VERSION 9

13 DECEMBER 2019

Pervasive Intelligence
Emerging technologies. Real impact.

TRUSTED AND TRANSPARENT VOTING SYSTEMS

VERIFY MY VOTE DEMONSTRATOR REQUIREMENTS AND DESIGN

PRESENTED BY: MATTHEW CASEY

PERVASIVE INTELLIGENCE LTD

38 TAVISTOCK ROAD, FLEET, HAMPSHIRE, GU51 4EJ

DOCUMENT INFORMATION

STATUS

Distribution	University of Surrey King's College London Electoral Reform Services
Status	Revised during implementation.

VERSION

Version	Author	Date	Comment
1	Matthew Casey	11 June 2018	Initial draft.
2	Matthew Casey	26 June 2018	Updated from internal review comments by the University of Surrey and King's College London: added additional election parameter publication to DL; send tracker number to voter; clarification of DL technology.
3	Matthew Casey	18 July 2018	Updated following discussion with Electoral Reform Services to add requirements to isolate voter private keys, enable post-voting sign-up for verification, and include no votes in the distributed ledger record for an election. Additional design changes to turn on or off access to the distributed ledger data via its interface for internal integrity checking during the election, and to link to an external questionnaire after voter verification.
4	Matthew Casey	30 November 2018	Revised for changes in implementation detail, including use of Verificatum with multiple tellers, and change of distributed ledger to Quorum.
5	Matthew Casey	17 December 2018	Additional commands to allow the creation of a Verificatum teller, plus changes to vote encryption and vote anonymisation and decryption commands.

Version	Author	Date	Comment
6	Matthew Casey	14 January 2019	Added a command to allow a voter's tracker number to be obtained given the voter's alpha and beta commitments and their encryption public key.
7	Matthew Casey	18 January 2019	Indicate that all teller proofs should be published independently. Add proof of knowledge to vote encryption.
8	Matthew Casey	8 February 2019	Revised software architecture for the Public VMV to use Ruby on Rails for rapid prototyping and an appropriate Quorum interface.
9	Matthew Casey	12 December 2019	Modified to optionally allow voter's keys to be pre-defined and not created during parameter initialisation and to allow the supply of externally encrypted votes.
10	Matthew Casey	13 December 2019	Added voter commands to allow separate creation of voter keys, vote encryption and commitment decryption.

GLOSSARY

Blockchain	A continuous, growing list of blocks of data which are cryptographical assured to be resistant to modification, and held within a distributed ledger.
Comma Separated Values (CSV)	A text file format to exchange multiple items of data with the same attributes. Rows in the file represent different data items, while columns, which are separated by commas, represent attributes.
Content Management System (CMS)	A system used to create and maintain digital content. A CMS abstracts the publication of content from its maintenance. For example, a CMS may be used to build an interactive website from source data, providing management tools to control the content and (re-)publish the website following content changes.
Distributed Ledger (DL)	Used to store data in a distributed system such that the data is replicated and synchronised across a number of nodes that share a consensus of the stored data.
Endpoint	A web service endpoint refers to a URI provided by a web service for a specific request.
Hypertext Transfer Protocol (HTTP)	Communication protocol used to exchange data with a web server.
Identity Credentials	Security credentials held by a person which are used to identify them. Within this document, these explicitly represent cryptographic identifiers, such as through a private and public key pair.
Java Archive (JAR)	Contains the compiled Java byte code for one or more Java classes. JAR files are typically used to encapsulate a complete Java software library or program.
Model-view-controller (MVC)	A software design pattern which explicitly splits responsibility for models (data and rules), views (output interfaces) and controllers (responding to actions, interacting with models and outputting views).
Private and Public Keys	For use in a cryptographic system where a public key can be widely disseminated and used, for example, to encrypt information which can only be decrypted using the privately held private key.
Representational state transfer (REST)	A method by which web services are provided which enables the definition of stateless operations on a server. The operation required by each RESTful request is defined through the URI and HTTP header information, which includes the type of operation (for example, a GET request to retrieve data, a POST request to store data, a PATCH request to update data and a DELETE request to delete data) together with any relevant state such that each request is self-contained.

Security Credentials	Within this document, these represent the credentials given to a voter in order for them to cast their vote. For example, security credentials are used to gain access to the voting pages of the ERS voting website.
Tracker Number	A unique number which is assigned to a voter and hence to the voter's cast vote. Tracker numbers are used with cast votes to ensure that the voter remains anonymous when viewing plaintext votes, since the association between tracker number and voter is never made public.
Uniform Resource Identifier (URI)	A reference to a networked resource. A URI consists of a scheme, optional user credentials, host, optional port, optional path and optional query parameters. For example, "http://192.168.0.3:3000/query" has scheme "http", a host represented by an IP address "192.168.0.3", port "3000" and path "query", with no user credentials or query parameters specified.
Verifiable Voting	A scheme whereby voters can cast their vote and certain properties of the election can be verified by voters or other observers. For example, eligibility verifiability makes it possible to verify that the vote tally is formed from votes cast by only those people who are eligible to vote; individual verifiability makes it possible for a voter to check that their vote was cast and counted correctly; and universal verifiability makes it possible to check that the tally has been computed correctly from the cast votes.
Verification Parameters	Within this document this term is used to encapsulate all of the parameters needed for the election to be verifiable. This is further identified as overall election verification parameters which are specific to the election, but not specific for an individual voter, versus voter verification parameters which are specific for a voter.
Vote	Within this document, a vote represents a vote cast by a voter. Votes may be held as plaintext or encrypted.
Web Bulletin Board (WBB)	In electronic voting systems, a WBB is an append-only data storage mechanism which allows data to be published to it that cannot subsequently be modified.

TABLE OF CONTENTS

INTRODUCTION

PURPOSE

The purpose of this document is to describe the requirements and high-level design elements for the Verify My Vote (VMV) demonstrator which is being implemented in conjunction with the Electoral Reform Services (ERS) to demonstrate verifiable voting in an election.

SCOPE

The purpose of the demonstrator is to be able to provide a working implementation of key aspects of verifiable voting using distributed ledger (DL) technologies. The demonstrator is being implemented by partners in the "Trusted and Transparent Voting Systems" EPSRC funded project (EP/P031811/1) [1], including the University of Surrey, King's College London, the Electoral Reform Services, Monax Industries and Crowdcube Limited.

As a demonstrator, only key aspects of the required technologies will be implemented and trialled in a real election alongside existing voting processes. The focus is on understanding how such verifiable voting can be integrated within existing processes and technologies, and as such, certain aspects of the required cryptography will not be included in the demonstrator, which, for example, rely upon external voter identity credentials.

This document brings together the requirements and high-level design for the demonstrator in order to establish agreement on what is to be implemented. This document supersedes the previous "Verify My Vote (VMV): Requirements Specification" document [2] by formalising the requirements and providing a high-level design for each required component of the demonstrator. As such, this document concentrates on providing sufficient detail for the demonstrator to be developed using an agile approach which takes into account that the demonstrator will evolve as various aspects of the research elements and their associated components are realised during implementation.

OVERVIEW

This document loosely adheres to the ISO/IEC/IEEE "International Standard – Systems and Software Engineering – Life Cycle Processes – Requirements Engineering" [3] and IEEE "Standard for Information Technology — Systems Design — Software Design Descriptions" by specifying requirements and how these are met through a series of design views governed by viewpoints [4]. The requirements are defined within the following section, with the high-level design in the subsequent section.

REQUIREMENTS

OVERVIEW

The over-arching requirement is to be able to demonstrate how verifiable voting may be incorporated into an existing, non-verifiable voting system. Since the demonstrator will integrate with a real-world system, and be trialled on a real election, the integration must be such that the existing election system processes must remain intact and unperturbed in the event that the verifiable aspects of the system become non-operational, allowing the election to be completed successfully. This necessitates a low-risk integration which does not modify the key processes of the existing system, while enabling verifiability to be demonstrated.

