Towards Trustworthy Internet Elections

Rui Joaquim
Inesc-ID, ISEL
rjoaquim@cc.isel.ip.pt
Carlos Ribeiro
Inesc-ID, IST
carlos.ribeiro@ist.utl.pt
Paulo Ferreira
Inesc-ID, IST
paulo.ferreira@inesc-id.pt

Abstract—We present the EVIV system, an End-to-end Verifiable Internet Voting system. EVIV offers strong integrity guarantees allied with privacy measures that allow the voter to vote privately in public PCs, such as a PC at a cybercafé or at a public library. In EVIV no election system component can change the voter’s vote, not even the PC used to cast the vote. The voter can verify that her vote was cast-as-intended just with the match of two small strings (4 to 5 alphanumeric characters), and anyone (cryptographically capable) can verify that the recorded votes are counted correctly.

Index Terms—e-voting; Internet voting; remote voting; integrity; privacy;

I. INTRODUCTION

In spite of the fact that Internet voting present risks to the voter’s privacy and the election’s integrity, evidence seems to point out that Internet voting has come to stay. Numerous Internet elections (∼139) were already performed worldwide, and many of them (∼40%) were actual real binding elections ([1]). Notable examples are the Switzerland and Estonia cases which are moving to/already have national binding Internet elections.

Usually, Internet voting systems require trust on the vote client platform (the vote casting PC) to give some guarantees of voter’s privacy and election’s integrity [2]–[4]. However, this is not easily achievable on Internet voting given that users’ computers are often vulnerable to a number of attacks (e.g. virus, worms, phishing). A better solution is to provide each user with a tamper resistant hardware to encrypt the votes, and a code voting solution to securely communicate between the tamper resistant hardware and the user, through the vulnerable PC, as it is proposed by Joaquim at al. [5]. Still the user must completely trust the hardware token, given that it may encrypt a different intention than the one specified by the voter.

In EVIV each voter has a unique Voter Security Token (VST). The VST is a tamper resistant security hardware, e.g. smart card or a security USB token. Once the voter gets her VST all the electoral process is online, which gives the voter full mobility. The VST provides strong voter’s authentication using digital signatures. Additionally, the VST is also responsible for vote encryption. At the vote casting process the voter communicates her vote choice to the VST using vote codes [6], [7], which protects the voter privacy from the PC used to cast the vote and allows the voter to cast her vote from any public PC, e.g. a PC at a cybercafé or at a public library.

From the perspective of the users experience, EVIV is very similar to the solution proposed by Joaquim at al. [5]. It also requires that each voter communicates with a private tamper resistant hardware using a code voting protocol. However, in EVIV the tamper resistant hardware does not need to be trusted. In fact, in EVIV, provided that at least one of a chosen set of trustees is trustworthy, each vote reflects the voter intentions even if all other entities collude against the voter.

EVIV exhibits a strong emphasis on the election integrity as it is an end-to-end verifiable voting system (E2E). A system is said to be E2E verifiable, if it allows for both voter cast-as-intended and universal counted-as-recorded verifications:

Voter Cast-as-Intended Verification - the voter can verify that her vote is published on a public bulletin board and accurately represents her choices without relying on any trusted hardware, e.g. the hardware that creates the vote encryption (VST in EVIV).

Universal Counted-as-Recorded Verification - everyone can verify that the final tally is the accurate sum of the valid votes published on the public bulletin board.

EVIV guarantees the election’s integrity without requiring trust on any component of the voting system. For the voter cast-as-intended verification, EVIV uses the MarkPledge technique [8]–[10] to provide cast-as-intended verification with a soundness of \((1 - 2^{-\alpha})^{(k-1)}\), with \(\alpha\) and \(\tau\) being configurable security parameters usually set to values between 20 and 30, and 1 and 5, respectively, and \(k\) the number of candidates in the election. The EVIV universal counted-as-recorded verification is provided by using a homomorphic vote counting process ([11]), which allows any third party to repeat the counting with knowing the actual intention of each individual vote.

From the above description the main characteristics of EVIV can be summarized by the next two properties:

P1 : The EVIV protocol ensures that every vote is counted as intended.

P2 : The EVIV protocol ensures that no one but the voter and her VST knows the voter’s vote.

EVIV is different from all other E2E Internet voting systems that we are aware of, cf. Section II, because it is the first to combine the following characteristics:

1) Full voter’s mobility (complete online voting process).
2) Strong voter’s authentication.
3) No single entity is able to know all votes.
4) Does not require an anonymous vote casting channel to protect the voter’s privacy.
5) Guarantees the privacy and integrity of the voter’s vote even if the voter votes from a public PC, e.g. a PC at a cybercafé or at a public library.
6) Highly sound voter cast-as-intended verification.

The remaining of this article is organized as follows: in the next Section we provide a brief introduction to electronic voting research with a focus on E2E Internet voting systems. Then, in Section III we describe EVIV’s integrity and privacy trust models. Section IV contains the description of the EVIV vote protocol followed by some implementation details in Section V. Section VI evaluates the protocol by proving properties P1 and P2. Section VII shows some key performance results. Finally, we conclude and point out future work in Section VIII.

II. RELATED WORK

Private, correct and verifiable elections were always the goal of the electronic voting research efforts which started about 30 years ago. The resulting voting protocols can be roughly categorized into three main categories (mix-net, blind signatures and homomorphic voting systems) accordingly to the vote anonymization technique used.

The mix-net approach was first proposed by Chaum in 1981 [12]. In a mix-net voting system the encrypted votes are anonymized by the mix-net and then decrypted and counted. The main problem of the mix-net approach is the computational cost of the correct mixing proof, which must preserve the votes anonymity. Over the years we have assisted to significant improvements on mix-nets constructions and respective proof of correct mixing, Confer [13] for a survey.

Blind-signatures voting protocols where introduced by Chaum [14] and Fujioka, Okamoto and Otha [15]. Usually, in a blind-signatures vote protocol the votes are validated by an election authority that issues a blind signature on the vote. The unblinded signed vote is then encrypted and submitted to a tallying authority through an anonymous channel. At the end of the election the tally authority decrypts and publishes the received votes, which allows the voters to verify if their vote has reached the tally authority (and claim if not). Additionally, everyone can attest the correctness of the election tally by verifying that each published vote has a signature of the election authority. The blind signature scheme and the anonymous submission channel prevents the election authority and anyone else from associating a decrypted vote to the voter who casted it. The main disadvantage of the blind signatures voting protocols is the requirement of an anonymous channel to cast the votes.

