Efficient End to End Verifiable Electronic Voting Employing Split Value Representations

Michael O. Rabin
Harvard SEAS
Columbia SEAS
Email: morabin@gmail.com

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

***Abstract*—We present a simple and fast method for conducting end to end voting and allowing public verification of correctness of the announced vote tallying results. In the present note voter privacy protection is achieved by use of a simple form of distributing the tallying of votes and creation of a verifiable proof of correctness amongst several servers, combined with random representations of integers as sums mod M of two values. At the end of vote tallying process, random permutations of the cast votes are publicly posted in the clear, without identification of voters or ballot ids. Thus vote counting and assurance of correct form of cast votes are directly available. Also, a proof of the claim that the revealed votes are a permutation of the concealed cast votes is publicly posted and verifiable by any interested party. We present two versions of the method, one assuring voter privacy and proof of correctness in the presence of information leaking devices, the other achieving the same goals as well as prevention of denial of service by failing or sabotaged devices.**

Advantages of this method are: Easy understandability by non-cryptographers, implementers, and ease of use by voters and election officials. Direct handling of complicated ballot forms. Independence from any specialized cryptographic primitives. Verifiable mix nets without using public-key or homomorphic cryptography, a novel result of significance beyond e-voting. Speed of vote-tallying and correctness proving: elections involving a million voters can be tallied and proof of correctness of results posted within a few minutes.

I. Introduction and Overview

End-to-End Verifiable Voting (E2EVV) systems provide high confidence that errors and fraud can be detected and that the announced election outcome is correct. See [1]–[5] for some surveys and results about E2EVV. Cramer et al. [6] have also used secret-sharing to ensure robustness of a voting system, as we do in Section X. Recently, E2EVV systems have been used in actual elections [2] and are proposed for use in new systems such as the STAR-Vote system in in Travis County (Austin) Texas [7].

The parties and agents involved in our E2EVV scenario are:

Voters: We assume n voters V_1, V_2, ..., V_n.

Tablets: Each voter uses a tablet to compose her vote. The tablet can also print out a receipt for the voter.

Election authorities: Individuals responsible for running the election.

Election Servers: Computers performing specific functions in the election.

Secure Bulletin Board (SBB): An election server providing a secure public append-only record of election-specific data, including all cast ballots, the final election outcome, and a proof of correctness of the election outcome.

Proof Server: An election server that produces a proof of correctness of the election outcome. In our method, the proof server is implemented with a two-dimensional array of independently-controlled computers; these servers are also servers in our mix-net implementation (so we also call them "mix-servers").

Tally Server: An election server that computes the election outcome from the publicly posted list of decrypted cast votes.

Adversary: The adversary attempts to cause an incorrect election outcome to be accepted. (An accepted proof of correctness as presented here, assures correctness of announced tally outcome no matter how the adversary acted.) An adversary may also attempt to violate the privacy of voters.

An election then comprises the following steps:

1) **Setup:** The list of eligible voters is determined. The list of ballot questions is determined. (For presentation purposes we assume only one question on the ballot, which may nonetheless require a complex answer such as a preference ordering of the choices. For more questions the entire method may be repeated.) Cryptographic keys are set up as necessary for tablets and election servers.
2) **Vote Casting:** Each voter uses a tablet to enter her choice on the ballot question. The system uses some convention for providing each ballot with a unique ballot id bid. The choice is "encrypted" (more on that later), and sent to the election servers with the bid. The voter is given a printed receipt with the bid and the hash of the encryption.
3) **Posting of Vote Records:** The bid's and encrypted choices are posted on the SBB at the end of election day.
4) **Verification of Postings:** Voters may access the SBB to verify that the encryptions of their choices are correctly posted (comparing their receipt with the hash of the posted encryption for their ballot).
5) **Mixing:** The mix servers anonymize the encrypted ballots by permuting their order and dissociating the encrypted ballots from identifying meta-data such as voter names or bid's. Each of $2m$ copies of the list

of n encrypted ballots is independently mixed and re-encrypted. The resulting $2m$ permuted lists are posted on the SBB.

6) **Random Challenge:** A "random challenge" (a long string of random digits) is derived by hashing the SBB contents and/or from a public dice-rolling ceremony. The unpredictability of this challenge to an adversary prevents the adversary from undetectably manipulating the election outcome.
7) **Proving consistency with cast votes:** A random half (determined by the random challenge) of the $2m$ lists of encrypted ballots are partially decrypted, to check that the mixing was properly done. The method used here depends on properties of split-value vote representations to ensure that voter privacy is preserved. The partial decryptions are posted on the SBB for all to confirm.
8) **Posting and verification of election outcome:** The other half of the $2m$ lists are all fully decrypted and posted on the SBB. Anyone may check that they are identical lists (albeit differently permuted). The final election outcome may be determined from any one of these lists.

Adversarial model. We assume that the adversary is trying to "rig" an election by trying to force an incorrect election outcome to be accepted (because it appears to have been proven correct) or to learn how some particular voters have voted.

In Sections III–IX the adversary is assumed *not* to be interested in causing the election to fail (that is, to not produce an election outcome or proof of correctness at all). Section X deals with adversaries who attempt to deny service by failing.

