VERSION 15

13 DECEMBER 2019

TRUSTED AND TRANSPARENT VOTING SYSTEMS

VERIFY MY VOTE DEMONSTRATOR USER MANUAL

PRESENTED BY: MATTHEW CASEY

PERVASIVE INTELLIGENCE LTD

38 TAVISTOCK ROAD, FLEET, HAMPSHIRE, GU51 4EJ

DOCUMENT INFORMATION

STATUS

Distribution	Public
Status	Published

VERSION

Version	Author	Date	Comment
1	Matthew Casey	30 November 2018	Initial draft with details of the parameter initialisation commands, inputs and outputs.
2	Matthew Casey	17 December 2018	Definition of vote encryption and vote anonymisation and decryption commands.
3	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.
4	Matthew Casey	16 January 2019	Modified for the inclusion of proofs of knowledge for commitment values.
5	Matthew Casey	18 January 2019	Indicate that all teller proofs should be published independently. Add proof of knowledge to vote encryption.
6	Matthew Casey	20 February 2019	Updated to include distributed ledger interface component.
7	Matthew Casey	12 March 2019	Updated Public VMV description and images after changes resulting from review.
8	Matthew Casey	2 April 2019	Updated layout and verification pages.
9	Matthew Casey	8 April 2019	Updated from final review by Surrey.
10	Matthew Casey	25 April 2019	Updated administration screen for election list to allow access to filtered uploads.
11	Matthew Casey	29 April 2019	Added Luxembourg as a partner.

Version	Author	Date	Comment
12	Matthew Casey	21 May 2019	Revised following trial review with revised verify vote pages, new vote status page, updated plaintext proof of knowledge and new scripting.
13	Matthew Casey	4 June 2019	Correct scripting after first use.
14	Matthew Casey	12 December 2019	Modified to optionally allow voter's keys to be pre-defined and not created during parameter initialisation. Includes the addition of the voter's public trapdoor key within a commitment, the optional provision of the public trapdoor key during association of voters and supply of externally encrypted votes.
15	Matthew Casey	13 December 2019	Added voter commands to allow separate creation of voter keys, vote encryption and commitment decryption.

GLOSSARY

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. The first row is typically used to provide column names.
Digital Signature Algorithm (DSA)	A cryptographic algorithm which is used to sign data so that it can be verified publicly. DSA signing requires a private key, with verification taking place using the corresponding public key.
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.
ElGamal Encryption	An encryption scheme which uses a public key to encrypt plaintext to a ciphertext which can only be decrypted by the corresponding private key.
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.
Multipurpose Internet Mail Extensions (MIME)	An identifier used to describe the format of data. For example, “text/csv” is associated with a text file which contains CSV data.
Mix-net	A distributed and independent set of computers which are used to shuffle and/or decrypt ciphertexts. The shuffling process is carried out sequentially on each mix-net node such that the list of input ciphertexts is randomised and then re-encrypted so that by the end of the mix, the input ciphertexts are randomly ordered and cannot be mapped back to their original order or values. When decrypting, the decryption occurs using a shared secret key on each mix-net node such that only a threshold of the available nodes can successfully perform the decryption.
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 (or secret) key.
Teller	In this document, a teller is a computer which is running as a mix-net node. Tellers are used to independently shuffle and decrypt ciphertexts.
Tracker Number	A unique number which (in encrypted form) is assigned to a voter to identify their 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.
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 ciphertext.
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 define the user manual 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 demonstrator provides a working implementation of important 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, and in collaboration with the University of Luxembourg.

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 describes how the demonstrator can be used to add verifiability to an election. As such it provides details of the sequence of actions required to operate the demonstrator alongside an ERS election. The use of the ERS system to perform voting will not be described in this document. Instead, the focus is on the user actions and commands that need to be executed using the demonstrator, together with their expected inputs, outputs and data formats, for the verifiable components of an election. In addition to each action, and where appropriate, the underlying cryptographic operations, will be described demonstrating what approaches have been taken to implement verifiability.

This document builds upon the requirements and design provided in [2].

OVERVIEW

The next section of this document provides an overview of the components used within the demonstrator. Each subsequent section of this document focuses on each major stage of the election: 1) election setup prior to the voting period, 2) voting, 3) vote encryption after the voting period, and 4) vote verification and data publication.

COMPONENT OVERVIEW

OVERVIEW

The demonstrator consists of five software components: 1) ERS system, 2) VMV command line, 3) mix-net, 4) VMV web service, and 5) distributed ledger (DL). The ERS system is an existing service which has been modified to support verifiability. The VMV command line tool is designed to run both within the ERS secure network and externally to setup the election parameters and process the resulting votes. Two stages of this command line tool require a cryptographically secure shuffle of encrypted data, and this will be performed by a mix-net. All resulting public data from the election

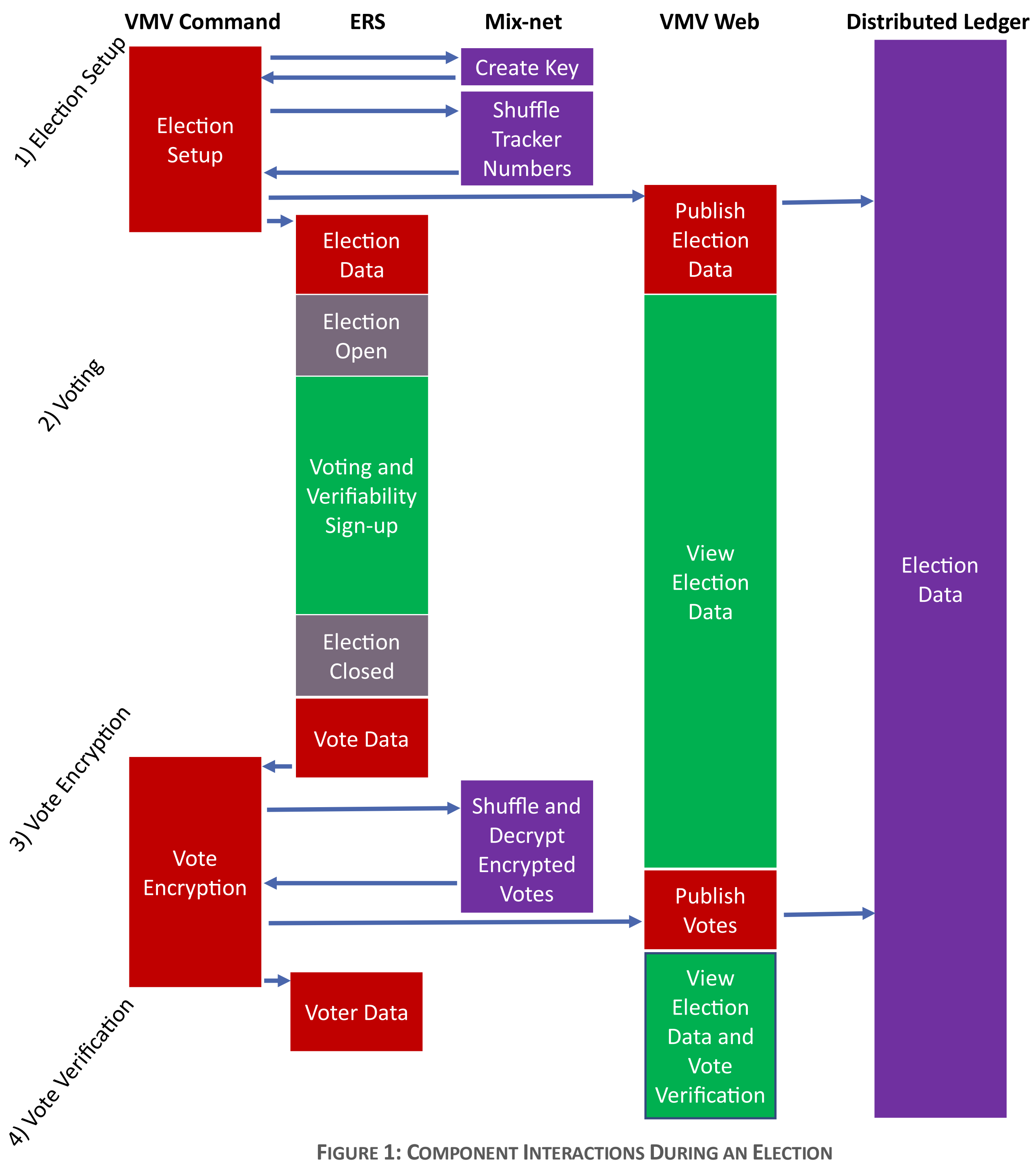


FIGURE 1: COMPONENT INTERACTIONS DURING AN ELECTION

will be published to the DL. This will be accessed via the VMV web service which provides public access to the published data.

The interaction between these components during an election is shown in Figure 1. Red boxes show operations performed by privileged administrators with access to secure data (either within the ERS secure network or on a separate secure network). Green boxes show components which are public-facing and hence can be accessed by any voter or interested member of the public. Purple components are distributed. Here, four mix-net nodes will be run by independent organisations. Similarly, for four DL nodes.

The combination of the VMV command line, mix-net, VMV web and DL support the implementation of the chosen cryptographic protocol which provides verifiability: Selene [3]. While the ERS system, VMV command line and VMV web are bespoke software components, the mix-net is implemented using the open-source Verificatum [4] system, while the DL uses Quorum [5].

SOFTWARE ENVIRONMENT

The software environment requirements for the bespoke software components are described below. Since the ERS system is out of scope of this document, it is not described here.

VMV COMMAND LINE

The VMV command line program is a Spring Boot [6] [7] application which requires Java 1.8 running on any suitable computer. This can either be provided by Oracle's implementation, or the Open JDK. In addition, since Java's built-in cryptography is being used, Java will need to be configured to use strong cryptography. Java 1.8 does not by default include the use of strong cryptographic keys, and hence this may need to be configured on the target computer using [8].

To run the VMV command line, at a command prompt execute:

```
java -jar vmv-x.x.x.jar
```

where "`x.x.x`" is replaced with the appropriate version number for the associated JAR file.

In addition, to use the scripts available to automate the initialisation of an election, and the encryption and mixing of the associated plaintext votes, the "expect" package needs to be installed.

MIX-NET

To reliably install and use the Verificatum mix-net, a 64-bit computer is needed, preferably running Linux. For example, a Debian [9] installation has worked reliably during testing. Verificatum must be built and installed on the target computer. This requires the `sudo`, `build-essential`, `m4`, `libgmp-dev` and `openjdk-8-jdk` packages installed. Once installed, Verificatum can be installed by following its installation instructions. Note that only the `gmpmee`, `vec`, `verificatum-vcr` and `verificatum-vmn` components are needed as a minimum. Depending upon the target computer, it may be necessary to configure their compilation using:

```
./configure CFLAGS='-std=c11 -Wno-sign-compare -Wno-newline-eof'
```

at the relevant installation stage.

The computer running the mix-net teller should be secured from public access since its filesystem will contain private key information. All data generated by the mix-net during operation should be retained and backed-up. If this data is lost on multiple tellers during the election, such that there is no longer a threshold of working tellers, then it will not be possible to complete the election verifiability processing because the mix will not be able to take place.

Using Verificatum, each teller listens for incoming requests on two TCP ports. The first port is used to communicate election data, while the second ‘hint’ port is used to determine if the teller is running. Access to these ports will need to be made available via the Internet, but it is suggested that access is restricted to a white list of IP addresses which includes all other mix-net tellers only, together with restricted `ssh` access to run commands.

Note that the VMV command line is used to setup and run a teller to ensure consistent parameterisation of the node during the required election stages. To achieve this, Verificatum must have been successfully installed with all of its commands available from the command prompt’s path information.

VMV WEB SERVICE

The VMV web service is a Ruby-on-Rails [10] 5.2.2 application. While Ruby applications can be run on a variety of different operating systems, a 64-bit computer running Linux is preferred. This will require Ruby version 2.3.3 or above and associated software components which are automatically installed when the Ruby-on-Rails application is deployed.

In addition to the base computer, the web service also requires access to a PostgreSQL [11] relational database (version 9.6 or higher), cloud storage and email. To access these services the Ruby-on-Rails credentials file should be edited to provide the necessary configuration parameters for each item. The credentials file holds all of the required application configuration. Since it contains sensitive information, such as the base secret for cookie encryption, etc., this file is protected and can only be edited when the master key file is available. The master key file is not part of the repository.

While the web service can be deployed to any suitable cloud architecture, it is assumed that it will be deployed to Amazon’s Web Services (AWS) [12]. This includes tools for the provisioning of the base computer using, for example, Elastic Beanstalk [13], relational database using RDS [14], cloud storage S3 [15] and email [16].

DISTRIBUTED LEDGER

Each distributed ledger node will have a suitably configured Quorum [5] installation. Quorum can be run using a number of different configurations, including Docker [17] which has proven to be the easiest way to install and run Quorum.

ELECTION SETUP

OVERVIEW

The election setup stage initialises all of the data needed for the election, including encryption parameters, and encryption and signing keys. All private key data is created on a computer running on a secure network or on a mix-net teller, whereas ERS is provided with public key data only.

To create the election data, the following steps are required on a secure, independent computer outside of the ERS network and on each teller:

1. Create election parameters **[on an independent computer]**
2. Create tellers **[on each teller]**
3. Create election keys using the mix-net **[on each teller]**
4. Create voter keys for the required number of voters **[on an independent computer]**
5. Create tracker numbers for the required number of voters **[on an independent computer]**
6. Shuffle the encrypted tracker numbers using the mix-net **[on each teller]**
7. Create the tracker number commitments **[on each teller]**
8. Decrypt the tracker number β commitments using the mix-net **[on each teller]**

Once these steps have been completed, the public election parameters, public election key, public voter keys and encrypted and shuffled tracker numbers with β commitments are supplied to ERS. The following stages are then run on the ERS network:

9. Export voter identifiers from the ERS system **[on an ERS computer]**
10. Associate voter identifiers with voter public keys and encrypted tracker numbers and their β commitments **[on an ERS computer]**
11. Import the associated data into the ERS system **[on an ERS computer]**

The following optional stages may also be run if all vote options are known in advance and are required to be mapped for encryption:

12. Export vote options from the ERS system **[on an ERS computer]**
13. Map vote options **[on an ERS computer]**
14. Import the vote options into the ERS system **[on an ERS computer]**

Once each relevant stage has been completed, the corresponding public data can be published to the DL.

Steps 1) through 11) can be automatically completed using the “election_initialisation.exp” script, together with its associated “election_common.exp” script. These scripts need to be downloaded to each teller, where the first teller is assumed to take the role of the independent computer, ERS computer and teller. The initialisation can then be completed by running the script on each teller, as follows.

On the first teller:

```
election_initialisation.exp <vmv_jar_file>
  <ssh_key_file> <sftp_host> <sftp_user>
  <election_name>
  <number_of_tellers> <threshold_tellers>
  <teller> <teller_ip> <teller_main_port> <teller_hint_port>
  <number_of_voters> <ers_voters_file> <ers_associated_voters_file>
```

On every other teller:

```
election_initialisation.exp <vmv_jar_file>
  <ssh_key_file> <sftp_host> <sftp_user>
  <election_name>
```

```
    <number_of_tellers> <threshold_tellers>
    <teller> <teller_ip> <teller_main_port> <teller_hint_port>
```

where

`vmv_jar_file`	Is the full path to the VMV JAR file.
`ssh_key_file`	Is the full path to the SFTP key file needed for the SFTP server.
`sftp_host`	Is the SFTP host domain name or IP address.
`sftp_user`	Is the SFTP user name.
`election_name`	Is the (unique) election name.
`number_of_tellers`	Is the number of tellers being used in the election.
`threshold_tellers`	Is the threshold of tellers needed for teller operations.
`teller`	Is the local teller number.
`teller_ip`	Is the publicly accessible IP address for the local teller.
`teller_main_port`	Is the main port on which Verificatum will listen for teller operations.
`teller_hint_port`	Is the hint port on which Verificatum will advertise that it is available.
`number_of_voters`	Is the maximum number of voters in the election.
`ers_voters_file`	Is the import file containing the ERS voter identifiers.
`ers_associated_voters_file`	Is the export file which will contain voter parameters associated with ERS voters.

Throughout this section the private or public nature of the files used is highlighted using red for private files which should not be published or shared and green files which contain only public information and hence can be published and shared. An option available for testing of the election process is to run the election without a mix-net (defined when creating the election parameters). In a normal election, the mix-net holds privately the election private key as a share of the secret key held by each teller such that a threshold of tellers is needed to decrypt data. When testing without a mix-net, the election private key must be explicitly provided for stages requiring decryption, rather than relying on the mix-net from which the private key is never shared. Consequently, files highlighted in purple indicate that, during a normal election with a mix-net, these files only hold public information, whereas without a mix-net, these files will contain private information as well.

Files which are highlighted as being shared or published can be uploaded to the VMV web service, as can some of the private files which are needed by, for example, the different teller nodes to complete operations. Details on how to publish these files can be found in under Vote Verification and Data Publication later in this document.

1. CREATE ELECTION PARAMETERS

The creation of election parameters is conducted on a secure, independent computer. The election parameters are DSA parameters with key lengths of $L=3072$ and $N=256$, as recommended and built using algorithms defined by [18] (section 4 and appendix A). The parameters consist of a prime modulus p with bit length L, q a prime divisor of $(p-1)$ with bit length N and a subgroup generator g of order q. These parameters are generated randomly using an algorithm which chooses p and q such that their likelihood of being prime is sufficient. This sufficiency is determined by the number of times that the Miller-Rabin primality test is performed against the probable prime, which we set as 64 iterations (a value of prime certainty of 128) as recommended by [19]. These DSA parameters can be used to generate both DSA and ElGamal key pairs, including the mix-net key pair with shared secret key.

To generate the parameters, start the VMV command line tool and enter (all on one line) **[on an independent computer]**:

```
create-election-parameters
  --publish public-election-params.csv
  --name <name>
  --number-of-tellers <tellers> --threshold-tellers <threshold>
  --dsa-l 3072 --dsa-n 256 --prime-certainty 128
```

where

`<name>`	Is the name of the election. For example, "ERS Election".
`<tellers>`	Is the number of tellers (mix-net nodes) to use. For example, 4.
`<threshold>`	Is the threshold number of tellers required for correct option. For example, 3.

Note that if the "`--dsa-l`", "`--dsa-n`" and "`–prime-certainty`" options are omitted, then their values will default to 3072, 256 and 128, respectively.

This operation will take some short time (up to 5 minutes) to generate the parameters. Since all of the parameters are public, they are output to the file "`public-election-params.csv`", which can be shared and published. This is a CSV file with the following fields:

`g`	The DSA *g* generator. A big integer as a decimal string.
`j`	The parameters are stored internally as Diffie-Hellman parameters which also define a *j*. This is not used and is therefore empty.
`l`	The DSA *L* bit length. An integer.
`m`	The DSA *N* bit length. The Diffie-Hellman parameters store this as an *m*. An integer.
`p`	The DSA *p* prime. A big integer as a decimal string.
`q`	The DSA *q* prime. A big integer as a decimal string.
`name`	The name of the election. A string.
`numberOfTellers`	The number of tellers. An integer.
`thresholdTellers`	The threshold number of tellers. An integer.

During testing it may be necessary to run the election stages without a mix-net. In this case, the option "`--no-tellers`" can be used to remove the use of tellers. Note that this mode of operation is insecure and should not be used for a real election since shuffling and decryption can be performed by a single party, therefore allowing votes to be mapped to a voter. When this mode is enabled, the number of tellers and their threshold are set to zero.

2. CREATE TELLERS

The mix-net is made up of a number of mix-net tellers. Each teller is used to independently perform operations on encrypted data which are designed to prevent any one party compromising the election by decrypting or modifying the election data, including votes. This is achieved by ensuring that each teller is run by an independent organisation, and then for each teller to hold privately a share of the election private key which is not shared with any other teller or with those running the election. Tellers are then used to shuffle and decrypt ciphertexts as a consortium in which a threshold of tellers is required to complete any operation. These operations are implemented via the Verificatum tool set, which requires initialisation with the election parameters in order to create a valid teller node. This initialisation is performed in two parts. The first uses the election parameters to create a local teller node, while the second then joins the tellers together by sharing and merging their public information.

Before the first stage can be executed, the "`public-election-params.csv`" file must be copied to all tellers to a suitable directory from which the VMV command will be run. The same directory will be used throughout.

