Eperio: Mitigating Technical Complexity in Cryptographic Election Verification[*]

Aleksander Essex
University of Waterloo
aessex@cs.uwaterloo.ca

Jeremy Clark
University of Waterloo
j5clark@cs.uwaterloo.ca

Urs Hengartner
University of Waterloo
uhengart@cs.uwaterloo.ca

Carlisle Adams
University of Ottawa
cadams@eecs.uottawa.ca

Abstract

Cryptographic (or end-to-end) election verification is a promising approach to providing transparent elections in an age of electronic voting technology. In terms of execution time and software complexity however, the technical requirements for conducting a cryptographic election audit can be prohibitive. In an effort to reduce these requirements we present Eperio: a new, provably secure construction for providing a tally that can be efficiently verified using only a small set of primitives. We show how common-place utilities, like the use of file encryption, can further simplify the verification process for election auditors. Using Python, verification code can be expressed in 50 lines of code. Compared to other proposed proof-verification methods for end-to-end election audits, Eperio lowers the technical requirements in terms of execution time, data download times, and code size. As an interesting alternative, we explain how verification can be implemented using TrueCrypt and the built-in functions of a spreadsheet, making Eperio the first end-to-end system to not require special-purpose verification software.

1 Introduction

Voting and cryptography make strange bedfellows. One has been the manifest duty of the members of free societies since antiquity. The other is a specialized and mostly recent discipline of mathematics. As such, designing a *cryptographically verifiable election* is as much of a technical pursuit as it is a negotiation between scientists and citizenry.

In the experience of ourselves and others, cryptographic election verification has faced criticism and resistance engendered by an ostensible lack of understandability as a result of the use of cryptography. We therefore consider improving understandability among the general public a critical direction of research and that as a step in this direction, we must examine approaches to reduce the technical complexity of implementing and conducting cryptographic election audits.

In this paper we present Eperio, a protocol for cryptographic election verification with this goal in mind. In its basic form, verifying an election with Eperio involves an auditor downloading a set of encrypted audit files from an election website, opening a subset of them by entering a password and comparing the files to one another for consistency using a spreadsheet application or small software script. We believe this can greatly simplify the typically in-depth technical component of cryptographic election verification, especially relative to related systems.

We approach the design of Eperio in two related settings. Firstly, the system should be secure under standard computational assumptions, while using cryptographic primitives that are simple and efficient. Secondly, the verification software of the desired system should be implementable, wherever possible, using either familiar software tools, or custom open-source software with an emphasis on compact code size.

Contributions The contributions of this paper include:

- The Eperio cryptographic election verification protocol and a proof of its security,
- A proposal for implementing cryptographic commitments using symmetric-key encryption,
- A software implementation of the Eperio system, which includes,
  - An election verification script written in 50 lines of Python,
  - An election verification procedure using TrueCrypt and OpenOffice Calc spreadsheet application as an alternative to an automated script,
- A performance analysis of Eperio demonstrating that it requires fewer cryptographic operations, smaller audit datasets, less execution time and fewer

[*] An extended abstract of this paper appeared at Electronic Voting Technology/Workshop on Trustworthy Elections 2010.

lines of software than other recently deployed cryptographic election verification systems.

2 Challenges and Approaches to Voting with Electronics

Electronic vote-counting methods have become increasingly prominent. By 2006, less than 1% of the United States voting-age population still voted by hand-counted paper ballot, with over 88% of the votes being tallied either by optical-scan, or some other form of electronic ballot tabulation [22]. However, software execution is not inherently observable, which has generated widespread concern over its use in elections. Furthermore, beginning in 2004 with Kohno *et al.* [33], independent reviews have continued to uncover serious security vulnerabilities in widely-used commercial systems; vulnerabilities that allow, among other things, undetectable tally manipulation if an attacker gains access to the system under reasonable threat scenarios. Recent studies have examined systems by ES&S [5], Premier/Diebold [11, 12], Sequoia [4, 10], and Hart-InterCivic [11,29].

In response, a return to hand-counted paper ballots has been suggested; however tallying by hand can be time-consuming for the long, multi-contest ballots common in many US precincts. Furthermore, while offering greater transparency than electronic voting, full confidence in an election result would require observing the process end-to-end, which is a large time-commitment, extends to only a single polling place, and is nullified if the ballots are recounted at a later date. We seek a solution that offers full confidence for all precincts and whose audit can be conducted at any convenient time after the election using commonplace computer software.

Our solution rests in cryptographic election verification, which allows voters, and other public organizations, to audit election *results* through the addition of three specialized components in the election process: an enhanced ballot, a privacy-preserving vote receipt, and a cryptographic proof of robustness. Through these components, a voter can be convinced that their vote was counted without being able to reveal to anyone how they voted.

Over two decades after the seminal groundwork was laid by Chaum [14], Benaloh [7] and others, cryptographic election verification has been used in real elections with binding results. An early Internet voting system to use cryptographic verification was RIES, used in a parliamentary election for about 70,000 absentee ballots in 2004 in the Netherlands [28]. In 2007, a university election in Canada used the optical-scan system Punchscan for 154 voters [24]. In 2009, the Internet system Helios was used in a university election in Belgium for 5,142 registered voters [2], and in a university election in the United States for 567 voters.[1] In 2009, the optical scan system Scantegrity was used in a municipal election in the United States for 1,722 voters [13].

Despite these important milestones, cryptographic voting is often criticized, among other things, for being difficult to implement and conduct. The recent municipal election in the United States provides a good example: the two independent auditors of that election each wrote several hundred lines of software code to pore over the two and a half gigabytes of cryptographic audit data.[2] Many systems in the literature employ advanced cryptographic primitives and techniques not typically found in standard software libraries. This often results in the cryptographic software components of such protocol needing to be custom coded, and has pointed us toward being less reliant on custom implementations.

Is simplicity central to the acceptance of cryptographic voting? On the one hand, there are examples of the general population accepting cryptography without understanding the protocols that enable it (e.g., online banking or wireless network privacy). It is also the case that some citizens do not fully appreciate current voting procedures (e.g., statistical recounts or ballot counterfoils) used to add some level of election integrity.

We contend that *universal verification*—the ability of anyone to participate in the election audit—is fundamental to the spirit of cryptographic election audits, and that lowering the technical complexity of the audits follows in that spirit.

2.1 End-to-End Election Verification

Eperio implements what the United States Election Assistance Commission defines as *end-to-end (E2E) verification* [44]. We consider E2E verification to be the conjunction of two properties: voter verifiable robustness and ballot secrecy. The notation of robustness requires that proof, or strong evidence, is made available to any voter that his/her voting intent is included, unmodified, in the election outcome. Simultaneous to this proof, ballot secrecy should maintain anonymous association between a voter and his/her voting intent. The security guarantees of these properties range from computational to unconditional (with robustness and ballot secrecy never being simultaneously unconditional). For the purposes of this paper we pursue robustness and privacy in a computational setting.

As a formal definition of ballot secrecy, Benaloh and Tuinstra define *receipt-freeness* as the property that a

[1] `http://princeton-helios.appspot.com`

[2] Software and audit results available online at `http://github.com/benadida/scantegrity-audit/` and `http://zagorski.im.pwr.wroc.pl/scantegrity/`

voter is not able to provide useful information about their vote after it is cast [9]. This definition is strengthened by Moran and Naor to account for an adversary that interacts with the voter before and after casting [34]. An even stronger notion, *coercion resistance*, is possible to achieve in remote and Internet voting settings [31].

With regards to robustness, a prevalent model for E2E voter verifiable robustness is to ensure three sequential properties. Firstly an auditor must be able to be convinced that if a voter were to attempt to obfuscate their vote, it would be obfuscated correctly (*Marked-as-Intended*). Secondly a voter must be able to be convinced that the obfuscation of their vote is included in a collection of obfuscated votes (*Collected-as-Marked*). Finally an auditor must be able to be convinced that the collection of obfuscated votes is properly deobfuscated, producing a self-consistent and correct tally (*Counted-as-Collected*). When these properties are individually and simultaneously shown to hold on the same data set we say the election was *Counted-as-Intended*.

E2E Ballot and Receipt In addition to being a means for a voter to register their vote, an E2E ballot provides a means for generating a privacy-preserving receipt of their vote. The literature proposes several methods for obfuscating a ballot to create an E2E ballot receipt such as encrypting a reference to the candidate and retaining only the ciphertext (*e.g.,* [1]), applying a randomized permutation to a canonical list of candidates and retaining only the position marked (*e.g.,* [17]), substituting a randomized code for each candidate and retaining just the code (*e.g.,* [16]), and splitting the vote into randomized partial shares and retaining only one of the shares (*e.g.,* [39]).

Although randomized candidate orderings are not legal in all jurisdictions, its simplicity facilitates our discussion. In this paper we will consider a ballot to be a paper ballot similar to the Prêt à Voter system [17] in which the list of candidates is independently shuffled on each ballot. To use this ballot style, a voter first locates their preferred candidate and marks the associated position. The voter then tears off just the marks (Figure 1 illustrates this), which is scanned and then retained as a receipt. The candidate list is shredded by the voter before leaving the polling place. We refer to each position (bubble) on the ballot that can be marked as a unique markable region (UMR). Adapting Eperio to accommodate a range of ballot styles (both paper and electronic) we leave for future work.

2.2 Election Auditors (A Who's Who)

At first glance it may seem necessary for average voters to understand the underlying cryptographic protocol in order to meaningfully participate. Although ideally every participant would possess such a knowledge base, realistically we see it as a *collaboration* between three user types: voter, election auditor and protocol analyst, each of which is associated with different knowledge and technical requirements.

Voter A voter is someone who casts a ballot (and receives a receipt) in an election. The primary responsibility of a voter with regard to E2E verification is to substantiate their contribution to the election using their ballot receipt. At a technical level this requires a voter to confirm the election's audit dataset is consistent with their ballot receipt. Conceptually they should be aware of the E2E properties and what role checking their receipt plays in the broader verification process. A voter may also participate as an election data auditor and/or protocol analyst, or delegate it to someone they trust. Alternatively they may choose to ignore the verification process altogether.

Election Data Auditor An election data auditor undertakes to substantiate the outcome of a *particular* election using the E2E protocol and the public audit dataset for that election. At a technical level they must be able to perform each of the audits either by using existing verification software, or by creating it themselves based on a protocol specification and be able to interpret the results. If using existing verification software, they must trust its correctness (or review the source code). They must understand the technical details of each audit and be convinced, at a high level, that the protocol is sound (i.e., convincing).

Protocol Analyst The role of the protocol analyst is to substantiate the security properties of the E2E protocol to the voters and elections auditors. They must poses sufficient expertise to decide at a formal level whether the protocol is complete, sound and secret.

As we demonstrate in section 7, Eperio simplifies the technical requirements of the election data auditor through smaller data and code sizes, execution times, and wider implementation options.

3 The Eperio Protocol

Eperio is closely related to the Punchscan and Scantegrity cryptographic election verification protocols. Unlike Punchscan and Scantegrity which use a mixnet-like structure to achieve the E2E integrity and privacy properties, Eperio combines its audit data into a single cryptographic table structure. This in turn permits a coarser, more efficient, cryptographic commitment scheme. It also facilitates interesting implementation options such as cryptographic commitments based on file-encryption and E2E verification in a spreadsheet.

As an intuition of this structure consider several ballot boxes, each of which contains a photocopy of each ballot cast in an election. If you shake one of the ballot boxes, the ballots will land with and ordering that is random and independent of the other boxes. However if you were to open that box and tally up the ballots, it will still produce the same winner as all the other boxes. In this way, verifying an election tally with Eperio constitutes proving that each such ballot box is a shuffled copy of (i.e., is *isomorphic* to) every other box.

This physical analogy is inherited from Aperio, a non-electronic, non-cryptographic E2E protocol upon which Eperio is based [23]. Aperio achieves the E2E properties based on physical security assumptions which, in the following sections, we will translate into a cryptographic setting.

3.1 Protocol Sketch

As a brief overview of the protocol, a set of trustees will jointly generate a table with three columns. The first column will contain a unique reference for each markable region (*e.g.*, optical scan bubble) on each ballot in the election. The second column will indicate whether a given region was marked or not marked (or alternatively if the ballot was selected for an audit). Finally the third column will contain the candidate/choice associated with each markable region.

