A Linked-List Approach to Cryptographically Secure Elections Using Instant Runoff Voting

Jason Keller[⋆] and Joe Kilian [⋆⋆]

Abstract. Numerous methods have been proposed to conduct cryptographically secure elections. Most of these protocols focus on 1-out-of-n voting schemes. Few protocols have been devised for preferential voting systems, in which voters provide a list of rankings of the candidates, and many of those treat ballots as if they were ballots in a 1-out-of-n voting scheme. We propose a linked-list-based scheme that provides improved privacy over current schemes, hiding voter preferences that should not be revealed. For large lists of candidates we achieve improved asymptotic performance.

Keywords: Electronic Voting, Secure Computation.

1 Introduction

Electronic voting is by far the most mature area of secure computation, with a vast literature (c.f. [17]). Most electronic voting protocols may be viewed as attempts to emulate the following physical metaphor: Voters cast ballots into a large box, at the conclusion of which the box is shaken and opened.

Much work has gone into efficiently and securely approximating this physical paradigm. However, this type of balloting represents merely one way of specifying and aggregating preferences. Numerous ways of aggregating preferences have been proposed, and indeed, are used in major political elections. We consider one such system, known as *instant runoff voting.*

1.1 Instant Runoff Voting

Ballots in a single transferable vote (STV) system are submitted as a list of ordinal preferences. The voters' first choices are counted, and any candidate receiving a certain quota of votes is declared a winner. One such example is the Hare-Clark quota, used in Australian elections:

$$\frac{\text{number of eligible votes}}{\text{number of open seats} + 1} + 1.$$

Votes in excess of the quota are proportionally "returned" to the voters, and applied to the next viable choice on their list. If not enough candidates reach

[⋆] Department of Computer Science, Rutgers University, Piscataway, NJ 08854 USA, `jakeller@eden.rutgers.edu`.

[⋆⋆] PNYLAB, LLC, `joe@pnylab.com`. Supported in part by NSF grant CCF-0728937. Work done in part while at Rutgers University.

their quota in this fashion, the candidate with the fewest number of votes is eliminated, and the process continues until all of the open seats are filled.

Although Arrow's theorem guarantees that there will be some cases for which Hare-Clark voting induces some pathology, it is attractive in practice for its ability to avoid "wasted" votes. One has comparatively less incentive (though some still exists) for strategically not supporting ones favorite candidate because the candidate is either assured to win or very likely to lose. Beyond its aesthetic appeal, the fact that it is in actual use for an important election motivates our attention.

We focus on the special case of Hare-Clark in which there is one open seat, and thus a candidate needs to win a majority of the votes in order to win the election. This is a special case known as Instant Runoff Voting (IRV), which is used in certain local jurisdictions in the United States, including elections in San Francisco [18] and Cambridge, Massachusetts [16]. In this scheme, if a candidate has a majority of votes, then he is elected. Otherwise, the candidate with the fewest votes is eliminated; counters look at the next choices of each ballot that had a vote for the recent loser. We note that for this special case, there is no need to redistribute excess winning votes; however, it remains necessary to eliminate candidates and redistribute these votes.

1.2 Difficulties with the Physical Paradigm

In simple voting an ideal physical ballot box with paper ballots is the "gold standard" against which electronic protocols are judged; indeed, there have been perhaps over-nostalgic calls for its use in practice. However, with instant runoff voting, merely severing the identification between voters and their preference list gives insufficient privacy. Particularly in the case where there is a large number of candidates, a full preference order may conceivably be used to identify a voter and thus leak information far beyond that revealed by the final vote counts, with obvious implications for privacy and coercibility. We note that this problem is not specific to a protocol implementation, but to the nature of what is to be revealed.

As a result, in actual physical elections, one has the choice of either revealing extra information or placing a great deal of trust in the discretion and trustworthiness of the election officials.

The secure multi-party computation paradigm [5, 14] is arguably a superior gold standard than any physical ballot box. One endeavors to simulate trusted election officials, who compute the correct results, but then only reveal that which is supposed to be revealed.

Thus, an intriguing aspect of this type of voting is that a cryptographic protocol may potentially offer a solution that is qualitatively superior to current best practices.

1.3 Related Work

Electronic voting has been a model problem of secure multi-party computation since it was proposed by Chaum [7]. Many protocols have been proposed for single-vote, first-past-the-post-style elections, leveraging homomorphic encryption or mix-network technologies; see, for example, [4, 9, 8, 2, 24, 12, 22, 26, 23]).

Without leaving the realm of simple elections, variations are possible in the security and privacy guarantees of the voting protocol. For example, receipt-free and incoercible voting schemes aim to prevent voter intimidation and vote selling by preventing the voter from being able to prove how they voted; see, for example, [3, 24, 20]. One may view this property as a closer approximation to the physical paradigm, in which the voter cannot prove which ballot is theirs. It should be noted that incoercibility does not follow from the generic multi-party solutions (though incoercibility can be generalized to this setting [6]).

Hevia and Kiwi [15] consider the problem of revealing the winner of the election, but keeping secret the vote tally. As with the problem we consider, the "ideal" physical implementation of voting does not guarantee as strong privacy conditions.

The techniques of "standard" electronic voting also yield solutions to simple preference voting, in which a voter may cast either zero or one votes for each candidate. For example, one can implement a k-candidate preference voting election by k simple 2-candidate elections in which the ith election is used to count votes for the ith candidate.