To execute the first part of the teller creation, start the VMV command line and enter the following **[on each teller]**:

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

where

`teller`	Is the unique local teller number. An integer within the range of 1 to the number of tellers inclusive. This number will be allocated to each teller.
`ip`	Is the IP address (or host name) of the teller such that it can be contacted via the network.
`teller-port`	Is the port on which the teller will listen for mix-net operations to be performed. Access to this port should be restricted to only allow other tellers to use it.
`hint-port`	Is the port on which the teller will provide a listener to show that it is available for operation. Access to this port should be restricted to only allow other tellers to use it.

The input to this command is the generated "`public-election-params.csv`" file. Using the supplied parameters, this command outputs a teller information file "`teller-information-<teller>.xml`" (with appropriate substitution of the teller number) which must be copied to all tellers into the same directory from which the VMV command was run. Note that the command creates a sub-directory "`Teller<teller>`" which will contain all of the Verificatum data. This must be kept and should be backed-up to persistent and secure storage. Note that this directory also contains the Verificatum private keys which should not be shared, especially with other tellers.

Once all the "`teller-information-<teller>.xml`" files have been copied to all tellers, execute the second part to finish creation of the teller **[on each teller]**:

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

where

`teller`	Is the unique local teller number.

The input to this command is the generated "`public-election-params.csv`" file together with a list of all the teller information files in the correct order from "`teller-information-1.xml`" up to "teller-information-n.xml", where "n" is the total number of tellers. This command does not output any information, but it does modify the content of the "Teller<teller>" sub-directory so that each teller knows how to contact all other tellers. Therefore, once complete, the teller has been initialised and, provided the host computer allows TCP connections from other tellers via the two TCP ports, it should be able to take part in the subsequent mix-net operations when required.

3. CREATE ELECTION KEYS

Once the election parameters and tellers have been created, the election public and private key pair can be generated. The election key pair is an ElGamal key pair generated using the election parameters. Since the mix-net is used to decrypt ciphertexts, the private key is generated by the tellers as a shared secret key which has a single, corresponding public key. Each teller therefore has its own secret key which is not communicated with any other node. It is therefore important to ensure that the teller data is not deleted and is backed-up to persistent storage. In this way, decryption can only take place if a threshold number of tellers is involved in the decryption. The key generation is therefore defined and performed by Verificatum [4].

Before this stage can be executed, it is assumed that the "public-election-params.csv" file has already been copied to all tellers. This should have been done during the creation of the teller.

To generate the election key pair, execute the following VMV command at the same time **[on each teller]**:

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

where

teller	Is the unique local teller number. This is not required if no tellers are being used.

The input to this command is the generated "public-election-params.csv" file. This command then uses the mix-net tellers to generate the key pair, outputting the public key to the "election-keys.csv" and "public-election-keys.csv" files, of which the latter can be shared and published. The "public-election-keys.csv" file is needed for all other non-teller operations to be completed and must therefore be shared. Each teller's copy of this file should be the same.

The "election-keys.csv" file has the following fields:

privateKey	The ElGamal private key. Empty when using a mix-net. A big integer as a decimal string.
publicKey	The ElGamal public key. A big integer as a decimal string.

The "public-election-keys.csv" file has the following fields:

publicKey	The ElGamal public key. A big integer as a decimal string.

If for testing, the election parameters were generated with the "--no-tellers" option, then the mix-net is not used to generate the key pair, and instead the key pair is generated locally. In this case, the "privateKey" field of "election-keys.csv" will contain the value of the private key and this should be held securely and not shared. This locally generated ElGamal key pair is generated by first choosing a random positive integer with a maximum of *(p-1)* as the private key, and then calculating the public key as $(g^x \bmod p)$.

4. CREATE VOTER KEYS

Each voter is associated with a DSA signing key pair and an ElGamal encryption key pair. Since voters do not routinely have their own signing key pair, the demonstrator can be used to generate both key pairs using the election parameters and the algorithms previously described. ERS hold only the public keys for the voters.

To generate the key pairs for all required voters, execute the following VMV command **[on an independent computer]**:

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

where

`number`	Is the number of voters. This must be known in advance of the election and is fixed throughout. An integer.

This command uses the generated "`public-election-params.csv`" file as input. When executed, the required number of DSA and ElGamal key pairs are generated for each voter. Both the private and public keys are then output to the "`voters-keys.csv`" file, while only the public keys are output to the "`public-voters-keys.csv`" file which can be shared and published. The "`voters-keys.csv`" file should therefore not be shared, since it has the following fields:

`privateKeySignature`	The DSA private key. A big integer as a decimal string.
`publicKeySignature`	The DSA public key. A big integer as a decimal string.
`privateKeyTrapdoor`	The ElGamal private key. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key. A big integer as a decimal string.

The "`public-voters-keys.csv`" file can be shared publicly, and has the following fields:

`publicKeySignature`	The DSA public key. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key. A big integer as a decimal string.

If public keys for all voters can be provided, then the above command can be skipped. As an alternative, the following command can be run by an individual voter **[on their own computer]:**

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

This command uses the generated "`public-election-params.csv`" file as input. When executed, the voter's DSA and ElGamal key pairs are generated. Both the private and public keys are then output to the "`voter-keys.csv`" file, while only the public keys are output to the "`public-voter-keys.csv`" file which can be shared and published. The "`voter-keys.csv`" file should therefore not be shared, since it has the following fields:

`privateKeySignature`	The DSA private key. A big integer as a decimal string.
`publicKeySignature`	The DSA public key. A big integer as a decimal string.
`privateKeyTrapdoor`	The ElGamal private key. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key. A big integer as a decimal string.

The "`public-voter-keys.csv`" file can be shared publicly, and has the following fields:

`publicKeySignature`	The DSA public key. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key. A big integer as a decimal string.

This latter file should be aggregated with all other public voters' keys to form the "`public-voters-keys.csv`" file, which must be provided in the format required at the necessary subsequent stages. Note also that if public keys are provided in this way that the encryption is assumed to have taken place externally to the demonstrator.

5. CREATE TRACKER NUMBERS

Every voter is associated with an encrypted tracker number such that it is not possible to know which plaintext tracker number is associated with a voter. Tracker numbers are just unique random integers generated such that they are sparse (selected randomly with a uniform distribution in the range *10,000,000* to *99,999,999*). These unique numbers are then transformed into numbers within the group defined by the election parameters for which g is a generator so that they can be encrypted. This is done by calculating $(g^n \bmod p)$, where n is the tracker number. This in turn is then encrypted using ElGamal with the election public key.

To generate these values, execute the following VMV command **[on an independent computer]:**

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

where

`number`	Is the number of voters as previously defined.

The input files to this command are the generated "`public-election-params.csv`" and "`public-election-keys.csv`" files. The output file "`public-tracker-numbers.csv`" contains the plaintext tracker number and group value, and the encrypted group number. Since the encrypted tracker number is subsequently shuffled and re-encrypted, this file can be shared and published. When decrypting votes and tracker numbers after voting has finished, the published mapping from the group value to the plaintext tracker number will be used.

The "`public-tracker-numbers.csv`" file contains the following fields:

`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.
`trackerNumber`	The unique tracker number. An integer.
`trackerNumberInGroup`	The tracker number within the group. A big integer as a decimal string.

6. SHUFFLE TRACKER NUMBERS

Once the tracker numbers have been created, they need to be shuffled using the mix-net so that no voter can be associated with a plaintext tracker number. This is achieved using a Verificatum shuffle, which on each teller takes as input the encrypted tracker numbers and shuffles and re-encrypts them so that there is no observable linkage between the inputs and outputs.

Before this stage can be executed, the "`public-election-params.csv`", "`public-election-keys.csv`" and "`public-tracker-numbers.csv`" files must be copied to all tellers such that the files exist in the same directory from which the VMV command line was run for the teller. Note that the "`public-election-params.csv`" and "`public-election-keys.csv`" files should already exist on the teller as they have been used previously.

Once the files have been copied, the following command is run at the same time **[on each teller]**:

```
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
```

where

`teller`	Is the unique local teller number. This is not required if no tellers are being used.

The input files to this command are the generated "`public-election-params.csv`" file, together with the public tracker number data in "`public-tracker-numbers.csv`". The output file "`shuffled-tracker-numbers.csv`" contains the shuffled and re-encrypted tracker numbers which can be published, together with the "`shuffle-proofs-<teller>.zip`" file. This file contains the following field:

`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.

The "`shuffle-proofs-<teller>.zip`" is a ZIP file which contains all of the data associated with the proof of shuffle as generated by Verificatum.

When the "`--no-tellers`" option is used to create the election parameters, Verificatum is not used to perform the shuffle. Instead the shuffle is performed locally and none of the encrypted tracker numbers is re-encrypted and the "`shuffle-proofs-<teller>.zip`" file will be empty. This version is therefore insecure in that the shuffled tracker numbers can be mapped to their un-shuffled versions.

7. CREATE TRACKER NUMBER COMMITMENTS

For a voter to be able to verify their vote after the election has finished, they need two commitment values which can be used to obtain their plaintext tracker number. These α and β commitment values are calculated using random numbers generated independently on each teller. During the election setup stage, only the β commitment value is calculated.

Before this stage can be executed, the "`public-election-params.csv`", "`public-election-keys.csv`", "`public-voters-keys.csv`" and "`shuffled-tracker-numbers.csv`" files must be copied to all tellers such that the files exist in the same directory from which the VMV command line was run for the teller. Note that the "`public-election-params.csv`", "`public-election-keys.csv`" and "`shuffled-tracker-numbers.csv`" files should already exist on the teller as they have been used or generated previously.

Once the files have been copied, the following command is run **[on each teller]**:

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

The input files to this command are the generated "`public-election-params.csv`" and "`public-election-keys.csv`" files, together with the public voter keys "`public-voters-`

keys.csv" and public shuffled tracker number data in "shuffled-tracker-numbers.csv". The output file "commitments-<teller>.csv" contains the random values used to create the commitments, together with the encrypted values used to create the β value, for the local "<teller>". This file must not be shared until after voting closes, and must not be made public. The file must be retained on the teller (and in its backup). The "public-commitments-<teller>.csv" file contains just the encrypted values and can be published and shared with all tellers, together with the "commitments-proofs-<teller>.csv" file.

The random value r_{ij} is a positive integer with bit length L modulus p, such that each teller j chooses their own random value for each voter i. The h term is then calculated as $(h_i^{r_{ij}}\ mod\ p)$ where h_i is the voter's public trapdoor encryption key. The g term is similarly calculated as $(g^{r_{ij}}\ mod\ p)$, where g is the election parameter group G generator. Both of these terms are in turn is then encrypted using ElGamal with the election public key. The random value is not retained.

This "commitments-<teller>.csv" file contains the following fields:

encryptedG	The encrypted *g* term of the commitment. A series of bytes as a base 64 encoded string.
encryptedH	The encrypted *h* term of the commitment. A series of bytes as a base 64 encoded string.
g	The *g* term of the commitment. A big integer as a decimal string.
h	The *h* term of the commitment. A big integer as a decimal string.
publicKey	The voter's trapdoor public key associated with the commitment.

This "public-commitments-<teller>.csv" file contains the following fields:

encryptedG	The encrypted *g* term of the commitment. A series of bytes as a base 64 encoded string.
encryptedH	The encrypted *h* term of the commitment. A series of bytes as a base 64 encoded string.
publicKey	The voter's trapdoor public key associated with the commitment.

This "commitments-proofs-<teller>.csv" file contains the following fields:

a1Dash	The A'_1 term of the commitment. A big integer as a decimal string.
a2Dash	The A'_2 term of the commitment. A big integer as a decimal string.
b1Dash	The B'_1 term of the commitment. A big integer as a decimal string.
b2Dash	The B'_2 term of the commitment. A big integer as a decimal string.
c	The *C* term of the commitment. A big integer as a decimal string.
d	The *D* term of the commitment. A big integer as a decimal string.

`pi11hash`	The $\pi_{1,1}$ proof hash term. A big integer as a decimal string.
`pi11signature`	The $\pi_{1,1}$ proof signature term. A big integer as a decimal string.
`pi12hash`	The $\pi_{1,2}$ proof hash term. A big integer as a decimal string.
`pi12signature`	The $\pi_{1,2}$ proof signature term. A big integer as a decimal string.
`pi21hash`	The $\pi_{2,1}$ proof hash term. A big integer as a decimal string.
`pi21signature`	The $\pi_{2,1}$ proof signature term. A big integer as a decimal string.
`pi22hash`	The $\pi_{2,2}$ proof hash term. A big integer as a decimal string.
`pi22signature`	The $\pi_{2,2}$ proof signature term. A big integer as a decimal string.
`pi23hash`	The $\pi_{2,3}$ proof hash term. A big integer as a decimal string.
`pi23signature`	The $\pi_{2,3}$ proof signature term. A big integer as a decimal string.
`pi31hash`	The $\pi_{3,1}$ proof hash term. A big integer as a decimal string.
`pi31signature`	The $\pi_{3,1}$ proof signature term. A big integer as a decimal string.
`pi32hash`	The $\pi_{3,2}$ proof hash term. A big integer as a decimal string.
`pi32signature`	The $\pi_{3,2}$ proof signature term. A big integer as a decimal string.
`pi4hash`	The π_4 proof hash term. A big integer as a decimal string.
`pi4signature`	The π_4 proof signature term. A big integer as a decimal string.
`pi5hash`	The π_5 proof hash term. A big integer as a decimal string.
`pi5signature`	The π_5 proof signature term. A big integer as a decimal string.

8. DECRYPT TRACKER NUMBER COMMITMENTS

Once the encrypted *h* and *g* values for the commitments have been calculated, they need to be used to calculate the encrypted β and then decrypted. Each of the encrypted h values for a voter from each teller are multiplied together and then decrypted to form the β. The decryption takes place using the mix-net, and once complete, the consolidated list of voter data can be built which contains the association between the voter public encryption keys, encrypted tracker numbers and the β commitments.

Before this stage can be executed, the "`public-election-params.csv`", "`election-keys.csv`", "`public-voters-keys.csv`" and "`shuffled-tracker-numbers.csv`" files must be copied to all tellers such that the files exist in the same directory from which the VMV command line was run for the teller. All "`public-commitments<teller>.csv`" files also need to be copied to all tellers. Note that the "`public-election-params.csv`", "`shuffled-tracker-numbers.csv`" and "`public-voters-keys.csv`" files should already exist on the teller as they have been used or generated previously. Note that this command uses the "election-keys.csv" file which, when using the "--no-tellers" option, contains the election private key, but which is not present when using a mix-net (but still contains an empty field for the private key).

Once the files have been copied, the following command is run at the same time **[on each teller]**:

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

where

teller	Is the unique local teller number. This is not required if no tellers are being used.

The input files to this command are the generated "public-election-params.csv" and "election-keys.csv" files, together with the public voter keys "public-voters-keys.csv", public shuffled tracker number data in "shuffled-tracker-numbers.csv" and all of the public "public-commitments<teller>.csv" files from every teller. The output file "public-voters.csv" contains the consolidated public voter information, together with the β value, and can be published and shared, together with the "decrypt-proofs-<teller>.zip" file.

The "public-voters.csv" file contains the following fields:

beta	The β component of the encrypted tracker number commitment. A big integer as a decimal string.
encryptedTrackerNumberInGroup	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.
publicKeySignature	The DSA public key for the voter. A big integer as a decimal string.
publicKeyTrapdoor	The ElGamal public key for the voter. A big integer as a decimal string.

9. EXPORT VOTER IDENTIFIERS FROM ERS

All the above stages are carried out on an independent computer or the mix-net teller nodes to generate the public and private data necessary to initialise verifiability for the election. The next stage is to associate the generated public data with voters within the ERS system. Here, the ERS system will only hold public key and encrypted information.

To establish the association between voters and the generated data, a list of all of the voter identifiers is needed from the ERS system. This should be provided as a single "ers-voters.csv" CSV file with just a single field:

id	The id of a voter. An integer.

If voters' public keys are being provided rather than generated by the demonstrator, then this file should instead contain the following fields:

id	The id of a voter. An integer.
publicKeySignature	The DSA public key for the voter. A big integer as a decimal string.

`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

This file should be kept within the ERS secure network and not shared or published.

10. ASSOCIATE VOTER IDENTIFIERS WITH VOTER DATA

On a computer within the ERS secure network, and which only has the election information (no private keys or plaintext tracker numbers), the association between ERS voters and the generated verifiability parameters can be formed. This is achieved by running the following VMV command **[on an ERS computer]**:

```
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
```

The input files to this command are the generated "`public-election-params.csv`" and "`public-election-keys.csv`" files, together with the public voter keys "`public-voters.csv`" generated during the decryption of the commitments, and the voter data exported from ERS "`ers-voters.csv`". The output file "`ers-associated-voters.csv`" contains the data which can be imported into the ERS system, including the voter's allocated public keys, encrypted tracker number and β commitment. The "`public-associated-voters.csv`" file contains a subset of this data which can be published.

The "`ers-associated-voters.csv`" file contains the following fields:

`beta`	The β component of the encrypted tracker number commitment. A big integer as a decimal string.
`id`	The id of the voter. An integer.
`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.
`publicKeySignature`	The DSA public key for the voter. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

The "`public-associated-voters.csv`" file contains the following fields:

`beta`	The β component of the encrypted tracker number commitment. A big integer as a decimal string.
`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.
`publicKeySignature`	The DSA public key for the voter. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

This file therefore does not provide any information about the voter, such as their identifier and can be published.

11. IMPORT VOTER DATA INTO ERS

Once each voter has been associated with the relevant election data through the generation of the "`ers-associated-voters.csv`" file, the data can be imported into the ERS system for persistent storage against the voter. This will enable ERS to maintain the associated data against the voter during the voting period, including sending the voter their β component of the encrypted tracker number commitment.

Assuming that this data is imported into a relational database, then the following additional fields are required per voter, matching the imported data:

`beta`	The β component of the encrypted tracker number commitment. A big integer as a decimal string.
`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.
`publicKeySignature`	The DSA public key for the voter. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

Here, the id of the voter is used to link the imported data with the original export.

12. EXPORT VOTE OPTIONS FROM ERS

Normally, all of the plaintext votes are encrypted once voting has closed. To encrypt plaintext votes, all unique vote options must be mapped to a number in the group defined by the election parameters. Optionally at this stage, if the vote options are known in advance of voting, for example if votes are to be encrypted elsewhere, then the vote options should be exported from the ERS system and mapped. This can be done on an ERS computer, although all vote option data is public.

The vote options should be provided as a single "`ers-vote-options.csv`" CSV file with two fields:

`option`	The vote option. A string.
`optionNumberInGroup`	The vote option number within the group. A big integer as a decimal string. This field is optional and will typically be left blank to be automatically mapped to a number in the group. If provided, the number must be unique and must be a member of the group.

If any vote option has a pre-assigned "`optionNumberInGroup`" (which is valid and unique) then it should be included. This file can be shared and published.

13. MAP VOTE OPTIONS

Once the vote options have been exported from the ERS system, they can be mapped to numbers in the group.

The input files to this command are the generated "`public-election-params.csv`" and the exported "`ers-vote-options.csv`" files.

To map the vote options, run the following VMV command **[on an ERS computer]**:

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

The "`public-vote-options.csv`" csv file can be imported into the ERS system as well as published. The mapped values can be used to encrypt votes:

`option`	The vote option. A string.
`optionNumberInGroup`	The vote option number within the group. A big integer as a decimal string.

If any vote option in the "`ers-vote-options.csv`" file has a pre-assigned "`optionNumberInGroup`" (which is valid and unique) then it will be retained for the vote option in the output "`public-vote-options.csv`" file. In contrast, all missing "`optionNumberInGroup`" entries will have newly generated values.

14. IMPORT VOTE OPTIONS INTO ERS

Once each vote option has been mapped to an option number in the group, the corresponding "`public-vote-options.csv`" file can be imported into the ERS system for persistent storage.

Assuming that this data is imported into a relational database, then the following additional fields are required, matching the imported data:

`option`	The vote option. A string.
`optionNumberInGroup`	The vote option number within the group. A big integer as a decimal string.

VOTING

OVERVIEW

Once the election has been setup and the verification data imported into the ERS system, the election proceeds via the ERS system as normal, where voters can use their security codes to log into the system and cast their votes.

The ERS system differs for verifiability in two places: 1) when voter data has been setup and a voter opted into verification, and 2) at vote confirmation.

