
# CALTECH/MIT
# VOTING TECHNOLOGY PROJECT

**A multi-disciplinary, collaborative project of the California Institute of Technology – Pasadena, California 91125 and the Massachusetts Institute of Technology – Cambridge, Massachusetts 02139**

## TITLE Practical End-to-End Verifiable Voting via Split-Value Representations and Randomized Partial Checking

**Name** Ronald L. Rivest

**University** MIT CSAIL

**Name** Michael O. Rabin

**University** Harvard SEAS, Columbia SEAS

**Key words:**

.

**VTP WORKING PAPER #122**
**Date: April 3, 2014**

# Practical End-to-End Verifiable Voting via Split-Value Representations and Randomized Partial Checking

Michael O. Rabin
Harvard SEAS, Columbia SEAS
Cambridge, MA 02138
New York, NY 10027
morabin@gmail.com

Ronald L. Rivest
MIT CSAIL
Cambridge, MA 02139
rivest@mit.edu

April 3, 2014

## Abstract

We describe how to use Rabin's "split-value" representations, originally developed for use in secure auctions, to efficiently implement end-to-end verifiable voting. We propose a simple and very elegant combination of split-value representations with "randomized partial checking" (due to Jakobsson et al. [16]).

**Keywords:** voting, end-to-end verifiable voting, split-value representations, randomized partial checking, Star-Vote.

## 1 Introduction

"End-to-end verifiable voting systems" provide high confidence that the errors and fraud can be detected, and that the election outcome is correct.

For brevity, we presume that the reader is generally familiar with such systems. See [1, 2, 9, 15, 22, 17] for some surveys, characterizations, and highlights of the development of end-to-end verifiable voting (E2EVV) systems.

Recently, E2EVV systems have been used in actual elections [9] and are proposed for use in new systems, such as the Star-Vote system in Travis county (Austin), Texas [5].

A key benefit of such systems is that they provid a convincing *proof* that the outcome is correct, while nonetheless protecting the privacy of individual voters' ballot.

Typically, each ballot is encrypted and posted on a secure public append-only bulletin board. A ballot may be represented (encrypted) in any of several ways, such as by using the El Gamal public-key encryption method [13].

A voter can then check that her ballot (more precisely, the ciphertext for her ballot) has been properly posted, without being able to convince anyone else how she voted.

Then the election outcome and associated tally, and proof of its correctness, are also posted for everyone to verify.

Making this work securely requires additional subtle protocols, such as a protocol to prove to a voter that the ciphertext for her ballot decrypts to her plaintext vote.

We show how using Rabin's "split-value" representation method can greatly simplify things. Split-value representations have been proposed for use in secure auctions [23, 21]; the extension to voting involves further innovations, however.

This focus of this paper is narrow: we only consider the combination of *split-value representations* with the *randomized partial checking* method for proving proof of the correctness of the election outcome.

The split-value method has considerable flexibility, however, and in a companion paper (to appear),

we consider end-to-end verifiable voting systems that combine split-value representations with other methods of providing proof of correct outcome.

The method proposed here is easily explained and understood; the key properties are those of the split-value representation, and those of randomized-partial checking. Both methods are highly intuitive.

The proposed method is also extremely efficient; each ballot choice only participates in a single commitment, whose total size (input and output combined) is 60 bytes. Digital ballot representations easily fit in a 2D barcode.

A key insight as to why split-value representations pair so well with randomized partial checking is that split-value commitments can be tested for equality, but only with at most one other split-value commitment, and randomized partial checking never needs to check equality of a value with more than one other value. This makes split-value representations an ideal partner for working with RPC.

**Outline of this note.** Section 2 gives some general orientation and working assumptions. Sections 3 and 4 discuss representations of plaintext votes and the operation of "adding" votes. Section 5 reviews the construction and properties of "split-value" representations of votes. Sections 6 and 7 propose ways of committing to values and to split values. Then Section 8 describes a probabilistic method for proving the equality of values represented by split-value commitments (without revealing the values committed to).

Section 9 reviews the more-or-less standard components of an end-to-end verifiable voting system, including the vote-casting procedure (including the printing of receipts for voters), the posting of information collected from cast votes on the election web site, the online checking by voters that their information is properly posted on the election web site, and the computation and posting of the election outcome.

Section 10 sets out the general framework by which the tally and proof servers can produce the correct election outcome and a proof of its correctness, without revealing individual votes. Mixnet technology is reviewed, and the "randomized partial checking" method for verifying the correct operation of a mixnet is described, as it is realized in our proposal.

Section 12 proves the soundness of our method; Section 13 provides some pointers to related work, and Section 15 final conclusions and final remarks.

# 2 General framework

This section provides some general orientation and working assumptions. These assumptions set the stage for our exemplary implementation but are not necessary. Our proposal is intentionally structured to be similar to that of Star-Vote.

For simplicity we assume the election has only one race; our methods extend naturally to handle elections with multiple races.

We assume that we are working within the framework of an end-to-end verifiable voting system that includes:

- A *voter registration* system that determines who is eligible to vote. We denote the voters as $V_1$, $V_2$, ..., $V_n$, where $n$ is the number of voters.
- A *voting tablet* upon which a voter makes her choice.
- *Paper ballots* that record the voters' choice.
- A *printer* which can print out a ballot with a voter's choice. The printer may also print out a *receipt* for the voter.
- A *ballot box* which receives the paper ballot. The ballot box may include a scanner that scans the ballot and produces a digital representation of the ballot.
- A *tally server* (TS) that receives the digital representation of each cast ballot. The tally server produces the final election outcome.
- A *proof server* (PS) that produces a proof that the election outcome is correct.
- An *web server* that maintains a secure public "bulletin board" for the election that includes a digital representation of each cast ballot, the

final election outcome, and proof that the election outcome is correct. The names of voters who have voted may also be posted.

In addition, we assume that the voting system has the following components or characteristics:

- We assume that each device or program in the system has a good and independent source of random numbers. If some devices have predictable sources of random numbers, then the privacy guarantees of our proposal are weakened, but the guarantees of correctness are not affected.

- We assume that each device or program in the system is capable of performing any necessary cryptographic operations, and that any necessary keys have been generated and distributed to the devices needing them before the election. (For example, the ballot boxes should have the public key of their tally server, and should be able to encrypt messages to the tally server using the public key of the tally server.)

- There is a unique *ballot id* (aka "bid") that is printed on the paper ballot, and that is used as an index to look up a voter's vote information on the election web site. The ballot id is *not* tied to the voter's name; it is an anonymous pseudonym for the voter. Ballot ids should be randomly assigned, but no two voters should have ballots with the same ballot id.

- The voter's printed *receipt* includes the ballot id and a copy (or hash) of the information about a voter's vote that is posted on the web site. The receipt acts as a commitment and as a backup for the information posted on the web site for a ballot id. We assume there is some way to assure the receipt's authenticity.

- The voting process includes a means (*ballot casting assurance*) by which the voter can assure herself that the information to be posted next to her ballot id correctly represents her choice. Benaloh's "cast or challenge" protocol [4] may be used here.

- The system provides a *dispute process* whereby a voter may protest that her receipt is not consistent with the information posted on the web site. For example, there may be no entry at all on the web site for her ballot id, or the information may be incorrect. The dispute process may include a process for updating or correcting the web site.

Our proposal does not use any "custom cryptography;" commitments are implemented with a NIST-standardized hash function, and encryption and digital signatures are implemented with NIST-standardized symmetric and public-key algorithms.

