Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting

Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen and Vanessa Teague

***Abstract*—There are many advantages to voting schemes in which voters rank all candidates in order, rather than just choosing their favourite. However, these schemes inherently suffer from a coercion problem when there are many candidates, because a coercer can demand a certain permutation from a voter and then check whether that permutation appears during tallying. Recently developed cryptographic voting protocols allow anyone to audit an election (universal verifiability), but existing systems are either not applicable to ranked voting at all, or reveal enough information about the ballots to make voter coercion possible.**

We solve this problem for the popular single transferable vote (STV) ranked voting system, by constructing an algorithm for the verifiable tallying of encrypted votes. Our construction improves upon existing work because it extends to multiple-seat STV and reveals less information than other schemes. The protocol is based on verifiable shuffling of homomorphic encryptions, a well-studied primitive in the voting arena. Our protocol is efficient enough to be practical, even for a large election.

I. Introduction

In elections of all kinds it is important that votes are tallied correctly, individual votes are private, voters are free of coercion, and that these properties are not just present but also clearly apparent to all. Electronic voting could improve the situation: some schemes offer *universal verifiability*, meaning that anyone can check that the announced tally is correct. Ideally it should be unnecessary to trust either the implementors of the program or the security of the computer used for voting. However, such voting systems must take particular care to prevent voters from being able to prove to coercers how they voted. This property is known as "receipt-freeness" or "coercion-resistance". Without it, coercers can either promise to reward voters if they show they voted in a particular way or threaten to cause harm to them if they do not.

Although many electronic voting systems are designed for simple plurality voting, richer voting schemes allow voters to express more than simply their single favourite candidate. The tally method we target is known as Single Transferrable Vote (STV) [1]. It can be used to fill single or multiple vacancies and with the latter, achieves a form of "proportional representation". The multiple-vacancy case is used in Australia, Ireland, Malta, Scotland, and Cambridge MA (USA). It is particularly susceptible to coercion and is the main focus of this paper. Single-seat STV is more widespread, with uses including the London Mayoral race and various other examples throughout the United States and the British Commonwealth. In this case, the coercion problem might still apply if there are many candidates. Hence our algorithm (with an obvious simplification) might still be useful.

If votes contain little information, for example, they are just "yes" or "no", or a choice of one of a small number of possible candidates, they can rarely be used to identify individual voters.[2] However, with STV a voter can specify any permutation of the candidates; this has much greater information content. Hence the "short ballot assumption" [2] fails even when there are not very many candidates. If all individual votes are ultimately revealed then this introduces a coercion problem, sometimes called the "Italian attack". For example, voters can be coerced to give a particular candidate their first preference, then the remaining preferences can be used to encode the identity of the voter. (A typical Australian Senate election has about 70 candidates, so there can be 70! different possible votes.) It is easy for a coercer to choose one vote that is very unlikely to occur, demand that a voter cast that particular vote, and then look through the ballots to see whether that vote appears. Indeed, there are so many possible votes that a coercer can choose a different required vote for a very large number of voters even when imposing some political constraints on the votes, such as only requiring those with the coercer's favourite candidate first. Although this method of coercion requires some work for the coercer, it is feasible and the risk has been described in the voting literature [3]. The problem has created a tradeoff between verifiability and coercion-resistance for paper-based STV elections,[3] but it applies even more strongly to electronic systems in which votes are posted on a bulletin board. Complete disclosure of all ballots exposes the voters to coercion, but incomplete exposure

A preliminary version of this paper appeared as [1]

J. Benaloh is with Microsoft Research; email: benaloh@microsoft.com

T. Moran is with the Center for Research on Computation and Society, Harvard University; email: talm@seas.harvard.edu

K. Ramchen, L. Naish and V. Teague are with the Department of Computer Science and Software Engineering, The University of Melbourne; emails: {vteague, lee}@csse.unimelb.edu.au, kramchen@gmail.com

[1] Sometimes known as "Quota Preferential", "instant run-off" or "alternative vote."

[2] This assumes that ballot separation is used so that only one race or question appears on each ballot.

[3] Until recently, the Australian Electoral Commission revealed the exact numbers of voters who selected the ticket (*i.e.*, recommended vote) of a major party, but released only the first preference of those who chose their own permutation. In practice this was usually enough to verify 3 or 4 of the 6 seats to be filled in each state. After pressure from voters' organisations advocating transparency, the AEC began releasing complete data.

can make verification far more difficult.

A. Cryptography background

We can roughly classify existing cryptographic voting schemes into two types by how they prove the correctness of the tally:

- **Homomorphic Tally Schemes.** In voting systems of this type, the encryption scheme of the ballot used in the voters' receipts has a "homomorphic" property: given $E(x)$, an encryption of x, and $E(y)$, an encryption of y, it is possible to compute $E(x+y)$ without knowing x, y or the secret key of the encryption. This property can be used to compute the sum of all the votes (e.g., think of a case when a vote for Alice is represented by an encryption of 1 and a vote for Bob by an encryption of 0). The final value can be verifiably decrypted and compared to the tally without revealing anything about individual votes.
- **Mixing Schemes.** Schemes of this type are the cryptographic equivalent of shuffling ballots in a box and then revealing the contents. Loosely speaking, a new set of encrypted ballots is published together with a proof that the new ballots encrypt a permutation of the values encrypted by the original encrypted ballots. This proof hides the permutation itself, however, so the new ballots can be verifiably decrypted (and compared to the tally).