When casting their vote, if a voter opts into verification (or if this is the default for the election), then the associated voters will receive an email will allow them to find out whether their β component is valid and whether it has been used for voting or not. This allows a voter to check that, for example, if they abstain from voting, that their vote is not used by somebody else. A VMV web service URL can be used by the voter to check their β component. An example URL is:

```
https://vmv.surrey.ac.uk/verify/status?election=<election>
                         &beta=<beta>
```

where

`election`	Is the unique name of the election as defined in the VMV web service. The name is URL encoded so that, for example, space characters are encoded as '+' or '%20'.
`beta`	Is the β value as a decimal number.

During vote confirmation, all voters will receive an email which indicates that their vote can be verified, and the steps that they should perform to complete verification once the election has completed. This email can include the β component of the encrypted tracker number commitment, although it cannot be used until the corresponding α value has been generated and stored within the ERS system after the election, at which point a subsequent email can be sent with both the α and β components.

When both the α and β components are provided, these can be formed into a URL that can be used by the voter to verify their vote using the VMV web service. An example URL is provided below:

```
https://vmv.surrey.ac.uk/verify?election=<election>
              &alpha=<alpha>&beta=<beta>
```

where

`election`	Is the unique name of the election as defined in the VMV web service. The name is URL encoded so that, for example, space characters are encoded as '%20'.
`alpha`	Is the α value as a decimal number.
`beta`	Is the β value as a decimal number.

VOTE ENCRYPTION

OVERVIEW

Once the election has completed, the next stage is to encrypt the plaintext votes, shuffle them so that there is no linkage between voters and their encrypted votes, then decrypt and publish the votes alongside their plaintext tracker number.

The encryption of plaintext votes involves encryption and signing of the vote using the voter's keys. In an ideal system, only the voter would hold their private encryption and signing keys, and when a vote is cast, it would be encrypted prior to being received by the ERS system. In this demonstrator, the voter's private keys are held separately to ERS, who store the plaintext vote. Vote encryption therefore includes this encryption and signing stage in order to allow votes to be securely shuffled prior to publication. Optionally, encrypted votes, their signatures and proof of encryption can be supplied based upon externally held voters' keys. In this case, the encryption stage will collate together these externally encrypted votes and calculate the corresponding α commitment values.

To publish the cast votes, the following steps are run on the ERS network:

15. Export voter data and plaintext votes from the ERS system **[on an ERS computer]**
16. Encrypt and sign the plaintext votes **[on an ERS computer]**
17. Import the encrypted and signed votes into the ERS system **[on an ERS computer]**

Once these steps have been completed, the encrypted and signed votes can be removed from the ERS network as they contain no private information or anything that identifies the voter. The following steps are then required on each teller to produce the votes ready for publication:

18. Mix the encrypted votes and corresponding encrypted tracker numbers **[on each teller]**

Mixing the encrypted votes and tracker numbers shuffles the votes across tellers and then performs a threshold decryption. The shuffle and decrypt ensures that there is no correspondence between the input encrypted votes and the outputs, and hence the votes and their tracker numbers can be published.

To verify that all operations have been carried out correctly, the α and β commitment values for a voter can be used, together with the voter's encryption public key, to obtain the corresponding tracker number. The tracker number can then be used to verify the corresponding plaintext vote against that which is stored on the ERS system. This verification step should therefore only be run on the ERS network:

19. Decrypt a voter's tracker number to verify a vote **[on an ERS computer]**

These steps can be automatically completed using the "election_encrypt.exp" script, together with its associated "election_common.exp" script. These scripts need to be downloaded to each teller, where the first teller is assumed to take the role of the independent computer, ERS computer and teller. The initialisation can then be completed by running the script on each teller, as follows.

On the first teller:

```
election_encrypt.exp <vmv_jar_file>
  <ssh_key_file> <sftp_host> <sftp_user>
  <election_name> <number_of_tellers> <teller>
  <ers_plaintext_voters_file> <ers_encrypted_voters_file>
```

On every other teller:

```
election_encrypt.exp <vmv_jar_file>
  <ssh_key_file> <sftp_host> <sftp_user>
  <election_name> <number_of_tellers> <teller>
```

where

`vmv_jar_file`	Is the full path to the VMV JAR file.
`ssh_key_file`	Is the full path to the SFTP key file needed for the SFTP server.
`sftp_host`	Is the SFTP host domain name or IP address.
`sftp_user`	Is the SFTP user name.
`election_name`	Is the (unique) election name.
`number_of_tellers`	Is the number of tellers being used in the election.
`teller`	Is the local teller number.
`ers_plaintext_voters_file`	Is the import file containing the ERS voters with their plaintext votes.
`ers_encrypted_voters_file`	Is the export file which will contain encrypted votes associated with ERS voters.

As previously used in this document, the private or public nature of the files used in this section is highlighted using red for private files, green for public files, and purple files indicating that, during a normal election with a mix-net, these files only hold public information, whereas without a mix-net, these files will contain private information as well.

Files which are highlighted as being shared or published can be uploaded to the VMV web service, as can some of the private files which are needed by, for example, the different teller nodes to complete operations. Details on how to publish these files can be found in under Vote Verification and Data Publication later in this document.

15. EXPORT PLAINTEXT VOTES

This stage exports the cast plaintext votes from the ERS system so that they can be encrypted. To achieve this, a list of the voter information with the plaintext votes is required. This should be provided as an "`ers-plaintext-voters.csv`" CSV file with the following fields:

`beta`	The β component of the encrypted tracker number commitment. A big integer as a decimal string.
`id`	The id of the voter. An integer.
`plainTextVote`	The plaintext vote. A string.
`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.
`publicKeySignature`	The DSA public key for the voter. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

Note that the order of the fields is important. This file should be kept within the ERS secure network and not shared or published.

If votes are being encrypted externally to the demonstrator, then this file should include the encrypted vote and its signature, and the plaintext vote may be blank. In this case, the file should have the following fields:

`beta`	The β component of the encrypted tracker number commitment. A big integer as a decimal string.
`encryptedVote`	The encrypted vote. A series of bytes as a base 64 encoded string.
`encryptedVoteSignature`	The signature for the encrypted vote. A series of bytes as a base 64 encoded string.
`id`	The id of the voter. An integer.
`plainTextVote`	The plaintext vote. A string. Should be blank.
`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.
`publicKeySignature`	The DSA public key for the voter. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

The plaintext vote is an arbitrary string intended to be a human readable version of the vote which can be published without further interpretation (once encrypted and mixed). However, the complete list of possible plaintext votes is required in order to perform the encryption. This should be provided as an "`ers-vote-options.csv`" CSV file with the following fields:

`option`	The vote option. A string.
`optionNumberInGroup`	The vote option number within the group. A big integer as a decimal string. May be missing to indicate that a suitable number should be generated.

All of the possible vote options should be provided in this file such that all of the exported plaintext votes map to only one of the vote options. This vote option file can be shared and published. The "`optionNumberInGroup`" is optional (although the field itself must be provided). If a value is provided for a vote option, then this should be strictly an option number within the group and should also be unique across all vote options.

16. ENCRYPT AND SIGN VOTES

Once the plaintext votes have been exported from the ERS system, they can be encrypted and signed and the remaining α commitment value for each voter can be calculated.

If voters' keys have been supplied previously, rather than being generated by the demonstrator, then this stage requires encrypted votes to be supplied. To achieve this, the following command can be run by an individual voter **[on their own computer]:**

```
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
```

The input files to this command are the generated "`public-election-params.csv`" and "`public-election-keys.csv`" files, together with the private and public keys for the voter "`voter-keys.csv`", and the possible vote options "`public-vote-options.csv`". When executed, this command will encrypt the "`<vote>`" using the public election key, producing a proof of encryption, and sign it using the voter's private signing key. The output will therefore be the encrypted vote in "`public-encrypted-voter.csv`" and the corresponding proof in "`encrypt-proof.csv`". All of these files may be collated together with the voter details for generation of the α commitment values.

The "`public-encrypted-voter.csv`" file can be published, and contains the following fields:

`encryptedVote`	The encrypted vote. A series of bytes as a base 64 encoded string.
`encryptedVoteSignature`	The signature for the encrypted vote. A series of bytes as a base 64 encoded string.
`publicKeySignature`	The DSA public key for the voter. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

This "`encrypt-proof.csv`" file contains the following fields:

`c1Bar`	The $c_{R,1}$ term of the proof. A big integer as a decimal string.
`c1R`	The $c_{R,2}$ term of the proof. A big integer as a decimal string.
`c2Bar`	The $\bar{c_1}$ term of the proof. A big integer as a decimal string.
`c2R`	The $\bar{c_2}$ term of the proof. A big integer as a decimal string.
`encryptedVoteSignature`	The signature for the encrypted vote. A series of bytes as a base 64 encoded string.

The above command is optional and is only used if voters are encrypting their own votes using privately held keys. If all votes are being encrypted by the demonstrator with the voter keys that it holds, or if the above command has been used and the encrypted votes collated, then encryption and/or generation of the α commitment values can be performed using the following command **[on an ERS computer]**:

```
encrypt-votes
  --election public-election-params.csv public-election-keys.csv
  --voters  voters-keys.csv ers-plaintext-voters.csv
            ers-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
```

* Note that "`ers-vote-options.csv`" should be replaced with "`public-vote-options.csv`" if the vote options were previously mapped using the "`map-vote-options`" command. If this is not done, then new option numbers in the group will be generated and used, invalidating any previously encrypted votes because they will have used the previous mapping.

This command will only encrypt and sign plaintext votes where encrypted votes have not been provided. If encrypted votes for all voters have been provided then this stage collates these together with the α commitment values.

Before this stage can be executed, the "`commitments-<teller>.csv`" files for each teller must be copied to the ERS computer on which this stage will run. These files must be shared securely with the ERS computer. This enables the α commitment value to be calculated by multiplying each of the voter's ($g^{r_{ij}}$ *mod p)* values from each teller together.

The input files to this command are the generated “`public-election-params.csv`” and “`public-election-keys.csv`” files, together with the private and public voter keys “`voters-keys.csv`”, the private teller commitment data from all tellers “`commitment-<teller>.csv`”, the voter data with plaintext votes exported from ERS “`ers-plaintext-voters.csv`”, the proofs of encryption if externally encrypted “`ers-encrypt-proofs.csv`” and the possible vote options “`ers-vote-options.csv`”. The “`voters-keys.csv`”, “`ers-encrypt-proofs.csv`” and “`ers-vote-options.csv`” files are optional. If the “`voters-keys.csv`” file is not supplied, then it is assumed that all votes have been encrypted previously such that “`ers-encrypt-proofs.csv`” contains the proof of encryption for every plaintext vote. If the “`ers-vote-options.csv`” file is not supplied, then the list of unique plaintext votes will be generated.

The output file “`ers-encrypted-voters.csv`” contains the encrypted and signed votes and the α commitment value which can be imported into the ERS system. The “`public-encrypted-voters.csv`” file contains a subset of this data, while the “`public-vote-options.csv`” file contains the mapping of vote options to their corresponding encryption inputs as numbers within the election parameters group *G*. These latter files, together with the “`encrypt-proofs.csv`” file can be published and shared.

Before the votes can be encrypted, they must all be mapped to a number which is contained within the election parameters group *G*, since ElGamal encryption can only be performed on such numbers. To achieve this, each vote options is mapped to unique random number (selected randomly with a uniform distribution in the range *1* to $2^{31}-1$). These unique numbers are then transformed into numbers within the group defined by the election parameters for which *g* is a generator so that they can be encrypted. This is done by calculating $(g^v \bmod p)$, where *v* is the vote option random number. Each vote is then mapped to the corresponding value and then encrypted using ElGamal with the election public key.

Once a vote has been encrypted, it is signed using the voter’s private DSA signature key. The private signature key is found by using the corresponding voter’s public signature key.

The “`ers-encrypted-voters.csv`” file contains the following fields:

`alpha`	The α component of the encrypted tracker number commitment. A big integer as a decimal string.
`encryptedVote`	The encrypted vote. A series of bytes as a base 64 encoded string.
`encryptedVoteSignature`	The signature for the encrypted vote. A series of bytes as a base 64 encoded string.
`id`	The id of the voter. An integer.

This file should not be shared or published and is used to update the ERS system with the encrypted votes. The id of the voter is used to match the encrypted vote to the corresponding voter on import.

The “`public-encrypted-voters.csv`” file can be published, and contains the following fields:

`beta`	The β component of the encrypted tracker number commitment. A big integer as a decimal string.
`encryptedVote`	The encrypted vote. A series of bytes as a base 64 encoded string.
`encryptedVoteSignature`	The signature for the encrypted vote. A series of bytes as a base 64 encoded string.
`encryptedTrackerNumberInGroup`	The encrypted tracker number in the group. A series of bytes as a base 64 encoded string.

`publicKeySignature`	The DSA public key for the voter. A big integer as a decimal string.
`publicKeyTrapdoor`	The ElGamal public key for the voter. A big integer as a decimal string.

The "`public-vote-options.csv`" can also be published and will be used to map decrypted votes to plaintext vote options for publication. It contains the following fields:

`option`	The vote option. A string.
`optionNumberInGroup`	The vote option number within the group. A big integer as a decimal string.

If the "`ers-vote-options.csv`" file is supplied, then if any vote option has a pre-assigned "`optionNumberInGroup`" (which is valid and unique) then it will be retained for the vote option in the output "`public-vote-options.csv`" file. In contrast, all missing "`optionNumberInGroup`" will have values generated for them.

This "`encrypt-proofs.csv`" file contains the following fields:

`c1Bar`	The $c_{R,1}$ term of the proof. A big integer as a decimal string.
`c1R`	The $c_{R,2}$ term of the proof. A big integer as a decimal string.
`c2Bar`	The $\overline{c_1}$ term of the proof. A big integer as a decimal string.
`c2R`	The $\overline{c_2}$ term of the proof. A big integer as a decimal string.
`encryptedVoteSignature`	The signature for the encrypted vote. A series of bytes as a base 64 encoded string.

17. IMPORT ENCRYPTED VOTES

Once each vote has been encrypted and signed, the values generated in the "`ers-encrypted-voters.csv`" file can be imported into the ERS system for persistent storage against the voter. This will therefore require additional database fields for the encrypted vote:

`alpha`	The α component of the encrypted tracker number commitment. A big integer as a decimal string.
`encryptedVote`	The encrypted vote. A series of bytes as a base 64 encoded string.
`encryptedVoteSignature`	The signature for the encrypted vote. A series of bytes as a base 64 encoded string.

Here, the id of the voter is used to link the imported data with the export.

18. MIX VOTES

While the above stages are carried out on an ERS computer within the ERS secure network because of the presence of plaintext votes associated with voters, the following is carried out on the mix-net teller nodes. This is possible because the data that is removed from the ERS secure network is encrypted and not linked to a voter.

Before this stage can be executed, the "`public-election-params.csv`", "`election-keys.csv`", "`public-vote-opions.csv`", "`public-tracker-numbers.csv`" and "`public-encrypted-voters.csv`" files must be copied to all tellers such that the files exist in the same directory from which the VMV command line was run for the teller. Note that the "`public-`

election-params.csv" and "election-keys.csv" files should already exist on the teller as they have been used or generated previously. Note that this command uses the "election-keys.csv" file which, when using the "--no-tellers" option, contains the election private key, but which is not present when using a mix-net (but still contains an empty field for the private key). The "public-tracker-numbers.csv" must be the original file output by the "create-tracker-numbers" command and hence which contains the "trackerNumber" and "trackerNumberInGroup" fields.

To shuffle and decrypt the encrypted votes and their associated encrypted tracker numbers, the following command is run at the same time **[on each teller]**:

```
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
```

where

teller	Is the unique local teller number. This is not required if no tellers are being used.

The input files to this command are the generated "public-election-params.csv" and "election-keys.csv" files, together with the public encrypted votes and encrypted tracker numbers in the "public-encrypted-voters.csv" file and the "public-vote-opions.csv", "public-tracker-numbers.csv" files which are used to convert the outputs. The output file "public-mixed-voters.csv" contains the shuffled and decrypted tracker numbers and the decrypted vote options which are both obtained from the vote options and tracker number files. This output file, together with the "mix-proofs-<teller>.zip" file can be published and shared.

The "public-mixed-voters.csv" file contains the following fields:

plainTextVote	The plaintext vote. A string.
trackerNumber	The unique tracker number. An integer.

19. DECRYPT TRACKER NUMBER

This stage can be optionally executed multiple times to verify that the plaintext votes published with their corresponding tracker numbers matches to the data held by the ERS system. As such, this must be carried out within the ERS secure network as it requires access to a voter's plaintext vote.

To decrypt the α and β commitment values to reveal a tracker number for a voter, run the following VMV command **[on an ERS computer]**:

```
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
```

where

alpha	Is the α commitment value for the selected voter.
beta	Is the β commitment value for the selected voter.
public-key	Is the public encryption key for the selected voter.

The input files to this command are the generated "`public-election-params.csv`" file, together with the public tracker numbers and the private and public voter keys in the "`public-tracker-numbers.csv`" and "`voters-keys.csv`". If the values can be correctly used to find the voter's private encryption key and to decrypt the tracker number in the group, then the corresponding tracker number will be displayed. This tracker number can then be used to find the published plaintext vote for the voter and match this against the data held by ERS.

VOTE VERIFICATION AND DATA PUBLICATION

OVERVIEW

Throughout the election, verification data may be published to the distributed ledger to allow for independent viewing and verification. This is achieved by using the VMV web service, which allows registered administrators to upload and optionally publish data, while anyone can access the web service and view published information.

The VMV web service therefore has two ways of accessing data: 1) as a public (non-registered) user, and 2) as a (registered) administrator. Access as a public user involves accessing the public content of the service, while access as an administrator is authenticated using a previously registered email address and password.

All data which is uploaded and optionally published via the VMV web service is generated by the VMV command line tool. The following lists the files which should be uploaded and whether they can be published or not. The VMV command and associated output file are also listed:

Election Parameters

Contains the public cryptographic parameters for the election.

`create-election-parameters: public-election-params.csv`

Teller Information File

Contains the Verificatum teller information for a specific teller.

`create-teller: teller-information-<teller>.xml`[1]

Election Keys

Contains the election encryption keys.

`create-election-keys: election-keys.csv`

Election Public Key

Contains the public election encryption key.

`create-election-keys: public-election-keys.csv`

Voters' Keys

Contains the private and public voter encryption and signing keys.

`create-voters-keys: voters-keys.csv`

Voter Public Keys

Contains the public voter encryption and signing keys.

`create-voters-keys: public-voters-keys.csv`

[1] While the teller information file contains only public information and is publishable, within the VMV web service this file is not published because it contains the IP address for the teller which is kept private.

Tracker Numbers

Contains the public plain text and encrypted tracker numbers used to allow a voter to verify their cast vote.

`create-tracker-numbers: public-tracker-numbers.csv`

Shuffled and Encrypted Tracker Numbers

Contains the public encrypted tracker numbers in a random order such that it is not possible to know which plain text tracker number will be associated with which voter.

`shuffle-tracker-numbers: shuffled-tracker-numbers.csv`

Shuffled and Encrypted Tracker Numbers Proofs

Contains the Verificatum proof of shuffling of the public encrypted tracker numbers.

`shuffle-tracker-numbers: shuffle-proofs-<teller>.zip`

Teller Commitments

Contains the private plain text commitment values for a specific teller. Commitment values are provided to a voter to allow them to obtain their tracker number.

`create-commitments: commitments-<teller>.csv`

Teller Public Commitments

Contains the public encrypted commitment values for a specific teller.

`create-commitments: public-commitments-<teller>.csv`

Teller Public Commitments Proofs

Contains the proof of knowledge of the commitment values for a specific teller.

`create-commitments: commitments-proofs-<teller>.csv`

Voter Public Keys with Encrypted Tracker Numbers

Contains the list of voter public keys, encrypted tracker number and partial commitment ready to be linked to a voter.

`decrypt-commitments: public-voters.csv`

Voter Public Keys with Encrypted Tracker Numbers Proofs

Contains the Verificatum proof of decryption of the partial commitment for each commitment ready to be linked to a voter.

`decrypt-commitments: decrypt-proofs-<teller>.zip`

Voter Public Keys with Encrypted Tracker Numbers (Associated)

Contains the list of voter public keys, encrypted tracker number and partial commitment which have been linked to voters. No identifying information about voters is included.

`associate-voters: public-associated-voters.csv`

Encrypted Votes with Tracker Numbers

Contains the encrypted and signed vote for each voter together with their voter public keys, encrypted tracker number and partial commitment.

`encrypt-votes: public-encrypted-voters.csv`

Vote Choices

Contains the unique list of all possible votes that can be (or have been) cast.

`encrypt-votes: public-vote-options.csv`

Encrypted Votes with Tracker Numbers Proofs

Contains the proof of encryption of each encrypted vote for each voter.