Innovations re other E2E methods. The elements of our end-to-end voting method are reasonably standard, except that

- Ballots are "encrypted" in a novel manner, using commitments to secret-shared split-value representations of the voters' choices.
- No modular exponentiations or public-key operations are required, yielding substantial efficiency improvements.
- The mix-net operation is proved correct in a new manner: rather than proving each step of the mix-net to be correct, the overall operation is proved correct.
- Because ballots are fully decrypted for the final tallying operation, there is no restriction on the tallying method used. Complex tallying rules (e.g. IRV) and write-in candidates are easily handled. Furthermore, no zero-knowledge proofs are required that the encrypted ballots are valid.

We thus show how using Rabin's Split Value Representation (SVR) of integers method greatly simplifies an E2E implementation. SVR methods have been proposed for implementation of secure auctions [8], [9]; the extension to voting involves, however, further innovations.

The current paper extends our previous works [10], [11] exploring such innovations; In particular, we note that our earlier work [10] has the problem that a single election server must know how everyone voted; the present work remedies that defect.

Outline of paper. We begin in Section II with some preliminary notation and a discussion of the properties of split-value representations, including methods for securely proving equality of the values represented.

Then Sections III–IX discuss each phase of our method in detail, from initial setup to creating and verifying the final proof of correctness of the election outcome.

Section X shows how to extend the basic method to one that tolerates a certain number of failures of the mix-net servers, by using Shamir's secret-sharing method.

Finally, Section XI provides some discussion of the practical aspects of our methods, and Section XII concludes.

II. Preliminaries

A. Notation

We let $x \parallel y$ denote the concatenation of x and y.

B. Representations modulo M

For a given race, votes and values used in the system are described by values modulo a given integer M. Here M is chosen large enough so that any voter choice (including a "write-in" choice) may be represented by a unique integer modulo M. In the following, additions and subtractions of values are performed mod M.

Our methods are independent of the way such values are used to represent candidates or complex choices (as with preferential balloting).

Some of our methods (see Section X) require that M be prime.

C. Split-Value Representations

Our methods are adapted from those of [8], [9].

Definition 1: Let x be a value modulo M, so that $0 \le x < M$. A *split value representation* of x is any vector

$$X = (u, v)$$

where u and v are values modulo M such that $x = u + v \pmod M$.

Definition 2: We define the *value* of a split-value representation $X = (u, v)$ modulo M to be

$$\textsc{Val}(X) = (u + v) \bmod M \ .$$

Note that there are M different split-value representations of any given value x, since u can be arbitrarily chosen from $\{0, 1, \ldots, M-1\}$, and then the corresponding v derived via $v = (x - u) \bmod M$.

Definition 3: A *random split-value representation* of a value x modulo M is a randomly chosen split-value representation of x modulo M.

D. Commitments

Commitment to values mod M We use a commitment function $\text{COM}(K, u)$ employing a (randomly chosen) key K to commit to value u modulo M.

It is assumed that COM is computationally hiding: given the value $C = \text{COM}(K, u)$, it is infeasible to gain any information about u.

Opening a commitment $\text{COM}(K, u)$ means to reveal K and u; this opening can be verified by re-computing $\text{COM}(K, u)$.

It also assumed that it is computationally infeasible to find two pairs (K, u) and (K', u') such that $\text{COM}(K, u) = \text{COM}(K', u')$. This renders the commitment by COM to be computationally binding; no one can open a commitment in more than one way.

COM can be implemented, say, by use of AES with 256 bit keys, or with the HMAC cryptographic hash function.

We sometimes write $\text{COM}(u)$ instead of $\text{COM}(K, u)$, with the understanding that a randomly chosen K is used (which is revealed with u when the commitment is opened).

Commitment to split-value representations

Our use of a commitment to a split-value representation is analogous to the "encryption" of a choice in other E2E methods.

Definition 4: A *commitment* $\text{COMSV}(X)$ *to a split-value representation* $X = (u, v)$ is a pair of commitments, one to each component:

$$\text{COMSV}(X) = (\text{COM}(u), \text{COM}(v)) \ .$$

Note that $\text{COMSV}(X)$ denotes commitment to a split-value vector representation of a value x, $0 \leq x < M$, while $\text{COM}(u)$ is a commitment to a value u, $0 \leq u < M$.

The following fact is crucial to the security of our methods.
Fact. If just one of the two coordinates u or v in a commitment to a random split value representation X of a value x is opened, then no information about the value x is revealed.

E. Proving equality of commitments

The nice thing about commitments to split-value representations is that they can be (probabilistically) proved equal without revealing the values represented.

Suppose a Prover asserts that

$$\begin{aligned} \text{COMSV}(X) &= (\text{COM}(u_1), \text{COM}(v_1)) \\ \text{COMSV}(Y) &= (\text{COM}(u_2), \text{COM}(v_2)) \end{aligned}$$

represent the same value: $\text{VAL}(X) = \text{VAL}(Y)$. To prove this, the Prover first reveals t, where

$$t = u_2 - u_1 \pmod{M} \text{ and} \quad (1)$$
$$t = v_1 - v_2 \pmod{M} \quad (2)$$

The Verifier then picks a random value $c \in \{1, 2\}$; if $c = 1$ he asks the Prover to open $\text{COM}(u_1)$ and $\text{COM}(u_2)$. Otherwise, the Prover must open $\text{COM}(v_1)$ and $\text{COM}(v_2)$. The Verifier correspondingly checks (1) or (2). The Prover fails if the checked equation fails.