Homomorphic tally schemes are usually simpler to code and understand, but are only applicable to social choice functions that can be computed from sums of individual votes (such as plurality elections). Mixing schemes are more general, but they require the contents of the ballots to be revealed as part of the tally correctness proof. For some social choice functions, such as STV, the contents of a ballot can suffice to identify individual voters, even when the explicit connection between voter and ballot is hidden by the mixing protocol.

B. Our Contributions

In this paper we construct an efficient protocol called *shuffle-sum* for verifiably tallying STV elections without revealing the ballot contents. To do this, we propose a new way to combine homomorphic tallying and mixing schemes. The main idea is to use different ballot representations for different operations on ballots, such that for a given operation, the ballot representation allows homomorphic tallying techniques to be used efficiently. We use publicly-verifiable mixing schemes to verifiably convert one ballot representation to another. The authorities produce a transcript in which each step of the tallying process can be independently verified while revealing very little information about the individual ballot contents, thus reducing the opportunities for coercion. We also provide an alternative data structure that allows for faster tallying at the expense of much less efficient setup. This variant, called *table-sum*, is described in Section III-A. In addition, we show how to reduce further the information revealed, as long as none of the authorities collude with the coercer (Section III-B).

Our tallying scheme receives as input a table of encrypted votes. The initial table of encrypted votes could be derived from an electronic voting scheme or from inputting and then encrypting paper ballots (though verification of the latter would be difficult). One example of an end-to-end verifiable election scheme that produces the right format is contained in Appendix A.

We have implemented the *table-sum* version of our scheme and tested it on a subset of the real data for the Victorian State election. Although the computation is expensive, it is feasible for real elections. See Section IV-B for details.

The remainder of this section gives an overview of STV and existing work on coercion-resistant STV tallying, then an overview of our main contribution, an algorithm for provably correct STV tallying that reveals very little unnecessary information. Section II contains a more complete description. Section III describes some optimisations and alternatives, including, in Section III-B1, some examples of coercion that can still occur even under (some) apparently coercion-resistant schemes. Analysis and the sketch of a security proof appear in Section IV. Finally, in Section V, we show how the same techniques can be applied to demonstrating a stable solution to the hospitals and residents problem.

C. STV Tallying

Single Transferable Vote is a social choice function (actually a family of similar functions) that attempts to reduce "wasted votes". In this paper we consider only one of the many slightly differing variations of STV[4]. However, our techniques will work for most of them with minor modifications. In the variant of STV we consider, many candidates are competing for a small number of seats. Voters must rank all the candidates.[5] Once voters have cast their ballots, a quota is computed: the quota is the number of votes required for a candidate to be elected. We will use the *Droop* quota: $q \leftarrow \left\lfloor \frac{n}{s+1} \right\rfloor + 1$, where n is the number of voters and s is the number of seats to be filled. This is the minimum number of votes required such that the number of candidates elected is never more than the number of open seats.

Tallying is an iterative process consisting of multiple rounds, each consisting of several steps:

1) **Compute first-preference tallies**. For each candidate, sum the weight of the ballots containing that candidate first in the ranking (all ballots initially have weight 1, but the weight may be reduced in successive rounds, as described below).
2) **Elect or eliminate candidates**. Every candidate who reached the quota in first-preference votes for this round is declared elected. If no candidate has reached the quota, the candidate with the fewest first-preference votes is eliminated.
3) **Reweight votes**. For every elected candidate, the ballots on which that candidate appears as a first-preference must be reweighted. This is achieved by multiplying by the *transfer value*, defined as $\frac{m-q}{m}$, where q is the quota

[4]One set of rules can be found at `http://www.prsa.org.au`

[5]It would be straightforward to modify our protocols to relax this restriction, but we make it here for simplicity. See Sections II-F, III-A2 and III-B3 for details of how to alter each variant of our algorithm.

and m is the total weight of first-preference ballots for the candidate. Informally, we can think of a vote as being only "partially used" if more votes were cast than were needed to guarantee election. The remainder of the vote will be transferred to the next preference on the ballot. If a candidate is eliminated, the transfer value is 1 (i.e., the entire vote will be transferred to the next preference). Ballots on which the first-preference candidate is neither elected nor eliminated also have transfer value of 1. Note that each ballot is reweighted at most once in every round in which a candidate is elected, using the transfer value for the first-preference candidate on that ballot (regardless of how many candidates are elected).

4) **Eliminate Candidates**. The candidates that were elected or eliminated in the previous step must now be removed from all of the ballots. In this step, the first preference on the ballot may change if the previous first preference candidate was elected or eliminated.