Homomorphic voting systems were introduced by Cohen (Benaloh) and Fischer [16] and Benaloh and Yung [17]. They rely on homomorphic encryption functions to calculate the final election tally without decrypting individual votes, only the final tally is decrypted, which ensures the privacy of each vote. The main disadvantage of the homomorphic voting systems is the restrictions imposed to the ballot/vote encryption format in order to be handled by homomorphic encryption functions, and the impossibility of dealing with open answer questions.

Over the years many vote protocols based on the described techniques where proposed [4], [11], [18]–[21]. Usually, vote protocols assume either a “cryptographic capable” voter, or a trusted device capable of acting in the name of the voter to perform the voter’s side cryptography. The incapacity of a common voter to perform cryptographic operations means that the voter must trust on the vote client machine that performs the vote encryption. This fact and the vulnerability of todays Internet “architecture/infrastructure”, which expose the voter’s computer to many threats, e.g. a computer virus that can undetectably change the voter’s vote, is one of the main arguments against the adoption of Internet voting cf. [2], [3].

In 2004, with the work of Chaum [22] and Neff [8] a new paradigm in electronic voting research has emerged: End-to-End (E2E) voting systems. The goal of E2E voting systems is to develop voting systems with both voter cast-as-intended verification and universal counted-as-recorded verification. The E2E voting systems were initially proposed to the poll station voting environment ( [8], [22]–[28]). Later some of the ideas were used to develop E2E Internet voting systems.

The Helios voting system was the first E2E Internet voting system [29]. The Helios system uses a mix-net universal verifiable vote count, to ensure the count-as-record property and borrows the cast-as-intended verification mechanism of [27]. In Helios, the voter uses a PC to establish a connection to a Ballot Preparation System (BPS). The vote choice is given in clear text to the PC which then forwards it to the BPS. The BPS prepares the vote encryption and sends it to the PC and the voter can then choose to cast or to verify it. The voter can create and verify as many vote encryptions as she wants. Therefore, assuming that the voter prepares $n$ encrypted votes and verifies $n-1$ of then, the voter cast-as-intended verification as a soundness of $1-1/n$. It is important to note that the vote encryption creation is an anonymous process and therefore anyone can verify the correct behavior of the BPS. Note however that the above mechanism does not ensure privacy, only integrity of the vote. In Helios, both the BPS and the vote preparation/casting PC must be trusted to guarantee the voter’s privacy.

Another E2E voting system for the Internet is the VeryVote system [30]. The VeryVote system has universal mix-net or homomorphic vote count verification and uses the ideas of [8] to implement a cast-as-intended verification. The VeryVote system also uses a code voting interaction [6], [7] which protects the voter’s privacy from the vote casting PC. The code voting interaction is based in code cards printed on code cards that must be previously delivered to the voter, e.g. by postal. The cast-as-intended verification mechanism used in VeryVote offers a soundness of $1-k!/2^\alpha$, where $\alpha$ is a configurable security parameter usually set to a value from 20 to 30 and, $k$ is the number of candidates. However, in VeryVote all vote encryptions are prepared by the election server that, consequently, must be trusted for the voter’s privacy.

The remaining E2E voting system, for the Internet, that we are aware of is the Scratch, Click and Vote (SCV) [31]. SCV provides a mix-net universal vote count and uses the voter cast-as-intended verification ideas of [23], [24] and [28] with
a “blind signature glue”. In SCV there is an Election Authority (EA) which prepares the encrypted votes and a Proxy that the voter uses to interact anonymously with the EA. The privacy of the voter is protected if the EA and the Proxy do not collude. In SCV the voter’s privacy is protected from the vote casting PC through the use of a coding card. The voter obtains her ballot by mail or by visiting certain authorities and gets her coding card from the Proxy, also, by mail or by visiting it. The SCV cast-as-intended mechanism as a soundness of 1 − 1/k, where $k$ is the number of candidates.

As can be seen in Table I, none of the described E2E voting protocols has all the properties of EVIV.

<table>
<thead>
<tr>
<th>III. TRUST MODELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Verifiability/Integrity Trust Model</td>
</tr>
<tr>
<td>EVIV guarantees voter cast-as-intended and counted-as-recorded verifications against a single malicious entity (election server or system player) or a collusion of entities under the following integrity assumptions:</td>
</tr>
<tr>
<td>IA1: The vote encryption and verification technique (MarkPledge) is not flawed.</td>
</tr>
<tr>
<td>IA2: There is a honest trustee among a set of chosen trustees.</td>
</tr>
<tr>
<td>IA3: At least one honest organization or entity with cryptographic abilities will verify that all the published data is correct and that only that data is used in the computation of the election tally.</td>
</tr>
<tr>
<td>IA4: The voter uses the verification services of a honest organization to verify her vote receipt.</td>
</tr>
<tr>
<td>B. Privacy Trust Model</td>
</tr>
<tr>
<td>EVIV guarantees that the privacy of the voter is protected against a single malicious entity or a collusion of entities even if she votes in a public computer. These guarantees are given under the following privacy assumptions:</td>
</tr>
<tr>
<td>PA1: The vote encryption and verification technique (MarkPledge) is not flawed.</td>
</tr>
<tr>
<td>PA2: There is no collusion of more than $t$ out of $n$ trustees, where $t$ and $n$ are configurable security parameters.</td>
</tr>
<tr>
<td>PA3: The VST does not disclose the voters’ vote choices.</td>
</tr>
<tr>
<td>PA4: Only the VST and the voter have knowledge of the vote codes. This assumption implies that the PC used to create the vote codes is trusted not to store or disclose them.</td>
</tr>
</tbody>
</table>

| IV. EVIV SYSTEM DESCRIPTION |
| We start the description of EVIV by presenting the system players and their responsibilities. Then, we describe the system components and their functionality within the system architecture, and finally we outline the vote protocol phases. Throughout this section the reader should confer Fig. 1. |

| A. System Players |
| Electoral Commission (EC) is responsible for the entire electoral process, namely the EC is responsible for the voters enrollment system, the actual voting system and the authentication of all the election public data. |
| Trustees exist in order to share the control over the voter’s privacy and the election integrity among several entities. The trustees should be the political parties and any other authorized entity, e.g. an election observer NGO. |
| Verification Organizations (VO) are responsible for independently validate the correctness of the election public data. The immediate candidates for VO are the entities directly interested in the election outcome, e.g. political parties; however any person/organization can be a VO and verify the validity and correctness of the election, provided that it has the computational means to do it. |
| Voter is any citizen with the right to vote. In EVIV the voter must enroll once with the EC, and for every election perform an online registration to be able to vote online on election day. Besides voting, the voter may also verify if her vote was cast-as-intended by performing a simple string match. |

| B. EVIV architecture |
| The EVIV architecture is comprised by the VST, the vote client platform and five servers. |
| Voter Security Token (VST) is the entity responsible for the vote encryption and the voter’s authentication by means of digital signature (the voter’s private key is inside the VST). Client Platform (PC) is simply the PC(s) or any other kind of interaction machine with a VST reader (e.g. phone, |
pda) together with the corresponding operating system and programs used by the voter during the vote protocol. From now on we refer to the vote client platform just as the PC.

