Practical Provably Correct Voter Privacy Protecting End to End Voting Employing Multiparty Computations and Split Value Representations of Votes

Michael O. Rabin
Columbia University SEAS
Harvard University SEAS

morabin@gmail.com

Ronald L. Rivest
MIT CSAIL

rivest@mit.edu

May 12, 2014

Abstract. Continuing the work of Rabin and Rivest [1] we present another simple and fast method for conducting end to end voting and allowing public verification of correctness of the announced vote tallying results. This method was referred to in [1] as the SV/VCP method. In the present note voter privacy protection is achieved by use of a simple form of Multi Party Computations (MPC). 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.

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 primitives. 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.

Introduction and Infrastructure

The method of vote tallying and correctness proving described in this note can be used in conjunction with any one of the vote casting, paper copy of vote and paper receipt for voter described in [1]; see that paper for additional context.

Votes and values used in the system are described by integers x < M where, say, M = 100.

In the following, additions and subtractions of values are performed mod M; we illustrate with M = 100. Thus 38 + 80 = 18; (– 17) = 83.

The voter casts her vote in a Voting Booth by use of a Tablet. The vote is transferred to a computer in the Voting Station. At the end of election-day Voting stations transfer the votes to a Tally Server (TS) at the Vote Tallying Center (VTC).

The VTC also has a Proof Server (PS) which will prepare a publicly verifiable proof of the correctness of the election results announced by the TS. 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* leak out information, privacy of voters is protected. Generalizations for other parameterizations will be described later.

The proof of correctness assures the correctness of the announced election results no matter how the devices acted.

The system uses 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 secret decryption key.

It also uses a commitment function COM(K, u) employing a (randomly chosen) key K to commit to value u. COM can be implemented, say, by use of AES with 256 bit keys. It is assumed that COM is *computationally hiding*: given the value C = COM(K, u), it is infeasible to gain any information about u. It also assumed that it is computationally infeasible to find two pairs (K, u) ≠ (K', u') such that COM(K, u) = COM(K', u'). This renders the commitment by COM to be *computationally binding*. A commitment C can be opened in only one way.

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 .

The Voter's Tablet takes the Voter's V vote w and randomly represents w as a triple (x,y,z) such that

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

It then creates random vector representations of x, y, and z as

$$X = (u_1 , v_1) \quad \text{where } Val(X) = (u_1 + v_1) \bmod M = x;$$

$$Y = (u_2, v_2) \quad \text{where } Val(Y) = (u_2 + v_2) \bmod M = y;$$

$$Z = (u_3 , v_3) \quad \text{where } Val(Z) = (u_3 + v_3) \bmod M = z.$$

Tablet chooses for X random keys K_1 , K_2 and sends to $P_{1,1}$ via the Voting Station the ballot representation consisting of

Ballot id (BLTid)

$COMSV(X) = (COM(K_1, u_1), COM(K_2 , v_1))$

$PKE(e_1 , K_1 \,||\, K_2))$.

Similar messages are sent to $P_{2,1}$, $P_{3,1}$ using different pairs of random keys for each commitment, and using e_2 for encrypting for $P_{2,1}$, and e_3 for encryption for $P_{3,1}$.

In this way the Tablet sends a portion of a distributed representation of vote w to the first device in each row (each portion being a commitment to a split-value representation of a share of w, where the shares add up to w modulo M).

We assume that there are n voters and that the system uses some convention for providing each ballot with a unique ballot id, BLTid.

All 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 the ballot id BLTid of her vote, and the

postings may be in order of ballot id. The voter may have a receipt from the Tablet giving the hash of what should be posted, to simplify comparisons.

Now each $P_{1,1}$, $P_{2,1}$, $P_{3,1}$, each using its private decryption key, internally opens its received commitments.

Remark. The slowest computation in the above is PKE decryption. A simple hybrid encryption method to set up private symmetric keys employing only one PKE decryption by the Proof Server per Tablet may be used to reduce the overall PKE decryption time significantly.