The existing system operated by ERS uses a content management system (CMS) to define the details of an election, and then publish this as an interactive website in order for voters to be able to log into the website and cast their votes. Voters are provided security credentials in order to be able to login to the website. When they cast their vote, the vote is stored in plaintext within a secure database. The website allows voting to take place even if the user's browser has JavaScript [5] disabled. Elections can also have postal and telephone votes, and under particular circumstances, votes can also be cancelled by voters by telephone. All of the resulting vote or vote cancellation data is maintained within the secure database.

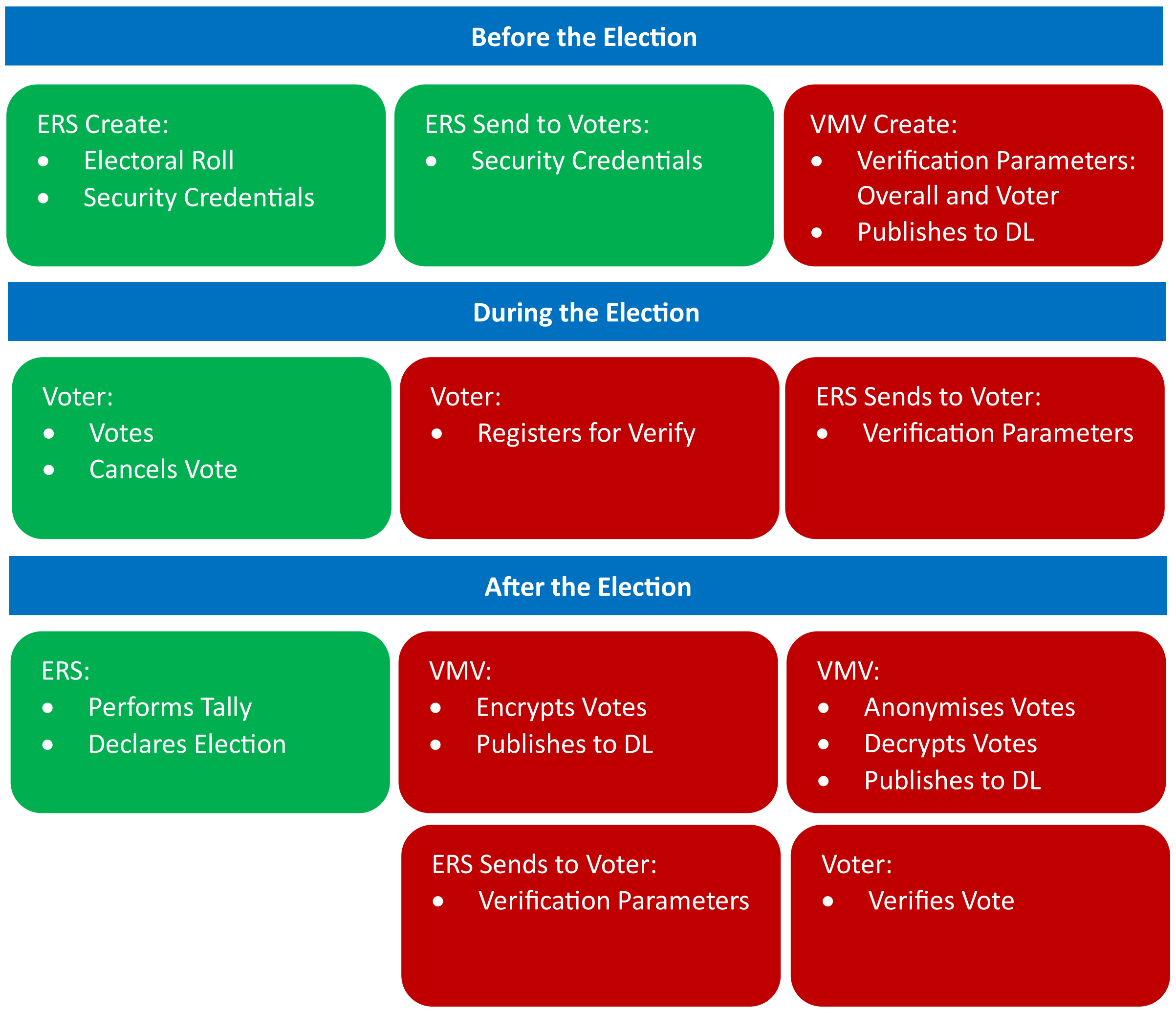

FIGURE 1: EXISTING (GREEN) VS. NEW (RED) PROCESSES

To include verifiability within this process, there are a number of stages that have been agreed where additional tasks will take place to generate and use verification parameters, and to publish the relevant vote information to enable a voter to verify their vote. Figure 1 shows an overview of the election process highlighting the relevant existing (green) and new parts (red) of the process, which are described below. Note that in these requirements we do not specify how the verification is achieved, but rather refer to the verification parameters and tasks necessary to achieve verification without reference to the cryptographic protocol or operations. These will be provided later in the design. Nonetheless, we assume that whichever scheme is used for verifiability that it fits within these identified tasks, such as publishing anonymised plaintext votes at the end of the election for voter verification.

BEFORE THE ELECTION

Existing:

- ERS define the election content, parameters and electoral roll for a client. The electoral roll is then used to generate the security credentials for each voter, and these are sent to the voter. All voter data is stored within the secure database. Certain parts of the overall election verification parameters are also published to the DL.

New:

- To enable verifiability, verification parameters for the overall election and for each voter need to be created, including the relevant security keys. In an ideal system, each voter would already have their own identity credentials consisting of a private key and a published public key. However, since this infrastructure does not yet exist, this step will also generate a private and public key pair for each voter. The private keys will be held externally from ERS.
- The verification parameters include all necessary values for the cryptographic protocol being implemented. These are created for the overall election and per voter, with the private and public aspects of the overall election verification parameters stored in the secure database, but only the public aspects of the voter's identity parameters stored in the database to maintain an integrity guarantee. The public elements of the overall election and voter verification parameters are also published to the DL.

To minimise change to the existing ERS set-up process, the verification parameters will be generated in two stages. The first stage will be run externally to ERS to generate voter identity parameters, allowing the private keys to be stored separately. The second stage will run within the secure ERS network to allocate identity parameters to voters. These programs will take as input key information about each voter, and output data for the election and for each voter that can be stored externally or within the secure database. The secure database will be modified to permit storage of the public data.

DURING THE ELECTION

Existing:

- The voter can cast their vote using their security credentials. Depending upon the client, votes may be cast via the website, post or telephone.
- If the voter has made a mistake, for example by inadvertently sharing their security parameters, they can contact ERS to have their current vote cancelled so that they can re-vote.
- All vote data is maintained in the secure database, with votes in plaintext.

New:

- When voting via the website, the voter will be presented with an option to register their interest in verifying their vote once the election has completed. Note that in a fully implemented verifiable election, all voters would automatically have the ability to verify, so an option to register for verification would not be required.

- If the voter registers for verification, this will be recorded with their vote information and an email sent to them by ERS which will include the necessary verification parameters.

To support registration for verifiability, the ERS system will be modified to display the registration option, recording registration in the secure database and sending the corresponding voter email. A voter will also be able to login into the ERS system after they have voted in order to register for verifiability at a later stage, but prior to the completion of the election.

In a fully implemented verifiable election, voters would use their own private security credentials to sign their encrypted vote which is generated on their own local computer, such that only signed and encrypted votes are transmitted and stored, with the corresponding data published to the DL for transparency. A similar secure process would also be implemented for vote cancellation, while postal and telephone votes would not be accepted. However, implementing this in the demonstrator would significantly impact on the existing election system. The demonstrator will therefore not include this real-time vote encryption and will instead encrypt and sign the votes at the end of the election. This ensures that all postal and telephone votes, together with vote cancellations, are captured for verification, rather than capturing snapshots of information which will need modification during the election.

AFTER THE ELECTION

Existing:

- ERS complete the election, including performing the tally and declaring the election.