Fact. If $\text{VAL}(X) \neq \text{VAL}(Y)$, then the Prover fails with probability at least 1/2.

It is very important that a given split-value commitment should not participate in more than one such proof. Otherwise both its components may be revealed, thus revealing the value represented.

Generalization to tuples We use a generalization of the above proof method, wherein X is replaced by a tuple X_1, X_2, X_3 such that $\text{VAL}(X) = \text{VAL}(X_1) + \text{VAL}(X_2) + \text{VAL}(X_3)$, and similarly for Y and Y_1, Y_2, Y_3. (This is for our default three-row proof server arrangement; more values are used if there are more rows.)

A proof of the equality that

$$\text{VAL}(X_1) + \text{VAL}(X_2) + \text{VAL}(X_3) = \\ \text{VAL}(Y_1) + \text{VAL}(Y_2) + \text{VAL}(Y_3)$$

proceeds just as before, except that opening the first component of X is replaced by opening the first component of each of X_1, X_2, and X_3, and opening the second component of X is replaced by opening the second component of X_1, X_2, and X_3; similarly for Y. Again a value t such that $X_1 + X_2 + X_3 = Y_1 + Y_2 + Y_3 + (-t, t)$ is posted by the Prover.

The basic fact (that a cheating Prover is unmasked with probability at least 1/2) remains true.

F. Proving Equality of Arrays of Vote Values

We further generalize such proofs of equality to proofs of equality for lists of length n of commitments to vote values.

In our mechanism votes are represented by triplets $T = (X, Y, Z)$ and committed to as

$$\text{COMT}(T) = (\text{COMSV}(X), \text{COMSV}(Y), \text{COMSV}(Z)) \ .$$

By definition,

$$\text{VAL}(T) = (\text{VAL}(X) + \text{VAL}(Y) + \text{VAL}(Z)) \bmod M \ .$$

Assume that a Prover has posted in a SBB two arrays of commitments to triplet representations of values:

$$\text{COMT}(T_1), \text{COMT}(T_2), ..., \text{COMT}(T_n)$$
$$\text{COMT}(T'_1), \text{COMT}(T'_2), ..., \text{COMT}(T'_n).$$

The Prover claims that $\text{VAL}(T_j) = \text{VAL}(T'_j)$ for $1 \leq j \leq n$.

To post a proof of correctness on the SBB, the Prover posts the values $t_1, \ldots, t_n$ required for proving the claimed equalities.

Afterwards, employing appropriate randomness (see Section VIII), n random independent values $c_j \in \{1, 2\}$, $1 \leq j \leq n$, are computed and posted by the Verifier.

Now the Prover constructs and posts a corresponding proof for each claimed equality $\text{VAL}(T_j) = \text{VAL}(T'_j)$, $1 \leq j \leq n$, which can be verified as shown above.

Theorem 1: If more than k of the claimed n value equalities are false then the probability of acceptance of the claim is at most $(1/2)^k$.

Proof: If for an index j, $\text{VAL}(T_j) \neq \text{VAL}(T'_j)$, then the probability of the inequality not being uncovered is at most $1/2$. Because of the independent random choice of the challenges $c_j \in \{1,2\}$, $1 \leq j \leq n$, the probability of not uncovering at least one of the k inequalities is most $(1/2)^k$. ■

This completes our review of the mathematical preliminaries needed for our methods.

III. SETUP

We now begin our more detailed description of our method, beginning in this Section with the Setup phase.

See Figure 1 for a graphic depiction of the overall method.

A. Choice of M

We assume that there is only one race in the election. (The entire method can be replicated for additional races.)

A value of M is chosen so that each possible choice a voter can make in this race (including write-in votes, if allowed), may be uniquely represented as a value w, where $0 \leq w < M$.

If the extensions of Section X are used that use Shamir's secret-sharing method [12] to handle failing servers, then M should be prime.

B. Tablets and Servers

The voter casts her vote in a voting booth by use of a Tablet. Multiple voters may vote on a single Tablet. A representation of the vote is transferred as described below from the Tablet to various to election servers.

Some of the servers are "mix servers" that anonymize the vote by removing identifying information and shuffling them according to a secret permutation.

The mix servers also act collectively as a "proof server" (PS) that prepares a publicly verifiable proof of the correctness of the election results.

The proof of correctness will be publicly posted by the PS on an electronic Secure Bulletin Board (SBB) accessible to voters, parties involved in the election, and the general public.

In this note, to achieve high assurance of voter privacy the PS consists of nine independent devices $P_{1,j}$, $P_{2,j}$, $P_{3,j}$, $j = 1, 2, 3$ (considered as three rows of three devices each).

It will be demonstrated that as long as no more than two devices may leak out information, privacy of voters is protected. Generalizations for other parameterizations will be described later. The obvious generalization to the case of ℓ leaky devices employs $(\ell + 1)^2$ devices.

C. Secure Channels

We assume that suitable arrangements are made for secure channels between the tablets and the election servers.

For example, one may use three pairs (e_j, d_j) of a public-key encryption method (PKE), for $j = 1, 2, 3$. Here e_j is a public encryption key, and d_j is the corresponding secret decryption key. Every voter Tablet has all public encryption keys e_j, $j = 1, 2, 3$. Every $P_{j,1}$ has the secret decryption key d_j .