We note that our proposal considers only "poll-site" voting; remote voting and voting over the internet are not within the scope of this proposal.

# 3 Representation of Votes

Let $\mathcal{V}$ denote a set of values that contains a representation of each possible choice a voter may make. We might have

$$\mathcal{V} = \{\text{YES}, \text{NO}, \text{ABSTAIN}\},$$

$$\mathcal{V} = (\text{The set of all possible candidate names})$$

$$\mathcal{V} = \{0, 1, 2, \ldots, 99\},$$

The set $\mathcal{V}$ may also contain elements that do not represent valid choices. For example, one may represent choices with two decimal digits in an election with only 79 candidates.

With a suitable $\mathcal{V}$, one may even represent write-in candidate names or ranked-choice votes.

However, one should ensure that any value in $\mathcal{V}$ requires the same number of bytes to represent, so that information isn't leaked by the representation lengths. We assume that $d = 4$ bytes suffices.

As a running example, we consider the last example above, so that each possible voter choice (including not voting) is coded as a decimal number of fixed length (two digits, in this example).

# 4 Adding values

To support split-value representations there needs to be a "addition operator" ADD allowing one to combine two values in $\mathcal{V}$ yielding a third value in $\mathcal{V}$.

For any pair of values $u$, $v$ in $\mathcal{V}$ let $\text{ADD}(u, v)$ the member of $\mathcal{V}$ that is the "sum" of $u$ and $v$. We may write $\text{ADD}(u, v)$ as $u + v$.

The addition operation must admit inverses, or equivalently "subtraction", so that given $u + v$ and $v$ we can compute $u$ as $(u + v) - v$, and that given $u + v$ and $u$ we can compute $v$ as $(u + v) - u$.

In technical terms, the set $\mathcal{V}$ with binary operation "+" forms a commutative group with identity 0; the inverse of element $u$ is written $-u$, so $u + (-u) = 0$.

For our running example, where $\mathcal{V}$ is the set of two-digit decimal numbers, the operation "+" might mean "addition modulo 100", so that $66 + 58 = 24$.

# 5 Split-Value Representations

A split-value representation of a value $x$ in $\mathcal{V}$ is a two-part additive representation of $x$.

More precisely, a *split-value representation* of $x$ is a pair $(u, v)$ of elements in $\mathcal{V}$ such that

$$x = u + v \ . \tag{1}$$

A value $x$ will have many split-value representations. In our running example (with two-digit values modulo 100), the value 24 may be split-value-represented as $(66, 58)$, as $(97, 27)$, or as $(16, 8)$.

One may pick at random a split-value representation $(u, v)$ of $x$ by first choosing $u$ uniformly at random from $\mathcal{V}$, and then computing the unique value $v$ such that $x = u + v$.

**Key property.** For a randomly-chosen split-value representation $(u, v)$ of $x$, revealing just $u$ or just $v$ reveals *nothing* about $x$.

(Note that a given value $u$ may be the first component of a split-value representation for *any* $x$.)

# 6 Commitments to values

We now consider cryptographic commitments to values in $\mathcal{V}$. Cryptographic commitments are well studied; we review only the essentials here. The reader familiar with commitments may skip this section and presume that $\text{COM}(u, r)$ defines an acceptable cryptographic commitment function that commits to a value $u$ using randomization parameter $r$.

Given a value $u \in \mathcal{V}$, a device may compute a *commitment* to $u$ as a value $c = \text{COM}(u, r)$ using a suitable procedure COM and an additional secret random input $r$.

The values $u$ and $r$ remain secret until the commitment is *opened*, at which time $u$ and $r$ are revealed. Others can then verify the correctness of the commitment by recomputing $c$.

We write $\text{COM}(u)$ instead of $\text{COM}(u, r)$ when random input $r$ may be understood from context.

**Properties.** A cryptographic commitment has properties similar to those of a sealed envelope.

The commitment should be *binding*: for a given commitment $c$ *no one* should be able to produce more than one value $u$ with an associated $r$ such that $c = \text{COM}(u, r)$. It shouldn't be possible to "open the envelope" in more than one way (revealing different $u$ values within).

The commitment should also be *hiding*: someone seeing the commitment $c = \text{COM}(u, r)$ should not thereby improve his chances of guessing $u$. An observer should not learn anything about $u$ by seeing $c$. The envelope hides its contents.

**Specifics.** There are many good ways to implement a commitment scheme.

We suggest using the HMAC (hashed message-authentication code) construction due to Krawczyk et al. [19] as a basis for a commitment function. HMAC uses an arbitrary hash function $h$. HMAC does exceptionally well as an implementation of a pseudo-random function, and thus provides an excellent foundation for a commitment function.

We suggest $h =$ SHA3-224 (the NIST Secure Hash Algorithm Standard 3 with 224-bit (28-byte) out-

put).[1]. Thus,

$$\textsc{Com}(u, r) = \textsc{hmac}_h(r, u) \ , \tag{2}$$

where $r$ is the HMAC key and $u$ is the message. Then COM takes a 28-byte random value $r$, an arbitrary-length message $u$, and produces a 28-byte output $c$ that is a commitment to $u$. Thus, a triple $(u, r, c)$ where $c = \textsc{Com}(u, r)$ requires 60 bytes to represent, with our assumption that $u$ may be represented in four bytes.

This method provides 112 bits of binding security (collision-resistance) and 224 bits of hiding security (pre-image resistance).

Other commitment schemes may be used to obtain different efficiency/security tradeofs. One might define $\textsc{Com}(u, r)$ as $\textsc{hmac}_{\textsc{sha1}}(r, u)$ or as $\textsc{sha1}(u \parallel r)$ (for fixed-length $u$).

# 7 Commitments to split values

We also use *split-value commitments*: commitments to a value $x$ in $\mathcal{V}$ produced as a pair of commitments to the components $u, v$ of a split-value representation $(u, v)$ of $x$:

$$\textsc{ComSV}(x) = (\textsc{Com}(u, r), \textsc{Com}(v, s))$$

where $r$ and $s$ are random values (28 bytes each). The length of $\textsc{ComSV}(x)$ is 56 bytes, since it consists of two 28-byte commitment values.

Such a split-value commitment may be opened by opening both of its component commitments.

We let $\textsc{Val}(A)$ denote the value committed to by split-value commitment $A$.

However, sometimes only one of the two component commitments is opened. In that case, the value committed to remains information-theoretically secret.

In the next section we see how this "half-opening" idea enables probabilistic proofs that two split-value commitments are commitments to the same value, without revealing what that value is.

[1] `http://en.wikipedia.org/wiki/SHA-3`

# 8 Proving equality of split-value commitments

We now show how one may (probabilistically) prove that two split-value commitments are to the same value, without revealing that value.

Suppose a device has produced two split-value commitments

$$\begin{aligned} c_1 &= \textsc{ComSV}(x) = (\textsc{Com}(u, r), \textsc{Com}(v, s)) \\ c_2 &= \textsc{ComSV}(x') = (\textsc{Com}(u', r'), \textsc{Com}(v', s')) \ . \end{aligned}$$

The device now wishes to prove to an examiner that $x = x'$, without disclosing $x$ (or equivalently, $x'$). How can it do so? We describe an efficient procedure for doing so, following Rabin et al. [23]. See Figure 1.

**Claim of shift value.** The device first outputs the claimed "shift" value $t$ such that both of the following equations hold:

$$t = u - u' \text{ and} \tag{3}$$

$$t = v' - v \ . \tag{4}$$

Such a value exists if and only if $x = x'$.

