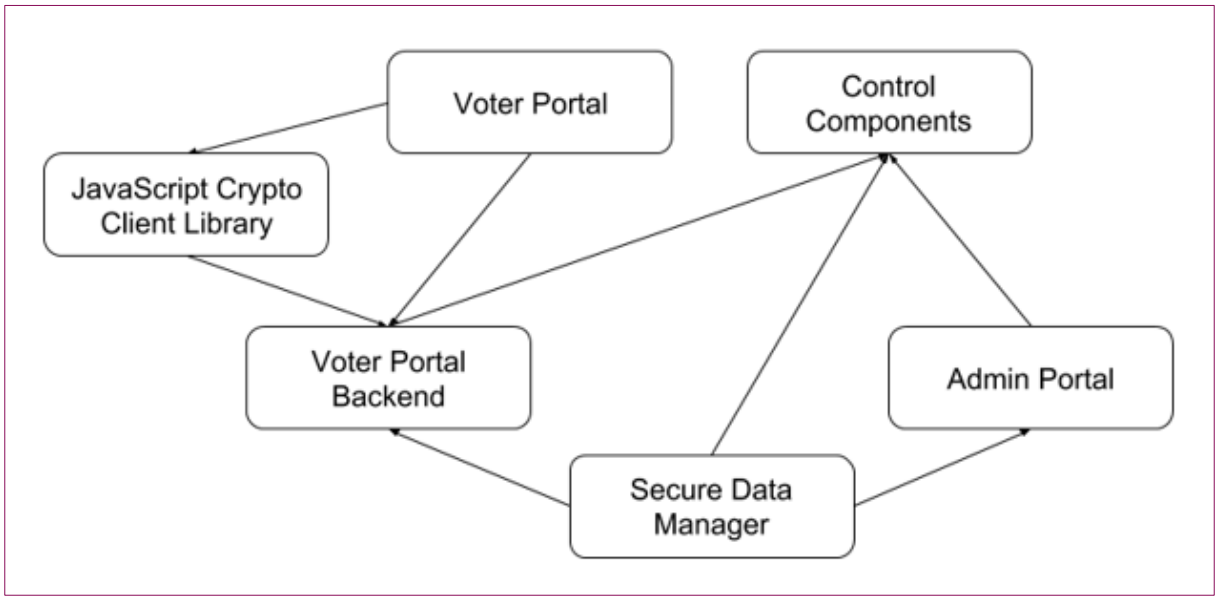

Figure 9 – Scytl sVote applications

- **Admin Portal (AP)**: Application used to define the Admin Board and all the elements of an Election Event (Ballots, Voting card sets, Electoral Authorities and ballot boxes). Requests needed information to the Control Components when it is required to complete the Election Event elements definition. These elements are then synchronized with the Secure Data Manager to make them secure.
- **Secure Data Manager (SDM)**: Local application that can be used on one or many computers, depending on the customer operational preferences and security requirements. The default configuration is to use the SDM on one online computer and two offline computers (for pre and post electoral operations). The SDM is used to synchronize content to and from both the Admin and Voter Portal, and to complete sensitive processes (voting card generation, mixing and decryption) in an isolated, offline, highly secure environment. It is first synchronized with the Admin Portal where all the entities for the Election Event have been previously defined. These elements are then included in the SDM to make them secure. In the case of the sensitive processes, the SDM also requires the participation of the Control Components before they can be completed. Once all the materials are secured, they are synchronized with the Voter Portal for the Election Event.
- **Voter Portal Backend**: Application that voters use to select their options and submit their votes during the Election Event. From the Voter Portal, all results are collected, and the confirmed votes are sent to the Control Components to begin the mixing process, so that they can later be synchronized back to the SDM to complete the mixing and decryption.
- **Voter Portal Frontend**: The frontend application that the voter interacts with. The Voter Portal frontend translates the selections of the Voter to an encrypted, secure vote that is transmitted to the Voter Portal Backend.

- **JavaScript Crypto Client Library:** Provides Voter Portal frontends with high-level APIs that they can use to complete the voting process.
- **Control Components**: Multiple instances of applications in different execution environments that participate in the sensitive processes of the voting protocol.

The Admin Portal and Voter Portal are both server-based applications, with a frontend and a backend, while the Control Component nodes and Secure Data Manager are standalone applications.

4.3 Level 2

Breakdown of Level 1 view, with the composition of the main components:

4.3.1 Voter Portal Backend

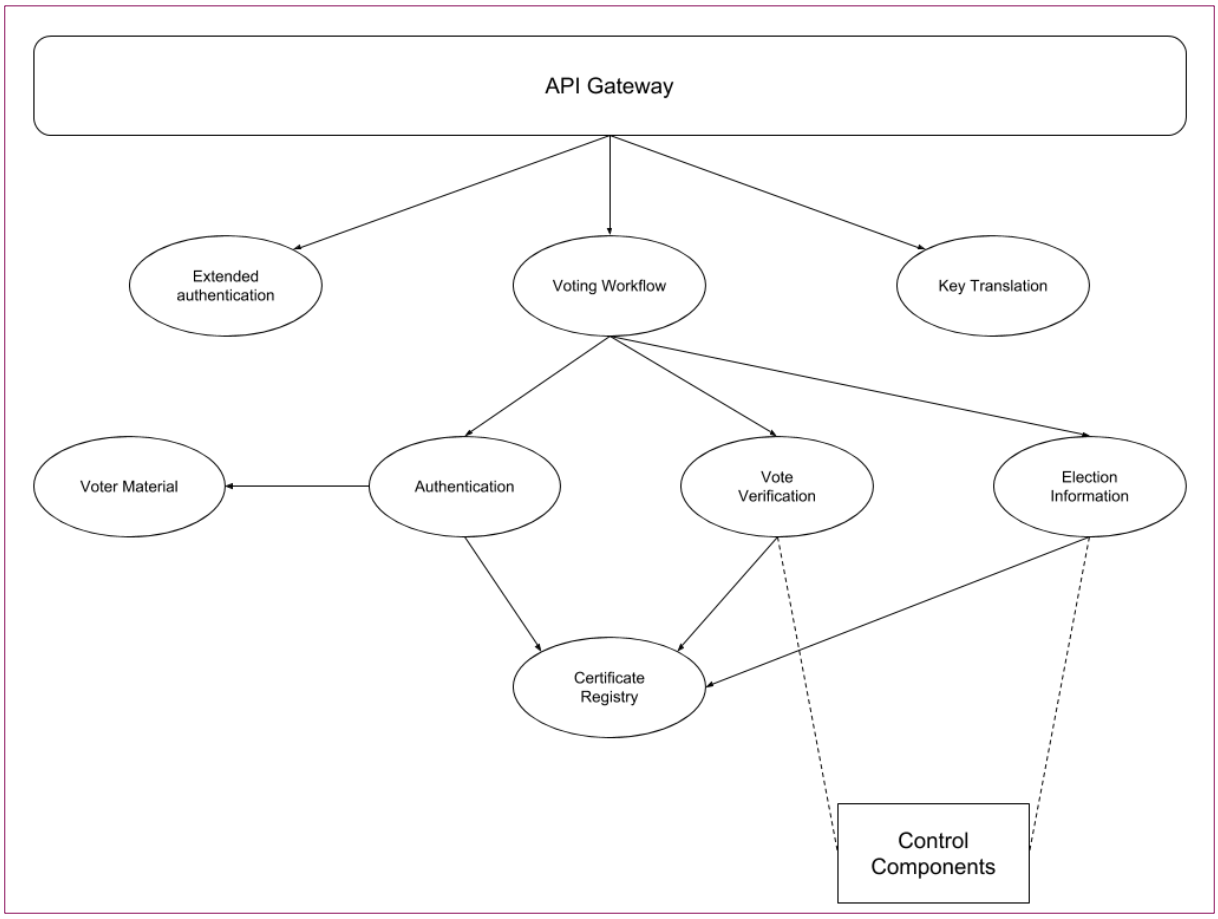

Figure 10 – The Voter portal microservices

The diagram above illustrates the microservices that form the Voter Portal backend. Infrastructure components (load balancers, reverse proxies) are not shown. The arrows represent the communication flows, so for example, the Vote Verification service asks the Certificate Registry during the voting process to provide the certificates that are used for validation purposes in the voting process.

Detailed explanation of the components:

- **API gateway:** This component serves as a proxy and as a filter. All requests reach the rest of the components through this microservice. It does not contain any business logic.
- **Extended authentication:** This service is used to authenticate a voter based on additional information, for example year of birth; it works in conjunction with the authentication service; The Extended authentication is optional, and the type of additional information used is configurable.
- **Voting workflow:** Responsible for receiving and managing requests from the client; it controls the workflow for the Election Event as well as the status of the voter.
- **Key translation:** This service is used to map external identifiers to internal identifiers, e.g. Election Event alias to an Election Event ID, or voting card alias to a voting card ID.
- **Voter material:** Provides information about the voter e.g. credentials, eligible Election Events, reference to verification codes.
- **Authentication:** This service is used to verify the identity of the voter; it performs the server-side of the challenge-response with the voter, generates the authentication token sent to the client (Voter Portal frontend), and performs token validations for every voter action.
- **Vote verification:** Verifies that a vote is cast as intended by computing, with the help of the Control Components, the Choice Return Codes corresponding to the encrypted vote; it also performs the second level of vote correctness by checking the content of the vote against the rules applicable to the voter.
- **Election information:** This service provides the ballot information i.e. the options for the different contests of an Election Event the voter is eligible to participate to; it performs vote validations, generates and stores signed receipts, stores the vote and the confirmation proof of the vote; it also performs the first level of vote correctness by checking the structure of the vote against the electoral system. It provides the confirmed votes (anonymized) to the Control Components so they can begin with their participation on the mixing process.
- **Certificate registry:** It contains the different certificates (platform root, tenant, admin board), checks their validity when uploaded, and returns them when requested.