New:

- Each of the votes in the ERS secure database is encrypted, signed and published to the DL.
- The set of votes is then anonymised and decrypted before being published in plaintext to the DL. This ensures that the information on the DL does not associate any plaintext vote with a voter.
- ERS sends all voters who registered for verifiability their verification parameters which are needed to look-up their vote.
- The voter accesses a website to verify their vote. This website is acting as a user interface to the DL, and hence can be used to view all other published information from the election.

A separate program to the ERS system will be used which will take as input the votes cast by each voter and will output the votes in encrypted form. This program must run within the secure network because it will be using each voter's plaintext vote. However, the encrypted output will be in a form that can be published to the DL and accessed publicly.

A separate program will also be used to anonymise and decrypt the votes such that the plaintext votes cannot be associated with any voter. However, each vote is associated with a unique tracker number that can be used to look-up votes, and the voter then uses their verification parameters to obtain their tracker number. The linkage between tracker number and voter is never released publicly. The output will be a list of votes against their tracker numbers. Although the anonymisation part of this process can be completed just with the encrypted votes and election public key, the decryption requires the election private key information held securely. This program will therefore run within the secure network, with the resulting anonymised plaintext votes published to the DL for public access.

To support those voters who have registered interest in verifiability, the ERS system will be modified to send the corresponding voters their verification parameters by email.

A web user interface for the DL will be used to allow voters to verify their vote using their verification parameters. This interface will also permit access to all other information published to the DL, such as the encrypted votes.

FUNCTIONAL COMPONENTS

These changes therefore require modification to the existing ERS system, together with the production of four new functional components:

MODIFICATION TO ERS

The changes required to the ERS system to support verifiability are:

- To output data about each voter for the generation of verification parameters.
- To input the generated overall election verification parameters and store these.
- To input the generated verification parameters for each voter, excluding their identity private keys, and store these against each voter.
- To present voters with the option to register for verifiability and store the result against each voter, emailing voters who register with relevant verification information at the time they register and at the end of the election.
- To output plaintext votes for each voter for encryption, anonymisation and decryption.

VERIFICATION PARAMETER INITIALISATION (PI)

A program which will run in two stages externally to ERS and internally within the secure network which will:

- Externally from ERS:
  - Generate and output the private and public verification parameters for the overall election.
  - Take as input the number of voters on the electoral roll and generate and output the private and public verification parameters for each voter.
- Internally within ERS:
  - Take as input the private and public verification parameters for the overall election and the public aspects of the verification parameters for each voter, and import these into the ERS secure database, associating voter verification parameters with each voter.

VOTE ENCRYPTION (VE)

A program running within the secure network which will:

- Take as input the overall election and voter verification parameters, and the plaintext votes for each voter who has voted, including cancelled votes, and any no votes.
- Encrypt, sign and output the encrypted votes.

VOTE ANONYMISATION AND DECRYPTION (VA)

A program running within the secure network which will:

- Take as input the overall election and voter verification parameters, and the encrypted votes for each voter.
- Anonymise the votes and decrypt them, outputting the anonymised, plaintext votes with their tracker numbers.

DISTRIBUTED LEDGER INTERFACE (DI)

A web interface with public and private functionality which acts as a user interface to the DL. The private functionality will be accessed by authorised users only, and will:

- Take as input the public elements of the overall election and voter verification parameters and publish these to the DL.
- Take as input the encrypted votes and publish these to the DL.
- Take as input the plaintext votes with their tracker number and publish these to the DL.

The public functionality will allow any user to access the data published to the DL:

- Allow a user to browse the encrypted vote information.
- Allow a user to browse and search the plaintext vote information.

USER CHARACTERISTICS

It is expected that two types of user will use the demonstrator:

Administrators:	who will maintain and control the ERS and separate demonstrator elements, for example by running the various component programs for initialisation, vote encryption and decryption.
Voters:	who will use the ERS system to cast their votes and use the VMV interface to verify their vote or view other election data published to the DL.

LIMITATIONS

The purpose of the demonstrator is to show how key aspects of verifiable voting can be incorporated into an existing election system. As such, the focus of the demonstrator is limited to those aspects necessary to demonstrate this, rather than building a fully verifiable system. As such, integration of the verifiable processing has been kept as separate programs which can be run alongside the existing election system, with a separate interface used to access the DL.

A full implementation of verifiable voting would require a more tightly coupled integration of processing, such as offering real-time encryption of votes on a voter's computer. However, such steps also require additional public infrastructure to support secure voter identification credentials, and hence the demonstrator is limited to showing those aspects of interest which include how verification can be integrated and how a DL can be used to make the election processes more transparent.

Another key aspect which requires additional infrastructure is the availability of independent organisations who perform tasks during the vote tally (referred to as tellers). For example, to cryptographically anonymise votes, several independent tellers can be used to shuffle encrypted votes such that no one teller can identify the source of a vote with a voter since each teller performs a random shuffle of the encrypted votes without knowing the original vote order (linked to voter). A threshold set of tellers is then needed to decrypt the votes before tallying. However, in the demonstrator, since the votes are held in plaintext by ERS, which will also hold the private security parameters for the election, the demonstrator will effectively have only one teller to perform the shuffle and decryption.

ASSUMPTIONS AND DEPENDENCIES

The demonstrator is underpinned by three key components:

1. An existing election system which already enables elections to be run.

2. A suitable cryptographic verification scheme which can be run alongside the existing election system.
3. A suitable DL architecture that can be deployed and used by administrators and voters, via a user interface.

These requirements assume that a suitable choice for each of these components is made. While 1) has been selected as the ERS system and used to form these requirements, 2) and 3) are left to be confirmed during the design stage of the demonstrator, within the constraints provided by these requirements. The choice for 2) depends upon the evaluation and implementation of ongoing research which may be subject to rapid change during development in order to ensure that implementation constraints can be satisfied while maintaining security. The choice for 3) depends upon the availability of suitable third-party products which can achieve the required consensus for data storage, appropriate interface access and deployment constraints.

APPORTIONING OF REQUIREMENTS

Development of the demonstrator will be in software development sprints. The requirements for each will be selected prior to the start of each sprint. This will enable, for example, key framework elements to be implemented while allowing for further evaluation and change of the selected cryptographic verification scheme and DL platform. At the end of each sprint, the application will be reviewed and evaluated against the selected requirements.

DETAILED REQUIREMENTS

The following specifies the functional and non-functional requirements for the changes to the ERS system and the new components needed for the demonstrator. Note that the requirements for the ERS system are limited to only those aspects which need to change, and do not include any existing aspects of the ERS system.

EXTERNAL INTERFACES

[EI1] The ERS system shall export voter information to CSV file(s) which shall be imported by the PI component.

[EI2] The PI component shall export overall election verification parameters to CSV(s) file which shall be imported by the ERS system.

[EI3] The PI component shall export voter verification parameters to CSV(s) file which shall be imported by the ERS system and DI component.

[EI4] The ERS system shall export overall election verification parameters and voter verification parameters to CSV file(s) which shall be imported by the VE and VA components.

[EI5] The ERS system shall export cast votes to CSV file(s) which shall be imported by the VE component.

[EI6] The VE component shall export encrypted votes to CSV file(s) which shall be imported by the VA and DI components.

[EI7] The VA component shall export plaintext votes and other verifiability evidence to CSV file(s) which shall be imported by the DI component.

[EI8] The ERS system shall present the option for a voter to register for verification via the voting web interface.

[EI9] The ERS system shall send voters who have registered for verification their verification parameters via email.

[EI10] The PI, VE and VA components shall be able to run on a computer supplied within the ERS secure network.

[EI11] The DI component shall interface to the DL.

[EI12] The DI component shall provide a voter user interface.

