An overview of the iVote 2015 voting system

Ian Brightwell[1], Jordi Cucurull[2], David Galindo[2] and Sandra Guasch[2]

[1] New South Wales Electoral Commission, Australia
[2] Scytl Secure Electronic Voting, Spain

Abstract. The Australian state of New South Wales (NSW) held their State General Election (SGE) on 28th March 2015. The iVote® electronic voting system was used at this election by eligible voters using telephones, smartphones or computers. Some 283,669 electors cast their vote through the iVote system, which set a new world record for the number of electors returning an electronic ballot for a binding parliamentary election. iVote operated in conjunction with the current paper system and by so doing strengthened the current system by providing two independent voting channel's which can be compared to identify if anomalies in count data exist that may effect the electoral outcome. iVote used in 2015 election followed from a previous system used at the preceding parliamentary election in 2011. The new system had the aim of improving voters' confidence that their preferences where captured and counted as they intended. This paper provides an introduction to iVote used in 2015 and the underlying cryptographic voting protocol.

Keywords: electronic voting, cryptographic protocol, vote privacy, cast-as-intended verifiability.

1 Introduction

On the 28th March 2015 all New South Wales (NSW) electors were required to vote at the State General Election (SGE). In this election a single candidate was elected for each of the 93 Legislative Assembly (lower house) districts, and 21 candidates for the state-wide Legislative Council (upper house). The Council election was for half of the chamber's 42 seats, the remaining seats would be elected at the next election in 2019. The possibility to vote remotely by electronic voting was provided for the second time at a general election by the iVote® system[1], thus enabling eligible electors to vote remotely using any suitable device or at official interstate venues. The iVote system was initially introduced in 2011 and allowed voters to cast their votes using Dual-Tone Multi-Frequency (DTMF) dialling over regular telephones or computers with suitable browsers and Internet access. iVote was initially implemented to satisfy the needs of the Blind and Low-Vision (BLV) community, who had won a court case which paved the way for the introduction of electronic voting. However,

[1] "iVote System" is a registered trademark of the NSW Electoral Commission.

the BLV community did not want a system which could only be used by them so the legislation also enabled other disadvantaged electors to use the system, for instance, those who live more than 20 km away from a polling place, those that were out of the state on Election Day, and those that could not attend a polling place due to a disability.

The motivation of the New South Wales Electoral Commission (NSWEC) for the introduction of the iVote system in NSW was:

(a) To assist electors who would otherwise not be able to vote independently or would have difficulty voting using existing channels.
(b) To assist electors who would, by virtue of location during the election period, not otherwise be able to vote reliably.
(c) To maintain confidence in the electoral process outcomes by reducing systemic errors for difficult to obtain or handle paper votes, improve counting accuracy (i.e. reduce counting, transcription and transposition errors), and to identify electoral anomalies by comparing electoral outcomes from two separate voting channels.

As a result of the success of iVote at the 2011 election, the NSW Joint Standing Committee on Electoral Matters (NSWEC's oversight body) supported the use of an improved version of iVote at the SGE 2015. At the core of iVote 2015 there is a new cryptographic voting protocol that is described in this paper. This design, through a combination of actions by the voter, NSWEC and third-party auditors, provides the following properties:

– the confidentiality of the vote is preserved (vote privacy);
– an elector's vote was cast as intended;
– all elector's votes cast are included in the final results for the election (recorded-as-cast, auditor verified);
– there is a reasonable probability all votes contributing to the election of candidates have not been mishandled or miscounted (alignment of voting patterns between separate voting channels);

The two main innovations for iVote 2015 with respect to the previous system are:

– voting options are encrypted at the voting terminal, thus offering end-to-end encryption;
– guarantees the cast-as-intended and recorded- as-cast properties.

These properties enable any voter to verify that their vote was received with the correct voter selections and was correctly recorded by the system.

The rest of this paper is devoted to presenting the NSWEC iVote 2015 system, its cryptographic voting protocol and data from the election. Section 2 gives an overview of iVote; while Sections 3 and 4 detail the Core Voting System, starting with the abstract workflow, followed by the cryptographic specification. Section 5 gives an account of the iVote use in the SGE election, and Section 6 presents the conclusions. Additionally, Appendix A presents a brief informal security analysis of some parts of the system; and Appendix B includes the description of the Zero-Knowledge Proofs used in the protocol.

2 iVote 2015 architecture

iVote protocol as used in 2015 is an expanded version of that used used in New South Wales in 2011. It offers three voting modes: remote DTMF phone voting, remote Internet voting and in-remote-venue Internet voting. The two Internet voting options use the same voting protocol, while DTMF phone voting uses a public switch telephone network (PSTN) phone connection in place of the Internet connection used for all other remote voting. The main difference between the two Internet voting modes is that in-remote-venue Internet voting takes place in a venue under the control of NSWEC, while remote Internet voting takes place at any location. See Tables 1 and 2 in Section 5 for more details on the voting types and devices used.

To understand some of the design choices behind iVote 2015, it is important to note that in the view of Commission, the risk of voter coercion is considered to be low in NSW [15]. Therefore coercion resistance is not a mandatory property for iVote. Not having to deal with voter coercion is a feature of the electoral process which gives the NSW Electoral Commission the opportunity to both provide iVoters with *preference*[2] receipts and publish all the preferences made by voters after the election.

The publication of all preferences allows any person with the appropriate skills to independently verify NSW's complex distribution of preference process without the need to review counting source code. The Commission believes that the confidence the public gains through this process far outweighs the risks associated with voter coercion. This is despite the NSW electoral process intrinsically giving voters infinite opportunity to enter a unique combination of preferences, which could allow them to prove how they voted, NSW does not consider this a significant issue.

iVote 2015 uses receipts from a verification component, called the Verification Server. This component allows voters to hear on any phone over the PSTN, at any time during the election, their vote after entering a given voter's

[2] Preference is a number marked on a ballot paper indicating the order of election for candidates.

unique credentials and voting receipt number. This gives voters the possibility of checking that their votes have been received by the system according to their intentions, and constitutes a defense against a potentially compromised voting device (particularly when the phone used for verification is different to the voting device). Nevertheless, should a voter be coerced they can revote and by so doing remove their original vote.

From the point of view of the voter, both with remote Internet voting and in-remote-venue Internet voting, casting and verifying a vote involves the following steps (see Figure 1):

1. **Voter registration**: Electors register online or through the iVote registration call centre. The voter introduces a PIN of *their* choice and, later, receives an iVote Number through a channel *different* from that of registration. This number and the PIN together constitute the voter's secret credentials, and will later be used to login into the iVote System and cast a vote in an election event.
2. **Vote casting**: The voters cast their votes by logging into the iVote system using either a computing device, including smartphones, or using a phone capable of DTMF dialling over PSTN. When a vote is cast, it is encrypted together with a *receipt number* randomly chosen, in the voter's device for Internet voting, or in the NSWEC servers for phone voting. *After* the vote has been cast, the receipt number is provided to the voter, and it can be used for a number of verifications. iVote uses checks that force receipt numbers to be unique.
3. **Cast-as-intended and recorded-as-cast verification**: The voter can check, up until the election closes, that their voting intent has been correctly registered by the system by calling the Verification Server. This server requests the voter's unique credentials (iVote Number and PIN), as well as the receipt number obtained after casting a vote. Then, the previously cast vote is opened and the contained voting options are read aloud to the voter by a text-to-speech server. Thus, the voter can verify the voting options read match their original intent.
4. **Vote decrypted verification**: At the end of the election, the receipt numbers obtained from the decrypted votes are uploaded to the Receipt Number Website. The receipt numbers allow the voters to check their votes were included in the count by searching their receipt numbers in the mentioned website, as they are unique per valid ballot.