Protocols for preferential voting schemes, such as IRV, adopt a similar approach. Aditya et al. consider elections for the Australian Senate and House of Representatives [1]. They examine the efficiency of balloting using a naive balloting representation and straight mix-network and homomorphic encryption schemes. For an election with k candidates, their scheme using homomorphic encryptions requires posting a ballot of size $O(k!)$ bits. Their basic mix-network based scheme requires a voter to post a number between 1 and $k!$, corresponding to each set of preferences. In their most efficient scheme, they leverage Australia's voting machine structure, and adapt it to the vector-ballot approach introduced by Kiayias and Yung [21] to handle elections with write-in ballots. Each vote is a 3-vector. The first position contains a homomorphically-encrypted vote, corresponding to one of twenty preset choices. The other two positions are used to represent "write-in" votes (in which voters list their preferences rather than choosing from a preset list). The write-in votes are submitted in blocks with some preset preferential votes to a shrink-and-mix network, while blocks with no write-in votes are tabulated.

1.4 Our contribution

We contribute a new protocol for instant runoff voting that has superior asymptotic performance when there are a large number of candidates and superior privacy guarantees.

The protocols of Aditya et al. may be applied to the case we consider, as it is a special case of their own. We thus compare our protocol to this solution, noting that the comparison is somewhat unfair due to their greater generality.

Although the work required of the voter in the protocol of [1] was small in other respects, the message length scales super-exponentially in the number of candidates. In our solution, the work per ballot construction is roughly quadratic in the number of candidates.

An arguably more important improvement is in our privacy guarantees. The protocol of [1] essentially attempts to mirror the privacy properties of existing systems. Thus, it is acceptable in their framework to reveal individual preference lists once the direct linkage with voters has been eliminated. Hence, this protocol necessarily suffers from the weaknesses of the physical solution with respect to privacy and coercion.

In our protocol, we first reveal the counts of the first-choice preferences each candidate obtained. Whenever a candidate is eliminated and their votes recast (using the next viable preference on the preference list), the new counts are also revealed. However, only these intermediate results are revealed.

One could, of course, strive for even stronger privacy guarantees, such as revealing only the winner(s), or only revealing the order of elimination. One might argue that our protocol necessarily reveals statistics, such as the second-choice preference statistics of those voters whose first choice candidate is the first to be eliminated.

However, revealing such intermediate counts seems to be reasonable and indeed often necessary from a procedural point of view. For most elections, the electorate wishes to know the final counts, not merely the winner. It would likely be considered unreasonable to declare that a candidate is eliminated without giving the actual vote count that was the basis of their elimination.

Furthermore, one can imagine using our protocols on a precinct by precinct basis, with intermediate counts reported to a conventional voting authority that decides who next to eliminate. Such regional counts can be useful in detecting vote fraud. Thus, it may be essential that the tallies from each round be revealed, and that elimination decisions can be made externally and in principle independently of a the results within an individual precinct.

1.5 Techniques Used

We make original use of standard electronic voting techniques, particularly the use of re-encryption mix networks (c.f. [7]) and group cryptography (c.f. [10]) and efficient proofs on committed values (c.f. [8]). On a very high level, voters generate linked lists of encrypted votes that specify their preferences. The encryptions are done with respect to a key that is held in aggregate by the election committee, who can decrypt elements using group cryptography. The head of the list corresponds to the highest ranked viable candidate. By using group decryption to decrypt these heads, the first round vote counts may be computed.

When a candidate is eliminated, we must efficiently search out the next element in the list. However, we must be very careful about leaking extraneous information. For example, it cannot be revealed what was the original ranking of the current head of a list. Nor can we reveal for any list the history of which elements are moved to the head (or we will reveal the list). For this reason, we keep all but the (current) head elements in a separate table of elements that is constantly remixed. This separation complicates the problem of finding the next element of a list. We use a system of random ID tags to allow us to use group decryption to find the next elements in the set.

An important technical problem we must deal with is that it would reveal too much to follow a link from an eliminated top-choice vote only to find another eliminated candidate. We must therefore perform surgery on our linked lists, deleting eliminated candidates from interiors of lists so we will never arrive at them.

To perform all of these list manipulations, we use three mix networks in different ways. Pieces of the ballots are proved consistent before being distributed among the mix networks. The consistency proofs are done using standard proofs of equality on committed values. We use standard witness-hiding techniques and heuristically replace the honest-verifiers with hash function using Gennaro's variant [13] of the Fiat-Shamir heuristic [11] (designed to avoid vote duplication attacks).

Summarizing, we present a scheme that uses a linked-list structure to represent a ballot, treats all ballots equally using three mix-networks, and also improves privacy by hiding preferences.

Road Map: In Section 2, we present the basic cryptographic elements of the protocol: mix-networks, group decryption, and plaintext equality proofs. We discuss the ballot design and voting procedure in Section 3. We briefly discuss efficiency and security in Section 4. We discuss other possible research directions in Section 5.

2 Preliminaries

We use a number of basic cryptographic primitives, which we review for self-containment of the exposition.

Re-encryption Mix-Networks: Mix-networks (or mixnets), which are used to create communication channels that are difficult to trace, consist of a series of servers that take a series of texts $M_1, \ldots, M_n$ and output a permutation $\pi(M_1), \ldots, \pi(M_n)$ of these texts. In re-encryption mixnets, each mix server takes in a series of encrypted messages and applies a re-randomization to each cipher text. In the case of an El Gamal cipher text this re-encryption corresponds to a selecting a random group element and applying a small number of group operations. Neff [22] describes a protocol for the shuffling of sequences of El Gamal pairs. We use a variant of Neff's protocol in which blocks of encryptions are mixed - the block are re-encrypted in random order, but the (plaintext) values within each block are preserved in their original order.

Secret Sharing and Group Decryption: We proceed with secret sharing as in [9]. To generate a private El Gamal key to distribute to counters, we use the (t, n) threshold protocol of Shamir [25]. Namely, for the secret exponent s, we announce shares $s_1, \ldots s_n$ for the counters, such that for any set Γ of t shares, we can recover the secret.

