Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

EVIV: An end-to-end verifiable Internet voting system

Rui Joaquim [a,b,*], Paulo Ferreira [b,c], Carlos Ribeiro [b,c]

[a] Instituto Politécnico de Lisboa, Instituto Superior de Engenharia de Lisboa - ISEL, ADEETC, R. Conselheiro Emídio Navarro, 1, 1959-007 Lisboa, Portugal

[b] INESC-ID, GSD, R. Alves Redol, 9, 1000-029 Lisboa, Portugal

[c] Universidade Técnica de Lisboa, Instituto Superior Técnico, DEI, Av. Rovisco Pais, 1, 1049-001 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 2 June 2012
Received in revised form
20 September 2012
Accepted 19 October 2012

Keywords:
E-voting
Internet voting
Remote voting
Integrity
Privacy

ABSTRACT

Traditionally, a country's electoral system requires the voter to vote at a specific day and place, which conflicts with the mobility usually seen in modern live styles. Thus, the widespread of Internet (mobile) broadband access can be seen as an opportunity to deal with this mobility problem, i.e. the adoption of an Internet voting system can make the live of voter's much more convenient; however, a widespread Internet voting systems adoption relies on the ability to develop trustworthy systems, i.e. systems that are verifiable and preserve the voter's privacy. Building such a system is still an open research problem.

Our contribution is a new Internet voting system: EVIV, a highly sound End-to-end Verifiable Internet Voting system, which offers full voter's mobility and preserves the voter's privacy from the vote casting *PC* even if the voter votes from a public *PC*, such as a *PC* at a cybercafé or at a public library. Additionally, EVIV has private vote verification mechanisms, in which the voter just has to perform a simple match of two small strings (4–5 alphanumeric characters), that detect and protect against vote manipulations both at the insecure vote client platform and at the election server side.

1. Introduction

In spite of the fact that Internet voting presents risks to the voter's privacy and the election's integrity, evidence seems to point out that Internet voting has come to stay. According to the Krimmer et al. (2007) study, numerous Internet elections (~140) had already occurred worldwide, and many of them (~40%) were actual real binding elections. These numbers have been increasing as more countries perform trials or adopt the Internet voting channel. Notable examples are the Switzerland and Estonia cases which are moving to/already have national binding Internet elections. A more recent example is Norway which had a trial on an Internet voting system in the 2011 local government elections (Ministry of Local Government and Regional Development, 2012).

The biggest challenges of Internet voting are the voter's privacy and coercion issues at the uncontrolled voting environment and the (in)secure platform problem, i.e. the (in) security of the vote casting *PC* that can be the home or office computer of even a computer at a cybercafé or at a public library (Jefferson et al., 2004; Kiayias et al., 2006; Dagstuhl Accord, 2007). 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; however, this is not easily achievable given that vote casting *PC*s are in uncontrolled environments and often vulnerable to a number of attacks (e.g. virus, worms, phishing).

EVIV addresses the voter's privacy at the vote casting *PC* in addition to the insecure platform problem. EVIV allows for re-voting (cast multiple votes) which may address weak forms of

* *Corresponding author*. Instituto Politécnico de Lisboa, Instituto Superior de Engenharia de Lisboa, ADEETC, R. Conselheiro Emídio Navarro, 1, 1959-007 Lisboa, Portugal. Tel.: +351 939457621.

E-mail addresses: rjoaquim@deetc.isel.ipl.pt (R. Joaquim), paulo.ferreira@inesc-id.pt (P. Ferreira), carlos.ribeiro@ist.utl.pt (C. Ribeiro).

0167-4048/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2012.10.001

coercion and also prevent, to some extent, vote buying attacks.

The design of EVIV is driven by the following goal: create a fully mobile End-to-End (E2E) verifiable Internet voting system that protects the voter's privacy from the vote casting *PC*. From this goal we have derived the following requisites:

1. EVIV must enable the voter to vote privately even from public *PC*s on election day, e.g. a *PC* at a cybercafé or at a public library.
2. EVIV must have privacy preserving voter recorded-as-intended verification, i.e. it must allow the voter to verify that her recorded vote accurately represents her choices without revealing her vote intention nor relying on any trusted hardware/software, e.g. the hardware/software that creates the electronic vote.
3. EVIV must provide privacy preserving universal counted-as-recorded verification[1] i.e. – allows anyone to verify that the final tally is the accurate sum of all the valid recorded votes while preserving the voters' privacy.
4. EVIV must not impose mobility restrictions to the voters.

These requisites define the high level characteristics that differentiate EVIV from other E2E verifiable Internet voting systems: EVIV, to our knowledge, is the first E2E verifiable Internet voting system that offers full mobility to the voter and preserves the voter's privacy from the vote casting *PC* (cf. Section 7).

To achieve the above described goal/requisites, the EVIV vote protocol combines a code voting protocol (Chaum, 2001; Oppliger, 2002) with the MarkPledge cryptographic voter's verifiable vote encryption technique (Neff, 2004; Adida and Neff, 2009; Joaquim and Ribeiro, 2012). The code voting protocol preserves the vote's privacy from the vote casting *PC* because the voter uses secret vote codes to select the candidate; on the other hand, the MarkPledge vote encryption technique allows the voter to verify, with a very high soundness, that her vote is cast and recorded-as-intended, performing just a simple match of two small strings (4–5 alphanumeric characters). Additionally, the vote encryption used in EVIV also supports well known universal counted-as-recorded verification techniques, e.g. verifiable homomorphic vote tally, cf. Section 3.1.3.

One important, and also differentiating, aspect of EVIV is that its code voting protocol does not rely on a centralized vote codes distribution. In EVIV the vote codes are not used to communicate the voter's choice to a remote election server; instead, in EVIV every voter has a voter security token (*VST*), which is responsible for the vote encryption, and to which the voter communicates her candidate selection. With the help of the *VST*, each voter generates the vote codes at home, which facilitates the logistics of the election and allows for a full online and mobile voting process.

The use of the *VST* to perform the vote encryption has the additional advantage of protecting the voter's privacy from the election server(s), which is a common problem of code voting systems (cf. Section 7). On the other hand, the limited computational capabilities of the *VST* are an extra challenge to the system implementation. To address this challenge we have developed MarkPledge 3 (Joaquim and Ribeiro, 2012), cf. Section 3.1, which is the less computational demanding specification of the MarkPledge technique and therefore suitable for computational constrained devices, such as the *VST*.

For simplicity we assume that the *VST* is a smart card or a USB security token; however, the *VST* may also be a secure element inside a smartphone, e.g. a specific security domain within a UICC (Universal Integrated Security Card); in this case the *PC* role is performed by the smartphone, which means that the voter may vote anywhere using her smartphone.

Summarizing, this paper contribution is a new voting system (EVIV) that gives the voter full mobility and allow her to vote privately in public computers without compromising the integrity of her vote.

The next section gives an overview of the EVIV vote protocol and the privacy and integrity trust models. Section 3 describes the main EVIV protocol building blocks. Then, Section 4 describes in detail the EVIV system architecture and vote protocol. The EVIV vote protocol is evaluated in Section 5. Section 6 describes the EVIV prototype implementation results. An overview of the related work is given in Section 7. Finally, Section 8 gives the conclusions and future work directions.

2. EVIV overview and trust models

In the design of EVIV we assume that each voter has a unique *VST*. It is assumed that once a person reaches the legal age to vote she goes to the local authorities and enrolls in official voters list. After the enrollment, the person (now also a voter) gets a digital voter identification token, i.e. the *VST*, that contains a unique cryptographic key pair (e.g. RSA key pair) which is used to authenticate the voter in subsequent elections, until the key pair expires.

After becoming a registered voter the EVIV electoral process for every election is the following:

1. A few weeks before the election day, the voter inserts her *VST* into a *PC* and connects to the Election Registrar service; there, the voter registers to vote online on the forthcoming election. After a successful registration the voter's *VST* creates a code card, containing one vote code for each candidate and a single vote confirmation code, which is printed using a printer connected to the *PC*. Each vote code and the confirmation code is a small (4–5 characters) text string.
2. On election day, cf. Fig. 1, the voter inserts her *VST* into a *PC* and enters her chosen candidate vote code in the PC that sends it to the *VST* (step ⓐ), which then prepares the vote encryption and a vote receipt; both are sent to the Ballot Box service through the *PC* (step ⓑ). The *PC* shows the vote receipt, containing one verification code for each candidate, to the voter that checks it by verifying that the verification code of the selected candidate matches the confirmation code in her code card (step ⓒ).
3. After the end of the Internet election period, all vote encryptions and vote receipts are made public on a public Bulletin Board. Every voter can then confirm that her vote was cast and recorded-as-intended by checking that her vote receipt is published on the Bulletin Board.

[1] A system with both recorded-as-intended and counted-as-recorded, verifications is said to by E2E verifiable.

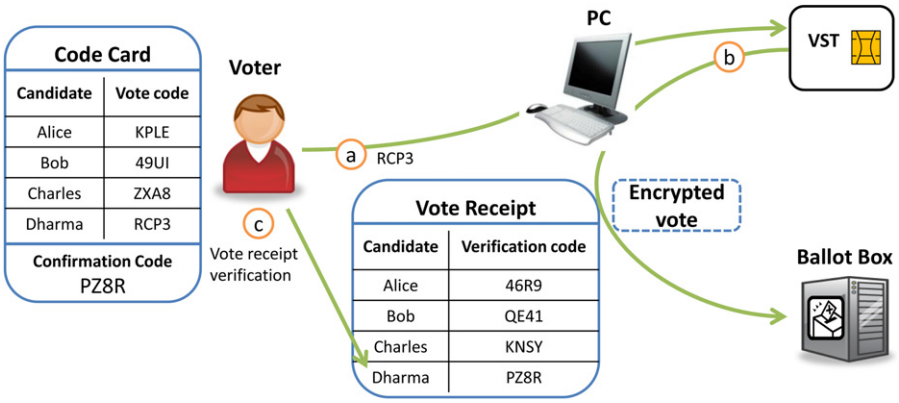

Fig. 1 – Candidate selection and receipt verification overview. In this example the voter selects the candidate Dharma with the vote code *RCP3*. Then, the *VST* creates the vote encryption and receipt and sends them to the Ballot Box through the *PC*. The voter gets and verifies the vote receipt by checking that the confirmation code on her code card (*PZ8R*) is the verification code of the candidate Dharma in the vote receipt.

4. Finally, the election tally is computed, in a privacy preserving and verifiable way, from the data published in the Bulletin Board. The tally and corresponding correctness proofs are also published in the Bulletin Board, allowing anyone to verify the correctness of the election tally (e.g. independent election observer organizations).

2.1. *Properties and trust models*

The EVIV vote protocol demands some changes in the usual voter's electoral process interaction; namely, it requires a pre-election registration and a simple visual vote receipt verification. These changes and the cryptographic techniques used in the EVIV vote protocol, cf. Section 4.3, gives EVIV the following four properties, which are proven in Appendix B:

$P1_{EVIV}$ – No votes can be added, deleted or modified without detection.
$P2_{EVIV}$ – Every vote is counted-as-recorded.
$P3_{EVIV}$ – Every voter can verify that her vote is recorded-as-intended with a soundness of $(1-2^{-\alpha})^{\rho\cdot(k-1)}$.[2]
$P4_{EVIV}$ – No one but the voter and her *VST* knows the voter's chosen candidate.

EVIV guarantees election integrity, i.e. properties $P1_{EVIV}$, $P2_{EVIV}$ and $P3_{EVIV}$, against a single malicious entity (system component or system player) or a collusion of malicious entities under the following Integrity Assumptions (A_I):

A_{I1_EVIV}: There is at least one honest trustee among a set of chosen trustees.[3]
A_{I2_EVIV}: At least one honest organization or entity with cryptographic capabilities will verify the correctness of all the data used in the tally.

[2] The protocol security parameters ρ and α are discussed in Sections 4.3.3 and 3.1.2 respectively. k is the number of running candidates.
[3] The Trustees are defined in Section 4.1.

The privacy of the voter ($P4_{EVIV}$) is protected against a single malicious entity or a collusion of malicious entities, even if the voter votes in a public computer, under the following Privacy Assumptions (A_P):

A_{P1_EVIV}: There is no collusion of more than t out of n trustees, where t and n are configurable security parameters.
A_{P2_EVIV}: The *VST* (which performs the vote encryption) does not disclose the voters' vote choices.
A_{P3_EVIV}: Only the *VST* and the voter have knowledge of the vote codes. This assumption implies that the *PC* used to create the vote codes does not store or disclose them. Note, however, that the vote codes generation may be performed at any chosen time by the voter, long before the election day, and even using an offline *PC*, cf. Section 4.4.

3. Building blocks

This Section describes two building blocks used in the EVIV protocol, namely the MarkPledge 3 vote encryption technique and the shared random number generation protocol.

3.1. *MarkPledge 3*

Due to its complexity, the details of the MarkPledge technique are left outside of the EVIV protocol description. Any of the MarkPledge specifications (Neff, 2004; Adida and Neff, 2009; Joaquim and Ribeiro, 2012) can be used with EVIV; however, EVIV requires the execution of the main cryptographic functions of the MarkPledge technique in a computational constrained device (i.e. the *VST*). Consequently, this section describes the MarkPledge 3 (MP3) specification which is the only viable solution to implement MarkPledge, and therefore EVIV, in a computational constrained device (Joaquim and Ribeiro, 2012).

We start by providing an overview of the MarkPledge technique before entering in the details of MP3. Every vote protocol that uses the MarkPledge technique must follow the following steps (possibly in a different order):

1. Create the vote encryption with one encryption of a *YESvote* for the selected candidate and independent *NOvote* encryptions for each of the not selected candidates. Both the *YESvote* and the *NOvotes* are created using the candidate Vote Encryption function $\mathcal{VE}_{pk}$, which also inserts a random confirmation code in each of the candidate encryptions.[4]
2. Pledge/commit the *YESvote* confirmation code to the voter.
3. Create a random vote encryption challenge value after the vote encryption.
4. Create a vote receipt comprised by one verification code for each candidate (cf. Fig. 1), based on the vote encryption and on the random challenge, using the candidate Receipt Creation function $\mathcal{RC}_{pk}$.
5. Voter visual verification that the pledged/committed *YESvote* confirmation code is the verification code of the selected candidate on the vote receipt.
6. Verify the validity of both the vote encryption and vote receipt using, respectively, the candidate Vote Validity function $\mathcal{VV}_{pk}$ and the candidate Receipt Validity function $\mathcal{RV}_{pk}$.
7. Perform a vote canonicalization of the vote encryption, using the candidate Vote Canonicalization function $\mathcal{C}_{pk}$, and count the votes.