**Challenge.** Then the device is challenged by the examiner. The examiner generates a random "challenge bit" (for example, by flipping a coin or using some other unpredictable random bit generator). Depending on the random challenge bit, the examiner asks for either the "left" commitments to be opened, or for the "right" commitments to be opened.

**Response.** If the random challenge bit is 0, the device must "open the left commitments' and produce

$$u, r, u', \text{and } r' \ ;$$

the examiner checks that $\textsc{Com}(u, r)$ and $\textsc{Com}(u', r')$ are correctly computed, and that $t = u - u'$.

Similarly, if the random challenge bit is 1, the device must "open the right commitments" and produce

$$v, s, v', \text{and } s' \ ;$$

the examiner checks that $\textsc{Com}(v, s)$ and $\textsc{Com}(v', s')$ are correctly computed, and that $t = v' - v$.

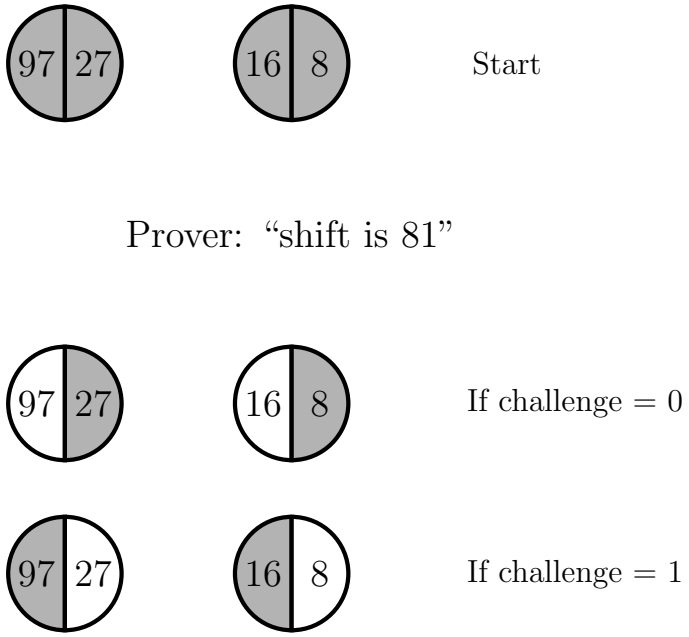


Figure 1: Proving equality of two split-value commitments. To start, there are two split-value commitments to the same value: in this example, to 24 (modulo 100): $(97, 27)$ and $(16, 8)$). The prover states that the "shift" value is 81, since $81 = 97 - 16$ and $81 = 8 - 27$ (modulo 100). Then the prover is given a challenge bit. If the challenge bit is 0, he opens the left-hand side of each commitment, and the examiner may verify that they differ by the given shift value. If the challenge bit is 1, the prover opens the right-hand side of each commitment, and the examiner may similarly verify that they differ by the shift value (subtracting in the reversed order). The opened values are shown with a white background, while the unopened values are shown with a dark gray background. No matter what the challenge bit, the examiner learns nothing about the value committed to (24, in this case). Yet the prover fails with probability at least $1/2$ if the split-value commitments are to different values.

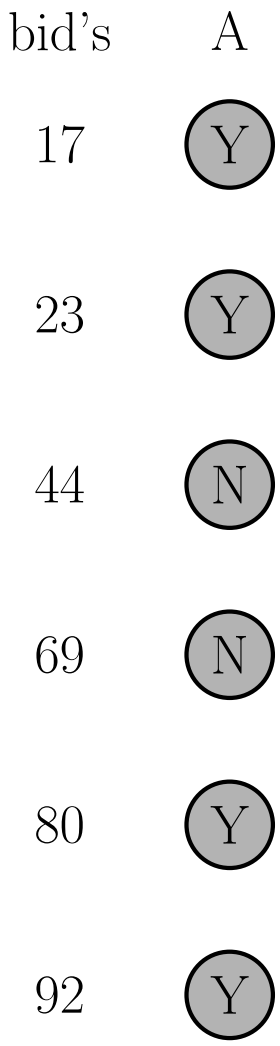


Figure 2: Initial posting for a six-voter election for a yes/no (Y/N) ballot question. The six voters' receipts are posted, each with a ballot-id and split-value commitment to a vote (which is either Y or N). The split-value commitments are unopened (shown as dark gray), so voter privacy is protected.

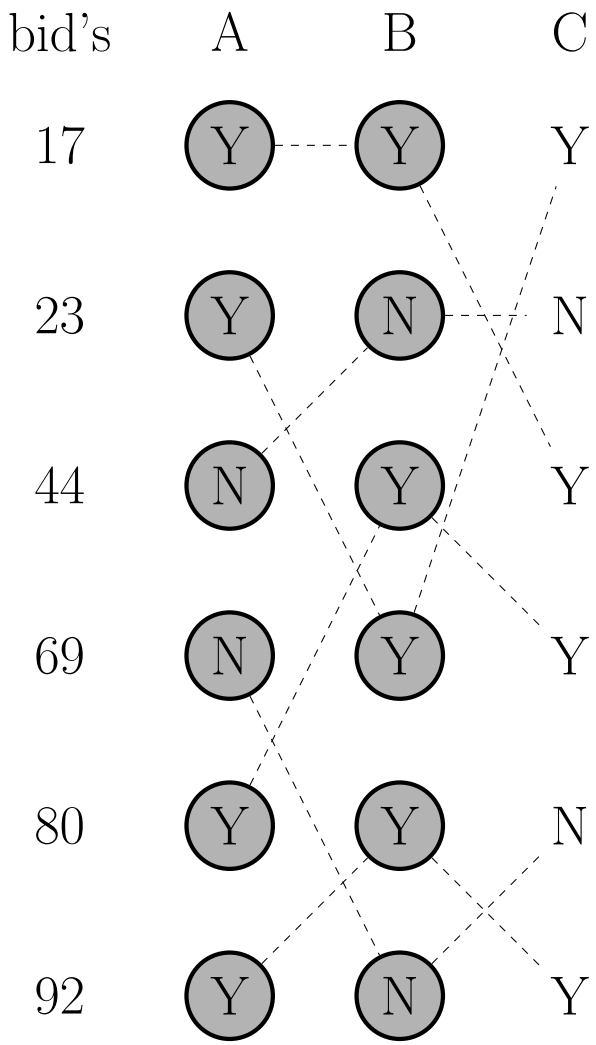


Figure 3: The first posting for the proof of correctness. Two new columns (B and C) are added. Column B split-value commits to a permutation of the votes committed to in column A. Column C lists in plaintext a permutation of the votes committed to in column B. The permutations are represented as dashed edges; these edges are committed to but not yet revealed. Values in column C are plaintext, enabling the election outcome to be computed by anyone. The final tally is 4 Y's to 2 N's; the ayes have won.

$$Q = 25314564321562316253652412345 6 \ .$$

Figure 4: Rolling dice to get the random number seed. Here thirty six-sided dice have been rolled to obtain the thirty-digit seed $Q$. The seed may be extended by appending the hash of the current state of the election web site.