The rows of this table are randomly shuffled (analogous to shaking a ballot box). It is easy to see that if the first two columns are revealed, the information should correspond to the set of all of the receipts in the election. If the last two columns are revealed, the information should correspond to the final tally. If all three columns are revealed (or the contents of an unrevealed column are implied through some functional dependency), then ballot secrecy is compromised. The Eperio protocol proves the correct formation of all three columns by only revealing the information implied in two of the three columns. It uses a composition of cut-and-choose and random audit techniques inspired by randomized partial checking [30].

3.2 Entities

The Eperio protocol relies on the following entities, which are standard in most E2E voting systems:

- A set of n **election trustees** (or the prover), $\mathcal{P}$, tasked with generating a verifiable tally. $\mathcal{P}$ is assumed to be a set of mutually distrustful and non-collusive trustees. However the protocol can tolerate $t \leq \frac{n-1}{2}$ trustees who collude or refuse to participate.
- The set of authenticated **voters** who cast ballots in the election.
- The first set of **verifiers**, $\mathcal{V}_1$, who verify that their receipts were collected correctly (*Collected-as-Marked*). $\mathcal{V}_1$ are either voters, or auditors that were given access to a copy of a voter's receipt.
- The second set of **verifiers**, $\mathcal{V}_2$, who verify that the ballots are printed correctly (*marked-as-intended*). $\mathcal{V}_2$ are either voters or auditors who went in person to obtain a ballot to audit.
- The third set of **verifiers**, $\mathcal{V}_3$, who verify that the tally is computed correctly from the collected receipts (*Counted-as-Collected*). $\mathcal{V}_3$ can include anyone in any location with access to the election data.
- A malicious **adversarial prover**, $\mathcal{P}'$, who will attempt to convince the verifiers that an incorrect tally is correct.
- A malicious **adversarial verifier**, $\mathcal{V}'$, who will attempt to break voter privacy and determine which candidate was selected on a given receipt (or any non-negligible information about this selection).

3.3 Functions

The Eperio protocol requires a set of standard functions from the cryptographic literature: a threshold key agreement modeled after the one due to Pedersen [37], a cryptographically secure pseudorandom number generator (PRNG), a perfect shuffle algorithm, a message commitment scheme (either perfectly binding or perfectly hiding), and a public coin (or random beacon) to generate non-interactive challenges.

3.3.1 Distributed Key Generation/Reconstruction

- ○ $(y_1, \ldots, y_n, \kappa) \leftarrow \mathsf{DKG}(n, t, l, f_1, \ldots, f_n)$
- ○ $\kappa \leftarrow \mathsf{KeyRec}(y_1, \ldots, y_{t+1})$

DKG accepts from each of the n trustees a polynomial of degree t, f_i, with coefficients of bit-length l. The coefficients are summed together, $\mod 2^l$, to produce a new polynomial $\hat{f}$. Each trustee i receives $y_i = \hat{f}(i)$ as their share and the value $\kappa = \hat{f}(0)$ forms a shared secret of bit-length l. t should be set such that $n \geq 2t + 1$. KeyRec accepts at least $t+1$ shares, $y_1, \ldots, y_{t+1}$, from a subset of the trustees and outputs the shared secret κ.

3.3.2 Pseudorandom Number Generation

- ○ $\{0,1\}^l \leftarrow \mathsf{PRNG}(\kappa, l)$

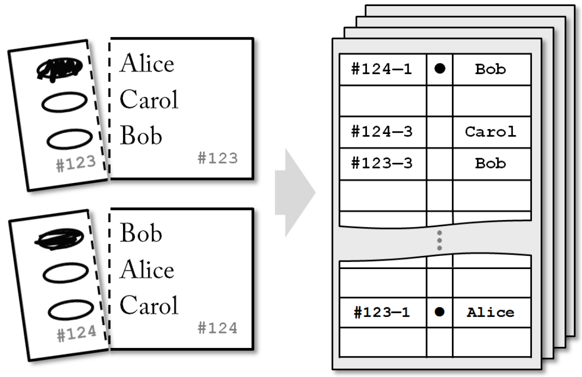


Figure 1: **Left: E2E-enabled optical scan ballots.** Each ballot consists of a unique serial number, a candidate list printed in an independent random order, and a perforation between the optical scan ovals and the candidate list. Upon marking the ballot, the candidate list is detached and shredded. The remaining piece is scanned and granted as a receipt. Because the candidate orderings are independent across ballots, knowing the mark position alone does not reveal how a voter voted. **Right: Eperio table.** Each optical scan oval (referenced by a serial number and absolute position), its mark-state (marked or unmarked) and the corresponding candidate name are recorded in a randomly assigned row.

PRNG is a stateful function which takes as input the shared secret, κ, as a seed and returns l new pseudo-random bits each time it is invoked. For simplicity, we omit l if the size of the output is clear from the context.

3.3.3 Permutation

- $\circ\ \pi(\mathbf{W}_i) \leftarrow \mathsf{Permute}(w_1, \ldots, w_u, \Pi)$
- $\circ\ \pi(\mathbf{W}_i) \leftarrow \mathsf{PermuteBlock}(w_1, \ldots, w_u, s, \Pi)$

$\mathsf{Permute}$ accepts as input a list $\mathbf{W}_i$ of u elements and a number, $\Pi = \mathcal{O}(u \log(u))$, of random bits sufficient to perfectly shuffle the list[3] and returns a list containing elements $w_1, \ldots, w_u$ in permuted order. For some s which divides u, $\mathsf{PermuteBlock}$ applies $\mathsf{Permute}$ independently to each of the u/s non-overlapping sub lists in $\mathbf{W}_i$.

3.3.4 Commitment

- $\circ\ c \leftarrow \mathsf{Commit}(m, r)$
- $\circ\ \{0, 1\} \leftarrow \mathsf{Reveal}(c, r, m)$

Commit takes as input an arbitrary length message m and a random factor r. It outputs a commitment to the message c. Reveal accepts values m, r, and c, and outputs 1 iff c is a valid commitment to m and r. Otherwise it outputs 0.

3.3.5 Public Coin Toss

- $\circ\ \{0, 1\} \leftarrow \mathsf{PublicCoin}()$

$\mathsf{PublicCoin}$ returns a uniformly random bit. The output should be unpredictable prior to being invoked and verifiable *a posteriori*.

3.4 Lists and Tables

The Eperio protocol relies on a particular data structure, called the Eperio table, which is constructed from a set of private inputs by the trustees. The Eperio table is a novel data structure and the primary contribution of this paper. It is shown in Figure 1 with a permutation-style ballot, which we use to illustrate the protocol.

To facilitate clarity, we also denote some intermediate lists and tables used in its construction. We use a bold typeface to denote ordered lists and tables, and a script typeface to denote unordered sets.

The following list and set are public inputs to the system decided on prior to the election.

UMR List: $\mathbf{U}$ is the list of each *unique markable region* (UMR) for the s candidates on each of the b ballots. Elements are encoded as a ballot serial number and a position, and are listed in ascending order. The length of the list is $u = s \cdot b$.

Candidate/Selection Roster: $\mathbb{S}$ is the set of selections or candidates to appear on the ballot, for each contest. Elements are encoded as a character string of arbitrary length. The size of the set is s. Without loss of generality, we assume a single contest.

These are used by the trustees, in conjunction with the functions defined above, to create the following private list.

Candidate/Selection List: $\mathbf{S} \leftarrow \mathbf{U} \times \mathbb{S}$ is the list of candidates for each position on a ballot composed by randomly selecting, without replacement per ballot, an element from $\mathbb{S}$. It is ordered by $\mathbf{U}$. The length of the list is u.

[3] Generating a random integer from random bits is non-deterministic when the integer is not a perfect power of 2. Perfect shuffling algorithms, like Fisher-Yates, require random integers. An upper bound on the *expected* number of bits is $2(\log_2 u!)$.

The marks list will denote the final status of a markable region. It is empty prior to the election and is provided as a public input to the system after every ballot has been cast.

> **Marks List:** $\mathbf{M}$ is the list of marks corresponding to each markable region in $\mathbf{U}$. Elements include marked (1), unmarked (0), and print audited (-1). The length of the list is u.

The concatenation of these three lists defines a table that collects all the private information of the election.

> **Print Table:** $\mathbf{P}$ is a table formed by joining $\mathbf{U}, \mathbf{M}$, and $\mathbf{S}$. The dimensions of the table are $u \times 3$.

A proof of election integrity subsumes a proof that the relations between each list in $\mathbf{P}$ is consistent with a universal view of the election. Pairwise, a correct **U**-**M** relation implies ballots were collected as marked (*Collected-as-Marked*), a correct **U**-**S** relation implies ballots were printed correctly (*Marked-as-Intended*), and a correct **M**-**S** relation implies the ballots were counted as collected (*Counted-as-Collected*). Note that revealing $\mathbf{P}_{(i,j)}$ for all i and j is sufficient, under our assumptions, for independently verifying the correctness of the tally. Unfortunately, this trivial approach would also destroy the privacy preserving property of the ballot receipt—in conjunction with $\mathbf{P}$, receipts would provide proof of which candidate was selected. Instead, we require a non-trivial approach that can both establish integrity and preserve ballot secrecy.

The Eperio table is a data structure that, with a set of queries, can prove the correct formation of $\mathbf{P}$ while maintaining the same level of privacy provided by only revealing the list of receipts and the final tally. Specifically, it is a collection of x instances of $\mathbf{P}$ that have been independently shuffled row-wise. By revealing portions of and relations on this structure, we will show that a complete and robust proof of integrity can be established with this minimal disclosure.

> **Eperio Table:** $\mathbf{E}$ is a table formed by x instances of $\mathbf{P}$, each of them independently shuffled row-wise. The dimensions of the table are $u \times 3 \times x$.

3.5 Protocol

We now outline the protocol for generating an Eperio table and the various outputs required for proving it encodes a correct tally. The focus of this paper is on the protocol for verifying this proof, which is orthogonal to the issue of how the data is generated. However the security proof we provide in the next section encompasses the generation of the data, and so we give it consideration. The first three steps of the protocol are conducted prior to the election: initial setup, generating the Eperio table, and generating the commitments to the Eperio table. These steps are performed with a *blackbox computation* (for more on this primitive, see Section 5).

3.5.1 Initial Setup

The setup assumes that a list of candidates, $\mathbb{S}$, is available as well as the number of ballots, b, to be used in the election. The first task is for the trustees to generate an election secret and receive shares of this secret.

○ $(y_1, \ldots, y_n, \kappa) \leftarrow \mathsf{DKG}(n, t, l, f_1, \ldots, f_n)$

3.5.2 Generate Eperio table

Next, the trustees generate the Eperio table $\mathbf{E}$. $\mathbf{U}$ is formed by listing the ballot and position numbers in order. To form $\mathbf{S}$, the candidate list is repeated b times and then randomly permuted on a ballot-by-ballot basis.

○ $\Pi_S \leftarrow \mathsf{PRNG}(\kappa)$
○ $\mathbf{S}_i \leftarrow \mathsf{PermuteBlock}(\mathbb{S}^b, s, \Pi_S)$

Table $\mathbf{P}$ is created by placing $\mathbf{U}$, $\mathbf{M}$, and $\mathbf{S}$ beside each other in columns. $\mathbf{M}$ is initially empty, but in future meetings will include the marks recorded during the election. $\mathbf{P}$ is used to print the ballots. We use the symbol : to denote an entire vector within a matrix.

○ $\mathbf{P}_{(:,1)} \leftarrow \mathbf{U}$
○ $\mathbf{P}_{(:,2)} \leftarrow \mathbf{M}$
○ $\mathbf{P}_{(:,3)} \leftarrow \mathbf{S}$

Finally, x independent row-wise shufflings of $\mathbf{P}$ are generated. Each shuffled instance of $\mathbf{P}$ is stored in the Eperio table, $\mathbf{E}_{(i,j,k)}$.

```
○ For $k = 1$ to $x$,
    ○ $\Pi_E \leftarrow \mathsf{PRNG}(\kappa)$
    ○ $\mathbf{E}_{(:,1,k)} \leftarrow \mathsf{Permute}(\mathbf{P}_{(:,1)}, \Pi_E)$
    ○ $\mathbf{E}_{(:,2,k)} \leftarrow \mathsf{Permute}(\mathbf{P}_{(:,2)}, \Pi_E)$
    ○ $\mathbf{E}_{(:,3,k)} \leftarrow \mathsf{Permute}(\mathbf{P}_{(:,3)}, \Pi_E)$
```

We define, for future use, the following function. It encapsulates all the steps in this 'generate Eperio table' section.