Using group cryptography, the authorities can simulate a single entity that alone has access to the decryption key. Decryptions of encrypted values by the group is comparatively straightforward and efficient. In our analysis, we will treat such decryptions as basic operations.

Plaintext Equality Proofs and proofs of knowledge: Given El Gamal encryptions of M_1 and M_2, $(\alpha_1, \beta_1) = (g^r, M_1 h^r)$ and $(\alpha_2, \beta_2) = (g^s, M_2 h^s)$, we can execute an efficient plaintext equality proof protocol, that proves that M_1 and M_2 are the same. Also, given an encryption of M and a known value of r, we must be able to produce (with proof) an encryption of $M' = M + r$. For most homomorphic encryption systems, one can compute the encryption of $M + r$ from an encryption of M.

It is also crucial that we can perform σ proofs of knowledge of encrypted values (i.e., proofs in which the prover sends an honest verifier a message, the honest verifier sends a random challenge to the prover, and the prover sends a reply). In practice, we "compress" such proofs using Gennaro's variant of the Fiat-Shamir heuristic in which the verifier's challenge is computed as a hash of the first message and the prover's identity (so as to avoid replaying other player's proofs). This heuristic results in a single message "certificate" that the player knows the values being committed to. We heuristically analyze our protocol as if the actual proofs were invoked.

The use of proofs of knowledge is crucial to both the correctness and privacy of our protocol. Intuitively, proving knowledge of a committed value prevents malleability attacks in which one commits to values that one doesn't know, but which are somehow related to other committed values.

3 Voting Scheme

3.1 Preliminary Setup

The protocol uses three mix networks. The pool of first place votes is sent to mix network 1, subsequent choices of each voter are sent to mix network 2, and elimination links are sent to mix network 3. At the start of each election, the authorities announce the public key used for all encryptions. Shares of the corresponding private key are distributed to the counters using the secret-sharing scheme described in the previous section.

We also assume the existence of a public "bulletin board" that is used as a staging area for the mix networks. As we describe below, the encrypted values sent through the mix networks are subject to various constraints that must be verified. The encrypted values and their consistency proofs are posted to the bulletin board and checked before being routed through the mix networks.

3.2 Counter initialization

The voting authorities collectively set up an El Gamal based public-key group encryption scheme. The public key is made public and is used for the re-encryption mixer. The private key is held in a distributed fashion by the group.

3.3 Ballot Design: Constructing the Linked List

On a high level, a ballot is composed of a set of preference elements, each of which consists of preference data and additional keys used to link the preference element. In the following discussion, i will denote the preference in the list. We will have multiple elimination rounds, index by j, each requiring separate links.

To establish a link, each preference element has a set of incoming keys (thought of as a large random number) $\mathrm{in}_{i,j}$, used to establish a connection with the preceding element in the list, and a set of outgoing keys, $\mathrm{out}_{i,j}$, used to establish links with following elements. To establish that $x_{i'}$ follows x_i in the linked list we set $\mathrm{out}_{i,j} = \mathrm{in}_{i',j}$. We similarly set up random tags $\mathrm{lose}_{i,j}$ that will aid in the removal of x_i if it corresponds to a candidate being eliminated.

For an election with k candidates, a (proper) voter does the following to construct a ballot (see Figure 1 in the appendix):

1. Determine the order of preferences, $x_1, \ldots, x_k$, where each x_i is a name (or number) representing each candidate.
2. For $i = 1, \ldots, k+1$ and $j = 1, \ldots, k$
   - Select the keys $\mathrm{in}_{i,j}$ for $i = 1, \ldots, k+1$ and $j = 1, \ldots, k$ independently at random (in fact, we require a further step, to ensure that keys are distinct; see Section 3.6). .
   - If $i \neq k+1$, let $\mathrm{out}_{i,j} = \mathrm{in}_{i+1,j}$. This operation creates the links between choices.
   - Otherwise, select $\mathrm{out}_{k+1,j}$ independently at random. This operation ends the list at the terminal choice.
   - Select the keys $\mathrm{lose}_{i,j}$ independently at random.
3. Post $(\widehat{x}_1, \widehat{\mathrm{in}}_{1,j}, \widehat{\mathrm{out}}_{1,j}, \widehat{\mathrm{lose}}_{1,j})$, encryptions of $(x_1, \mathrm{in}_{1,j}, \mathrm{out}_{1,j})$, for $j = 1, \ldots, k$ to mix network 1.
4. For $i = 2, \ldots, k+1$ and $j = 1, \ldots, k$, post the tuple $(\widehat{x}_i, \widehat{\mathrm{in}}_{i,j}, \widehat{\mathrm{out}}_{i,j}, \widehat{\mathrm{lose}}_{i,j})$ to mix network 2.
5. For $i = 1, \ldots, k+1$ and $j = 1, \ldots, k$, post the tuple $(\widehat{x}_i, \widehat{\mathrm{lose}}_{i,j})$ to mix network 3.

To complete the ballot, the voter posts plaintext equality proofs [19] made non-interactive by Gennaro's modification to the Fiat-Shamir heuristic [13] to verify that the linked list is composed properly, namely that $\mathrm{in}_{i+1,j} = \mathrm{out}_{i,j}$. To verify that the removal links point to the proper candidate to be removed, the voter must also prove that x_i and $\mathrm{lose}_{i,j}$ are equal across mix networks. Similarly, the voter posts proofs of knowledge of the encrypted values. All such proofs are posted to the public bulletin board, and may be verified by all interested parties.
Remark: For our analysis, it is useful to enforce other constraints on the ballot. For example, there is no real point in having a duplicated a name on ones list, and we may optionally wish to restrict the names to a specific list of candidates. The former may be accomplished using proofs of inequality. The latter may be accomplished used standard mix-net proofs - one writes down a list of encrypted names and proves that it is a permutation of the allowed list.