4.3.2 JavaScript Crypto Client Library

The JavaScript Crypto Client Library provides high-level APIs that permit Voter Portal frontends to perform the full voting process. This library liberates Voter Portal frontends from having to be concerned about many low-level details are related to the voting process. For example, some of the high-level steps in the voting process (such as authentication and casting the vote) actually involve multiple intermediate interactions between the client and the backend, with cryptographic operations being performed during each of these intermediate steps. All of these details are hidden within the JavaScript Crypto Client Library. The cryptographic operations performed within this library include the opening of KeyStores, encrypting the voter's choices, signing data and generating Zero Knowledge Proofs.

The JavaScript Crypto Client Library makes use of the Web Workers API for the execution of tasks. The use of workers, together with the use of "promises" (essentially objects that act as proxies for the result of operations that might not have finished yet), allows for the creation of asynchronous client code, thus enhancing the user experience and improving performance.

The JavaScript Crypto Client Library includes the following components:

- **CryptoLibJS**: This is a JavaScript cryptography library that provides a range of cryptographic functionality. The JavaScript Crypto Client Library includes CryptoLibJS as a dependency. The functionality that is provided in CryptoLibJS is designed to be compatible with the functionality provided by the Java-based version of the library, which is called CryptoLib. This ensures that clients (who are using CryptoLibJS) can interact with a backend (using which is CryptoLib).
- There are three APIs defined in the JavaScript client library. They are the following:
    - **ovWorker**: This API allows clients to perform calculations in the background during the session, for example when the user is interacting with the Voter Portal. Performing such calculations during moments when no other computationally intensive operations are being executed in the client reduces the overall time for a client to complete the voting process. For example, in the case of ElGamal encryption, it is possible to perform part of the encryption process before actually possessing the plaintext to be encrypted. This is because part of the encryption process only needs the encryption key and a source of randomness. Such calculations are known as precomputations. Later, once the actual plaintext is available, the encryption process can be completed and it will require less computations than would have been required if precomputations had not been done.
    - **ovAPI**: This is the main API provided by the JavaScript client library. It provides a simple and straightforward API for a client to complete the voting process. Through the use of just three main methods (for authentication, sending the vote and finally casting the vote), it transparently takes care of all the details and intermediate requests. Most Voter Portal frontends would be expected to use ovAPI.
    - **ovMsgAPI**: It is possible that a Voter Portal frontend might be developed that contains its own cryptographic engine. In such a case, the Voter Portal frontend might not need to call the high-level API provided by ovAPI. Such a Voter Portal frontend might prefer to perform some operations, such as encryption and proof generation itself. However, such a Voter Portal frontend might still need access to an API for sending requests to the backend. The API provided in ovMsgAPI allows frontends to send messages to the backend and obtain the response.

      Note: If a Voter Portal frontend uses ovAPI (which is the recommended approach), then it does not need to use ovMsgAPI because the sending of messages is handled internally within the implementation of ovAPI.

4.3.3 Secure Data Manager

The main objective of the Secure Data Manager is to provide the necessary cryptographic functionalities for configuring an election in a secure way. It can be connected to the Admin Portal, the Voter Portal and the Control Components.

Data can be transferred through USB drives between different instances of the same SDM application, so migrating the configuration or the post electoral data to a secure location can be achieved by this functionality. These are the components:

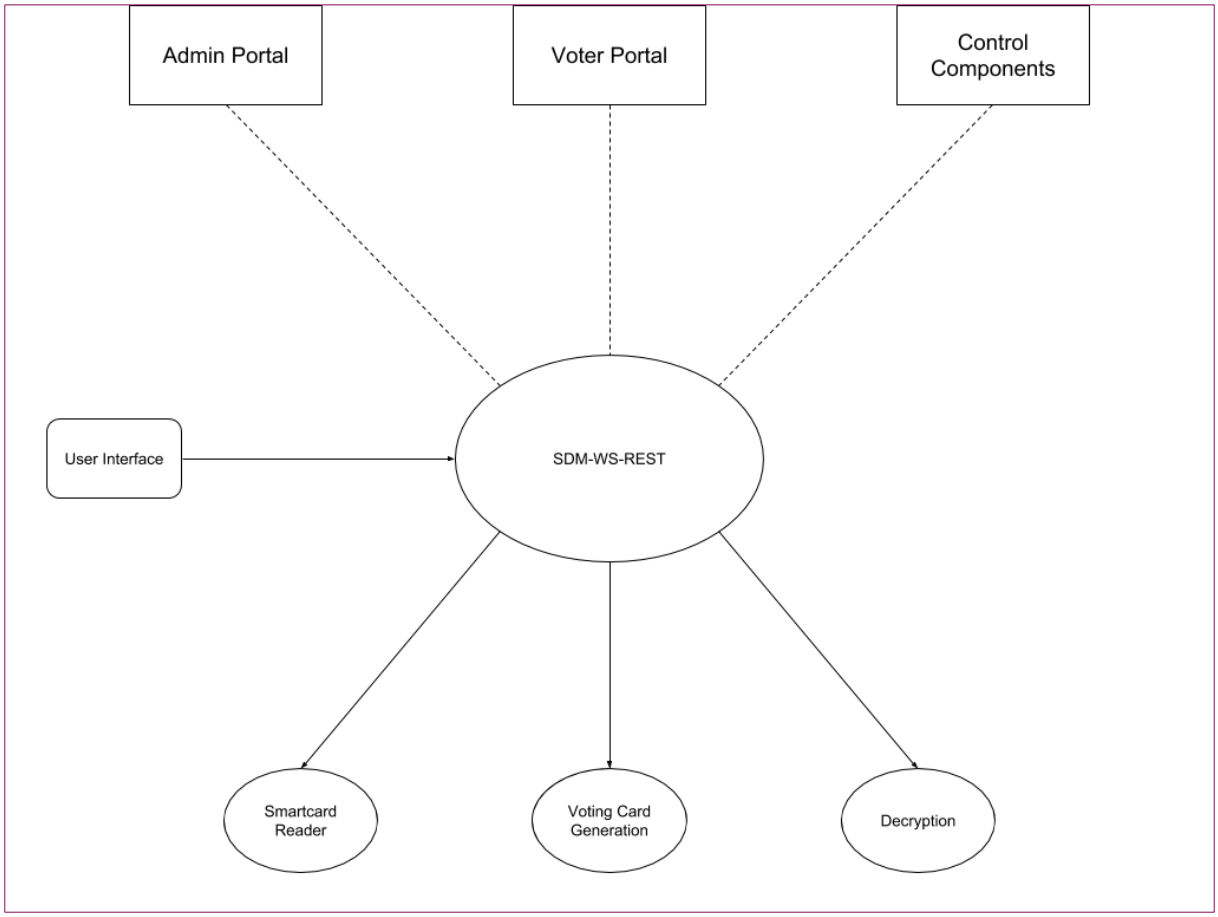

Figure 11 – The SDM microservices

- **Voting card generation:** It generates for each voter the information that will allow them to cast a vote online, i.e. voter credentials (start voting key) and verification data (Choice Return Codes, ballot casting key). It requires the participation of the Control Components to complete the process.
- **Decryption:** Service that decrypts the votes in a ballot box, returning the options for each vote. To achieve that, it shuffles and re-encrypts votes in a ballot box, decrypts the votes and decomposes the decrypted vote in valid option representations. The mixing and decryption operations correspond to the final execution for the mixing Control Components and generate Zero Knowledge Proofs accordingly.

- **Smartcard reader:** Library for managing the communication for smartcards, for writing and reading shares for the Admin Board and the Electoral Authority constitution.
- **Sdm-ws-rest:** This is the 'backend' of the SDM application. This module is a standalone component, that communicates with the rest of the services (voting card generation, decryption, smartcard reader, UI, Control Components and the admin and voter portals). Apart from it, this module is responsible for controlling the election workflow.
- **User Interface:** Web application that is rendered with the help of an embedded browser, and an embedded webserver. This module communicates with the sdm-ws-rest only.

4.3.4 Control Components

The main objective of the Control Components is to participate of the sensitive operations of the voting protocol in a way that any attempt to abuse the system is always detected. There are two Control Components implemented from the architecture point of view: The Choice Return Codes Control Component and the Mixing Control Component.