The creation of the challenge value, in step 3, depends of the vote protocol. In the EVIV case we use the shared random number generation protocol described in Section 3.2. The functions referred in 1, 4, 6, and 7 for MP3 are described in detail in Joaquim and Ribeiro (2012). For completeness, the following section provides a brief insight into each of the functions internals as well as their main properties. The length of the confirmation code, verification codes and vote encryption challenge is defined by the MarkPledge security parameter α, usually to a value between 20 and 30 bits (4–5 characters string).

3.1.1. MarkPledge 3 functions details

In the description below all encryptions are performed using the exponential variant of the ElGamal cryptosystem with the public cryptographic key parameters p, q and g and the election public key pk, cf. Appendix A.

3.1.1.1. Candidate vote encryption function $\mathcal{VE}_{pk}$.

$$\mathcal{VE}_{pk}(b,\theta,r) = \langle cvote = \langle u,v\rangle, voteValidity\rangle$$
$$= \langle \langle u = \mathcal{E}_{pk}(b,\tau), v = \mathcal{E}_{pk}(\theta,\delta)\rangle, voteValidity\rangle$$

An MP3 candidate vote encryption $cvote = \langle u,v\rangle$ is composed by two independent encryptions: u is the encryption of either $b = 1$ for a *YESvote* or $b = -1$ for a *NOvote*; v is the encryption of a random confirmation (commit) code ($\theta \in_R \mathbb{Z}_q$), which in the case of a *YESvote* is pledged to the voter.[5]

Both encryptions use exponential ElGamal with the randomization factors τ and δ derived from the input value $r = \tau \parallel \delta$. The *voteValidity* data proves that u is an ElGamal exponential encryption of a value $b \in \{-1, 1\}$. In MP3, the *voteValidity* data is the output of the ballot validity proof protocol of Cramer et al. (1997).

3.1.1.2. Candidate receipt creation function $\mathcal{RC}_{pk}$.

$$\mathcal{RC}_{pk}(b,\theta,\tau,\delta,c) = \langle \vartheta, \omega\rangle$$
$$\vartheta = \begin{cases} \theta & \text{if } b = 1 \text{ (YESvote)} \\ 2\cdot c - \theta \bmod q & \text{if } b = 0 \text{ (NOvote)} \end{cases}$$
$$\omega = \tau\cdot(c - \vartheta) + \delta \bmod q$$

The $\mathcal{RC}_{pk}$ function generates a receipt, i.e. a verification code ϑ. It outputs the confirmation code θ, if the *cvote* is a *YESvote*, or outputs the θ symmetric value, taking c as the symmetry axis, if the *cvote* is a *NOvote*. The ω data is the combination of the randomization factors used in the (*cvote*) u and v encryptions, which is needed to verify that the verification equation results in an encryption of the challenge value c, as described by the $\mathcal{RV}_{pk}$ function below. In order to work in accordance with the ElGamal homomorphic properties, both the ϑ and ω values are computed mod q.

The output of the $\mathcal{RC}_{pk}$ function acts as a window into the encrypted vote, ensuring the voter that the vote machine encrypted the vote correctly, i.e. it encrypted hers intents. Provided that the vote is encrypted and the voter is informed of the chosen θ before c is disclosed, it is not possible to the entity using the function (the VST) to produce anything but the θ for the chosen candidate and some unpredictable value (before knowing c) to the remaining candidates.

3.1.1.3. Candidate vote validity function $\mathcal{VV}_{pk}$.

The candidate vote validity function corresponds to the ballot validity proof in Cramer et al. (1997), $\mathcal{VV}_{pk}(cvote = \langle u,v\rangle, voteValidity) = \{True, False\}$. It is used to ensure that the u component of the *cvote* is in fact the encryption of $b = 1$ or $b = -1$, i.e. it is a valid *cvote*. The function outputs *True* if the *cvote* is valid and *False* otherwise. This function can be used in the middle of the voting process, to ensure the correctness of the vote as soon as possible, or at the end of the election, before the vote counting process.

3.1.1.4. Candidate receipt validity function $\mathcal{RV}_{pk}$.

$$\mathcal{RV}_{pk}(cvote = \langle u,v\rangle, c, \vartheta, \omega) = \mathcal{RV}_{pk}\left(\langle \mathcal{E}_{pk}(b,\tau), \mathcal{E}_{pk}(\theta,\delta)\rangle, c, \vartheta, \omega\right)$$
$$= \mathcal{E}_{pk}(c,\omega) \stackrel{?}{=} u^{c-\vartheta}\cdot v$$

The receipt validity function corresponds to the zero knowledge validation of the verification code ϑ, which can be conducted by any trusted third party by verifying that $u^{c-\vartheta}\cdot v$ is the encryption of c, without any special knowledge but the public values. This is possible by a reconstruction of the encryption of c using the ω encryption factor, revealed by the $\mathcal{RC}_{pk}$ function. For more details on the math behind the receipt verification please refer to Joaquim and Ribeiro (2012).

3.1.1.5. Candidate vote canonicalization function $\mathcal{C}_{pk}$.

$$\mathcal{C}_{pk}(cvote = \langle u,v\rangle, c, \vartheta) = u = \mathcal{E}_{pk}(b,\tau) = canonicalVote$$

The MP3 candidate vote canonicalization goal is to strip the vote from any identifying marks before it is count. The ν part contains the confirmation code θ, which is a number with traceability features once decrypted, but u is just the encryption of $b \in \{-1, 1\}$ that depends only on the vote type and

[4] The specific way in which this code is encoded in the candidate encryption is different on each MP.

[5] For usability reasons, only α bits of the confirmation code (θ) value is pledged to the voter, cf. Section 3.1.2.

nothing more. Therefore, in MP3, the canonicalization function consists only in striping the vote from everything except the encrypted vote type (Yes or No vote).

3.1.2. *Adjusting the voter's view of MP3 output to the α parameter*

The MarkPledge technique has the security parameter α that defines the length of the verification and confirmation codes. Usually, α is set to a value between 20 and 30, which means that the voter must compare 4–5 character strings. However, in MP3 the challenge (c), the verification code (ϑ) and the confirmation code (θ) domains, are defined by the cryptosystem parameter q and not by α. Since the size of q is in the hundreds of bits range we clearly have a usability issue. To solve this usability issue it is proposed a change in the voter's view of the MP3 functions output, namely the voter's view of both the verification code ϑ and the confirmation code θ should be truncated to α bits by applying the mod 2^{α} operation to the referred values.

Assuming a uniform and random distribution of ϑ and θ over $\mathbb{Z}_q$, the voter verification has a statistical soundness of $1 - 2^{-\alpha}$, just because $q \ggg 2^{\alpha}$, i.e. the voter still performs the verification of a random value uniformly distributed over $\mathbb{Z}_{2^{\alpha}}$, cf. (Joaquim and Ribeiro, 2012).

Note that in EVIV the size of the challenge is not a problem because it is generated without the voter's collaboration, cf. Section 4.3.

3.1.3. *Homomorphic vote tally*

MP3 allows the use of an efficient homomorphic vote tally process, using and independent homomorphic vote aggregation for each candidate. Thus, instead of decrypting each vote before counting it, it is performed the homomorphic addition of every encrypted vote $vote^j = cvote_1^j \parallel cvote_2^j \parallel \ldots \parallel cvote(b)_k^j$, where $cvote_i^j = \langle u_i^j, v_i^j \rangle, j = 1 \ldots n$, n is the number of valid votes and k is the number of candidates in each vote.

Given that the vote validity function $\mathcal{VV}_{pk}$ ensures that each $u_i^j = \mathcal{E}_{pk}(1)$ or $u_i^j = \mathcal{E}_{pk}(-1)$, then the vote counting for candidate i will be $count_i = n + d_i/2$, where d_i is the decryption of the homomorphic addition $\oplus_{j=1}^{n} u_i^j$. However, to ensure democracy, the protocol must also guarantee that each vote is counted for only one candidate, which means that the system must ensure that there is only one $u_i^j = \mathcal{E}_{pk}(1)$ in each vote (the *sumValidity* proof in the EVIV protocol, cf. Section 4.3). Once again, given that each u_i^j is the encryption of the value 1 or -1, it is only necessary to prove that $\oplus_{i=1}^{k} u_i^j = \mathcal{E}_{pk}(2-k)$, e.g. using the Chaum and Pedersen (1992) protocol for proving the equality of discrete logarithms, or by revealing the sum of the encryption factors of the u_i^j elements, as suggested for the validation of the c encryption in the $\mathcal{RC}_{pk}$ and $\mathcal{RV}_{pk}$ functions.

3.2. *Shared random number generation protocol*

To ensure the randomness and freshness of the challenge used in EVIV protocol, we use a simple two round random number generation protocol conducted by a set of trustees $\mathcal{T}$. The protocol steps are as follows:

1. In the first step each trustee $t_i \in \mathcal{T}$ secretly generates a random number r_i and commits to it by publishing a signed hash of r_i on a public bulletin board.
2. After the commitment of all trustees, each trustee reveals its random number. Then, all trustees verify the correctness of the commitments published in the first step. If all commitments are correct, the shared random number is computed by applying a bitwise exclusive or to all the random numbers r_i generated by the trustees.

The random generation process is monitored by the electoral commission that validates the process by signing the final generated number and all messages that originated it. It is obvious that if one trustee is honest the random number generated will be fresh and random.

4. EVIV system description

The description of EVIV starts by presenting the system players and their responsibilities (Section 4.1); it continues with the description of the system components and their functionality within the system architecture (Section 4.2); finally Section 4.3 outlines the vote protocol phases. For all these sections please refer to Fig. 2.

4.1. *EVIV system players*

The EVIV system has four system players: the electoral commission, the voter, the trustees and independent organizations.

Electoral Commission (not represented in Fig. 2) is the entity responsible for the entire electoral process; namely, the Electoral Commission is responsible for the voters enrollment system, the actual voting system and the authentication of all election public data.

Voter is any citizen with the right to vote. The voter must enroll once with the Electoral Commission, 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 is cast and recorded-as-intended by performing a simple string match.

Trustees exist in order to share the control over the voter's privacy and the election's integrity among several entities. The trustees can be the political parties and/or any other authorized entity (e.g. an election observer non governmental organization).

Independent Organizations are responsible for independently validate the correctness of the election public data. The immediate candidates for these organizations are the entities directly interested in the election outcome, e.g. political parties; however, any person/organization can perform the role of an Independent Organization and verify the validity and correctness of the election, provided that it has the computational means to do it.

4.2. *EVIV architecture*

The EVIV architecture is constituted by the Enrollment Service, the Election Registrar, the Ballot Box, the Bulletin Board, the Verification Service, the VST and the vote client platform (PC).

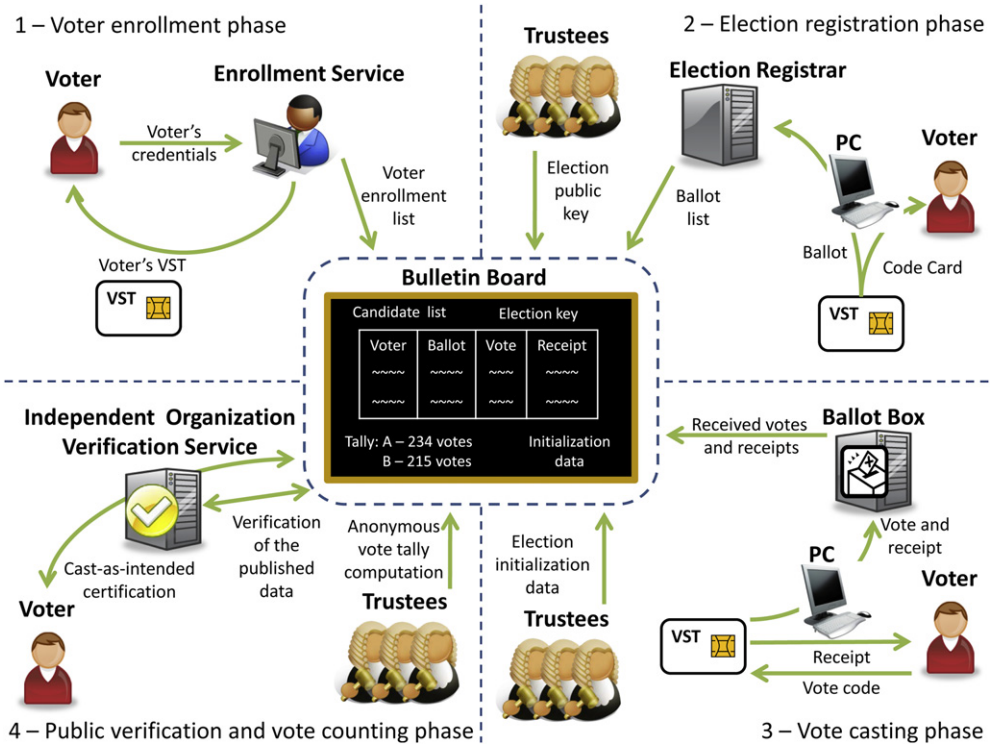

Fig. 2 – Overview of the EVIV vote protocol phases. The phases are presented clockwise starting at the left upper corner with the voter enrollment phase.

Enrollment Service is 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 provides the election registration service that allows voters to register for voting online on a particular election.

Ballot Box provides the vote casting service on election day. The Ballot Box service performs the voter authentication and a vote encryption correctness verification before accepting the vote.

Bulletin Board is the service responsible for the publication of all election public data. The data published cannot be deleted and is always authenticated, i.e. digitally signed.

Verification Service is a service that verifies the correction and validity of votes and receipts. Each independent organization should run an instance of the verification service.

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 the *PC*(s) or any other kind of interaction machine with a *VST* reader (e.g. mobile phone, pda) together with the corresponding operating system and programs used by the voter during the vote protocol.

EVIV supports several instances (possibly on different entities/servers) of all services. In EVIV this is easy because, cf. Section 4.3, all data used to setup the election and compute the election tally is public and authenticated (even the encrypted votes) which facilitates the application of load balancing and fault tolerance techniques. Note also that, the first four services (Enrollment, Election Registrar, Ballot Box and Bulletin Board) may even be run by the same entity/server given that the system is immune to their collusion. This ensures that EVIV may be used from very small elections with just one server, to very large elections with every service replicated many times ensuring both scalability and immunity to lock out problems.

Under the trust models assumptions described in Section 2.1, in EVIV, and with respect to the voter's privacy, we assume that the *VST* and a configurable threshold of trustees (that share the election private key) are honest. On the other hand, regarding the election integrity, we only assume that there is one honest Trustee and one honest Verification Service accessible to the voter.

4.3. *EVIV protocol*

The EVIV's vote protocol builds on top of a shared threshold ElGamal election key pair, cf. Appendix A. The private election key is shared by a set of trustees $\mathcal{T}$ in such a way that only the cooperation of a (configurable) set of t trustees is able to decrypt a message encrypted with the election public key. The votes are encrypted, under the election public key, using the MarkPledge technique.