Overall Plan. The Multi-Party Proof Server $P_{1,j}$, $P_{2,j}$, $P_{3,j}$, j = 1, 2 ,3 will create and publicly post m arrays of length n each of which is, verifiably, a secret random permutation of the votes w_1 , …, w_n *in the clear*.

This will be achieved by transforming the n concealed cast votes into 2m arrays each of length n and each consisting of a different random permutation of the votes re-concealed by the Proof Server. The number 2m used depends on the degree of correctness assurance the system is designed to achieve. It will be seen, following Theorem 2 in Section 5, that for example 2m=24 provides very high assurance.

Cut and Choose. m of the 2m arrays will be randomly selected, posted in the original order of the cast votes and proven to conceal the same vote values as the array of cast votes.

Each of the remaining m arrays will be opened in its permuted order to reveal a permutation of the votes w_1 , …, w_n .

The required computations by the Proof Server are additions mod M of integers of size at most M, and concealment of integers u of size at most M as COM(K, u) by any fast commitment function COM(,). These computations are done on the individual $P_{i,j}$ and *not* in a multi-party fashion. They are executable on ordinary desk top computers at the rate of millions of operations per second.

1. Split Value Representations of Values and Proofs of Equality

The method of Sections 1-2 follow [Rabin et-al, 2] , [Micali, Rabin, 3].

Let $0 \le x < M$ be a value. A *random split value representation* of x is a vector X = (u, v) where u is randomly chosen from [0, M – 1] and v = (x – u) mod M. Thus Val(X) = (u +v) mod M = x.

A commitment COMSV(X) to a vector X = (u, v) is a pair of commitments, one to each component: COMSV(X) = (COM(K_1, u), COM(K_2, v)). Note that COMSV(X) denotes commitment to a split-value vector representation of a value x, $0 \le x < M$, while COM(K,u) is a commitment to a value u, $0 \le u < M$.

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 in a proof process, then no information about the value x is revealed.

We now begin to consider triplets (or later, tuples) of split-value representations.

Let X= (u_1 ,v_1), Y = (u_2 ,v_2), Z = (u_3 ,v). Let X'= (u'_1 ,v'_1),Y' = (u'_2 ,v'_2), Z' = (u'_3 , v'_3).

Clearly

$$Val(X) + Val(Y) + Val(Z) = Val(X') + Val(Y') + Val(Z') \quad (1)$$

if and only if there exists a value t such that

$$X + Y + Z = X' + Y' + Z' + (t, -t). \quad (2)$$

Assume that a Prover has posted on a SBB six commitments, COMSV(X), COMSV(Y), …, COMSV(Z'), and claims that (1) holds. He can post a proof of this claim as follows:

1. Prover posts and signs a value t.
2. By means of independent randomness (see Section 3), a random $c \in \{1, 2\}$ is created and posted on the SBB.
3. If c = 1, Prover opens the first commitments in COMSV(X), COMSV(Y), COMSV(Z), COMSV(X'), COMSV(Y'), COMSV(Z'), revealing and posting u_1 , u_2 , u_3 , u'_1 , u'_2 , u'_3 .

4. Similarly, if c = 2, Prover opens the second commitments.
5. A Verifier viewing the posted proof will check that the commitments were correctly opened. If c = 1 the verifier will check and accept only if $(u_1 + u_2 + u_3) \bmod M = (u'_1 + u'_2 + u'_3 + t) \bmod M$. Similarly, if c = 2, now using –t.

Lemma. If (1) is false then the probability that the posted proof will be accepted is ≤ ½.

Proof. If the claim (1) is false then at least one of the equations mentioned in the Bullet 5 above is false. The random choice of c will reveal a false equation with probability at least ½. Q.E.D.