The iVote ecosystem consists of three subsystems/processes, which are independently run, in order to distribute trust: first of all we find the Registration system (developed, hosted and supported by NSWEC), next the Core Voting

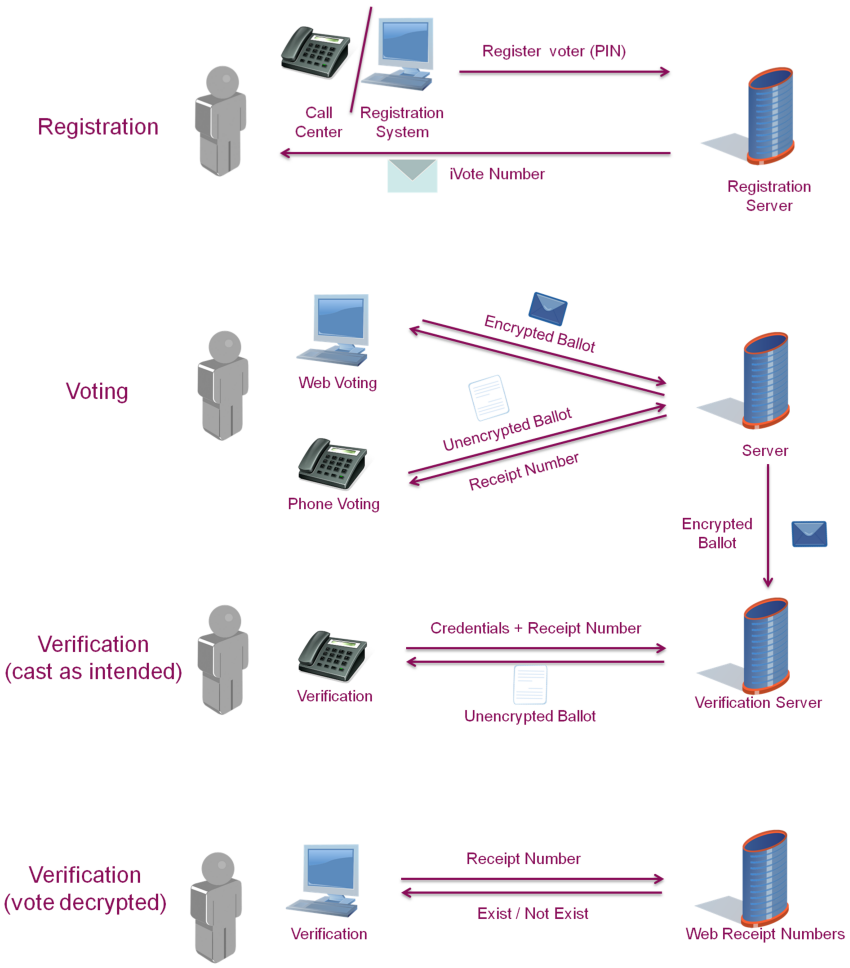

Fig. 1. Voter flow in iVote Core 2015 system

systems (developed and supported by Scytl, hosted by another contractor at arms length to NSWEC) and finally the Audit and Verification processes (developed and supported by NSWEC contractor, hosted by separate contractor at arms length to NSWEC). They are depicted in Figure 2, where different colors mean only accessible by different, distrustful parties, who can testify that the other party did not access certain systems.

3 The Core Voting Protocol - General Principles

We start by giving the cryptographic protocol workflow, and continue by giving a formal description of its most relevant algorithms.

In iVote a key roaming mechanism is used to provide digital certificates to voters when casting their votes. The digital certificate is protected by a secret that is derived from the voter's secret credentials. This secret is not stored in a remote database and therefore cannot be accessed to impersonate the voter. Because of the lack of space, we have omitted in this description the details of the cryptographic primitives involved in the key roaming mechanism, as it is common in most of the works in the literature on electronic voting [5, 8, 4].

iVote2015 is composed of several components (see Figure 2), which are operated by different parties in order to decrease the chances that these components fall under the control of a single entity. Using these components the main procedures of the election are as follows:

- **Election configuration**: This step is devoted to generating the election cryptographic keys, the servers' keys, the voter keys and the election configuration files. The election cryptographic keys use threshold cryptography techniques, thus they are divided in shares that are stored in smartcards given to the election representatives, i.e. to the Electoral Board. The generation of cryptographic materials is performed by the *Offline Voting Management System*, a component executed in an air-gapped server. Later, the public keys and configuration produced are uploaded to the *Online Voting Management System*, from which they are spread to the rest of the system.
- **Voter registration**: Voters register in the election using the *Registration System* by providing personal data, which is used to identify them as eligible voters, and the voters' secret PIN. After the final electoral roll (i.e. the list of elegible voters that registered in iVote) is available, the *Credential Management* component allocates an iVote Number that, together with the PIN, is linked to a pre-computed set of voter keys by the *Ballot Controller* component. The Credential Manager, which manages the electors for a given election event, will keep the electoral roll during the whole election. The

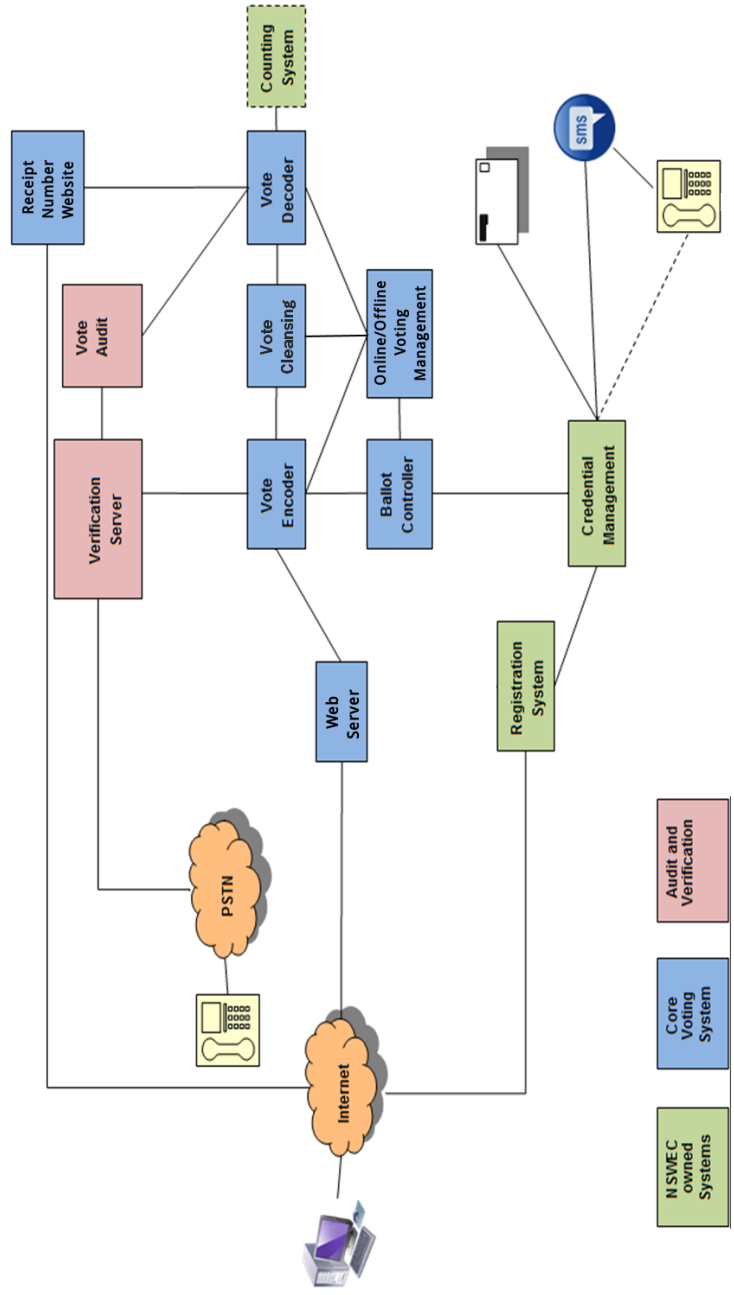

Fig. 2. iVote architecture

electoral roll links an actual voter with a unique unpredictable pseudonymous identity[3] cryptographically derived from the iVote Number and PIN. In that way, the *Core Voting System* is ignorant of the real identity of the voter.