EVIV's vote protocol is divided in four phases (cf. Fig. 2): 1 – the voter enrollment phase; 2 – the election registration phase; 3 – the vote casting phase; and 4 – the public verification and vote counting phase. The following sections detail these four phases.

4.3.1. *Voter enrollment phase*

The voter enrollment phase consists of an off-line enrollment process where the voter, through the Enrollment Service, identifies herself to the Electoral Commission and receives her *VST* with the voter's private key and a certificate signed by the Electoral Commission. The voter can then use her *VST* to vote in all subsequent elections. The public output of the vote enrollment phase is the publication of a list of all voters' certificates in the Bulletin Board.

1. $\mathcal{V} \rightarrow$ *Enrollment Service* : *voterCredentials*
   To be able to vote the voter ($\mathcal{V}$) must be registered in the electoral roll. This registration is usually performed in person at the local authorities offices using the Enrollment Service provided by the Electoral Commission. The voter starts the enrollment process by going into a local authority office and presenting her credentials, i.e. an identity proof.
2. *Enrollment Service* $\rightarrow \mathcal{V}$: *VST*
   After verifying the voter's identity, the Enrollment Service assigns a *VST* to the voter, e.g. a smart card. The *VST* contains a private key, and the corresponding public key certificate issued by the Electoral Commission. The *VST* key pair is generated inside the *VST*, thus is only known to it. The *VST* key pair now becomes the voter's key pair.
3. *EC* → *BB*: $\{electoralRoll\}_{EC}$
   At some predetermined time before the election, the Electoral Commission (*EC*) uses the Enrollment Service to create the electoral roll, containing the list of voters and corresponding public keys. The electoral roll is signed and published on the public Bulletin Board (*BB*).

4.3.2. *Election registration phase*

The election registration phase is done sometime before the election (e.g. a month), and is divided into two stages. First, there is a setup stage (performed by the Electoral Commission and by the Trustees) to setup election public information/parameters (e.g. candidate list and election key pair). Then, there is a ballot registration stage, where each voter registers her ballot.

4.3.2.1. *Election setup stage*

1. *EC* → *BB*: $\{electionParameters\}_{EC}$, $\{candidateList\}_{EC}$
   The election registration phase starts with the Electoral Commission publishing in the Bulletin Board the election candidate list and the public election parameters, such as: the election date and the election security parameters (e.g. election key pair parameters, cf. Appendix A).
2. $\mathcal{T} \rightarrow BB : \{keyGenerationData, pk\}_{\mathcal{T}}$
   The second step in the election registration phase is the creation of a shared threshold ElGamal election key pair by the set of Trustees $\mathcal{T}$, cf. Appendix A.1. The inputs (cryptographic key parameters) messages, the public outputs of the key generation protocol and the election public key (*pk*) are published in the public Bulletin Board. Each trustee signs her messages before sending them to the Bulletin Board.
3. *EC* → *BB*: $\{pk\}_{EC}$
   The Electoral Commission verifies the election public key generation data, published by the Trustees, and validates the election public key by signing and publishing it on the Bulletin Board.

The voters can now start registering to participate in the election.

4.3.2.2. *Ballot registration stage*.

The voter registers herself to participate in the election by creating and registering a ballot. The process starts with the voter connecting her *VST* to an Internet connected *PC* and establishing a secure connection to the Election Registrar (SSL/TLS connection). In this connection, the voter is authenticated by digital signature means using her private key inside the *VST*, which may require the introduction of a PIN. Then, the following takes place (cf. Fig. 3).

1. *ER* → *PC* → *VST*: $\{candidateList\}_{EC}$, $\{pk\}_{EC}$
   First, the Election Registrar (ER) sends the candidate list and the election public key to the *VST*.
2. *VST* → *PC* → *ER* : $\{ballot\}_{\mathcal{V}}$
   The *VST* creates the voter's ballot, signs it with the voter's private key (which is inside the *VST*), and sends it to the Election Registrar using the *PC* Internet connection.
   An EVIV ballot is comprised of k candidate vote encryptions ($cvote_i$, $i = 1 \ldots k$), in a random order, and the corresponding *voteValidity* proofs, where k is the number of candidates. Each $cvote_i$ and corresponding $voteValidity_i$ proof are created by the MarkPledge vote encryption ($\mathcal{VE}_{pk}$) function, cf. Section 3.1.

$$\mathcal{VE}_{pk}(b_i, \theta_i, r_i) = \langle cvote_i, voteValidity_i \rangle$$

   where b_i, θ_i and r_i are input parameters defining the candidate vote type ($b_i = -1 \rightarrow NOvote$; $b_i = 1 \rightarrow YESvote$) and the secret values needed to verify the vote (θ_i and r_i).
   In the ballot there are $k - 1$ independent *NOvotes* ($b_{i \neq j} = -1$) and one *YESvote* ($b_j = 1$). Additionally, the ballot has a *sumValidity* data proving that there is only one *YESvote* entry in the ballot, cf. Section 3.1.3.

$$ballot = \|_i^k cvote_i \|_i^k voteValidity_i \parallel sumValidity$$

   Notice that, in order to transform the ballot into the vote it is only necessary to rotate the ballot entries such that the $cvote_j$ (the *YESvote*) entry becomes aligned with the selected candidate, cf. Fig. 4, although at this time in the protocol no one but the *VST* knows the position of the *YESvote* in the ballot, because none of the *cvotes* reveals its type.

3. $ER \rightarrow BB : \{\{ballot\}_{\mathcal{V}}\}_{ER}$
   $ER \rightarrow PC \rightarrow VST : \{\{ballot\}_{\mathcal{V}}\}_{ER}$
   Upon the ballot reception, the Election Registrar verifies the ballot correctness by using the $\mathcal{VV}_{pk}(cvote_i, voteValidity_i) = \{True, False\}$ function on each candidate encryption to verify that they actually encrypt a value $b_i \in \{-1, 1\}$; and using the *sumValidity* data to verify that there is only one which encrypts a value $b_j = 1$

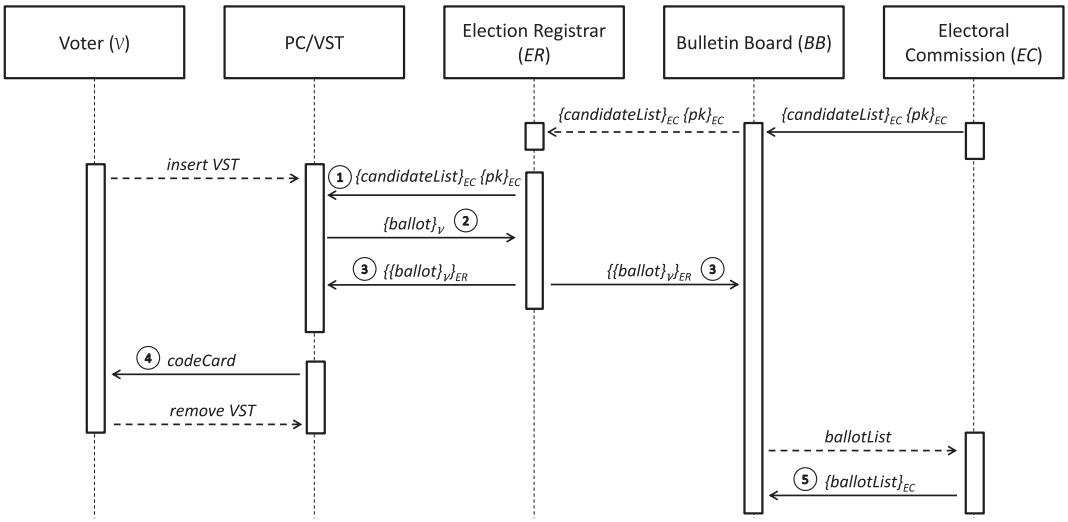

Fig. 3 – Ballot registration stage interaction in the election registration phase.

$$\wedge_{i}^{k} \mathcal{VV}_{pk}(cvote_i, voteValidity_i) \stackrel{?}{=} True \quad (1)$$

$$One\ Yes\ Vote(ballot, sum\ Validity) \stackrel{?}{=} True \quad (2)$$

If both verifications succeed, the Election Registrar validates the *ballot* by signing and publishing it on the Bulletin Board. The Election Registrar signature is also sent to the *VST* as a proof of registration.

After this step the ballot is registered and the connection to the Election Registrar is closed.

4. $VST \rightarrow PC \rightarrow \mathcal{V}$: *codeCard*

Once the ballot registration is confirmed, the *VST* creates a code card for the election. The code card is composed of a vote code for each candidate (selected randomly) and one confirmation code. The confirmation code must correspond to the ballot's *YESvote* secret value (θ_j). The voter then prints the code card or writes it down on a paper and removes the *VST* from the *PC*.

Note that the code card generation step does not require a connection to any entity, i.e. it is performed offline. Thus, as explained in Section 4.4, it could be performed in another *PC* without an Internet connection (e.g. to better protect the voter's privacy).

5. $EC \rightarrow BB$: $\{ballotList\}_{EC}$

Finally, the Electoral Commission verifies all ballots published in the Bulletin Board and validates them by issuing a signature on the list of all published valid ballots.

The Electoral Commission verifies the correctness of each ballot the same way the Election Registrar verifies it in the third step of the ballot registration stage.

4.3.3. *Vote casting phase*

The actual vote casting process can be performed anywhere, including public places such as cybercafés and public libraries, without compromising the voter's privacy. The *PC* used to cast a vote will not be able to know or change 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 code card secret.

4.3.3.1. Vote casting initialization stage. Similarly to the election registration phase, the vote casting phase is also divided in two stages: the initialization stage and the actual vote casting stage (cf. Fig. 5). The goal of the initialization stage is to provide a fresh random election challenge, which is essential to the correct use of the MarkPledge technique, and consequently to the end-to-end verifiability of EVIV.

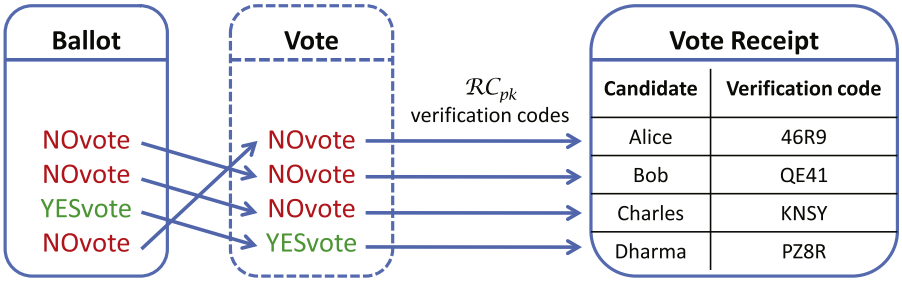

Fig. 4 – This figure illustrates the vote and receipt creation from the ballot entries. The vote is simply the rotation necessary to apply to the ballot entries to align the *YESvote* to the selected candidate. The receipt is the values computed from the ballot/vote entries by the MP $\mathcal{RC}_{pk}$ function.

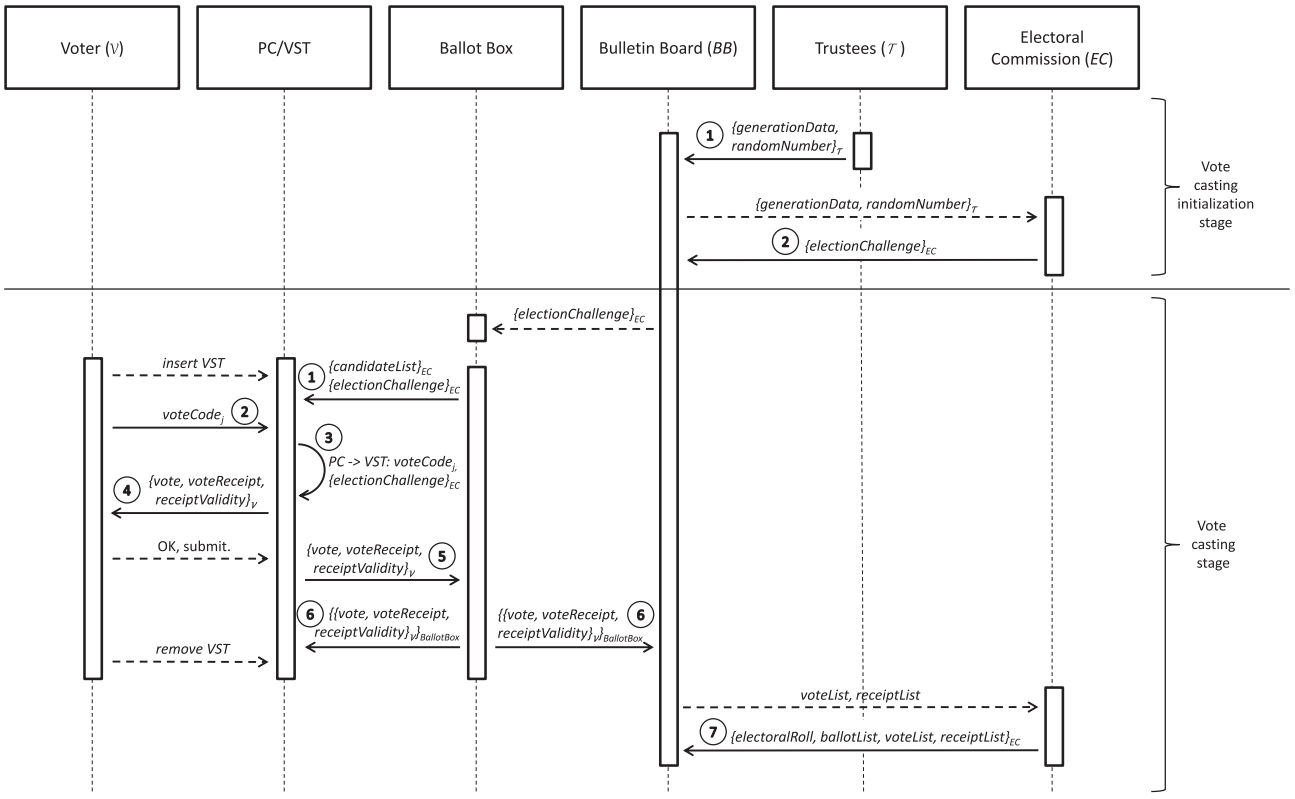

Fig. 5 – Vote casting phase interaction.

1. $\mathcal{T} \rightarrow BB : \{generationData, randomNumber\}_{\mathcal{T}}$

   The vote casting phase is initialized with an election random number generated by the set of Trustees $\mathcal{T}$ (using the distributed random number generation protocol described in Section 3.2) which ensures a random and fresh number provided that at least one trustee is honest.