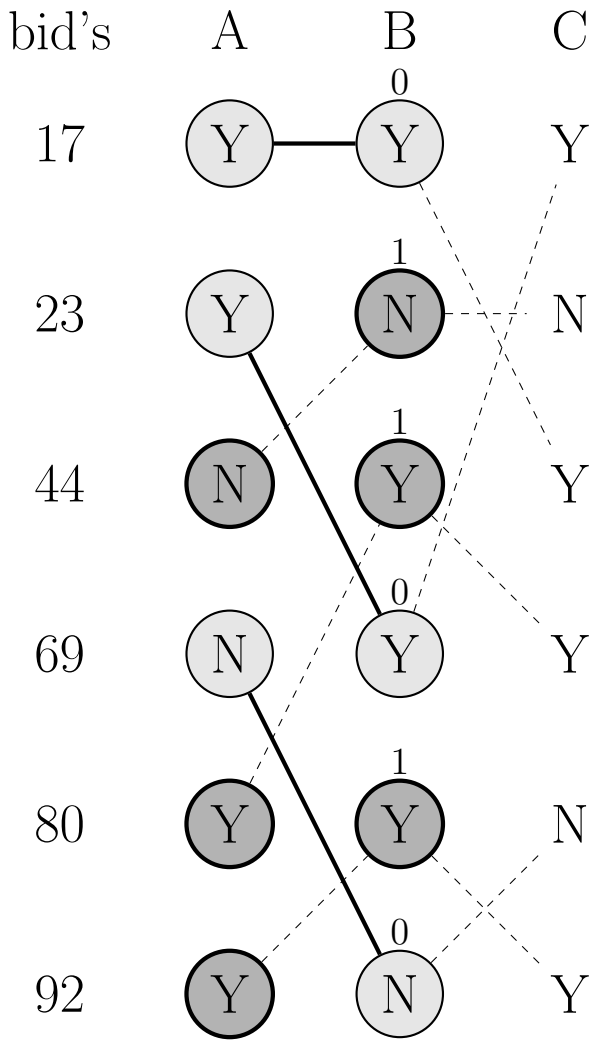


Figure 5: Revealing some AB edges. Six challenge bits are derived from the dice roll; they are shown above the corresponding B nodes. For those B nodes with a 0 challenge bit, the commitment to the edge connecting it to an A node is opened (so the edge is shown solid/revealed). The B node and the A node participate in a split-value commitment equality test, so each is half-opened (and shown in light gray).

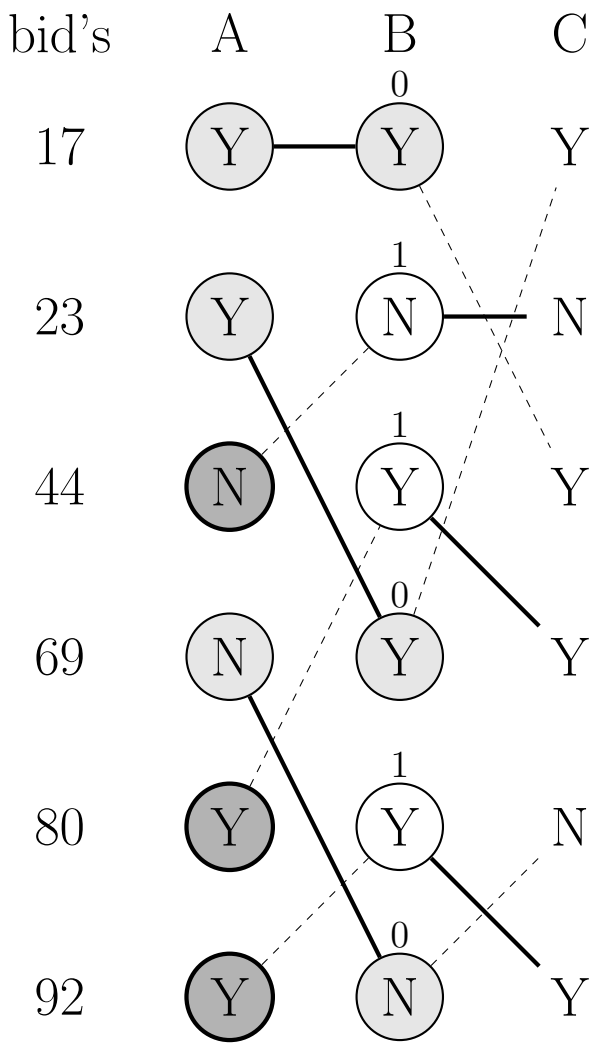


Figure 6: Revealing some BC edges. For those B nodes with a 1 challenge bit, the B node is fully opened (shown in white) and the commitment to the edge connecting it to a C node is opened (so the edge is shown solid). No solid path exists from an A node to a C node, since each B node is either proved equal to an A node (for a 0 challenge bit) or proved equal to a C node (for a 1 challenge bit), but not both. This graph represents the final state of the posted proof: all white B nodes are fully revealed, the light-gray nodes in columns A and B are half-opened, the dark gray nodes in column A are unopened, all dashed edges are unrevealed, and the all solid edges are revealed. Anyone may check the consistency of the revealed data.

**Proof.** The proof of equality for the given pair of split-value commitments then consists of the shift value, the challenge, and the response. This is an interactive proof, since the examiner produces his challenge after the device states the shift value. Standard methods, such as the strong Fiat-Shamir heuristic (see Bernhard et al. [7]), may be used if desired to convert this proof into a non-interactive one.

**Privacy.** Note that the examiner gains no information about $x$ (or $x'$) from the information disclosed.

**Soundness.** A device that tries to cheat and persuade the examiner that two split-value commitments are to the same value, when they are not, can satisfy the examiner with probability at most $1/2$, since at most one of the two equations $t = u - u'$ and $t = v' - v$ is valid. Thus, the examiner has a chance of at least $1/2$ of unmasking a cheating device. This suffices for our needs.

**Summary.** One can probabilistically prove the equality of the values $x$, $x'$ represented by two given split-value commitments, without revealing any information about the value(s) $x$, $x'$ so represented.

However, an equality proof between two split-value commitments involves opening one of the two commitments in each split-value commitment pair. Thus, each split-value commitment may be used in at most one such equality test.

# 9 Vote Casting, Posting, Online Checking, and Outcome Determination.

This section reviews the vote casting process, the posting of (copies) of the voters' receipts, the online checking of these postings, and the posting of the election outcome.

These procedures are fairly standard for an end-to-end verifiable voting system, but are included here for completeness.

These are the steps that take place before the posting of the proof of correctness for the election outcome, which is described in the next section.

## 9.1 Checking In, Tablets, and Vote Casting

This section gives an overview of the vote-casting process from the voter's perspective.

**Check-in.** Voter $V_i$ checks in, and receives a random ballot id $bid_i$. No record is kept of the association between the voter's name or index $i$ and the ballot id she receives. Every voter receives a *distinct* ballot id.

**Making a choice.** The voter uses a touch-screen "voting tablet" to compose her vote. The voter initializes the tablet using a ballot-style code she obtained at check-in. The ballot style code is not the same as her ballot id, which she also enters into the tablet.

After the voter has entered her choice $x_i$ (remember that we assume there is only a single race, for expository simplicity), the tablet computes a split-value commitment to $x_i$:

$$\textsc{ComSV}(x_i) = (\textsc{Com}(u_i), \textsc{Com}(v_i)) \ .$$

**Printing ballot and receipt.** The voter requests that her completed ballot and associated receipt be printed.

The printed ballot gives the ballot id, the split-value commitment to her choice, and the values required for opening the split-value commitment:

$$(bid_i, \textsc{ComSV}(x_i), u_i, r_i, v_i, s_i) \tag{5}$$

where

$$\textsc{ComSV}(x_i) = (\textsc{Com}(u_i, r_i), \textsc{Com}(v_i, r_i)) \ . \tag{6}$$

The printed receipt contains just the barcode and split-value commitment:

$$(bid_i, \textsc{ComSV}(x_i)) \ . \tag{7}$$

Alternatively the receipt may contain just the ballot id and a hash of the split-value commitment. We don't consider this simple variant further.