However, in such an implementation, the public-key decryption may become an overall computational bottleneck. Thus, we recommend using a simple hybrid encryption method to set up private symmetric keys, employing only one PKE encryption key per tablet and the corresponding PKE decryption key by the Proof Server per Tablet. This reduces the overall PKE decryption time significantly.

Each proof server also has secure channels to every other proof server in the same row or column.

IV. VOTE CASTING

We assume that the Voter's Tablet is given (or creates) a unique ballot id bid for each voter.

The Voter's Tablet takes the Voter's V vote w, where $0 \leq w < M$, and randomly represents w as a triple (x, y, z) such that

$$w = (x + y + z) \bmod M \ .$$

It then creates random split-value representations of x, y, and z as $X = (u_1, v_1)$, $Y = (u_2, v_2)$, and $Z = (u_3, v_3)$. Tablet chooses for X random keys K_1, K_2 and sends to $P_{1,1}$ the ballot representation:

$$bid, \text{COMSV}(X), PKE(e_1, (K_1, u_1) \parallel (K_2, v_1)))$$

where

$$\text{COMSV}(X) = (\text{COM}(K_1, u_1), \text{COM}(K_2, v_1)).$$

Similarly a message containing $\text{COMSV}(Y)$ is sent to $P_{2,1}$ and a message containing $\text{COMSV}(Z)$ is sent to $P_{3,1}$, using different pairs of random keys for each commitment, and using e_2 for encryption for $P_{2,1}$, and e_3 for encryption for $P_{3,1}$. In this way the Tablet sends to the first device in each row a portion of a distributed representation of vote w (each portion being a commitment to a split-value representation of a component of w, where the components add up modulo M to w).

The use of the above split value representations X, Y, Z, for x, y, z, is one of the main innovations of this paper. It is used in creating the publicly verifiable proof of correctness of the submitted votes and of the tally of the election.

As part vote-casting, the voter may participate in a "*cast-or-challenge*" protocol see Benaloh [13] to verify that her Tablet has faithfully represented her choice(s). We omit details.

V. POSTING OF VOTE RECORDS

In E2EVV each ballot is encrypted and posted on a secure public append-only Bulletin Board (SBB) [14].

All encrypted ballot information received from Tablets is publicly posted on the public Secure Bulletin Board, so that voters may confirm their correct reception. To simplify procedures, a voter is given on her receipt the ballot id bid of her vote, and the postings may be in order of ballot id.

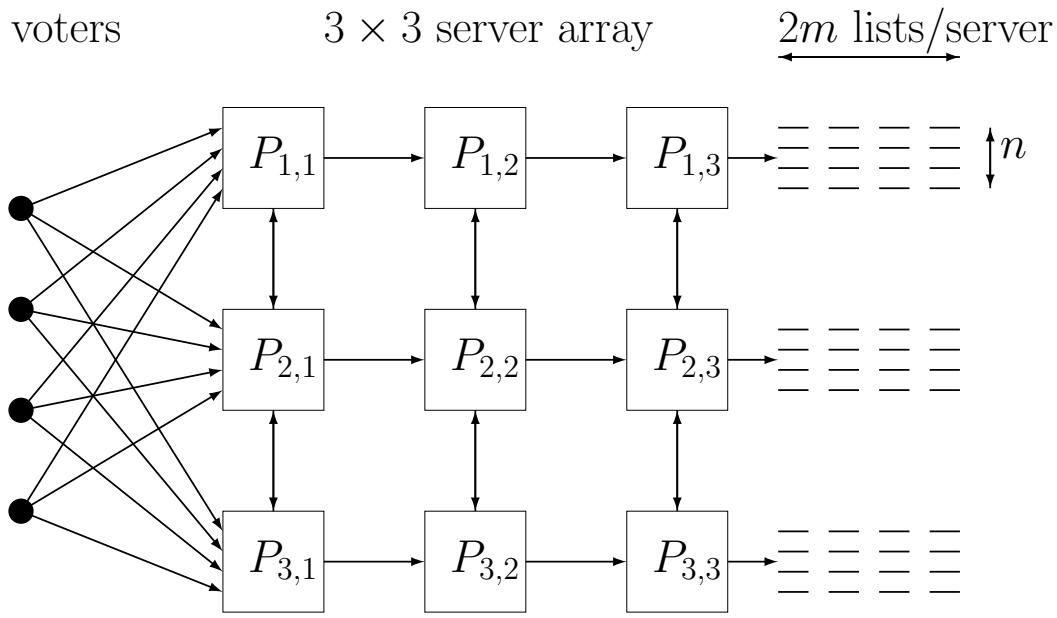

Fig. 1. An illustration of the method for $n = 4$ voters. Information flows from left to right. Each voter sends an encrypted share of his vote to each of the servers in the first column; these encrypted shares are also posted on a secure bulletin board. Each column obfuscates and reshuffles its data (each server in a column using the same random permutation) before sending it on to the next column. The information flow from the first column to the output is repeated $2m = 4$ times (with different randomness used each time). A "cut-and-choose" method randomly selects m columns of output lists to be re-routed back to be compared for consistency with the input. The other m columns are opened, checked for consistency, and posted to reveal the election outcome. A proof of the correctness of the election outcome is then prepared and posted, as described in the text.