– **Vote casting**: The voter authenticates with the *Vote Encoder* by using its iVote Number and PIN, and votes through the web or telephone using the *Web-Voting Client* or the *Voice-Voting Client*, respectively. The Web-Voting Client is used to vote through a computer, smartphone or tablet which is implemented as Javascript code that runs in the browser of the voter's device. The Voice-Voting Client is used to vote through a regular phone with DTMF capability. It is a Java code that runs in a NSWEC server. The first one encrypts the vote on the client-side, while the second one does it on the server-side. In the following lines we describe the workflow behind casting a vote (for the details see Section 4):

  - **Envelopes generation**: The voting client generates two envelopes containing the encrypted voting options: one is encrypted using the Electoral Board's public key (called *envelope for counting*), and the other one is encrypted using the Verification Server public key (called *envelope for verification*). The reason for the existence of two different envelopes is that they are decrypted differently: the envelope for counting is decrypted using the election's decryption key, while the envelope for verification is opened by the voter on inputs its secret credentials. These envelopes contain the encrypted Receipt Number. The voting client sends the hash of the Receipt Number to the Vote Encoder so it can check the uniqueness of the hash against those associated with the set of ballots. In the unlikely event that this Receipt Number hash is not unique in the ballot database, the envelopes generation phase is re-started anew.
  - **Proof generation**: The voting client generates a set of non-interactive zero-knowledge proofs (NIZKPs) to prove that:
    * both envelopes, when decrypted, contain the same voting options,
    * the ciphertexts have been freshly generated,
    * the Receipt Number contained in the envelopes matches the hash provided to the Vote Encoder.
  - **Vote casting**: The two envelopes, the proofs and a hash of the Receipt Number are sent to the Vote Encoder, which verifies the validity of the envelopes and proofs, and the eligibility of the voter. This is done by verifying a signature on the envelopes, which was computed by the vot-

[3] Referred to as id in the formal description in Section 3.

ing client using the voter's keys that the *Ballot Controller* assigned (at the registration stage) to the voter.
  - **Vote storage**: The Vote Encoder forwards the second envelope to the Verification Server, and stores the first envelope in a ballots' database. A control code calculated by the server is delivered together with the receipt number to the voter.

– **Vote verification**: In order to verify the content of their vote, the voter makes a phone call to an automatic system called *Verification Server*, providing their PIN and iVote number, together with the Receipt Number corresponding to the voter's ballot. With this information, the Verification Server is able to locate and decrypt the *envelope for verification*, and reveal the selected voting options to the voter. This service is not available once the election is finished.
– **Ballots export**: Once the voting phase ends, the *envelopes for counting* and the electoral roll are exported using the Online Voting-Management System to the Offline Voting-Management System, which is located in an air-gapped server.
– **Cleansing**: This operation validates the signatures of the ballots, authentication tokens proving the voters have legitimately authenticated themselves, timing values associated to the ballots and the eligibility of the corresponding voters. It also discards the votes that were cast with disabled credentials (credentials are disabled, for example, if it has been detected they have voted through another channel such as any vote which uses online mark-off like pre-poll or a postal voting; votes taken in a polling places can not be checked for mark-off as paper rolls are used currently). Finally, it separates the votes from their pseudonyms, in order to protect the voter's privacy at the decryption stage.
– **Decryption**: The Electoral Board come together, reconstruct the private election key, and decrypt the envelopes for counting using the *Vote Decoder* component to obtain the cleartext voting options and Receipt Numbers. NIZKPs of correct decryption are generated to prove the correctness of this process.
– **Audit**. The purpose of this procedure is two-fold:
  - to ensure that the voting options decrypted match the encrypted voting options in the Verification Server,
  - to verify that the decryption process has behaved honestly by validating the decryption NIZKPs.

  This audit entails re-encrypting the decrypted votes obtained from the Vote Decoder and comparing them with the ones stored in the Verification Server (once any information on the voters' pseudonyms has been removed). If

there are mismatches, the decryption NIZKPs are validated in order to ensure the decryption was correctly performed.
- **Publication of receipt numbers**: The decrypted Receipt Numbers are uploaded to the *Receipt Number Website* for public inspection.

4 Core Voting Protocol - Formal description

We start by describing the cryptographic building blocks used in the protocol. Next, we give the cryptographic protocol at the core of the iVote 2015 system.

4.1 Building blocks

We use the ElGamal IND-CPA cryptosystem in a given group $\mathbb{G}$, where the Decisional Diffie-Hellman assumption holds. This implies there exists efficient product and exponentiation operations, $\mathsf{prod} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ s.t. $\mathsf{prod}(g_1, g_2) := g_1 \cdot g_2$ and $\mathsf{exp} : \mathbb{G} \times \mathbb{Z} \to \mathbb{G}$ s.t. $\mathsf{exp}(g, x) := g^x$, respectively.

More specifically, we use the ElGamal's multi-ciphertext variant [10]; an authenticated symmetric encryption scheme $\mathcal{AE} = (\mathsf{Enc}, \mathsf{Dec})$; a key derivation function KDF that maps elements in $\mathbb{G}$ to one-time keys for $\mathcal{AE}$; an unforgeable signature scheme $\mathcal{S} = (\mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify})$; and several NIZK proof systems obtained through Maurer's zero knowledge proofs unification framework [13]. In the following lines we recall the syntax and main security properties of each of these blocks.

Definition 1 (ElGamal with multiple ciphertexts). *Let $\mathbb{G} = \langle g \rangle$ be a cyclic finite group, where g is a generator, and which has prime order q. The latter implies that $\mathbb{G} = \{1, g, g^2, \ldots, g^{q-1}\}$ as a set. The* ElGamal cryptosystem with multiple messages *is given by a tuple $\mathcal{E} = (\mathsf{EG.Keys}, \mathsf{EG.Enc}, \mathsf{EG.Dec})$ defined as follows:*

- $\mathsf{EG.Keys}(\mathbb{G}, n)$ *picks $x_1, \ldots, x_n$ uniformly at random in $\mathbb{Z}_q$ and sets $h_1 = g^{x_1}, \ldots, h_n = g^{x_n}$. The public encryption key is $\mathsf{pk} = (g, h_1, \ldots, h_n)$, while the secret decryption key is defined as $\mathsf{dk} = (x_1, \ldots, x_n)$.*
- $\mathsf{EG.Enc}(\mathsf{pk}, m_1, \ldots, m_n)$ *for messages $m_1, \ldots, m_n \in \mathbb{G}$ works a follows. It picks r uniformly at random in $\mathbb{Z}_q$ and outputs the ciphertext $C = (c_0, c_1, \ldots, c_n) = (g^r, h_1^r \cdot m_1, \ldots, h_n^r \cdot m_n)$.*
- $\mathsf{EG.Dec}(\mathsf{dk}, C)$ *for a ciphertext $C = (c_0, c_1, \ldots, c_n)$ outputs a tuple of n messages in $\mathbb{G}$ by computing $m_1 = c_1/(c_0^{x_1}), \ldots, m_n = c_n/(c_0^{x_n})$.*