**Bulletin Board (BB)** is the server responsible for the publication of all election public data. The data published cannot be deleted and is always authenticated, i.e. it is signed.

**Voters’ Enrollment Server (VES)** or servers are responsible for the enrollment process of every voter. The enrollment process is the process by which each voter is assigned a security token (VST). After enrollment the voter may participate in several elections until the assigned VST expires.

**Election Registrar (ER)** server provides the election registration service that allows voters to register to vote online on a particular election.

**Ballot Box (BBox)** server provides the vote casting service on election day. The BBox server performs the voter authentication, and a verification of the correctness of the vote encryption (the correctness of the vote will also be verified by VO).

**Verification Server (VeryServ)** provides a verification service to the voters. Each VO should run their instance of a VeryServ.

### C. EVIV Vote Protocol

The EVIV vote protocol is divided in four phases: the voter enrollment phase, the election registration phase, the vote casting phase and the public verification and vote counting phase.

1) **Voter Enrollment Phase:** The voter enrollment phase consists of an off-line enrollment process where the voter identifies herself to the EC and receives her VST with the voter’s private key and a certificate signed by the EC. The voter can then use her VST to vote in all subsequent elections.

2) **Election Registration Phase:** The actual voting process of each election starts with the election registration phase. The election registration phase can be done several months before the election and should be done on a privacy preserving PC, i.e. the PC used in this phase should be trusted not to leak...
private user data. This phase has the following steps:

1) The vote registration phase begins with the generation of a threshold election key pair by the trustees. This key pair is used to generate the specially crafted ballots and to decrypt the final tally. The distributed key generation protocol must guarantee that the decryption algorithm requires at least a subset \( t \) of the trustees. After verifying the correctness of the election key pair generation, the \( EC \) generates and publishes the election certificate (i.e. the election key signed by the \( EC \)) in the \( BB \). The \( EC \) is also responsible for publishing, in the \( BB \), the election’s candidate list, signed with its private key.

2) The publication of the election certificate starts the voters’ registration process. Each voter’s registration process comprises two independent actions, namely: the ballot creation and the generation of a list of codes for the vote casting phase.

**Ballot creation** has four phases:

a) The voter first connects her \( VST \) to a \( PC \) with Internet connection and establish a secure connection (e.g. SSL/TLS) between the \( PC \) and the \( ER \), using the \( VST \) and \( ER \) credentials for authentication (the \( VST \) may require a PIN to release their credentials).

b) Then, the election’s certificate and the signed candidate list are downloaded from the \( BB \), through the \( ER \), to the \( VST \).

c) The \( VST \) uses the election’s certificate and candidate list to produce a specially crafted ballot (cf. section V-B) signed with the \( VST \) private key, which is stored in the \( VST \) persistent memory and sent back to the \( ER \).

d) The \( ER \) verifies the correctness of the ballot construction and, at the end of the registration phase, publishes it in the \( BB \). At the end of the election only the votes created from published ballots should be counted.

**Code list** generation has two phases:

a) After the ballot creation the \( VST \) creates a Code Card (CC) with a random 4 or 5 character code associated with each candidate name and a confirmation code to be used in the cast-as-intended verification process.

b) The CC is then printed through the PC. The voter’s registration process is now complete and the voter can disconnect the \( VST \) from the PC.

The election’s registration phase may continue until the vote casting phase starts.

3) **Vote Casting Phase:** The vote casting phase starts with the generation and publication of the “election initialization data” by the trustees. After this publication the \( BB \) is able to receive votes.

The vote casting process can be performed anywhere including public places such as cybercafés or public libraries without compromising privacy.\(^1\) The vote casting process has the following steps:

1) The voter first connects her \( VST \) to a \( PC \) with Internet connection and establish a secure connection (e.g. SSL/TLS) to the \( BB \), using the \( VST \) credentials.

2) The election initialization data and the signed candidate list are downloaded from the \( BB \), and the latter is displayed to the voter.

3) The voter then communicates her vote choice to the \( VST \) and this creates the vote encryption as follows, cf. Fig. 2:

   a) The voter enters the vote code in her \( CC \) which is associated to her chosen candidate.

   b) The vote code and the election initialization data are sent from the \( PC \) to the \( VST \).

   c) The \( VST \) verifies the signature on the election initialization data and the validity of the vote code, and generates both the vote encryption and the vote receipt using the ballot stored in the registration phase. The vote encryption and vote receipt are then signed with the \( VST \) private key. The vote receipt is then displayed to the voter for confirmation through the \( PC \).

   d) The voter verifies that the confirmation code on her \( CC \) is associated with the selected candidate in the vote receipt and if so confirms the vote. Otherwise, the voter must abort the vote casting process. The voter can then resume to step 3 or begin from scratch on another computer.

4) Finally, the vote and the receipt are sent to the \( BB \).

   As a safeguard the \( BB \) should verify the correctness of the received data before accepting it.

At the end of the vote casting phase the \( EC \) validates and signs all the data received by the \( BB \) (vote encryptions, vote receipts) and publishes it in the \( BB \).

\(^1\)The \( PC \) used to cast a vote will not be able to know the voter’s choice, however the voter must still protect her privacy from other people at the public place. Essentially, the voter must keep her \( CC \) secret.
4) Public Verification and Vote Counting Phase: The public verification and vote counting phase is the last phase of the EVIV protocol. In this phase there is a first verification step which aims to verify the correctness/validity of the published data. Follows the vote counting process and a final verification step to confirm the correctness of the vote counting process.

Public Encrypted Vote Verification is comprised of three tasks for each vote:

1) The first task may be done at the same time as the vote casting phase, given that it only depends on the availability of the ballots. In the task, the VO should verify if the ballot only allows for a vote in one candidate, and that it was signed by the voter’s VST (cf. Section V-D for the details).

2) Then, the VO should verify if the encrypted vote was correctly built from the registered ballot and that the vote receipt matches the encrypted vote and the election initialization data. It should also verify the signature correctness on both the vote and the receipt (cf. Section V-D for the details).

3) Finally, the voter may use a VeryServ instance of one (or more) trusted VO to complete the cast-as-intended verification. The voter just has to verify that the confirmation code on her CC matches the verification code in the verified vote receipt.

If any error is detected in this verification phase the voter should be able to cast another vote. This solution is already used in real world elections, e.g. the Estonian electoral process [32] allows to vote on a polling station on election day and overwrite the electronic vote casted online. Note also that any correction to the encrypted vote occurs before the vote counting process and without revealing the content of the encrypted vote, therefore preserving the voter’s privacy.