FUNCTIONS

Modification to ERS

[ER1] The ERS system shall be able to export voter information for each voter before the election is started.
[ER2] Voter information imported to or exported from the ERS system shall include the unique voter identifier for each voter.
[ER3] The ERS system shall be able to store overall election verification parameters.
[ER4] The ERS system shall be able to export overall election verification parameters.
[ER5] The ERS system shall be able to store public voter verification parameters for each voter.
[ER6] The ERS system shall be able to export public voter verification parameters for each voter.
[ER7] During the election, the ERS system website shall present voters with an option to register for vote verification.
[ER8] When a voter registers for vote verification, the ERS system shall record the registration and send the voter an email containing the voter's verification parameters.
[ER9] The voter's verification parameters sent to them during the election shall not include any verification parameters which make it possible for the voter (or anyone else) to identify the tracker number associated with the voter's cast vote.
[ER10] At the end of the election, the ERS system shall send all voters who registered for vote verification an email containing the voter's verification parameters.
[ER11] The voter's verification parameters sent to them after the election shall include all verification parameters necessary for them to verify their vote.
[ER12] The ERS system shall be able to export votes cast by voters after the end of the election.
[ER13] Exported cast votes shall include for each voter the unique voter identifier, verification parameters, plaintext vote and whether the vote was cancelled or a no vote.

Verification Parameter Initialisation (PI)

[PI1] The PI component shall be able to import information for each voter exported from the ERS system.
[PI2] The PI component shall be able to generate and export overall election verification parameters, including appropriate keys.
[PI3] The PI component shall be able to generate and export voter verification parameters for each voter.

Vote Encryption (VE)

[VE1] The VE component shall be able to import the overall election verification parameters exported by the ERS system.
[VE2] The VE component shall be able to import voter verification parameters exported by the ERS system.
[VE3] The VE component shall be able to import cast votes for each voter exported from the ERS system.
[VE4] The VE component shall encrypt, sign and export each vote.

Vote Anonymisation and Decryption (VA)

[VA1] The VA component shall be able to import the overall election verification parameters exported by the ERS system.
[VA2] The VA component shall be able to import voter verification parameters exported by the ERS system.
[VA3] The VA component shall be able to import encrypted votes exported by the VE component.
[VA4] The VA component shall anonymise and decrypt the encrypted votes and export each plaintext vote with its tracker number, and other verification information.

Distributed Ledger Interface (DI)

[DI1] The DI component shall allow administrators to publish data to the DL.
[DI2] The DI component shall be able to import the public elements of the overall election and voter verification parameters exported by the PI component.
[DI3] Imported public elements of the overall election and voter verification parameters shall be published to the DL.
[DI4] The DI component shall be able to import encrypted votes exported by the VE component.
[DI5] Imported encrypted votes shall be published to the DL.
[DI6] The DI component shall be able to import plaintext votes exported by the VA component.
[DI7] Imported plaintext votes shall be published to the DL.
[DI8] The DI component shall allow voters to browse through all data published to the DL.
[DI9] Voters will be able to search for published plaintext votes using their verification parameters.
[DI10] Voters will be able to verify the integrity of the DL.

OPERATION

[OP1] All components shall be installed directly or via a network connection to the associated computer.
[OP2] All verification data stored by the ERS system or externally shall be held in persistent storage.
[OP3] All data published to the DL shall be held in persistent storage.
[OP4] All persistent storage shall be automatically backed-up to independent storage.
[OP5] The DI component user interface shall be available via a public Internet address.

DESIGN CONSTRAINTS

[DC1] The DI component user interface shall produce HTML and CSS [6] for display using the latest versions of Google Chrome [7], Apple Safari [8] and Mozilla Firefox [9] on desktop and mobile devices.

SOFTWARE SYSTEM ATTRIBUTES

[SS1] Administrator access to the DI component shall be protected to allow authorised access only.
[SS2] All communication with the DI component shall use encryption with SSL certificates with the HTTPS protocol.

HIGH-LEVEL DESIGN

OVERVIEW

The demonstrator consists of five functional components: 1) modified ERS system, 2) Verification Parameter Initialisation (PI) component, 3) Vote Encryption (VE) component, 4) Vote Anonymisation and Decryption (VA) component, and 5) the Distributed Ledger Interface (DI) component. The ERS system, PI, VE and VA all run within the ERS secure network, accessing data which is kept private, the PI will also run externally, but privately, to ERS in order to generate voter verification parameters, while the DI is publicly accessible as an interface to the DL, albeit operating securely with administrator-only functions.

Underlying the PI, VE and VA components is the suitable choice of a cryptographic scheme for verifiable voting, together with the choice of DL and its configuration. The cryptographic scheme chosen for the demonstrator is Selene [10], which provides the required levels of verifiability and produces plaintext votes for verification to provide a better user experience. The DL is implemented using Quorum [11].

VERIFIABILITY USING SELENE

Selene consists of a number of cryptographic stages during the election which enable verification. Alongside voters, a full implementation of Selene requires multiple independent tellers (vote counters) who are each responsible for shuffling encrypted data in a way which makes it difficult to identify the source of the information. For example, prior to the release of the pairs of tracker numbers and plaintext votes, the encrypted pairs are mixed by successive tellers such that at the end, no one teller can re-identify the source of the vote, and hence it can be decrypted without compromising the identity of the voter. Although in the demonstrator the use of multiple tellers is not strictly required because the election system already stores plaintext votes against a voter's identity, in order to fully evaluate the demonstrator, multiple tellers will be used. Just as for the DL nodes, this will require external, independent hosting of teller nodes which the VA component will communicate with.

For election transparency, Selene also relies upon being able to publish information to a Web Bulletin Board (WBB), which is a storage mechanism which prevents modification of any information which has been published to it, thus maintaining a reliable record of the election. In the demonstrator, the role of the WBB will therefore be taken by the DL.

The following summarises the key stages of the Selene process, highlighting which elements are to be implemented within the PI, VE and VA components for the demonstrator, and where the DL takes the role of the WBB.

Set-up

The set-up process initialises all of the election cryptographic parameters and therefore corresponds to processing performed by PI, with data published via DI to the DL.

1. Generate election parameters, including cryptographic keys **[PI]**.
2. Generate and encrypt tracker numbers **[PI]**.
3. Publish public elements of the election parameters and encrypted tracker numbers to the **[DL]**.
4. Mix the encrypted tracker numbers and assign one (encrypted) to each voter voters **[PI]**. Mixing ensures that plaintext tracker numbers cannot be associated with a voter.
5. Generate tracker number commitments for all tellers in the mix using each voter's public key **[the demonstrator will generate key pairs for all voters]** and publish **[DL]**.

Voting

During voting, each voter casts their vote, which is encrypted and signed. In the demonstrator, to minimise disruption to the existing ERS system, and in particular because of vote cancellation, encryption of votes will occur after voting has closed and will be performed by VE, with data published via DI to the DL.

1. Voter casts their signed and encrypted vote, which is only accepted if there is not a duplicate entry **[VE]**.
2. Encrypted votes are published **[DI]** to the **[DL]**, but only made public once the election has closed.

Mixing and Decryption

Mixing of encrypted tracker numbers and vote pairs ensures that, when decrypted, no plaintext vote can be associated with a voter. This is performed by VA, with data published via DI to the DL.

1. For each vote, the encrypted tracker number and encrypted vote are extracted and shuffled to anonymise the voter **[VA]**.
2. A threshold set of tellers decrypt the tracker number and vote pairs **[in the demonstrator we have a threshold of four]** which are published **[DI]** to the **[DL]**.
3. Voters are sent the information needed to reveal their tracker number using their public key **[ERS]**.
4. Voters obtain their tracker number from an α and β pair, and use the tracker number to look-up their vote **[DI]**, *or generate new information needed to obtain a different tracker number in the case of coercion* ***[in the demonstrator, the generation of voter verification parameters for a different tracker number in the case of coercion is not being implemented].***

DISTRIBUTED LEDGER

The distributed ledger in the demonstrator provides the append-only data store required by Selene. The DL technology to be used in the demonstrator will be Quorum [11]. This is a permissioned platform based upon Ethereum [12], which enables proof-of-authority nodes to be used in the consensus algorithm.