It is easy to see that $\mathsf{EG.Dec}(\mathsf{dk}, \mathsf{EG.Enc}(\mathsf{pk}, m_1, \ldots, m_n)) = (m_1, \ldots, m_n)$ *for a legitimate key pair* $(\mathsf{pk}, \mathsf{dk})$. *Additionally, it is difficult to derive any information on any individual message if only their mutiple-encryption is known. Typical values of q are in the range $]2^{256}, 2^{257}[$.*

Definition 2 (Authenticated Encryption). *An* authenticated encryption *scheme for a key space $\mathcal{K} = \{0,1\}^\kappa$ and initialization vector space $\{0,1\}^\tau$ is given by a tuple $\mathcal{AE} = (\mathsf{Enc}, \mathsf{Dec})$ defined as follows:*

- $\mathsf{Enc}_K(m)$ *on a key $K \in \mathcal{K}$ and bit-string m, picks a so-called Initialization Vector IV uniformly at random in $\{0,1\}^\tau$ and outputs a ciphertext (IV, c).*
- $\mathsf{Dec}_K((IV, c))$ *outputs a bit-string m or an error symbol $\perp$.*

Any authenticated encryption scheme satifies that $\mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m$. Additionally, $\mathsf{Dec}_{K'}(\mathsf{Enc}_K(m)) = \perp$ with high probability when K is random and $K' \neq K$. A typical such encryption scheme is AES-GCM*, with $\kappa = 128$ and $\tau = 96$.*

For re-encryption purposes, $\mathsf{Enc}_K^{IV}(m)$ stands for encrypting the message m with a pre-existing fixed Initialization Value IV.

Definition 3 (Digital Signature). *A digital signature scheme is given by a tuple $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ defined as follows:*

- $\mathsf{KeyGen}(\lambda)$ *on input a security parameter λ outputs a signing key pair (vk, sk).*
- $\mathsf{Sign}(sk, m)$ *on input a bit-string m outputs a signature σ.*
- $\mathsf{Verify}(vk, m, \sigma)$ *on input a message m and a given signature σ, outputs 1 if they match, and 0 otherwise.*

Any digital signature scheme satifies that $\mathsf{Verify}(vk, \mathsf{Sign}(sk, m)) = 1$. Additionally, it is difficult to find a valid signature for a given message m and verification key vk without knowing the corresponding signing key sk. A typical signature scheme is FDH-RSA *with 2048 bit keys.*

Definition 4 (Hash function). *A hash function $\mathcal{H}$ mapping strings to natural numbers in the interval $[0, q-1]$. This means that:*

- *Computing $\mathcal{H}(m)$ is fast for any bit-string m.*
- *Given only $\mathcal{H}(m)$ it is infeasible to recover m.*
- *It is difficult to find $m_1 \neq m_2$ such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$.*

Typical hash functions are SHA-256 *and* SHA-512/224.

4.2 iVote Core Voting Protocol - A Formal Description

Formally, the core iVote 2015 voting protocol can be described using eight algorithms $\mathcal{V}^{\mathsf{iVote}} = (\mathsf{Setup}_{EB}, \mathsf{Setup}_{VS}, \mathsf{Credential}, \mathsf{Vote}, \mathsf{Validate}, \mathsf{VerifyVote}, \mathsf{Box}, \mathsf{Tally}, \mathsf{Verify})$ defined below:

$\mathsf{Setup}_{EG}(\mathbb{G})$ is run by the Electoral Board to create the election encryption and decryption keys. Starts by calling the multiple-ciphertext ElGamal key generator with $n = 3$. Next, chooses a random element $f \in \mathbb{G}$. The outputs of this algorithm are the election encryption and decryption keys as $\mathbf{pk}_{EB} \leftarrow (\mathbb{G}, q, h_1, h_2, h_3, f, \mathcal{H}, \mathcal{G}, \mathcal{AE}, \mathcal{S})$ and $\mathbf{sk}_{EB} = (x_1, x_2, x_3)$, respectively. In the election public encryption key, the tuple $(\mathcal{H}, \mathcal{G}, \mathcal{AE}, \mathcal{S})$ stands for the hash functions, authenticated encryption and digital signature concrete primitives to be used to encrypt a vote.

$\mathsf{Setup}_{VS}(\mathbf{pk}_{EG})$ is run by the Verification Server to create the verification encryption and decryption keys. Calls the multiple-ciphertext ElGamal key generator with $n = 2$. The outputs of this algorithm are the Verification Server encryption and decryption keys as $\mathbf{pk}_{VS} \leftarrow (\mathbb{G}, q, h_4, h_5, f, \mathcal{H}, \mathcal{G}, \mathcal{AE}, \mathcal{S})$ and $\mathbf{sk}_{VS} = (x_4, x_5)$, respectively.

$\mathsf{Credential}(\mathbf{pk}, id)$ generates a signing key pair for each voter using as input their pseudo-identity id. It outputs $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{SKeyGen}(1^\lambda)$. The public signing key upk is added to the electoral roll.

$\mathsf{Vote}\,(\mathbf{pk}, id, \mathsf{upk}, \mathsf{usk}, v)$ is run in the voter's device. This algorithm receives as inputs voter's pseudonym id and signing keys $(\mathsf{upk}, \mathsf{usk})$, and casts a ballot b corresponding to vote v. The ballot construction is a randomized procedure, which is initialized by choosing uniformly at random:

- $\mathcal{K}$ in the group $\mathbb{G}$, from which a symmetric encryption key is derived as $K = \mathsf{KDF}(\mathcal{K})$;
- RN, the receipt number[4], from the natural numbers smaller that 10^{10};
- $\mathcal{X}$, the Random eXtension, in the group $\mathbb{G}$, to protect against the Vote Encoder brute forcing the ballots to decrypt the vote preferences.

These values are used to create an *envelope for counting* and an *envelope for verifying* as follows:

(i) Encryption of $v \in \mathbb{V}$ for counting:

1. Compute the symmetric encryption of v and of RN as $\mathsf{Enc}_K(v)$ and $\mathsf{Enc}_K(RN)$.
2. Compute a commitment to RN as $\alpha = f^{\mathcal{H}(RN)}$.
3. Compute a multiple ElGamal encryption, with the Electoral Board encryption key, of $\mathcal{K}$, $\mathcal{H}(RN)$ and $\mathcal{X}$:
   - $A = g^r$.
   - $B = h_1^r \cdot \mathcal{K}$.
   - $C = h_2^r \cdot g^{\mathcal{H}(RN)}$.

[4] We note that the receipt number RN has ten decimal digits, since RN needs to be typed in a phone by dialling, in order to verify a vote.

– $D = h_3^r \cdot \mathcal{X}$.

The envelope for counting (to be stored at the Vote Encoder), is:

$$C_{count} = [\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN), A, B, C, D]\,.$$

(ii) Computation of a multiple ElGamal encryption, with the Verification Server encryption key, of $\mathcal{X}$ (the Random eXtension) and $f^{\mathcal{H}(RN)}$ (the commitment to RN) :
- $A = g^r$.
- $E = h_4^r \cdot \mathcal{X}$.
- $F = h_5^r \cdot f^{\mathcal{H}(RN)}$.