   The election random number, and the data that originate it, are signed by the Trustees and published in the Bulletin Board.
2. $EC \rightarrow BB$: $\{electionChallenge\}_{EC}$

   The Electoral Commission verifies the election random number generation data and validates it by signing and publishing the election challenge value:

$$electionChallenge = H(randomNumber \| electoral\ roll \| ballot\ list)$$

   where H is a cryptographic hash function.

   After the *electionChallenge* publication the voters can start casting their votes.

4.3.3.2. *Vote casting stage*. The actual vote procedure starts when the voter opens the vote client application, on an Internet connected PC, and establishes a secure and authenticated connection to the Ballot Box (SSL/TLS connection). Then, the following takes place:

1. $Ballot\ Box \rightarrow PC$: $\{candidateList\}_{EC}$, $\{electionChallenge\}_{EC}$

   The Ballot Box starts by sending the *electionChallenge* and the list of candidates to the PC.
2. $\mathcal{V} \rightarrow PC : voteCode_j$

   The PC asks the voter to vote, which she does by typing the vote code of her chosen candidate ($voteCode_j$) that is found in her code card next to her chosen candidate ($candidate_j$) (step ⓐ in Fig. 6).
3. $PC \rightarrow VST$: $voteCode_j$, $\{electionChallenge\}_{EC}$

   The PC then forwards the vote code and the election challenge to the VST (step ⓑ in Fig. 6).
4. $VST \rightarrow PC \rightarrow \mathcal{V} : \{vote, voteReceipt, receiptValidity\}_{\mathcal{V}}$

   After receiving the vote code, and if the $voteCode_j$ is part of the valid vote codes on the voter's code card, the VST prepares the voter's vote and receipt from the corresponding ballot and the *electionChallenge* value. This process is illustrated in Fig. 4 and described next:

   (a) First, the VST creates the vote by computing the rotation necessary (l times) to apply to the ballot entries to align the *YESvote* entry with the selected candidate. The encrypted vote is simply the ballot entries rotation.

$$vote = \left\| _{i}^{k} cvote_{(i+l) \bmod k} \right.$$

   (b) Then, the VST computes the ballot challenge value c by selecting a random number $0 \leq z < \rho$, and then computing $c = \mathcal{F}(electionChallenge, z)$:

$$\mathcal{F}(electionChallenge, z) = \begin{cases} electionChallenge & \text{if } z = 0 \\ H(\mathcal{F}(electionChallenge, z-1)) & \text{if } z > 0 \end{cases}$$

   where H is a cryptographic hash function.

   (c) The VST computes a set of verification codes ϑ_i by applying the MarkPledge receipt creation $\mathcal{RC}_{pk}(b_i, \theta_i, \tau_i, \delta_i, c) = \langle \vartheta_i, \omega_i \rangle$ function to the encrypted vote entries (rotated ballot entries) cf. Section 3.1.

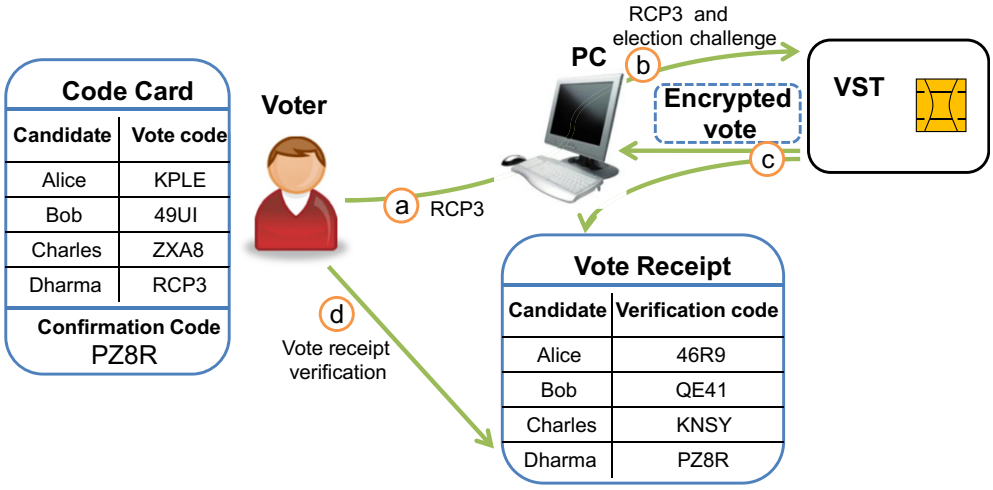

Fig. 6 – Candidate selection and receipt verification overview. In this example the voter selects the candidate Dharma with the vote code RCP3. Then, the VST creates the vote encryption and receipt and sends it to the PC, which displays the vote receipt to the voter. The voter verifies the vote receipt by checking that the confirmation code on her code card (PZ8R) is the verification code of the candidate Dharma in the vote receipt.

The $\mathcal{RC}_{pk}$ function takes the random r_i secret value, known only to the *VST*, and the challenge generated in the previous step to generate a verification code for each candidate encryption and a proof of correct generation (ω_i). The concatenation of the verification codes and the corresponding proofs create the vote receipt ($voteReceipt = ||_i^k \vartheta_i$) and receipt validity ($receiptValidity = ||_i^k \omega_i$) data.

In order to ensure voter recorded-as-cast verification, the challenge must not have been known at the time of the ballot creation; however, because of this requisite, the existence of two equal verification codes in the receipt is a possibility. If this happens the vote receipt is considered invalid and the *VST* goes back to step (b). The probability of having an invalid receipt after ρ attempts can be set as small as desired (cf. Appendix B).

After creating the vote out of the ballot and generating the vote receipt and the respective validity data, the *VST* signs the triplet (vote, voteReceipt, receiptValidity) and sends it to the *PC* (step © in Fig. 6) that displays it to the voter for confirmation (step ⓓ in Fig. 6).

5. $PC \rightarrow Ballot\ Box : \{vote, voteReceipt, receiptValidity\}_{\mathcal{V}}$

If the voter confirms the correction of the receipt, by checking that the confirmation code on her code card matches the receipt's verification code for the selected candidate, the signed triplet is sent to the Ballot Box using the *PC* Internet connection.

6. $Ballot\ Box \rightarrow BB : \{\{vote, vote\ Receipt, receipt\ Validity\}_{\mathcal{V}}\}_{Ballot\ Box}$
$Ballot\ Box \rightarrow PC \rightarrow VST :$
$\{\{vote, vote\ Receipt, receipt\ Validity\}_{\mathcal{V}}\}_{Ballot\ Box}$

Upon the reception of the signed triplet the Ballot Box verifies the vote and corresponding receipt correctness, using the MarkPledge receipt validity ($\mathcal{RV}_{pk}$) function (cf. Section 3.1) and verifies that the ballot rotation that "creates" the vote is in accordance with the vote receipt entries.

$$\wedge_i^k \mathcal{RV}_{pk}(cvote_i, c, \vartheta_i, \omega_i) \stackrel{?}{=} True \quad (3)$$

If everything is OK, the Ballot Box signs the verified data and publishes it in the public Bulletin Board. A copy is sent back to the *VST*.

In order to simplify a second and independent receipt verification, the voter can print the receipt and ask an Independent Organization to verify it. This verification may be conducted at any time, during or after the vote casting phase. In fact, every vote and receipt pair are verified by Independent Organizations at the "public verification and vote counting phase".

7. $EC \rightarrow BB$: $\{electoralRoll, ballotList, voteList, receiptList\}_{EC}$

Immediately after the vote casting period, the Electoral Commission verifies all receipts validity data using the MarkPledge $\mathcal{RV}_{pk}$ function, similarly to the Ballot Box verification (cf. Eq. (3)). Then, the Electoral Commission validates all votes and receipts by publishing a signature over the list of all the voter-ballot-vote-receipt associations.

Note that the Electoral Commission verification can be performed in real time as the vote-receipt pairs get published in the Bulletin Board, thus the message corresponding to this step can be published without any further delay at the end of the vote casting period.

4.3.4. *Public verification and vote counting phase*

Finally, there is a public verification and vote counting phase where the Trustees perform an anonymous homomorphic vote tally computation. In this phase, anyone can verify all the election public data, such as the ballots, the receipts validity proofs and the election tally computation, without compromising the voters privacy. Independent Organizations provide Verification Services to allow the voter to perform an independent verification of her vote receipt.

The public verification and vote counting phase has three stages: i) election data verification, ii) vote tally, and iii) vote tally verification.

4.3.4.1. Election data verification

1. $IO_i \rightarrow BB : \{electoralRoll, ballotList, voteList, receiptList\}_{IO_i}$

   This phase starts with a first verification of all public data by the various Independent Organizations. Each Independent Organization (IO_i) verifies the ballots validity and the corresponding published votes and receipts, using the MP verification functions $\mathcal{VV}_{pk}$ and $\mathcal{RV}_{pk}$ (cf. Eq. (1) in the 3rd step of the ballot registration stage in the election registration phase and Eq. (3) in the 6th step of the vote casting stage in the vote casting phase). The Independent Organizations also verify that there is only one *YESvote* in each ballot similarly to the verification performed by the Election Registrar in step 3 of the ballot registration stage in the election registration phase (cf. Eq. (2)).

   Each Independent Organization publicly commits to the election data verification by publishing a signature on the Bulletin Board over all the voter-ballot-vote-receipt associations.

2. $\mathcal{V} \rightarrow Verification\ Service_i : voterID$
   $Verification\ Service_i \rightarrow \mathcal{V} : \{verifiedReceipt\}_{IO_i}$

   The voter can verify independently her vote receipt by asking the Verification Service of any Independent Organization for a verified copy of her vote receipt. Then, the voter should again check if the confirmation code on her code card matches the verification value corresponding to her chosen candidate on the 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 (Estonian National Electoral Commitee, 2012) allows the voter to vote on a polling station at the election day and override 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.

4.3.4.2. Vote tally

1. $EC \rightarrow BB$: $\{homomorphicVotesAggregation\}_{EC}$

   The vote tally starts with the publication of the homomorphic aggregation, by the Electoral Commission, of all the votes that were not protested by the voters. The homomorphic vote aggregation is as described in Section 3.1.3 and aims to protect the voter's individual privacy, i.e. no individual vote is decrypted, only the votes aggregation that is homomorphically computed.

2. $\mathcal{T}_t \rightarrow BB : \{voteTally, decryptionProof\}_{\mathcal{T}_t}$

   Then, a subset $\mathcal{T}_t$ of at least t Trustees decrypts the homomorphic votes aggregation in a verifiable way, cf. Appendix A.1. The vote tally and the decryption proof are then signed by the Trustees and published in the Bulletin Board.

3. $EC \rightarrow BB$: $\{homomorphicVotesAggregation, voteTally, decryptionProof\}_{EC}$

   The Electoral Commission verifies the vote tally decryption proofs and validates it by publishing a signature linking the homomorphic votes aggregation to the final vote tally and decryption proof.

4.3.4.3. Vote tally verification

1. $IO_i \rightarrow BB$:
   $\{homomorphicVotesAggregation, voteTally, decryptionProof\}_{IO_i}$

   Each Independent Organization verifies the homomorphic vote aggregation and vote tally decryption proofs, signs everything and publishes the signature on the Bulletin Board.

   The homomorphic vote tally aggregation can be verified just by redoing the homomorphic sum (cf. Section 3.1.3). The decryption proof is verified according to the threshold decryption algorithm used (cf. Appendix A.1). Note that anyone with sufficient knowledge and computational power can verify all the election data as Independent Organizations do. This is true because the verification is based only on public information published in the Bulletin Board.

4.4. A note on the code card generation

Printing the code card while registering the ballot in the election registration phase is convenient for the voter; however, this means that, besides the voter and the *VST*, also the *PC* used in the registration has access to the code card. Thus, the registration *PC* can compromise the voter's privacy because it knows the confirmation code; therefore, it must be trusted in the EVIV privacy trust model cf. Section 2.1. Nevertheless, because the voter verifies her vote receipt using a trusted independent organization, not even a collusion between the registration *PC* and the vote casting *PC* is able to compromise the vote-receipt verification soundness.

To better protect the voter's privacy, the code card can be generated using an independent (and offline) *PC* any time after the voter's ballot registration and before the beginning of the vote casting phase.

5. Protocol evaluation

This section discusses the EVIV properties introduced in Section 2.1, for which we provide detailed proofs in Appendix B. Additionally, we also discuss the coercion resistance limitations of EVIV and, briefly, the EVIV properties that simplifies the design of defenses against common network infrastructure attacks.

$P1_{EVIV}$ – No votes can be added, deleted or modified without detection.

In EVIV an attacker (insider or not) cannot modify votes nor add votes for abstaining voter's because the votes and the receipts are validated by the voter's digital signature, i.e. it is not possible to create a valid vote or receipt without the voter's *VST*. Given that the votes and the receipt are published in the *BB* every voter can easily verify that her vote enters the tally process, i.e. it is not deleted.

$P2_{EVIV}$ – Every vote is counted-as-recorded.

In EVIV every vote is counted-as-cast because it uses a public verifiable homomorphic vote tally process.

$P3_{EVIV}$ – Every voter can verify that her vote is recorded-as-intended with a soundness of $(1 - 2^{-\alpha})^{\rho \cdot (k-1)}$.

The vote and receipt are created and signed by the voter's *VST*; thus, only the *VST* and the *PC* (to which the *VST* is

connected) can try to manipulate the voter's vote.

Assuming that the voter has access to a certified/verified vote receipt, the correct use of the MarkPledge technique in the EVIV vote protocol allows the voter to identify any vote manipulation by the *VST* with a soundness of $(1-2^{-\alpha})^{\rho\cdot(k-1)}$, cf. the proof in Appendix B. The *PC* can also try to modify the voter's vote by guessing a valid candidate vote code; however, given that it cannot create a fake vote receipt (i.e. it cannot forge the voter's signature on a fake receipt) the voter easily detects the attack by visually verifying the vote receipt.

P4$_{\text{EVIV}}$ – No one but the voter and her *VST* knows the voter's chosen candidate.

The encrypted vote, the vote receipt and the homomorphic tally process data are all available in the *BB* and do not reveal the voter's vote intention. Assuming that the *VST* is honest, there is an honest threshold of the election key holders and that the code card is only known to the voter and her *VST*, then we can say that no one but the voter and her *VST* knows the voter's chosen candidate.

Note that the *VST* is only considered honest to preserve the voter's privacy. As explained above, the EVIV integrity evaluation considers the *VST* one not trusted system component.

5.1. *Coercion and receipt freeness*

Even considering the *VST* honest for the privacy evaluation, EVIV is not coercion resistant nor receipt free because the voter can provide a proof of her vote to a coercer/person. The proof is simply the voter's code card, namely the verification code on her code card. However, in EVIV the code card is not authenticated, i.e. it is written on a paper by the voter or printed on demand by the voter using a regular printer and paper; thus, after voting, the code card is not a proof any more, i.e. the voter can create a fake code card with any verification code that appears on the receipt. Moreover, although not specified in Section 4.3, the EVIV system can be easily extended to support vote recasting, given that both the encrypted vote and receipt are public and authenticated by the voter's digital signature. In the EVIV case, and because the vote receipt is public and authenticated, the vote recasting only defeats weak forms of coercion and vote buying attacks.