Figure 1 shows an example of each component: a portion of a vote and a removal tag, for an election with 3 candidates. A concrete example and diagram showing a full voter's posting are included in the next subsection.

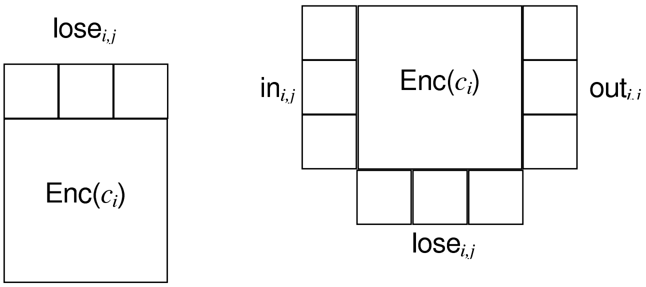


Fig. 1. A visualization of the components of a voter's ballot. A choice posted to mix networks 1 or 2 is on the left. A removal tag posted to mix network 3 is on the right. See figure 2, in the appendix, for an example of a complete ballot posted by a voter.

3.4 An Example

Consider an election with three candidates: A. Smith, B. Jones, and C. Johnson, in which a voter wants to post a vote of (Johnson, Smith, Jones) in that order. His ballot will be constructed as follows (we give a graphical example of a three candidate ballot in Figure 2):

x_1, **C. Johnson**

- Encrypt x_1.
- For $j = 1, 2, 3$
  - Select $\text{in}_{1,j}$ independently (indeed, select all keys $\text{in}_{\cdot,j}$ at random).
  - Set $\text{out}_{1,j} = \text{in}_{2,j}$ after $\text{in}_{2,j}$ has been selected.
  - Select $\text{lose}_{1,j}$ independently.
- Encrypt $\text{in}_{1,j}$, $\text{out}_{1,j}$, and $\text{lose}_{1,j}$.
- Create copies of $\widehat{x}_1$ and $\widehat{\text{lose}}_{1,j}$ by re-randomizing the encryption. As a tuple, these copies are the removal tag that gets posted to mix network 3.
- Post the tuple $(\widehat{x}_1, \widehat{\text{in}}_{1,j}, \widehat{\text{out}}_{1,j}, \widehat{\text{lose}}_{1,j})$ to mix network 1.

x_2, **A. Smith and** x_3, **B. Jones**

- Proceed as with x_1. Compute the tuples $(\widehat{x}_2, \widehat{\text{in}}_{2,j}, \widehat{\text{out}}_{2,j}, \widehat{\text{lose}}_{2,j})$ and $(\widehat{x}_3, \widehat{\text{in}}_{3,j}, \widehat{\text{out}}_{3,j}, \widehat{\text{lose}}_{3,j})$ as above.
- Post those tuples to mix network 2.
- Post the (re-encrypted) removal tags $(\widehat{x}_2, \widehat{\text{lose}}_{2,j})$ and $(\widehat{x}_3, \widehat{\text{lose}}_{3,j})$ to mix network 3.

x_4, **the terminal choice**

  - Encrypt x_4.
  - For $j = 1, 2, 3$
    - Select $\text{in}_{4,j}$ randomly and encrypt.
    - Select $\text{out}_{4,j}$ randomly and encrypt.
    - Select $\text{lose}_{4,j}$ randomly and encrypt.
  - Post $(\widehat{x}_4, \widehat{\text{in}}_{4,j}, \widehat{\text{out}}_{4,j}, \widehat{\text{lose}}_{4,j})$ to mix network 2.

In order to prove that a vote is valid, the voter must prove the following using plaintext equality proofs:

- Given $\widehat{\text{in}}_{2,j}$ and $\widehat{\text{out}}_{1,j}$, show that $\text{in}_{2,j} = \text{out}_{1,j}$ (i.e., that $\widehat{\text{in}}_{2,j}$ and $\widehat{\text{out}}_{1,j}$ encrypt the same value)
- Given $\widehat{\text{in}}_{3,j}$ and $\widehat{\text{out}}_{2,j}$, show that $\text{in}_{3,j} = \text{out}_{2,j}$.
- Given $\widehat{\text{in}}_{4,j}$ and $\widehat{\text{out}}_{3,j}$, show that $\text{in}_{4,j} = \text{out}_{3,j}$.

Similarly, show that

- x_1 in network 1 = x_1 in network 3.
- x_i in network 2 = x_i in network 3 (for $i > 1$).
- $\text{lose}_{1,j}$ in network 1 = $\text{lose}_{1,j}$ in network 3.
- $\text{lose}_{i,j}$ in network 2 = $\text{lose}_{i,j}$ in network 3 (for $i > 1$).

3.5 Counting and Elimination

Counting: After polls close, counters begin tallying votes:

1. The counters verify the posted proofs of plaintext equality, and accept those votes whose proofs pass.
2. The mix networks shuffle the pools of votes. The removal tags are mixed in round 1 only.
3. The counters leave the output of mix network 2, the voters' subsequent choices, encrypted.
4. The counters decrypt the first slots, representing the choice of candidate, of the first-place votes (from mix network 1) and of the removal tags.
5. Counters discard terminal choices or votes for eliminated candidates that show up in the primary vote pool.
6. Actual counting is trivial. The counters read the decrypted names of the first-place votes. A candidate is declared the winner if he has enough votes. Otherwise, a candidate is eliminated.

Elimination: When a candidate L is eliminated, the counters act accordingly:

1. They announce the candidate L to be eliminated in round r, and locate the removal tags corresponding to L in mix network 3. Recall that this network contains pairs consisting of encrypted names and encrypted lose values. The counters can collectively decrypt all of the names, and then for all entries corresponding to L, decrypt the corresponding lose values. These values may then be efficiently matched to their corresponding entries in mix net 2, as discussed below.