ARCHITECTURE

The **ERS system** is a separate software solution which will be modified to integrate the required verifiability functions, interfacing with the other components in the demonstrator via CSV files. Since the PI, VE and VA are the only components which require the use of cryptographic operations within the Selene scheme, and will be operating as standalone elements within the secure network on private data, it is sensible to group these functions into a single **Private VMV** software program which can be run from the command line with the corresponding input CSV files. In contrast, the DI component is intended to provide both public and private access to the DL, allowing administrators to publish data and voters to look up data. To achieve this, it must run a web user interface and this **Public VMV** will therefore be a separate software service running in the cloud. Finally, the DL consists of third-party software deployed as a series of communicating nodes held securely at different premises and accessible from the DI. The demonstrator therefore consists of four software elements as shown in Figure 2, together with their interactions.

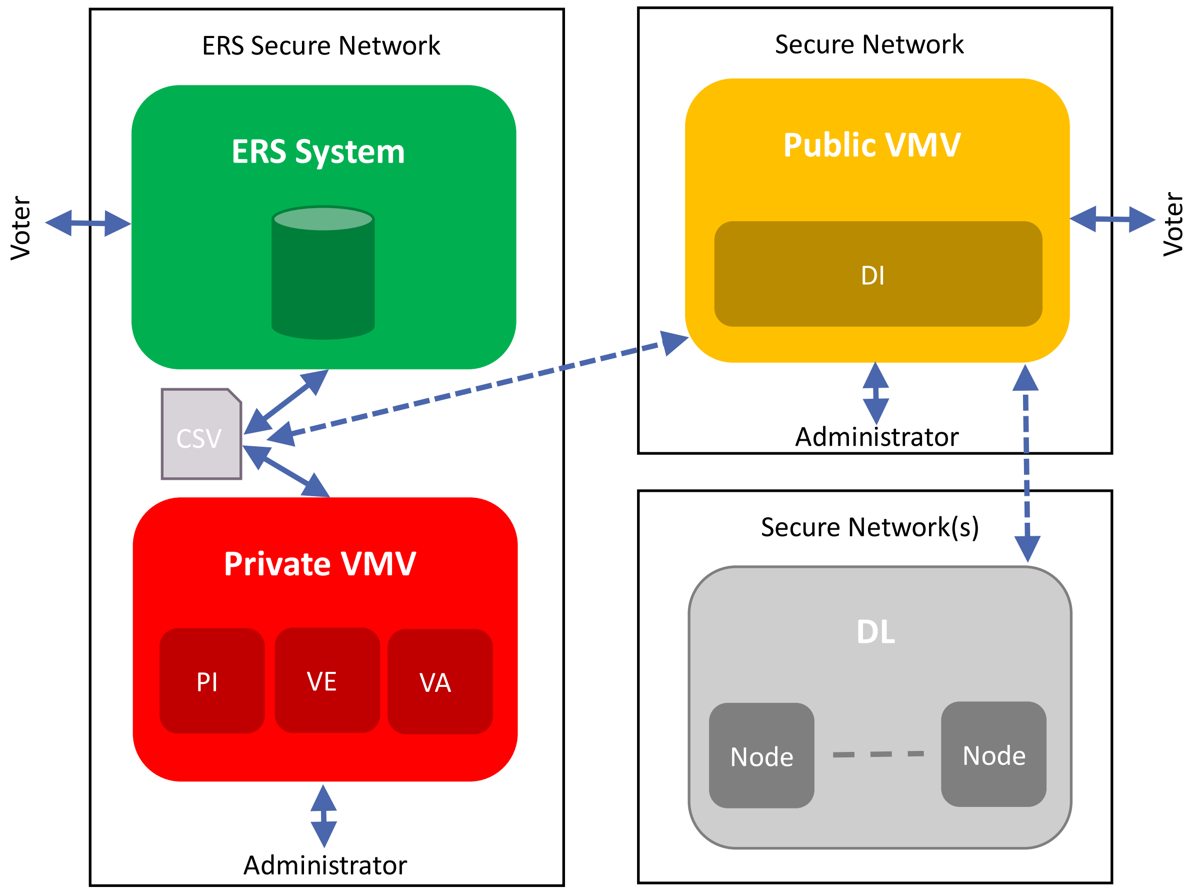

FIGURE 2: DEMONSTRATOR SOFTWARE ARCHITECTURE

ERS SYSTEM

As existing software, the ERS system will be modified to include the required functions and data stores. Except for storage of the overall election verification parameters, changes for data storage are anticipated to only require additional fields in the vote results table, including fields for the encrypted tracker number as an α and β pair, a voter's public encryption key, a voter's private signing key and the voter's encrypted vote. The overall election verification parameters can be held securely as a separate file.

PRIVATE VMV

Although computers running on the ERS secure network run Microsoft software, in order to ease development and provide some level of confidence that the developed programs will run on unknown hardware (or virtual machines), the Private VMV will be developed using Java [13].

Java provides portability, while third-party libraries, such as Bouncy Castle [14], provide industry-proven implementations of cryptographic functions. To further ease development, the Spring [15] framework will be used, with the program deployed within a Maven-built JAR file [16]. Spring enables the rapid development of Java code using dependency injection to promote loose coupling of classes. With Spring Boot [17], a Maven-built JAR file contains all required third-party dependencies for an application, which can be run simply from the command line using Java. One important note for cryptographic operations with Java is the previous use of limited key lengths (for US export laws). For any installation of Java prior to version 9, to use 'unlimited' strength cryptography, the Java installation needs modification (cf. [18]). From Java 9 onwards, 'unlimited' strength policies are installed and used by default. The demonstrator will require the use of Java's 'unlimited' strength policies.

By combining the PI, VE and VA components together in the single Private VMV program, common operations can be shared between each component, such as (creating or) loading and using the overall election verification parameters. To enable the different component operations to be selected from the command line, Spring Shell [19] will be used to support command line operation of the Java program.

Shuffling of votes will be achieved using Verificatum [20]. This enables the shuffling of votes using a number of independent teller nodes. The demonstrator will therefore consist of (at least) four installations of Verificatum nodes to enable shuffling. The VA will then interface with Verificatum to complete the shuffle.

It is anticipated that separate Verificatum nodes will be run by ERS, the University of Surrey and King's College London, depending upon available hardware (or virtual machines). For example, a node can be run on an AWS instance such that the node is protected behind a firewall that will only permit access by administrators and the Private VMV (say, via whitelisted IP addresses).

PUBLIC VMV

The Public VMV provides a web service to enable the publishing of data to the DL and includes a user interface to enable the browsing and searching of the published data, such as looking up a vote using an α and β pair.

The Public VMV is therefore a public-facing service, running in the cloud, which can be accessed via an Internet address for voter and administrator functions. Administrator functions, such as publishing data, will be secured to enable authorised access only.

To enable rapid development the Public VMV will be written in Ruby [21] using the Ruby on Rails framework [22]. Ruby on Rails provides support for a model-view-controller (MVC) development which lends itself well to the development of web services and web user interfaces using RESTful requests. Alongside this, Ruby on Rails also provides support for authentication and security either built-in or via third-party plug-ins (known as gems). Ruby on Rails applications can be easily deployed to cloud infrastructure, such as Amazon Web Services (AWS) [23] using Elastic Beanstalk [24].

DISTRIBUTED LEDGER

The DL will run Quorum nodes, which provide an interface which can be used by websites or via JavaScript to execute smart contracts within the blockchain. The demonstrator will therefore consist of (at least) four installations of Quorum with suitable configuration and smart contracts to enable proof-of-authority to execute transactions on the blockchain. The DL will then interface to the blockchain to publish data, while also using the interface to allow the content of the blockchain to be browsed or searched.

Similar to Verificatum nodes, it is anticipated that separate Quorum nodes will be run by ERS, the University of Surrey and King's College London. Quorum nodes will only allow access for administrators and the Public VMV.