5.2. *Network infrastructure attacks*

EVIV, as any other network protocol, may be subjected to network infrastructure attacks. Although, EVIV does not specify any security measures to counteract this type of security attack, it possesses properties that may simplify the design of such security measures.

DoS attacks against the election infrastructure with the intent to lock-out one or several electors can be mitigated by replicating the Election-Registrar and Ballot-Box services. Given that these services are stateless and that every ballot or vote submission is authenticated and, therefore allowing revoting, most of the problems associated with replication (Dini, 2003) do not exist and replication is easy.

Phishing for credentials or vote codes are ineffective in EVIV and therefore do not require special defense mechanisms. In fact, authentication credentials never leave the *VST* and code votes can only be used by someone with the *VST*. Spoofing the identity (DNS, IP, etc.) of election services is also ineffective provided that at least one verification organization is not spoofed (cf. integrity assumption **A$_{\text{I2_EVIV}}$**).

6. Implementation results

This section discusses the implementation results regarding the technical viability of EVIV, i.e. the performance of its critical components. It starts by identifying the time-critical operations in EVIV and then presents the results of a prototype implementation of such time-critical operations.

6.1. *Time-critical operations*

Given the cryptographic nature of EVIV, the time/computational critical operations are those directly related to the vote encryption, receipt creation and verification, and vote tally computation:

- Vote encryption and verification: $\mathcal{VE}_{pk}$ and $\mathcal{VV}_{pk}$
- Receipt creation and verification: $\mathcal{RC}_{pk}$ and $\mathcal{RV}_{pk}$
- Tally computation: $\mathcal{C}_{pk}$ and homomorphic tally computation

Fig. 7 shows a schematic view of the EVIV protocol, where it is possible to identify the entity responsible for each cryptographic operation and the dependencies between the cryptographic operations.

The trustees are in charge of creating the election key pair and deciphering the homomorphic vote aggregation. These two operations are easily performed within a few seconds/minutes (cf. Section 6.2).

Independent Organizations verify all the election's public data, namely: the election key pair creation, each vote's encryption, receipt and canonicalization, the homomorphic tally aggregation and the homomorphic vote aggregation decryption. These operations can be spanned across the entire election period after the publication of the related data by the Electoral Commission; therefore, provided that they can be performed in a single day, usual minimum election period, we consider the costs acceptable.

The Electoral Commission/election servers also verify all the election's public data; however, these verifications are critical because the data must be verified before being accepted and published. The two most time critical operations are the vote receipt verification and the election tally computation (vote canonicalization, homomorphic aggregation and tally decryption verification). The vote receipt verification time is critical because the voter's vote is only accepted after the receipt verification, and the election tally time computation is critical because everyone demands a fast tally output.

In EVIV each voter's *VST* only has to create one vote/receipt pair ($\mathcal{VE}_{pk}$ and $\mathcal{RC}_{pk}$) however, we assume that the *VST* has very limited computational capabilities and must do its computations in a human acceptable time. Our prototype implements the *VST* as a smart card, cf. Section 6.2.1.

6.2. *Prototype results*

After the identification of the critical operations we have implemented a prototype to evaluate the viability of EVIV, using the MP3 ballot/receipt verification technique (cf. Section 3.1). The implementation uses the "standard" ElGamal setup

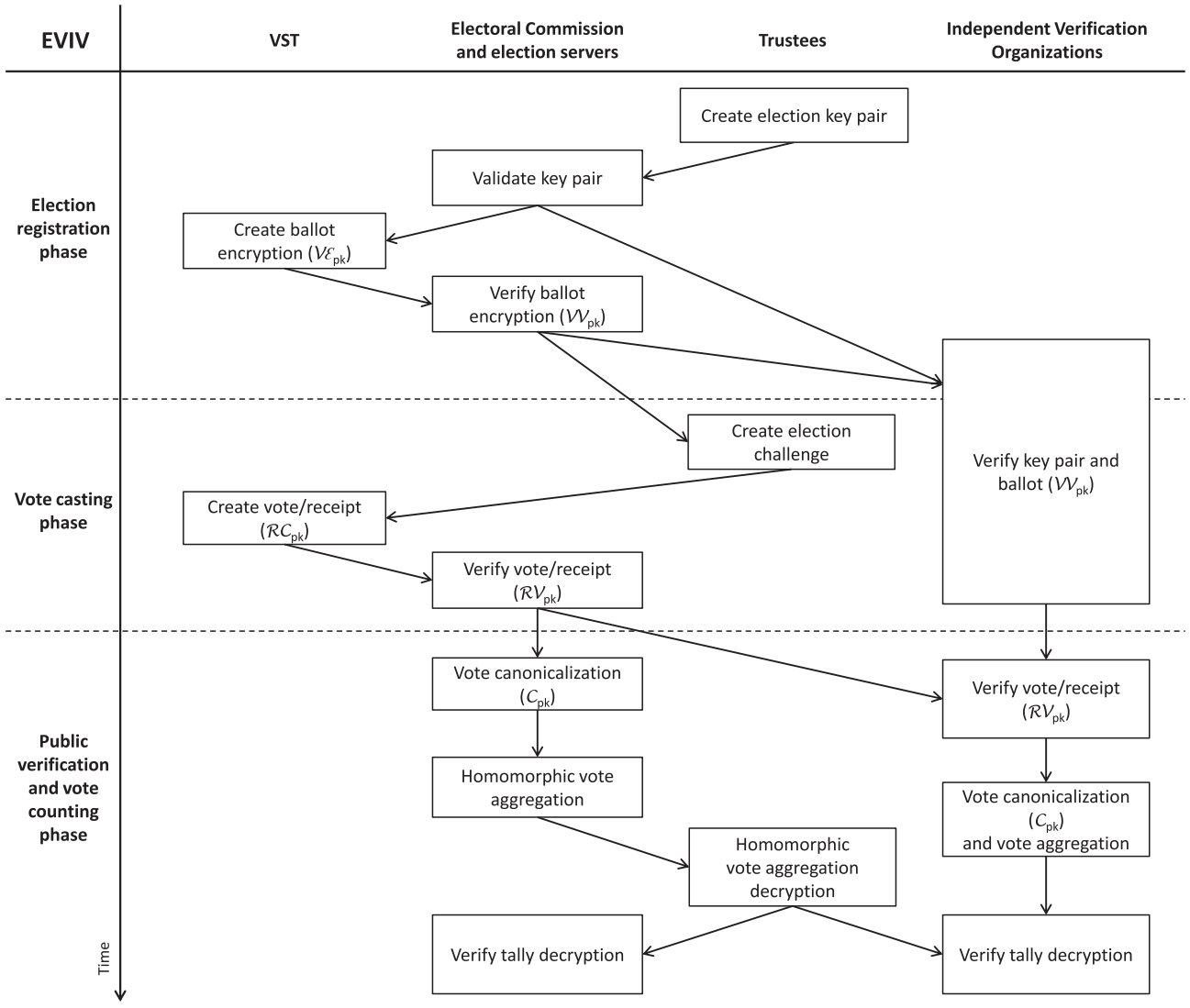

Fig. 7 – Schematic view of the EVIV protocol illustrating the time dependencies between the main cryptographic operations.

over the $\mathbb{Z}_p^*$ subgroup G_q of order q, where p and q are large primes such that $q|p-1$ (cf. Appendix A) more specifically, we performed tests with the following (p, q) setups: (1024, 160) bits and (2048, 256) bits.

The machine we used to evaluate the performance of the operations performed by Election Servers/Electoral Commission, Trustees and Independent Organizations has the following setup: dual Nehalem chipset 5500 LE computer, with two Quad Core Xeon at 2.0 GHz 4 MB Cache processors and 16 GB of RAM, running Linux 64 bits Ubuntu 2.6.32. The test program is coded in Java, compiled to run in native code using the GCJ 4.4.3 ahead-of-time compiler for the Java Language, and run 8 threads. The tests were performed with a 10 candidates and 10,000 voters election setup. The *VST*, which performs the vote encryption and creates the vote receipt is implemented using smart cards (cf. Section 6.2.1).

Our implementation has only one trustee and therefore the election key pair is a "regular" ElGamal key pair. Regarding the vote counting phase, our tests show that the vote encryption aggregation and the aggregation decryption takes just a few seconds: the vote aggregation of 1 million votes is performed in less than 20 s and the decryption of the aggregation is performed in less than 20 ms. The aggregation and decryption times obtained allow us to estimate that the election results can be computed within a few seconds or minutes, even taking into account that the distributed decryption process time is proportional to the number of trustees sharing the private key.

The other relevant cryptographic operations of the EVIV protocol are the MarkPledge functions, which were implemented accordingly to the MP3 specification (Joaquim and Ribeiro, 2012). The vote verification $(\mathcal{VV}_{pk})$ and receipt verification $(\mathcal{RV}_{pk})$ functions, which are performed by both the Electoral Commission/election servers and the Independent Organizations, are analyzed below. The vote encryption $(\mathcal{VE}_{pk})$ and receipt creation $(\mathcal{RC}_{pk})$ functions, which are performed by the *VST*, are analyzed separately in Section 6.2.1.

Table 1 and Figs. 8 and 9 show the server $(\mathcal{VV}_{pk})$ and $(\mathcal{RV}_{pk})$ times per candidate and the total number of vote validations

Table 1 – Server vote and receipt verification times for the ElGamal (p-q) parameters configurations of (1024-160) bits and (2048-256) bits. Columns 2 and 3 show the per candidate verification times, while columns 4–6 show how many (million) votes and receipts verifications can be performed in 24 h (with our server setup) in an election with 10 candidates.

Parameters (p-q) bits	1 candidate		10 candidates votes in 24 h		
	$\mathcal{VV}_{pk}$	$\mathcal{RV}_{pk}$	$\mathcal{VV}_{pk}$	$\mathcal{RV}_{pk}$	$\mathcal{VV}_{pk} + \mathcal{RV}_{pk}$
1024-160	292 μs	141 μs	29.59 M	61.28 M	19.95 M
2048-256	1346 μs	647 μs	6.42 M	13.35 M	4.34 M

that can be performed, in 24 h, with our test setup. Note that, due to the MarkPledge technique, the total time per vote/receipt is proportional to the number of candidates running in the election. From the data shown it is possible to infer that in the (2048-256) bits configuration, a server like ours, in an election with 10 candidates, can perform about 70 vote validations or 150 receipt validations per second, which we consider acceptable for the Electoral Commission/election servers real time constrains. Using the same configuration, Independent Organizations can perform a total of 4.34 million validations in 24 h of both vote and receipts.

The times for weaker and outdated (1024-160) bits configuration are given to show the performance relation between the two configurations. This data is important to the *VST* performance analysis, given that there was no available off-the-shelf developer smart cards supporting the (2048-256) bits configuration (cf. Section 6.2.1). On average the (2048-256) bits configuration results are 4.6 times greater than the (1024-160) bits configuration results.

6.2.1. *VST prototype implementation results*

The *VST* is a personal security token that, for convenience, should be familiar to the voters. In our prototype the *VST* is implemented as a smart card, similar to today bank cards with chip, electronic national id cards or electronic passports.

We have considered two smart card technologies for our *VST* implementation, namely JavaCard (Oracle, 2012) and MULTOS (MULTOS, 2012). Both JavaCard and MULTOS define secure multi-application smart-card virtual machines with a well defined application programing interface (API); however, the JavaCard API is much more restricted and limits the access to the smart card crypto-coprocessor functions. On the other hand, the MULTOS offers a "lower" level API which gives a broader access to the crypto-coprocessor capabilities, including access to large integer modular arithmetic. Thus, when it is necessary to prototype protocols with non standard cryptographic operations (e.g. the MarkPledge candidate encryptions) the MULTOS platform is the best choice (Joaquim and Ribeiro, 2012; Mostowski and Vullers, 2011).

Our *VST* prototype is implemented in a MULTOS MC1-36K-61 smart card with an Infineon SLE66 chip, in which we have tested the vote creation $\mathcal{VE}_{pk}$ and receipt creation $\mathcal{RC}_{pk}$ functions. Table 2 and Figs. 10 and 11 show the $\mathcal{VE}_{pk}$ and $\mathcal{RC}_{pk}$ times. Although MULTOS smart cards are more flexible than JavaCards, we were unable to get access to development cards with 2048 bits "free" exponent modular exponentiation; the

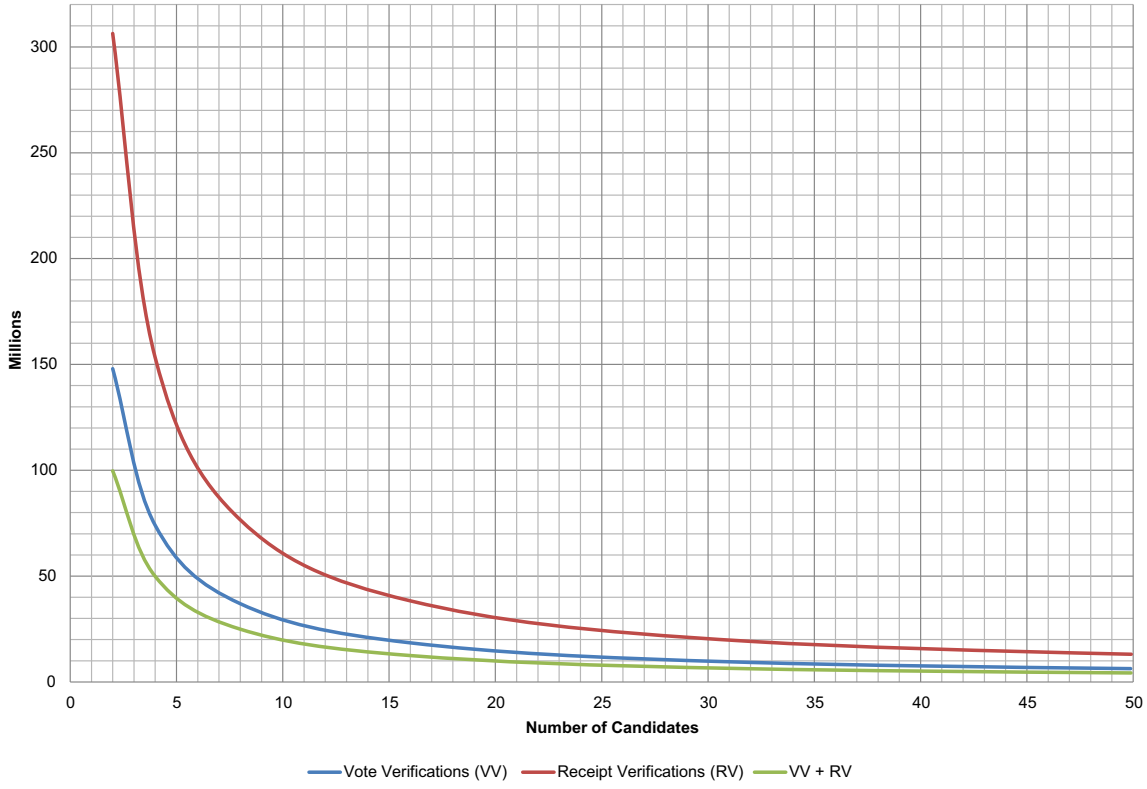

Fig. 8 – Vote and receipt verifications at the server over a 24 h period with a 1024-160 bits configuration. This graph is based on the data presented in Table 1.