The printed ballot and receipt have information in human-readable form, but may also have a 2D barcode (e.g. pdf417 format[2]) giving the same information.

If, as we assume, the paper ballot is the only means of communication between the tablet and the ballot box, a 2D barcode is a good way to represent information on the ballot. Similar voting systems might instead use an available electronic link between the tablet and the ballot box.

**Ballot-casting assurance.** The voter has some way to ensure that her ballot and receipt correctly represents her choice (in both printed and barcode form). This may be accomplished, for example, using Benaloh's "cast or challenge" protocol [4].

**Casting.** The voter casts her ballot in the ballot box, which contains a scanner. The information on the ballot is scanned and transmitted to the tally server. The voter keeps the receipt, and may take it home.

The vote-casting process should ensure that the voter may not take the receipt home unless the corresponding ballot has actually been cast.

## 9.2 Posting of receipts

When the ballot is cast and scanned, the tally server obtains the information given in equation (5), from which it also obtains the subset available on the receipt (equation (7)).

The transmission of the information of equation (5) from the ballot box to the tally server is cryptographically protected, say by using encryption with the public key of the tally server, and authentication via the ballot box's digital signature. We omit details, as the desired confidentiality and integrity can be assured in any number of standard ways.

After the polls close the tally server posts (via the web server) a copy of all of the receipts the voters have been given. That is, it posts the values

$$(bid_i, \textsc{ComSV}(x_i)) \ .$$

[2]See `http://www.makebarcode.com/specs/pdf417.html`; this format can store up to 686 bytes per square inch.

obtained from each cast vote. Note that $u_i$, $r_i$, $v_i$, and $s_i$ are not posted, as these values would reveal $x_i$. The receipts are posted in order by ballot id.

All information posted on the election web site is digitial signed by the poster.

**Privacy.** Note that the tally server knows how each voter voted (more accurately, it knows the association between $bid_i$ and $x_i$). We trust the tally server not to reveal this correspondence. Perhaps it runs a trusted implementation on a hardware security module. Voter privacy is primarily assured through this combination of anonymous ballot id numbers and a trusted tally server implementation.

## 9.3 Online checking

A few days are allowed for voters to confirm that the web site correctly posts their receipts. Voters may check using a home computer, or using some service (perhaps provided by political parties). Note that receipts do not need to be protected from disclosure; they are posted on the web site.

If a voter's receipt (with her ballot id) does not appear on the web site, or appears incorrect, the voter may protest, effecting an investigation and possible web site correction. We omit details, which are admittedly delicate, as voter privacy must be maintained as much as possible. Note that the voter may be protesting maliciously to discredit the election or the voting system. The corresponding paper ballot (which has the unique ballot id printed on it) may need to be consulted during an investigation.

As usual with E2EVV systems, it is recommended but not mandatory that voters check online for their receipts. It assumed that a sufficient sample of voters do check, so that significant web site manipulation would almost certainly detected. (If a problem is detected, the best remedy may well be to count the paper ballots.)

**Checking names.** The web site may also post the names of all voters casting votes. There should be as many names as receipts, but voter names are not associated with receipts. Voter names are posted alphabetically, while receipts are posted in order by ballot id.

Members of the public may notice and report names of non-voters on the list (who had perhaps died or moved away). It isn't clear what remedy there is for such occurrences, other than to clean up the voter registration list for the next election. (Some versions of E2EVV voting enable one to delete receipts from ineligible voters; in the simple version we propose here this isn't possible, since there are no records linking voter names with ballot ids.)

**Deadline.** Once the protest period is over, the online record of receipts becomes the official electronic record of the cast votes.

### 9.4 Outcome Determination.

The tally server now knows what it needs to compute the election outcome and post it on the web site.

The tally server can compute the correct election outcome, since it has not only all of the split-value commitments to the voters' choices, but also the values $u_i$, $r_i$, $v_i$, and $s_i$. The tally server may thus compute each $x_i$ as $u_i + v_i$. Once the tally server computes all the $x_i$'s it can apply the appropriate voting rule to determine the election outcome, which it posts on the election web site.

Our proposal handles arbitrary voting rules, not just the standard plurality rule (where the candidate with the most votes wins). "Instant-runoff voting" or other complex rules may be used, since the values $x_i$ are obtained in full plaintext form, and may have any agreed-upon format or interpretation. For example, $x_i$ may list candidates in order of preference. Similarly, the value $x_i$ may be represent a write-in vote.

## 10 Proving Correctness

This section describes how the proof server provides a proof of correctness for the posted election outcome.

The tally server's posted claim about the election outcome should not be blindly trusted (elections are too important!), so the voting system must also produce a compelling *proof* that the posted election outcome is correct.

The *proof server* supplies such a proof. The proof server may be the same computer as the tally server, but now it is playing a new role. We assume that the proof server knows all the information the tally server knows.

The proof of correctness, however, must not reveal the linkage between any given ballot id and its associated plaintext vote. Otherwise the proof would enable coercion and vote-selling.

The proof server already knows the correspondence between ballot ids and plaintext vote. So, our goal is not to prevent the tally and proof servers from knowing this correspondence; that is not possible.

**Proof goal.** *The proof server must produce a compellling proof that the election outcome is correct without disclosing any correspondence between ballot ids and plaintext votes.*

Because the proof of correctness is available for all to review, losing candidates are motivated to "throw in the towel," knowing they have really lost fair and square.

This property is a major desideratum for any voting system, since if losing candidates and their supporters believe they have been cheated of victory by election fraud there may be riots or worse. Thus, the proof of correctness should be compelling.

**Proof review.** Once the proof is posted, election officials, losing candidates, and the public review this proof.

If the proof is satisfactory, the stated election outcome is accepted as final and official.

If the proof is not satisfactory or is not sufficiently convincing (as might rarely happen when the margin of victory is extremely small), additional assurance may be gained by counting or sampling the paper ballots. A procedure such as a ballot-polling risk-limiting audit might be a reasonable procedure to use in some cases (see Lindeman et al. [20]).

We emphasize that *any* discrepancy in the posted proof is adequate justification for rejecting the entire proof. If a commitment fails to open properly or the split-value representations don't add up properly, then one has definitive evidence that there is either fraud or serious error in the proof system. The best remedy then is probably to count the paper ballots,

in addition to performing an investigation as to the cause of the discrepancy.

**Proof structure.** We suggest the proof take the following form, which is familiar in the voting literature, as it is the proof structure for a verifiable mixnet:

- In addition to the first posted list of $(bid, \textsc{ComSV}(x_i))$ pairs, one or more (two, in our case) new lists are posted that omit ballot ids, and which have different split-value commitments to the same plaintext choices in different (randomized) orders. See Figures 2–6.
- Proof is posted that each successive list contains commitments to the same set of values as in the preceding list. The final list is thus a list of split-value commitments to the same plaintext values as in the first list (but in different order).
- The commitments in the final list are all opened, and the correct election outcome may be computed by anyone from the world-readable list of all of the plaintext votes.

**Proving equality of lists.** We know two general approaches for providing proof of correctness that follow the above model:

- an approach based on *randomized partial checking* (RPC) due to Jakobsson et al. [16]), and
- an approach based on *value consistency proofs* (VCP) due to Rabin et al. [23, 21] for secure auctions.