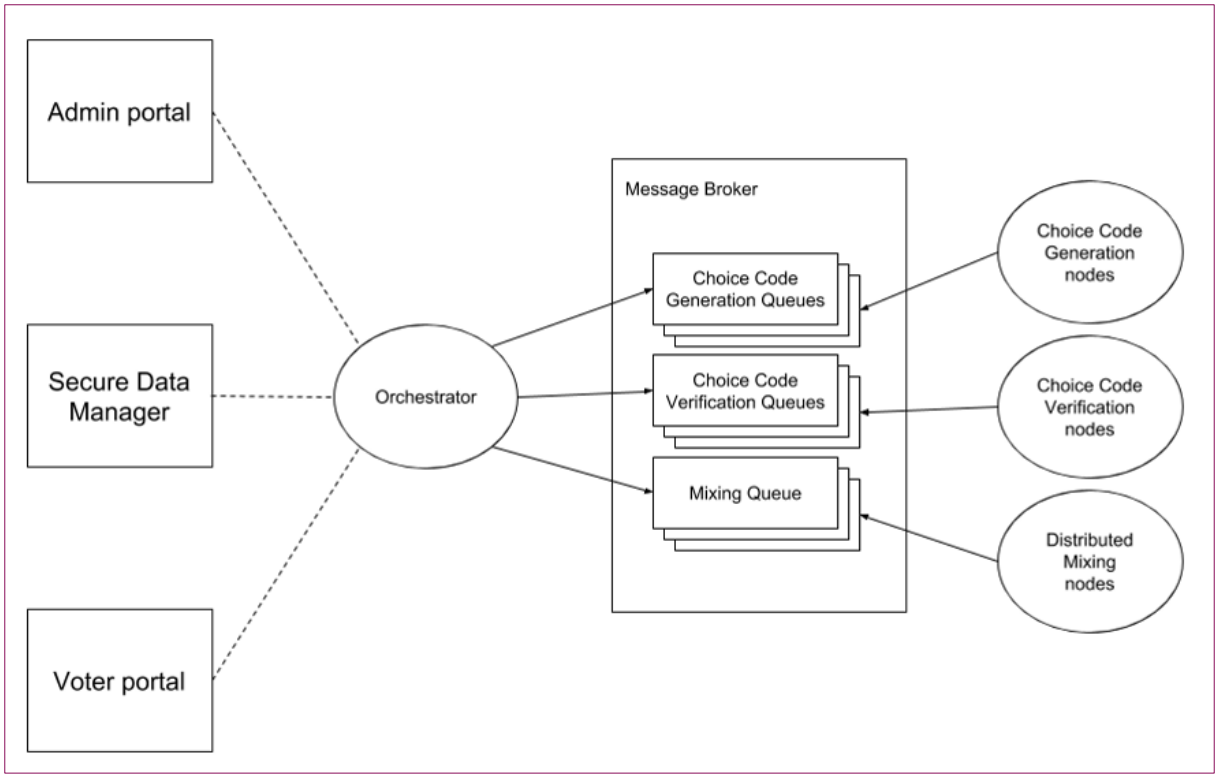

Figure 12 – Control Components

- **Orchestrator:** Responsible for receiving the requests from the other applications, it takes care of publishing them in the message broker queues and awaits listening to manage the responses (reply to the other applications, persist or forward to another queue).
- **Choice Return Codes Generation nodes:** Part of the Choice Return Codes Control Component. Each of these nodes is responsible for participating in the generation of cryptographic material (during election configuration) and the initial calculation of the Choice

Return Codes (during voting card generation) so during the voting phase, the participation of their associated verification node is mandatory to re-calculate the vote Choice Return Codes for the voter. The contributions to the final result performed by each of these nodes can be consolidated with the other nodes contributions, so these nodes can participate in parallel.

- **Choice Return Codes Verification nodes:** Part of the Choice Return Codes Control Component. Each of these nodes is responsible for participating in the recalculation of the vote Choice Return Codes (during voting the voting phase) using the cryptographic material generated by their associated generation node. The contributions to the final result performed by each of these nodes can be consolidated with the other nodes contributions, so these nodes can participate in parallel.
- **Distributed Mixing nodes:** Part of the Mixing Control Component. Each of these nodes is responsible for participating in the generation of cryptographic material (during the configuration phase) and in the mixing of the ballot box (in the form of multiple partial mixings and decryptions of the votes in the ballot box during the post electoral phase). The contributions to the final result performed by each of these nodes requires of a previous node contribution, so these nodes cannot participate in parallel.
- **Message Broker:** Responsible for decoupling the Control Component from the web-based applications architecture. Control Component nodes subscribe to this message broker to retrieve new contribution requests and publish their resulting contributions.

5 Runtime view

The election process can be divided into 7 steps with different components involved as shown below:

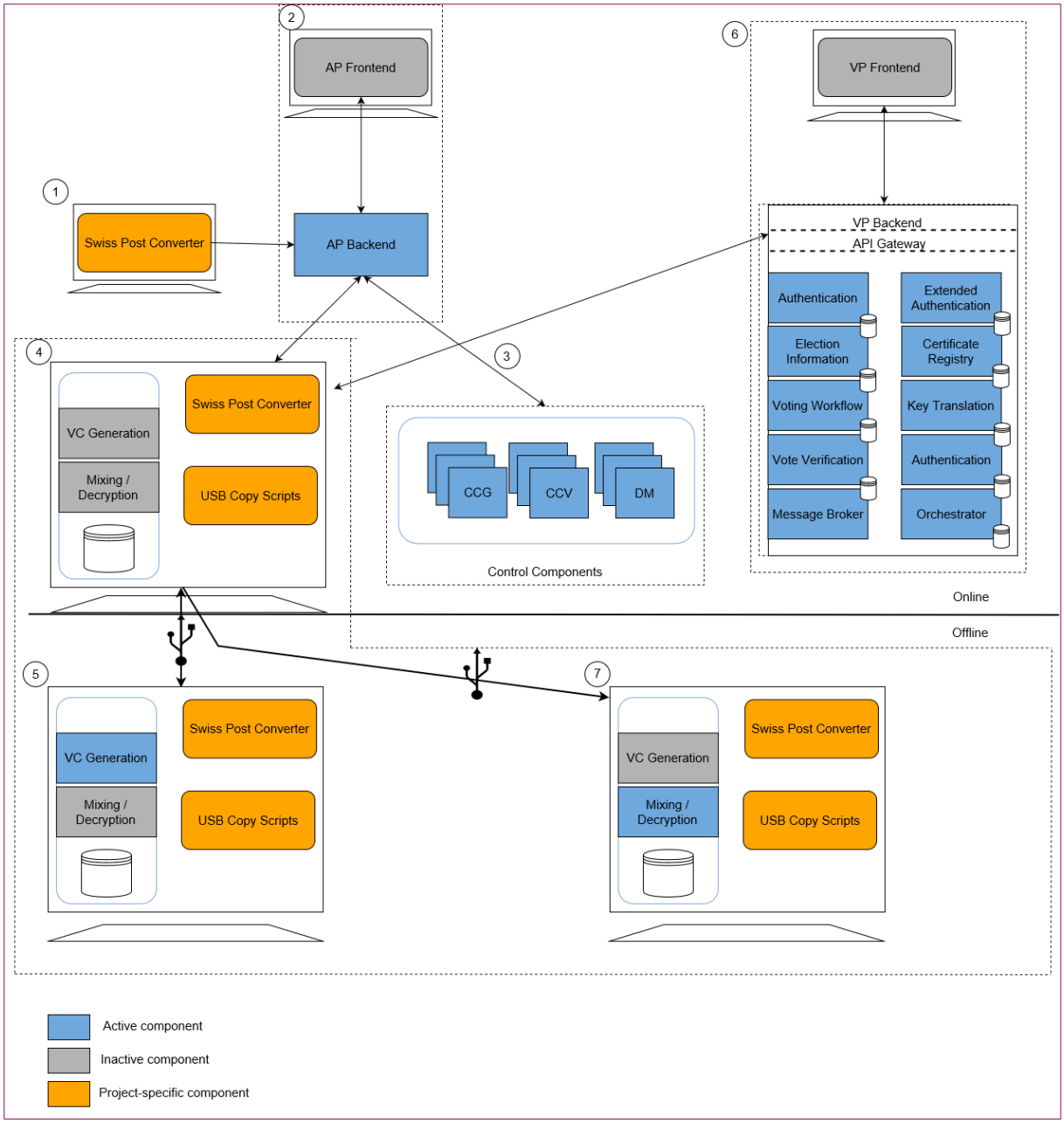

Figure 13 – Components involved in the election process

The workflow is as follows:

1) The Data is converted to a format that is supported by the Admin Portal (a JSON file containing the relevant parts for configuring an election).

2) The Election Administrator confirms that the imported election configuration is correct, and, if necessary, makes the appropriate adjustments, and locks the election.

3) The locking of the election triggers the communication to the Control Components to generate the required certificates and ElGamal key-pairs so that they can be used throughout the lifecycle of the specific election.

4) The election is synchronized to an SDM machine that has a connection to the Admin Portal, and then that data is physically transferred to an offline Secure Data Manager that is situated in a secure location.

5) The election data is generated and the Admin Board reviews and signs the election data, and the Electoral Authority seals the ballot boxes. During this phase, the election data must be transferred to the online SDM as for some calculations, the inclusion of the Control Components is required. After the Choice Return Code Generation process has finished, the operator transfers the data generated in the Control Components to the offline SDM and finishes the election configuration.

6) The final data is transferred to the online SDM where it is synchronized to the Voter Portal, and the Voting phase can begin in the Voter Portal. This phase terminates when the election is closed.

7) The Election Administrators execute the online mixing and partial decryption process that includes the utilization of the Control Components, then download the ballot boxes from the Voter Portal to the online SDM, and move that data to one or more offline SDM machines, in which the last step of the mixing and the decryption processes will take place.