2. For each choice c in the pools of votes, the counters decrypt $\widehat{\text{lose}}_{c,r}$ and $\widehat{\text{in}}_{c,r}$.
3. For each removal tag, the counters decrypt $\widehat{\text{lose}}_{L,r}$, and search for $\text{lose}_{L,r}$ in the pools of votes.
4. When a matching lose key is found, the counters check that the choice slot encrypts L, to ensure that they are eliminating the proper vote.
5. Link forwarding is now performed; see Figure 3. The counters decrypt $\widehat{\text{out}}_{L,r}$ and search for an incoming key $\text{in}_{c,r}$. The counters use a plaintext equality test to ensure that the correct link is being followed.
6. The counters set $\widehat{\text{in}}_{c,j} = \widehat{\text{in}}_{L,j}$, for $j = r, \ldots, k$. This redirects the links from the eliminated choice to a choice that is still competing in the election.
7. If a vote for L was in the primary choice pool, the counters promote the choice found by following the link.
8. At the end of round r, the counters discard $\text{in}_{c,r}$, $\text{out}_{c,r}$, and $\text{lose}_{c,r}$ are discarded for each candidate c. All keys corresponding to round r are now discarded, and counters will use keys corresponding to round $r + 1$ for the next elimination.
9. Counters remix the votes using mix networks 1 and 2.

Remark. Eliminating a candidate and forwarding links illustrates the need for a terminal choice. If a voter's last choice is eliminated, the previous choice will now link to the terminal choice, instead of having hanging links. The terminal choice serves as an "anchor" that will always be among the pool of candidates.

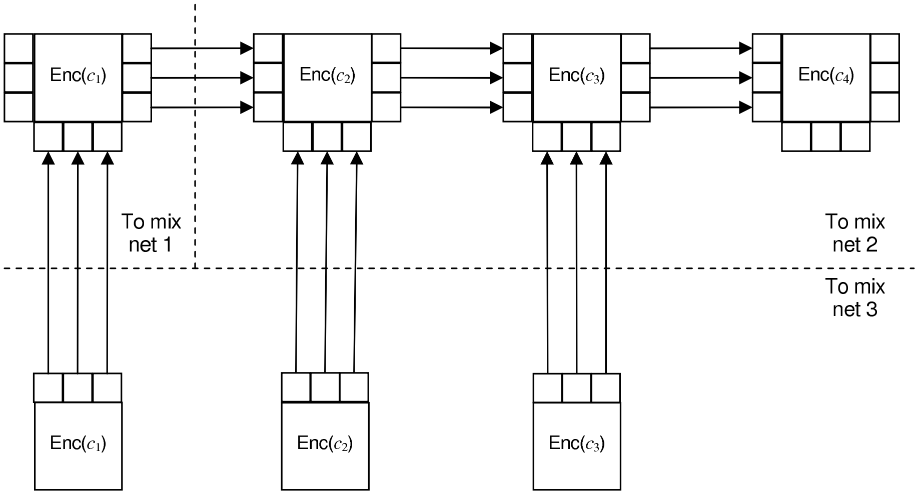
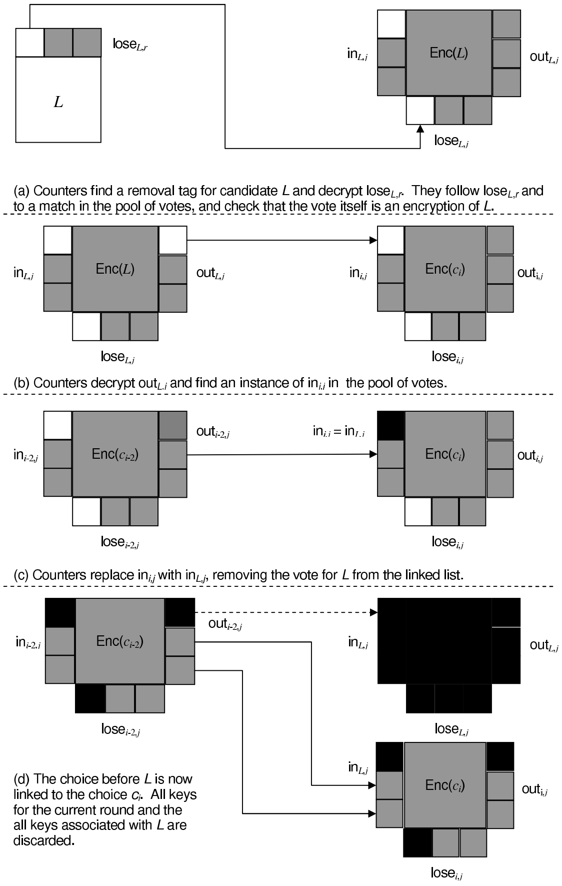


Fig. 2. A sample ballot for an election with three candidates.



(a) Counters find a removal tag for candidate L and decrypt lose$_{L,r}$. They follow lose$_{L,r}$ and to a match in the pool of votes, and check that the vote itself is an encryption of L.

(b) Counters decrypt out$_{L,j}$ and find an instance of in$_{i,j}$ in the pool of votes.

(c) Counters replace in$_{i,j}$ with in$_{L,j}$, removing the vote for L from the linked list.

(d) The choice before L is now linked to the choice c_i. All keys for the current round and the all keys associated with L are discarded.

Fig. 3. An example of link forwarding. Encrypted items are in gray, decrypted items are in white, and discarded items are in black.

3.6 Ensuring distinctness and unrelatedness of keys

Recall that a link is created by generating a random tag that appears in multiple places in the mix net. The correctness of the protocol requires that the tags be distinct and the privacy of the protocol depends on the the inability of an adversarial coalition to create nontrivial relations between their tags and those of good voters.