ERS SYSTEM

Requirements:	EI1, EI2, EI3, EI4, EI5, EI8, EI9, ER1, ER2, ER3, ER4, ER5, ER6, ER7, ER8, ER9, ER10, ER11, ER12, ER13, OP2, OP4
Interfaces:	Existing ERS system and user interfaces; CSV files

The ERS system will be modified to enable the operations of the PI, VE and VA to be performed.

To support the operations of the PI, the ERS system shall provide a means to export voter information **[EI1, ER1]**, which will include the unique identifier for each voter **[ER2]**. Once the PI has completed generating the verification parameters, the ERS system will be able to import the overall election verification parameters **[EI2]** from a CSV file and store them **[ER3]**. This could be just the storage of the CSV file, as the parameters themselves are not used by the ERS system. Similarly, the system will be able to import voter verification parameters **[EI3]** and store these **[ER5]**. It is anticipated that these voter verification parameters will be stored as columns within the database table that is currently used to store a voter's security credentials and plaintext vote (once cast). Both the overall election and voter verification parameters will be stored persistently **[OP2]** with appropriate back-up **[OP4]**.

To support the operations of the VE and VA, the ERS system will be able to export **[EI4]** the overall election verification parameters **[ER4]** and voter verification parameters **[ER6]** which were generated by the PI and stored by the ERS system. For the VE, exporting of the cast plaintext votes will also be supported from the database table **[EI5, ER12]**, and this will include the unique identifier for each voter, verification parameters and cancellation flag **[ER13]**.

Via the CMS, when a voter logs into the system to vote, they will be presented with an option to register for vote verification **[EI8, ER7]**. The system will record the voter's registration when casting their vote, if selected, and follow this up with an email **[EI9, ER8]** with the voter's verification parameters. During the election, these parameters will only be those elements which the voter is allowed to have prior to the election period finishing (in Selene, this is the β) **[ER9]**. After the election, another email will be sent to all those registered for verification which will include all of the required verification parameters (in Selene, the α and β, but this will also include the tracker number) **[ER11]**, together with instructions on how to verify their vote **[ER10]**.

PRIVATE VMV

Requirements:	EI1, EI2, EI3, EI4, EI5, EI6, EI7, EI10, PI1, PI2, PI3, VE1, VE2, VE3, VE4, VA1, VA2, VA3, VA4, OP1
Interfaces:	Spring Shell commands and parameters; CSV files

The Private VMV will be a built as a Java [13] command line program, using Spring Boot [17] and Spring Shell [19] to provide a single Java archive which can be installed **[OP1]** and executed with appropriate commands on a computer (or virtual machine) within the ERS secure network which has access to the required private data **[EI10]** or externally to generate voter key pairs. For example, to obtain help on the available command line parameters, run the application JAR file using:

```
java -jar private-vmv.jar
```

and then obtain a list of the available commands using

```
shell:> help
```

where "`shell:>`" is the Spring Shell prompt.

VERIFICATION PARAMETER INITIALISATION (PI)

The functions of the PI allow the verification parameters to be generated and will be run in two stages when the election is first set-up.

Externally to ERS

To create and export a new set of overall election verification parameters **[EI2, PI2]**, enter the following shell command:

```
shell:> create-election-parameters
        --publish public-election-params.csv
        --name <name>
        --no-tellers |
        --number-of-tellers <tellers> --threshold-tellers <threshold>
        --dsa-l 3072 --dsa-n 256 --prime-certainty 128
```

This will create the parameters and export them to the "`public-election-params.csv`" CSV file, which can be published to the DL. The "`<name>`" of the election, the number of "`<tellers>`" and the "`<threshold>`" number of tellers are used to initialise the tracker number and vote mix tellers. If "`no-tellers`" is supplied, then no tellers are used (no Verificatum). The "`dsa-l`", "`dsa-n`" and "`prime-certainty`" parameters are used to control the strength of the generated keys.

Once the election parameters have been defined, each mix-net teller can be initialised using the following commands:

```
shell:> create-teller
        --election public-election-params.csv
        --teller <teller>
        --ip <ip>
        --teller-port <teller-port>
        --hint-port <hint-port>
        --publish teller-information-<teller>.xml
```

This will use the previously generated public election parameters "`public-election-params.csv`" CSV file to create the local teller parameters. The "`<teller>`" number is a unique integer from 1 to the total number of tellers inclusive. The "`<ip>`" address is the IP address (or DNS name) of the host at which the teller can be contacted, with "`<teller-port>`" the main port number for the teller and "`<hint-port>`" the hint port number used to check if the teller is available (see [20] for details). The output "`teller-information-<teller>.xml`" XML file contains the Verificatum parameters for the teller which should be shared with all other tellers.

Once a teller has been created and all the output files shared with each teller, these teller information files need to be merged on each teller using:

```
shell:> merge-teller
        --election public-election-params.csv
        --teller <teller>
        --teller-information teller-information-1.xml ...
```

For teller "`<teller>`" and using the previously generated public election parameters "`public-election-params.csv`" CSV file, this will merge all of the teller information files starting with "`teller-information-1.xml`" up to "`teller-information-n.xml`", where "`n`" is the total number of tellers. The files should be given in the correct order. This command will fail if there are insufficient teller information files supplied.

To create and export the election keys using Verificatum **[EI2, PI2]**, enter the following shell command:

```
shell:> create-election-keys
        --election public-election-params.csv
        --teller <teller>
        --output election-keys.csv
        --publish public-election-keys.csv
```

For teller "`<teller>`" and using the previously generated public election parameters "`public-election-params.csv`" CSV file to create the election keys and export the private (if available) and public key to "`election-keys.csv`", and the public key, for publication to the DL, to the "`public-election-keys.csv`" CSV file.

To create and export the voter keys **[EI3, PI3]**, the following shell commands will be used:

```
shell:> create-voters-keys
        --election public-election-params.csv
        --number-of-voters <number>
        --output voters-keys.csv
        --publish public-voters-keys.csv
```

This will use the previously generated public election parameters "`public-election-params.csv`" CSV file and the number of voters "`<number>`" to create the private and public aspects of the voter verification parameters and export them to the "`voters-keys.csv`" CSV file. The public elements of the voter verification parameters which will be stored by ERS and published to the DL will be exported to the "`public-voters-keys.csv`" CSV file.

If voters are creating and holding their own key pairs, then the following command can be used by each voter to create their encryption and signature keys:

```
shell:> voter-create-keys
        --election public-election-params.csv
        --output voter-keys.csv
        --publish public-voter-keys.csv
```

This will use the previously generated public election parameters "`public-election-params.csv`" CSV file to create the private and public aspects of the voter verification parameters for a single voter and export them to the "`voter-keys.csv`" CSV file. The public elements of the voter verification parameters which can published to the DL will be exported to the "`public-voter-keys.csv`" CSV file.

Once the number of voters is known, then the tracker numbers can be created:

```
shell:> create-tracker-numbers
        --election public-election-params.csv public-election-keys.csv
        --number-of-voters <number>
        --publish public-tracker-numbers.csv
```

This will use the previously generated public election parameters "`public-election-params.csv`" and keys "`public-election-keys.csv`" CSV files and the number of voters "`<number>`" to create the tracker numbers and export them to the "`public-tracker-numbers.csv`" CSV file. Since the tracker numbers are not associated with voters and the encrypted tracker numbers will be re-encrypted, the plaintext and encrypted tracker numbers can be published to the DL.

Before tracker numbers can be associated with voters, they need to be shuffled such that the original plaintext tracker number cannot be associated with a voter, who is associated only with an encrypted tracker number. To do this, run:

```
shell:> shuffle-tracker-numbers
      --election public-election-params.csv
      --teller <teller>
      --tracker-numbers public-tracker-numbers.csv
      --publish shuffled-tracker-numbers.csv shuffle-proofs-<teller>.zip
```

For teller "`<teller>`" and using the previously generated **[EI4, EI6]** public election parameters "`public-election-params.csv`" CSV file and the public tracker numbers "`public-tracker-numbers.csv`" CSV file to create the list of shuffled tracker numbers and export them to the "`shuffled-tracker-numbers.csv`". This file and the "`shuffle-proofs-<teller>.zip`" file can be published to the DL.