`encrypt-votes: encrypt-proofs.csv`

Final Votes

Contains the shuffled and decrypted list of all plain text votes cast and their associated tracker numbers. No identifying information about voters is included or was used during the shuffle and decryption.

`mix-votes: public-mixed-voters.csv`

Final Votes Proofs

Contains the Verificatum proof of shuffling and decryption of the encrypted votes and encrypted tracker numbers.

`mix-votes: mix-proofs-<teller>.zip`

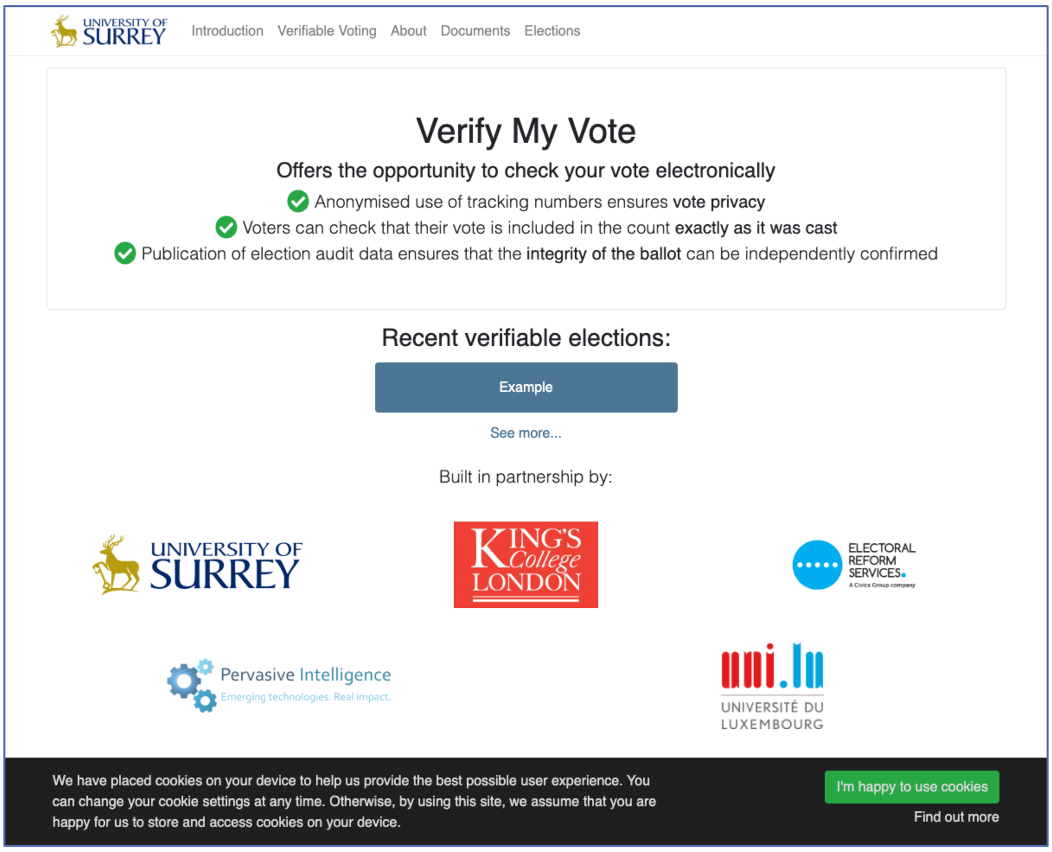
where

`teller`	Is the unique teller number. Where tellers are being used there will be one upload of the corresponding file type per teller.

As previously used in this document, the private or public nature of the files used in the above is highlighted using red for private files, green for public files, and purple files indicating that, during a normal election with a mix-net, these files only hold public information, whereas without a mix-net, these files will contain private information as well.

PUBLIC ACCESS AND FUNCTIONS

The public content of the VMV web service is accessed via the Home page URL https://vmv.surrey.ac.uk, an example of which is shown in Figure 2 which also shows the cookie notification.



Verify My Vote

Offers the opportunity to check your vote electronically

✓ Anonymised use of tracking numbers ensures **vote privacy**

✓ Voters can check that their vote is included in the count **exactly as it was cast**

✓ Publication of election audit data ensures that the **integrity of the ballot** can be independently confirmed

Recent verifiable elections:

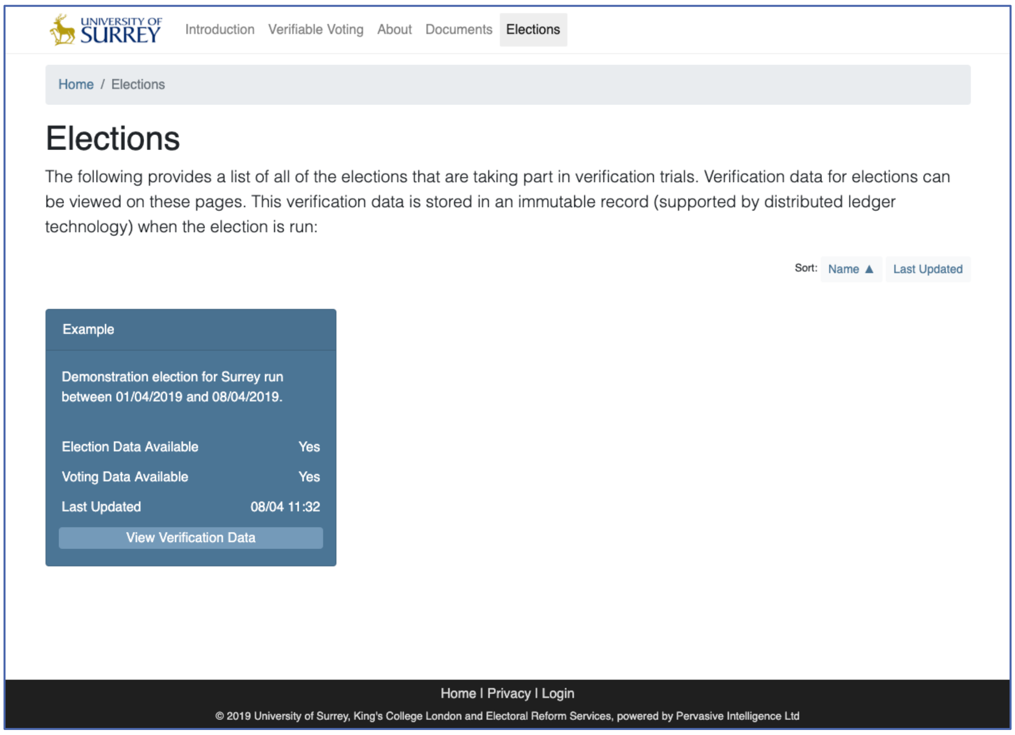
Built in partnership by:

We have placed cookies on your device to help us provide the best possible user experience. You can change your cookie settings at any time. Otherwise, by using this site, we assume that you are happy for us to store and access cookies on your device.

FIGURE 2: VMV WEB SERVICE – HOME

The home page provides general information about the project and access to election and other data via links. The header menu bar provides access to all available public pages, while the footer (visible once the cookie notice has been dismissed) provides access to the Privacy and Login pages. The Login page is for administrator access.

Selecting one of the recent elections will show data for the corresponding published election. A published election may also be accessed from the Elections page, an example of which is shown in Figure 3.



Elections

The following provides a list of all of the elections that are taking part in verification trials. Verification data for elections can be viewed on these pages. This verification data is stored in an immutable record (supported by distributed ledger technology) when the election is run:

Example

Demonstration election for Surrey run between 01/04/2019 and 08/04/2019.

Election Data Available	Yes
Voting Data Available	Yes
Last Updated	08/04 11:32

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 3: VMV WEB SERVICE – ELECTIONS

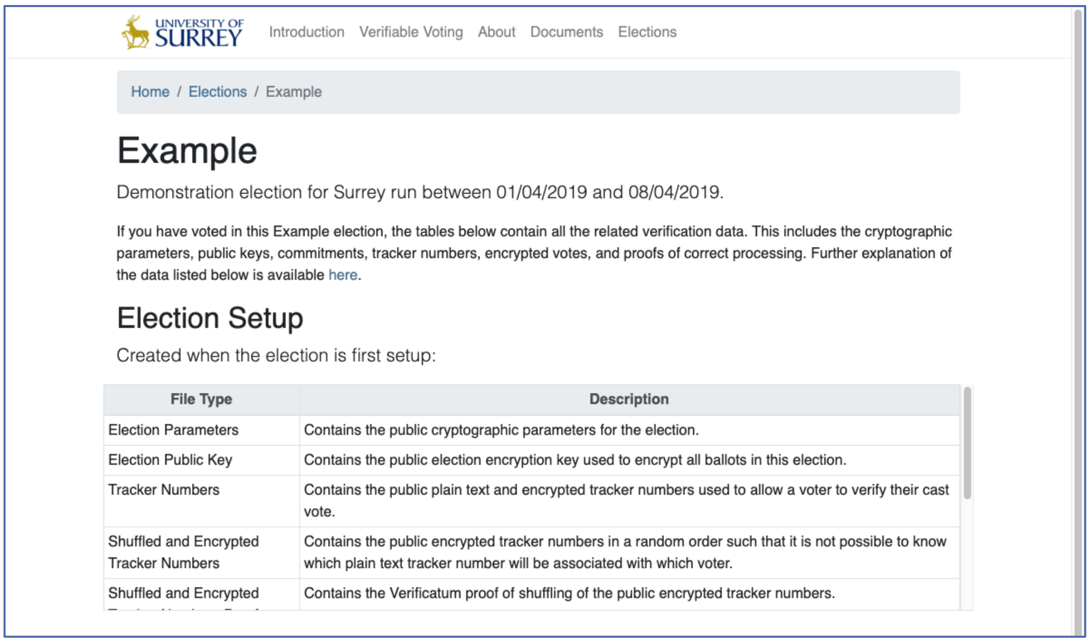
This example shows that one election has been published. Each page containing data, such as a list of published elections, contains a description of the content at the top, controls to modify the sort order or page of the data then the list of items. Here, the “Example” election shows details of whether data has been published for the election, whether vote verification is available, and the time at which the data was last updated. Selecting “View Verification Data” shows the list of data which has been published for the election. This page can also be accessed directly from the home page by selecting one of the recent elections.

An election’s Verification Data page shows the list of election data which can be publicly viewed, as shown in Figure 4. Three tables are displayed corresponding to different stages of the election: election setup, pre-election voter setup and post-election. Each entry in a table represents one of the files which has been uploaded to the VMV web service by an administrator and published to the distributed ledger. A brief description of each file is provided to give context.

Selecting one of the files will show one of two different pages depending upon the type of file. For data which can be browsed through record-by-record, a Browse page is shown. For data generated by Verificatum which can only be viewed by downloading the file, the Verificatum page is shown.

BROWSING CONTENT

The Browse page allows the user to look at all of the data contained in the file which was published. The data fields available in each file vary depending upon the type of file. For example, the “`public-election-params.csv`” file contains a single row with nine fields defining the cryptographic and election parameters, whereas the "`public-voters-keys.csv`" file contains one row per voter with two fields each containing the signature and trapdoor (encryption) public keys.



Home / Elections / Example

Example

Demonstration election for Surrey run between 01/04/2019 and 08/04/2019.

If you have voted in this Example election, the tables below contain all the related verification data. This includes the cryptographic parameters, public keys, commitments, tracker numbers, encrypted votes, and proofs of correct processing. Further explanation of the data listed below is available here.

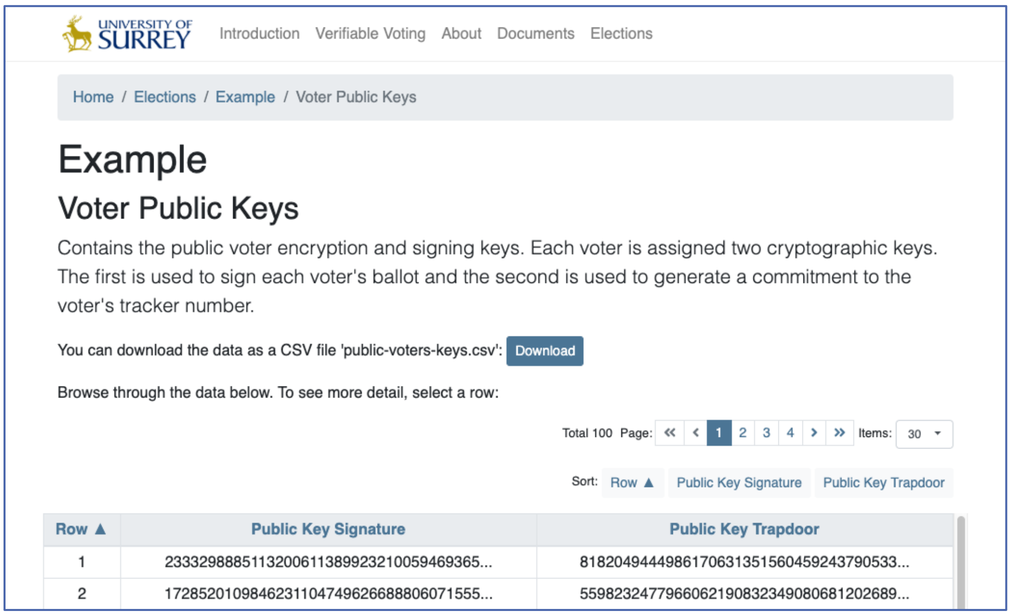
Election Setup

Created when the election is first setup:

File Type	Description
Election Parameters	Contains the public cryptographic parameters for the election.
Election Public Key	Contains the public election encryption key used to encrypt all ballots in this election.
Tracker Numbers	Contains the public plain text and encrypted tracker numbers used to allow a voter to verify their cast vote.
Shuffled and Encrypted Tracker Numbers	Contains the public encrypted tracker numbers in a random order such that it is not possible to know which plain text tracker number will be associated with which voter.
Shuffled and Encrypted	Contains the Verificatum proof of shuffling of the public encrypted tracker numbers.

FIGURE 4: VMV WEB SERVICE – ELECTION DETAILS

An example of a Browse page for a "`public-voters-keys.csv`" is shown in Figure 5. This shows an optionally longer description of the file together with a button which can be used to download the complete file content as published to the distributed ledger. Below this are the common controls used to page through the data or change the sort order. The sort order can also be changed by clicking on one of the column headings of the table displaying the data. Each row of the table shows the series of fields, with each truncated so that the data fits within the available screen space. To view the full set of data for a row, the row can be selected by clicking on it, and this will show the Content page.


UNIVERSITY OF SURREY
Introduction
Verifiable Voting
About
Documents
Elections
Download
Total 100 Page: « ‹ 1 2 3 4 › » Items: 30
Sort: Row ▲ Public Key Signature Public Key Trapdoor

Home / Elections / Example / Voter Public Keys

Example

Voter Public Keys

Contains the public voter encryption and signing keys. Each voter is assigned two cryptographic keys. The first is used to sign each voter's ballot and the second is used to generate a commitment to the voter's tracker number.

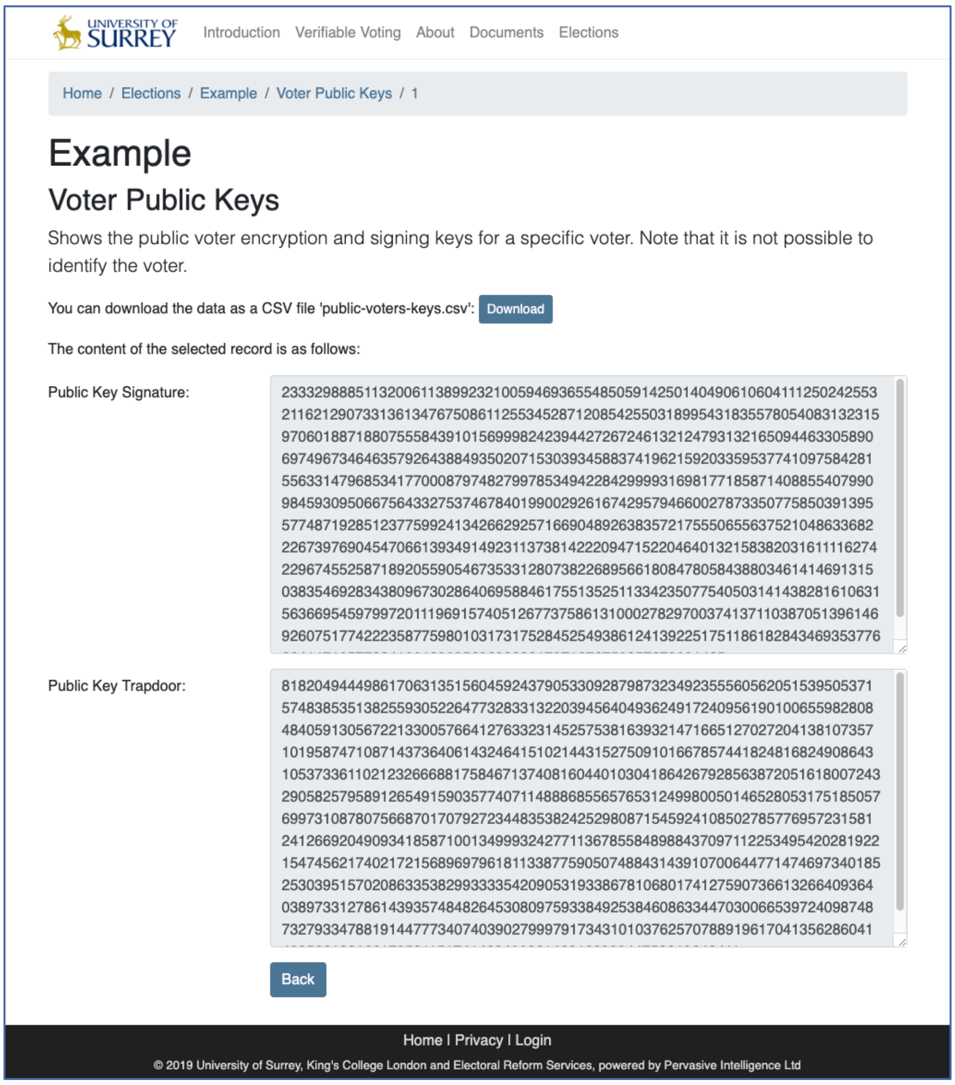
You can download the data as a CSV file 'public-voters-keys.csv':

Browse through the data below. To see more detail, select a row:

Row ▲	Public Key Signature	Public Key Trapdoor
1	2333298885113200611389923210059469365...	8182049444986170631351560459243790533...
2	1728520109846231104749626688806071555...	5598232477966062190832349080681202689...

FIGURE 5: VMV WEB SERVICE – BROWSE ELECTION FILE

An example of a Content page for a single row selected from a “`public-voters-keys.csv`” file is shown in Figure 6. This shows a description of the content and a button which can be used to download the whole file. Below this are then listed each of the fields with their content which can be viewed in full.



Example

Voter Public Keys

Shows the public voter encryption and signing keys for a specific voter. Note that it is not possible to identify the voter.