We denote these approaches as SV/RPC ("split-value/randomized partial checking") and SV/VCP ("split-value/value-consistency proof").

The SV/RPC method is the focus of this paper, and is described in the following sections. The SV/VCP method will be described in a companion paper.

The SV/RPC method takes advantage of the following beautiful coincidence:

- a split-value commitment may take part in at most one equality test, and
- the randomized partial-checking method never needs a value (a split-value commitment) to take part in more than one equality test!

The "fit" here is excellent, and the resulting method is exceptionally simple and elegant.

Our method enjoys the key property of the original randomized-checking procedure that a "proof" of a false election outcome will be accepted with a probability that decreases exponentially with the number of votes changed in the false result. Therefore an acceptable proof is compelling evidence that the election outcome is correct to within a few votes. (If the margin of victory is just a few votes, jurisdictions typically hand-count all of the paper ballots anyway to provide the best assessment of the correct election outcome.)

SV/VCP achieves a higher degree of assurance than SV/RPC, at the cost of needing the tally server to replicate split-value commitments a number of times. (See paper to appear.)

**Vote-selling and coercion.** Note that the correspondence between ballot ids and plaintext votes is not publicly known but *is* known to the tally and proof servers, and thus potentially known to election officials. A corrupt election official might ask that a voter reveal her ballot id, look up her plaintext vote on the tally server, and then reward or punish the voter depending on her vote. This issue also arises when paper ballots are accessed for the purposes of a post-election audit.

Protection against potentially corrupt election officials should be provided by running the tally server in a hardware security module with limited functionality, and by having strict access controls on the paper ballots (which have the ballot ids printed on them).

We omit further discussion of these important and rather tricky issues.

# 11 Mixnets and Randomized Partial Checking

This section reviews mixnets and the randomized partial checking method of Jakobsson et al. [16] for producing a "verifiable mixnet."

**Mixnets.** David Chaum [10]) introduced mixnets as a way of providing anonymity in voting or similar applications.

A sequence of "mix-servers" successively permute a list of encrypted values, each using its secret permutation. The final list is thus scrambled by the composition of the secret permutations. No single mix-server knows how inputs correspond to outputs overall. For voting, the final list is decrypted, and an election outcome can be determined from the revealed plaintext votes.

Mixnets normally use *public-key* technology. Each mix server either decrypt its inputs, or re-randomizes them. The former scenario is for *decryption mix-nets*, where each element is an "onion" made by successively encrypting a value with the public key of each mix-server, in reverse order. The latter scenario is for *re-encryption mix-nets*, where inputs are encrypted with a public-key encryption method (such as ElGamal's [13]) that enables re-randomization without knowledge of the secret key.

**Perspective on our proof.** Our situation is similar but we use split-value commitments rather than public-key encryption. We are thus using shared-secret-key methods, and the mix server(s) know all of the plaintext votes.

It is important to realize that the *first* mix-server knows how ballot ids correspond to plaintext votes, although the second and later servers do not.

Our goal here is thus not to protect the correspondence from being known to the server(s) (at least not to the first server), but to allow the server(s) to produce a proof of correctness of the election outcome that does not publicly reveal the correspondence.

We must trust the server(s) not to surreptitiously reveal this correspondence.

Because we are using private-key technology (split-value commitments), there seems little security benefit in having more than one mix-net server. (Of course, having a backup server available for reliability might be a good idea! But this is a different issue, and the backup knows everything the original server knows.)

Our SV/RPC design therefore uses a simplified "mix-net" with split-value commitments (rather than public-key encryption) and only a single "mix-server".

Our proof server plays the role of this single "mix-server", and effects two "rounds" of the mix-net, so there will be a total of three lists (two of split-value commitments, and a final list of plaintext votes).

**Randomized partial checking.** We now describe how randomized partial checking allows the proof server to produce convincing proof that the posted election outcome is correct, without thereby revealing the correspondence between ballot ids and the final list of fully opened split-value commitments.

Jakobsson, Juels, and Rivest [16] propose a method called "randomized partial checking" (RPC) for providing such proof. Here mix-servers partially reveal correspondences between each pair of adjacent columns, but in such a way that no path can be traced from any value in the first column to a value in the last column.

The reader may find it helpful to review the RPC paper [16] before proceeding.

We now sketch a simplest possible instantiation of this idea, with only three lists and a single mix server.

The approach is really a very straightforward combination of the use of split-value representations and the randomized partial-checking procedure. There is little in the way of additional "engineering" needed to make these ideas work well together.

The only point to notice is that the split-value equality test is probabilistic rather than deterministic; this really doesn't change much; a small adjustment is needed in the computation of the potential success probability for an adversary who tries to provide proof for an incorrect election outcome.

The proof server provides proof of correctness, using the RPC method, as follows. There are three parts:

- **Part I:** Posting of the receipts, and the three-column graph structure.
- **Part II:** Receiving the challenge.
- **Part III:** Responding to the challenge.

**Part I: Posting receipts and the graph structure.** The proof server posts the following values.

The first two lists have already been described, as they are the ballot ids and the associated split-value commitments from the voters, and are exactly what appears on the voters' receipts. We assume that the claimed election outcome has also been posted.

1. **Receipts: ballot ids and split-value commitments (A).**
   The receipts for the the voters, containing the ballot ids $bid_1$, $bid_2$, ..., $bid_n$ and associated split-value commitments $A_1$, $A_2$, ..., $A_n$; (where $A_i = \textsc{ComSV}(x_i)$ is the split-value commitment from the $i$-th ballot). We suggest that receipts be listed in order of ballot id, assuming that ballot ids were randomly assigned.

2. **Reordered split-value commitments (B).**
   The split-value commitments $B_1$, $B_2$, ..., $B_n$; here each $B_j$ is a split-value commitment to a value $y_j$:
   $$B_j = \textsc{ComSV}(y_j)$$
   and the $y'_j s$ are a permutation of the $x_i$'s:
   $$y_j = x_{a_j}$$
   for all $j$, where $(a_1, a_2, \ldots, a_n)$ is a secret randomly-chosen permutation of $\{1, 2, \ldots, n\}$. Note that $B_j$ will almost surely have a different "split" of $y_j$ into $(u'_j, v'_j)$, as well as different randomization parameters than $A_{a_j}$ for the commitments, so that $A_{a_j}$ and $B_j$, although they are split-value commitments to the same value, are different.

3. **Reordered plaintext choices (C).**
   The values $C_1$, $C_2$, ..., $C_n$ where $C_j$ is a plaintext value $z_j$:
   $$C_k = z_k$$
   and the $z_k$'s are a permutation of the $y_j$'s:
   $$y_j = z_{c_j}$$
   for all $j$ where $(c_1, c_2, \ldots, c_n)$ is a secret randomly-chosen permutation of $\{1, 2, ..., n\}$.

4. **AB edge commitments ($a_j$'s and corresponding shift values).**
   For each $j$, $j = 1, 2, \ldots, n$, a commitment to the pair $(a_j, t_j)$ where $t_j$ is the "shift" needed in the split-value proof that $B_j$ and $A_{a_j}$ are commitments to the same value.

5. **BC edge commitments ($c_j$'s).**
   For each $j$, $j = 1, 2, \ldots, n$, a commitment to $c_j$.

The proof server randomly chooses the permutations $\{a_j\}$ and $\{c_j\}$, and the new splits and randomization parameters for the split-value commitments $B_j$.

For each vote, eight values are posted: the ballot id, two values each for the split-value commitments in $A$ and $B$, one value for each value in $C$, one value for each AB edge commitment, and one value for each BC edge commitment.