○ $\mathbf{E} \leftarrow \mathsf{GenE}(\kappa)$

3.5.3 Generate Commitments

The trustees are now ready to commit to the data in $\mathbf{E}$. For each instance $1 < k < x$, they will commit to both the $\mathbf{E}_{(:,1,k)}$ and $\mathbf{E}_{(:,3,k)}$ columns. This requires $x \times 2$ commitments. The random factors for these commitments are stored in an $x \times 2$ table $\mathbf{R}$. The resulting commitment is stored in the corresponding table $\mathbf{C}$.

```
○ For $i = 1$ to 2, $j = 2i - 1$, and $k = 1$ to $x$,
    ○ $\mathbf{R}_{(i,k)} \leftarrow \mathsf{PRNG}(\kappa)$
    ○ $\mathbf{C}_{(i,k)} \leftarrow \mathsf{Commit}(\mathbf{E}_{(:,j,k)}, \mathbf{R}_{(i,k)})$
○ Publish $\mathbf{C}$
```

Note that $2i - 1$ is simply the mapping $\{1 \to 1, 2 \to 3\}$, used to denote that commitments to the first and third columns of $\mathbf{E}$ are stored, respectively, in the first and second rows of $\mathbf{C}$. $\mathbf{C}$ is published to the bulletin board.

3.5.4 Voting

Registered and authenticated voters are issued a paper ballot with a randomized candidate list according to $\mathbf{P}$. After marking the ballot, the candidate list is detached and destroyed (*e.g.,* placed in to a paper shredder). The remaining strip is scanned by an optical scanner and the strip is retained by the voter as a receipt. The optical scanners will record for each ballot which position was marked, as well as the ballots that were print audited. After the election, these are placed into the list $\mathbf{M}$. Without knowing $\mathbf{P}_{(:,3)}$, this information does not reveal which candidate was voted for and can be published.

```
○ Publish $\mathbf{M}$
```

3.5.5 Compute Tally

At least $t + 1$ trustees submit their election secrets to the blackbox computation, which regenerates the key and the Eperio table. This time, the completed marks list is shuffled along with the rest of the table.

```
○ $\kappa \leftarrow \mathsf{KeyRec}(y_1, \ldots, y_{t+1})$
○ $\mathbf{E} \leftarrow \mathsf{GenE}(\kappa)$
```

For each instance x, they publish the corresponding marks list. Each of these lists is a shuffled version of the original $\mathbf{M}$.

```
○ Publish: $\mathbf{E}_{(:,2,:)}$
```

Finally, a tally is computed from any $\mathbf{E}_{(:,2,k)}$ and $\mathbf{E}_{(:,3,k)}$ pair of columns, and the list of totaled values for each candidate, denoted τ, is published. This can be considered an *asserted tally*, as the purpose of E2E verification is to prove that this tally is correct.

3.5.6 Generate the Linkage List

To ensure that the Eperio table is consistent with what actually appears on the printed ballots in the election, verifiers have the ability to keep an issued ballot for purposes of auditing its printing. If a ballot was chosen to be print audited and the first position contained candidate Bob, then a row corresponding to this markable region will exist in $\mathbf{E}$ at an unknown row. The row will be different for each instance. If the ballot is printed correctly, each corresponding row in each instance should contain Bob in the third column. The print auditor would like assurance of this fact.

However since commitments to only entire columns $\mathbf{E}_{(:,1,k)}$ and $\mathbf{E}_{(:,3,k)}$ exist, this fact cannot be directly revealed without revealing both columns for a given instance. Doing this would reveal which candidate was selected for every receipt and cannot be pursued. Instead, the election trustees will indirectly establish this fact. The trustees assert the row number, a, in each instance corresponding to every audited markable region. Which markable regions are audited is contained in the marks list, $\mathbf{M}$, with -1 recorded for that entry. The list of asserted row numbers is called the linkage list, $\mathbf{L}$, and it is made public.

```
○ for $i = 1$ to $u$ and $k = 1$ to $x$:
    ○ if $\mathbf{M}_i$ = -1:
        ○ Find: $a$ s.t. $\mathbf{E}_{(a,1,k)} = \mathbf{U}_i$
        ○ $\mathbf{L}_{(i,k)} \leftarrow a$
```

3.5.7 Audit Challenge and Response

After the tally has been posted, the trustees prove to an independent auditor that the tally was calculated correctly. They do this through a cut-and-choose protocol. First the trustees regenerate the election secret and the Eperio table.

```
○ $\kappa \leftarrow \mathsf{KeyRec}(y_1, \ldots, y_{t+1})$
○ $\mathbf{E} \leftarrow \mathsf{GenE}(\kappa)$
```

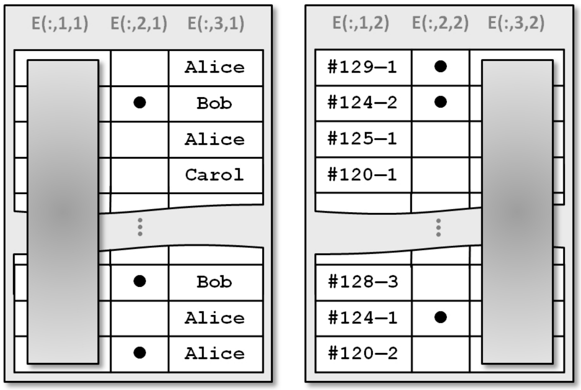
Next, they invoke the public coin toss function to generate one flip for each of the x instances.

```
○ for $k = 1$ to $x$:
    ○ $z \leftarrow \mathsf{PublicCoin}()$
    ○ $\mathbf{Z}_k \leftarrow z$
    ○ Publish: $\mathbf{R}_{(z+1,k)}$
    ○ Publish: $\mathbf{E}_{(:,2z+1,k)}$
```

Each flip is recorded in $\mathbf{Z}_k$. Depending on the flip, they either reveal the first two or last two columns in each instance. Recall that $\mathbf{E}_{(:,2,:)}$ was published previously. This is illustrated in Figure 2.

3.6 Verification

We now show the steps that the verifiers take to check that the published data corresponds to a tally that is correct. Recall there are three sets of audits (and corresponding verifiers). The first set, $\mathcal{V}_1$, are the voters



E(:,1,1)	E(:,2,1)	E(:,3,1)
		Alice
	●	Bob
		Alice
		Carol
	⋮	
	●	Bob
		Alice
	●	Alice

E(:,1,2)	E(:,2,2)	E(:,3,2)
#129–1	●	
#124–2	●	
#125–1		
#120–1		
	⋮	
#128–3		
#124–1	●	
#120–2		

Figure 2: **Auditing Eperio table instances**. Two example instances of the Eperio table during auditing. Each instance alleges to contain the same information, but in an independently shuffled order. **Left:** `E(:,3,1)` was challenged (then revealed), allowing verifiers to tally the election. **Right:** `E(:,1,2)` was challenged (then revealed) allowing voters to check their receipts. The grey bars symbolize cryptographic commitments that will remain unopened to protect ballot secrecy.

who check their receipts (or provide a copy to someone they delegate to check on their behalf). If the receipt corresponds to ballot number b and contains s positions that are either marked (1) or unmarked (0), the auditor should check that the status of each position i on the receipt matches the status recorded at $\mathbf{M}_j$, where $j = s(b-1)+i$.

The second set of verifiers, $\mathcal{V}_2$, should check the linkage list against their print audited ballots. Let $a = \mathbf{L}_{(i,k)}$ for an i on their ballot and an instance k. Depending on the random coin for instance k, $\mathcal{V}_2$ should check that $\mathbf{E}_{(a,1,k)}$ matches the associated markable region on the ballot or $\mathbf{E}_{(a,3,k)}$ matches the associated candidate. They should do this for all i on their print audited ballot and each k in the election.

These two audits establish that the reported marks correspond to what appeared on voter's completed ballots and that what appears on the ballot corresponds to what is in the pre-committed Eperio table. The final step is to ensure that the asserted tally, τ, corresponds to the marks. Recall that for each of the x instances, a random coin was flipped to reveal value $z \in_r \{0,1\}$. The third set of verifiers, $\mathcal{V}_3$, should do the following.

```
○ for $k = 1$ to $x$:
  ○ $z \leftarrow \mathbf{Z}_k$
  ○ Check $\mathsf{Reveal}(\mathbf{C}_{(z+1,k)}, \mathbf{R}_{(z+1,k)}, \mathbf{E}_{(:,2z+1,k)})$
  ○ If $z = 0$:
    ○ Check $\{\mathbf{E}_{(:,1,k)}, \mathbf{E}_{(:,2,k)}\} \cong \{\mathbf{U}_i, \mathbf{M}_i\}$
  ○ If $z = 1$:
    ○ Check $\{\mathbf{E}_{(:,2,k)}, \mathbf{E}_{(:,3,k)}\} \cong \tau$
```

Here $\cong$ means that the two pairs of tables are isomorphic—*i.e.,* they are a permuted representation of the same information.

4 Security

In this section, we summarize the main results of our security proof. See Appendix A for details. Given a transcript of the entire protocol, the asserted tally can be either accepted or rejected. If the asserted tally is correct, the decision will always be to accept (completeness). If the asserted tally is not correct, the decision will be to reject with a high probability (soundness). Finally, the outputs do not provide any information that can be used by a computationally bounded adversary to determine any non-negligible information about which candidate was voted for by any voter (computational secrecy).

Let $\mathcal{P}$ be an unbounded prover (the election authority) and $\mathcal{V}$ be a PPT-bounded verifier. Either entity may employ a malicious strategy and we denote this with a prime $(\mathcal{P}', \mathcal{V}')$. Recall that τ represents the asserted tally and let ρ be the asserted receipts.

Soundness The soundness of Eperio relies on two assumptions:

1. The function $\mathsf{Commit}(m, r)$ is perfectly binding. That is, for any m_1 such that $\mathsf{Commit}(m_1, r_1) = c_1$, there does not exist any r_2 and $m_2 \neq m_1$ such that $\mathsf{Reveal}(c_1, r_2, m_2) = 1$.
2. The function $\mathsf{PublicCoin}()$ is perfectly unpredictable before invocation.

Let b'_r be the number of modified ballot receipts, and $0 \leq p_1 \leq 1$ represent the fraction of voters who conduct a receipt check. Let b'_p be the number of misprinted ballots, and $0 \leq p_2 \leq 1$ be the fraction of ballots that are print audited. Recall there are x instances are in $\mathbf{E}_{(:,:,k)}$, for $1 \leq k \leq x$. Given the above assumptions hold, it is proven (in the Appendix) that the probability of $\mathcal{V}$ rejecting a malformed transcript from $\mathcal{P}'$ is:

$$\mathbf{Pr}[\mathtt{REJECT}_{\mathcal{P}',\mathcal{V}}] = \min[(1-(1-p_1)^{b'_r}), \\ (1-(1-p_2)^{b'_p})(1-\frac{1}{2^x})].$$

Computational Secrecy The secrecy of Eperio relies on the following assumptions:

1. The maximum number of colluding trustees is t.
2. At least one trustee submits to DKG an f_i drawn with uniform randomness from $\mathbb{Z}_l^t$.

3. All outputs are computed with a blackbox.
4. Any polynomial-sized output from $\mathsf{PRNG}(\kappa)$ provides no non-negligible advantage to a PPT-bounded adversary in guessing either κ, the next bit in the output, or any unobserved previous bit.
5. The function $\mathsf{Commit}(m, r)$ is semantically secure and computationally hiding. That is, given either $c_1 = \mathsf{Commit}(m_1, r_1)$ or $c_2 = \mathsf{Commit}(m_2, r_2)$ for any chosen m_1 and m_2, a PPT-bounded adversary should have no non-negligible advantage in guessing which message was committed to.