2. Proving Equality of Arrays of Vote Values

In our mechanism votes are represented by triplets T = (X, Y, Z) and committed to as COMT(T) = (COMSV(X), COMSV(Y), COMSV(Z)). By definition, Val(T) = (Val(X) + Val(Y) + Val(Z)) mod M.

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

$$COMT(T_1), COMT(T_2), \ldots, COMT(T_n)$$

$$COMT(T'_1), COMT(T'_2), \ldots, COMT(T'_n).$$

The Prover claims that $Val(T_j) = Val(T'_j)$ for $1 \le j \le n$.

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 independent randomness (see Section 3), n random values $c_j \in \{1, 2\}$, $1 \le j \le n$, are computed and posted.

Now a proof for each claimed equality $Val(T_j) = Val(T'_j)$, $1 \le j \le n$, is created, posted and can be verified along the lines in the above 3-5 and Proof Verification given 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 $(½)^k$.

This theorem follows directly from the Lemma.

3. Randomness

The PS will require two types of randomness for preparing its posted proof of correctness of posted tally results.

Internal randomness. As will be seen in Section 5, the PS will prepare and post additional arrays of commitments to vector representations of the votes and random permutations of these arrays. This will require randomness for the creation of the commitments and of the permutations. The PS will use for these tasks internally generated randomness.

Random challenges. Zero Knowledge Proofs (ZKP) involve random challenges. In Interactive ZKPs the random challenges are created by a Verifier who, ideally, uses physical generation of randomness.

We want Non-Interactive proofs of correctness which will be posted for general verification. Thus the random challenges will be created by computing hash functions on posted array data. This idea goes back to Fiat-Shamir. Strictly speaking our proofs are not ZKP, but we do not go into the issue of theoretically classifying them

To prevent the PS from privately creating data and extracting randomness until it can cheat, the hash operation incorporates a seed that a trusted outside committee prepares and hands over to the PS only after the posting. There are various ways of implementing this creation of random challenges.

The above mode of course only creates pseudo-random challenges. In the Theorems that follow we disregard this point and state the results as if truly random challenges were employed.

4. Obfuscating and Shuffling

$P_{1,1}$ has publicly posted, in order of the BLTids, the pairs for the X-components of the n votes.

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 || K_2) part. Using the keys K_1, K_2 the PS opens COMSV(X) and computes Val(X) = x.

Now $P_{1,1}$ has the sequence of X-components of votes: x_1, … , x_n .

Similarly $P_{2,1}$ computes y_1, …, y_n and $P_{3,1}$ computes z_1, …, z_n. Here the vote in $Ballot_1$ = (x_1 + y_1 + z_1) mod M. Voter privacy is protected since at least one of the three devices , say $P_{1,1}$, is not gossipy .

Definition. 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) mod M is equal to (x_1 + y_1 + z_1) mod M --- that is, if S'_1 and S_1 represent the same value.

Obfuscating: The method for $P_{1,1}$, $P_{2,1}$, $P_{3,1}$ to obfuscate S_1 = (x_1, y_1 ,z_1) 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) mod M = 0 and to compute x'_1 = (p_1 + x_1) mod M by $P_{1,1}$, etc.

Shuffling: $P_{1,1}$ has now opened x'_1 , … , x'_n , $P_{2,1}$ has opened y'_1, … , y'_n and similarly for $P_{3,1}$.

Now $P_{1,1}$, $P_{2,1}$, $P_{3,1}$ choose a random permutation $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$.

$P_{1,1}$ securely sends to $P_{1,2}$ the array ($x'_{\pi(1)}$, … , $x'_{\pi(n)}$) of obfuscated values, $P_{2,1}$ securely sends to $P_{2,2}$ the array ($y'_{\pi(1)}$, … , $y'_{\pi(n)}$), similarly for $P_{3,1}$ and $P_{3,2}$.

Next $P_{1,2}$, $P_{2,2}$, $P_{3,2}$ again obfuscate and shuffle their arrays, using a new random permutation π_1 and transfer the result to $P_{1,3}$, $P_{2,3}$, $P_{3,3}$.