Vote Counting Process is comprised of three tasks:

1) First, the BBox performs an homomorphic aggregation of the encrypted votes accepted for the vote count process.

2) Then, at least $t$ trustees collaborate to decrypt the encrypted aggregation. The decryption process besides outputting the final election results also outputs proofs of correct decryption, cf. [11]. Note that the homomorphic aggregation is only possible if the aggregated votes are all valid votes.

3) The homomorphic aggregation, the final election results and proofs of correct decryption are all validated by the $EC$ and published in the $BB$.

Public Verification of the Election Tally is the final step of the election. Anyone with sufficient knowledge, e.g. VO, can verify the aggregation of the encrypted votes by performing themselves the aggregation. Then it is possible to use the published proofs of correct decryption to validate the published final election results.

V. VOTEPROTOCOL IMPLEMENTATION DETAILS

In this section we revisit the EVIV vote protocol and clarify the implementation details that are implicit in the vote protocol description (section IV-C).

A. Voter Enrollment Phase Details

In the voter enrollment phase the voter gets her VST with the voter’s private key which will be used to authenticate the voter and her vote in the vote protocol. The voter’s private key and all other keys used in the vote protocol to authenticate entities or data are of the RSA type. This choice makes the real deployment of EVIV simpler because RSA keys are commonly used worldwide for these purposes; thus real entities may already have their own RSA key pair and VST supporting hardware with native RSA is widely available, e.g. smart cards.

B. Election Registration Phase Details

In the election registration phase the voter registers her ballot. The ballot is a “pre-prepared” encryption of the final vote. EVIV uses a special vote encryption primitive which requires an ElGamal key pair [33]. Therefore the first step of the election registration phase is the creation of an ElGamal election key pair by the trustees.

The ElGamal cryptosystem can be easily transformed in a robust threshold cryptosystem where the private key is shared among $n$ trustees. The decryption of a message can then be computed by a configurable subgroup of at least $t$ trustees without reconstructing/leaking the private key. Details on the distributed key generation and decryption protocols can be found in [34].

The second step of the election registration phase is the creation of the ballot using the MarkPledge technique. When correctly used the MarkPledge technique allows for a simple voter cast-as-intended verification with soundness $1 - 2^{-\alpha}$ ([8]–[10], [13], [25], [30]). The voter only has to match the confirmation code (4 to 5 characters) within the verification codes that appear in the vote receipt, cf. section IV-C3. The variable $\alpha$ is a configurable security parameter that defines the size in bits of the confirmation code.

As far as we know there are three different implementations of the MarkPledge technique and EVIV may use any of them. The complex implementation details of each implementation are outside the scope of this article, however they all may be abstracted by a common API and a set of properties on their data as is defined in [10], cf. Table II. The MarkPledge API can be mapped to any of the MarkPledge implementation proposals.$^2$

The MarkPledge vote encryption is based on the Vote Encryption Primitive $\mathcal{VEP}_{pk}$. The $\mathcal{VEP}_{pk}$ is used to create a special ciphertext for each candidate (vote) under the election public key $(pk)$. The selected candidate gets a special ciphertext representing a $YESvote$ encryption.

$^2$In some MarkPledge implementation proposals the validity of the vote encryption can only be verified after an anonymous vote decryption. Therefore, in those cases the homomorphic vote tally process propose in EVIV must be replaced by a mix-net vote tally process.
\begin{align*}
\forall \mathcal{E}_\mathcal{P}_\mathcal{P}(b, \text{ccode}, r) &= \langle \text{cvote}, \text{voteValidity} \rangle \\
\text{cvote} &= \begin{cases} 
\text{cv} \in \text{NOvote} & \text{if } b = 0 \\
\text{cv} \in \text{YESvote} & \text{if } b = 1
\end{cases}
\end{align*}

\begin{align*}
\mathcal{R}_\mathcal{E}_\mathcal{P}_\mathcal{P}(\text{cvote}, r, \text{chal}) &= \langle \text{ccode}, \text{receiptValidity} \rangle \\
\text{ccode} &= \begin{cases} 
\text{cv} & \text{if } \text{cvote} = \mathcal{E}_\mathcal{P}_\mathcal{P}(1, \text{ccode}, r) \\
\psi(\text{ccode}, \text{chal}) & \text{if } \text{cvote} = \mathcal{E}_\mathcal{P}_\mathcal{P}(0, \text{ccode}, r)
\end{cases}
\end{align*}

\begin{align*}
\forall \mathcal{V}_\mathcal{P}_\mathcal{P}(\text{cvote}, \text{voteValidity}) &= \begin{cases} 
\text{True} & \text{if } \text{cvote} \in \{\text{NOvote} \cup \text{YESvote}\} \\
\text{False} & \text{otherwise}
\end{cases}
\end{align*}

\begin{align*}
\mathcal{R}_\mathcal{V}_\mathcal{P}_\mathcal{P}(\text{cvote}, \text{chal}, \text{ccode}, \text{receiptValidity}) &= \text{validity} \\
\text{validity} &= \begin{cases} 
\text{True} & \text{if } 3 \text{b, ccode, r : cvote} = \mathcal{E}_\mathcal{P}_\mathcal{P}(b, \text{ccode}, r) \land \\
(\text{ccode}, \text{receiptValidity}) = \mathcal{R}_\mathcal{E}_\mathcal{P}_\mathcal{P}(\text{cvote}, r, \text{chal}) \\
\text{False} & \text{otherwise}
\end{cases}
\end{align*}

\begin{align*}
\mathcal{C}_\mathcal{P}_\mathcal{P}(\text{cvote}, \text{chal}, \text{ccode}) &= \text{canonicalVote} \\
\text{canonicalVote} &= \begin{cases} 
\varepsilon_\mathcal{P}_\mathcal{P}(\text{yes}) & \text{if } \text{cvote} \in \text{YESvote} \\
\varepsilon_\mathcal{P}_\mathcal{P}(\text{no}) & \text{if } \text{cvote} \in \text{NOvote}
\end{cases}
\end{align*}

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
</table>
| **MP**<sub>1</sub> | \forall x, \text{chal} : Pr[\psi(\text{O}, \text{chal}) = x] = \frac{1}{|\text{ccode}|} \\

Where \text{O} is a uniform random variable on the domain of \text{ccode} and |\text{ccode}| is the size of the domain of \text{ccode}.

| **MP**<sub>2</sub> | \forall x, \text{ccode} : Pr[\psi(\text{ccode}, \text{C}) = x] = \frac{1}{|\text{ccode}|} \\