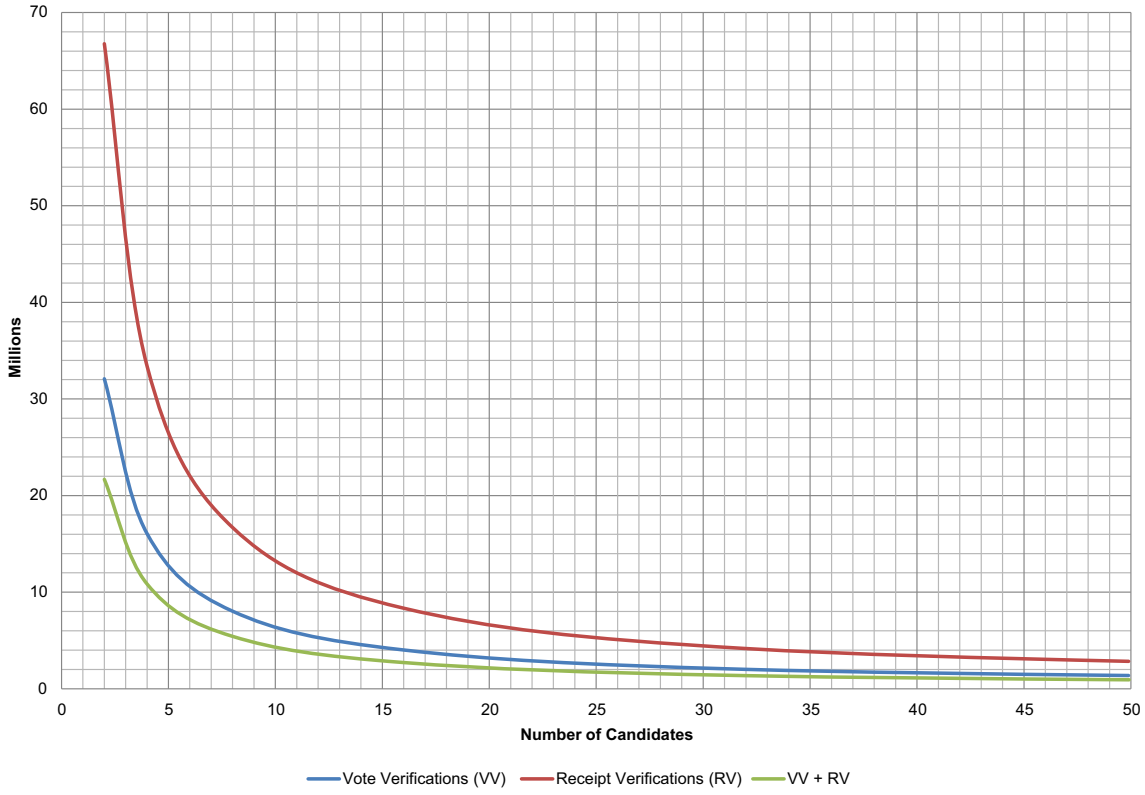

Fig. 9 – Vote and receipt verifications at the server over a 24 h period with a 2048-256 bits configuration. This graph is based on the data presented in Table 1.

available cards only supported 2048 bits "free" modular exponentiation with a 32 bits exponent. Thus, the times shown in Table 2 are real times for the (1024-160) bits configuration and estimated for the (2048-256) bits configuration in a similar hardware (more details below).

As expected, the receipt creation is much faster than the vote encryption. In our development cards the receipt creation function, which is necessary to perform during the usually short vote casting period, takes only 0.065 s in the (1024-160) bits configuration and is estimated to take about 0.267 s in the (2048-256) bits configuration, which allows to create a vote receipt in less than 3 s for an election with 10 running candidates. The vote encryption, which is performed in the election registration phase, prior to the election day, takes 2.8 s in the (1024-160) bits configuration and is estimated to take about 11.5 s in the (2048-256) bits configuration; thus, it would take about 2 min to encrypt a vote for an election with 10 running candidates. In addition, it is important to note that the time constrains in the election registration phase are looser because this phase can span over a larger period of time (a few weeks).

Table 2 – Per candidate vote encryption ($\mathcal{VE}_{pk}$) and receipt creation $\mathcal{RC}_{pk}$ times in a MULTOS MC1-36K-61 smart card, for the ElGamal (p-q) parameters configurations of (1024-160) bits and (2048-256) bits. The (2048-456) bits configuration times are estimated to be about 4.1× the times of the (1024-160) bits configuration.

Parameters (p-q)	1 candidate			
	$\mathcal{VE}_{pk}$			$\mathcal{RC}_{pk}$
	cvote	*voteValidity*	Total	
1024-160	1.2 s	1.6 s	2.8 s	0.065 s
2048-256[a]	4.9 s	6.6 s	11.5 s	0.267 s

a The (2048-256) bits configuration times are an estimative.

Given that there was no available MULTOS developer smart card able to support the (2048-256) bits configuration we did some tests to confirm the around 4× factor between configurations observed in the server times, cf. Section 6.2. Our tests consisted in implementing the modular exponentiation, for both configurations, using a mix of direct hardware functions and software. It was implemented the "exponentiation by squaring" algorithm given its simplicity and the availability of a direct modular multiplication function up to 2048 bits. Table 3 shows the times obtained and the estimative for the hardware only modular exponentiation times. The times obtained show a factor of 4.1× between the two configurations, which is in line with the 4.6× factor observed in our server times.

Although not available as a developer card, there are MULTOS cards with more powerful hardware, which, accordingly to the analysis in Mostowski and Vullers (2011), could cut the times by 30% in the 1024 bits configuration and present a factor of 2× to 2.5× when going from 1024 bits to 2048 bits (RSA) exponentiations. Considering the study presented in Mostowski and Vullers (2011) and our times for the (1024-160) bits configuration we expect that, with the (2048-256) bits configuration, a single candidate vote encryption can

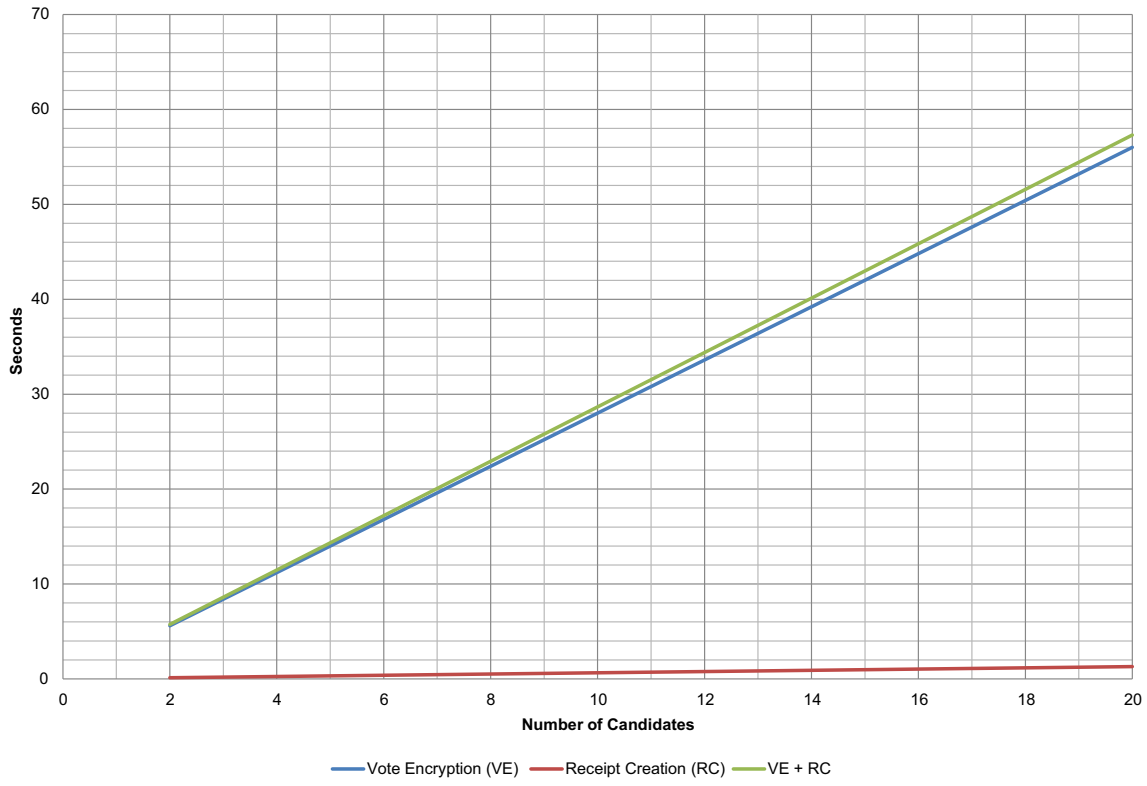

Fig. 10 – Vote and receipt creation times at the *VST* with a 1024-160 bits configuration. This graph is based on the data presented in Table 2.

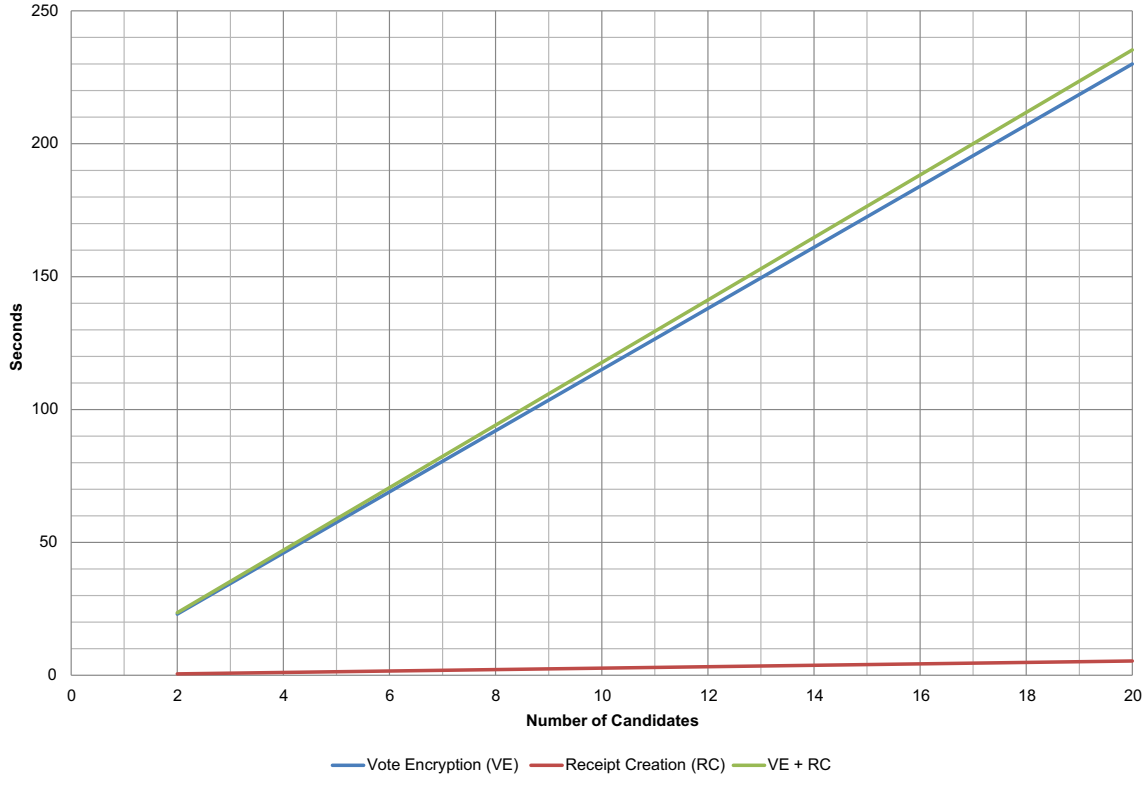

Fig. 11 – Vote and receipt creation times at the *VST* with a 2048-256 bits configuration. This graph is based on the data presented in Table 2.

Table 3 – Modular exponentiation times in a MULTOS MC1-36K-61 smart card. The hardware times use the API modular exponentiation function. The hardware & software times reflect our implementation of the modular multiplication algorithm "exponentiation by squaring" using the hardware assisted modular multiplication function.

Modular multiplication		
Parameters (p-q)	Hardware	Hardware & software
1024-160	72 ms	1940 ms
2048-256	296 ms[a]	7970 ms

a Time estimative.

be done in about 5 s in a state-of-the-art smart card, thus taking about 50 s to encrypt a full 10 candidates vote.

7. 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 accordingly to the vote anonymization technique used: mix-nets (Chaum, 1981; Juels et al., 2005; Furukawa et al., 2010), blind signatures (Chaum, 1988; Ohkubo et al., 1999; Joaquim et al., 2003) and homomorphic voting systems (Cohen and Fischer, 1985; Cramer et al., 1997; Juels et al., 2005; Kiayias et al., 2006).

Usually vote protocols assume a "cryptographic capable" voter, however the incapacity of a common voter to perform cryptographic operations means that the voter must trust the vote client machine to perform the voter's side cryptography, e.g. the vote encryption. This fact and the vulnerability of current Internet "architecture/infrastructure", which exposes the voter's computer to many threats, e.g. a computer virus aiming to undetectably change the voter's vote, is one of the main arguments against the adoption of Internet voting (Jefferson et al., 2004; Dagstuhl Accord, 2007).

In 2004, with the work of Chaum (2004) and Neff (2004), a new paradigm in electronic voting research has emerged: E2E voting systems. The goal of an E2E voting system is to incorporate both voter cast-as-intended verification and universal counted-as-recorded verification mechanisms to allow the election's integrity verification based only on publicly published data. The E2E voting systems were initially proposed to the poll station voting environment (Chaum, 2004; Popoveniuc and Hosp, 2010; Chaum et al., 2005; Neff, 2004; Adida and Neff, 2006; Moran and Naor, 2006; Benaloh, 2006; Rivest and Smith, 2007; Chaum et al., 2008a,b). Later, some of the ideas were used to develop E2E Internet voting systems.