(iii) Computation of a one-time encryption of the pre-key $\mathcal{K}$, using $g^{\mathcal{H}(RN)}$ and $\mathcal{X}$ for the one-time key, as $G = g^{\mathcal{H}(RN)} \cdot \mathcal{X} \cdot \mathcal{K}$.

The envelope for verifying (to be stored at the Verification Server) is:

$$C_{ver} = [\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN),\ A, E, F, G]\,.$$

(iv) Proof generation for ensuring that the contents of the envelopes C_{count} and C_{ver} are consistent. For this purpose, the voting client generates five zero-knowledge proofs[5]:
- π_0: proves the knowledge of the randomness r, in order to ensure plaintext independence.
- π_1: proves that the product of B, C, D (from the envelope for counting) contains the encryption of $G = g^{\mathcal{H}(RN)} \cdot \mathcal{X} \cdot \mathcal{K}$ (from the envelope for verification).
- π_2: proves that D (from the envelope for counting) encrypts the same value than E (from the envelope for verification).
- π_3: proves that C (from the envelope for counting) encrypts the same value committed in F (from the envelope for verification).
- π_4: relates the receipt number used in the envelopes with its hash value $\mathcal{H}(RN)$ sent to the Vote Encoder.

(v) Sign the envelopes: $\sigma_1 = \mathsf{Sign}(\mathsf{usk}, C_{count}||\pi_0),\ \sigma_2 = \mathsf{Sign}(\mathsf{usk}, C_{ver})$.

(vi) Cast the ballot $b = (\mathsf{upk}, C_{count}, C_{ver}, \pi_0, \ldots, \pi_4, \mathcal{H}(RN), \sigma_1, \sigma_2)$.

$\mathsf{Validate}(\mathbf{pk}_{EB}, \mathbf{pk}_{VS}, b)$ is run by the Vote Encoder to decide whether a ballot b is valid. It performs several checks:
- the uniqueness of the receipt number through the implicit value $\mathcal{H}(RN)$.
- the signatures and the cryptographic proofs are valid.
- the encryptions $\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN)$ appear in both envelopes.

[5] Due to lack of space, we defer the reader to Appendix B for details of some of these proofs, which are in any case only of interest to specialists.

The successful verification of the proofs confirms that both envelopes would result in the same preferences being decrypted, and that the receipt number is unique in the ballots' database.

$\mathsf{Box}(\mathbf{pk}, \mathsf{BB}, b, id)$ is run by the Vote Encoder to update the ballot database BB with a valid ballot b. It parses ballot b as $(\mathsf{upk}, C_{count}, C_{ver}, \pi_0, \ldots, \pi_4, \mathcal{H}(RN), \sigma_1, \sigma_2)$ and BB is left unchanged if upk is not a legitimate verification key, or $\mathsf{Validate}(\mathbf{pk}, b)$ rejects, or id has previously stored a ballot in BB. Otherwise, it adds $(id, \mathsf{upk}, C_{count}, \pi_0, \sigma_1)$ as a new entry to BB and forwards $(id, \mathsf{upk}, C_{ver}, \sigma_2)$ to the Verification Server.

$\mathsf{VerifyVote}(\mathbf{pk}, id, RN, C_{ver}, \mathbf{sk}_{VS})$ is run by the Verification Server, which is an automatic telephone service, that reads aloud the voting options that a voter has cast to the ballot box database. When this algorithm is called, the voter with pseudonym id has previously authenticated himself to the service (by using his iVote number and PIN). The Verification Server decrypts, upon the request by a voter with pseudonymous identity id, their corresponding envelope $C_{ver} = [\mathsf{Enc}_{K'}(v'), \mathsf{Enc}_{K'}(RN'),\ A, E, F, G]$, as follows:

(1) Decrypts the multiple ElGamal encryption (A, E, F) from C_{ver}, using the decryption key $\mathbf{sk}_{VS}$. Let us call the resulting plaintexts are δ and θ.
(2) Checks the validity of the receipt number RN provided by the voter, by testing whether $\theta = f^{\mathcal{H}(RN)}$. If the comparison is not successful, an error notification is provided to the voter, and the process is aborted.
(3) Recovers $\mathcal{K}$ from C_{ver} using $g^{\mathcal{H}(RN)}$ and δ, and let us call $K = \mathsf{KDF}(\mathcal{K})$.
(4) Uses K to decrypt $\mathsf{Enc}_K(RN')$ and checks whether $RN = RN'$. If not, it provides an error notification.
(5) Finally, it uses K to decrypt $\mathsf{Enc}_K(v')$ and uses the result to play the audio rendition of the decrypted options to the voter.

$\mathsf{Tally}(\mathbf{pk}_{EB}, \mathsf{BB}, \mathbf{sk}_{EB})$ is run by the Electoral Board to obtain the voting options contained in the ballot database. For each entry $(id, \mathsf{upk}, C_{count}, \pi_0, \sigma_1)$ in BB:

(1) Validates π_0 and σ_1 and discards the entry if any rejection occurs.
(2) Checks that upk does not appear previously in the BB, and that upk is in the electoral roll. Discards the entry in case of any rejection.
(3) Parses C_{count} as $[\mathsf{Enc}_{K'}(v'), \mathsf{Enc}_{K'}(RN'), A, B, C, D]$, and computes the multiple ElGamal decryption of (A, B, C) using $\mathbf{sk}_{EB}$, obtaining $\mathcal{K}$ and $g^{\mathcal{H}(\overline{RN})}$, and a proof π_5 of correct decryption. Let $K = \mathsf{KDF}(\mathcal{K})$.
(4) Let $v = \mathsf{Dec}_K(\mathsf{Enc}_{K'}(v'))$ and $RN = \mathsf{Dec}_K(\mathsf{Enc}_{K'}(RN'))$.
(5) Checks that $g^{\mathcal{H}(\overline{RN})} = g^{\mathcal{H}(RN)}$. If not, discards v and RN.

The following lists are provided as the output of this algorithm:

- a list L_v of the (valid) cleartext votes v;
- a list L_{RN} of receipt numbers RN;
- a list L_A of tuples $\{IV, v, RN, K\}$;
- a list L_P of proofs of correct decryption π_5;
- a list L_C of envelopes that have caused any error.

An independent random permutation is applied to the elements in each list before they are released. The goal is to remove, as much as possible, any link between the list of votes and the list of envelopes for counting.

$\mathsf{Verify}(\mathbf{pk}_{EB}, \mathbf{pk}_{VS}, \mathsf{BB}_{VS}, L_v, L_{RN}, L_A, L_C)$ is run by the Auditors, to check whether the ballots stored in the Verification Server database contain the same list of voting options that was announced by the Electoral Board. The following checks are performed:

(i) Consistency between the lists L_v, L_{RN}, L_A, L_C output by Tally, and the database BB_{VS} of the Verification Server, is checked. Let the entries in BB_{VS} be in the format $[\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN),\ A, E, F, G]$. Then:

1. For each entry $\{IV', v', RN', K'\} \in L_A$, it computes the re-encryptions $\{\mathsf{Enc}_{K'}^{IV'}(v'), \mathsf{Enc}_{K'}^{IV'}(RN')\}$ and adds them to a list $L_{\widehat{C}}$.
2. For each entry C_{ver} in BB_{VS}, it checks that the corresponding encryptions $(\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN))$ appear in list $L_{\widehat{C}}$. Any pair of encryptions $(\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN))$ for which there is not such a correspondence must appear in the list L_C.