The latter problem is implicitly dealt with in the full privacy analysis, and follows from the fact that all of the tags come with proofs of knowledge (here we assume the idealized version of the protocol, where the proofs of knowledge are carried out). The values of the tags chosen by the adversarial players must be decided upon, and known to the adversarial players (via the extractor for the proof), given only the encryptions of these tags and zero-knowledge proofs based on these encryptions. If any nontrivial polynomial-time relation R held (with probability greater than chance) between the values chosen by the good voters and the values known to the adversaries, this could be used to obtain a distinguisher that breaks the underlying probabilistic encryption scheme.

However, nothing stops colluding voters (or even a single voter) from making two tags equal when they should not be. We solve this problem by using a standard coin-flipping in the well protocol. The interactive form of this protocol is as follows:

1. The tag creator generates a random tag T, and encrypts it, generating C.
2. A randomizer generates a random r.
3. The tag creator generates an encryption C' of $T' = T + r$. Note that for most homomorphic encryption systems, C' can be generated from C and r.

In this ideal interactive scenario, the value of T' is random. Following Gennaro, we heuristically choose r as a hash of C, the identity of the tag creator, and a representation of the "place" of this tag in the protocol as a whole (we simply ask that this representation never appear twice in the same election).

Of course, if a tag is prescribed to be equal to an earlier generated value, we simply create the commitment with this earlier value (and prove equality).

It can be shown that if T and C are chosen correctly (a random value and a random encryption), then the distribution of T' is indistinguishable from random. This is not true if T is chosen adversarially. However, by a standard argument, T' cannot be chosen to collide with any other tag value, except with negligible probability, if one replaces the hash function with a random oracle. We heuristically assume the same holds true for a suitable cryptographic hash function.

We note that the tags are homomorphically encrypted for use in the mix-net; one can achieve greater efficiency (at some loss of clarity) by putting a randomization step in at this point. Even further efficiency can be obtained by limiting the range of r, say to 192 bits even if the range of the tags is much larger.

4 Analysis

4.1 The framework and limits of our analysis

Aside from the analysis of efficiency, we cannot formally analyze our protocol in its recommended usage, which makes use of variants of the Fiat-Shamir heuristic. We instead, following a long tradition, analyze the "idealized" protocol, in which the parties engage in true proofs of knowledge and coin-flipping protocols with a trusted external party.

We also assume that while some of the counters may be corrupt, sufficiently many are honest so that the mix-net and group decryption protocols are secure and serially composable.

We also assume that the (essentially external) decisions as to which candidate is eliminated in any phase are independent of the "internals" of the protocol (i.e., based on the encrypted , though they may of course depend on the tallies of who has how many votes. We note that any sensible decision procedure will not look any deeper than the precincts vote sub-totals. This limitation may be relaxed, particularly if k is small - essentially giving the adversary full choice over the elimination sequence requires a $k!$ increase in the computational hardness of breaking the probabilistic encryptions and subverting the mix-net, coin-flipping and group decryption protocols.[1]

Thus, we view and analyze our protocol, and the attacks on it, as follows.

1. The voters, both good and malicious, prepare their encrypted lists, and perform the requisite proofs and coin-flipping protocols with an honest party. The malicious voters may see the encryptions generated by the good voters, and the transcripts of these protocols, but must engage in the proofs and coin flipping protocols anew (this is why we use Gennaro's trick to prevent the reuse of the Fiat-Shamir proofs). It is in the creation of these encrypted ballots that we allow the adversary the most freedom of operation.
2. For each phase of the counting process, the counters engage in various secure computations (mix net operations and group decryptions) on the encrypted values. As we assume that the adversary is unable to corrupt these protocols (sufficiently), we assume that
   – The operations proceed correctly.
   – The adversary is able to see the inputs and output of these operations, but not the actual operation of the protocol.

   These two assumptions are justified based on the correctness and simulatability of the underlying sub-protocols. Given the inputs and outputs, anyone can simulate the set of messages comprising the execution of the secure computation.

After some of these secure computations, tallies of votes for each surviving candidate are generated. We call these tallies *ideal snapshots*. We call the output of the secure computations *protocol snapshots*.

Thus, we can view the attack on the protocol as comprising the (mis)generation of ballots followed by the observation of a series of protocol snapshots. We compare such an attack with an ideal attack, which works as follows:

1. The voters, adversarial or not, create ordered lists of candidates.
2. Initially, or after a candidate has been eliminated, the tallies of current first choice votes for candidate are revealed, corresponding to the ideal snapshot defined above.

To analyze correctness, we observe that our protocol (at least in its idealized form) ensures that the ballots correspond to well-defined lists of candidates, and that the resulting "ideal snapshots" are what they should be given given this list. To analyze privacy, we go on to show that given the information that may be extracted from the adversarial voters and the ideal snapshots, one may generate simulated protocol snapshots that are computationally indistinguishable from the actual protocol snapshots.

[1] We suspect that with some care, the $k!$ factor may be reduced to $k^{O(1)}$. However, a slightly more intricate analysis is required.

4.2 Efficiency

In a correct vote, each choice consists of a name slot and $O(k)$ keys. The complete construction of the linked list requires $O(k^2)$ key values. Because El Gamal encryption and the plaintext equality proof take a constant number of exponentiations, a quadratic number of exponentiations is needed to cast a vote. Each ballot will also require $O(k^2)$ encryptions. The centers must perform shuffles on $O(nk^2)$ encrypted values per elimination round. Group decryptions must be performed on $O(nk)$ encrypted values per elimination round.

4.3 Correctness

To show that this protocol is correct, we show that accepted ballots correspond to independent, well-formed lists of names, and that the protocol performs the correct operations on these lists.

Lemma 1 summarizes the result of the zero-knowledge proofs of knowledge and coin-flipping protocols.