VI. Verification of Postings

The voter was given a paper receipt from the Tablet giving a hash value of what should be posted, to enable simple verification of correct inclusion of her ballot.

Every voter can then verify that the cipher text of her ballot has been properly posted, this without her being able to convince anybody what her actual vote was. (The voter does not know how to open any commitments.)

VII. Mixing

The implementation of a fast verifiable mix-net, described in this Section, is one of the main contributions of this paper.

We emphasize that the required computational primitives are just additions mod M of integers of value at most M, and concealment of integers u of size at most M as $\textsc{Com}(K, u)$ by a fast commitment function $\textsc{Com}(\cdot, \cdot)$. These primitives are done on individual proof servers $P_{i,j}$, not in a multi-party fashion, and are executable on ordinary laptop or desktop computers at the rate of millions of operations per second.

Our mix-net, consisting of $P_{1,j}$, $P_{2,j}$, $P_{3,j}$, $j = 1, 2, 3$, creates and publicly posts $2m$ arrays of length n, each of which is a secret random permutation of the (encrypted) votes $w_1, \ldots, w_n$.

Why are $2m$ permuted lists produced, instead of a single one, as is usual for mix-nets? The answer is that we need half of them to check against the posted inputs, and half to produce the desired election outcome. Because no split-value commitment can be compared for equality more than once, we need multiple copies to make this approach work out.

The actual number $2m$ used depends on the degree of correctness assurance the system is designed to achieve; Theorem 3 in Section IX shows that $2m = 24$ provides high assurance.

Decryption. To begin, $P_{1,1}$, $P_{2,1}$, $P_{3,1}$, each using its private decryption key, opens its received commitments.

The Proof Server PS device $P_{1,1}$ has the secret decryption key d_1. It decrypts for each Ballot component X the $PKE(e_j, (K_1, u_1) \parallel (K_2, v_1))$ part. The revealed values (K_1, u_1), (K_2, v_1) are checked as the correct opening of $\textsc{ComSV}(X)$, enabling $P_{1,1}$ computes $\textsc{Val}(X) = x = (u_1 + v_1) \bmod M$.

Now $P_{1,1}$ has the sequence of X-components $x_1, \ldots, x_n$ of the n vote values $w_1, w_2, \ldots, w_n$.

Similarly $P_{2,1}$ computes $y_1, \ldots, y_n$ and $P_{3,1}$ computes $z_1, \ldots, z_n$. Here the first vote is $w_1 = (x_1 + y_1 + z_1) \bmod M$.

Even though first-column devices now have components in the clear, the distribution of a vote value w as the sum mod M of x, y and z and sending each component to a different $P_{j,1}$, $j = 1, 2, 3$, ensures that if at most two devices are leaky, the vote remains secret.

First column obfuscates and shuffles. To create an output array consisting of the n vote values concealed and randomly permuted, the servers $P_{1,1}$, $P_{2,1}$, $P_{3,1}$ comprising the first column of the PS first *obfuscate* and then *shuffle* the list of n vote values, before passing them on to the next column.

Obfuscating: The first-column proof servers create an *obfuscation* of the list of n vote values.

Definition 5: We say that $S'_1 = (x'_1, y'_1, z'_1)$ is an *obfuscated form* of $S_1 = (x_1, y_1, z_1)$ if

$$x'_1 + y'_1 + z'_1 = x_1 + y_1 + z_1 \pmod M ,$$

that is, if S'_1 and S_1 represent the same value.

The method for $P_{1,1}$, $P_{2,1}$, $P_{3,1}$ to obfuscate the first vote value (represented as a triple $S_1 = (x_1, y_1, z_1)$ in the three servers) is to chose three random values p_1, q_1, r_1 in the range 0 to $M - 1$, subject to $(p_1 + q_1 + r_1) \bmod M = 0$ and to compute $x'_1 = (p_1 + x_1) \bmod M$ by $P_{1,1}$, etc. Similar obfuscation is done on the components of the other $n-1$ votes w_2 , ..., w_n using different randomly chosen triplets p_j, q_j, r_j for each obfuscation.

Shuffling: $P_{1,1}$ has now the values x'_1 , ..., x'_n, $P_{2,1}$ has the values y'_1, ..., y'_n and similarly for $P_{3,1}$. Now $P_{1,1}$, $P_{2,1}$, $P_{3,1}$ together choose a random permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$.

Send data to next column. Then $P_{1,1}$ transmits the array $x'_{\pi(1)}, \ldots, x'_{\pi(n)}$, to $P_{1,2}$. Similarly $P_{2,1}$ transmits the array $y'_{\pi(1)}, \ldots, y'_{\pi(n)}$, to $P_{2,2}$ and $P_{3,1}$ transmits the array $z'_{\pi(1)}, \ldots, z'_{\pi(n)}$, to $P_{3,2}$.

Second column obfuscates and shuffles. The second column $P_{1,2}$, $P_{2,2}$, $P_{3,2}$, repeats the same process of obfuscation and shuffling, sending the obfuscated-shuffled array to the third column $P_{1,3}$, $P_{2,3}$, $P_{3,3}$.