The Helios voting system was the first E2E Internet voting system (Adida, 2008). Helios borrows the cast-as-intended verification mechanism of Benaloh (2006), where the voter is allowed to create n votes and verify $n-1$, achieving a soundness of $1-1/n$. Although the Helios cast-as-intended verification mechanism resists to the collusion of all system components, it is easy to explore a mix of social engineering and vote client PC vulnerabilities to bypass it (Estehghari and Desmedt, 2010; Heiderich et al., 2012).

Another E2E voting system proposed for the Internet is the Scratch, Click and Vote (SCV) system (Kutyłowski and Zagórski, 2010). SCV provides a mix-net universal vote count and uses the voter cast-as-intended verification ideas of Popoveniuc and Hosp (2010), Chaum et al. (2005) and Rivest and Smith (2007) with a "blind signature glue". The SCV cast-as-intended mechanism has a soundness of $1-1/4k$, where k is the number of candidates. The cast-as-intended verification mechanism and the privacy of the voter can be broken if the two election servers of SCV collude.

The remaining three Internet E2E voting systems we are aware of (Ryan and Teague, 2009; Joaquim et al., 2009; Heiberg et al., 2010) use a code voting (Chaum, 2001; Oppliger, 2002) style voter interaction, combined with some cryptographic techniques to provide privacy and voter recorded-as-intended verification.

The Pretty Good Democracy (PGD) achieves E2E verifiability by enhancing a code voting protocol with some ideas used in the Chaum et al. (2008a) and Chaum et al. (2005) systems. PGD have some coercion resistance (receipt-freeness) at the cost of a vote verification mechanism that can be bypassed by a collusion of two election servers. The PGD system was later enhanced to support expressive voting schemes in which the voter lists the candidates in order of preference (Heather et al., 2010).

The VeryVote system (Joaquim et al., 2009) combines the code voting approach with the highly sound Neff (2004) cast-as-intended verification mechanism (MarkPledge). The cast-as-intended verification mechanism used in VeryVote offers a soundness of $1-k!/2^{\alpha}$, where α is a configurable security

Table 4 – Comparison between EVIV characteristics and other Internet E2E voting systems.

	Internet E2E					
	Helios (v3)	PGD	VeryVote	SCV	Heiberg et al.	EVIV
Highly sound voter cast-as-intended verification	×	×	✓	×	✓	✓
Cast-as-intended verification resistant to the collusion of all system components	✓	×	✓	×	×	✓
Protects voter's privacy from a compromised voting PC	×	✓	✓	✓	×	✓
Voters' privacy is not broken by a simple collusion of system components	✓	✓	×	×	×	✓
Universal tally verification	✓	✓	✓	✓	✓	✓
Full voter's mobility	✓	×	×	×	×	✓
Strong voter's authentication	×	×	×	×	✓	✓

parameter usually set to a value from 20 to 30 and k is the number of candidates. In VeryVote all vote encryptions are prepared by the election server which, consequently, must be trusted to ensure the voter's privacy.

In the Heiberg et al. (2010) system, which uses only verification/confirmation codes, each verification code is generated by a cooperation of two servers in order to protect the privacy of the voters. The soundness of the cast-as-intended verification mechanism depends on the size of verification code and can be as high as desired. In this system the privacy of the voter is broken if the two election servers collude, and the cast-as-intended verification mechanism can be broken if both the two elections servers and the vote casting PC collude. Norway (Gjsteen, 2012) uses a similar system, although due to privacy/coercion concerns not enough data is made public, which makes it not E2E verifiable.

With the exception of Helios, all other E2E Internet voting systems require the delivery of a code card to the voter prior to the election, e.g. by postal. As can be seen in Table 4, none of the described E2E voting protocols has all the properties of EVIV.

8. Conclusions and future work

EVIV 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. Moreover, the EVIV protocol does not requires auditing computer systems, only the data produced 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 public proof of correctness.

In the distributed voter's privacy model of EVIV no entity is able to break the voters privacy since each encrypted vote is created by each voter's *VST* (voter security token). 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; although, this does not provide protection against coercion/vote buying attacks, in which the voter gives/sells her code card to the coercer/vote buyer. A very important future work is to consider the integration of strong anti coercion mechanisms in EVIV.

The voter cast-as-intended verification in EVIV is highly sound and requires the voter to perform only the match of two small 4–5 alphanumeric strings. More precisely, the soundness of the voter cast-as-intended verification is $(1-2^{-\alpha})^{\rho\cdot(k-1)}$, where k is the number of running candidates and α and ρ are configurable security parameters usually set to values between 20 and 30, and 1 and 5, respectively.

Our prototype implementation shows that there should not be any problem with the computational demands of the vote protocol at the servers side. However, the assumed *VST* limited computational capabilities limits the use of EVIV to elections with a small number of candidates. This problem should be addressed in the future, e.g. using elliptic curve cryptography.

Another important future work is to test the usability of EVIV and extend its application range by enhancing the voter cast-as-intended verification mechanism in order to support multiple candidate selection and candidate ranking with the same high soundness.

Acknowledgments

This work was partially supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/113993/2009, PTDC/EIA-CCO/122542/2010 and PEst-OE/EEI/LA0021/2011. Rui Joaquim was partially supported by Ministério da Ciência, Tecnologia e Ensino Superior grant PROTEC SFRH/BD/50135/2009.

Appendix A. The ElGamal cryptosystem

For completeness, this appendix describes the well known ElGamal cryptosystem that is used in the MarkPledge specifications and consequently in EVIV. The ElGamal cryptosystem works in the $\mathbb{Z}_p^*$ subgroup G_q of order q, where p and q are large primes such that $q|p-1$. Both primes p, q and a generator g of G_q are public parameters of the system. The ElGamal key pair consists of a private key s and the corresponding public key $h = pk = g^s \bmod p$. The private key s is a randomly chosen integer such that $0 < s < q$. Algorithms to generate secure ElGamal parameters can be found in (NIST, 2009).

In the ElGamal cryptosystem, a message $m \in G_q$ is encrypted by selecting a random integer value $r \in \mathbb{Z}_q$, and constructing the following pair $\mathcal{EG}_h(m,r) = \langle x, y\rangle = \langle g^r \bmod p, h^r \cdot m \bmod p\rangle$. To recover the message m one computes $m = y/x^s$.

The EVIV protocol uses an ElGamal variant known as exponential ElGamal (Cramer et al., 1997). In exponential ElGamal the message to encrypt m is chosen from $\mathbb{Z}_q$ and it is encrypted as g^m, instead of m, in order to respect the ElGamal message space, i.e. $\mathcal{E}_h(m,r) = \mathcal{EG}_h(g^m, r)$. The exponential ElGamal has the following homomorphisms (we have omitted the mod p notation from the equations):

Additive homomorphism between two encryptions

$$\begin{aligned}\mathcal{E}_h(m_1,r_1)\oplus\mathcal{E}_h(m_2,r_2) &= \langle g^{r_1}, h^{r_1}\cdot g^{m_1}\rangle \cdot \langle g^{r_2}, h^{r_2}\cdot g^{m_2}\rangle \\ &= \langle g^{r_1}\cdot g^{r_2}, h^{r_1}\cdot g^{m_1}\cdot h^{r_2}\cdot g^{m_2}\rangle \\ &= \langle g^{r_1+r_2}, h^{r_1+r_2}\cdot g^{m_1+m_2}\rangle \\ &= \mathcal{E}_h((m_1+m_2)\bmod q, (r_1+r_2)\bmod q) \end{aligned} \quad (4)$$

Multiplicative homomorphism between an encryption and a value n

$$\begin{aligned}\mathcal{E}_h(m,r)\otimes n &= \langle g^r, h^r\cdot g^m\rangle^n = \langle (g^r)^n, (h^r)^n\cdot(g^m)^n\rangle = \langle g^{r\cdot n}, h^{r\cdot n}\cdot g^{m\cdot n}\rangle \\ &= \mathcal{E}_h((m\cdot n)\bmod q, (r\cdot n)\bmod q)\end{aligned} \quad (5)$$

A.1. Threshold ElGamal

Cryptographic vote protocols in general, and EVIV in particular, share the private key of the election among a set of trustees to protect the voter's privacy. In EVIV, the election private key is shared, among a set of n trustees, in such a way that to decrypt a message it is necessary the collaboration of

$t \leq n$ trustees. In Cramer et al. (1997) the reader can find more details on how to create a (t, n)-threshold election key pair, for the ElGamal cryptosystem, and how to decrypt a message using the shared private key.

Appendix B. EVIV integrity and privacy proofs

This section presents the sketch proofs for the following four EVIV properties, early described in Section 2:

P1$_{\text{EVIV}}$ – No votes can be added, deleted or modified without detection.
P2$_{\text{EVIV}}$ – Every vote is counted-as-recorded.
P3$_{\text{EVIV}}$ – Every voter can verify that her vote is recorded-as-intended with a soundness of $(1 - 2^{-\alpha})^{\rho \cdot (k-1)}$.
P4$_{\text{EVIV}}$ – No one but the voter and her *VST* knows the voter's chosen candidate.

where α is the security parameter that defines the domain $\mathbb{Z}_{2^\alpha}$ of several important protocol parameters: the challenge c, the confirmation (commit) code θ and the verification code ϑ.[6] ρ is the number of challenges that the *VST* can use; and, k is the number of candidates running in the election.

The EVIV proofs use the privacy and integrity trust models defined in Section 2.1 and the following vote and receipt validity definitions:

Definition 1. *A vote is valid if it is the concatenation of k cvotes, one of type YESvote and k − 1 of type NOvote, corresponding to the voter's registered and published ballot.*

Definition 2. *A receipt is correct, with respect to a specific vote* $= cvote_1 \| \ldots \| cvote_k$*, if it is the concatenation of the corresponding k verification codes, i.e.* $\vartheta_1 \| \ldots \| \vartheta_k$.

Definition 3. *A receipt is valid if it is correct and every verification code in it is unique.*

B.1. EVIV integrity proofs

Theorem 1. **(P1$_{\text{EVIV}}$)** *No votes can be added, deleted or modified without detection.*

Proof Sketch. In EVIV only the voter can cast a vote because the vote is only considered for the election tally if it has the voter's signature on it, which is performed by the voter's *VST*. For the same reason no vote can be modified, as it would invalidate the voter's signature on it.

The Electoral Commission signs, and publishes in the Bulletin Board, the vote-receipt list at the end of the vote casting phase, which locks the valid votes accepted for the election tally. Additionally, Independent Organizations also verify the contents of the Bulletin Board and provide a verification service to the voters.

Thus, in EVIV no votes can be added, deleted or modified without detection, in any phase of the protocol.

Theorem 2. *Every publicly recorded vote is valid.*

Proof Sketch. Provided that MarkPledge is not flawed the $\mathcal{VV}_{pk}$ function attests that a *cvote* is either a *YESvote* or a *NOvote*, to anyone knowing the correspondent *voteValidity* data.

After the election registration phase, every $cvote_i$ and $voteValidity_i$ comprising a *ballot* are public, thus, anyone may use the $\mathcal{VV}_{pk}$ to verify that every $cvote_i$ in the *ballot* is either a *YESvote* or a *NOvote*. Given that the *sumValidity* data also becomes public after the election registration phase, every one may attest that only one of the *cvotes*, in the *ballot* is a *YESvote*.

In EVIV the final vote is a simple rotation of the ballot entries, i.e. one of its entries is a *YESvote* and all other entries are *NOvotes*. Thus, if a ballot is valid the vote is also valid.

Lemma 1. *Every published vote is valid and suitable for homomorphic aggregation.*

Proof Sketch. Given that every public vote is valid (Theorem 2) and that by definition every valid vote contains one well formed *YESvote* and $k - 1$ well formed *NOvotes*, the vote is suitable for homomorphic tally if the output of the canonicalization function $\mathcal{C}_{pk}$ is, which is the case for all MarkPledge specifications (Joaquim and Ribeiro, 2012).

Theorem 3. **(P2$_{\text{EVIV}}$)** *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.

Given that no vote can be deleted after being published (Theorem 1), there is at least one honest Independent Organization verifying the election public data (A_{I2_EVIV}), and the votes are suitable for the homomorphic aggregation (Lemma 1) every vote is counted-as-recorded.

Lemma 2. *Every publicly recorded receipt is correct with respect to a valid vote.*

Proof Sketch. Provided that MP is not flawed the $\mathcal{RV}_{pk}$ function attests the correct computation of every verification code ϑ_i from the corresponding pair $\langle cvote_i, c \rangle$.

Given that, after the vote casting phase, the challenge c and every verification code ϑ_i, receipt validity ω_i and $cvote_i$ are public data, anyone can verify that each verification code ϑ_i in the receipt was correctly computed from the reordered ballot entries that compose the vote.

Lemma 3. *The challenge used by $\mathcal{RC}_{pk}$ function 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 (A_{I1_EVIV}), the protocol (cf. Section 3.2) used to generate the challenge ensures that the challenge is fresh and cannot be predicted.

Lemma 4. *The challenge is generated after the commitment of the confirmation code and vote encryption entries.*

[6] This assumption makes the proof valid for every MarkPledge specification; although, as explained in Section 3.1.2 when using MP3 the α parameter only defines the voter's view on those values, which does not affect the soundness of the voter recorded-as-intended verification.

Proof Sketch. In EVIV both the *YESvote* verification code and vote encryption entries are committed in the election registration phase. The *YESvote* verification code is printed as the confirmation code on the voter's code card and the vote encryption entries (*cvotes*) are the ballot entries which are published at the end of election registration phase. Under assumption A_{I1_EVIV}, only when the election registration phase ends, at the vote casting phase initialization, the *electionChallenge* is created by the trustees. Consequently, the challenge is generated after the commitment of the confirmation codes and vote encryption entries.

Lemma 5. *Every verification code computed with $\mathcal{RC}_{pk}$ function in EVIV is a randomly uniform distributed variable.*

Proof Sketch. There are two "types" of verification codes, the ones created from *YESvotes* and the ones created from *NOvotes*. The verification codes (ϑ_i) computed from *YESvotes* are equal to the embedded confirmation codes $(\vartheta_i = \theta_i)$ that, by definition, are uniformly distributed random numbers. The verification codes computed from *NOvotes* are uniform distributed random numbers according with the MarkPledge $\mathcal{RC}_{pk}$ function, provided that the challenge *c* is a uniform random number (Lemma 3) and not known at the time of the *NOvotes* creation (Lemma 4).

Theorem 4. *Every vote and challenge combination has a, public verifiable, probabilistic valid receipt.*

Proof Sketch. A receipt is valid if it is correct and every verification code ϑ_i in it is unique. Under Definition 2 a receipt computed with the $\mathcal{RV}_{pk}$ function is correct and can be publicly proven under Lemma 2. However, because the challenge value is not know at the time of the ballot entries and confirmation code commitment (Lemma 4) it is possible the existence of two equal verification codes in the receipt.