5.1 Election configuration process

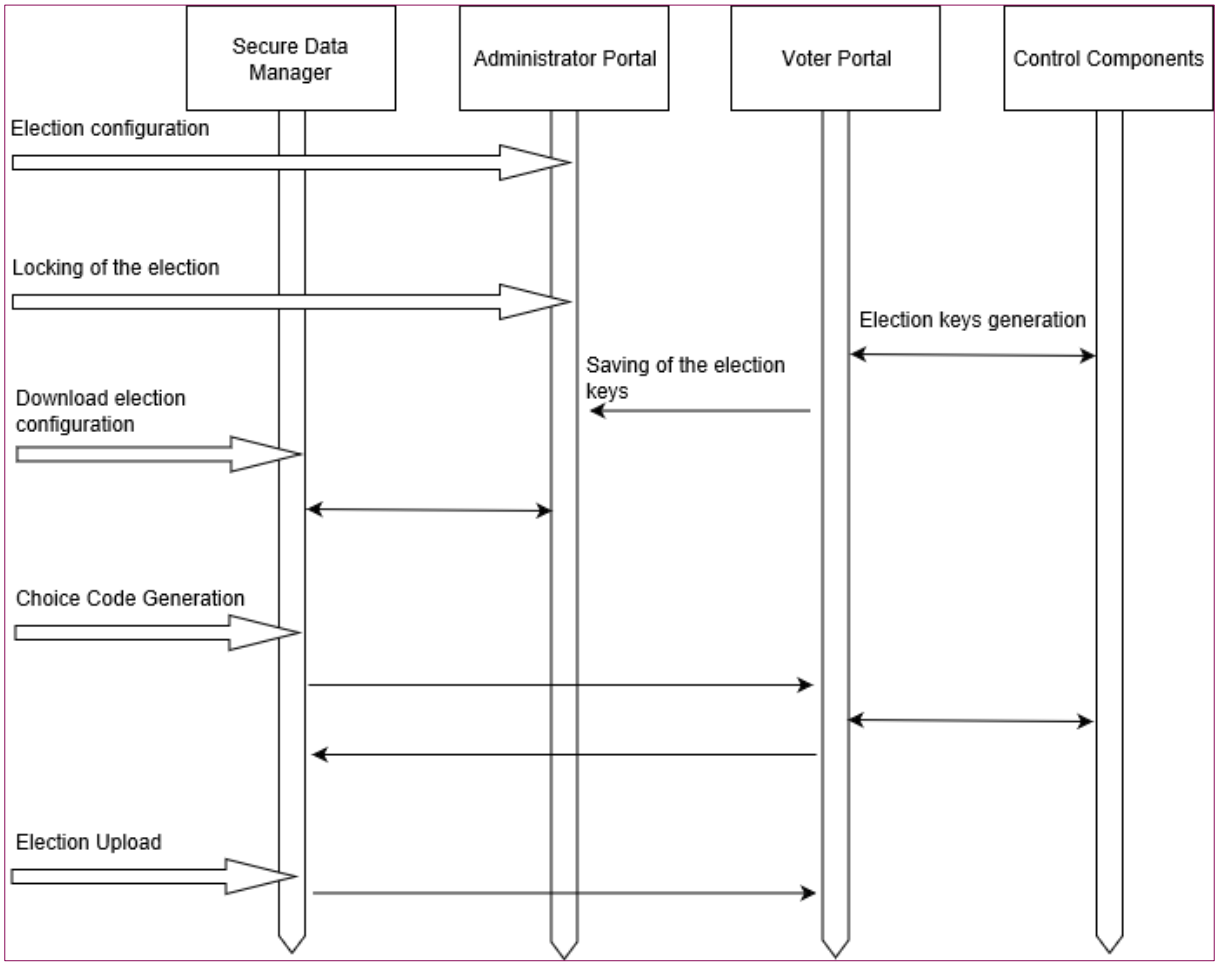

Figure 14 – Election configuration process

During the election configuration phase, four systems are primarily in use:

- The Admin Portal (AP) is used to create (or import) the election and verify that the contests and election information are correct. The AP serves as a CRUD application for managing the entities (attributes) for the different components of the election, like ballot boxes, Voting Card sets, ballots, etc.).
- The Secure Data Manager (SDM) is responsible for making the election secure in the electoral process.
- The Voter Portal (VP) serves as a proxy between the AP and the SDM to communicate with the Control Components.
- The Control Components (CCs), whose main responsibility is to provide the universal verifiability of Scytl sVote.

When the election has been correctly configured in the AP, all its attributes must be locked, so that the SDM can download the election and start the securitization process. During the locking of the election,

the AP issues a request through the VP to the CCs to generate the necessary public and private keys for the election. The public signed parts of the generated keys are returned through the VP to the AP The download of the election configuration happens in the form of the SDM authenticating to the AP and downloading the election configuration. This synchronization can happen multiple times, and only the reviewed and locked components will be synchronized to the SDM.

During a secure electoral process, the data after this step is transferred to a completely isolated machine where the cryptographically sensitive operations can take place.

In the middle of the secure operation, during the Choice Return Code computation phase, the generated partial data must be transferred to the online SDM from which the SDM has to initiate the request through the VP to the CCs so that the CCs can return the generated Choice Return Codes to the VP for temporal storage and finally to the SDM. After the online actions of the Choice Return Code Generation has been completed, the data is transferred to the offline SDM again, where the final election data is signed. At the end of the process, the data is uploaded from the SDM to the VP.

5.1.1 Voting card generation with the Control Components

On the secured SDM, the election operator creates the keys for the Admin Board (if they do not exist yet), generates the voting cards and signs the election content., like ballots.

During the Voting Card generation process, all components of the election are secured. The Admin Board key is used for signing all sensitive data that is created during this operation. The generation of the voting cards is done through the utilization of the config-generator crypto service, that is communicating with the SDM through a REST API. However, they should not be considered separate applications as the functionality of this crypto service is tied to the workflow of the SDM.

The Voting Card generation must be in 3 steps:

1) Pre-generate all the data that is necessary for Choice Return Code generation.
2) Submit necessary data to the Control Component so that they can generate the Choice Return Codes in a secure way.
3) Continue the voting card generation with the obtained secure Choice Return Codes.

Parts of the generated data can be used for external components to generate printed instructions with printed return code(s) to be sent to the voters for securing the voting process in later stages.

The following diagram provides a more detailed overview of the data flow between the online SDM and the Control Components that explains the workflow of the Choice Return Code Generation process, that includes the SDM, Control Component Orchestartor (CCO), the Message Broker with its Message Queues, and the Choice Return Code Generation Control Component instances.

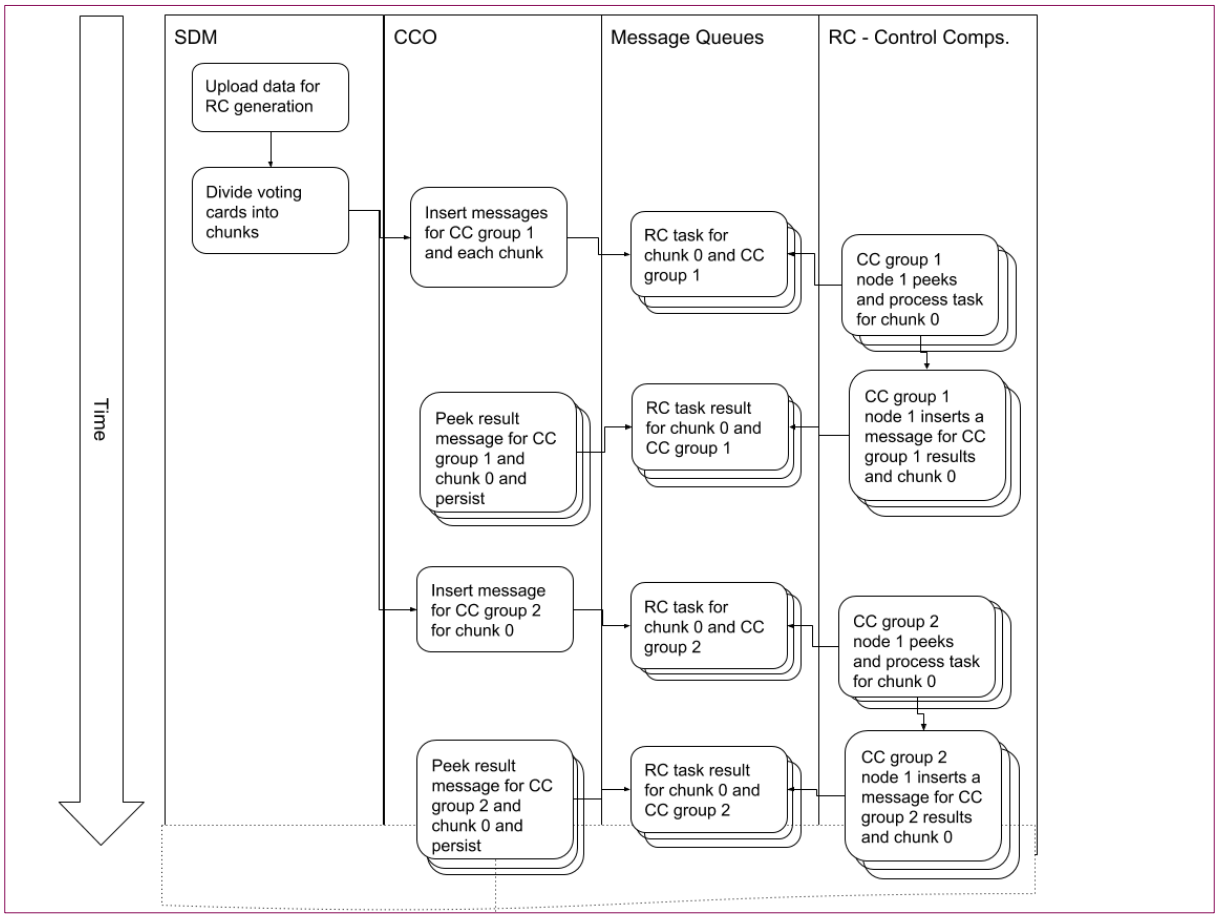

Figure 15 – Authentication, voting and casting phases (1)