You can download the data as a CSV file 'public-voters-keys.csv':

The content of the selected record is as follows:

Public Key Signature:

```
23332988851132006113899232100594693655485059142501404906106041112502425553
2116212907331361347675086112553452871208542550318995431835578054083132315
9706018871880755584391015699982423944272672461321247931321650944633058890
6974967346463579264388493502071530393458837419621592033595377410975842811
5563314796853417700087974827997853494228429999316981771858714088554079900
9845930950667564332753746784019900292616742957946600278733507758503913955
5774871928512377599241342662925716690489263835721755506556375210486336822
2267397690454706613934914923113738142220947152204640132158382031611116274
2296745525871892055905467353312807382268956618084780584388034614146913155
0383546928343809673028640695884617551352511334235077540503141438281610631
5636695459799720111969157405126773758613100027829700374137110387051396146
9260751774222358775980103173175284525493861241392251751186182843469353776
```

Public Key Trapdoor:

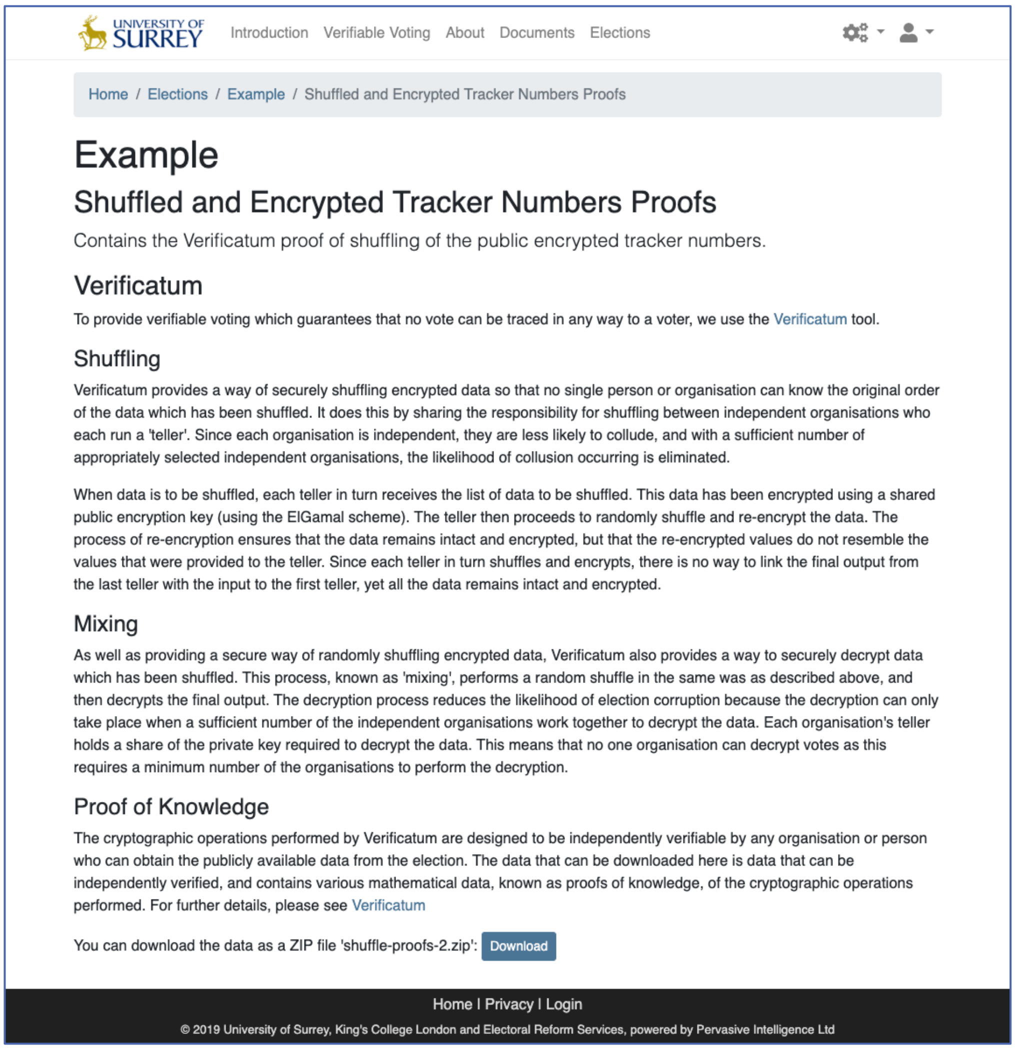
```
81820494449861706313515604592437905330928798732349235556056205153950537111
5748385351382559305226477328331322039456404936249172409561901006559828088
4840591305672213300576641276332314525753816393214716651270272041381073577
1019587471087143736406143246415102144315275091016678574418248168249086433
1053733611021232666881758467137408160440103041864267928563872051618007243
2905825795891265491590357740711488868556576531249980050146528053175185057
6997310878075668701707927234483538242529808715459241085027857769572315811
2412669204909341858710013499932427711367855848988437097112253495420281922
1547456217402172156896979618113387759050748843143910700644771474697340185
2530395157020863353829933335420905319338678106801741275907366132664093644
0389733127861439357484826453080975933849253846086334470300665397240987488
7327933478819144777340740390279997917343101037625707889196170413562860411
```

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 6: VMV WEB SERVICE – ELECTION FILE CONTENT

DOWNLOADING VERIFICATUM DATA

The Verificatum page allows the user to download a file which was produced from Verificatum which contains proof of knowledge data for a particular mix-net operation, including shuffling tracker numbers, decrypting commitments and mixing (shuffling and decryption) votes. Since this data is contained in binary files, all of the files for the proof of knowledge for a particular stage are placed into a single ZIP archive which can be downloaded. An example Verificatum page is shown in Figure 7, which shows a long description of what the file contains.


UNIVERSITY OF SURREY
Introduction
Verifiable Voting
About
Documents
Elections
Home / Elections / Example / Shuffled and Encrypted Tracker Numbers Proofs

Example

Shuffled and Encrypted Tracker Numbers Proofs

Contains the Verificatum proof of shuffling of the public encrypted tracker numbers.

Verificatum

To provide verifiable voting which guarantees that no vote can be traced in any way to a voter, we use the Verificatum tool.

Shuffling

Verificatum provides a way of securely shuffling encrypted data so that no single person or organisation can know the original order of the data which has been shuffled. It does this by sharing the responsibility for shuffling between independent organisations who each run a 'teller'. Since each organisation is independent, they are less likely to collude, and with a sufficient number of appropriately selected independent organisations, the likelihood of collusion occurring is eliminated.

When data is to be shuffled, each teller in turn receives the list of data to be shuffled. This data has been encrypted using a shared public encryption key (using the ElGamal scheme). The teller then proceeds to randomly shuffle and re-encrypt the data. The process of re-encryption ensures that the data remains intact and encrypted, but that the re-encrypted values do not resemble the values that were provided to the teller. Since each teller in turn shuffles and encrypts, there is no way to link the final output from the last teller with the input to the first teller, yet all the data remains intact and encrypted.

Mixing

As well as providing a secure way of randomly shuffling encrypted data, Verificatum also provides a way to securely decrypt data which has been shuffled. This process, known as 'mixing', performs a random shuffle in the same was as described above, and then decrypts the final output. The decryption process reduces the likelihood of election corruption because the decryption can only take place when a sufficient number of the independent organisations work together to decrypt the data. Each organisation's teller holds a share of the private key required to decrypt the data. This means that no one organisation can decrypt votes as this requires a minimum number of the organisations to perform the decryption.

Proof of Knowledge

The cryptographic operations performed by Verificatum are designed to be independently verifiable by any organisation or person who can obtain the publicly available data from the election. The data that can be downloaded here is data that can be independently verified, and contains various mathematical data, known as proofs of knowledge, of the cryptographic operations performed. For further details, please see Verificatum

You can download the data as a ZIP file 'shuffle-proofs-2.zip': Download

Home | Privacy | Login

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 7: VMV WEB SERVICE – VERIFICATUM DATA

VOTE STATUS

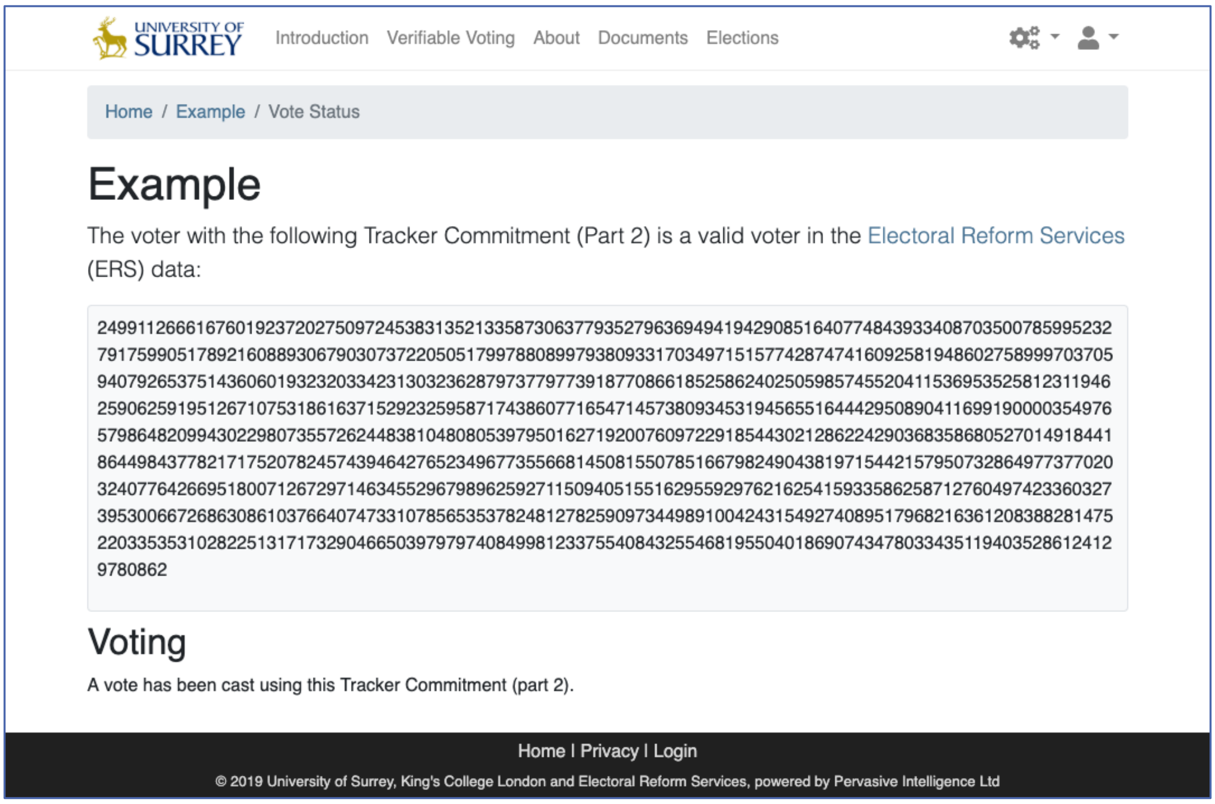
When an election and the associated voter information has been setup, and voters have received their corresponding β component, they may check the status of their vote. Using the β component and corresponding URL supplied to them by ERS via email, the user can determine whether their β component exists for the election, and whether it has been used to cast a vote (if votes have been published).

When a valid URL is used, which contains the corresponding election name and correct β component, then the Vote Status page is displayed as shown in Figure 8.

VOTE VERIFICATION

While the public pages provide one way in which public users can browse any of the published election data, for a particular election, a voter may also access the web service to verify their vote. This is achieved using their α and β components supplied by ERS via email once the election period has closed and the votes have been encrypted. The user can verify their vote using the URL supplied to them which contains their α and β components as described in the Voting section above. Alternatively, the user may just use their tracker number, if they know it.

When a valid URL is used, which contains the corresponding election name and correct α and β components, then the Verify Vote page is displayed as shown in Figure 9.


UNIVERSITY OF SURREY
Introduction
Verifiable Voting
About
Documents
Elections
Home / Example / Vote Status

Example

The voter with the following Tracker Commitment (Part 2) is a valid voter in the Electoral Reform Services (ERS) data:

```
249911266616760192372027509724538313521335873063779352796369494194290851640774843933408703500785995232
```

Voting

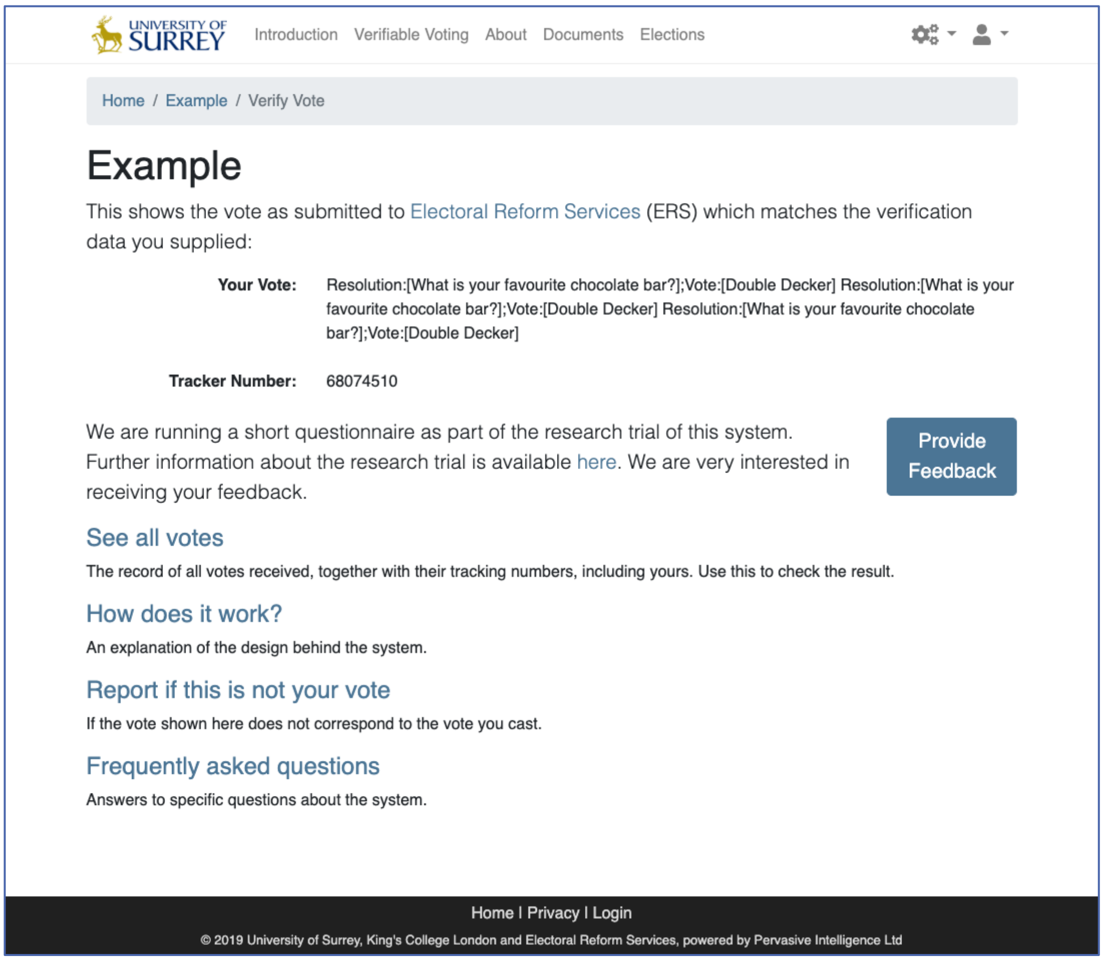
A vote has been cast using this Tracker Commitment (part 2).

Home I Privacy I Login
© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 8: VMV WEB SERVICE – VOTE STATUS

To obtain the plaintext vote and plaintext tracker number from the α and β components, the α and β components are treated as an ElGamal ciphertext and decrypted using the voter's private trapdoor (encryption) key to give the corresponding tracker number. The steps performed by the VMV web service to achieve this are:

1. Obtain the "Public Encrypted Voters" file for the election using the unique election name.
2. Use the β to look up the voter's public trapdoor key (*pk*) in the "Public Encrypted Voters" file.
3. Obtain the "Voters' Keys" file for the election using the unique election name.
4. Use the voter's public trapdoor key to look up the voter's private trapdoor key (*sk*) in the "Voters' Keys" file.
5. Obtain the "Public Election Parameters" file for the election using the unique election name.
6. Lookup the election *p* value in the "Public Election Parameters" file.
7. If all values have been obtained decrypt the (α, β) ElGamal ciphertext to obtain the tracker number in the group: $t_g = \beta\alpha^{p-1-sk} \mathrm{mod}\, p$.
8. Obtain the "Public Tracker Numbers" file for the election using the unique election name.
9. Use the tracker number in the group (t_g) to look up the plaintext tracker number (*t*) in the "Public Tracker Numbers" file.
10. Obtain the "Public Mixed Voters" file for the election using the unique election name.
11. Use the plaintext tracker number (*t*) to look up the plaintext vote (*v*) in the "Public Mixed Voters" file.



Example

This shows the vote as submitted to Electoral Reform Services (ERS) which matches the verification data you supplied:

Your Vote: Resolution:[What is your favourite chocolate bar?];Vote:[Double Decker] Resolution:[What is your favourite chocolate bar?];Vote:[Double Decker] Resolution:[What is your favourite chocolate bar?];Vote:[Double Decker]

Tracker Number: 68074510

We are running a short questionnaire as part of the research trial of this system. Further information about the research trial is available here. We are very interested in receiving your feedback.

See all votes

The record of all votes received, together with their tracking numbers, including yours. Use this to check the result.

How does it work?

An explanation of the design behind the system.

Report if this is not your vote

If the vote shown here does not correspond to the vote you cast.

Frequently asked questions

Answers to specific questions about the system.

Home | Privacy | Login

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 9: VMV WEB SERVICE – VERIFY VOTE WITH VOTE

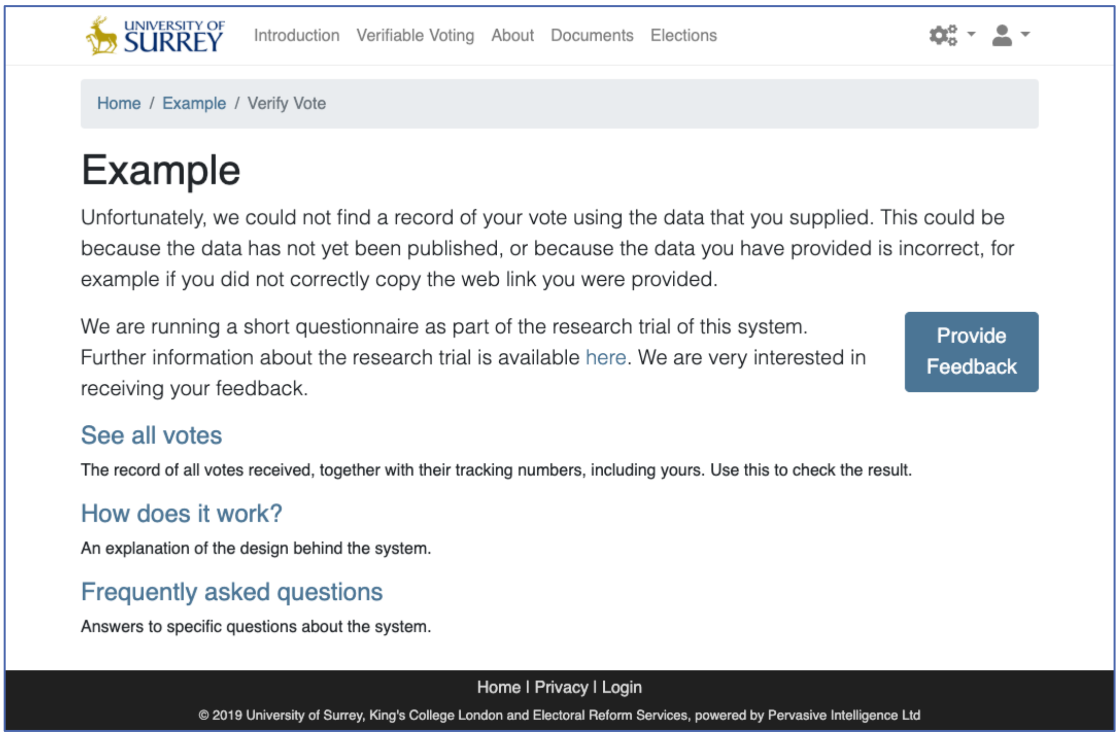
For these steps to be available, the “Election Parameters”, “Tracker Numbers”, “Encrypted Votes with Tracker Numbers” and “Final Votes” files must have been uploaded to the VMV web service and published, while the private “Voters’ Keys” file must have been uploaded.

Selecting the “See all votes” link presents the full plain text vote content for the election, with the corresponding vote record highlighted. Selecting “How does it work?” shows the Introduction page.

If the voter wishes to query the displayed vote, then they can report this to ERS via the “Report if this is not your vote” link. This shows the Report My Vote page, from which the voter can enter a reason for their report and upon submission, this will send an email to ERS containing the name of the election and the voter’s β value.

Selecting the “Frequently asked questions” link shows a list of pre-defined questions about verification. This also includes links to the public versions of the files used to verify the vote. These links can be used to browse the associated election data.

Assuming that all values are correctly obtained above, then the plaintext vote (*v*) and tracker number (*t*) are then displayed. If the values cannot be obtained, then an alternative message is displayed, as shown in Figure 10.



Example

Unfortunately, we could not find a record of your vote using the data that you supplied. This could be because the data has not yet been published, or because the data you have provided is incorrect, for example if you did not correctly copy the web link you were provided.

We are running a short questionnaire as part of the research trial of this system. Further information about the research trial is available here. We are very interested in receiving your feedback.

See all votes

The record of all votes received, together with their tracking numbers, including yours. Use this to check the result.

How does it work?

An explanation of the design behind the system.