Given that in MarkPledge 3, every verification code is a uniform random variable (Lemma 5) distributed over the MarkPledge code space ([0, 2^α[), the probability of having two colliding verification codes (i.e. an invalid receipt) is given by the birthday paradox probability $p_i \approx 1 - e^{-k\cdot(k-1)/2^{\alpha+1}}$.

In MarkPledge 1 and 2 it is not possible to have two colliding *NOvotes* verification codes, thus the probability that none of the $k-1$ *NOvotes* verification codes in a receipt matches the *YESvote* verification code is $(1-2^{-\alpha})^{(k-1)}$, and the probability of having an invalid receipt is given by $p_i = 1-(1-2^{-\alpha})^{(k-1)}$.

Because the *VST* may extract the verification codes several times the probability that it finds a valid receipt, after a maximum of ρ attempts, is $p_v = 1-(p_i)^\rho$. From p_v it is clear that the probability of success can be made as high as required by increasing the maximum number of attempts ρ.

Given that the receipt is public anyone may verify that every verification code in it is unique.

Theorem 5. *For a valid $\langle vote, receipt\rangle$ pair the EVIV verification process, of the voter intention, is sound with probability $p = (1-2^{-\alpha})^{\rho\cdot(k-1)}$, where k is the number of candidates, α the security parameter of MarkPledge and, ρ 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 *YESvote* entry is in another candidate. Given that the receipt is valid, i.e. every verification code ϑ_i is different and was computed from the corresponding $cvote_i$, the only way that that can happen is if the *VST* guesses the verification code of one of the *NOvotes* in the vote. However, by MarkPledge $\mathcal{RC}_{pk}$ function, before knowing the value of the challenge *c* (Lemma 4), the verification codes of the *NOvotes* are uniformly distributed over the MarkPledge code space ([0,2^α[). Therefore the probability that a dishonest *VST* is able to guess at least one of the $k-1$ verification codes of *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 $k-1$ verification codes of *NOvotes* in one of the ρ attempts of the receipt generation is $q = 1-(1-2^{-\alpha})^{\rho\cdot(k-1)}$.

Theorem 6. ($P3_{EVIV}$) *Every voter can verify that her vote is recorded-as-intended with a soundness of $(1-2^{-\alpha})^{\rho\cdot(k-1)}$.*

Proof Sketch. Given that votes cannot be removed or altered after publication (Theorem 1) and there is at least one honest Independent Organization (A_{I2_EVIV}), 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 2 and 4, respectively). Finally, under Theorem 5, the voter is able to identify the *YESvote* entry in the vote receipt with a soundness $(1-2^{-\alpha})^{\rho\cdot(k-1)}$.

B.2. EVIV privacy proofs

Lemma 6. *The ballot creation process preserves the voter's privacy.*

Proof Sketch. Provided that MarkPledge is not flawed, and under assumptions A_{P2_EVIV}, neither the MarkPledge technique nor the *VST* create implicit channels; therefore, the only outputs of the ballot creation process are the ballot and the ballot validity data, which do not reveal the position of the *YESvote*.

Lemma 7. *The vote casting process preserves the voter's privacy.*

Proof Sketch. Provided that MarkPledge is not flawed, and under assumptions A_{P2_EVIV}, neither the MarkPledge technique nor the *VST* create implicit channels, therefore the only information disclosed in the vote casting process is the vote code of the selected candidate to the *PC* and to the *VST*, and the vote and vote receipt to the public. Assuming again that MarkPledge is not flawed neither the vote nor the receipt reveal the position of the *YESvote* without being decrypted. Under assumption A_{P3_EVIV} only the voter and her *VST* know the association between each candidate and the vote code on the voter's code card. Therefore, revealing the vote code of the selected candidate does not reveal the identity of the selected candidate.

Lemma 8. *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 9. *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 A_{P3_EVIV}) the voter cast-as-intended verification process preserves the voter's privacy.

Lemma 10. *The vote tabulation process preserves the voter's privacy*.

Proof Sketch. The security of the ElGamal cryptosystem and A_{P1_EVIV} guarantees that no individual vote is decrypted. Only the homomorphically aggregated sum of the encrypted votes gets decrypted by the trustees. Therefore, the privacy of the individual voter is preserved by the vote counting process. All tabulation data concerning the homomorphic aggregation and election tally decryption is already public.

Theorem 7. ($P4_{EVIV}$) *No one but the voter and her VST knows the voter's chosen candidate*.

Proof Sketch. Lemmas 6–10 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 7 attest that even the vote code, used to choose the candidate, can be made public without compromising the voter's privacy. Additionally, Lemmas 6–10 under the EVIV privacy trust model prove that there is no privacy risk in any election phase:

- By privacy assumption A_{P3_EVIV} and Lemma 6 there is no privacy risk in the election registration phase.
- By Lemma 7 there is no risk in the vote casting phase.
- By Lemmas 8–10 there in no privacy risk in the public verification and vote counting phase.

Therefore it is possible to conclude that, under the EVIV 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.

REFERENCES

Adida B. Helios: web-based open-audit voting. In: SS'08: Proceedings of the 17th USENIX Security symposium. Berkeley, CA, USA: USENIX Association; 2008. p. 335–48.

Adida B, Neff A. Efficient receipt-free ballot casting resistant to covert channels. In: USENIX EVT/WOTE; 2009.

Adida B, Neff CA. Ballot casting assurance. In: EVT 2006. Berkeley, CA, USA: USENIX Association; 2006.

Benaloh J. Simple verifiable elections. In: EVT 2006. Berkeley, CA, USA: USENIX Association; 2006.

Chaum D. Elections with unconditionally secret ballots and disruption equivalent to breaking rsa. In: EUROCRYPT 88. vol. 330 of LNCS. Springer; 1988. p. 177–82.

Chaum D. Surevote. International patent WO 01/55940 A1. http://www.surevote.com; August 2001.

Chaum D. Secret-ballot receipts: true voter-verifiable elections. IEEE Security and Privacy 2004;2:38–47.

Chaum D, Carback R, Clark J, Essex A, Popoveniuc S, Rivest RL, et al. Scantegrity ii: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In: USENIX/Accurate EVT 08; 2008a.

Chaum D, Essex A, Carback R, Clark J, Popoveniuc S, Sherman AT, et al. Scantegrity: end-to-end voter verifiable optical-scan voting. IEEE Security & Privacy May/June; 2008b.

Chaum D, Pedersen T. Wallet databases with observers. In: CRYPTO '92. vol. 740 of LNCS. Springer; 1992. p. 89–105.

Chaum D, Ryan PY, Schneider S. A practical voter-verifiable election scheme. In: ESORICS 2005. vol. 3679 of LNCS. Berlin/Heidelberg: Springer; 2005. p. 118–39.

Chaum DL. Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 1981;24(2): 84–90.

Cohen JD, Fischer MJ. A robust and verifiable cryptographically secure election scheme. In: SFCS '85: Proceedings of the 26th annual symposium on foundations of Computer Science. IEEE Computer Society; 1985. p. 372–82.

Cramer R, Gennaro R, Schoenmakers B. A secure and optimally efficient multi-authority election scheme. In: EUROCRYPT 97. vol. 1233 of LNCS. Berlin/Heidelberg: Springer; 1997. p. 103–18.

Dagstuhl Accord, http://www.dagstuhlaccord.org/; 2007.

Dini G. A secure and available electronic voting service for a large-scale distributed system. Future Generation Computer Systems 2003;19(1):69–85.

Estehghari S, Desmedt Y. Exploiting the client vulnerabilities in internet e-voting systems: hacking helios 2.0 as an example. In: Proceedings of the 2010 conference on Electronic voting technology/workshop on trustworthy elections. EVT/WOTE'10. USENIX Association; 2010.

Estonian National Electoral Commitee. Internet voting in Estonia, http://www.vvk.ee/voting-methods-in-estonia/engindex/; April 2012.

Furukawa J, Mori K, Sako K. An implementation of a mix-net based network voting scheme and its use in a private organization. In: Chaum D, Jakobsson M, Rivest R, Ryan P, Benaloh J, Kutylowski M, et al., editors. Towards trustworthy elections. vol. 6000 of LNCS. Berlin/Heidelberg: Springer; 2010. p. 141–54.

Gjsteen K. The Norwegian internet voting protocol. In: Kiayias A, Lipmaa H, editors. E-voting and identity. vol. 7187 of Lecture notes in Computer Science. Berlin/Heidelberg: Springer; 2012. p. 1–18.

Heather J, Ryan P, Teague V. Pretty good democracy for more expressive voting schemes. In: Gritzalis D, Preneel B, Theoharidou M, editors. Computer security ESORICS 2010. vol. 6345 of Lecture notes in Computer Science. Berlin/Heidelberg: Springer; 2010. p. 405–23.

Heiberg S, Lipmaa H, Van Laenen F. On e-vote integrity in the case of malicious voter computers. In: Proceedings of the 15th European conference on Research in computer security. ESORICS'10. Berlin, Heidelberg: Springer-Verlag; 2010. p. 373–88.

Heiderich M, Frosch T, Niemietz M. The bug that made me president: a browser- and web-security case study on helios voting. In: Kiayias A, Lipmaa H, editors. E-voting and identity. vol. 7187 of Lecture notes in Computer Science. Berlin/Heidelberg: Springer; 2012. p. 89–103.

Jefferson D, Rubin AD, Simons B, Wagner D. A security analysis of the secure electronic registration and voting experiment (serve), http://www.servesecurityreport.org/paper.pdf; January 2004.

Joaquim R, Ribeiro C. An efficient and highly sound voter verification technique and its implementation. In: Kiayias A, Lipmaa H, editors. E-voting and identity. vol. 7187 of Lecture notes in Computer Science. Berlin/Heidelberg: Springer; 2012. p. 104–21.

Joaquim R, Ribeiro C, Ferreira P. Veryvote: a voter verifiable code voting system. In: E-voting and identity. vol. 5767 of LNCS. Springer; 2009. p. 106–21.

Joaquim R, Zúquete A, Ferreira P. Revs: a robust electronic voting system. IADIS – International Journal of WWW/Internet, http://www.servesecurityreport.org/paper.pdf; December 2003.

Juels A, Catalano D, Jakobsson M. Coercion-resistant electronic elections. In: Proceedings of the 2005 ACM workshop on Privacy in the electronic society; 2005. p. 61–70.

Kiayias A, Korman M, Walluck D. An internet voting system supporting user privacy. In: ACSAC '06: Proceedings of the 22nd annual Computer Security Applications conference. IEEE Computer Society; 2006. p. 165–74.
Krimmer R, Triessnig S, Volkamer M. The development of remote e-voting around the world: a review of roads and directions. In: E-voting and identity. Vol. 4896 of LNCS. Berlin/Heidelberg: Springer; 2007. p. 1–15.
Kutyłowski M, Zagórski F. Scratch, click and vote: E2e voting over the internet. In: Towards trustworthy elections. vol. 6000 of LNCS. Springer; 2010. p. 343–56.
Ministry of Local Government and Regional Development. e-vote 2011-project web site, http://www.regjeringen.no/en/dep/krd/prosjekter/e-vote-2011-project.html?id=597658; September 2012.
Moran T, Naor M. Receipt-free universally-verifiable voting with everlasting privacy. In: CRYPTO 2006. vol. 4117 of LNCS. Springer; September 2006. p. 373–92.
Mostowski W, Vullers P. Efficient U-prove implementation for anonymous credentials on smart cards. In: Kesidis G, Wang H, editors. Proceedings of the 7th international ICST conference on security and privacy in Communication networks, SecureComm 2011. vol. 96 of Lecture notes of the Institute for Computer Sciences, Social-informatics and Telecommunications Engineering (LNICST). Springer-Verlag; 2011. p. 243–60.
MULTOS. Multos, http://www.multos.com/; September 2012.
Neff CA. Practical high certainty intent verification for encrypted votes, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.1006; 2004.
NIST. Digital signature standard (dss); June 2009. FIPS 186-3.
Ohkubo M, Miura F, Abe M, Fujioka A, Okamoto T. An improvement on a practical secret voting scheme. In: ISW '99: Proceedings of the second international workshop on Information Security. Springer-Verlag; 1999. p. 225–34.
Oppliger R. How to address the secure platform problem for remote internet voting. In: Erasim E, Karagiannis D, editors. 5th conference on "Sicherheit in Informationssystemen" (SIS 2002). Vienna, Austria: vdf Hochschulverlag; October 2002. p. 153–73.
Oracle. Javacard, http://www.oracle.com/technetwork/java/javacard/overview/index.html; September 2012.
Popoveniuc S, Hosp B. An introduction to punchscan. In: Chaum D, Jakobsson M, Rivest R, Ryan P, Benaloh J, Kutylowski M, et al., editors. Towards trustworthy elections. vol. 6000 of Lecture notes in Computer Science. Berlin/Heidelberg: Springer; 2010. p. 242–59.
Rivest RL, Smith WD. Three voting protocols: threeballot, vav, and twin. In: EVT 2007. Berkeley, CA, USA: USENIX Association; 2007. p. 16.
Ryan PYA, Teague V. Pretty good democracy. In: 17th international workshop on Security Protocols; 2009.

Rui Joaquim graduated and obtained his Ms.C. and Ph.D. in Systems and Computer Science in 2002, 2005 and 2012, respectively, from the Universidade Técnica de Lisboa (Instituto Superior Técnico - IST/UTL). Since 2003, he is a Professor at Instituto Superior de Engenharia de Lisboa (ISEL).

He joined the distributed systems group of INESC-ID in 2001, where is he has been doing research on security with a focus on electronic voting systems. He is author or co-author of several peer-reviewed scientific communications.

Carlos Ribeiro graduated in Electrical Engineering at IST in 1989 and on that school obtained his MSc in 1993 and Ph.D. in Computer Engineering in 2002. Since 1993 engaged in teaching at the IST, where was responsible for the creation of several Master and PhD courses in the area of computer security. He has published two technical books in computer architecture and operating systems.

From 1987 he does research on Information Security in the distributed systems group of INESC, Lisbon. He was coordinator of several national and European research projects. He is currently prorector at the technical university of Lisbon.

Paulo Ferreira is Associate Professor with Habilitation at the Computer and Information Systems Department at the Universidade Técnica de Lisboa (Instituto Superior Técnico – IST/UTL), where he has been teaching classes in the areas of Distributed Systems, Operating Systems, Mobile Computing, and Middleware.

In 1996, he received his Ph.D. degree in Computer Science from Université Pierre et Marie Curie (Paris-VI). His M.Sc. (1992) and Bs.E.E. (1988) are both from IST.

He is a researcher at INESC-ID since 1986 and author of more than 80 peer-reviewed scientific communications. He is a member of IEEE, ACM and of the EuroSyS and Midleware steering committees.