Once shuffled, encrypted tracker numbers are associated with voter encryption public keys and corresponding encrypted commitment values are generated. These commitment values will be used by the voter to verify their vote. To create the encrypted commitment values, run:

```
shell:> create-commitments
      --election public-election-params.csv public-election-keys.csv
      --teller <teller>
      --voters public-voters-keys.csv
      --tracker-numbers shuffled-tracker-numbers.csv
      --output commitments-<teller>.csv
      --publish public-commitments-<teller>.csv
                commitments-proofs-<teller>.csv
```

For teller "`<teller>`" and using the previously generated **[EI4, EI6]** public election parameters "`public-election-params.csv`" and keys "`public-election-keys.csv`" CSV files, the voter parameters "`public-voters-keys.csv`" CSV file and the shuffled tracker numbers "`shuffled-tracker-numbers.csv`" CSV file to create the list of shuffled tracker numbers and their beta commitments allocated to a voter key pair and export them to the "`commitments-<teller>.csv`" and "`public-commitments-<teller>.csv`" CSV files, where "`<teller>`" should be substituted with the teller number. These files write the commitments in the same order as the provided list of voters. The "`public-commitments-<teller>.csv`" and "`commitments-proofs-<teller>.csv`" files may be published.

The final stage of creating all of the required data is to decrypt the commitments. To do this, run:

```
shell:> decrypt-commitments
        --election public-election-params.csv election-keys.csv
        --voters public-voters-keys.csv
        --tracker-numbers shuffled-tracker-numbers.csv
        --commitments public-commitments-1.csv ...
        --publish public-voters.csv decrypt-proofs-<teller>.zip
```

This will use the previously generated **[EI4, EI6]** public election parameters "`public-election-params.csv`" and public and private keys "`election-keys.csv`" CSV files, the public voter parameters "`public-voters-keys.csv`" CSV file, the shuffled tracker numbers "`shuffled-tracker-numbers.csv`" CSV file and the previously generated encrypted commitments "`public-commitments-<teller>.csv`" for every "`<teller>`" to decrypt the commitments and create the consolidated list of public voter data "`public-voters.csv`" CSV file. The "`public-voters.csv`" and "`decrypt-proofs-<teller>.zip`" may be published.

Within the ERS Secure Network

To associate the create parameters with voters:

```
shell:> associate-voters
        --election public-election-params.csv public-election-keys.csv
        --voters public-voters.csv ers-voters.csv
        --output ers-associated-voters.csv
        --publish public-associated-voters.csv
```

This will use the previously generated public election parameters "`public-election-params.csv`" and keys "`public-election-keys.csv`" CSV files, the pre-allocated list of voters with encrypted tracker numbers and commitments "`public-voters.csv`", and the voter data "`ers-voters.csv`" CSV file **[EI1]** exported from the ERS system **[PI1]**, and associate each voter with their keys, encrypted tracker number and commitments. The private and public parameters of the association are exported to the "`ers-associated-voters.csv`" CSV file for import into the ERS system, while the public parameters are exported to the "`public-associated-voters.csv`" CSV file, which can be published to the DL.

Optionally, to map the arbitrary, plaintext vote options to option numbers in the group:

```
shell:> map-vote-options
        --election public-election-params.csv
        --vote-options ers-vote-options.csv
        --publish public-vote-options.csv
```

This will use the previously generated public election parameters "`public-election-params.csv`" and the supplied list of vote options "`ers-vote-options.csv`" CSV files. Each vote option will be mapped to an option number in the election parameter group. The resulting "`public-voter-options.csv`" CSV file can be published to the DL.

VOTE ENCRYPTION (VE)

The function of the VE is to encrypt and sign the plaintext votes using the pre-created parameters. During this stage, the alpha part of the commitment is also created and associated with the voter. Each encrypted vote is also checked to ensure uniqueness as required by Selene.

To encrypt and sign and export the encrypted tracker number and encrypted vote pairs **[EI6, VE4]**, enter the following shell command:

```
shell:> encrypt-votes
        --election public-election-params.csv public-election-keys.csv
        --voters  voters-keys.csv ers-plaintext-voters.csv
                  encrypt-proofs.csv
        --votes ers-vote-options.csv
        --commitments commitments-1.csv ...
        --output ers-encrypted-voters.csv
        --publish public-encrypted-voters.csv public-vote-options.csv
                            encrypt-proofs.csv
```

This will use the previously generated **[EI4, EI5]** public election parameters "`public-election-params.csv`" and keys "`public-election-keys.csv`" CSV files **[VE1]**, the private and public voter parameters "`voters-keys.csv`", the voter data with their cast votes "`ers-plaintext-voters.csv`" and "`ers-encrypt-proofs.csv`" exported from the ERS system, vote options "`ers-vote-options.csv`" and private teller commitment data "`commitments-<teller>.csv`" CSV files **[VE2, VE3]** to create the encrypted and signed votes and export them to the "`ers-encrypted-voters.csv`" CSV file, which contains the voter data as well as the encrypted votes which can be re-imported into ERS. The "`voters-keys.csv`" or "`ers-encrypt-proofs.csv`" files are optional depending upon whether votes are encrypted externally. The "`public-encrypted-voters.csv`", "`public—voter-options.csv`" and "`encrypt-proofs.csv`" CSV files can be published to the DL.

If externally created voter keys are being used, then alternatively to the above, a voter can encrypt their vote using the following command:

```
shell:> voter-encrypt-vote
         "<vote>"
         --election public-election-params.csv public-election-keys.csv
         --voter voter-keys.csv
         --votes public-vote-options.csv
         --publish public-encrypted-voter.csv encrypt-proof.csv
```

This will use the previously generated **[EI4, EI5]** public election parameters "`public-election-params.csv`" and keys "`public-election-keys.csv`" CSV files **[VE1]**, the private and public voter parameters for the voter "`voter-keys.csv`" and the vote options "`public-vote-options.csv`" CSV files **[VE2, VE3]** to encrypt the "`<vote>`" and sign it. The encrypted vote and its signature are exported to the "`public-encrypted-voter.csv`" CSV file, which contains just the encrypted vote, its signature and the voter's public keys. The "`ers-encrypt-proof.csv`" file contains the proof of encryption. All of these output files can be published, and the "`public-encrypted-voter.csv`" CSV file should be aggregated together with all other encrypted votes for vote anonymisation and decryption.

VOTE ANONYMISATION AND DECRYPTION (VA)

The function of the VA is to anonymise and decrypt the encrypted votes using the pre-created parameters. The anonymisation process follows the shuffle defined for Selene and is provided as a separate command to allow the shuffle to be run on or off site from ERS. Typically the shuffle will be run off site, whereas a verification of the produced data must be run from within the ERS network.

Externally to ERS

To shuffle the encrypted votes and decrypt them **[EI7, VA4]**, enter the following shell command:

```
shell:> mix-votes
         --election public-election-params.csv election-keys.csv
         --teller <teller>
         --votes public-vote-options.csv
         --tracker-numbers public-tracker-numbers.csv
         --voters public-encrypted-voters.csv
         --publish public-mixed-voters.csv mix-proofs-<teller>.zip
```

For teller "`<teller>`" and using the previously generated **[EI4, EI6]** public election parameters "`public-election-params.csv`" and keys "`election-keys.csv`" CSV files **[VA1]**, the "`public-vote-options.csv`" and "`public-tracker-numbers.csv`" files, and the encrypted votes "`public-encrypted-votes.csv`" CSV file **[VA2, VA3]** generated by the VE to create the list of shuffled and decrypted votes, and save them to the "`public-mixed-voters.csv`" CSV file. This latter file and the "`mix-proofs-<teller>.zip`" file can be published to the DL.