Frequently asked questions

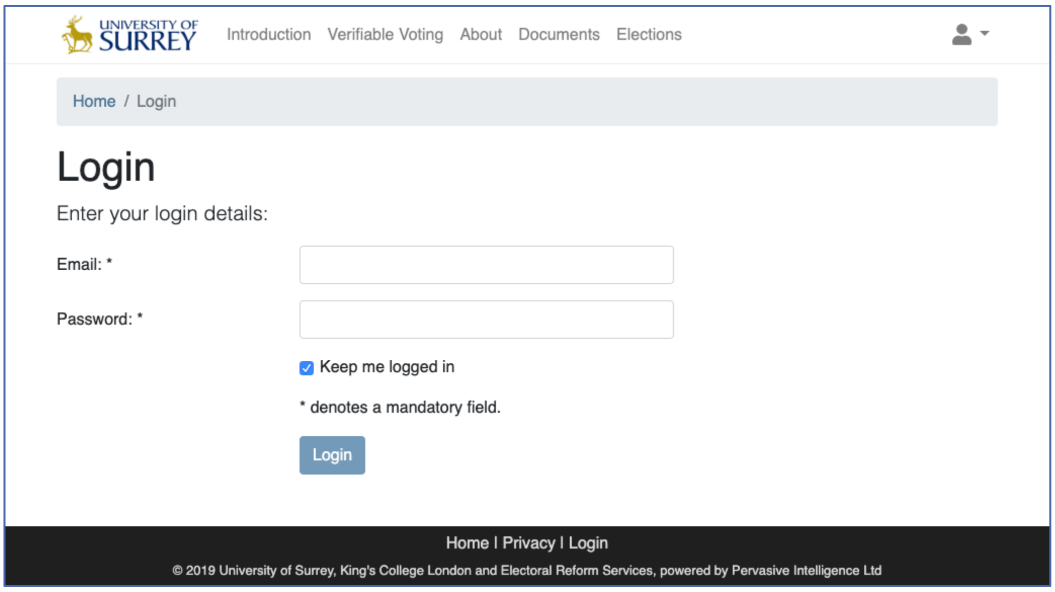
Answers to specific questions about the system.

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 10: VMV WEB SERVICE – VERIFY VOTE WITHOUT VOTE

ADMINISTRATOR ACCESS AND FUNCTIONS

Administrator access to the VMV web service can be achieved via the Home page URL https://vmv.surrey.ac.uk and selecting the "Login" link from the footer. Direct access to the Login page is via the https://vmv.surrey.ac.uk/login URL. The Login page allows an administrator to log into the service using their previously registered email address and password, as shown in Figure 11.



Login

Enter your login details:

Email: *

Password: *

☑ Keep me logged in

* denotes a mandatory field.

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 11: VMV WEB SERVICE – LOGIN

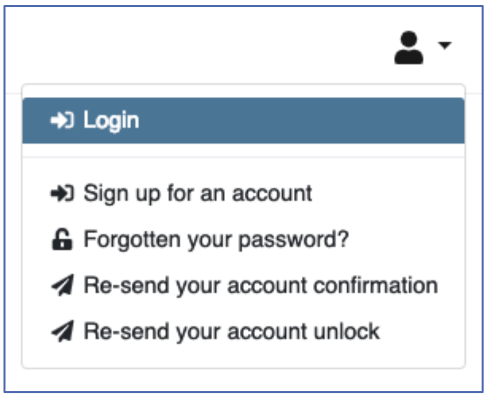

Login
Sign up for an account
Forgotten your password?
Re-send your account confirmation
Re-send your account unlock

FIGURE 12: VMV WEB SERVICE – ACCOUNT MENU

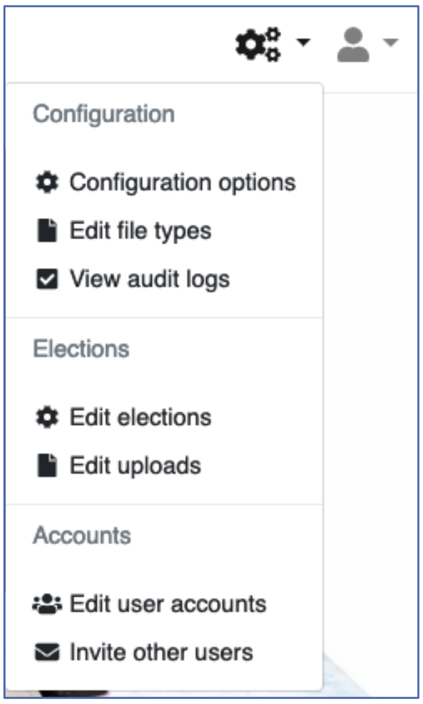


FIGURE 13: VMV WEB SERVICE – ADMINISTRATION MENU

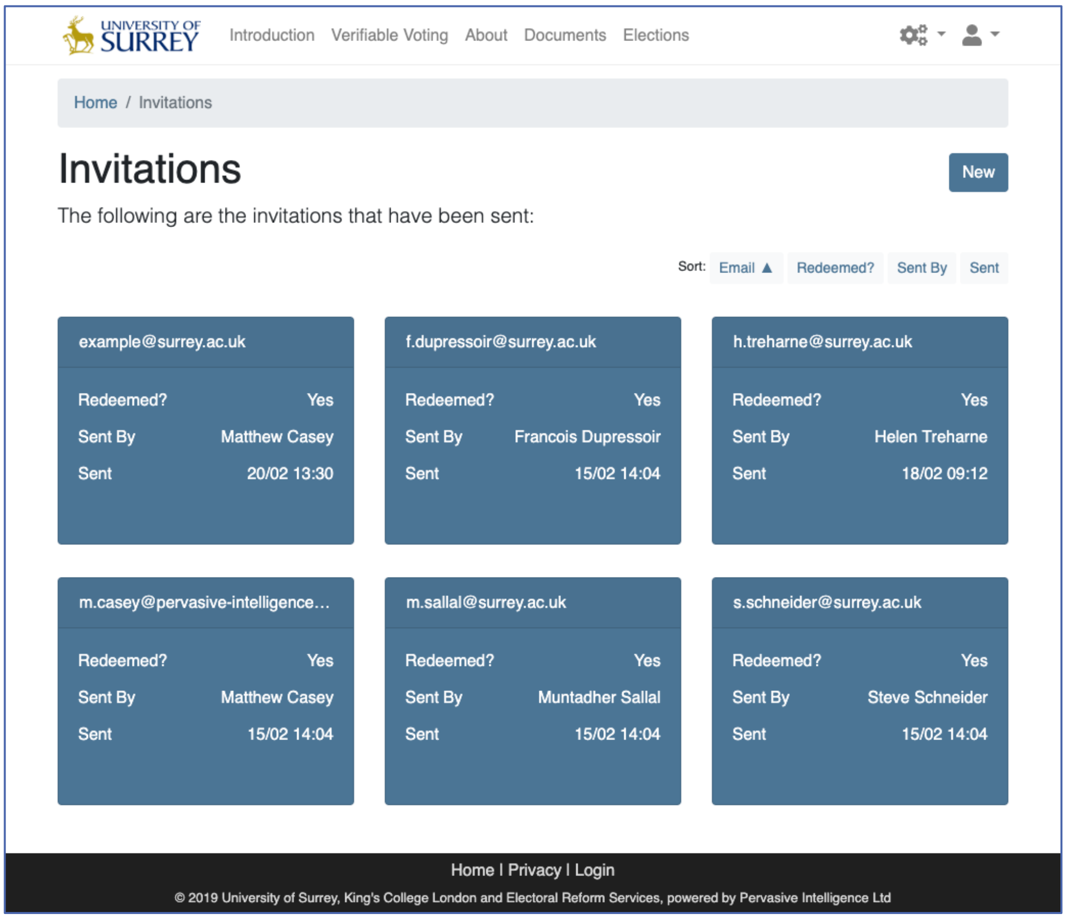
Displayed on the Login page is an extra header menu for account functions, as shown in Figure 12. This menu includes items which allow an administrator to sign up for an account, to reset their password if they have forgotten it, to re-send the account confirmation email, and to re-send the account unlock email.

Attempting to log into the service with either an incorrect email address or password will be rejected. If an incorrect password is used for a registered account three times, then the account will be locked, and an email sent to the user to unlock their account so that they can try again. A user can request this unlock email to be sent again via the account menu. If the user has forgotten their password, then they can request that an email is sent to their account providing details on how to reset their password. If an unregistered email address is used in any of the account functions, then the request is ignored.

Once logged into the service, administrators have access to an additional Administration header menu which enables them to configure the service, including file types and viewing audit logs, add or modify elections, add or modify uploaded files, and invite or manage administrator accounts as shown in Figure 13. Note that during login, users may optionally allow the browser to keep a record of their login for a period of 2 weeks. This is achieved by storing an encrypted cookie for the site within the browser.

SIGNING UP NEW ADMINISTRATORS

To become a registered administrator, an existing administrator must first send an invitation to allow the target user to sign up to the service. The system is setup with a number of existing administrators who can log into the service. To send an invitation, an existing administrator selects the "Invite other users" administration menu option to show the Invitations page. This will show the list of current invitations which have been sent (and optionally redeemed) as shown in Figure 14.



Invitations

The following are the invitations that have been sent:

Email	Redeemed?	Sent By	Sent
example@surrey.ac.uk	Yes	Matthew Casey	20/02 13:30
f.dupressoir@surrey.ac.uk	Yes	Francois Dupressoir	15/02 14:04
h.treharne@surrey.ac.uk	Yes	Helen Treharne	18/02 09:12
m.casey@pervasive-intelligence…	Yes	Matthew Casey	15/02 14:04
m.sallal@surrey.ac.uk	Yes	Muntadher Sallal	15/02 14:04
s.schneider@surrey.ac.uk	Yes	Steve Schneider	15/02 14:04

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

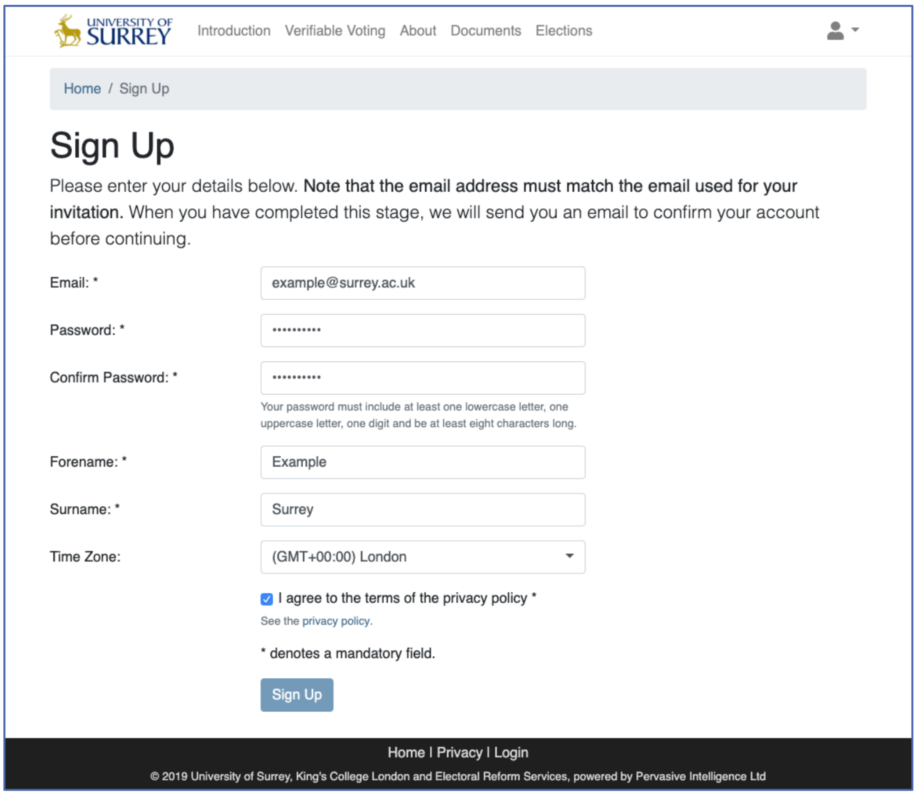
FIGURE 14: VMV WEB SERVICE – INVITATIONS

To create an invitation, select "New", enter the email address for the new administrator and press "Save". This will send an invitation email to the new user and return to the Invitations page.

Any invitations which have not yet been redeemed (used to create an account) by the target user can be modified to change the email address or deleted to prevent the invitation from being used. This is achieved via the icons shown in the top right-hand corner of the corresponding box for the invitation.

When a new user receives an invitation email, they may sign up to access the service. While the email contains the link to access the Sign Up page, this can also be accessed via the account menu "Sign up for an account" item. The Sign Up page is shown in Figure 15.

To successfully sign up for an account, the user must enter the email address corresponding to their invitation, a password which has at least one lowercase letter, one uppercase letter and one digit, and which is at least eight characters long, confirmation of the password again, their forename and surname, and finally they must confirm they agree with the privacy policy (as shown in the Privacy page). The use also optionally set their time zone.



Sign Up

Please enter your details below. **Note that the email address must match the email used for your invitation.** When you have completed this stage, we will send you an email to confirm your account before continuing.

Email: *	example@surrey.ac.uk
Password: *	••••••••••
Confirm Password: *	••••••••••
	Your password must include at least one lowercase letter, one uppercase letter, one digit and be at least eight characters long.
Forename: *	Example
Surname: *	Surrey
Time Zone:	(GMT+00:00) London

✓ I agree to the terms of the privacy policy *

See the privacy policy.

* denotes a mandatory field.

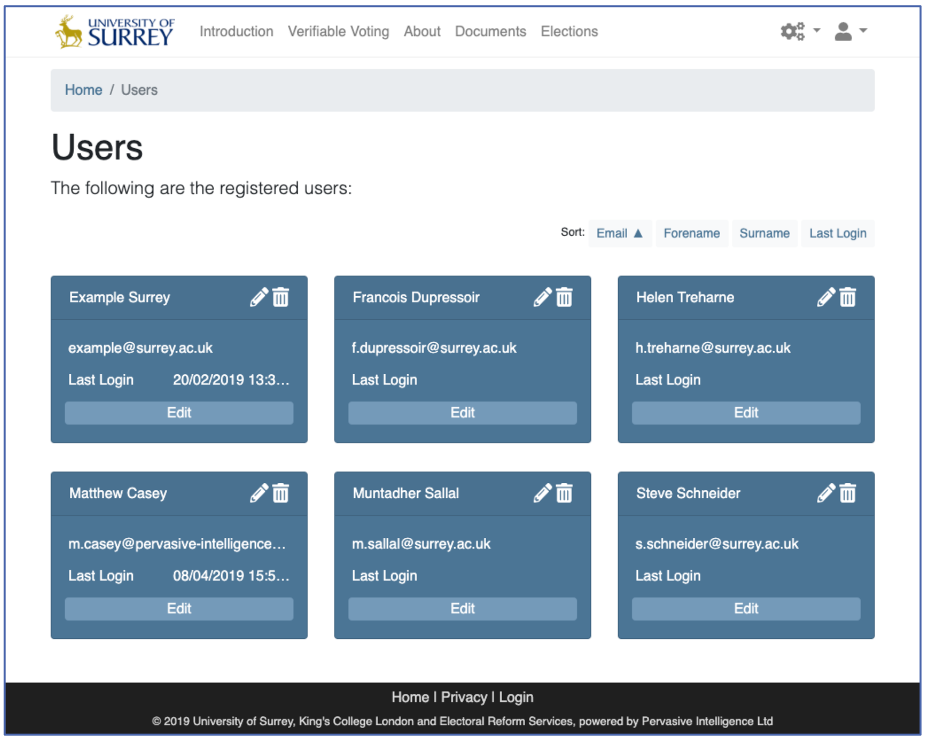
© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 15: VMV WEB SERVICE – SIGN UP

Once they have pressed "Sign Up" with all required and valid values, the service will send them a confirmation email. This email will contain a link which must be followed in order to confirm that their email address is valid. Once confirmed, the user may then log into the service via the Login page. If the user wants the confirmation email to be sent again to their email address, this can be achieved via "Re-send your account confirmation" from the account menu.

If at any time a user wishes to modify their account, this can be achieved via the "Edit user account" account menu item, which is displayed (along with "Logout") on the account menu for a logged in user. This Profile page allows the user to modify their email address (which will need confirmation again), password, forename, surname and time zone. The user may also cancel their account from here, which will remove their details from the service.

Restricted properties of a user account may also be edited by any administrator via the "Edit user accounts" administration menu item. This will show the list of registered administrators, as shown in Figure 16, from which a user can be selected and edited. Editing allows changes to the user's forename, surname and time zone. Accounts may also be deleted from here.



Users

The following are the registered users:

Example Surrey
example@surrey.ac.uk
Last Login 20/02/2019 13:3…
Edit

Francois Dupressoir
f.dupressoir@surrey.ac.uk
Last Login
Edit

Helen Treharne
h.treharne@surrey.ac.uk
Last Login
Edit

Matthew Casey
m.casey@pervasive-intelligence…
Last Login 08/04/2019 15:5…
Edit

Muntadher Sallal
m.sallal@surrey.ac.uk
Last Login
Edit

Steve Schneider
s.schneider@surrey.ac.uk
Last Login
Edit

Home I Privacy I Login
© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

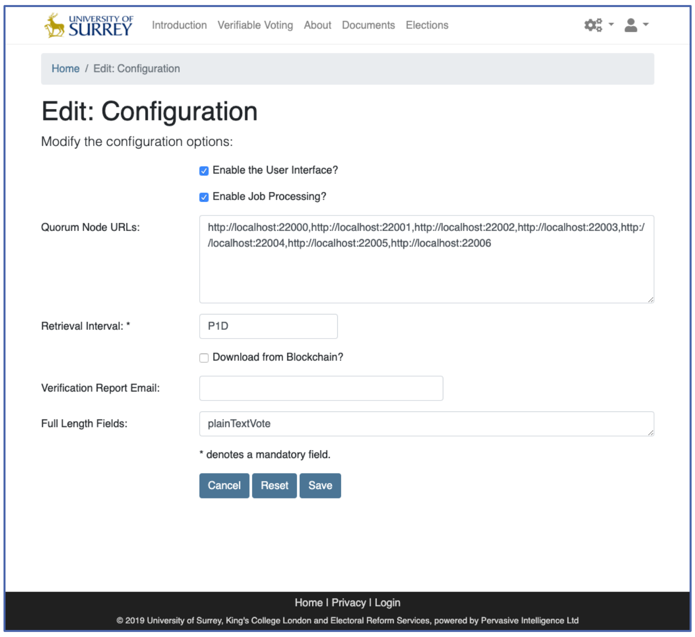
FIGURE 16: VMV WEB SERVICE – USERS

CONFIGURATION OPTIONS

Administrators may modify the overall service configuration by selecting the "Configuration options" administration menu item. This allows the configuration options to be changed as shown in Figure 17.

The configuration options are:

Enable the User Interface	Are public (unregistered) users allowed access to election data? Selecting this will allow public users to access all published elections and associated data. Not selecting this will turn off all public access to published data.
Enable Job Processing	When a file is uploaded to the service and published, it is committed to the distributed ledger. This process of committing and retrieving data from the distributed ledger is performed by background jobs. Selecting this option will enable these jobs, while not selecting this will cause all jobs to be queued pending re-selection of this option.



Edit: Configuration

Modify the configuration options:

[✓] Enable the User Interface?

[✓] Enable Job Processing?

Quorum Node URLs: http://localhost:22000,http://localhost:22001,http://localhost:22002,http://localhost:22003,http://localhost:22004,http://localhost:22005,http://localhost:22006

Retrieval Interval: * P1D

[ ] Download from Blockchain?

Verification Report Email:

Full Length Fields: plainTextVote

* denotes a mandatory field.