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

Maintenance of Voter Privacy: As long as no more than two of the nine devices $P_{i,j}$ leak out unintended data, there are at least one row and one column in the 3×3 array of devices $P_{i,j}$ that do not contain an improper device.

This, combined with the obfuscation and shuffling from one column of devices to the next and the final obfuscation and shuffling by the third column $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ of devices, results in complete secrecy of votes by individual voters. This even if the above output arrays of $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ are made public and two devices of the PS leak out all their data. We shall prove this statement following the next remark. It is assumed about the Proof Server that the communication between any two sub-devices is secure.

Remark. At the same time, if computations were properly done, then $(x'''_{\sigma(1)} + y'''_{\sigma(1)} + z'''_{\sigma(1)}) \bmod M = w_{\sigma(1)}$,etc. I.e. from the output arrays of $P_{1,3}$, $P_{2,3}$, $P_{3,3}$, the σ permutation of the votes w_1 , …, w_n can be directly read off.

Proof of Statement: 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/fragments of proper 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 improper device. So the outputs posted by the third column retain voter privacy.

5. Proof of Correctness

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 $COMSV(X'''_{\sigma(i)})$ 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 2, $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ together prepare and publicly post:

$$COMT(T_{\sigma(i)}) = (COMSV(X'''_{\sigma(i)}), COMSV(Y'''_{\sigma(i)}), COMSV(Z'''_{\sigma(i)})), \ 1 \le i \le n. \quad (3)$$

This process of obfuscation, shuffling and posting an array of the form (3) is repeated by the PS 2m times, where as stated in the Overall Plan, m is chosen according to the desired assurance of correctness. Each of these posted arrays is of course created by use of a different permutation.

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

Each of these m chosen arrays (3) is rearranged by the Proof Server in the order of of BLTids, hence in the order of the submitted-posted concealed ballots.

Now the randomness is used to open 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 2. By Theorem 1, if even one of these m arrays differs from the ballot array by more than k values then the probability of acceptance is $< (1/2)^k$.

For brevity we omit the simple details of how $P_{1,3}$, $P_{2,3}$, $P_{3,3}$ compute and post the pairs $(t_i , - t_i)$, $1 \le i \le n$ used in the proofs of value equality.

Now all the other m permuted arrays are opened and the values are revealed. Only if all opened arrays are permutations of the same values is the proof of correctness accepted.

Call a permuted array of values *k-good* if when re-arranged in the order of posted concealed ballots it differs from the concealed ballot values in fewer than k locations.

Theorem 2. 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

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

where C(2m, m) denotes the binomial coefficient "choose m out of 2m". We postpone the proof to the full paper.

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 $1.38/2^{20}$.

6. 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. 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, rather than the method described in the example of the previous sections.

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 the voter Tablet and the shares of each vote are securely sent to four devices P[1, 1],..., 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 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.

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 ≥ f + l + 2 (to protect votes from leaking), use an (r, l+2) secret-sharing method, and choose c ≥ l +1 (to protect the shuffles). If f = 0, then the number of rows and the number of columns need only be l +1, as in the example of the previous sections.

7. Time Requirements for Creation of Proof, 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, or about 50 seconds. It also requires about 10^6 openings of pairs of commitments, taking 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. The running time required is about 18 seconds (six seconds if the last-column processors do this in parallel). The time to create the random permutations themselves is assumed to be negligible (at most a couple of seconds).

If each commitment COM(u) is assumed to require 30 bytes, then the overall size of the proof is about 25X2X3 X 30X10^6 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.

Acknowledgments

We wish to thank Tal Rabin for advice on proactive secret sharing.

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

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. M. O. Rabin, 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.
2. M. O. 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 Wroclav (2007), pp. 63-76.
3. Micali S., Rabin M.O. Cryptography Miracles, Secure Auctions, Matching Problem Verification. CACM, 57 (2), February 2014. pp. 85-93.