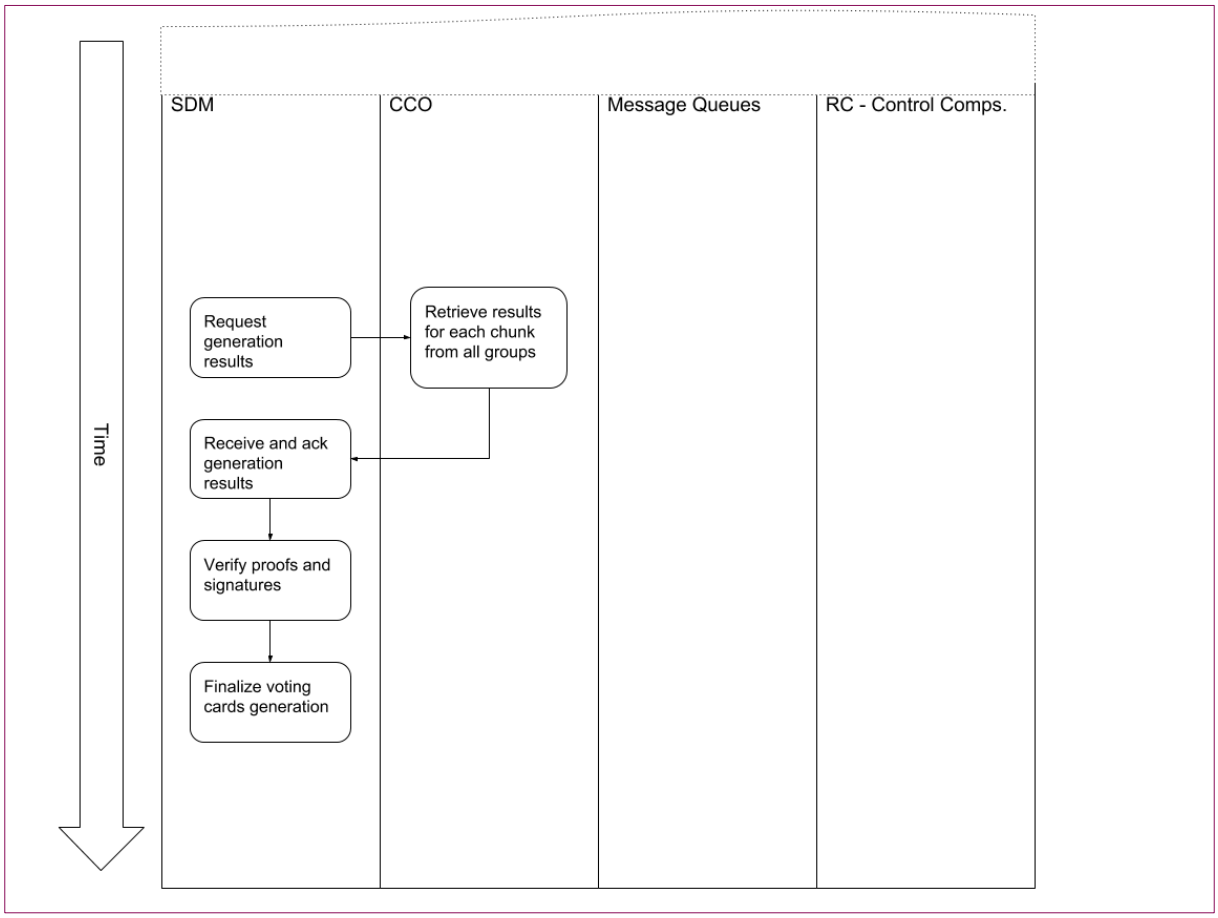

Figure 16 – Authentication, voting and casting phases (2)

5.2 Voting phase

The voting phase can be divided in 6 sections:

1) Voter requests authentication challenge.
2) Voter sends the reply to the challenge and receives the ballot.
3) Voter sends a vote.
4) Voter casts a vote.
5) Voter requests the receipt for the vote.

In the following diagram, the overall process is presented together with the authentication phase, the voting phase and the vote cast phase.

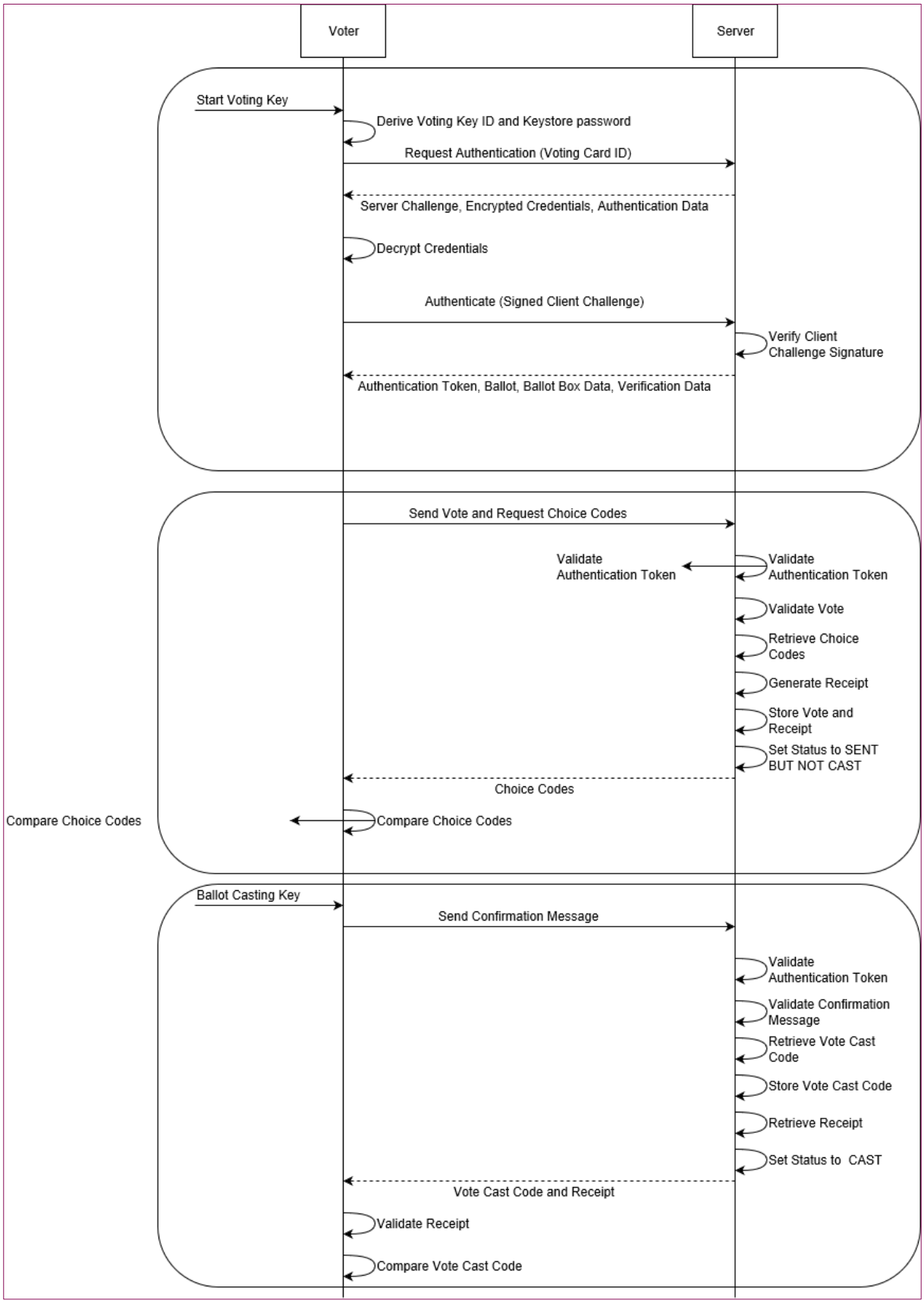

Figure 17 – Authentication, voting and casting phases (3)

5.3 Post-electoral phase

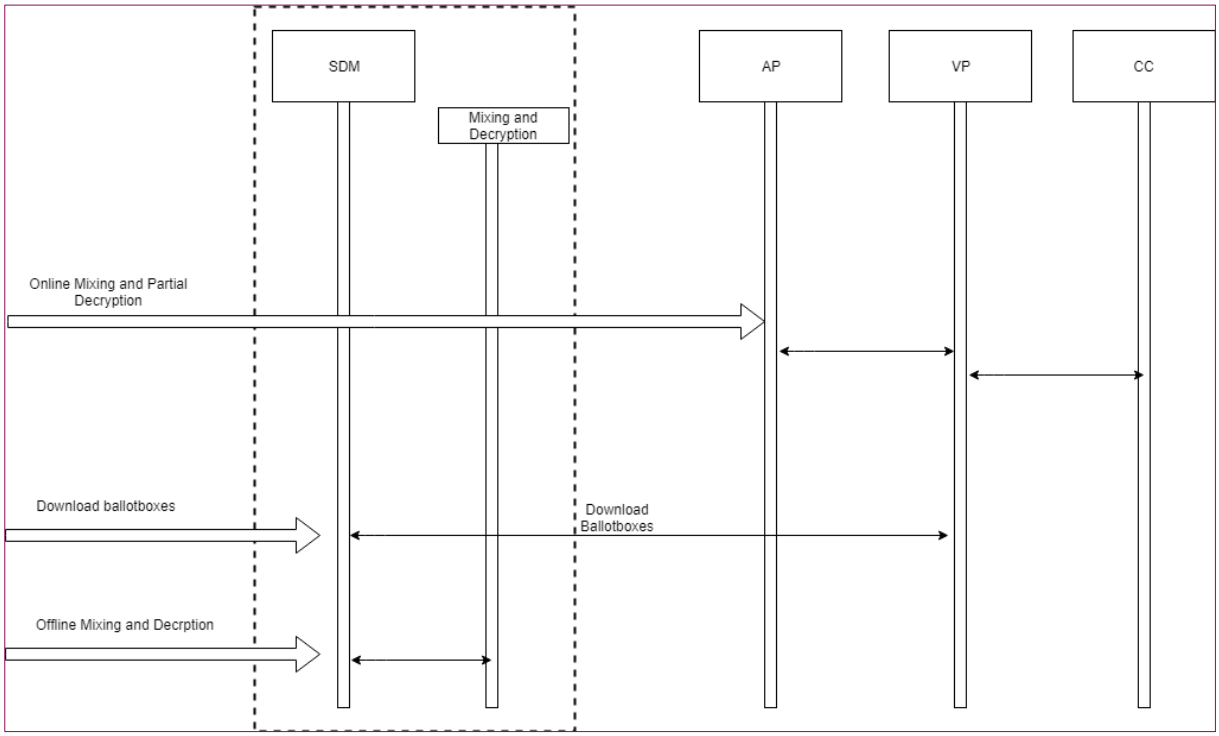

Figure 18 – Post-electoral workflow

Once the election is over, the ballot boxes can be sent to the Control Components to start the online mixing and partial decryption process. Each ballot box contains a date-time value which specifies the period of the time in which it is open to receive votes. The “Closed” status not only indicates that the ballot box will not receive more votes, but it also means that the process mixing and partial decryption can start.