Last column obfuscates and shuffles. Finally, $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ again obfuscate and shuffle so that $P_{1,3}$ has the array $(x'''_{\sigma(1)}, \ldots, x'''_{\sigma(n)})$. Similarly for $P_{2,3}$ and the array $(y'''_{\sigma(1)}, \ldots, y'''_{\sigma(n)})$ and for $P_{3,3}$. Here σ denotes the permutation of the original order of the ballots into the present arrays.

Posted of lists of votes. Server $P_{1,3}$ creates and posts on the SBB commitments $(\text{COMSV}(X'''_{\sigma(1)}), \ldots, \text{COMSV}(X'''_{\sigma(n)}))$ to split-value representations of the components $(x'''_{\sigma(1)}, \ldots, x'''_{\sigma(n)})$. Similarly, $P_{2,3}$ creates and posts $(\text{COMSV}(Y'''_{\sigma(1)}), \ldots, \text{COMSV}(Y'''_{\sigma(n)}))$ and so does $P_{3,3}$.

This total posted array of $3n$ commitments is one of the $2m$ lists produced by the mix-net; the whole process is repeated $2m$ times to obtain the set of all $2m$ lists.

Remark. Note that in our method of shuffling, unlike in mix-nets, components of votes are not shuffled amongst rows going from one column to the next. They rather stay within the same row obfuscated and in shuffled order.

Theorem 2: **(Maintenance of Voter Privacy.)** As long as no more than two of the nine servers $P_{i,j}$ leak out unintended data, there are at least one row and one column in the 3×3 array of servers $P_{i,j}$ that do not contain an improper server. This, combined with the obfuscation and shuffling from one column of servers to the next and the final obfuscation and shuffling by the third column $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ of servers, results in complete secrecy of votes by individual voters, even if the above output arrays of $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ are made public and two servers of the PS leak out all their data.

We shall prove this theorem following the next remark. It is assumed that the communications between any two mix servers is secure.

Remark. If computations were properly done, then $(x'''_{\sigma(1)} + y'''_{\sigma(1)} + z'''_{\sigma(1)}) \bmod M = w_{\sigma(1)}$, etc. That is, from the output arrays of $P_{1,3}$, $P_{2,3}$, $P_{3,3}$, the votes $w_1, \ldots, w_n$ can be directly read off (in the order σ).

Proof: In first phase of obfuscation and shuffling going from the first column $P_{1,1}$, $P_{2,1}$, $P_{3,1}$ to the second column $P_{1,2}$, $P_{2,2}$, $P_{3,2}$, obfuscating a typical $S_1 = (x_1, y_1, z_1)$ into $S'_1 = (x'_1, y'_1, z'_1)$ by use of p_1, q_1, r_1. Note that $P_{1,1}$ keeps x_1 and x'_1 in its own memory. Similarly for $P_{2,1}$, $P_{3,1}$ and their components of S_1 and S'_1.

This implies that even though p_1, q_1, r_1 are known to all three of $P_{1,1}$, $P_{2,1}$, $P_{3,1}$, nothing is revealed about components of votes stored in non-leaky devices.

The same holds about obfuscation and shuffling going from the second column $P_{1,2}$, $P_{2,2}$, $P_{3,2}$ to the third column $P_{1,3}$, $P_{2,3}$, $P_{3,3}$.

Once the third column $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ is reached either it or one of the two preceding columns do not contain any leaky device. Thus third-column outputs protect voter privacy. ■

VIII. RANDOM CHALLENGE

We note that the proof servers have a need for random values of two distinct flavors:

- *Internal randomness.* The PS needs random values to create random split-value representations random permutations, etc. These values should be unpredictable to outsiders, but need not be unpredictable to the proof servers themselves. For these purposes, the proof servers may use what we call "internal randomness": truly random sources available only to each proof server.
- *External randomness (for challenges).* The proofs of correctness need random challenges (e.g. for the cut-and-choose of m lists out of $2m$, or for the proofs of equality of split-value commitments) that are unpredictable even to the proof servers (as they may be malicious). These random challenges may be obtained in either of two ways: in the Fiat-Shamir style [15] as the hash of the current SBB, or from a random external source (e.g. a dice-rolling ceremony). The former approach has the advantage that the (pseudo-)random values obtained by hashing the SBB

may be verified by anyone, but has the disadvantage that an evil proof server may try many values to be posted on the SBB until the SBB hash is to its liking. Thus, the value of $2m$ may need to be significantly larger if the Fiat-Shamir method is used. Our analyses assume that the challenges are derived from a truly random external source; appropriate adjustments to the value of $2m$ should be applied if the Fiat-Shamir method is used.

IX. Proof of Correctness

The election outcome and associated tally, as well as a proof of correctness of the announced results, are also posted on the SBB, and can be verified by anyone.

Posting of split-value representations of mix-net outputs. The device $P_{1,3}$ creates random split-value vector representations $X'''_{\sigma(i)}$ for $x_{\sigma(i)}$, $1 \leq i \leq n$, and commitments $\text{ComSV}(X'''_{\sigma(i)})$ for $1 \leq i \leq n$. Similarly for $P_{2,3}$ with the $y'''_{\sigma(i)}$, and $P_{3,3}$ with the $z'''_{\sigma(i)}$.