(ii) Correctness of the Tally algorithm:

1. For every entry $\mathcal{K}$ in L_P, it checks that $\mathsf{KDF}(\mathcal{K})$ belongs to L_A.
2. It computes $g^{\mathcal{H}(RN')}$ using the sub-entries RN' from list L_A, and checks that the resulting values belong to list L_P.
3. It verifies the proofs π_5 from list L_P using the corresponding values $\mathcal{K}$, $g^{\mathcal{H}(RN)}$, A, B, C, and the public keys h_1, h_2.

An informal security analysis is given in Appendix A and helps understanding the choice of some of the cryptographic building blocks of the protocol.

4.3 Comparison with Helios and Norwegian protocol

We proceed to a comparison of the cast-as-intended verification mechanisms of iVote 2015 with those of Helios and the Norwegian voting systems [1, 7, 14].

Helios uses the *immediate decryption* mechanism [2], where the voter's device encrypts a vote and the voter is allowed to challenge the encryption generated. In case they choose to challenge it, the device reveals the randomness

which was used to encrypt the voting options. After auditing the challenged encryption, the voting options are encrypted again with fresh randomness prior to casting the vote, so that the voter cannot use the randomness provided for audit as a proof to a third party of how they voted. This approach presents several drawbacks, such as usability (this randomness is a rather large string, cumbersome to be typed by a voter), and the fact that it does not allow for simple verification (i.e. verification must be done using a secondary computing device, under the assumption that at least one of the two devices is not compromised), and has also an impact on the efficiency of the vote casting phase (a vote is generally encrypted several times before casting).

The Norwegian protocol uses a different approach based on the so-called *return codes* [12, 11, 14]. In these proposals the voting client sends an encrypted vote to the remote voting servers, where return codes are calculated and sent back to the voter for verification. Voters possess a *verification card* where return codes are shown against matching voting options, and verification can be made by rather simple visual inspection. More concretely, return codes are computed from the probabilistic encryption of voting options, but at the same time they have to be deterministic: during the voting phase, the values computed by the server-side from an encrypted vote have to match those computed during the verification card generation phase. Therefore, the randomness from the voting options encryption has to be removed when computing the return codes, which poses a serious risk on the vote secrecy. This was solved in the Norwegian voting system [7, 14] by splitting the generation of the return codes in two independent entities, which were assumed not to collude. This approach is technically complex, and requires sending a verification card by postal mail to the voters (which implies infrastructure and operational costs).

The approach followed in iVote 2015 does not suffer from these drawbacks on efficiency, usability or infrastructure. Indeed, when a vote is cast, two different envelopes containing this vote are stored, in the Ballot Box (at the Vote Encoder) and in the Verification Server, respectively. The envelope in the Ballot Box can only be opened by the Electoral Board; while the envelope in the Verification Server can be opened by the voter when they enter their secret credentials and the random 10 digit receipt number. The simplicity of the iVote 2015 mechanism is due to the fact that coercion is not considered a real threat on the NSW State General election, and thus the existence of a mechanism that allows voters to recover their voting options during the lifetime of the election does not pose a privacy breach. In contrast, the Helios and Norwegian cast-as-intended verification mechanisms do not allow to retrieve the voter's voting option until after the end of the election, and thus the iVote 2015 approach does not comply with their requirement.

5 An account of iVote 2015 during the SGE 2015 election

Voters could register to use iVote 2015 from 12th February 2015 until about 2pm ESDT Saturday 28th March 2015. Voters could cast their vote from 8am ESDT Monday 16th March until 6pm ESDT Saturday 28th March 2015. Registration could be done from a phone to an operator or through a supported web browser (see Table 1).

	Registration device		
Registration method	Std phone	Smartphone	Computer
Remote over Internet using browser		✓	✓
Remote over PSTN talking to operator	✓	✓	⋆

Table 1. Methods and devices used for registering to iVote in 2015 (⋆ means the operator enters the registration into a computer using a similar interface to electors registering over the Internet)

Voting could be done from both a phone to an operator or a phone with DTMF capability or through a supported web browser on a smartphone or desktop/laptop (see Table 2).

	Voting device		
Voting method	Std phone	Smartphone	Computer
Remote over Internet using browser - receive iVote number by SMS or email		✓	✓
At interstate Venue over Internet using browser iVote number provided in the system		✓	✓
Remote over PSTN using DMF phone - iVote number by SMS or email or operator calling	✓	✓†	✓††
Remote over PSTN talking to an operator - receive iVote number by SMS or email	✓	✓	⋆

Table 2. Methods and devices used for voting with iVote in 2015 († voter could use smartphone as a telephone to vote but this is not recommended for security reasons; †† voter could use Skype or similar VOIP service vote but this is not recommended for security reasons)

Official voting statistics for voter participation in iVote 2015 can be found in Table 3. The statistics showed more than a 500% increase in the use of iVote from 2011, confirming voters' growing interest in the use of remote electronic voting channels. About 1.7% of the votes cast were verified, and no complaint

was made regarding the vote not matching the elector's intent. Additionally, it should be noted that the real failure rate of postal votes was over 600% higher than iVote in 2015 and consistent with the pattern observed in 2011. It can be concluded therefore that iVote is a more reliable channel for voting than postal voting and offers a greater level of certainty.

iVote	**SGE 2011**		**SGE 2105**	
	Votes	%	Votes	%
Registered but voted some other way	2,756	5.4%	10,827	3.6%
iVoted	46,864	91.7%	283,669	94.6%
Registered **but did not vote at all**	1,483	2.9%	5,394	**1.8%**
Accepted iVote registrations	51,103		299,890	

Postal Vote	**SGE 2011**		**SGE 2105**	
	Votes	%	Votes	%
Registered but voted some other way	34,709	11.0%	54,736	18.8%
Postal Voted	245,295	77.8%	203,577	69.9%
Registered **but did not vote at all**	35,178	11.2%	33,122	**11.4%**
Accepted Postal Vote registrations	315,182		291,435	

Table 3. Evolution of electronic voting acceptance in NSW from iVote 2011 to iVote 2015

While the voting period of iVote 2015 was still running, a team of researchers discovered a vulnerability [16], that potentially could allow a sophisticated attacker to alter the voting client code running on the voter's browser and modify their intended voting options. Roughly speaking, it could be described as a Man-in-the-Middle-Attack that leverages the FREAK vulnerability [3]. The latter was publically disclosed two weeks before the election and was patched on the core voting system servers but not on an associated Piwik monitoring server. According to [16], the attack works if a voter uses iVote from a malicious network (i.e., from a WiFi access point that has been infected by malware) which attackers are monitoring. The researchers advised CERT Australia of the vulnerability at 2 pm on the 20th of March. This was despite the fact they knew about the vulnerability several days earlier and had advised the media of this vulnerability during that period and prior to notifying CERT Australia. At around noon on the 21st of March the Electoral Commission changed iVote to disable the code which used the vulnerable server.

From the point of view of the integrity and secrecy of the vote, the reported attack's potential damage is similar to having malware installed in the voter's device. The latter possibility was already taken into consideration when designing iVote 2015, and the defence against it (regarding the integrity of the vote) was the inclusion of a verification mechanism using DTMF phones. Detection that a large scale attack was undertaken relied on the the absence of voters advising that their vote was not captured as intended after verification, which was the case for iVote in 2015. Also the voting pattern for iVoters did not deviate significantly from votes taken taken by other voting channels, again suggesting that a large-scale attack was not performed and indeed giving increased confidence that the votes were cast as intended and the correct candidates had been elected.