Let ϵ_{A4} and ϵ_{A5} be the advantage specificied in assumptions 4 and 5. Given all of the assumptions hold, we prove (in the Appendix) that the advantage of $\mathcal{P}'$ recovering non-negligible information about any voter's selection given the full transcript as opposed to just the final tally is:

$$|\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C}, \mathbf{L}, \mathbf{E}_{(:,2,:)}, \mathbf{R}_z) = 1] - \mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\tau)) = 1]| \leq \epsilon_{A4} + \epsilon_{A5}.$$

Additional Claims In addition to the above proofs, we also show that Eperio is complete and can be modified to have everlasting privacy (*i.e.,* $\mathcal{V}$ is unbounded and $\mathcal{P}$ is PPT-bounded).

5 Practical Primitives

In this section, we revisit a few of the cryptographic primitives needed in Eperio. In particular, we are interested in options that allow for useful deployment options.

Blackbox Computation The Eperio protocol requires the generation of the Eperio table to be done using a blackbox computation. While in theory, the task performed by the blackbox could be made into a multiparty computation (where only privacy is required as correctness is provided by Eperio), we instead propose the use of a semi-trusted computer. It is semi-trusted in the sense of only providing *private* evaluation of functions; the *correctness* can be determined through the audit. That said, mechanisms are provided to encourage correct evaluation. To this end, we assume disclosed source code for the functions to be evaluated is provided in advance and some attestation mechanism is available to ensure it is the same code running on the computer.

All tasks performed by the trustees can be accomplished by regenerating the Eperio table, and the regeneration of this table can be accomplished through a threshold of secret shares from the trustees. Therefore the computer should not have any persistent memory and its internal state should be purged after the outputs have been published. While this assumption may seem strong, in each of the recent occasions where end-to-end verifiable voting systems were used in real-life binding elections, semi-trusted computers were deployed: *e.g.,* Punchscan at the University of Ottawa [24], Helios at Université Catholique de Louvain (for key generation from the description in their paper) [2], and Scantegrity at Takoma Park, MD [16].

Cryptographic Commitment We are interested in using symmetric-key file encryption as a commitment function for its speed, simplicity and widespread availability of software implementations. For Eperio, this means putting the columns of the Eperio table into individuals files, encrypting them under a randomly chosen key, and posting the encryption as a commitment. To open the commitment, the encryption key is revealed and the file can be decrypted. While our implementation of Eperio can be easily modified to work with any standard commitment function, we use this approach to simplify the experience for voters who want to verify the proof for themselves. File encryption utilities are readily available, easy to use, and the commitment has message recovery.

Let $\mathcal{E}$ be a pseudorandom permutation (PRP): $\{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^m$. Let M be a message of L m-bit blocks and let IV be a random m-bit initialization vector. Define E to be the cipher block chaining (CBC) mode of encryption that encrypts $M = m_1, \ldots, m_L$ under k-bit key K and m-bit IV. E is defined as: $c_0 = IV$ and $c_i = \mathcal{E}(K, c_{i-1} \oplus \ m_i)$. Assume M is exactly $L \cdot \ m$ bits long and $k = m$. Let D be its inverse decryption function applied to $C = c_0, \ldots, c_L$ under K: $M = D(K, C)$.

Theorem 1: As defined, E is indistinguishable under a chosen plaintext attack (CPA). [6]

Conjecture 2: As defined, D behaves like a pseudorandom function with respect to collisions when C is held constant (note the difference from standard assumptions on M and C with a fixed K). That is, $M \leftarrow D_C(K)$ has random collisions for a fixed C and a variable K.

Theorem 3: Define a *commitment with message recovery* function as $(c, \mathrm{IV}) \leftarrow \mathsf{Commit}(M, K) = E(K, \mathrm{IV}, M \| f(M))$, where E is, as defined, CBC mode with a pseudorandom permutation, and $f(M) =$

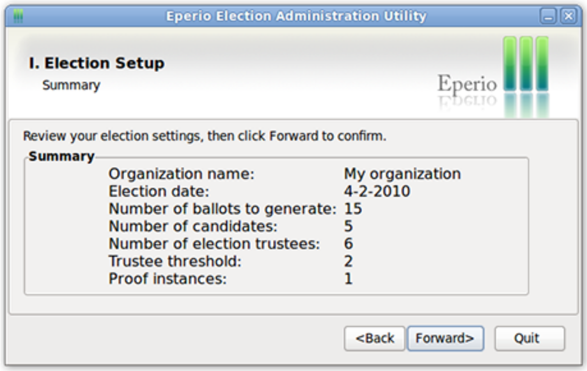
$M\|M$ is a redundancy function.[4] Define $M' \leftarrow \mathsf{Reveal}(C, K) = D(K, C)$. M' is only accepted when M' has the correct form $M\|f(M)$ for some M. We show that such a commitment is computationally hiding under Theorem 1 and statistically binding under Conjecture 2.

We omit a proof of Theorem 3 here but it is included in the full version of the paper. Because the commitment has message recovery, its Reveal function differs slightly from the commitment used earlier. We have demonstrated a very specific statistically hiding commitment function that can be constructed from a block cipher, assuming conjecture 1 holds. We model this ideal functionality in the real-world with AES-128-CBC in the next section.

Public Coin Toss Voting systems often require the use of a public coin for the purposes of fairly implementing the cut-and-choose aspect of the audits. In the case of conventional voting, it is used to select precincts for manual recounts. Cordero *et al.* suggest a protocol using dice [21]. Clark *et al.* note that dice outcomes are only observable by those in the room, and suggest a protocol for auditing E2E ballots using stock market prices [18], which has recently been given a more formal analysis [19]. This was suggested earlier by Waters *et al.* outside of the voting context [42]. A further alternative is to use the Fiat-Shamir heuristic [26], which is secure in the random oracle model. However, a requirement for Fiat-Shamir is that the challenge space is large. In our case, the number of challenge bits is the same as the number of proof instances—for 10 or 20 instances, Fiat-Shamir is not secure. Thus, we use the stock market protocol. The output from a statistically-sound PRNG is seeded with a random extraction of a pre-selected portfolio of closing stock prices. Evaluation of challenges occurs at least a full business day after the audit data has been committed to [19].

6 Implementation

Election Generation Tool The hardware/OS platform design of the election generation software tool follows directly from previous systems [2, 16, 24] whereby a diskless, stand-alone computer is booted from a Linux live-CD. The Eperio election generation software is then loaded via a USB-key after being certified by external experts. Election data is generated and written back on to the USB sticks, at which point the computer is shut down (it and the live-CD are then optionally placed beyond use). The election generation software itself is separated in to graphical user-interface, and cryptographic/back-end components. The primary cryptographic operations (i.e., file-encryption based commitments) were realized using function calls to OpenSSL. We implemented the election generation tool in Python using the GTK+ GUI library, which is a standard to most GNOME Linux live-CD distributions. The graphical layout consists of a multi-panel "wizard" work flow (shown in figure 3) to incrementally guide election trustees through parameter selection and password input. Unlike the Punchscan/Scantegrity software implementation, which requires some initial system configuration, the Eperio trustee interface software is intended to be self-contained and runnable directly following a live-CD boot. A specification of the audit data is given in Appendix C.



Eperio Election Administration Utility

I. Election Setup

Summary

Review your election settings, then click Forward to confirm.

Summary

Organization name:	My organization
Election date:	4-2-2010
Number of ballots to generate:	15
Number of candidates:	5
Number of election trustees:	6
Trustee threshold:	2
Proof instances:	1

Figure 3: Election generation wizard. Trustees are guided through the complete process of selecting election parameters and setting up strong passwords prior to generating ballots and cryptographic audit data.

Bulletin Board The implementation of secure public, append-only bulletin boards is an open area of research. In its simplest form, the bulletin board consists of signed election audit data available on a public FTP server administered by the election authority, and which is closely monitored and archived/mirrored by any interested party. Mirroring of audit data can be an important peer-service provided between verifiers since the election authority may maliciously attempt to modify commitment data throughout the course of the election. It also ensures long-term archiving of the audit data, which may otherwise might be removed following the election.

Verification Script As a primary objective of Eperio, the verification script was designed to be compact and execute swiftly. A Python script externally calls OpenSSL to decrypt relevant commitment files and then

[4] We thank and acknowledge Ronald L. Rivest for suggesting this redundancy function for creating a binding commitment from a block cipher. Any errors in the analysis are our own.

performs the audits on them. We implemented proofs of the *Marked-as-Intended* and *Counted-as-Collected* properties in a compact fifty (50) lines of Python code (see Appendix D). As will be discussed in the following section, this represents the smallest implementation of a verification interface relative to other major released implementations by an order of magnitude. We tested this implementation on Ubuntu 7.10 (Python 2.5) and 9.10 (Python 2.6) as well as on Mac OSX Leopard and found that it could be executed without additional installation or configuration. Verifiers using Windows would be required to install Python and OpenSSL, or as an alternative, they could be directed to burn and boot Linux live CD. With the verification scripts on a USB key the entire audit can be completed without actually installing or configuring software on a verifier's machine.

Verification using Spreadsheets An interesting alternative to writing a custom codebase is to use a spreadsheet that can import a CSV file as a worksheet. We believe that spreadsheets can help broaden the appeal of E2E verification, given many citizens who are not accustomed to reading/writing/running code *do* use spreadsheets. Spreadsheets have grown to become one of the most familiar computer applications, capturing a diverse cross-section of users in government, enterprise, and consumer sectors. They are also widely available: OpenOffice.org, a freely-downloadable open-source office productivity suite (which includes the Calc spreadsheet application), reports over a hundred million downloads.[5]

We have developed a set of simple manual verification steps using TrueCrypt and OpenOffice Calc. As an overview of this implementation, the user manually copies and pastes the revealed encryption keys into the TrueCrypt volume password dialog to mount the encrypted volume. The unencrypted CSV file can now be loaded directly in to the spreadsheet program as a worksheet. The verifier can then complete the audit using only simple spreadsheet operations (`COPY`, `PASTE`, `SORT`, etc). A full set of directions are given in Appendix E. As an alternative to manual verification, most spreadsheet applications integrate power macro/scripting languages that could automate the audit checks.

Spreadsheets as an all-in-one election audit tool? Although many spreadsheets support basic file-encryption, cases of improper implementation (as in [43]) have led to a general distrust of spreadsheets as an encryption service. We observed that OpenOffice's file encryption leaks partial information about a file's contents by storing an unencrypted/unsalted hash of the first 1024 bytes of the compressed spreadsheet[6], making it unsuitable as an implementation of the commitment scheme proposed in section 5. Indeed it is not clear whether any of the major contemporary spreadsheet applications would be suitable for implementing file-encryption based commitments. Nevertheless, the prospect of an "all-in-one" election verification tool already installed on most voters' computers is an intriguing one.

7 Performance Comparison

Let us examine the technical requirements of election audits by comparing Eperio to several other major implementations. For space reasons we restrict our comparison to recent systems that have been deployed in binding elections, specifically Punchscan, Prêt à Voter, Helios and Scantegrity II. As outlined in section 3, Eperio uses a cut-and-choose protocol to verify the correctness of the ballot data structure. Through the use of the *linkage-list* construct for print auditing, and the symmetric-key commitment scheme outlined in section 5, the protocol allows for the commitment of entire columns, causing the number of required symmetric-key block operations to grow with the number of voters/candidates divided by the block size. The compactness of the verification process follows from it: verification involves running a file-decryption utility, followed by a sort and comparison of columns with the asserted outcome.

Both Punchscan [38] and Scantegrity [16] rely on a similar cut-and-choose protocol to audit their respective data structures, but utilize a particular commitment and print-audit scheme that requires each table element to be committed to separately resulting in a much larger dataset and number of symmetric-key operations. Unlike Eperio and Scantegrity however, Punchscan encodes ballot information in a way that is mostly invariant to the number of candidates, resulting in some savings in terms of data/number of overall block operations, although it requires additional ballot commitments as a result of its particular ballot style. Punchscan was first deployed during University of Ottawa Graduate Students Association's (GSAÉD) annual general election in 2007 [24]. This election dataset (denoted `GSAED07` in table 2) and verification source code (written in C#) are available online.[7] Scantegrity was deployed during the 2009 Takoma Park municipal election (supra. note 2). This election dataset (denoted `TAKOMA09` in table 2) and verification source code (Python and JAVA) are also available online (supra. note 3). We used the Python implementation to produce timing results in table 2.