Using the notation of Section II-F $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ together prepare and publicly post for $1 \leq i \leq n$:

$$\text{ComT}(T_{\sigma(i)}) = (\text{ComSV}(X'''_{\sigma(i)}), \text{ComSV}(Y'''_{\sigma(i)}), \text{ComSV}(Z'''_{\sigma(i)})) \quad (3)$$

This process of obfuscation, shuffling and posting an array of the form (3) is repeated by the PS $2m$ times, where $2m$ is chosen to yield the desired assurance of correctness. Each of these posted arrays is of course created by use of a different permutation of $\{1, \ldots, n\}$.

Cut and Choose. By use of randomness extracted from all posted data together with an independent random seed, m of the posted lists (3) are randomly chosen for a proof of value-consistency with the posted concealed votes (see Introduction).

Proving consistency with cast votes: Each of these m chosen arrays (3) is rearranged by the Proof Server in the order of of bids, hence in the order of the submitted-posted concealed ballots. This is done by backtracking for the chosen arrays, the permutations used by each column.

The permutations σ for the m chosen arrays are posted, as are the values $(t_i, -t_i)$ used in the proof. For brevity we omit the simple details of how $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ compute and post the pairs $(t_i, -t_i)$, $1 \leq i \leq n$.

Now the randomness is used to open one coordinate in each of the commitments in the posted concealed ballots and the corresponding commitment in each of the m rearranged arrays (3) and prove equality of values by the method of Section II-F.

By Theorem 1, if even one of these m lists differs from the ballot list by more than k values then the probability of acceptance is at most $(1/2)^k$.

Posting and verification of the election outcome: Now all the other m permuted lists are opened and the values are revealed. Only if all opened lists are permutations of the same values is the proof of correctness accepted. The election outcome is then the result of applying the appropriate tallying function or election outcome determination function to any of the opened lists. (We assume that the election outcome does not depend on the order of the ballots.)

Level of assurance provided. We now analyze the level of assurance provided by the posted proof.

Definition 6: Call a permuted array of n values *k-good* if when re-arranged in the order of the originally concealed n ballots posted by the tablets on the PS, it differs from the concealed ballot values in fewer than k locations.

Theorem 3: The probability that the opened arrays (3) are permutations of the same values but they are not k-good, i.e. the probability of accepting an announced tally result differing from the correct tally by more than k vote values is at most

$$1/C(2m, m) + (1/2)^k \approx \sqrt{3.14m}/2^{2m} + (1/2)^k,$$

where $C(2m, m)$ is the binomial coefficient "2m choose m".

Proof: Call H the set of m lists of n ballots revealed by $P_{1,3}$, $P_{2,3}$, $P_{3,3}$. Assume that one, and therefore all, of these ballot lists is not k-good. The probability that in the cut and choose the set H is chosen to be opened is $1/C(2m, m)$. If H is not chosen then the proof of value consistency is conducted on at least one array of n concealed ballots which is not k-good. The probability of this happening and proof of correctness being accepted is at most $(1 - 1/C(2m, m))(1/2)^k$. ■

For the case of no more than 20 wrong votes we use $2m = 24$ and the probability of accepting a proof of correctness while there are more than 20 discrepancies is less than $1.38/2^{20}$.

X. Countering Denial of Service Attacks (Device Failure)

It is relatively straightforward, using well-known secret-sharing methods, to provide increased robustness against the possibility that one or more of the proof server devices may fail. As noted in the introduction, Cramer et al. [6] have also used secret sharing to improve robustness of a voting system. (Their paper employs homomorphic encryption and unlike the present work reveals only the final value of the vote count.)

These methods allow construction of systems satisfying specified robustness requirements in addition to voter privacy protection. When failures may occur, then obfuscation is done by the method of proactive secret sharing (see [16]), rather than the method described in the example of the previous sections. Because Shamir secret sharing is used, M is chosen to be a prime number, say $M = 1009$.

For example, suppose we wish to protect against one device failure and one leaky device; we'll use a PS with four rows and two columns. The votes are $(4, 3)$-shared by in the finite field F_M by the voter Tablet and the shares of each vote are securely sent to four devices $P_{1,1}, \ldots, P_{4,1}$ comprising the first column of the PS. With $(4, 3)$-secret-sharing each value is split into four shares, such that any three (but not any two) suffice to reconstruct the value.

Every first-column Proof Server device $P_{j,1}$ $(4, 3)$-shares the value 0 among the 4 devices in the first column. Every $P_{j,1}$ adds the received shares of 0 to its input share. (This is

done separately for each vote.) The first column devices shuffle the obfuscated quadruples and every $P_{j,1}$ sends its obfuscated share to $P_{j,2}$. The second column of the PS obfuscates and shuffles, produces the results as output.

Now the servers $P_{1,2}$, $P_{2,2}$, $P_{3,2}$, $P_{4,2}$ of the second column each prepares an array of commitments to split value representations of its permuted array of shares of the n vote values w_1 , w_2 , $\ldots w_n$. These commitments are posted on the SBB. This whole process is repeated $2m$ times. Then the m permuted arrays of the (4, 3) shares of the n vote values w_1 , w_2 , $\ldots$, w_n , are posted as in Sections VII–IX.

In general, if at most f devices may fail (where $f > 0$) and at most ℓ may be leaky, then PS may have r rows and c columns, where $r \geq f + \ell + 2$ (to protect votes from leaking), use an $(r, \ell + 2)$ secret-sharing method, and choose $c \geq \ell + 1$ (to protect the shuffles). If $f = 0$, then the number of rows and the number of columns need only be $\ell + 1$, as in the example of the previous sections.