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

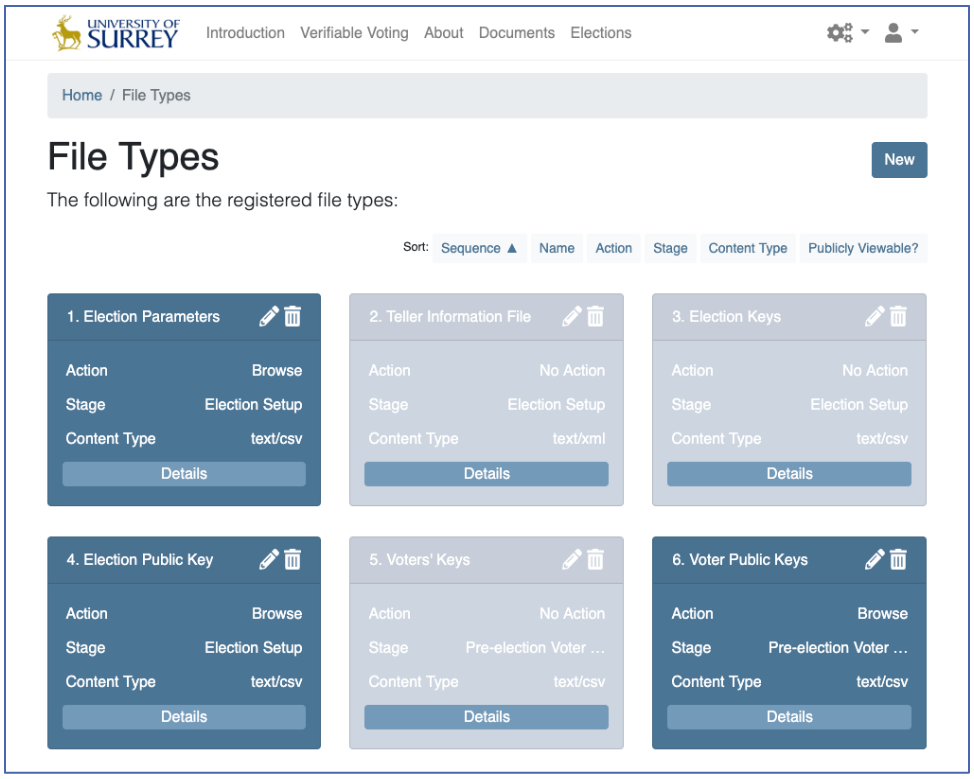
FIGURE 17: VMV WEB SERVICE – CONFIGURATION OPTIONS

Quorum Node URLs	To commit and retrieve data from the distributed ledger, the service must be configured with one or more URLs for each Quorum node in the cluster. When performing a commit or retrieval, the service will randomly pick one of the configured URLs to access the cluster. Each access to the distributed ledger will select a random URL.
Retrieval Interval	When committed to the blockchain, relevant data in CSV files is retrieved to populate the service cache of data which is viewed by public users. Since retrieving data is a time-consuming operation, the retrieval interval can be set so that this only occurs at the specified frequency for each published upload. The interval is defined in ISO8601 format so that, for example, “P1D” means one day, while “PT60M” means 60 minutes.
Download from Blockchain	When a public user requests the download of a file which has been published, the file can be retrieved from the service cache or directly from the distributed ledger. Selecting this option will force all such downloads to come directly from the distributed ledger, while not selecting it will use the cached version of the file.
Verification Report Email	When a voter wishes to report that their vote may be incorrect, this is the optional email address to which the report is sent.
Full Length Fields	When browsing an election file’s content which is displayed as a table, all of the displayed values will be truncated automatically so that the table fits nicely within the displayed page. However, for some fields, such as plain text votes, it is beneficial to be able to show the whole field without truncation. This can be achieved by entering the field name in this option, with multiple fields defined as a comma separated list.

Changes to the configuration options take effect immediately the “Save” button is pressed.

FILE TYPES

The service is pre-configured with all required file types which are needed either for publication or for vote verification. To view the list of configured file types, select "Edit file types" from the administration menu, which shows the list as in Figure 18.



File Types

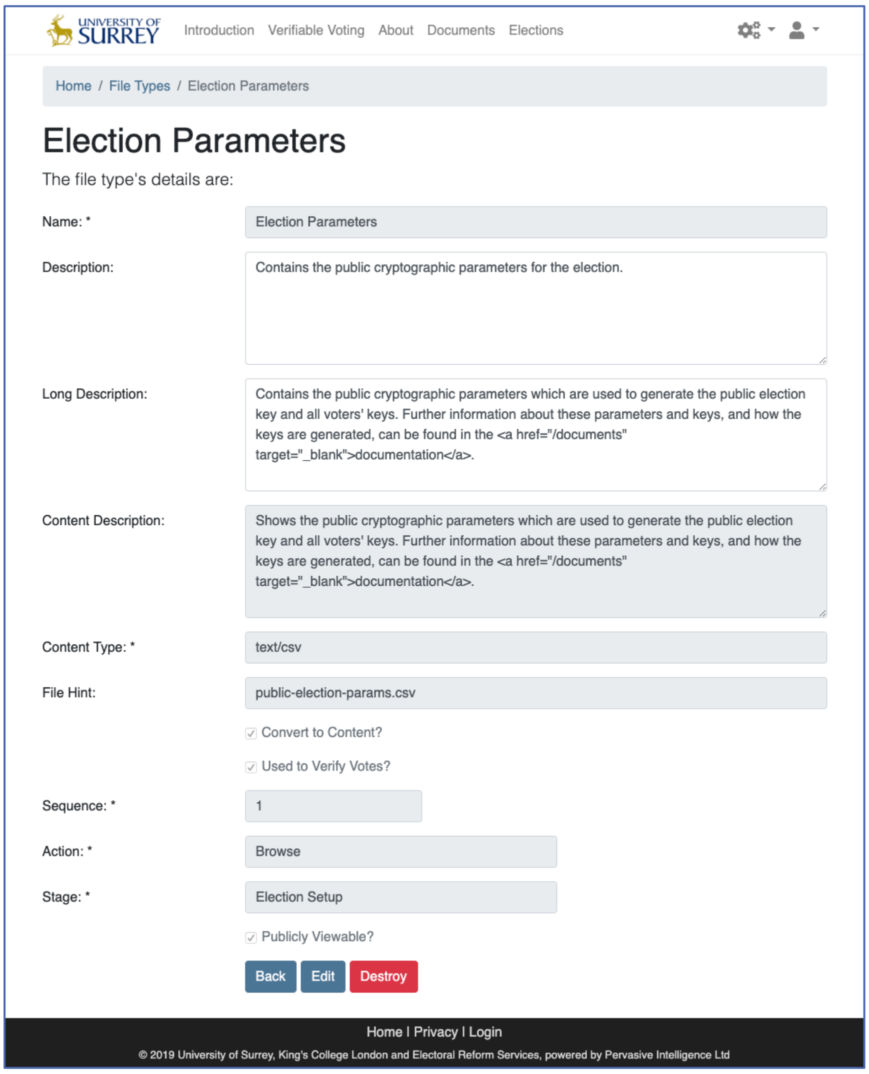
The following are the registered file types:

1. Election Parameters	Action: Browse	Stage: Election Setup; Content Type: text/csv
2. Teller Information File	Action: No Action	Stage: Election Setup; Content Type: text/xml
3. Election Keys	Action: No Action	Stage: Election Setup; Content Type: text/csv
4. Election Public Key	Action: Browse	Stage: Election Setup; Content Type: text/csv
5. Voters' Keys	Action: No Action	Stage: Pre-election Voter ...; Content Type: text/csv
6. Voter Public Keys	Action: Browse	Stage: Pre-election Voter ...; Content Type: text/csv

FIGURE 18: VMV WEB SERVICE – FILE TYPES

To view the complete configuration for a file type, select "Details" in the corresponding box to show the File Type page, an example of which is shown in Figure 19. Every file type has a number of associated properties:

Name	The name of the file type.
Description	The description of the file type displayed on an election's Details page.
Long Description	The optional description of the file type displayed on an election file's Browse or Verificatum page. If no long description is provided, then the description is used.
Content Description	The description of the file type displayed on an election file's Content page.
Content Type	The type of content as a MIME type. Used to restrict the type of file which can be uploaded (only certain browsers, such as Chrome, enforce this).
File Hint	A hint at the file name that is to be uploaded.
Convert to Content	Can the content of the file be converted into data which can be viewed via the Browse and Content pages?
User to Verify Votes	Is the file needed to verify votes?
Sequence	What is the typical sequence number associated with the file type when generated during an election. For example, since generation of the election parameters is the first step in an election, the file type associated with this has sequence one.



Election Parameters

The file type's details are:

Name: *	Election Parameters
Description:	Contains the public cryptographic parameters for the election.
Long Description:	Contains the public cryptographic parameters which are used to generate the public election key and all voters' keys. Further information about these parameters and keys, and how the keys are generated, can be found in the <a href="/documents" target="_blank">documentation</a>.
Content Description:	Shows the public cryptographic parameters which are used to generate the public election key and all voters' keys. Further information about these parameters and keys, and how the keys are generated, can be found in the <a href="/documents" target="_blank">documentation</a>.
Content Type: *	text/csv
File Hint:	public-election-params.csv
	✓ Convert to Content?
	✓ Used to Verify Votes?
Sequence: *	1
Action: *	Browse
Stage: *	Election Setup
	✓ Publicly Viewable?

Back
Edit
Destroy
Home | Privacy | Login
© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

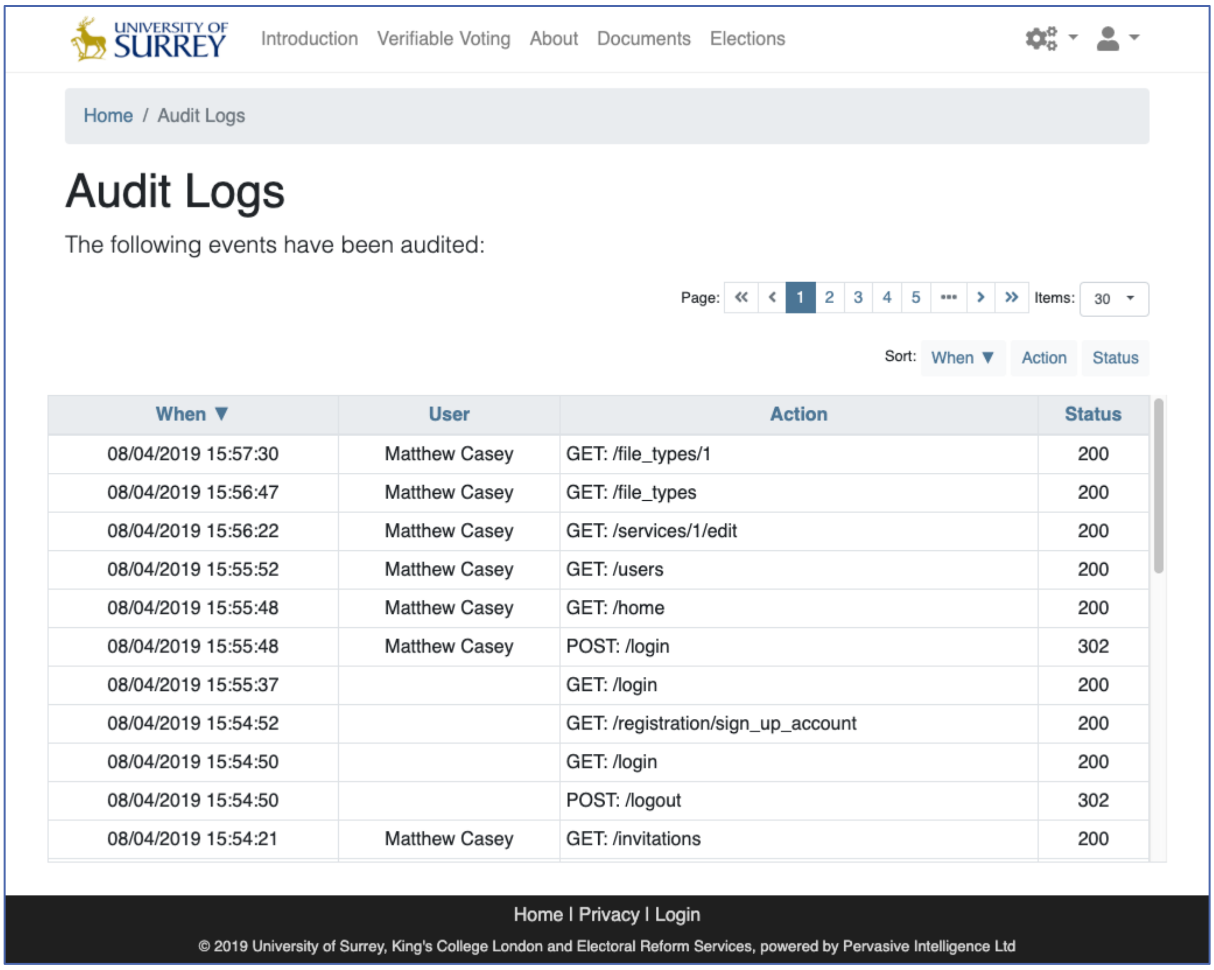
FIGURE 19: VMV WEB SERVICE – EXAMPLE FILE TYPE

Action	What action is performed when a public user wants to view the content of the file. No action means that the file cannot be viewed, Browse will link the content to the Browse and Content pages, Verificatum will link the content to the Verificatum page, while Proof of Knowledge will also link the content to the Browse and Content pages as it refers to proof of knowledge data held in a CSV file.
Stage	Is the stage of the election that this file is associated with, for example during Election Setup. This changes where the file is displayed within the available sections of the election data.
Publicly Viewable	Is the content publishable? If this is not selected, then the file can never be made public and will not be committed to the distributed ledger. If selected, the file can be committed and published.

It is strongly recommended that these file types are not modified from their pre-configured state, however, the values can be changed by selecting “Edit” from the File Type page. The file type may also be destroyed, but this will only be allowed if there are no corresponding uploads.

AUDIT LOGS

All access to the service's advertised RESTful endpoints is audited to ensure that there is a record of activity, for example, recording verification of votes. Selecting "View audit logs" from the administration menu will show the audit logs in descending time order (most recent first), as shown in Figure 20.



Audit Logs

The following events have been audited:

Page: « ‹ 1 2 3 4 5 ••• › » Items: 30
Sort: When ▼ Action Status

When ▼	User	Action	Status
08/04/2019 15:57:30	Matthew Casey	GET: /file_types/1	200
08/04/2019 15:56:47	Matthew Casey	GET: /file_types	200
08/04/2019 15:56:22	Matthew Casey	GET: /services/1/edit	200
08/04/2019 15:55:52	Matthew Casey	GET: /users	200
08/04/2019 15:55:48	Matthew Casey	GET: /home	200
08/04/2019 15:55:48	Matthew Casey	POST: /login	302
08/04/2019 15:55:37		GET: /login	200
08/04/2019 15:54:52		GET: /registration/sign_up_account	200
08/04/2019 15:54:50		GET: /login	200
08/04/2019 15:54:50		POST: /logout	302
08/04/2019 15:54:21	Matthew Casey	GET: /invitations	200

Home | Privacy | Login

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 20: VMV WEB SERVICE – AUDIT LOGS

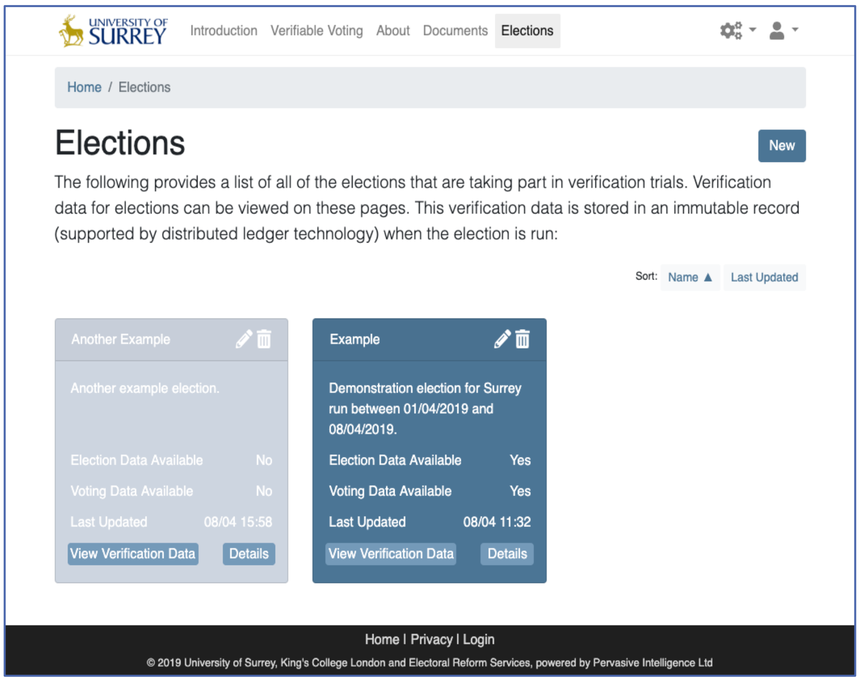
An audit log contains only four pieces of data: the time of the event, the optional user accessing the RESTful endpoint (blank if a public user), the action performed (the RESTful verb followed by the endpoint), and the corresponding HTTP status. Note that the action deliberately does not include the URL parameters passed to the endpoint to ensure that, for example, any vote verification using a voter's α and β are not recorded.

As an example, when a public user verifies their vote, an audit log will be recorded for the corresponding time with no user listed, and action of "GET: /verify" and a status of 200 if the α and β are valid and result in a vote being retrieved for verification. If, however, the α and/or β are invalid, a status of 202 will be recorded.

ELECTIONS AND UPLOADS

The core function of the service is to be able to upload and publish election data. This is achieved via the "Edit elections" and "Edit uploads" administration menu items.

The “Edit elections” menu item displays the Elections page, as shown to a public user for all published elections. The key differences are that an administrator will see all unpublished elections, be able to add new elections, modify or delete existing elections and access upload details for the election. An example is shown in Figure 21, which shows an unpublished election as a semi-transparent blue box together with the edit and delete controls and “New” button.



Elections

The following provides a list of all of the elections that are taking part in verification trials. Verification data for elections can be viewed on these pages. This verification data is stored in an immutable record (supported by distributed ledger technology) when the election is run:

Another Example

Another example election.

Election Data Available	No
Voting Data Available	No
Last Updated	08/04 15:58

Example

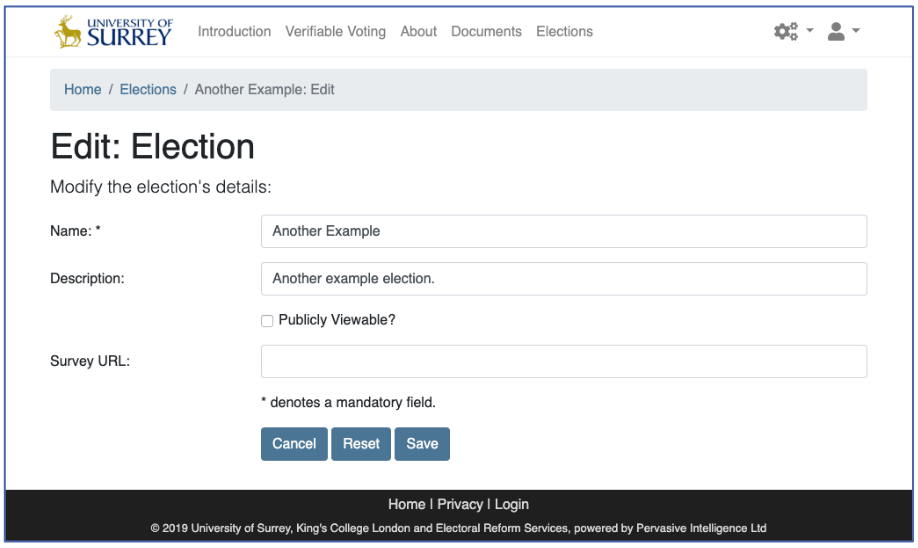
Demonstration election for Surrey run between 01/04/2019 and 08/04/2019.

Election Data Available	Yes
Voting Data Available	Yes
Last Updated	08/04 11:32

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 21: VMV WEB SERVICE – ELECTIONS FOR ADMINISTRATORS

Pressing “New” or editing an existing election shows the Election page as in Figure 22. This shows that the election name can be entered and whether the election is publicly viewable. Making an election publicly viewable allows it to be shown in the Elections page and viewed by public users. If an election is not publicly viewable, then no data for the election can be seen publicly, including any uploads which have been made publicly viewable.