[5] `http://stats.openoffice.org`

[6] Open Document Format for Office Applications (OpenDocument) v1.2

[7] `http://punchscan.org/gsaed/`

System	Mod Exp.	Mod Mult.	Symm. key ops	Data (B)	LOC (approx)
Eperio	–	–	$3iv(\log(2v)+2)/16$	$6iv(\log(2v)+2)+16i$	50
Scantegrity	–	–	$7.5civ$	$168civ$	1000
Punchscan	–	–	$7.5iv+9v$	$168iv+224v$	2000
Prêt-á-voter	$1.5itv$	–	itv	$288itv+192v$	1000
Helios v2.	$9cv+ct$	$9cv$	$2.5cv$	$2064cv$	1500

Table 1: Technical requirements for election verification: comparison of Eperio with other implementations for cryptographic audits involving v voters, c candidates, t trustees and i proof instances.

Prêt à Voter [17], also an E2E mechanism for optical-scan elections, uses randomized-partial checking of the correct behaviour of nodes in a decryption mixnet [30]. Each election trustee separately maintains two mix nodes in this network, and auditing the mixnet involves verifying the public-key decryption of half of the ciphertexts emanating from a given mix node. The number of public-key operations a verifier must perform therefore is the number of trustees times the number of voters. Auditing ballot printing also involves decryptions proportional to the number of trustees. Prêt à Voter differs from Eperio, Punchscan and Scantegrity in that it does not precommit to cryptographic ballot forms—they can be generated (and audited) on-demand. This allows only voted ballots to have to be decrypted by the mixnet, but the print audit still requires more ballot forms be printed than actually voted upon. Prêt à Voter was deployed in 2007 for the University of Surrey Students' Union (USSU) Sabbatical Elections. The audit dataset (USSU07) is not currently available online. The verification interface was written in JAVA.[8]

Helios v2. [2] is an E2E mechanism for remote voting that uses homomorphic tallying (i.e., tallying under encryption) to protect voter privacy. To ensure the tally is correct, a zero-knowledge proof of correctness accompanies the encryption of each candidate on each ballot in the election. Likewise, a proof of decryption accompanies the final tally. The primary computational requirement of Helios election verification comes from auditing the inputs—each ballot requires 4 modular exponentiations per candidate. The Helios verification interface was written in Python.[9] Helios v2. was deployed in 2009 in an election at Universite Catholique de Louvain. The the election dataset (UCL09) is not currently available online. Timing analysis of a reference election of the original implementation, Helios v1., is presented in [1] and is denoted in table 2 as HELv1REF.

Audit dataset size For an election involving v voters, c candidates, t trustees and i proof instances, Table 1 expresses each system in terms of the number of modular exponentiations, modular multiplications, and symmetric-key block operations required during the verification process, along with the size of the audit dataset (bytes) and contributed software lines of code (LOC) of the respective implementation. In the case of paper-ballot systems, we include an additional 0.5v ballots for the print audit. In the case of systems with pre-committed ballots, an additional 0.5v ballots are included to account for spoilage. Although the execution and data complexities of the systems are all linear in the election parameters, Eperio's performance advantage stems from its smaller constant factor.

Timing Basic timing analysis of election verification on available data is presented in Table 2. Our test platform was a 1.8GHz dual-core HP laptop running Ubuntu Linux 9.10. Eperio timings were performed on data *simulating* equivalently sized elections.

Comparing execution times of the Eperio verification script relative to three other systems (Punchscan, Scantegrity, Helios) on three election datasets shows Eperio significantly faster in time-to-verify. As a concrete example, using the Punchscan verification tool to audit the 2007 Punchscan election in [24], it took us 75 seconds to complete, while an equivalently sized election run with Eperio took us 9 seconds.

The timing result for the Scantegrity audit on the TAKOMA09 dataset includes several audits relating to the Scantegrity II ballot and voting method of the specific election that we did not replicate, suggesting that a more precise timing analysis would produce a smaller margin.

Code size: is less really more? Table 1 lists the approximate code size of published verification software showing Eperio having a significantly smaller code base than related systems. As a performance metric however, software lines of code (LOC) is limited. There is debate as to how code lines should be counted, as well as which components to even include. It also does not take into account whether more efficient implementations could be created. Setting these questions aside, are small code bases important for E2E audit software?

While ultimately smaller code bases are not a proof of relative simplicity or ease of implementation, we believe it is a signal of simplicity as well as a gesture for

[8] http://www.pretavoter.com

[9] http://www.heliosvoting.org/

Election	Scantegrity	Punchscan	Helios	Eperio
TAKOMA09	1127s	–	–	162s
GSAED07	–	75s	–	9s
HELv1REF	–	–	14400s	1s

Table 2: Time to verify election audit dataset. Eperio times are for simulated datasets of equivalently sized election.

engaging a wider audience.

8 Related Work

Throughout the 1980s and 1990s, many protocols for cryptographic voting were proposed, the majority based on either mix networks, originating with Chaum [14], or additive homomorphic encryption, originating with Benaloh and Tuinstra [9]. The first generation of mixnet protocols appeared before techniques for provably correct mixing were known, including a well-studied protocol by Fujioka *et. al* [27], while second generation protocols, originating with Sako and Killian [40], use a correctness proof.

Recently, implementation issues are given more consideration. VoteHere [36] and Votegrity [15] are early electronic systems where voters do not need to perform traditional cryptography, while VoteBox [41] allows voters to obtain an encrypted ballot from an untrusted machine using a cut-and-choose protocol due to Beneloh [8]. Prêt à Voter [17], Punchscan [38], Scratch & Vote [3], ThreeBallot [39], and Scantegrity [16] are examples of systems that use paper ballots, require no cryptography for vote capture, and offer a privacy preserving receipt to each voter. Civitas [20] is an internet system based on a protocol by Juels *et al.* [31] for high coercion elections, while Helios is designed for low coercion internet elections [1].

The cryptographic proofs in these systems use a variety of approaches, including zero-knowledge proofs, cut-and-choose protocols, and randomized partial checking [30]. Eperio is perhaps closest to Scantegrity, which also breaks election data into markable regions and uses a blackbox computation. However Eperio uses a more compact data structure, no pointers, and commits to data at a lower level of granularity than Scantegrity.

9 Future Work

This paper presented an E2E protocol in with single-party privacy assurance. Future work will be aimed at providing similar ease of verification but with multi-party construction of the Eperio table. We also focused on an E2E ballot style with randomized candidate ordering. Integration of Eperio with code-based ballots, specifically those with the subtle non-repudiation properties of the Scantegrity II ballot are of interest. Also, this paper considered *plurality* elections, however voting methods, such as *single transferable vote* (STV) and *instant run-off voting* (IRV) provide unique privacy challenges not addressed by the basic Eperio data table. Finally, and in many ways most pertinent, a formal usability study of election verification relative to other systems would be provide valuable feedback regarding the direction of this work.

10 Conclusion

The contemporary verification process for electronic voting is deficient. Independent security reviews are generally rare, time-constrained affairs subject to non-disclosure. While we advocate greater transparency for existing election technology, we also contend that end-to-end verification offers a distinct advantage: verification becomes a task of checking election *data* not software and equipment. This may appear to simply shift the problem: the prevailing methods for verifying E2E election data is with software. However it is not a simple shift. In nearly all E2E systems, the verification code is smaller than the hundreds of thousands of lines of code in a modern DRE.

With Eperio, the software is *much* smaller—four orders of magnitude smaller—and verification can even be performed manually without any custom software. By making verification more accessible to voters, we contend that Eperio is an important democracy enhancing technology.

Acknowledgment

The authors would like to thank Ron Rivest, David Chaum, and Richard Carback, Douglas W. Jones and the anonymous reviewers for helpful feedback. The authors acknowledge the partial support of this research by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] B. Adida. Helios: web-based open-audit voting. *USENIX Security Symposium 2008.*

[2] B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater. Electing a university president using open-audit voting: analysis of real-world use of Helios. *EVT 2009*.

[3] B. Adida and R. L. Rivest. Scratch & Vote: self-contained paper-based cryptographic voting. *WPES 2006*.

[4] A. W. Appel, M. Ginsburg, H. Hursti, B. W. Kernighan, C. D. Richards, G. Tan, and P. Venetis. The New Jersey voting-machine lawsuit and the AVC Advantage DRE voting machine. *EVT 2009*.

[5] A. Aviv, P. Cerny, S. Clark, E. Cronin, G. Shah, M. Sherr, and M. Blaze. Security evaluation of ES&S voting machines and election management system. *EVT 2008*.

[6] M. Bellare, A. Desai, E. Jokiph, and P. Rogaway. A concrete security treatment of symmetric encryption: an analysis of the DES mode of operation. *FOCS 1997*.

[7] J. Benaloh. Verifiable secret-ballot elections. PhD thesis, Yale University, 1987.

[8] J. Benaloh. Simple verifiable elections. *EVT 2006*.

[9] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. *STOC 1994*.

[10] M. Blaze, A. Cordero, S. Engle, C. Karlof, N. Sastry, M. Sherr, T. Stegers, and K.-P. Yee. Source code review of the Sequoia voting system. *State of California's Top to Bottom Review*, 2007.

[11] K. Butler, W. Enck, H. Hursti, S. McLaughlin, P. Traynor, and P. McDaniel. Systemic issues in the Hart InterCivic and Premier voting systems: reflections on project everest. *EVT 2008*.

[12] J. A. Calandrino, A. J. Feldman, J. A. Halderman, D. Wagner, H. Yu, and W. P. Zeller. Source code review of the Diebold voting system. *State of California's Top to Bottom Review*, 2007.

[13] R. Carback, J. Clark, A. Essex. et al. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy. *USENIX Security*, 2010.

[14] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[15] D. Chaum. Secret-ballot receipts: true voter-verifiable elections. *IEEE Security and Privacy*, 2(1), 2004.

[16] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. *EVT 2008*.

[17] D. Chaum, P. Y. Ryan, and S. A. Schneider. A practical, voter-verifiable, election scheme. *ESORICS 2005*.

[18] J. Clark, A. Essex, and C. Adams. Secure and observable auditing of electronic voting systems using stock indices. *IEEE CCECE 2007*.

[19] J. Clark and U. Hengartner. On the use of financial data as a random beacon. *EVT/WOTE 2010*.

[20] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: toward a secure voting system. *IEEE Symposium on Security and Privacy 2008*.

[21] A. Cordero, D. Wagner, and D. Dill. The role of dice in election audits. *WOTE 2006*.

[22] Election Data Services Inc. Voting equipment summary by type as of 11/07/2006. 2006.

[23] A. Essex, J. Clark, and C. Adams. Aperio: High Integrity Elections for Developing Countries *Towards Trustworthy Elections*. Lecture Notes in Computer Science, vol. 6000, 2010.

[24] A. Essex, J. Clark, R. T. Carback, and S. Popoveniuc. Punchscan in practice: an E2E election case study. *WOTE 2007*.

[25] Federal Constitutional Court of Germany. Judgement 2 BvC 3/07, 2 BvC 4/07, Verfahren über die Wahlprüfungsbeschwerden, 2009.

[26] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. *CRYPTO '86*.

[27] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *ASIACRYPT '92*.

[28] E. Hubbers, B. Jacobs, and W. Pieters. RIES: Internet voting in action. *COMPSAC 2005*.

[29] S. Inguva, E. Rescorla, H. Shacham, , and D. S. Wallach. Source code review of the Hart InterCivic voting system. *State of California's Top to Bottom Review*, 2007.

[30] M. Jacobsson, A. Juels, and R. L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. *USENIX Security Symposium 2002*.

[31] A. Juels, D. Catalano, and M. Jacobsson. Coercion-resistant electronic elections. *WPES 2005*.

[32] J. Katz. Universally composable multi-party computation using tamper-proof hardware. *EUROCRYPT '07*

[33] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting system. *IEEE Symposium on Security and Privacy 2004*.

[34] T. Moran and M. Naor. Split-ballot voting: Everlasting privacy with distributed trust. *CCS 2007*.

[35] T. Moran and G. Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. *EUROCRYPT '08*.

[36] C. A. Neff. Practical high certainty intent verification for encrypted votes. Technical report, VoteHere, 2004.