Where \text{C} is a uniform random variable on the domain of \text{chal} and |\text{ccode}| is the size of the domain of \text{ccode}.

| **MP**<sub>3</sub> | \forall \text{cvote, voteValidity, ccode, receiptValidity, chal, canonicalVote} : \\

\forall \mathcal{E}_\mathcal{P}_\mathcal{P}(\text{B}, \text{O}, \text{R}) = \langle \text{cvote}, \text{voteValidity} \rangle \land \\
\mathcal{R}_\mathcal{E}_\mathcal{P}_\mathcal{P}(\text{cvote}, \text{O}, \text{R}, \text{chal}) = \langle \text{ccode, receiptValidity} \rangle \land \\
\mathcal{C}_\mathcal{P}_\mathcal{P}(\text{cvote, chal, ccod}) = \text{canonicalVote} \land \\
Pr[\text{cvote} \in \text{YESvote}] = Pr[\text{cvote} \in \text{NOvote}] \\

Where \text{O} and \text{R} are uniform random variables on the domain of \text{ccode} and \text{r}, respectively, and \text{B} is a binary uniform random variable.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Property & Definition \\
\hline
**MP**<sub>1</sub> & \forall x, \text{chal} : Pr[\psi(\text{O}, \text{chal}) = x] = \frac{1}{|\text{ccode}|} \\

Where \text{O} is a uniform random variable on the domain of \text{ccode} and |\text{ccode}| is the size of the domain of \text{ccode}.

**MP**<sub>2</sub> & \forall x, \text{ccode} : Pr[\psi(\text{ccode}, \text{C}) = x] = \frac{1}{|\text{ccode}|} \\

Where \text{C} is a uniform random variable on the domain of \text{chal} and |\text{ccode}| is the size of the domain of \text{ccode}.

**MP**<sub>3</sub> & \forall \text{cvote, voteValidity, ccode, receiptValidity, chal, canonicalVote} : \\

\forall \mathcal{E}_\mathcal{P}_\mathcal{P}(\text{B}, \text{O}, \text{R}) = \langle \text{cvote, voteValidity} \rangle \land \\
\mathcal{R}_\mathcal{E}_\mathcal{P}_\mathcal{P}(\text{cvote, O, R, chal}) = \langle \text{ccode, receiptValidity} \rangle \land \\
\mathcal{C}_\mathcal{P}_\mathcal{P}(\text{cvote, chal, ccod}) = \text{canonicalVote} \land \\
Pr[\text{cvote} \in \text{YESvote}] = Pr[\text{cvote} \in \text{NOvote}] \\

Where \text{O} and \text{R} are uniform random variables on the domain of \text{ccode} and \text{r}, respectively, and \text{B} is a binary uniform random variable. \\
\hline
\end{tabular}
\caption{The MarkPledge API properties.}
\end{table}

(\mathcal{E}_\mathcal{P}_\mathcal{P}(1, \text{ccode} \text{r}_i)) and each of the other candidates gets an independent special cipher text of a \text{NOvote} encryption (\mathcal{E}_\mathcal{P}_\mathcal{P}(0, \text{ccode} \text{r}_j \text{r}_k)). The \text{ccode} is an embedded commit code for the \text{cvote} ciphertext. The parameter \text{r} is the randomness used to create the ciphertext.

Since the voter’s choice in not yet known at the ballot preparation time, EVIV uses a “pre-prepared” vote encryption that we call ballot. The ballot is therefore a set of one \text{YESvote} and \(k - 1\) \text{NOvote}s in random order, where \(k\) is the number of running candidates in the election.

The ballot construction ends with the creation of a zero knowledge public verifiable vote validity data (VoteValidity = voteValidity \_1 \parallel \ldots \parallel voteValidity \_k \parallel globalVoteValidity). The voteValidity \_i is an output of the \mathcal{E}_\mathcal{P}_\mathcal{P} primitive that may be used by the \forall \mathcal{V}_\mathcal{P}_\mathcal{P} \_i primitive to prove, in zero knowledge, that \text{cvote} \_i is either a \text{YESvote} or a \text{NOvote}. The globalVoteValidity is an extra proof data, extracted from \text{cvotes} using the randomness used to create them, which proves in zero knowledge that only one \text{cvote} is a \text{YESvote} and all others are \text{NOvote}s. For more details on the construction of these proofs data consult [10], [13].

The voter cast-as-intended verification is based on the embedded commit code \text{ccode} in the \text{YESvote} entry of the ballot. The \text{VST} pledges this \text{ccode} value by printing it in the \text{CC} as the confirmation code.

Finally, the election registration phase ends with the publication of the ballots and corresponding validity proof data.

### C. Vote Casting Phase Details

After the publication of all accepted ballots at the end of the election registration phase, the vote casting phase starts with the creation and publication of a challenge by the trustees (referred as election initialization data in Section IV-C3). This is created with the distributed random number generation algorithm described in [30], which assures the generation of a random value if at least one trustee is honest. The algorithm creates the random challenge basically by joining random values created by the trustees using the bitwise XOR function.

As described in Section VI-A, it is essential to ensure that the challenge (chal in the MarkPledge implementation) is not known to the system before the ballot creation, to guarantee
the vote soundness. For this reason in EVIV the trustees only create the challenge after the publication of all the ballots that will be accepted in the election.

The $VST$ creates the final vote and receipt from the ballot committed in the election registration phase and the challenge created by the trustees. The final vote is simply the rotation necessary to perform on the ballot entries, to align the $YESvote$ to the selected candidate, cf. Fig. 3. The vote receipt ($voteReceipt$) is created using the Receipt Extraction Primitive $REP_{pk}$. The $voteReceipt$ is just a set of verification codes ($vcodes_i$) extracted from each vote entry by the $REP_{pk}(vote, r, chal)$. Following the definition of the $REP_{pk}$ (cf. Table II) and properties $MP_{p1}$ and $MP_{p2}$ (cf. Table III), the $vcodes$ extracted from a $YESvote$ is not affected by the challenge value, i.e., each $YESvote$ has a constant $vcodes = code$ embedded at the time of the $YESvote$ creation. On the other hand, the $vcodes$ computation of a $NOvote$ requires both the challenge and the $code$ embedded at the time of the $NOvote$ creation. Therefore, at the election registration phase the $VST$ can only commit to the $vcodes$ of the $YESvote$, any other $vcodes$ is unpredictable at that time because it depends on the challenge.

The $VST$ also provides a vote receipt correctness proof data $ReceiptValidity = receiptValidity_1 \parallel \ldots \parallel receiptValidity_k$. The $ReceiptValidity$ data allows a zero knowledge verification that the verification codes in the vote receipt where correctly extracted from each vote entry by the $REP_{pk}(vote, r, chal)$.