This first part of the proof sets the stage for the following challenge/response, which completes the proof.

**Graphical model.** What has been posted so far may be viewed as a graph with $3n$ vertices, with $n$ vertices in each of lists $A$, $B$, and $C$ (see Figures 3–6):

- Column $A$ and column $B$ form a bipartite graph that is a complete matching, with $B_j$ connected to $A_{a_j}$.
- Column $B$ and column $C$ form a bipartite graph that is a complete matching, with $B_j$ connected to $C_{c_j}$.

The graph structure is hidden; we only have for each $B_j$ a commitment to $a_j$, saying where $B_j$'s value came from, and a commitment to $c_j$, saying where $B_j$'s value goes to.

The proof server wishes now to prove that the plaintext values $C$ are to the same values as the commitments $A$ (in permuted order).

Opening all of the AB and BC edge commitments would obviously work to demonstrate the desired property, but this would reveal the correspondence between ballot ids and plaintext vote values, since one could then trace the path from each $A_i$ to where it ends up (as plaintext) in column $C$.

So, the trick is to have the proof server only open half of the "edge commitments" in the RPC manner (see [16, 18]).

**Part II: Obtaining the challenge bits.**

This part involves producing a number of *challenge bits* $q_j$ for use in the RPC protocol deciding which edge commitments to open, as well as a number of bits $q'_j$ for deciding left-half/right-half split-value commitment openings when the edge being opened is an AB edge.

The challenge bits are determined by using a *seed* for a suitable pseudo-random generator.

We emphasize that while the proof-generation procedure is randomized, the election outcome of course is not! (At this point, the election outcome has already been determined and posted.)

**Obtaining the seed and challenge bits.**

The seed (call it $Q$) may be determined by rolling thirty six-sided dice in a public ceremony, and following those digits with the 56 hex digits of the SHA3-224 hash of the current election web site, yielding a 86-character string. The dice-rolling should take place only *after* all the election web site data so far has been posted and signed. Representatives from different parties might each roll some of the dice.

We note that dice-rolling to pick a seed is sometimes done to determine which precincts are audited in a post-election audit.

The seed $Q$ is posted to the web site and signed.

Our proposal uses two sets of $n$ challenge bits each, which we call $q_j$ and $q'_j$ ($j = 1, 2, \ldots, n$). These bits will be unpredictable to an adversary (before the dice roll). These bits should be unbiased: the probability of a 0 is 1/2, and the probability of a 1 is 1/2.

With this goal in mind, we define $\textsc{lsb}(x)$ as the least-significant bit of $x$, and

$$q_j = \textsc{lsb}(\text{SHA3-224}(j \parallel Q \parallel \text{“0”})) \ .$$

where $j$ is represented in decimal. Similarly, we define

$$q'_j = \textsc{lsb}(\text{SHA3-224}(j \parallel Q \parallel \text{“1”})) \ .$$

These details are rather arbitrary; there are many reasonable ways to compute suitable pseudo-random bits $q_j$ and $q'_j$ from $Q$ and $j$. These bits appear to an adversary to be independent and unbiased.

**Part II: Posting the responses to the challenges.**

For each $j$, $j = 1, 2, \ldots, n$, the proof server must produce one of two proofs:

- if $q_j = 0$, a proof that $\textsc{Val}(B_j)$ and $\textsc{Val}(A_{a_j})$ are equal, or
- if $q_j = 1$, a proof that $\textsc{Val}(B_j)$ and $\textsc{Val}(C_{c_j})$ are equal.

In the first case, the proof server opens the commitment for $(a_j, t_j)$, revealing the AB edge between $B_j$ and $A_{a_j}$. The split-value commitments $B_j$ and $A_{a_j}$ are proven to represent equal values, using the value $q'_j$ to determine whether "left-halves" or "right-halves" should be opened.

In the second case, the proof server opens the commitment for $c_j$, revealing the BC edge between $B_j$ and $C_{c_j}$. Both halves of split-value commitment $B_j$ are opened, and the value $y_j$ committed to is thereby revealed (it should be equal to $C_j$).

The proof is not accepted by the verifier if any revealed edge connects different values, if the revealed $a_j$'s contain repeated values, or if the revealed $c_j$'s contain repeated values.

**No ABC paths.** Because the proof provided links each $B_j$ either to an $A_i$ or to a $C_k$, *but not both*, there is *no path* in the revealed graph edges from any $A_i$ (with associated ballot id) through a $B_j$ to a final $C_k$ (which is plaintext). Thus, the proof does not reveal the association between ballot ids and associated plaintext vote choices.

**At most one test per commitment.** Note the very important property that each split-value commitment in column $A$, or column $B$ participates in *at most one* equality test. This means that the split-value representation is really ideal for use with an RPC method.

# 12 Security

Note that if the claim made by the proof server that the vote values posted as described in Sections 10–11

in column C are a permutation of the vote values hidden by commitments in column A is correct, then the proof server can always construct a proof of correctness acceptable to every verifier. Thus our method is *complete*: a true statement can always be proved.

If the posted proof is not accepted, i.e. if even one of the checks fails, then the announced election outcome is not accepted and further actions are required. We now turn to proof of soundness of the method, i.e., that if the posted election outcome is not correct then the probability that the posted proof is accepted is exponentially small. The precise formulation of this statement follows.

**Soundness.** The usual RPC analysis then applies with only small modifications. Note that an adversary might undetectably modify a given ballot with probability 3/4, since it could cheat on either the $AB$ link or the $BC$ link, and a modification of one such link might be missed since only the left commitments (or only the right commitments) are opened.

Although an adversary might get away with probability 3/4 in cheating on one vote, we have the following.

**Theorem 1** *A malicious proof server who modifies $k$ votes has probability at most*

$$(3/4)^k$$

*of producing an acceptable proof of election outcome correctness.*

The proof is given in the Appendix.

**Example:** A malicious prover who tries to modify the tally by changing only 24 votes has a chance of at most one in 1000 of not being caught, and a malicious prover who changes only 40 votes has a chance of at most one in 100,000 of not being caught. Trying to change 100 votes has a probability of at most one in $3 \times 10^{12}$ of not being caught—truly negligible!

So unless the election is extremely close, fraud large enough to have affected the election outcome will almost surely be caught. If the election is extremely close, election officials (or the candidates) may anyway insist on (or be legally required to have) a full recount of the paper ballots.

# 13 Related Work

Rabin, Servedio, and Thorpe [23] give an initial auction design based on split-value representations. Micali and Rabin [21] extend this work to show how split-value representations can be used to implement secure Vickrey auctions, including a way to counter collusion between bidders.

Damgård and colleagues have recently extended their earlier works [8, 11, 12] on efficient MPC to that of *publicly-verifiable secure multiparty computation* [3]. This latest work is directly relevant here. Indeed, they mention secure voting as a possible application. And their representations are based on information-theoretically secure additively-shared representations, as are ours. But their constructions (especially the setup) are relatively complex, and require multiple rounds of interactions between the servers.

The Eperio voting system [15] is worth pointing out as another interesting E2EVV system, as it uses no public-key operations and is quite efficient. It is designed to work only with the common "mark-sense" elections (pick one of several choices), rather than more complex election types.

The SOBA ballot-auditing procedure [6] is compatible with the procedure described here, as we take care to have matching ballot ids on the electronic and paper records for a ballot.

# 14 Additional Results, Variations and Extensions

There are a number of ways that one can modify our proposal to handle additional concerns or to achieve different tradeoffs between competing goals.