Lemma 1. *Suppose we have a collection of submitted ballots that have passed the zero-knowledge proofs of knowledge given in the Section 3.3. Then, assuming that all the ballot creators run in probabilistic polynomial time and that the probabilistic encryptions are secure, the following will hold almost always:*

1. *All accepted ballots can be mapped to a well-formed list of names and well formed tag values; all such values may be extracted from the entity submitting the ballot (and hence performing the proofs of knowledge).*
2. *All tag values that are specified by the protocol to be equal will be equal; any two tag values that are not specified to be equal will not be equal.*

One important consequence of the proofs of knowledge is that vote duplication or other forms of mauling are impossible. Suppose that the good voters have vote lists $\{L\}$ and generate the (essentially) random tags $\{t\}$ used for the linked lists. We consider two types of adversary. The *ideal model* adversary, A', chooses vote lists $\{L\}$ and tags $\{t'\}$, without seeing $\{L\}$ and $\{t\}$. The *real model* adversary, A^* sees a transcript consisting of the actual ballots generated by the good voters, and is allowed to generate ballots for itself. However, it must perform the specified proofs of equality and knowledge on these ballots; let $\{L^*\}$ and $\{t^*\}$ be the lists and tags obtained by the extractor for these proofs (by Lemma 1, these lists are well defined with all but negligible probability). Lemma 2 asserts that A^* cannot use its extra information to any better effect than A'.

Lemma 2. *For any probabilistic polynomial time adversary, A^*, there is a probabilistic polynomial time adversary A' such that $(\{L\}, \{t\}, \{L'\}, \{t'\})$ is computationally indistinguishable from $(\{L\}, \{t\}, \{L^*\}, \{t^*\})$.*

Proof. (Sketch) We use a standard hybrid argument. Given A^*, we create a hybrid adversary, A_1, that runs A^* given the encryptions, but with simulated proofs instead of actual proofs. The output of this adversary must be computationally indistinguishable from that of A^*, or we have a violation of the zero-knowledge property. We define A' as the adversary that generates random encrypted values and runs A_1. The output of A' must be computationally indistinguishable from that of A_1, or there would be a violation of the semantic security of the encryption.

We pause to reflect on the meaning of Lemma 1 and Lemma 2 for the types of attacks that can be staged during the ballot reconstruction phase. The adversary must create ballots that correspond to well formed lists and tags, such that the set of tags have no spurious duplications. The lists and tag values had might as well be chosen independently of the honest voters. In short, the adversary acts no differently than an adversary that chooses its lists and tags and engages in the protocol.

It remains to consider the remainder of the protocol. Recall, we assume that the adversary is assumed not to be able to corrupt enough counters to interfere with the mix-net and group decryption operations.

We observe that the details of the ballots (other than the fact that they are valid) are essentially irrelevant to the rest of the protocol. The proofs are essentially dropped once they are verified, leaving only the choice of encryptions. Recall that a re-encrypting mix-net replaces the encryption of some value x with a random encryption of x. Thus, the precise encryptions chosen by the adversary almost immediately become irrelevant, as summarized in Lemma 3.

Lemma 3. *The result of the first re-encrypting mix-net operation depends only on the values of the lists and tags encrypted in the ballots, not on the ballots themselves.*

Thus, the only effective difference between a general adversary that chooses its ballots and a comparatively ideal adversary that chooses its list of candidates and then participates in the protocol is that the general adversary can specify its tags arbitrarily (but not to collide spuriously). By a straightforward but tedious argument, one can show the following:

Lemma 4. *Given a set of well-formed ballots, corresponding to a set of lists of candidates, with no spurious tag collisions, and sequence of candidate eliminations, the vote counts produced at each round will be the same as that produced by the ideal vote-counting algorithm on these lists of candidates.*

Hence, the (partial) freedom to choose the tag values is irrelevant to the intermediate counts of the protocol.

The above Lemmas imply the correctness of our (idealized) protocol.

Privacy The methodology of the previous section can be extended to simultaneously establish privacy as well. Consider the view of the adversary attempting to corrupt the election. At the time it selects its ballots, it has only seen probabilistic encryptions of the good voters' lists and tags, and zero-knowledge proofs on these values. As with the proof of Lemma 2, we can simulate this view with simulated proofs on random committed values. It remains to simulate the views of the later parts of the protocols. As before, we use the extraction property of the proofs to extract the lists $\{L'\}$ and tags $\{t'\}$ specified by the adversary. By the previous section (particularly Lemma 3), once the ballots have been constructed and tested, these values are the only aspects that are relevant to future steps of the protocol.

We consider the view of the adversary in the ideal and actual settings. In the ideal setting, the adversary sees $\{L'\}$ and $\{t'\}$ and then sees the sequence of intermediate vote counts (one initial, and one for each elimination phase). In reality, the adversary sees a sequence of "snapshots" consisting of encrypted values output by the mix net, of which some subset are revealed at each stage, as specified by the protocol and which candidates are eliminated. Additionally, there is the adversaries view of the actual secure computations we are invoking, but these are assumed to be simulatable. Lemma 5 states that one can simulate the snapshots given the information available in the ideal model.

Lemma 5. *Given the vote lists $\{L'\}$ and tags $\{t'\}$ given by the adversary, and the sequence of vote totals generated in each elimination phase, and the identities of each eliminated candidate, one can in probabilistic polynomial time generate simulations of the output of each secure computation operation that are computationally indistinguishable from the outputs of the protocol.*

The proof is a tedious but straightforward hybrid argument.