6 Conclusions

In this paper we have presented a general description of the inner workings of the iVote 2015 system, and we have introduced its new underlying cryptographic voting protocol. iVote 2015 was used in the New South Wales State General Election in March 2015, and received 283,669 votes, making it the biggest politically binding on-line election to date. iVote 2015 improved on the transparency, integrity and verifiability properties of the voting and counting processes with respect to the previous version run in 2011. In this paper we have presented the cryptographic design used to provide the main properties desired, namely: voter privacy, cast-as-intended verifiability and election integrity through audits from independent observers.

The NSWEC contends that the use of iVote in conjunction with current paper voting gives greater certainty in electoral outcome than in the past when only paper ballots were used. The addition of a completely separate voting channel, provided by iVote, allows a comparison of results between the electronic and paper channels which, when aligning appropriately, will improve certainty in the electoral outcome.

The iVote 2015 system does not support what is known as *universal verifiability* [9], which states that anyone can verify that all recorded votes are properly tallied. This is partly due to the choice of a symmetric authenticated encryption (AE) scheme to encrypt the votes, that is inherited from the previous iVote 2011 design. Using an AE scheme prevents the re-randomisation of encrypted votes that is normally needed to shuffle votes, which makes it harder to use the verifiable mix-nets technique that is in turn a building block for universally verifiable voting systems [17]. The use of a symmetric encryption scheme for encrypting the votes is motivated by the fact that in NSW, depending on the election, the

voting method known as "below the line" might imply choosing, in decreasing order, a few hundred candidates, which makes it impractical to encode the vote using a pure asymmetric encryption scheme (like ElGamal [6]). Another area of future research would consist on finding a practical vote encoding that would be supported by ElGamal.

It is envisaged that iVote will continue to replace postal voting and overseas venues and may be used in the future to take absent votes at all pre-polls and selected high volume polling places. This would mean iVote could take an extra 150,000 votes which would greatly reduce the large number of errors experienced with absent vote handling. The implementation of iVote for these situations would use a verifiable paper trail rather than remote voting's phone verification service. However, it is *not envisaged that iVote will replace in-district voting at polling places and pre-polls using paper ballots* which currently represents over 80% of the votes taken at a general election.

Acknowledgments

We thank Tim Batt and Sam Campbell for comments that have greatly helped this manuscript. In addition, we thank Richard Buckland, Annabelle McIver, Carroll Morgan and Roland Wen for their helpful comments on the exposition.

References

1. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348. USENIX Association, 2008.
2. Josh Benaloh. Simple verifiable elections. In Dan S. Wallach and Ronald L. Rivest, editors, *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*. USENIX Association, 2006.
3. Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. FREAK attack. `https://freakattack.com/`, 2015.
4. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for Helios under weaker trust assumptions. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2014.
5. Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.
6. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

7. Kristian Gjøsteen. The norwegian internet voting protocol. *IACR Cryptology ePrint Archive*, 2013:473, 2013.
8. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, pages 61–70. ACM, 2005.
9. Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.
10. Kaoru Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, volume 2274 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2002.
11. Helger Lipmaa. Two simple code-verification voting protocols. *IACR Cryptology ePrint Archive*, 2011:317, 2011.
12. Dahlia Malkhi, Ofer Margo, and Elan Pavlov. E-voting without 'cryptography'. In Matt Blaze, editor, *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda, March 11-14, 2002, Revised Papers*, volume 2357 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.
13. Ueli M. Maurer. Unifying zero-knowledge proofs of knowledge. In Bart Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2009.
14. Jordi Puigalli and Sandra Guasch. Cast-as-intended verification in norway. In Manuel J. Kripp, Melanie Volkamer, and Rüdiger Grimm, editors, *5th International Conference on Electronic Voting 2012, (EVOTE 2012), Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC, July 11-14, 2012, Castle Hofen, Bregenz, Austria*, volume 205 of *LNI*, pages 49–63. GI, 2012.
15. Rod Smith. Internet voting and voter interference - A report for the New South Wales Electoral Commission. `http://www.elections.nsw.gov.au/__data/assets/pdf_file/0003/118380/NSWEC_2013_Report_V2.0.pdf`, March 2013.
16. Vannesa Teague and J. Alex Halderman. Security flaw in New South Wales puts thousands of online votes at risk. `https://freedom-to-tinker.com/blog/teaguehalderman/ivote-vulnerability/`, 2015.
17. Douglas Wikström. A sender verifiable mix-net and a new proof of a shuffle. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*, pages 273–292. Springer, 2005.

A Informal security analysis

We would like to briefly address, on an informal basis, the security concerns that have guided the different cryptographic mechanisms that are included in iVote 2015 to provide cast-as-intended verifiability.

Let us recall the existence of two different envelopes that contain the voter's choices:

- envelope for counting: $C_1 = [\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN), A, B, C, D]$
- envelope for verification: $C_2 = [\mathsf{Enc}_K(v), \mathsf{Enc}_K(RN),\ A, E, F, G]$

$\mathcal{AE}_K = (\mathsf{Enc}_K(\cdot), \mathsf{Dec}_K(\cdot))$ stands for an authenticated encryption scheme with an initialization vector, in our case AES-GCM 128 with 96 bit nonces. Every encryption $c = \mathsf{Enc}_K(m)$ consists of three parts (e, t, IV), where: e is the proper encryption of the message m, from which inherits the length; t is the authentication tag, with 128 bits; and IV is a randomly chosen 96-bit nonce.

A first key observation is that it is infeasible to find c, K, K' such that both $\mathsf{Dec}_K(c)$ and $\mathsf{Dec}_{K'}(c)$ succeed, i.e. at least one these operations will output an error symbol $\perp$. This implies that, if $c = \mathsf{Enc}_K(m)$ was honestly generated, then $\mathsf{Dec}_{K'}(c) \neq \perp$ only if $K = K'$, with overwhelming probability. Therefore, every $\mathsf{Enc}_K(m)$ can be seen as a commitment to K. This is the basis of our Tally and Verify algorithms.

Secondly, as there are two envelopes stored in different databases, we need to ensure that they contain the same voting options v. Since efficient NIZKs for AES-GCM are not yet available, we choose to ensure that the opening key K that is encrypted in both envelopes is the same. This amounts to proving that the same K is encrypted in both envelopes. This is achieved by the NIZKs π_1, π_2 and π_3.

Thirdly, we use a random eXtension value $\mathcal{X} \in \mathbb{G}$ in $G = g^{\mathcal{H}(RN)} \cdot g^{RE} \cdot \mathcal{K}$, to prevent the Vote Encoder from computing $\mathcal{K}$ by brute-forcing RN (recall that, for usability purposes, RN is a 10 digits number, and thus has low entropy). It shall be noticed that more trust is placed in the Verification Server as regards to the secrecy of the vote, since this server could brute-force its own ballot database.

Finally, since the receipt number RN is also used to give evidence to the voters that their ballot was decrypted, we need to ensure that all RN's are unique in the Ballot Box. This is achieved by the voting client providing $\mathcal{H}(RN)$ to the Vote Encoder, and thus we need the NIZK proof π_4 to relate the encrypted RN to $\mathcal{H}(RN)$.

A future area of research is a formal cast-as-intended verifiability analysis.

B Zero-Knowledge Proofs

The proof descriptions and implementations follow the *Maurer's unified proofs* setting [13]. For simplicity in notation, we will denote $\mathcal{H}(RN) = J$, besides the existing A, B, C, E, F, G. Notation `String` means the constant string "String".