[37] T. P. Pedersen. A threshold cryptosystem without a trusted party. *EUROCRYPT '91*.

[38] S. Popoveniuc and B. Hosp. An introduction to Punchscan. *WOTE 2006*.

[39] R. L. Rivest and W. D. Smith. Three voting protocols: Threeballot, VAV, and Twin. *EVT 2007*.

[40] K. Sako and J. Kilian. Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. *EUROCRYPT '95*.

[41] D. R. Sandler, K. Derr, and D. S. Wallach. VoteBox: a tamper-evident, verifiable electronic voting system. *USENIX Security Symposium 2008*.

[42] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for DOS resistance. *CCS 2004*.

[43] H. Wu. The misuse of RC4 in Microsoft Word and Excel. *Cryptography ePrint Archive*, 2005/007, 2005.

[44] United States Election Assistance Commission (EAC). The 2008 voluntary voting system guidelines (v1.1), 2009.

A Security Proof

In this section, we prove that Eperio is *complete*, *sound*, and *computationally secret*. Given a transcript of the entire protocol, the asserted tally can be either accepted or rejected. If the asserted tally is correct, the decision will always be to accept (completeness). If the asserted tally is not correct, the decision will be to reject with a high probability (soundness). Finally, the outputs do not provide any information that can be used by a computationally bounded adversary to determine any non-negligible information about which candidate was voted for by any voter (computational secrecy).

To maintain consistency with the literature, we will denote the trustees as the prover ($\mathcal{P}$), who we initially consider unbounded in computation power. The three types of verifiers ($\mathcal{V}$) are considered to be bounded to probabilistic polynomial time (PPT). These bounds are used to demonstrate unconditional integrity, and are latter reversed for a brief consideration of everlasting privacy. Either entity may employ a malicious strategy and we denote this with a prime ($\mathcal{P}', \mathcal{V}'$).

The Eperio verification protocol is challenge-response. The print audit uses an *interactive challenge* where verifiers choose which ballots at random to check that the correspondence between ballot positions and candidates is consistent with the committed $\mathbf{E}$ table. The receipt check also uses an *interactive challenge* where a random fraction of voters will choose to check that the correspondence between ballot positions and marks is consistent with the published marks $\mathbf{E}_{(:,2,:)}$. Finally the tally audit uses a *public coin challenge* where the prover is asked to reveal either $\mathbf{E}_{(:,1,x)}$ or $\mathbf{E}_{(:,3,x)}$ for a given x.

Let Ω represent the description of an election; a mapping between each ballot and the candidate who was voted. Consider the pair $\langle \mathbf{E}, \mathbf{R} \rangle$, an Eperio table and the random factors used in committing to it. These two items are sufficient for generating all the other data structures needed in the verification protocol. Denote $\mathcal{E}$ as the set of all $\langle \mathbf{E}, \mathbf{R} \rangle$ pairs that would correctly encode Ω, and $\bar{\mathcal{E}}$ as the set of all other pairs.

A.1 Completeness

An abbreviated transcript of the verification protocol follows:

$\mathcal{P} \rightarrow \mathcal{V}$:	$\mathbf{C}$	Commitment
$\mathcal{P} \leftarrow \mathcal{V}$:	$\mathbf{M}$	Challenge
$\mathcal{P} \rightarrow \mathcal{V}$:	ρ, $\mathbf{L}$, $\mathbf{E}_{(:,2,:)}$, τ	Response
$\mathcal{P} \leftarrow \texttt{Coin}$:	$\mathbf{Z}$	Challenge
$\mathcal{P} \rightarrow \mathcal{V}$:	$\mathbf{R}_z$	Reveal

Recall that τ represents the asserted tally and let ρ be the asserted receipts. Assume $\mathbf{E} \in \mathcal{E}$ for Ω. If $\mathcal{P}$ and $\mathcal{V}$ behave honestly and follow the protocol outlined in the previous section, then τ, ρ and $\mathbf{L}$ will also be consistent with Ω and the three verifications performed by $\mathcal{V}$ will succeed with certainty.

It may be worthwhile to note that $\mathcal{V}$'s acceptance is conditional on many things beyond τ. If $\mathcal{P}$ were to commit to wrong values and reveal marks in the wrong position, but still report the correct τ, the proof will be rejected. We do not classify this as a true negative error as we are determining whether the proof is acceptable.

A.2 Soundness

We now consider whether $\mathcal{V}$ will ever accept a proof when $\mathbf{E} \in \bar{\mathcal{E}}$. If the probability of rejecting a false proof is much greater than 1/2, we say Eperio is sound. The soundness of Eperio relies on two assumptions:

1. The function $\mathsf{Commit}(m, r)$ is perfectly binding. That is, for any m_1 such that $\mathsf{Commit}(m_1, r_1) = c_1$, there does not exist any r_2 and $m_2 \neq m_1$ such that $\mathsf{Reveal}(c_1, r_2) = m_2$.

2. The function $\mathsf{PublicCoin}()$ is perfectly unpredictable before invocation.

Let $\mathrm{tr}_{(\mathcal{P}'(\mathbf{E},\Omega),\mathcal{V}(\mathbf{M},\mathbf{Z}))}(\alpha)$ denote a transcript of a proof given by a malicious prover with private information $\mathbf{E}_{(i,j,k)}$ and Ω to an honest verifier with challenges $\mathbf{M}$ and $\mathbf{Z}$. The proof concerns some public information α. We shorten the notation to $\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\alpha)$. Given a transcript, $\mathcal{V}$ will decide in PPT-time to either $\texttt{ACCEPT}$ or $\texttt{REJECT}$ the transcript as a valid proof.

The logic of $\mathcal{V}$'s strategy in terms of the three types of verification is as follows,

$$\begin{aligned} \texttt{ACCEPT}_{\mathcal{V}}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C}, \mathbf{L}, \rho, \tau, \mathbf{E}_{(:,2,:)}, \mathbf{R}_z)) &\leftarrow \\ \texttt{ACCEPT}_{\mathcal{V}_1}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\rho)) &\cap \\ \texttt{ACCEPT}_{\mathcal{V}_2}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C}, \mathbf{L}, \mathbf{R}_z)) &\cap \\ \texttt{ACCEPT}_{\mathcal{V}_3}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C}, \rho, \tau, \mathbf{E}_{(:,2,:)}, \mathbf{R}_z)). & \qquad (1) \end{aligned}$$

We prove the soundness of the three right-hand decisions independently and then consider the implication of their conjunction. Since the proof protects against very strong adversaries, we also remark on what a real world adversary may want to achieve as it relates to each proof.

Receipt Check First consider the verification that the asserted receipts, ρ, match the physical receipts retained by the voters: $\texttt{ACCEPT}_{\mathcal{V}_1}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\rho))$. This check does not prove anything about the committed values directly, but is simply designed to decide whether ρ alone is reputable. If it is, this fact can be used to establish other facts more directly associated with the asserted tally τ.

A malicious prover has a binary choice at this stage: to assert a correct value ρ or an incorrect ρ'. If ρ' is asserted, the transcript's acceptablity decreases in proportion to the number of voters verifying their receipts. Let b'_r be the number of modified ballot receipts in ρ', and $0 \leq p_1 \leq 1$ represent the fraction of voters who conduct a receipt check. Further assume the distribution of b'_r and the ballots associated with p_1 are independent. In this case, the probability of rejecting an incorrect transcript is tightly approximated by:

$$\mathbf{Pr}[\mathtt{REJECT}_{\mathcal{P}',\mathcal{V}_1}] \quad = \quad (1-(1-p_1)^{b'_r}). \qquad (2)$$

In terms of real-world adversaries, this verification step is very important. A malicious set of trustees could use such an attack to explicitly move votes from one candidate to another in ρ. This result can be generalized to any voting system where voters check the inclusion of their obfuscated votes. Increasing certainty in the tally proof can often be made exponential in some chosen parameter (as we will show is the case with Eperio), but this certainty will at some point overtake the certainty that the inputs are correct, which are fixed by p_1 and invariant to the security parameter of the system. Since the soundness of the overall system rests in the weakest link—increasing the certainty of the tally does not necessarily add security overall! Further attention should be paid to methods for increasing partition in the receipt check, or introducing mechanisms to capture all the receipts. The only system we are aware of to address this is VoteBox, which includes their earlier AUDTITORUM protocol that could capture all receipt information [41].

Print Audit Next consider $\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C}, \mathbf{L}, \mathbf{R}_z)$ which is used to determine if the linkage list, $\mathbf{L}$, implies that the ballots were printed in a way that is consistent with the binary relation $\mathrm{R}(\mathbf{E}_{(i,1,k)}, \mathbf{E}_{(i,3,k)})$ for all i and k. These values are committed to in $\mathbf{C}$ and the partial relation $\mathbf{E}_{(i,2z+1,k)}$ is opened by $\mathbf{R}_z$.

If a malicious prover misprints b'_p ballots, and $0 \leq p_2 \leq 1$ is the fraction of ballots that are print audited, the probability of a modified ballot being selected for audit is $1-(1-p_2)^{b'_p}$. There are different ways to misprint a ballot, however any surjective modifications are trivially detectable. Consider a misprinted ballot to be one that bijectively swaps the order of some candidates.

$\mathcal{P}'$ could escape detection by decommiting to the misprinted value, however we assume this is impossible by soundness assumption 1. Instead, $\mathcal{P}'$ must assert which i in $\mathbf{E}_{(i,j,k)}$ matches the printed ballot. For a given k, $\mathcal{P}'$ could choose i such that $\mathbf{E}_{(i,1,k)}$ is correct, and will escape detection if the coin flip is 0 but be detected if $z = 1$. Alternatively, the prover could choose the row such that $\mathbf{E}_{(i,3,k)}$ is correct, and will escape detection if $z = 1$ but be detected if $z = 0$. Given the second soundness assumption, $\mathcal{P}'$ cannot predict z to a probability better than $1/2$ for each instance $1 \leq k \leq x$. In this case,

$$\mathbf{Pr}[\mathtt{REJECT}_{\mathcal{P}',\mathcal{V}_2}] \quad = \quad (1-(1-p_2)^{b'_p})(1-\frac{1}{2^x})(3)$$

In our estimate, a real-world adversary is unlikely to gain from misprinting ballots. The attack would begin with some prior knowledge of the expected distribution of votes—*e.g.*, pre-election polls showing another candidate leading or an expected bias for a given precinct—and would involve mapping the leading candidate's votes to the adversary's preferred candidate and the least preferred candidate to the main competitor. With $x = 20$ and 5% of ballots being print audited, the election authority can be 99% certain that no more than 89 ballots were misprinted and 50% certain that no more than 13 were. With 20% checked, no more than 20 ballots were misprinted with 99% certainty and no more than 3 with 50% certainty. While a 99% detection rate tolerates a fairly high number of misprinted ballots, this attack is marginalized by the observation that a successful execution will only probabilistically gain votes for the attacker. For example, if working from poll data showing a 51-49% split between two candidates, misprinting 89 ballots will on average only net the attacker 2 votes.

Tally Audit Finally consider the verification that the now validated receipt list, ρ, implies that the asserted tally, τ, is correct. The partial relation $\mathbf{E}_{(i,2z+1,k)}$ is opened by $\mathbf{R}_{(z+1,j)}$ and can be considered along side $\mathbf{E}_{(:,2,:)})$. If $z = 0$, this pair of values should match ρ. If $z = 1$, the pair should match τ. Thus our consideration is $\mathtt{ACCEPT}_{\mathcal{V}_3}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C}_{(i,j)}, \rho, \tau, \mathbf{E}_{(:,2,:)}, \mathbf{R}))$.