For example, one could put more than one split-value representation of a voter's choice on a ballot, for additional reliability or a higher degree of proof assurance.

We can improve somewhat the error bound of Theorem 1, using an idea from Ergün et al. [14, Section 3.1] on "permutation enforcement," and by using biased $q_j$ bits; the bound can be improved to $(2/3)^k$.

We also have a rather different method, based on multi-party computations, that provides enhanced privacy against potentially malicious servers.

Our best soundness bound for $k$ discrepancies, using the SV/VCP method, is $\sqrt{\pi n}/2^{2n} + 2^{-k}$.

We also have additional results relating to the use of special-purpose hardware and threshold computation by the servers for reliability.

As noted, these results will appear in papers under preparation.

# 15 Conclusion

We have presented a very simple method for implementing an end-to-end verifiable voting (E2EVV) system, based on a combination of split-value commitments and randomized partial checking.

The proposed system is easily understood and very efficient.

# Acknowledgments

The second author gratefully acknowledges support from his Vannevar Bush Professorship.

We thank Marco Pedroso, Ronitt Rubinfeld, and Simha Sethumadhavan for helpful discussions and comments.

## Note on Intellectual Property (IP)

We do not know of any patent or other restrictions on the use of the ideas proposed here. We have not filed for any such patents (and will not). The authors place into the public domain any and all rights they have on the use of the ideas, methods or systems proposed here. As far as we are concerned, others may freely make, use, sell, offer for sale, import, embed, modify, or improve the ideas, methods, or systems described in this note. There is no need to contact us or ask our permission in order to do so. We ask only that this paper be cited as appropriate.

# References

[1] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, MIT EECS, August 2006.

[2] Ben Adida and Ronald L. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In Roger Dingledine and Ting Yu, editors, *Proceedings of the 5th ACM workshop on privacy in electronic society*, pages 29–39. ACM, 2006.

[3] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. Cryptology ePrint Archive, Report 2014/075, 2014. `http://eprint.iacr.org/2014/075`.

[4] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, pages 14–14. USENIX Association, 2007.

[5] Josh Benaloh, Mike Byrne, Philip Kortum, Neal McBurnett, Olivier Pereira, Philip B Stark, and Dan S Wallach. STAR-vote: A secure, transparent, auditable, and reliable voting system. *arXiv preprint arXiv:1211.1904*, 2012.

[6] Josh Benaloh, Douglas Jones, Eric L Lazarus, Mark Lindeman, and Philip B Stark. SOBA: secrecy-preserving observable ballot-level audit. *Proceedings of EVT/WOTE*, 2011.

[7] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[8] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology

ePrint Archive, Report 2008/068, 2008. http://eprint.iacr.org/.

[9] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II municipal election at Takoma Park: The first E2E binding governmental election with ballot privacy. In Ian Goldberg, editor, *Proceedings USENIX Security 2010*. USENIX, August 11-13, 2010.

[10] David Chaum. Untracable electronic mail, return addresses, and digital pseudonyms. *Comm. ACM*, 24(2):84–90, February 1981.

[11] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, , and Nigel P. Smart. Practical covertly secure MPC for dishonest majority, or: Breaking the SPDZ limits. Cryptology ePrint Archive Report 2012/642, 2012. http://eprint.iacr.org/2012/642.

[12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[13] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, IT-31(4):469–472, July 1985.

[14] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Information and Computation*, 189(2):135–159, 2004.

[15] Aleksander Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams. Eperio: Mitigating technical complexity in cryptographic election verification. In *Proceedings of the 2010 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[16] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Dan Boneh, editor, *Proceedings USENIX Security 2002*, pages 339–353. USENIX, 2002. Also available as IACR eprint 2002/025.

[17] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. Cryptology ePrint Archive, Report 2013/615, 2013. http://eprint.iacr.org/.

[18] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. In Ed Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2013.

[19] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: keyed-hashing for message authentication. IETF Informational RFC 2104, February 1997. http://tools.ietf.org/html/rfc2104.

[20] M. Lindeman, P. B. Stark, and V. S. Yates. BRAVO: Ballot-polling risk-limiting audits to verify outcomes. In *Proc. EVT/WOTE'12*. USENIX, 2012. https://www.usenix.org/system/files/conference/evtwote12/evtwote12-final27.pdf.

[21] Silvio Micali and Michael O. Rabin. Cryptography miracles, secure auctions, matching problem verification. *CACM*, 57(2):85–93, February 2014.

[22] Stefan Popoveniuc, John Kelsey, Andrew Regenscheid, and Poorvi Vora. Performance requirements for end-to-end verifiable elections. In *Proceedings of the 2010 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[23] M. Rabin, R. Servedio, and C. Thorpe. Highly efficient secrecy-preserving proofs of correctness of computations and applications. In *Proceedings*

*of 22nd IEEE Symposium on Logic in Computer Science*, pages 63–76. IEEE, 2007.

# Appendix. Proof of Theorem 1.

**Proof:**

Our treatment otherwise follows the standard RPC argument. Note however that we are not using Fiat-Shamir, but rather fresh randomness, to generate the seed $Q$ (this makes the arguments simpler and stronger).

Some B-list items are properly matched to A-list items: call $j$ "A-good" if

$$\begin{aligned}&\textsc{Val}(B_j) = \textsc{Val}(A_{a_j}) \text{ and}\\ &\qquad \text{there is no } j' > j \text{ such that } a_j = a_{j'} \ ;\end{aligned}$$

else $j$ is "A-bad," meaning

$$\begin{aligned}&\textsc{Val}(B_j) \neq \textsc{Val}(A_{a_j}) \text{ or}\\ &\qquad \text{there is a } j' > j \text{ such that } a_j = a_{j'} \ .\end{aligned}$$

Similarly some B-list items are properly matched to C-list items: call $j$ "C-good" if and only if

$$\begin{aligned}&\textsc{Val}(B_j) = C_{c_j} \text{ and}\\ &\qquad \text{there is no } j' > j \text{ such that } c_j = c_{j'} \ ;\end{aligned}$$

else $j$ is "C-bad," meaning

$$\begin{aligned}&\textsc{Val}(B_j) \neq \textsc{Val}(C_{c_j}) \text{ or}\\ &\qquad \text{there is a } j' > j \text{ such that } c_j = c_{j'} \ .\end{aligned}$$

If the prover tries to show that an A-bad $j$ is A-good, then he will fail with probability at least 1/2 (either due to the split-value commitment equality test, or the 1/2 chance that an $a_{j'}$ for $j' > j$ that duplicates $a_j$ will be revealed).

If the prover tries to show that a C-bad $j$ is C-good, then he will fail with probability at least 1/2, similarly (although simpler, since $B_j$ and $C_{c_j}$ both get fully revealed).

The number of places where A and C differ is at most the number of A-bad $j$'s plus the number of C-bad $j$'s. (Note that the *last* $a_j$ equal to a given value (the A-good one) may be considered as the one that "counts," and the others (the A-bad ones) correspond to values corresponding to values in the A-list that were thus missed. Similarly for the BC comparisons.)

Thus, we get a bound of

$$(3/4)^k \ ,$$

where $k$ is the sum of the number of A-bad $j$'s and the number of C-bad $j$'s, since for each A-bad or C-bad $j$ there is a 1/2 chance the prover will pick the wrong edge to reveal, or (if he picks the right edge) will choose to reveal the wrong (left-side/right-side) side of the commitments, allowing the cheating to be undetected with probability at most 3/4. ∎