The operations are made in the ElGamal group $\mathbb{G}$ with q elements; $\mathcal{G}$ is a hash function mapping strings to $\mathbb{Z}_q$.

π_0: Transformation $\phi(r) : (g) \to (g^r) = (A) = b$

Proof:

- Compute w random in $\mathcal{Z}_q$.
- Compute $t = \phi(w) = (g^w)$
- Compute $c = \mathcal{G}(A, t, \texttt{SchnorrProof:VoterID} = id, \texttt{ElectionEventID} = eeID)$
- Compute $s = w + c \cdot r$

Proof is: s, c

Verification:

- Verify that s and c are between 1 and $q - 1$.
- Compute $\phi(s) = (g^s)$
- Compute $t' = \phi(s) \cdot b^{-c}$
- Compute $c' = \mathcal{G}(A, t', \texttt{SchnorrProof:VoterID} = id, \texttt{ElectionEventID} = eeID)$
- Check that $c' = c$

π_1: Transformation $\phi(r) : (g, \prod_{i=1}^{i=3} h_i) \to (g^r, \prod_{i=1}^{i=3}(h_i)^r) = (A, \frac{B \cdot C \cdot D}{G}) = (b_1, b_2)$

Proof:

- Compute w random in $\mathcal{Z}_q$.
- Compute $(t_1, t_2) = \phi(w) = (g^w, \prod_{i=1}^{i=3}(h_i)^w)$
- Compute $c = \mathcal{G}(A, \frac{B \cdot C \cdot D}{G}, t_1, t_2, \texttt{PlaintextProof})$
- Compute $s = w + c \cdot r$

Proof is: s, c

Verification:

- Verify that s and c are between 1 and $q - 1$.
- Compute $\phi(s) = (g^s, \prod_{i=1}^{i=3}(h_i)^s) = (z_1, z_2)$
- Compute $(t'_1, t'_2) = (z_1 \cdot b_1^{-c}, z_2 \cdot b_2^{-c})$
- Compute $c' = \mathcal{G}(A, \frac{B \cdot C \cdot D}{G}, t'_1, t'_2, \texttt{PlaintextProof})$
- Check that $c' = c$

π_2: Transformation $\phi(r) : (g, \frac{h_3}{h_4}) \rightarrow (g^r, (\frac{h_3}{h_4})^r) = (A, \frac{D}{E}) = (b_1, b_2)$

Proof:

- Compute w random in $\mathcal{Z}_q$.
- Compute $(t_1, t_2) = \phi(w) = (g^w, (\frac{h_3}{h_4})^w)$
- Compute $c = \mathcal{G}(A, \frac{D}{E}, t_1, t_2, \mathsf{SimplePlaintextEqualityProof})$
- Compute $s = w + c \cdot r$

Proof is: s, c

Verification:

- Verify that s and c are between 1 and $q - 1$.
- Compute $\phi(s) = (g^s, (\frac{h_3}{h_4})^s) = (z_1, z_2)$
- Compute $(t'_1, t'_2) = (z_1 \cdot b_1^{-c}, z_2 \cdot b_2^{-c})$
- Compute $c' = \mathcal{G}(A, \frac{D}{E}, t'_1, t'_2, \mathsf{SimplePlaintextEqualityProof})$
- Check that $c' = c$

π_3: Transformation $\phi(r, \mathcal{H}(RN)) : (g, h_2, h_5, f) \rightarrow (g^r, h_2^r \cdot g^{\mathcal{H}(RN)}, h_5^r \cdot f^{\mathcal{H}(RN)}) = (A, C, F) = (b_1, b_2, b_3)$

Proof:

- Compute w_1, w_2 random in $\mathcal{Z}_q$.
- Compute $(t_1, t_2, t_3) = \phi(w_1, w_2) = (g^{w_1}, h_2^{w_1} \cdot g^{w_2}, h_5^{w_1} \cdot f^{w_2})$
- Compute $c = \mathcal{G}(A, C, F, t_1, t_2, t_3, \texttt{PlaintextExponentEqualityProof})$
- Compute $s_1 = w_1 + c \cdot r$ and $s_2 = w_2 + c \cdot \mathcal{H}(RN)$

Proof is: s_1, s_2, c

Verification:

- Verify that s_1, s_2 and c are between 1 and $q - 1$.
- Compute $\phi(s_1, s_2) = (g^{s_1}, h_2^{s1} \cdot g^{s_2}, h_5^{s1} \cdot f^{s_2}) = (z_1, z_2, z_3)$
- Compute $(t'_1, t'_2, t'_3) = (z_1 \cdot b_1^{-c}, z_2 \cdot b_2^{-c}, z_3 \cdot b_3^{-c})$
- Compute $c' = \mathcal{G}(A, C, F, t'_1, t'_2, t'_3, \texttt{PlaintextExponentEqualityProof})$
- Check that $c' = c$

π_4: Transformation $\phi(r) : (g, h_2) \rightarrow (g^r, h_2^r) = (A, \frac{C}{g^J}) = (b_1, b_2)$

Proof:

- Compute w random in $\mathcal{Z}_q$.
- Compute $(t_1, t_2) = \phi(w) = (g^w, h_2^w)$
- Compute $c = \mathcal{G}(A, \frac{C}{g^J}, t_1, t_2, \mathsf{PlaintextProof})$
- Compute $s = w + c \cdot r$

Proof is: s, c

Verification:

- Verify that s and c are between 1 and $q - 1$.
- Compute $\phi(s) = (g^s, h_2^s) = (z_1, z_2)$
- Compute $(t_1', t_2') = (z_1 \cdot b_1^{-c}, z_2 \cdot b_2^{-c})$
- Compute $c' = \mathcal{G}(A, \frac{C}{g^J}, t_1', t_2', \mathsf{PlaintextProof})$
- Check that $c' = c$

π_5: Transformation $\phi(x_1, x_2) : (g, A) \rightarrow (h_1, h_1^r, h_2, h_2^r) = (h_1, \frac{B}{\mathcal{K}}, h_2, \frac{C}{g^{\mathcal{H}(RN)}}) = (b_1, b_2, b_3, b_4)$

Proof:

- Compute w_1, w_2 random in $\mathcal{Z}_q$.
- Compute $(t_1, t_2, t_3, t_4) = \phi(w_1, w_2) = (g^{w_1}, A^{w_1}, g^{w_2}, A^{w_2})$
- Compute $c = \mathcal{G}(h_1, \frac{B}{\mathcal{K}}, h_2, \frac{C}{g^{\mathcal{H}(RN)}}, t_1, t_2, t_3, t_4, \texttt{DecryptionProof})$
- Compute $s_1 = w_1 + c \cdot x_1$
- Compute $s_2 = w_2 + c \cdot x_2$

Proof is: s_1, s_2, c

Verification:

- Verify that s_1, s_2 and c are between 1 and $q - 1$.
- Compute $\phi(s_1, s_2) = (g^{s_1}, A^{s_1}, g^{s_2}, A^{s_2}) = (z_1, z_2, z_3, z_4)$
- Compute $(t_1', t_2', t_3', t_4') = (z_1 \cdot b_1^{-c}, z_2 \cdot b_2^{-c}, z_3 \cdot b_3^{-c}, z_4 \cdot b_4^{-c})$
- Compute $c' = \mathcal{G}(h_1, \frac{B}{\mathcal{K}}, h_2, \frac{C}{g^{\mathcal{H}(RN)}}, t_1', t_2', t_3', t_4', \texttt{DecryptionProof})$
- Check that $c' = c$