For additional protection against possibly malicious servers, one may for example employ Trusted Platform Module (TPM) technology. Work in progress (to appear) presents additional methods for countering malicious servers who attempt to actively disrupt the protocol. Of course, when paper ballots are available (as with Scantegrity or Star-Vote), one can always recover the correct election outcome by counting them.

XI. Practical Aspects

We consider some practical aspects of the proposed method, such as time and storage requirements.

Assume that the number n of ballots is 10^6, the number of tablets is 10^4, and that we use $2m = 24$. The following numbers are for a typical desktop computer or laptop, which can execute 200 private-key operations (e.g. RSA 2048-bit) per second or 8 million commitments (AES operations) per second. Assume that PS has $r = 3$ rows and $c = 3$ columns.

Time to decrypt votes from tablets: This requires 10^4 private-key operations (using a hybrid method) per first-column PS device—about 50 seconds. It also requires about 10^6 openings of pairs of commitments—under a second. The 50 seconds for the private-key operations is the major component of the running time. The last-column PS devices must prepare 24 arrays of length n with 6 commitments per vote—about 18 seconds (six seconds if the last-column processors do this in parallel). The time to create the random permutations is negligible.

Size of proof: If each commitment $\textsc{Com}(u)$ is assumed to require 30 bytes, then the overall size of the proof is about $25 \times 2 \times 3 \times 30 \times 10^6$ bytes (4.5GB), about the size of a movie; the proof can be downloaded on an typical internet connection in a few minutes at most, and checked in a couple of minutes on a typical laptop.

Code: A 2800-line python program for running simulated elections was written and tested; in experiments it performs flawlessly and rapidly. (See https://github.com/ron-rivest/split-value-voting .)

XII. Conclusion

The methods presented here provide new ways for implementing verifiable mix-nets and thus end-to-end verifiable voting. The new methods are particularly efficient since they do not require any modular exponentiations or public-key operations. We believe that the efficiency and generality of this solution render it practical for actual deployment in elections.

Acknowledgment

We thank Tal Rabin for advice on proactive secret sharing. The second author gratefully acknowledges support from his Vannevar Bush Professorship. We thank the anonymous EVOTE reviewers for numerous constructive suggestions.

References

[1] B. Adida and R. L. Rivest, "Scratch & vote: self-contained paper-based cryptographic voting," in *Proceedings of the 5th ACM workshop on privacy in electronic society*, R. Dingledine and T. Yu, Eds. ACM, 2006, pp. 29–39.

[2] R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P. L. Vora, "Scantegrity II municipal election at Takoma Park: The first E2E binding governmental election with ballot privacy," in *Proceedings USENIX Security 2010*, I. Goldberg, Ed. USENIX, August 11-13, 2010.

[3] A. Essex, J. Clark, U. Hengartner, and C. Adams, "Eperio: Mitigating technical complexity in cryptographic election verification," in *Proceedings of the 2010 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, ser. EVT/WOTE'10. Berkeley, CA, USA: USENIX, 2010, pp. 1–16.

[4] H. Jonker, S. Mauw, and J. Pang, "Privacy and verifiability in voting systems: Methods, developments and trends," Cryptology ePrint Archive, Report 2013/615, 2013.

[5] S. Popoveniuc, J. Kelsey, A. Regenscheid, and P. Vora, "Performance requirements for end-to-end verifiable elections," in *Proceedings of the 2010 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, ser. EVT/WOTE'10. Berkeley, CA, USA: USENIX, 2010, pp. 1–16.

[6] R. J. F. Cramer, M. Franklin, L. A. M. Schoenmakers, and M. Yung, "Multi-authority secret-ballot elections with linear work," Centrum voor Wiskunde en Informatica, Tech. Rep. CS-R9571, 1995.

[7] J. Benaloh, M. Byrne, P. Kortum, N. McBurnett, O. Pereira, P. B. Stark, and D. S. Wallach, "STAR-vote: A secure, transparent, auditable, and reliable voting system," *arXiv preprint arXiv:1211.1904*, 2012.

[8] S. Micali and M. O. Rabin, "Cryptography miracles, secure auctions, matching problem verification," *CACM*, vol. 57, no. 2, pp. 85–93, February 2014.

[9] 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*. IEEE, 2007, pp. 63–76.

[10] M. O. Rabin and R. L. Rivest, "Practical end-to-end verifiable voting via split-value representations and randomized partial checking," April 3, 2014, CalTech/MIT Voting Technology Project Working Paper 122.

[11] ——, "Practical provably correct voter privacy protecting end to end voting employing multiparty computations and split value representations of votes," May 12, 2014, CalTech/MIT Voting Technology Project Working Paper 124.

[12] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[13] J. Benaloh, "Ballot casting assurance via voter-initiated poll station auditing," in *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*. USENIX, 2007, p. 14.

[14] C. Cullane and S. Schneider, "Peered bulletin board for robust use in verifiable voting systems," arXiv.org/abs/1401.4151, Jan. 16, 2014.

[15] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. Crypto '86*, ser. Lecture Notes in Computer Science, vol. 263. Springer, 1986, pp. 186–194.

[16] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *Proc. 10th ACM Symp. Princ. Distr. Comp.* ACM, 1991, pp. 51–61.