Within the ERS Secure Network

To verify that one or more voter's data has been published correctly, a voter's alpha and beta commitments can be decrypted using the voter's private encryption key to check that the corresponding tracker number and plaintext vote are correct. This can be achieved using the following shell command:

```
shell:> voter-decrypt-tracker-number
         --election public-election-params.csv
         --alpha <alpha> --beta <beta> --public-key <public-key>
```

```
        --tracker-numbers public-tracker-numbers.csv
        --voters voters-keys.csv
```

For the selected "`<alpha>`", "`<beta>`" and "`<public-key>`" for a voter, this will use the previously generated public election parameters "`public-election-params.csv`", the private and public voter parameters "`voters-keys.csv`" and the published tracker numbers "`public-tracker-numbers.csv`" CSV files to find the voter's corresponding private encryption key, decrypt their alpha and beta to obtain the tracker number in the group, and match this to the voter's tracker, which will be output.

PUBLIC VMV

Requirements: EI3, EI6, EI7, EI11, EI12, DI1, DI2, DI3, DI4, DI5, DI6, DI7, DI8, DI9, DI10, OP1, OP5, DC1, SS1, SS2

Interfaces: Receive administrator and voter RESTful requests; Send RESTful requests to DL

The Public VMV will be a built as a Ruby on Rails [22] web application which can be installed **[OP1]** and executed on a server (or virtual machine) to provide a web interface via a public Internet interface **[OP5]**.

The service will provide a web user interface serving HTML, CSS [6] and JavaScript [5] web pages **[DC1]**. All pages will be served using an SSL certificate **[SS2]**. Public pages will be provided to give a home page ('**/**') and about page ('**/about**') to allow visiting users, such as voters **[EI12]**, to obtain further information about the project and the verification.

From these public pages, an administrator will be able to login ('**/login**') using a username and password to access administrator-only functions **[SS1]**. These pages will include an overall configuration page which will be used to set service options, such as whether voters can look-up votes once the election has finished (and after the anonymised and decrypted votes have been published), and pages for administrators to publish data **[DI1]**.

DISTRIBUTED LEDGER INTERFACE (DI)

The DI functions enable administrators to publish election information to the DL, and for voters to browse and search published information from the DL **[DI10, EI11]**.

An authorised administrator will be able to upload the publishable elements for an election's data and voter verification parameters for publication to the DL. This includes data for the election parameters **[DI2, DI3]** output by the PI component **[EI3]**, encrypted votes **[DI4, DI5]** output from the VE component **[EI6]**, and anonymised and decrypted tracker number and plaintext vote pairs **[DI6, DI7]** output from the VA component **[EI7]**.

Once enabled by an administrator, public pages will be made available to allow voters to browse through the published election data **[DI8]**, and to search for their plaintext vote **[DI9]**. An administrator shall be able to enable or disable public visibility of the data. However, this will not affect what is published on the DL, only the accessibility of the data via the DI. This will enable administrators to check the data prior to making it visible.

When a voter has entered their α and β pair to obtain their tracker number, and subsequently verified their vote, they will be prompted to take part in an electronic survey. The survey will be developed and maintained externally to the DI using, for example, SurveyMonkey [25].

Note that while we use the same endpoints for both voter browsing and administrator publishing of DL data, the operations performed by the endpoint will depend upon user authentication and the associated RESTful verb for the request. For example, when available, a voter may browse the decrypted votes by sending a GET request to the associated endpoint, while an authenticated administrator can send a POST request to the send endpoint to publish data to the DL.

DISTRIBUTED LEDGER

Requirements:	DI1, DI8, OP1, OP3, OP4
Interfaces:	Quorum interface

The DL will be implemented using (at least) four Quorum nodes so that data can be published **[DI1]** or retrieved **[DI8]** by the DI. Each node will be installed to appropriate hardware (or virtual machine) **[OP1]** with access granted only to the DI and administrators, and storage which is persistent **[OP3]** and backed-up **[OP4]**.

BIBLIOGRAPHY

[1] EPSRC, "Trusted and Transparent Voting Systems," 2017. [Online]. Available: http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/P031811/1. [Accessed June 2018].

[2] University of Surrey, "Verify My Vote (VMV): Requirements Specification," 2018.

[3] IEEE Computer Society, "ISO/IEC/IEEE 29148:2011 Systems and software engineering -- Life cycle processes -- Requirements engineering," 2011. [Online]. Available: https://www.iso.org/standard/45171.html. [Accessed September 2017].

[4] IEEE Computer Society, "IEEE Standard for Information Technology — Systems Design — Software Design Descriptions," 2009. [Online]. Available: http://standards.ieee.org/findstds/standard/1016-2009.html. [Accessed September 2017].

[5] W3C, "JavaScript Web APIs - W3C," 2016. [Online]. Available: https://www.w3.org/standards/webdesign/script.html. [Accessed February 2018].

[6] W3C, "HTML & CSS - W3C," 2016. [Online]. Available: https://www.w3.org/standards/webdesign/htmlcss. [Accessed February 2018].

[7] Google, Inc., "Chrome Web Browser," 2017. [Online]. Available: https://www.google.com/chrome/. [Accessed September 2017].

[8] Apple, Inc., "macOS - Safari - Apple (UK)," 2017. [Online]. Available: https://www.apple.com/uk/safari/. [Accessed September 2017].

[9] Mozilla, 2017. [Online]. Available: https://www.mozilla.org/en-US/firefox/. [Accessed September 2017].

[10] P. Ryan, P. Roenne and V. Iovino, "Selene: Voting with Transparent Verifiability and Coercion-Mitigation," in *Abstract book of 1st Workshop on Advances in Secure Electronic Voting*, Springer, 2015.

[11] JPMorgan Chase & Co., "Quorum | J.P. Morgan," 2018. [Online]. Available: https://www.jpmorgan.com/global/Quorum. [Accessed November 2018].

[12] Ethereum Foundation, "Ethereum Project," 2018. [Online]. Available: https://www.ethereum.org/. [Accessed June 2018].

[13] Oracle, "java.com: Java + You," 2018. [Online]. Available: https://java.com/. [Accessed February 2018].

[14] Legion of the Bouncy Castle Inc., "bouncycastle.org," 2013. [Online]. Available: https://www.bouncycastle.org/. [Accessed February 2018].

[15] Pivotal Software, Inc., "Spring," 2018. [Online]. Available: https://spring.io/. [Accessed February 2018].

[16] The Apache Software Foundation, "Maven - Welcome to Apache Maven," 2018. [Online]. Available: https://maven.apache.org/. [Accessed June 2018].

[17] Pivotal Software, Inc., "Spring Boot," 2018. [Online]. Available: https://projects.spring.io/spring-boot/. [Accessed February 2018].

[18] Oracle, "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for JDK/JRE 8 Download," 2018. [Online]. Available: Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files. [Accessed June 2018].

[19] Pivotal Software, Inc., "Spring Shell," 2018. [Online]. Available: https://projects.spring.io/spring-shell/. [Accessed June 2018].

[20] D. Wikström, "https://www.verificatum.com/," 2018. [Online]. Available: https://www.verificatum.com/. [Accessed November 2018].

[21] Ruby Community, "Ruby Programming Language," 2019. [Online]. Available: https://www.ruby-lang.org/en/. [Accessed January 2019].

[22] Ruby on Rails, "Ruby on Rails | A web-application framework that includes everything needed to create database-backed web applications according to the Model-View-Controller (MVC) pattern.," 2019. [Online]. Available: http://rubyonrails.org/. [Accessed January 2019].

[23] Amazon Web Services, Inc., "Amazon Web Services (AWS) - Cloud Computing Services," 2018. [Online]. Available: https://aws.amazon.com/. [Accessed June 2018].

[24] Amazon Web Services, Inc., "AWS Elastic Beanstalk – Deploy Web Applications," 2017. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/. [Accessed September 2017].

[25] SurveyMonkey, "SurveyMonkey: The UK's Most Popular Free Online Survey Tool," 2018. [Online]. Available: https://www.surveymonkey.co.uk/. [Accessed July 2018].