The $ReceiptValidity$ data is authenticated and sent for publication along with the vote and the vote receipt.

Given that the extracted $vcodes$ of $NOvotes$ cannot be predicted at the time of the vote creation, it is possible that the receipt created with the challenge generated by the trustees contain two or more equal $vcodes$. If one of the $vcodes$ of $NOvotes$ is equal to the $code$ of the $YESvote$ (i.e., the confirmation code printed on the $CC$), the voter is not able to verify the intention of her vote. A receipt with two or more equal $vcodes$ is said to be invalid. In order to reduce the probability that a receipt is invalid, the $VST$ tests its validity before discloing it to the voter, and if the receipt is invalid the $VST$ generates another receipt using the a hash of the challenge. If again, the receipt is invalid the $VST$ keeps trying for a specified number of times.

D. Public Verification and Vote Counting Phase Details

The verification and counting phase may be performed by any $VO$ running a $VeryServ$. The $VO$ verifies both the $VoteValidity$ data and the $ReceiptValidity$ data. The $ReceiptValidity$ is verified using the Receipt Verification Primitive $RVP_{pk}$ to verify individually each of the $vcodes$ in the vote receipt. The $VoteValidity$ verification is performed in two steps: first, it is verified that each $vcodes$ is either a $YESvote$ or a $NOvote$ with the Vote Verification Primitive $VVP_{pk}$; the second step is the verification of the $globalVoteValidity$ data. The verification steps of the $globalVoteValidity$ can be found in [10].

If both validations are successful there is a cryptographic link between the ballot, the vote, and the vote receipt. Therefore, the voter may verify the correctness of her vote by checking that the confirmation code is in the correct position on the receipt, which may be done using a $VO$ of her choice.

All votes validated by the $VO$ that are not protested by the voters enter the vote counting process. In order to achieve efficient universal counted-as-recorded verification the system uses a homomorphic vote count process ([11], [13]). In a homomorphic vote count, all vote encryptions are aggregated so that at the end we get the encryption of the sum of all the aggregated votes. Then, the trustees decrypt only the encryption resulting from the aggregation. Since the aggregation is a public operation it can be easily verified by anyone. The final decryption of the aggregated results is performed by the trustees using a public verifiable threshold decryption algorithm [34]. The homomorphic vote count preserves the voters’ privacy because no individual vote is ever decrypted.

However, before the homomorphic aggregation takes place it is necessary to perform a vote canonicalization process [9], [10] to normalize the $vcodes$ and eliminate the embedded $vcodes$, which is performed using the Canonicalization Primitive $CP_{pk}$ on each $vcodes$. The aggregation of canonical $YESvotes$ with canonical $NOvotes$ yields the quantity of $YESvotes$ in the aggregation. Thus, to reach the final election results it suffices to aggregate separately the ciphertexts of each candidate. Then, the trustees decrypt the individual sums for each candidate and provide the final election results. Everyone can then verify the correctness of the aggregations and corresponding decryptions.

VI. EVALUATION

This section provides proof sketches for properties P1 and P2. Within the proofs we use the MarkPledge primitives and properties presented respectively in Tables II and III. Before proving the properties let us define vote and receipt validity.

**Definition 1.** A vote is valid if it is the concatenation of $k$ $vcodes$, one of type $YESvote$ and $k−1$ of type $NOvote$. 

![Fig. 3](image-url)
Definition 2. A receipt is correct, with respect to a specific vote $v = \text{cvote}_1 || \ldots || \text{cvote}_k$, if it is the concatenation of $k$\text{cvote}_i = \mathcal{R}\mathcal{E}\mathcal{P}_{pk}(\text{cvote}_i, \text{chal})$

Definition 3. A receipt is valid if it is correct and every $\text{cvote}_i$ in it, is unique.

A. Property P1 (Election Integrity)

Theorem 1. Every publicly recorded vote is valid.

Proof Sketch. Under integrity assumption IA1, the $\mathcal{Y}\mathcal{V}\mathcal{P}_{pk}$ attests that a $\text{cvote}$ is either a $\text{YESvote}$ or a $\text{NOvote}$, to anyone knowing the correspondent $\text{voteValidity}$ data.

Given that, after the election registration phase, every $\text{cvote}_i$, and $\text{voteValidity}_i$, comprising a ballot are public, anyone may use the $\mathcal{Y}\mathcal{V}\mathcal{P}_{pk}$ to verify that every $\text{cvote}_i$ in the ballot is either a $\text{YESvote}$ or a $\text{NOvote}$. Given that the $\text{globalVoteValidity}$ data also becomes public after the election registration phase, every one may attest that only one of the $\text{cvotes}$, in the ballot is a $\text{YESvote}$ (cf. section V-D).

In EVIV the final vote is a simple rotation of the ballot entries, therefore if a ballot is valid, the vote is also valid, i.e. one of its entries is a $\text{YESvote}$ and all other entries are $\text{NOvotes}$.

Lemma 1. Every publicly recorded receipt is correct with respect to a valid vote.

Proof Sketch. Under the integrity assumption IA1, the $\mathcal{R}\mathcal{E}\mathcal{P}_{pk}$ attests the correct extraction of a $\text{vcode}$ from a pair $\langle \text{cvote}, \text{chal} \rangle$

Given that, after the vote casting phase, every $\text{vcode}_i$, $\text{cvote}_i$, $\text{chal}$ and $\text{receiptValidity}_i$ are public data, anyone can verify that each $\text{vcode}_i$ in the receipt was correctly extracted from the reordered ballot entries that compose the vote.

Lemma 2. The challenge used by $\mathcal{R}\mathcal{E}\mathcal{P}_{pk}$ is uniformly distributed and could not be predicted before being generated by the trustees.

Proof Sketch. Given that there is at least one honest trustee (IA2), the protocol (cf. section V-C) used to generate the challenge ensures that the challenge is fresh and can not be predicted.

Lemma 3. Every $\text{vcode}$ extracted with $\mathcal{R}\mathcal{E}\mathcal{P}_{pk}$ in EVIV is a randomly uniform distributed variable.

Proof Sketch. There are two types of $\text{vcodes}$, the ones extracted from $\text{YESvotes}$ and the ones extracted from $\text{NOvotes}$. The $\text{vcodes}$ extracted from $\text{YESvotes}$ are equal to the embedded codes that, by definition, are uniformly distributed random numbers. The $\text{vcodes}$ extracted from $\text{NOvotes}$ are uniform distributed random numbers according with the MarkPledge property $MP_{P2}$ provided that the $\text{chal}$ is a uniform random number (Lemma 2).

Theorem 2. Every vote and challenge combination has a public verifiable, probabilistic valid receipt.