Edit: Election

Modify the election's details:

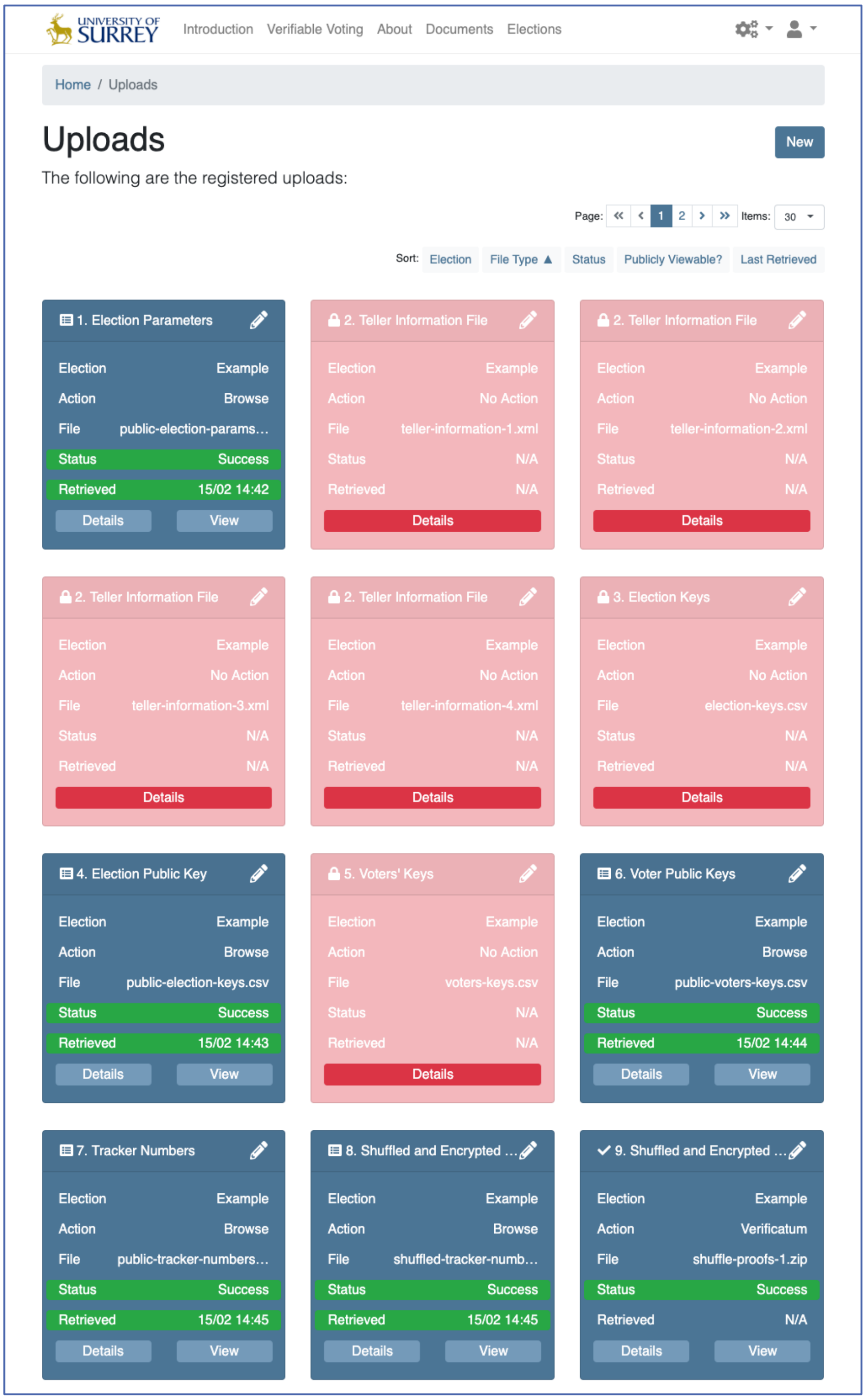
Name: *	Another Example
Description:	Another example election.
	☐ Publicly Viewable?
Survey URL:	

* denotes a mandatory field.

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 22: VMV WEB SERVICE – ELECTION

The optional description is used on the list of elections to help users identify each election. The optional Survey URL is used on the Verify Vote page. If a URL is provided, then the corresponding survey request is displayed. If no URL is provided, the request is not displayed.


UNIVERSITY OF SURREY
Introduction
Verifiable Voting
About
Documents
Elections
Home / Uploads
New
Page: « ‹ 1 2 › » Items: 30
Sort: Election File Type ▲ Status Publicly Viewable? Last Retrieved

Uploads

The following are the registered uploads:

1. Election Parameters

Election	Example
Action	Browse
File	public-election-params…
Status	Success
Retrieved	15/02 14:42

Details View

2. Teller Information File

Election	Example
Action	No Action
File	teller-information-1.xml
Status	N/A
Retrieved	N/A

Details

2. Teller Information File

Election	Example
Action	No Action
File	teller-information-2.xml
Status	N/A
Retrieved	N/A

Details

2. Teller Information File

Election	Example
Action	No Action
File	teller-information-3.xml
Status	N/A
Retrieved	N/A

Details

2. Teller Information File

Election	Example
Action	No Action
File	teller-information-4.xml
Status	N/A
Retrieved	N/A

Details

3. Election Keys

Election	Example
Action	No Action
File	election-keys.csv
Status	N/A
Retrieved	N/A

Details

4. Election Public Key

Election	Example
Action	Browse
File	public-election-keys.csv
Status	Success
Retrieved	15/02 14:43

Details View

5. Voters' Keys

Election	Example
Action	No Action
File	voters-keys.csv
Status	N/A
Retrieved	N/A

Details

6. Voter Public Keys

Election	Example
Action	Browse
File	public-voters-keys.csv
Status	Success
Retrieved	15/02 14:44

Details View

7. Tracker Numbers

Election	Example
Action	Browse
File	public-tracker-numbers…
Status	Success
Retrieved	15/02 14:45

Details View

8. Shuffled and Encrypted …

Election	Example
Action	Browse
File	shuffled-tracker-numb…
Status	Success
Retrieved	15/02 14:45

Details View

✓ 9. Shuffled and Encrypted …

Election	Example
Action	Verificatum
File	shuffle-proofs-1.zip
Status	Success
Retrieved	N/A

Details View

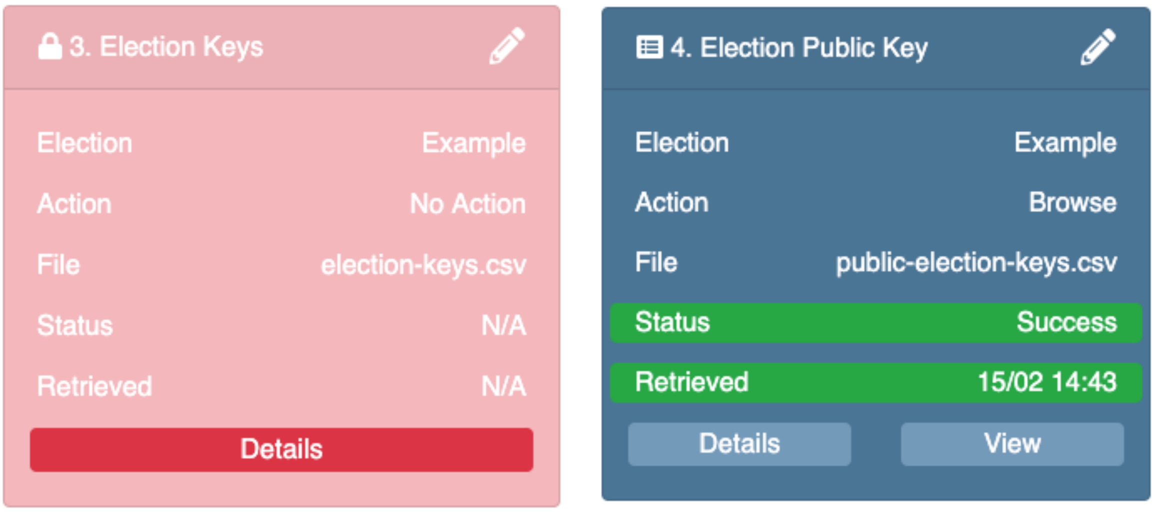
FIGURE 23: VMV WEB SERVICE – UPLOADS

Once an election has been created, then files can be uploaded for it and optionally published. This is achieved by accessing the "Edit uploads" administration menu item. This will show the Uploads page, and example of which is shown in Figure 23.

The uploads page allows an administrator to upload new files, modify existing uploads and monitor publication of files to the distributed ledger.

Uploads which have been made public are shown as blue boxes, while those which can be made public, but which have not yet been made public, are shown in semi-transparent blue. Uploads which can only be private are shown as red boxes to make it clear that these are private files only.

Each upload box shows details about the upload and its publication status, with private and public uploads showing different values as shown in Figure 24.



3. Election Keys

Election	Example
Action	No Action
File	election-keys.csv
Status	N/A
Retrieved	N/A

4. Election Public Key

Election	Example
Action	Browse
File	public-election-keys.csv
Status	Success
Retrieved	15/02 14:43

FIGURE 24: VMV WEB SERVICE – EXAMPLE UPLOADS

In this example, the left-hand upload is for the "`election-keys.csv`" file, which is held privately on the service, while the right-hand upload is the corresponding public version of the file "`public-election-keys.csv`" which can be published. Both files are produced by the "`create-election-keys`" command.

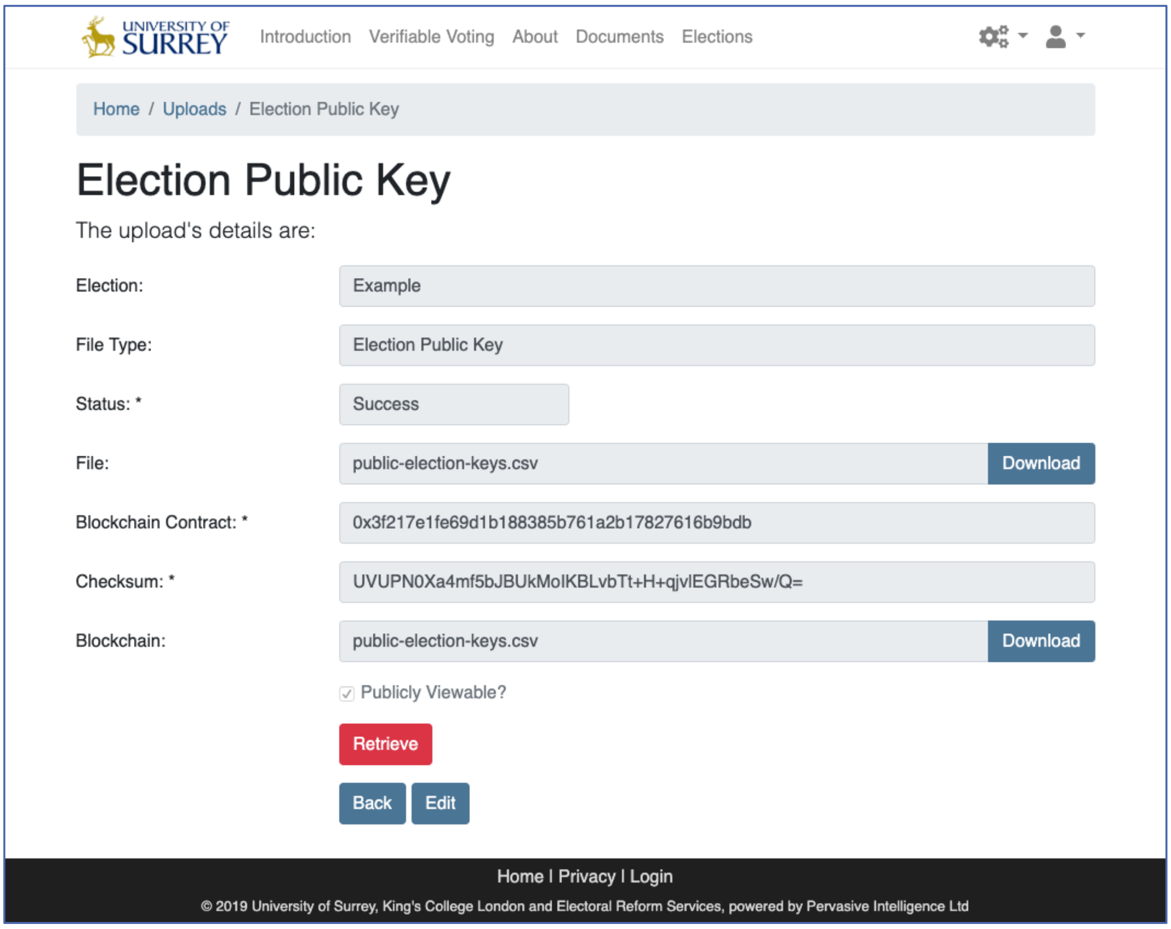
The top bar of each box shows the type of file and its corresponding sequence number, as defined for the file type. The icon is a visual indication of the type of file (for example, a padlock for a private file) and its associated action. The edit icon allows the upload to be edited. Below the top bar the details of the upload are shown:

Election	The election associated with the upload.
Action	The configured file type action for the type of upload. For example, "No Action" for private files, "Browse", "Verificatum" or "Proof of Knowledge" for publishable files.
File	The name of the file uploaded.
Status	The publication status of the file, if applicable.
Retrieved	The retrieval date and time of the file, if applicable.

The publication status is only applicable if the associated file type allows publication and if the upload has been set as publicly viewable. When updated to be publicly viewable, the file is committed to the distributed ledger using a background job. The status of this job is shown on the upload box. When the status is "Creating" the upload can be published but it has not yet been marked as publicly viewable. When "Pending" the associated file is being committed to the distributed ledger. When "Success" (and with a green background) the file has been successfully committed to the distributed ledger, but "Failed" (with a red background) indicates that the commit was unsuccessful and should be re-tried. Only when the status is "Success" will the upload become publicly viewable.

The retrieved date/time is when the file associated with the upload has been retrieved from the distributed ledger and cached so that it can be browsed. This is only relevant for file types which can have their content retrieved defined by the “Convert to Content” option on the file type. When the content can be retrieved, a periodic job (at an interval defined by the retrieval interval configuration option) will run which automatically downloads the content from the distributed ledger and caches it, replacing any previously cached version of the content (but not the original file which was uploaded). The retrieved date/time records when the content was last cached.

Selecting the “View” button where available for an upload box will show the publicly available Browse or Verificatum page for the upload. Selecting the “Details” button shows the Upload Details page, as shown in Figure 25, which lists all of the properties of an upload.



Home / Uploads / Election Public Key

Election Public Key

The upload's details are:

Election:	Example
File Type:	Election Public Key
Status: *	Success
File:	public-election-keys.csv Download
Blockchain Contract: *	0x3f217e1fe69d1b188385b761a2b17827616b9bdb
Checksum: *	UVUPN0Xa4mf5bJBUkMoIKBLvbTt+H+qjvIEGRbeSw/Q=
Blockchain:	public-election-keys.csv Download
	✓ Publicly Viewable?

© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

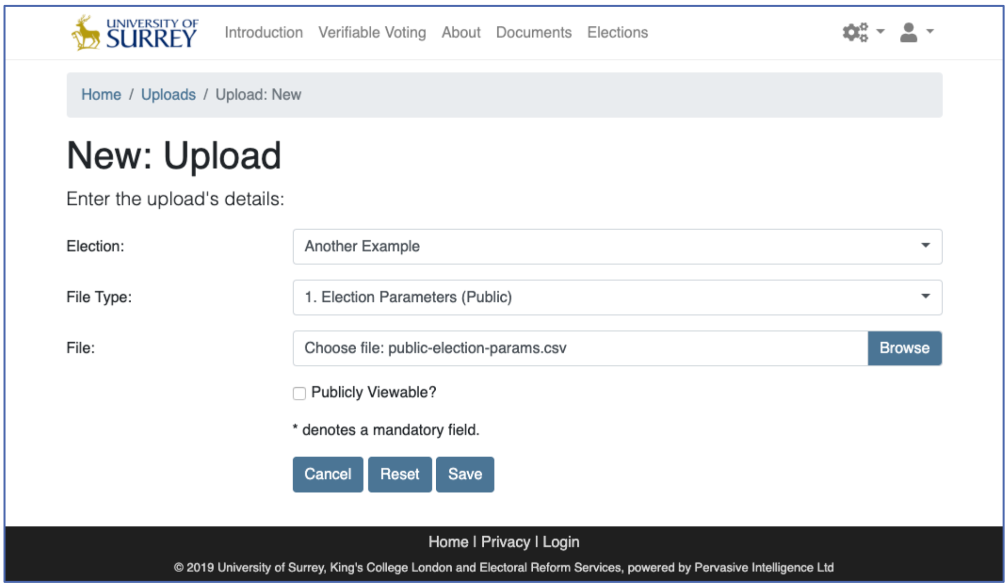
FIGURE 25: VMV WEB SERVICE – UPLOAD DETAILS

This shows the election name, file type, file (with a button which can be used to download the cached file) and publication status as before, together with additional fields for files which have been published to the distributed ledger:

Blockchain Contract	The contract address associated with the committed file.
Checksum	A SHA256 hash of the file. The hash is computed prior to committing the file to the distributed ledger, and then checked every time the file is retrieved from the ledger.
Blockchain	Allows the distributed ledger version of the file to be directly downloaded from the blockchain contract.

From this page, the upload can be edited by selecting the “Edit” button. Additional buttons for “Re-commit” and “Retrieve” may also be shown. The “Re-commit” button is shown if the file was attempted to be committed to the distributed ledger, but the commit failed. This allows the commit to be tried again. When successfully committed, and the file type allows the content to be retrieved, the file can also be retrieved again ahead of any periodic automatic retrieval by selecting the "Retrieve" button.

To create a new upload, the "New" button can be selected from the Uploads page. This shows the Upload page as in Figure 26. When editing an existing upload, this page is also shown with the existing upload's details.


UNIVERSITY OF SURREY
Introduction
Verifiable Voting
About
Documents
Elections
Home / Uploads / Upload: New

New: Upload

Enter the upload's details:

Election: Another Example

File Type: 1. Election Parameters (Public)

File: Choose file: public-election-params.csv

☐ Publicly Viewable?

* denotes a mandatory field.

Cancel
Reset
Save
Home | Privacy | Login
© 2019 University of Surrey, King's College London and Electoral Reform Services, powered by Pervasive Intelligence Ltd

FIGURE 26: VMV WEB SERVICE – UPLOAD

The Upload page allows the election and file type for the upload to be selected from the available list. Once these have been selected, the file can be chosen via the "Browse" button. If the file allows publication, the publicly viewable option may be selected. To upload the file, press "Save". Note that when editing an upload the file has already been selected and cannot be changed.

When saving, the file is first uploaded to the service and cached. If the upload details are correct, and the file has been marked as publicly viewable, a job to submit it to the distributed ledger is then queued. If the upload is created without being publicly viewable, the upload can be edited at a later point and marked as publicly viewable, at which point the corresponding commit job will be queued. Once published, the uploaded file cannot be removed from the distributed ledger, but it can be hidden from public viewing by unselecting the publicly viewable option.

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] M. Casey, "Trusted and Transparent Voting Systems: Verify My Vote Demonstrator Requirements and Design," 2018.

[3] 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.

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

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

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

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

[8] Oracle, "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for JDK/JRE 8 Download," 2018. [Online]. Available: https://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html. [Accessed October 2018].

[9] Software in the Public Interest, "Debian -- The Universal Operating System," 2018. [Online]. Available: https://www.debian.org/. [Accessed November 2018].

[10] 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].

[11] The PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open source database," 2019. [Online]. Available: https://www.postgresql.org/. [Accessed February 2019].

[12] Amazon Web Services, Inc., "Amazon Web Services (AWS) - Cloud Computing Services," 2019. [Online]. Available: https://aws.amazon.com/. [Accessed February 2019].

[13] Amazon Web Services, Inc., "AWS Elastic Beanstalk – Deploy Web Applications," 2019. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/. [Accessed February 2019].

[14] Amazon Web Services, Inc., "Amazon Relational Database Service (RDS) – AWS," 2019. [Online]. Available: https://aws.amazon.com/rds/. [Accessed February 2019].

[15] Amazon Web Services, Inc., "Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Simple Storage Service," 2019. [Online]. Available: https://aws.amazon.com/s3/. [Accessed February 2019].

[16] Amazon Web Services, Inc., "Amazon Simple Email Service (Amazon SES)," 2019. [Online]. Available: https://aws.amazon.com/ses/. [Accessed February 2019].

[17] Docker, Inc., "Enterprise Container Platform | Docker," 2019. [Online]. Available: https://www.docker.com/. [Accessed February 2019].

[18] National Institute of Standards and Technology, "FIPS PUB 186-4 Digital Signature Standard (DSS)," 2013.

[19] M. R. Albrecht, J. Massimo, K. G. Paterson and J. Somorovsky, "Prime and Prejudice: Primality Testing Under Adversarial Conditions," in *Proceedings of the 25th ACM Conference on Computer and Communications Security*, 2018.