After the online part of the mixing process –that was triggered through the Admin Portal (AP)– has been completed, the ballot boxes can be downloaded to the Secure Data Manager (SDM). To download the ballot boxes, the SDM connects to the Voter Portal (VP) and requests all ballot boxes that are available for downloading to be transferred to the local storage.

The next step in a secured election is to transfer the data to an offline, secured SDM, so that the cryptographic operations can be executed in an isolated environment.

The offline mixing and final decryption can start after the Admin Board has been activated with the usage of the smartcards from which the private key of the Admin Board is reconstructed.

The mixing process shuffles the ballots, permuting them first and re-encrypting them afterwards. To ensure that the process has been carried out flawlessly and without any malicious action, a proof is generated.

After Mixing, the ballot box can be decrypted to extract the information of the voting. This process receives a mixed ballot box and decrypts each of the ballots. For that, the Electoral Authority’s private key is required.

The following diagram explains the mixing and partial decryption process, with the emphasis on the mixing nodes in the Control Components:

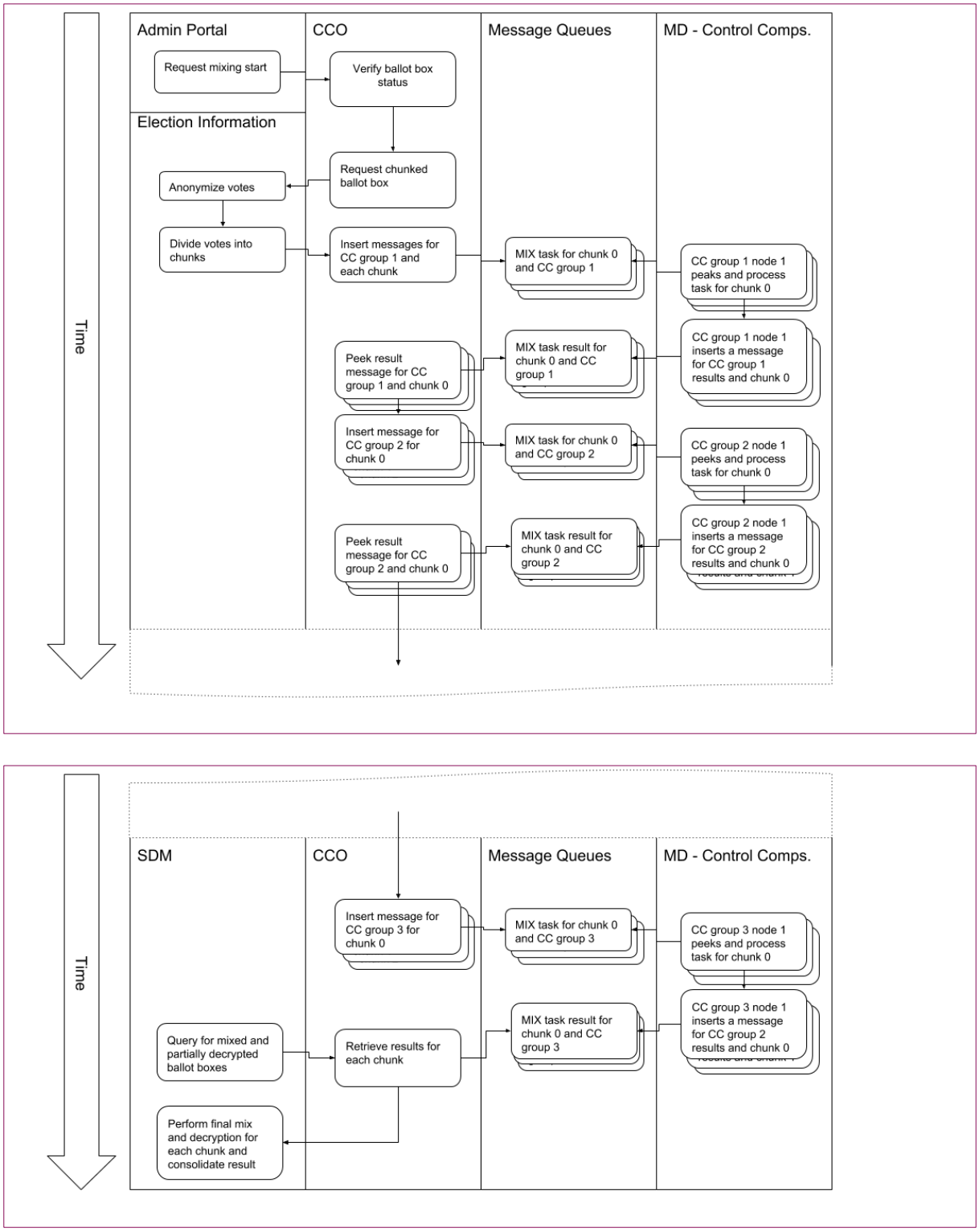

Figure 19 – Mixing and partial decryption process

6 Deployment view

Scytl sVote has a classical 3-tier design with frontend, backend and persistence (database) layers. The system is deployed with common clustering layout for fault-tolerance and high-scalability purposes. Therefore, the incoming traffic to the frontend portals is routed properly by a hardware load balancer on the transport layer (TCP) and by software load balancer (reverse proxy) on the application layer (HTTP/S).

All nodes are doubled or tripled in active-active mode. All microservices (presented as RESTful endpoints) are also deployed in multiple nodes layout, each in a separate web container. The common enterprise database uses its own capabilities for clustering support (Oracle RAC). The monitoring and logging solutions are also deployed in a dedicated node.

- **Frontend - Voter** (web servers): These components manage the Internet frontend (web connections) coming from the voters. These components are therefore accessible from the Internet and can also access the authentication services provided by the backend components.
    - To prevent the backend servers from being accessible from the Internet, the frontend servers are connected to different mutually isolated subnets using different network interfaces: The external subnet and the backend subnet. Connections from voters come from the external subnet, while frontends connect to the backend servers using the backend subnet.
- **Backend** (application servers): These components partly manage the core functionality of the online voting process. They are completely isolated from any public (Internet) access and are connected to the front-end servers and database servers using two different subnets: backend and database subnets respectively. The rest of the core functionality is delegated to the Control Components through the Message Queues.
- **Database** (database servers): These components are only accessible from the backend servers and host the election information (e.g. election configuration information, electoral roll, etc.) required to manage the online voting process. Database components use a dedicated storage for the database information storage. This dedicated storage is also connected to the external backup service.
- **Message Queue Servers:** The message queues allow communication between the backend and the Control Components. It is deployed in a cluster of two master-slave server instances with queue mirroring to allow load balancing and fast failover. In addition, they run behind a TCP load balancer configured in round robin mode. The host machine is within the same network as the backend.
- **Control Components** (standalone applications): These components manage the rest of the core functionality distributing the security responsibility among 4 identical instances (called nodes) deployed in isolated machines, with each instance containing a group of services for Choice Return Code operations and -except for the last one- Mixing operations. They are

protected by a firewall blocking all incoming connections so the only way such components can receive requests is through the Message Queues they are subscribed to. Each Control Component node holds its own database which must be independent from the backend database or other Node's database.

- **Logging component**: All the previous components send a copy of the log entries to a central logging and monitoring server. For this purpose, all the previous components are connected to a Management subnet for sending the information to this logging server.