If $\mathbf{E}_{(:,2,:)}$ is correct, then the tally produced by any $\mathbf{E}_{(i,3,k)}$ is also correct. However if $\mathbf{E}_{(:,2,:)}$ is not correct, it is possible for $\mathcal{P}'$ to assert a tally τ' that is incorrect and acceptable to $\mathcal{V}$. For a given k, $\mathcal{P}'$ could choose $\mathbf{E}_{(:,2,:)}$ to match τ, and will escape detection if the coin flip is 0 but be detected if $z = 1$. Alternatively, $\mathcal{P}'$ could choose $\mathbf{E}_{(:,2,:)}$ to match τ', and will escape detection if $z = 1$ but be detected if $z = 0$. Given the second soundness assumption, $\mathcal{P}'$ cannot predict z to a probability better than $1/2$ for each instance $1 \leq k \leq x$. In this case,

$$\mathbf{Pr}[\mathtt{REJECT}_{\mathcal{P}',\mathcal{V}_3}] \quad = \quad (1-\frac{1}{2^x}). \qquad (4)$$

In terms of real-world adversaries, this verification step is very important. Like in the receipt check, a malicious set of trustees could use such an attack to explicitly move votes from one candidate to another in τ.

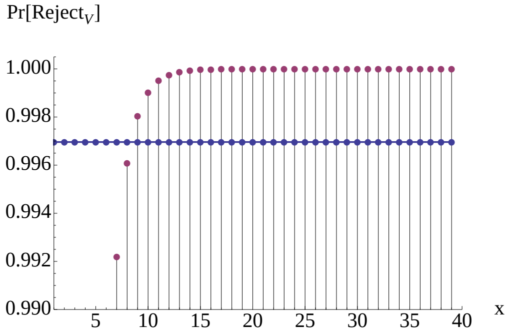


Figure 4: **The Weakest Link**. The probability that the transcript of an E2E-enabled election will be discovered to be unsound, given ballots were manipulated in an attempt reverse the outcome, is the minimum of two data series: a receipt audit (horizontal line) and tally audit (asymptote). The tally audit converges on 100% as the number of proof instances, x, is increased. The receipt audit however is a function of the number of voters checking their receipts relative to the incidence of fraud. For example, if only a random 5% of voters checked their receipts in a tight race, such as the 2008 Minnesota senate race in which the victory margin was 225 votes, the probability of detecting the fraud would be 99.7%, meaning at $x = 9$, committing to additional instances would not have increased overall effective soundness.

Overall Effective Soundness We assume $\mathcal{P}'$ will adopt a proof strategy that minimizes the soundness of the proof. Thus given a transcript with an incorrect public input, the probability of it being rejected by V is,

$$\mathbf{Pr}[\texttt{REJECT}_{\mathcal{P}',\mathcal{V}}] = \min[(1-(1-p_1)^{b'_r}), (1-(1-p_2)^{b'_p})(1-\frac{1}{2^x}), (1-\frac{1}{2^x})]. \tag{5}$$

By estimating p_1 and b'_r, we can use this equation to determine a suitable x for our implementation. For example, if the close 2008 Minnesota senate race was run with Eperio, the attacker would need to modify at least 113 marks to overthrow the 225 victory margin.[10] Setting aside misprinted ballots, if say 5% of voters would check their receipts, an incorrect ρ will be detected with 99.7% certainty. Thus x must be increased until the probability of detecting an incorrect τ is greater than this probability, which occurs at $x = 9$ instances. This is shown in Figure 4.

A.3 Computational Secrecy

We now prove that Eperio achieves computational secrecy. This relies on the following assumptions:

1. The maximum number of colluding trustees is t.
2. At least one trustee submits to DKG an f_i drawn with uniform randomness from $\mathbb{Z}_l^t$.
3. All outputs are computed with a blackbox.
4. Any polynomial-sized output from $\mathsf{PRNG}(\kappa)$ provides no non-negligible advantage to a PPT-bounded adversary in guessing either κ, the next bit in the output, or any unobserved previous bit.
5. The function $\mathsf{Commit}(m, r)$ is semantically secure and computationally hiding. That is, given either $c_1 = \mathsf{Commit}(m_1, r_1)$ or $c_2 = \mathsf{Commit}(m_2, r_2)$ for any chosen m_1 and m_2, a PPT-bounded adversary should have no non-negligible advantage in guessing which message was committed to.

Our approach will be to demonstrate that a transcript provides a malicious verifier, $\mathcal{V}'$, with no non-negligible information about which candidate was selected for any ballot in the election. Let RecoverSel be a PPT-algorithm that returns 1 if it successfully recovers such information. Note that given a correct list of receipts and tally, some information may be determinable. For example, the adversary may be able to determine if a voter abstained or voted randomly (if the adversary instructs voters to, say, always mark the first position). If everyone votes for the same candidate then RecoverSel will trivially succeed. For reasons such as these, we define a baseline success, ψ, which depends on the security of the voting interface (out-of-scope in our paper) and only theorize that Eperio is computationally secret relative to ψ.

$$\mathbf{Pr}[\mathsf{RecoverSel}(\texttt{View}_{\mathcal{V}'}(\rho,\tau)) = 1] \triangleq \psi \tag{6}$$

We begin by proving,

$$|\mathbf{Pr}[\mathsf{RecoverSel}(\texttt{View}_{\mathcal{V}'}(\rho,\tau,\mathbf{C})) = 1] - \mathbf{Pr}[\mathsf{RecoverSel}(\texttt{View}_{\mathcal{V}'}(\rho,\tau)) = 1]| \leq \epsilon_1 + \epsilon_2. \tag{7}$$

Assume there exists a polynomial time algorithm $\mathsf{RecoverSel}'()$ that can distinguish these two views with greater than negligible advantage. Such an algorithm could take as input any set of commitments $\mathbf{C}$ and return non-negligible information about the messages committed to. Consider two cases: (i) the random factors in the commitments are indistinguishable from uniform randomness and (ii) the random factors are distinguishable by a bounded $\mathcal{V}'$.

Case (i) is in violation of assumption 5. Given $\mathsf{RecoverSel}'()$ runs in polynomial time, it cannot achieve more information than ϵ. Let this information be ϵ_1.

[10] http://electionresults.sos.state.mn.us/

If case (ii) holds, this can be further broken into two cases: (ii-a) the output of the $\mathsf{PRNG}()$ function follows some distinguishable distribution and (ii-b) the seed to the $\mathsf{PRNG}()$ function follows some distinguishable distribution. Case (ii-a) is in violation of assumption 4 which implies that if $\mathsf{RecoverSel}'()$ runs in polynomial time, it cannot achieve more information than ϵ. Let this information be ϵ_2. Given that $\mathsf{DKG}()$ is information theoretic, case (ii-b) can only hold if all of the trustee shares follow a known distribution, a violation of case (ii-b).

Now consider the following,

$$\begin{aligned} &|\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C}, \mathbf{L}, \mathbf{E}_{(:,2,:)}, \mathbf{R}_z) = 1] \\ &-\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C})) = 1]| \leq \epsilon_3 + \epsilon_4. \end{aligned} \tag{8}$$

Again assume $\mathsf{RecoverSel}'()$ can distinguish these two views with greater than ϵ advantage. Such an algorithm could take as input several input-output pairs of a permutation (*e.g.,* from $\mathbf{L}$ or knowledge of one's own receipt) and output non-negligible information about how the permutation maps other pairs not in $\mathbf{L}$. Given that the $\mathsf{Permute}()$ and $\mathsf{PermuteBlock}()$ functions are implemented with perfect shuffles, $\mathsf{RecoverSel}'()$ cannot be polynomial in time unless if the $\mathsf{PRNG}()$, its seed κ, or all the trustee shares follow a distinguishable distribution. As shown by cases (ii-a) and (ii-b) above, this cannot be. Similarly, the information leaked by $\mathbf{R}$ is addressed by case (ii) above.

This demonstrates that the Eperio verification protocol is computationally secret in a single instance. We complete the proof by considering the composition of multiple instances of independently shuffled tables. Give a single instance, k, is secure, consider as an inductive step the addition of $k+1$,

$$\begin{aligned} &|\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C}_{(:,k)}, \mathbf{C}_{(:,k+1)}, \mathbf{L}_{(:,k)}, \\ &\mathbf{L}_{(:,k+1)}, \mathbf{E}_{(:,2,k)}, \mathbf{E}_{(:,2,k+1)}, \mathbf{R}_{z,k}, \mathbf{R}_{z,k+1}) = 1] - \\ &\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C}_{(:,k)}, \mathbf{L}_{(:,k)}, \\ &\mathbf{E}_{(:,2,k)}, \mathbf{R}_{(z,k)}) = 1]| \leq \epsilon_5 \end{aligned} \tag{9}$$

Given the output of instance k, we show how a deterministic polynomial time simulator could generate a second transcript, $k+1$, that is indistinguishable from a real transcript. Intuitively, there is only one set of marks $\mathbf{E}_{(:,2,:)}$ that will satisfy the binary relation $\mathrm{B}(\mathbf{E}_{(i,1,k)}, \rho)$. However there are many permutations of $\mathbf{E}_{(i,3,k)}$ that satisfy the binary relation $\mathrm{B}(\mathbf{E}_{(:,2,:)}, \tau)$. The output of Eperio only proves that τ is correct, it does not give any information about which receipts the votes for each candidate in τ came from. Any mapping of the results in τ is plausible, and the simulator, $\mathcal{S}_{\mathcal{V}'}$, only needs to generate a transcript of one of them.

$\mathcal{S}_{\mathcal{V}'}$ will generate $\mathbf{E}$ such that $\mathbf{E}_{(i,1,k+1)}$ is randomly shuffled along i for $k+1$. Next, $\mathcal{S}_{\mathcal{V}'}$ will fill in $\mathbf{E}_{(:,2,k+1)}$ according to ρ. Then, $\mathcal{S}_{\mathcal{V}'}$ will fill in $\mathbf{E}_{(i,3,k+1)}$ for each i in $\mathbf{L}_{(i,j)}$ with the appropriate candidate. Finally, $\mathcal{S}_{\mathcal{V}'}$ will randomly assign candidates to the rest of $\mathbf{E}_{(i,3,k+1)}$ such that τ is satisfied. The simulator will commit to this, generating $\mathbf{C}_{(i,j)}$.

The verifier, $\mathcal{V}'$, will observe random bits $\mathbf{Z}_i$ and challenge $\mathcal{S}_{\mathcal{V}'}$ to reveal $\mathbf{R}_{(z+1,j)}$. Since we have gone through the difficulty of validating all of the public parameters, $\mathcal{S}_{\mathcal{V}'}$'s output will be acceptable at this stage with probability 1. It does not need to rewind the protocol.

Consider a bound on the size of $\mathcal{E}$ for a given Ω. Recall that b is the number of ballots, s the number of selections, and x the number of instances. The construction of $\mathbf{E}_{(i,j,k)}$ requires two permutations, $\Pi_S = s!^b$ and $\Pi_E = (bs)!^x$, while $\mathbf{R}_{(i,j)}$ requires two l-bit random factors for each instance, yielding 2^{2xl} possible values. The number of random bits for any $\mathcal{E}$ is thus on the order of $2xl + b \cdot \mathcal{O}(s \log s) + x \cdot \mathcal{O}(bs \log bs)$. For large enough b and s, this is much greater than the l bits of true randomness in κ. Thus, only a small subset of $\mathcal{E}$ are possible outputs of the described protocol. Due to the properties of the PRNG, a bounded simulation strategy cannot distinguish this subset from a randomly generated $\mathcal{E}$ however a unbounded $\mathcal{V}'$ can. This is the intuition behind Eperio offering only *computational* secrecy.

A.4 Everlasting Privacy

We now briefly sketch how to modify Eperio to achieve everlasting privacy. It requires three simple modifications. First, the commitment function is replaced with one that is perfectly hiding and computationally binding. This affects the soundness proof as committed values can be opened, by an unbounded adversary, to any value. Therefore the soundness of the argument is only as hard as the computational bindingness of the commitments. This also changes assumption 5 of the secrecy proof. Second, trustees generate all of the randomness needed to do the permutations and random factors. From a practical standpoint, this significantly increases the complexity of the protocol as a DKG is needed to threshold share every piece of randomness. Finally, given the direct source of randomness, the PRNG can be eliminated. This removes assumption 4 of the secrecy proof.

The first 3 secrecy assumptions still hold, demonstrating that everlasting privacy is achievable but only within a narrow view of the protocol. For example, if at least $t+1$ trustees reveal their private inputs, privacy can still be broken. This holds for any distributed voting system in the literature.