Proof Sketch. A receipt is valid if it is correct and every $\text{vcode}$ in it, is unique. Under Definition 2 a receipt extracted with the $\mathcal{R}\mathcal{E}\mathcal{P}_{pk}$ primitive is correct and can be publicly proven under Lemma 1.

Given that every $\text{vcode}$ is a uniform random variable (Lemma 3), the probability that a correct receipt is invalid, i.e. two or more $\text{vcode}$ are equal is given by the birthday paradox probability $p_1 = 1 - \frac{1}{2^{k-1}}$, where $k$ is the number of candidates. Given that a $\mathcal{VST}$ extracts the $\text{vcodes}$ several times, using a hash chain on the $\text{chal}$, until it finds a valid receipt, the probability that it may find a valid receipt, after a maximum of $\tau$ attempts, is $p_\tau = 1 - p_1^{\tau}$. From $p_\tau$ it is clear that the probability of success can be made as high as required by increasing the number of maximum attempts $\tau$.

Given that the receipt is public any one may verify that every $\text{vcode}_i$ in it is unique.

Lemma 4. The challenge is generated after the commitments to the confirmation code and vote encryption entries.

Proof Sketch. In EVIV both the $\text{YESvote}$ verification code and vote encryption entries are committed in the election registration phase. The $\text{YESvote}$ verification code is printed as the confirmation code on the voter’s $CC$ and the vote encryption entries (cvote) are the ballot entries which are published at the end of election registration phase (cf. Section V-B). Only then, when the election registration phase ends, at the vote casting phase initialization, the challenge is created by the trustees.

Theorem 3. For a valid $\langle \text{vote}, \text{receipt} \rangle$ pair the EVIV verification process, of the voter intention, is sound with probability $p = (1 - 2^{-\alpha})^\tau \cdot (k-1)$, where $k$ is the number of candidates, $\alpha$ the security parameter of MarkPledge and, $\tau$ the maximum number of attempts to generate a valid receipt.

Proof Sketch. Just prove the opposite. The verification process is not sound, with probability $q = 1 - p$, if the voter may be fooled into believe that she voted in one candidate and the $\text{YESvote}$ entry is in another candidate. Given that the receipt is valid, i.e. every $\text{vcode}$ is different and was extracted from the corresponding $\text{cvote}$, the only way that that can happen is if the $\mathcal{VST}$ guesses the $\text{vcode}$ of one of the $\text{NOvotes}$ in the vote. However, by MarkPledge property $MP_{P2}$, before knowing the value of $\text{chal}$ (Lemma 4), the $\text{vcodes}$ of $\text{NOvotes}$ are uniformly distributed over the MarkPledge code space $([0, 2^\alpha])$, therefore the probability that a dishonest $\mathcal{VST}$ is able to guess at least one of the $k-1$ $\text{vcodes}$ of $\text{NOvotes}$ is given by $q = 1 - (1 - 2^{-\alpha})^{k-1}$, and the probability that it is able to guess at least one of the $k-1$ $\text{vcodes}$ of $\text{NOvotes}$ in one of the $\tau$ attempts of receipt generation is $q = 1 - (1 - 2^{-\alpha})^\tau \cdot (k-1)$.

Lemma 5. Once the lists of ballots and votes/vote-receipts are published no entries can be deleted, changed or added.

Proof Sketch. The lists of ballots and votes/vote receipts are signed by the $EC$ before the publication in the $BB$. Therefore, the deletion, change or addition of any entry in these lists is easily detectable.
Theorem 4. Every voter can verify that her vote is cast-as-intended.

Proof Sketch. Given that votes cannot be removed after publication (Lemma 5) and there is at least one honest VO (IA4), which is able to perform the public validation of votes and receipts, then every voter has access to her vote and vote receipt and is assured of their validity (Theorems 1 and 2, respectively), and under Theorem 3 the voter is able to identify the YESvote entry in the vote receipt with a soundness \((1 - 2^{-\alpha})^\tau (k-1)^{-1}\).

Lemma 6. Every published vote is valid and suitable for homomorphic aggregation.

Proof Sketch. Under IA1, the YESvote and NOvote encryptions are suitable for homomorphic aggregation after being processed by CP pk. Given that, each vote has only one YESvote and all other entries are NOvotes (Theorem 1), the entries of every vote for each candidate, after being processed by CP pk, can be homomorphic aggregated.

Theorem 5. Every vote is counted-as-recorded.

Proof Sketch. EVIV uses a homomorphic vote count process. Since the homomorphic aggregation of the encrypted votes is a public operation everyone can perform/verify it. The decryption process of the homomorphic aggregation result is also public verifiable, because it produces public proofs of correct decryption (cf. Section V-D). Given that no vote can be deleted after publication (Lemma 5) and there is at least one honest VO (IA3), every vote is counted-as-recorded.

Theorem 6. The EVIV protocol ensures that every vote is counted as intended.

Proof Sketch. The proof results directly from Theorems 4 and 5, given that the first one ensures that the vote is record as intend and the second ensures that is counted as recorded.

B. Property P2 (Voter’s Privacy)

Lemma 7. The ballot creation process preserves the voter’s privacy

Proof Sketch. Under assumptions PA1 and PA3, neither the MarkPledge technique nor the VST create implicit channels, therefore the only output of the ballot creation process are the ballot and the ballot validity data. Under MarkPledge property MP p3, neither the ballot nor the ballot validity data reveals the position of the YESvote before being open. Under assumption PA4 only the voter and the VST knows the association between each candidate and the vote code on the voter CC, therefore revealing the vote code of the selected candidate does not reveal the identity of the selected candidate.

Lemma 9. The vote and receipt validity verifications preserve the voter’s privacy.

Proof Sketch. The only output of the vote and receipt validity verifications is a true or false value indicating if the vote and receipt are valid.

Lemma 10. The voter cast-as-intended verification process preserves the voter’s privacy.

Proof Sketch. The voter verifies her vote by visually checking that the confirmation code is the verification code associated to the chosen candidate in the vote receipt. Since no data is generated by the cast-as-intended verification process and the confirmation code is not given to anyone (assumption PA4) the voter cast-as-intended verification process preserves the voter’s privacy.

Theorem 7. The EVIV protocol ensures that no one but the voter and her VST knows the voter’s chosen candidate.

Proof Sketch. Lemmas 7 to 11 prove that with the exception of the vote code used to communicate the voter’s choice to the VST all data generated by the voting process is already public and preserves the voter’s privacy. Lemma 8 attest that even the vote code, used to chose the candidate, can be made public without compromising the voter’s privacy. Additionally, Lemmas 7 to 11 under our privacy trust model prove that there is no privacy risk in any election phase:

- By privacy assumption PA4 and Lemma 7 there is no privacy risk in the election registration phase.
- By Lemma 8 there is no risk in the vote casting phase.
- By Lemmas 9, 10 and 11 there is no privacy risk in the public verification and vote counting phase.