5 Discussion

Receipt Freeness: One of the more obvious deficiencies of this protocol is its lack of receipt-freeness. It seems likely that, at the cost of modestly greater complexity, one can make a receipt-free version of this protocol using standard techniques (though we do not claim such a result). The natural approach would be for voters to interact with a voting entity to securely compute a ballot; the voter inputs its preferences, but has no more knowledge of the proofs and encryptions than if another voter had cast a ballot with the same preference list. While general secure computation is impractical, the operations required for constructing a ballot, namely creating randomized encryptions for the candidate names, random tags and proofs of equality of these tags, are quite amenable to this approach.

Practicalities: It should be pointed out that we have ignored an entire space of trust and security issues, assuming for example that voters have completely trustworthy implementations of their part of the protocol. We view this work as an early step towards efficient preference-based voting.

Extension to multiple winners: This protocol only covers the case of an election with a single victor. If the election is for multiple seats, winners get "eliminated." They keep a quota's worth of first-choice votes, with the surplus getting redistributed with a fractional weight. From this protocol, a STV protocol, which modifies this protocol by preserving preference hiding and using the same ideas for link forwarding, but taking the fractional redistribution of votes into account, may arise.

Handling multiple losers and Write-in votes: It may be foreseeable that a number of candidates with relatively small tallies of votes will not be able to garner enough votes to win the election. In this protocol, the votes have to be reshuffled after each elimination, or authorities may reveal significant link information. We would like to modify this protocol so that multiple losing candidates can be removed efficiently. This would also allow for the inclusion of write-in candidates. Write-in candidates with a significant number of votes will stay in the vote pool, while the occasional sporadic write-in vote will be eliminated promptly.

Incomplete Voting: A voter may not need to fill out a complete ballot, instead opting for ranking t-out-of-k candidates. In San Francisco elections, for example, voters select only three out of k candidates when voting. This scheme is adaptable to such an incomplete vote, so long as voters post one key per candidate. Each vote listing t candidates will take $O(tk)$ bits. Schemes that encode a full list of choices in one ballot will now require at least $\log(k!) + 1$ bits. If t is sufficiently small, then this system also improves on the space efficiency of previous schemes. On the other hand, the privacy of some ballots will be compromised, as terminal choices will appear in the primary pool of votes; counters may be able to reconstruct ballots consisting of only eliminated candidates. One potential solution to this is to have a voter post dummy choices to fill out the ballot.

6 Acknowledgments

We thank the anonymous reviewers for many useful comments.

References

1. Aditya, R., Boyd, C., Dawson, E., and Viswanathan, K. Secure e-voting for preferential elections. In *Electronic Government: Second International Conference, EGOV 2003* (2003), vol. 2739, pp. 246–249.
2. Baudron, O., Fouque, P.-A., Pointcheval, D., Stern, J., and Poupard, G. Practical multi-candidate election system. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 2001), ACM Press, pp. 274–283.
3. Benaloh, J., and Tuinstra, D. Receipt-free secret-ballot elections (extended abstract). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 1994), ACM, pp. 544–553.
4. Benaloh, J. C., and Yung, M. Distributing the power of a government to enhance the privacy of voters. In *PODC '86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1986), ACM, pp. 52–62.
5. c. Yao, A. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science* (1986), pp. 162–167.

6. Canetti, R., and Gennaro, R. Incoercible multiparty computation (extended abstract). In *IEEE Symposium on Foundations of Computer Science* (1996), pp. 504–513.
7. Chaum, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24* (1981), 84–88.
8. Cramer, R., Damgård, I., and Schoenmakers, B. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology — Crypto '94*, Springer-Verlag, pp. 174–187.
9. Cramer, R., Gennaro, R., and Schoenmakers, B. A secure and optimally efficient multi-authority election scheme. *Lecture Notes in Computer Science 1233* (1997), 103+.
10. Desmedt, Y. G., and Frankel, Y. Threshold cryptosystems. In *Advances in Cryptology — Crypto '89*, Springer-Verlag New York, Inc., pp. 307–315.
11. Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — Crypto '86* (New York, 1987), Springer-Verlag, pp. 186–194.
12. Furukawa, J., and Sako, K. An efficient scheme for proving a shuffle. In *Advances in Cryptology — Crypto '01* Springer-Verlag, pp. 368–387.
13. Gennaro, R. Achieving independence efficiently and securely. In *PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1995), ACM Press, pp. 130–136.
14. Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing* (New York, NY, USA, 1987), ACM, pp. 218–229.
15. Hevia, A., and Kiwi, M. Electronic jury voting protocols. *Theor. Comput. Sci. 321*, 1 (2004), 73–94.
16. http://www.cambridgema.gov/Election/proprep.html.
17. http://www.cs.st-andrews.ac.uk/ tws/research/bibliography.pdf.
18. http://www.sfgov.org/site/election_index.asp.
19. Jakobsson, M., and Juels, A. Addition of elgamal plaintexts. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security* (London, UK, 2000), Springer-Verlag, pp. 346–358.
20. Juels, A., Catalano, D., and Jakobsson, M. Coercion-resistant electronic elections. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society* (New York, NY, USA, 2005), ACM, pp. 61–70.
21. Kiayias, A., and Yung, M. The vector-ballot e-voting approach. *Lecture Notes in Computer Science 3110* (2004), 72–89.
22. Neff, C. A. A verifiable secret shuffle and its application to e-voting. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security* (New York, NY, USA, 2001), ACM Press, pp. 116–125.
23. Peng, K., Boyd, C., and Dawson, E. Simple and efficient shuffling with provable correctness and zk privacy. *Advances in Cryptology — Crypto '05*, Springer-Verlag, pp. 188–204.
24. Sako, K., and Kilian, J. Receipt-free mix-typc voting scheme-a practical solution to the implementation of a voting booth. In *Advances in Cryptology-EUROCRYPT '95* (1995), vol. 921, Springer-Verlag, pp. 393–403.
25. Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (1979), 612–613.
26. Wikstrom, D. A universally composable mix-net, 2004.