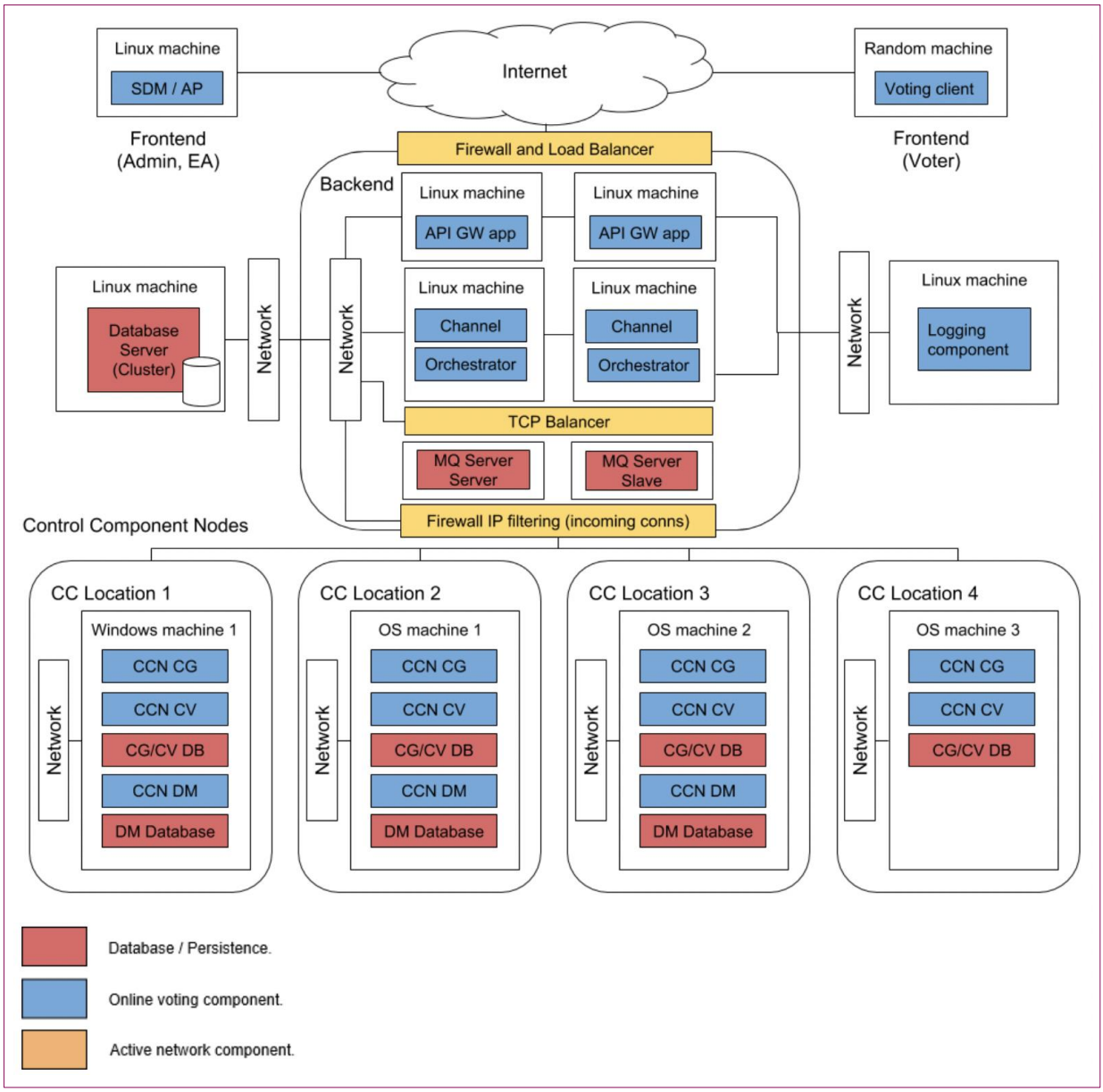

Figure 20 – Deployment architecture

The technology stack used at each layer is the following:

- **Frontend layer**: The Frontend layer (Voter Portal and Admin Portal frontends) is accessible through HTTPS (`mod_ssl`) and provides the static HTML by cluster of Apache HTTP instances (5 is commonly used) balanced by hardware load balancers and a reverse proxy. It also connects to the backend layer (TomEE) by AJP connector. Each of the tenants has a separate VirtualHost configuration with its own ProxyBalancer and ProxyPass entries.
- **Backend layer**: The Backend layer (Voter Portal and Admin Portal backends, SDM and the RESTful microservices) is not directly accessible by the end users, but only from the frontend layer using the AJP connector. All the applications are Java 8 based and require a Java Cryptography Extension, a Bouncy Castle provider and an Apache TomEE web container. Each of the deployment units is packed in separate .WAR files and are available in the webapps directory of the application server. One of the backend services, the orchestrator, connects to the RabbitMQ using a specific Java connector.
- **Database layer**: The database layer is managed by COTS RDBMS, and namely Oracle Enterprise 12c installed with the option of Real Application Clusters (RAC) for fault-tolerance and high-scalability purposes.
- **Message Queues layer**: The MQ server is a RabbitMQ 3.7.x version which requires the installation of Erlang Virtual Machine.
- **Control Components layer**: The Control Components are standalone Java 8 applications and require Java Cryptography Extension, and Bouncy Castle provider. They connect to the RabbitMQ using a specific Java connector. Their own databases are Oracle Enterprise 12c.

7 Cross-Cutting concepts

7.1 Secure Logging

Secure Logging is a key feature of the platform as it provides three main benefits:

1) Immutable logs that are easy to parse.
2) Continuous traceability throughout the different components.
3) Verification of the downloaded ballot boxes against the server logs providing means to verify that ballot boxes are intact.

Further information about the Secure Logger component can be found in the following document: Scytl sVote Secure Logger Architecture.

7.2 Spring Batch

The cryptographic services in the SDM are modules that have the following properties:

1) Consume and modify data on the file-system, and at the end write the modified data to the filesystem.
2) Handle huge amount of data that is independent of each other (processing of the data can be in parallel).
3) Execution of CPU intensive operations.

For satisfying these requirements, the module selected was Spring-batch, which is based on batch processes. The overall design of these modules is explained through an example, which in this case will be the “voting-card-generation module” that is executed in the election configuration phase.

The workflow starts with “requirement gathering”, which in this context means the preloading of the data that although is not the core part of the batch process, is necessary for executing the data transformation. Such data includes, for example, the certificates in the election, the election configuration, etc.

The second step is the execution of the processing of the different batches which is done in a multi-threaded way to maximize data throughput. A processing task usually has three key components:

- Reader
- Task
- Writer

In the case of the voting card generation, the reader is substituted by a dummy reader, because the data is to be generated from zero, and not transformed. In other modules, the reader is responsible for reading the ballot boxes, for example.

The task is the part where the actual data transformation takes place. In the case of the voting card generation, the class in action is the `VCGeneration class`.

The third step is executed by a writer object, that is fed by the task process. In the case of the voting card generation, the class is the `OutputQueueWriter`.

After the parallel execution of the tasks, the outputs are stored in a queue to preserve the order between the different items belonging to the same batch. The queue works as a buffer, so that it can be consumed by the last phase of the batch jobs, which is the actual writing of the data to the disk. This can produce from one to several files, split the data into multiple files, etc. This flow depends on the current module.

At the very last step of the Spring-batch execution, the module creates a job to produce hashes and signatures on the files that were generated, so it can be ensured that the files are not tampered with in later processes.

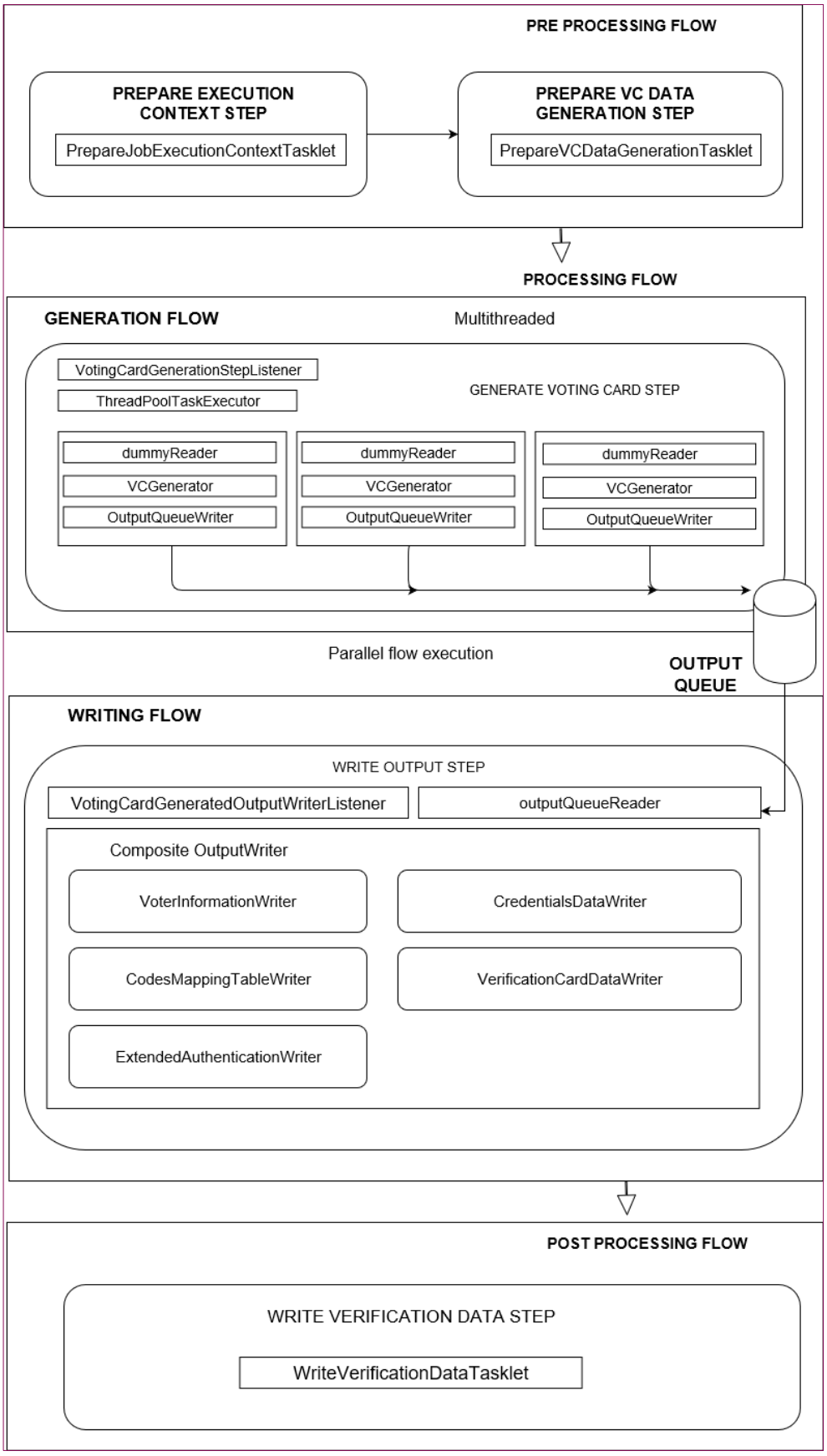

Figure 21 – Spring Batch workflow

7.3 CryptoLib

CryptoLib and CryptoLib-js are cryptographic libraries, that have the same functionality. While CryptoLib is implemented in Java, the CryptoLib-js is developed in JavaScript, and apart from platform-dependent features (for example, how to configure them, or what source to use for random generator) they are equal in features as well as complementary counterparts. Having the same cryptographic functionality implemented in both Java and JavaScript, enables an easy implementation of the required security requirements in terms of cryptography on both the server and the client side.