B Proof of Theorem 3

Proof of Theorem 3. To address the hidingness, consider an attack game where an adversary A provides equal-length messages M_1 and M_2 to an oracle O. O generates a random K and IV, flips a coin z, and returns $C_z = E(K, \mathrm{IV}, M_z)$. A must guess if C_z is a commitment to M_1 or M_2 (or that both are possible under a different K). We say Commit is *computationally hiding* if the adversary's advantage over a random guess is negligible in k. Assume an algorithm $z \leftarrow \mathcal{A}_h(m_1, m_2, c_z, \mathrm{IV})$ exists that can guess z correctly with non-negligible advantage. It is straightforward to see how this algorithm would provide an advantage in a chosen plaintext attack game: the adversary would submit the CPA challenge values to $\mathcal{A}_h$ and respond with z. Since E is CPA-secure under Theorem 1, $\mathcal{A}_h$ can be assumed to be not computable by a PPT-bounded adversary. Thus the commitment scheme is computationally hiding.

We say Commit is *statistically binding* for a correctly formed M' if,

$$p \triangleq \mathbf{Pr}[\exists M' \text{ s.t. } (C, \mathrm{IV}) = \mathsf{Commit}(M', K')| \\ (C, \mathrm{IV}) = \mathsf{Commit}(M, K) \ \&\ M \neq M'] \leq \mathsf{negl}(k). \quad (10)$$

Recall redundancy $f(M) = M \| M$; that is, $\mathsf{Commit}(M, K) = E(K, \mathrm{IV}, M\|M\|M)$.[11] If Conjecture 2 is true, the probability p that M' and K', as defined in Equation 10, exist given some distinct M and K can be determined. A random function has the following form, where x is the size of domain of the commitment function and y, the co-domain.[12]

$$p = 1 - (1 - \frac{1}{y})^{x-1} \quad (11)$$

Recall the input to $\mathcal{E}$ is a m-bit message block and a k-bit key. For a message M with L blocks ($|M| = L \cdot m$), the domain of Commit is $x = 2^{k+Lm}$. The 2 repetitions of M are fixed by choice of the domain, and the initialization vector is a constant. The co-domain is $y = 2^{3Lm}$, with the factor of 3 due to the redundancy function f. We are interested in an upper-bound to equation 11 with these values of x and y. As we are not trying to prove a general case, recall our assumption that $m = k$, and note that L is a positive integer such that p is maximized when L is minimized. The upper bound at $L = 1$ is given as Equation 12.

$$\begin{aligned} p &\leq 1 - (1 - \frac{1}{2^{3k}})^{2^{2k}-1} && (12) \\ &\approx 1 - \exp(-\frac{2^{2k}}{2^{3k}}) && (13) \\ &= 1 - \exp(-2^{-k}) && (14) \\ &\leq \mathsf{negl}(k) && (15) \end{aligned}$$

Equation 13 follows from the approximations that $(1 - 1/y)^x \approx \exp(-x/y)$ and $(1 - 1/y)^{-1} \approx 1$ for large y. ■

We have demonstrated a very specific statistically hiding commitment function that can be constructed from a block cipher, assuming conjecture 2 holds.

[11] We would like to thank and acknowledge Ronald L. Rivest for suggesting this approach in creating a binding commitment from a block cipher. Any errors in the analysis are our own.

[12] *Intuition:* It may help to conceptualize this as having x balls and y bins. We throw the first ball into a random bin, and ask for the probability that at least one of the remaining balls will be thrown into the same bin.

C Eperio Audit Data Specification

All (unencrypted and decrypted) election data files are formatted as UTF-8 encoded Comma Separated Values (CSV). Columns are separated by the delimiter ',' (`0x2C`) and rows are separated by a carriage return (`0x0A`).

File Naming and Formatting Convention

For an election of b ballots, c candidates, I proof instances, of which p were print audited, the following files will be generated and posted to the bulletin board through the course of the election verification timeline

`$electionDir$/AssertedReceipts.csv`: is a $bc \times 2$ list containing an assertion of all receipt information created during an election. Each element in $(:, 1)$ is unique election references: $s{-}p$ where $1 \leq s \leq b$ is a ballot serial number and $0 \leq p \leq c - 1$ a mark position. Elements in column (:,2) represent the corresponding mark state registered during the election, either `0`, `1`, or `-1` indicating the corresponding unique reference was either unmarked, marked or print audited respectively.

`$electionDir$/AssertedTally.csv`: is a $bc \times 2$ list containing an assertion of aggregate voting intent created during an election. Column $(:, 1)$ is a list of b repetitions of c candidate names. Elements in column $(:, 2)$ represent the corresponding mark state.

`$electionDir$/LinkageList.csv`: is a $p \times i + 2$ list containing an assertion of the association between receipt $(:, 1)$ and candidate $(:, 2)$ for all print audited ballots, as well as an assertion of the row number the entry in each of the I instances of shuffled receipt (`U`) and tally (`S`) lists.

`$electionDir$/ChallengeKeys.csv`: is a $i \times 2$ list containing a respond to the audit challenge. Elements at $(i, 1)$ represent the relative file path to the file that was challenged: the i-th shuffled instance of the receipt list `$electionDir$/`i`/U`i or tally list `$electionDir$/`i`/S`i respectively. Elements at (i,2) represent the corresponding revealed decryption key to the file (128-bits, `BASE64` encoded).

For each instance $1 \leq i \leq I$:

`$electionDir$/`i`/S`i`.csv`: is a $bc \times 3$ list representing a shuffled instance of column (:,1) of `AssertedTally.csv` repeated three times (as per the commitment scheme described in section 5) in columns $(:, 1)$, $(:, 2)$ and $(:, 3)$ respectively.

`$electionDir$/`i`/U`i`.csv`: is a $bc \times 3$ list representing a shuffled instance of column $(:, 1)$ of `AssertedReceipts.csv` repeated three times as above.

`$electionDir$/Marks/M`i`.csv`: is a $bc \times 1$ list representing a shuffled instance of column $(:, 2$ of `AssertedReceipts.csv` and `AssertedTally.csv`.

File Encryption Conventions

128-bit file encryption keys are derived by the semi-trusted trustee workstation and `BASE64` encoded for convenience of transmission over the internet. The two file encryption options (OpenSSL and TrueCrypt) use their own password-based key derivation functions. OpenSSL and TrueCrypt encrypted files are denoted by the `.enc` and `.tcr` extensions, and use AES-128-CBC and AES-128-XTS encryption modes respectively. There is no inherent way to bypass the key-derivation process of these applications to directly apply a binary key. As such the `BASE64` encoded keys are supplied as passwords and each program (OpenSSL, TrueCrypt, etc) will derive its own (binary) encryption key using its own particular password based key derivation function. In OpenSSL the workstation applies the `-a` command option to `BASE64` encode the encrypted output for safe internet transport.

D Source code for Eperio verification interface (Python)

```python
import csv, os, string, time, sys
def openCSV(fileName):
        data = []
        fileContents = open(fileName, 'r')
        for i in fileContents:
                data.append(i.strip().split(","))
        return data
def verifyFailed(errorMsg):
        print errorMsg
        sys.exit()
def verifyCommitment(instance):
        os.system("openssl enc -d -aes-128-cbc -a -in " + instance[0] + ".enc -out " + instance[0] + ".csv -pass pass:" + instance[1])
        data = openCSV(instance[0].strip('"') + ".csv")
        col = []
        for i in data:
                if i[0] != i[1] or i[0] != i[2]:
                        verifyFailed("Invalid Commitment")
        return [i[0] for i in data]
def getMarks(instance):
        marks = openCSV("./Marks/M"+ instance[0][2:4] + ".csv")
        return [i[0] for i in marks]

assReceipts = sorted(openCSV('./AssertedReceipts.csv'))
assTally = sorted(openCSV('./AssertedTally.csv'))
pwds = openCSV('./ChallengeKeys.csv')
linkageList = openCSV('./LinkageList.csv')
for i, instance in enumerate(pwds):
        merged=[]
        data = verifyCommitment(instance)
        marks = getMarks(instance)
        for index in range(len(data)):
                merged.append([data[index], marks[index]])
        if instance[0][5] == "U" :
                for ballot, linkage in enumerate(linkageList):
                        if linkageList[ballot][0] != merged[int(linkageList[ballot][i+2])-1][0] or merged[int(linkageList[ballot][i+2])-1][1] != "-1" :
                                verifyFailed("Linkage list FAIL: Receipt instance #%d does not does not match linkage list" % (i+1))
                if assReceipts == sorted(merged):
                        print "Instance %d verified" % (i+1)
                else:
                        verifyFailed("Verification FAIL: Receipt instance #%d does not match asserted receipt list!" % (i+1))
        elif instance[0][5] == "S":
                for ballot, linkage in enumerate(linkageList):
                        if linkageList[ballot][1] != merged[int(linkageList[ballot][i+2])-1][0] or merged[int(linkageList[ballot][i+2])-1][1] != "-1" :
                                verifyFailed("Linkage list FAIL: Tally instance #%d do\texttt{BASE64} encodedes not does not match linkage list" % (i+1))
                if assTally == sorted(merged):
                        print "Instance %d verified" % (i+1)
                else:
                        verifyFailed("Verification FAIL: Tally instance #%s does not match asserted tally list!" % (i+1))
        else:
                verifyFailed("Instance filename is incorrectly formatted")
```

E User Instructions for Spreadsheet Election Verification

The following instructions were tested in OpenOffice.org Calc 3.0 and TrueCrypt 6.0a. The user is assumed to have downloaded all relevant data files to directory `/$electionDir$/`. Let i represent the instance number the verifier wishes to check.

Part I: Verify Commitments
In Calc:

- Open `$electionDir$/ChallengeKeys.csv`
- Let `$filePath$` represent the contents of cell Ai

In TrueCrypt:

- Select a free drive letter (without loss of generality assume `Z:`)
- Click `Select File` → Select `$electionDir$/$filePath$.trc` → Open

In Calc:

- Click on cell Bi → `CTRL` + `C`

In TrueCrypt:

- Click Mount → `CTRL+V` → OK

In Calc:

- Open `Z:/$filePath$.csv`,
- In cell `D1` type formula `=A1=B1`,
- Click on cell `D1` once and double click on the square in the bottom right corner of the cell to apply the formula to all cells in the column,
- In cell `E1` type formula `=A1=C1`,
- Click on cell `E1` once and double click on the square as before,
- In cell `F1` type formula `=COUNTIF(D1:E65536;0)`,
- If `F1` equals 0, the commitment is verified.

Part II: Comparing decommitted list to assterted list
In Calc:

- Let `$asserted$` represent `$electionDir$/AssertedReceipts.csv` if `$filePath$` is /i/Ui, otherwise let it represent `$electionDir$/AssertedTally.csv` if `$filePath$` is /i/Si
- Open `$electionDir$/$asserted$.csv`,
- Click `Data` → `Sort`. Select `Sort by Column A` (Ascending) and `Then by Column B` (Ascending) → OK,
- In `Z:/$filePath$.csv` click on column `A` → `CTRL+C`,
- Open `$electionDir$/Marks/M`i`.csv`,
- Right click on column `A` → `Insert Columns` → `CTRL+V`,
- Click `Data` → `Sort`. Select `Sort by Column A` (Ascending) and `Then by Column B` (Ascending) → OK,

- Holding CTRL click columns A and B → CTRL+C,
- In the asserted list, click column C → CTRL+V,
- In cell E1 type formula =A1=C1 and apply formula to all cells in the column as previously described,
- In cell F1 type formula =B1=D1 and apply to all cells in the column,
- In cell G1 type formula =COUNTIF(E1:F65536;0),
- If G1 equals 0, Z:/$filePath$.csv matches $electionDir$/AssertedReceipts.csv.

PART III: Comparing decommitted list to linkage list
In Calc:

- Let X = A if the asserted receipt list was opened during Part II otherwise let X = B if the asserted tally was opened instead. Let Y be the $i+2$-th letter of the alphabet,
- Open $electionDir$/LinkageList.csv,
- For each print audited ballot $1 \leq j \leq p$:
  - Click on cell Yj → CTRL+C,
  - In Z:/$filePath$.csv in the name box (top left) type 'A' → CTRL+V → Enter. The audit fails if the contents of the active (highlighted) cell do not match the contents of cell Xj in the linkage list.
  - In $electionDir$/Marks/Mi.csv in the name box type 'A' → CTRL+V → Enter. The audit fails if the contents of the active cell does not equal -1.