Therefore we conclude that under our privacy trust model the EVIV protocol guarantees that no one but the voter and her VST (which creates the vote encryption) knows the voter’s vote choice.

VII. IMPLEMENTATION RESULTS

The EVIV protocol is highly scalable given that most of the required services may be replicated (cf. Fig. 1). The Election Register service (ER) and the Ballot Box service (BBox) are...
both stateless services, therefore they can be replicated for both scalability and fault tolerance. The Bulletin Board (BB) and the Verification Service (VeryServ) may be partitioned in several instances of BB and VeryServ services by voter ID, i.e. each instance of the BB and of VeryServ handle the ballots of only a specific set of voters, thus dividing the workload. Therefore, the only critical element, in terms of performance of the solution, is the VST, which we have tested.

Besides the VST, we have also tested the VeryServ performance in order to evaluate the amount of computer power required from a small Verification Organization (VO) to test and count all the votes.

In our prototype the VST is a JCOP 31v2.2 smart card which has a JavaCard 2.2.1 programming interface [35]. Due to the performance restrictions intrinsic to a smart card application we have implemented the MarkPledge technique as specified in [10], as it presents significant improvements over the previous MarkPledge specifications.

Although more efficient than previous implementations this MarkPledge implementation also requires the calculation of modular exponentiations and multiplications of large integers. Unfortunately, the JavaCard API does not provide support for these operations and implementing them as pure Java code would yield an unacceptable penalty time [36]. Thus, we use the cryptographic JavaCard API to access the RSA algorithm implementation, which is performed by the smart card cryptographic processor and has an acceptable performance.

The exponentiation function is performed using the RSA algorithm without message padding, with a prime module of 1024 bits. For the modular multiplication we use the formula \((a+b)^2-(a-b)^2=4.a.b\), as in [36]. The addition, subtraction and division by 4 were implemented in pure Java code and the squaring function is done by the internal JavaCard RSA implementation. The direct consequence of this implementation is that the calculation of modular multiplications takes slightly more time than the calculation of modular exponentiations, although less than it would cost if it were implemented with pure Java code.

Another consequence of this implementation is the module of the ElGamal messages and random factors used. Given that the generation of the ballot validation proof requires the multiplication of ElGamal random factors, and multiplications requires modular exponentiations, the module of ElGamal random factors must be equal to the module of the exponentiations used in those multiplications. On the other hand the cryptographic processor of JCOP 31v2.2 cannot do exponentiations for modules with less than 512 bits, which has an impact on the performance of the overall system, given that ElGamal messages with modules between 160 and 224 bits are already safe (cf. the finite field cryptography parameters size defined in [7]).

The results, in Table IV, show that the vote casting phase is very fast. For an election with 10 candidates the VST takes around 5 seconds to generate the receipt. On the other hand the registration is a bit more slow. Again, for an election with 10 candidates, the VST takes around 1.5 minutes \((2.5 + 10(2.3 + 6.5)\) seconds) to produce the ballot and the respective correction proof. Although we consider the performance acceptable for an election with a small number of candidates, our current prototype implementation is very penalized with the lack of a hardware assisted modular multiplication function in the JavaCard used.

For the VeryServ performance evaluation we have used a dual Nehalem, chipset 5500 LE computer, with two Quad Core Xeon at 2.0 Ghz 4MB Cache processors and 16GB of RAM, running Linux 32 bit Ubuntu 2.6.32. The test program was coded in Java running 8 threads. We run our tests for an election setup with 10 candidates and 10000 voters. We have not considered the communication costs given that it is dependent on the communication link between the VeryServ and the BB services, however, for reference, a VeryServ with a 100 Mbps link would take \(\sim 22\) sec to transfer a 1 million votes with 10 candidates \((\sim 14\) GBytes).

The verification and counting phase has three stages: the ballot verification, the receipt verification, and the vote tally (Table IV). The last two can only be executed after the end of the election, only then the electoral commission signs the final vote list and closes it. However, the first one may start after the end of the registration phase, given that the final ballot list is defined at that stage. Assuming that the vote casting phase takes around 24 hours then a single VeryServ service will be able to verify around 11 million ballots of 10 candidates each\(^3\). It also would take the same VeryServ service around 11 hours to verify the overall election tally for the same 11 million votes, after the end of the election.

### VIII. Conclusions and Future Work

The EVIV system gives the voter full mobility and offers strong integrity guarantees allied with privacy measures that allow the voter to vote privately in public PCs, such as a PC at a cybercafé or at a public library.

The EVIV protocol does not require auditing computer systems, only the data produce by them, i.e. the correction of the code executed by each service can be verified by checking the output of the election, given that every result has a correspondent proof of correctness associated.

\(^3\)A VO with 2 machines similar to our own would be able to verify all the ballots of any US state in 24 hours.

### TABLE IV

<table>
<thead>
<tr>
<th>Phase</th>
<th>VST - Parameter initialization</th>
<th>VST - Ballot generation</th>
<th>VST - Ballot proof data generation</th>
<th>Vote casting phase</th>
<th>VST - Vote receipt creation</th>
<th>Public verification and counting phase</th>
<th>VeryServ - Ballot validity verification</th>
<th>VeryServ - Vote receipt validity verification</th>
<th>VeryServ - Vote tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration phase</td>
<td>2.5 sec</td>
<td>2.3k sec</td>
<td>6.5k sec</td>
<td></td>
<td>0.5k sec</td>
<td></td>
<td>782k µsec</td>
<td>335k µsec</td>
<td>1k µsec</td>
</tr>
<tr>
<td>Public verification and counting phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[5 + 10(2^3 + 6.5)\]
EVIV has a distributed voter’s privacy model. In EVIV no entity is able to break the voters privacy since each encrypted vote is created by each voter’s VST. Therefore, if an attacker wants to know who voted for who, the attacker must perform a large scale attack to the PCs used to create the vote codes. This attack can be made virtually impossible by allowing the voter to create her vote codes from an off-line PC.

EVIV offers the highest soundness for voter cast-as-intended verification off all known E2E Internet voting systems, requiring the voter to perform only the match of two small 4 to 5 alphanumeric strings. The soundness of the voter cast-as-intended verification is $(1 - 2^{-\alpha}) \cdot \tau \cdot (k - 1)$, with $\alpha$ and $\tau$ being configurable security parameters usually set to values between 20 and 30, and 1 and 5, respectively.

As future work it is important to test EVIV in a production environment. It is also important to extend the application range of EVIV by enhancing the voter cast-as-intended verification mechanism in order to support multiple candidate selection and candidate ranking with the same high soundness.

REFERENCES