Further information about the CryptoLib component can be found in the following document: Scytl sVote CryptoLib Architecture.

7.4 Technology stack

The following table reflects the current technology stack used to implement the mentioned components:

Admin portal	**Frontend**	HTML5/CSS3, AngularJS
	Backend	Java, J2EE, REST, Oracle DB, Hibernate, Tomcat
Voter Portal	**Frontend**	HTML5/CSS3, AngularJS
	Backend	Java, J2EE, Spring, REST, Oracle DB, Hibernate, Bouncy Castle, TomEE, RabbitMQ
Secure Data Manager		HTML5/CSS3, AngularJS, Java, Spring, OrientDB, REST, Hibernate, Bouncy Castle
Communication		REST APIs, JSON files, AMQP

Table 3 – Technology stack

8 Glossary

Control Component (CC): At least four execution environments with different Operating Systems to ensure that they do not share a common threat. The Control Components rely on collaboration during the calculation of the sensitive data, but there is no direct communication among them. Control Components are defined by the at least four Control Component nodes, that are participating in the calculation of the same cryptographically sensitive data.

Control Component node (Control Component node) - Node (N): A group of computers (can be only 1) that have the responsibility to calculate ¼ of the cryptographically secure operations assigned to the CC they are participating in. All instances of the same group have the same configuration and take part in the calculation of the same cryptographic operation.

Control Component node instance (CCNI) - Node instance (NI) - Instance (I): A single computer instance (virtual or physical) that is part of a Control Component node.

Control Component Orchestrator (CCO): Orchestrator that is responsible for distributing the tasks among the Control Component node that are participating in the mixing / Choice Return Code generation / Choice Return Code verification.

RC3: Result Codes Control Component.

MC2: Mixing Control Component.

RC-node: Result Codes Control Component node.

M-node: Mixing Control Component node.

CC index: The number that identifies the Control Component that a node belongs to, e.g. Control Component node 1 has a node index of 1, and it defines a node that belongs to the first Control Component.

M-chunk: A data chunk produced by one of the M-nodes.

RC-chunk: A data chunk produced by one of the RC-nodes.

9 Appendix

9.1 Component dependency breakdown

9.1.1 Voter Portal

API Gateway: ov-commons, CryptoLib, javax.json, jackson-jaxrs, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Authentication: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Certificate Registry: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Extended Authentication: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Election Information: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Key Translation: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Voter Material: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Vote Verification: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Voting Workflow: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee.

Orchestrator: ov-commons, CryptoLib, javax.json, jackson-jaxrs, hibernate, retrofit, Secure Logger, bouncy-castle, ov-keystore, j2ee, rabbitmq-ampq-client.

OV-Client (Frontend library): node-forge, sjcl, lodash, q, semver, xhr2.

Voter portal (Frontend website): ov-client, angular, lodash, showdown, soocss, cssmin.

9.1.2 Admin Portal

Backend: Spring-security, spring-data-jpa, spring-webmvc, spring-hateoas, ov-commons, CryptoLib, javax.json, jackson-jaxrs, retrofit, Secure Logger, bouncy-castle, ov-keystore.

Frontend: Angular (and angular extensions), lodash, showdown, soocss, cssmin, ng-file-upload, query, moment-timezone.

9.1.3 Secure Data Manager

Voting Card Generation: Spring-webmvc, orient-db, spring-batch, spring-hateoas, ov-commons, CryptoLib, javax.json, jackson-jaxrs, retrofit, Secure Logger, bouncy-castle, ov-keystore.

Tallying: Spring-webmvc, orient-db, spring-batch, spring-hateoas, ov-commons, CryptoLib, javax.json, jackson-jaxrs, retrofit, Secure Logger, bouncy-castle, ov-keystore.

Mixing: Spring-webmvc, orient-db, spring-batch, spring-hateoas, ov-commons, CryptoLib, javax.json, jackson-jaxrs, retrofit, Secure Logger, bouncy-castle, ov-keystore.

SDM Backend: Shares, jersey, spring-web, spring-hateoas, spring-boot, hibernate, orient-db, ov-commons, CryptoLib, javax.json, jackson-jaxrs, Secure Logger, bouncy-castle, ov-keystore, codehaus-groovy.

SDM Frontend: Angular (and angular extensions), lodash, ng-file-upload.

SDM Executable: SDM frontend, SDM backend, tomcat, chromium, jetty.

9.1.4 Control Components

Choice Return Codes Generation: Spring, control-components-commons, ov-commons, rabbitmq-amqp-client, bouncy-castle.

Choice Return Codes Verification: Spring, control-components-commons, ov-commons, rabbitmq-amqp-client, bouncy-castle.

Distributed Mixing: Spring boot, control-components-commons, ov-commons, scytl-mixing, rabbitmq-amqp-client, bouncy-castle.

Control Components Commons: Spring, CryptoLib, Log4j, rabbitmq-amqp-client, ov-commons, oracle-jdbc6.

9.1.5 Component explanation

Secure Logger: Scytl implementation of immutable logging framework.

CryptoLib: Collection of cryptographyl functions.

Ov-commons: Collections of shared beans, utility and factory classes

Scytl-Mixing: Project-independent engine for mixing vote sets.

Javax.json: JSON library.

Jackson-jaxrs: Library for implementing RESTful services.

Hibernate: ORM module for accessing databases.

Retrofit: Library for facilitating communications between different components of http/https requests.

Bouncy Castle: Alternative to the Java Cryptography implementation.

OV-keystore: Library for storing public and private keys for inter-service authentication.

J2EE: Java enterprise framework.

OV-client: JavaScript API for interacting with the voter portal frontend.

Angular: JavaScript framework for creating feature-rich html contents.

Loadash: JavaScript framework for modular applications.

Showdown: JavaScript markdown to HTML converter.

SOOCSS: Collection of standard html building blocks for applications developed by Scytl.

Cssmin: Library for minifying the CSS files.

Spring-security: Security-related library provided by Spring.

Spring-data-jpa: Persistence library provided by Spring.

Spring-webmvc: Model-View-Controller library provided by Spring.

Spring-hateoas: Library for facilitating RESTful API creation provided by Spring.

Spring-batch: Batch processing library provided by Spring.

Spring-web: Web application library provided by Spring.

Spring-boot: Lightweight framework for standalone applications provided by Spring.

Orient-db: NoSQL database used for storing json documents.

Shares: Library developed by Scytl for interacting with smart-cards.

Jersey: RESTful webservice framework.

Tomcat: Embedded webserver in SDM.

Chromium: Embedded webbrowser in SDM.

Jetty: Embedded webserver in SDM.

Rabbitmq-amqp-client: Driver for Java communication with the RabbitMQ queues.

Oracle-jdbc6: Java connector for the Oracle database.

9.2 EV Solution intellectual property rights notice (the Notice)

Scytl sVote is part of a larger system called EV Solution, developed under the "Framework Agreement" entered into by and between Post CH Ltd (Swiss Post) and Scytl Secure Electronic Voting, S.A. (Scytl) on September 30th, 2015.

Parts of this EV Solution system and other relevant details are defined below.

9.2.1 Definitions

The following terms shall have the meanings specified below:

"**EV Solution**" means an online voting system consisting of the Scytl Standard Software (also referred to as Scytl sVote or Scytl Online Voting 2.0) in combination with the Swiss Post-Scytl Software, and all the associated middleware provided by Scytl as a bundle with the Scytl Standard Software and the Swiss Post-Scytl Software. Software below middleware (e.g. Linux OS and Windows OS and Oracle software) that are needed to run the EV Solution are not part of the EV Solution.

"**Intellectual Property Rights**" or "**IPRs**", for the purposes of this Notice and pursuant to the Framework Agreement, means copyright and patent rights (if any), know-how and trade secrets, performance rights and entitlements to such rights.

"**Scytl Online Voting 2.0**" is the brand name that was used to identify Scytl Standard Software in the market.

"**Scytl Standard Software**" means all software developed by Scytl for the EV Solution, whose architecture, specifications and capabilities are described in Scytl sVote documents, excluding Swiss Post-Scytl Software and software developed by Scytl independently to the EV Solution.

"**Software**" means software code (source code and object code), user interfaces and documentation (preparatory documentation and manuals) and including releases and patches etc.

"**Scytl sVote**" means the registered trademark proprietary to Scytl, that identifies Scytl Standard Software in the market.

"**Swiss Post-Scytl Software**" means the software developed for the EV Solution (excluding Scytl Standard Software) pursuant to the Framework Agreement. Swiss Post-Scytl Software comprises of the following:

i. Key Translation Module: A mapping service that translates external IDs to internal IDs for specific entities so that external systems can integrate with sVote.

ii. Swiss Post Integration Tools: A group of applications that allow the integration between Swiss Post's applications and sVote through file conversions.

iii. Swiss Post Voting Portal Frontend: Frontend application that guides the voters throughout all the voting steps enabling them to successfully cast a vote for a particular election.

9.2.2 Copyright notice

9.2.2.1 Scytl Standard Software

All intellectual property rights in the Scytl Standard Software are Scytl's sole property. Scytl owns and shall retain all rights, title and interest in and to the Scytl Standard Software. Scytl Standard Software is licensed to Swiss Post under the terms and conditions described in the Framework Agreement.

9.2.2.2 Swiss Post-Scytl Software

All intellectual property rights in the Swiss Post-Scytl Software are the joint property of Scytl and Swiss Post (Joint IP).

9.2.2.3 EV Solution

All intellectual property rights in the EV Solution other than Joint IP will be owned by Scytl or by third parties as applicable.