The tallying process continues until all seats have been filled or until the number of remaining candidates equals the number of open seats (in which case those candidates are elected).

D. Prior work

There has been great progress in recent years on universally verifiable election schemes. Some schemes [4], [5], [6], [7] use homomorphic encryption and tally the encrypted ballots, revealing only the total. However, none of these schemes supports preferential voting. Other very successful schemes are based on mix networks [8], [9], [10].

Of these, Prêt à Voter [8] easily incorporates preferential voting [11], and most of the others could be modified to support it. While none of these schemes introduce a coercion problem directly, they all end with a step that decrypts all the votes and then publicly tallies them, and the publication of these raw votes may provide opportunities for coercion . The aim of this paper is to devise a tallying step for STV that can be used at the end of a voting scheme without introducing a coercion problem. We do this by using homomorphic encryption. We assume a particular format for the encrypted ballot, which could easily be achieved by modifying Prêt à Voter as described in Appendix A.

There are several existing cryptographic schemes for verifiable STV tallying that limit the information revealed. Heather [12] describes how to implement STV counting securely in Prêt à Voter. McMahon [13], Keller and Kilian [14], and Goh & Gollé [15] describe structures for secure STV tallying that are not attached to a particular electronic voting scheme. None of these allow the re-weighting necessary for the multiple-seat case. More subtly, every one of these works reveals every candidate's running tally after every redistribution, which still leaves open the possibility of some coercion, especially if there are many candidates. This is described in Sections III-B1b and III-B1c. (Our scheme also reveals this information, but can be modified (at some cost) to hide it if necessary—see Section III-B.) Wen and Buckland [16] describe an alternative scheme which includes many variants of multiple-seat STV and other counting schemes and has the same asymptotic complexity as Shuffle-Sum and Tally-Sum. Their scheme does not need to reveal any more than the set of candidates that were elected, but is structured very differently from ours. Specifically, Wen and Buckland depend very heavily on general primitives for secure multiparty computation, and is consequently much less efficient than ours in practice. A similar coercion problem was addressed for the Condorcet voting scheme in [17], but their techniques do not extend to STV. The main advantages of our scheme are that it can perform the re-weighting step and hence tally multiple-seat races, and that it reveals much less information than other schemes.

We know of no existing definition of privacy for electronic voting that explicitly considers the possibility that just revealing anonymised votes allows coercion. Crutchfield *et. al.* [18] consider the privacy violations of publicly-released voting statistics, but their model is specifically for single-selection voting schemes, so it does not apply to our case. (They are interested in cases where precinct-based election results are unanimous or nearly so.) Previous cryptography literature has concentrated on *receipt-freeness* [19], which means that a voter cannot prove to the coercer how she voted, even if she wants to. Often this is used interchangeably with the term *coercion-resistance*, introduced in [20], though sometimes a distinction is made between a coercer who can communicate with the voter before or during the election (requiring coercion-resistance) and one who cannot (requiring receipt-freeness, which is then a weaker property). In this paper we consider only the stronger version, and call it coercion-resistance.

Moran and Naor [6] define coercion-resistance in a more general sense (extending [21] by Canetti and Gennaro). Loosely speaking, a scheme is coercion-resistant by their definition if any successful coercion attack against it would also succeed in an *ideal world*, in which — instead of running the protocol — a trusted third party (the *ideal functionality*) collects the voters' inputs and then announces the results.

In simple plurality elections, the ideal functionality reveals the number of voters for each candidate, which is equivalent to revealing all the (anonymised) votes. For STV, defining the ideal functionality is much harder: as we have just argued, an ideal functionality that revealed all the votes would expose voters to coercion via the Italian attack. The simplest functionality for STV would be one that outputs the set of winners and nothing else, though this is probably overly restrictive, making the problem too difficult and denying voters access to statistics that they may be interested in. In general the question of whether a particular scheme securely implements a particular ideal functionality is independent of the question of whether coercion is still possible under that functionality.

For most of this paper we assume the output of the ideal functionality is the number of first-preference votes each candidate received in each election round and the transfer values used to reweight the votes after candidates are eliminated (this, or similar data, is usually revealed in current STV implementations). In Section III-B the ideal functionality reveals only the order of candidate eliminations or elections, and an approximation to each transfer value.

We describe only the tally mechanism of an election,

which takes place without any voter interaction. Thus, we cannot claim our scheme is coercion-resistant (according to any of the above definitions) without taking into account the particular voter input stage used in the implementation. Our claim is that our scheme does not reveal any information about the votes beyond that revealed by the ideal functionality. Thus, given that the voter input scheme has appropriate coercion-resistence properties, the complete scheme will also be coercion-resistant.

E. Informal Protocol Description

In this section we give an informal description of the protocol and the intuitions behind it. Our protocol is only concerned with tallying published, encrypted ballots — we assume the existence of some other mechanism for casting votes and convincing voters that the published ballots are consistent with their choices. Schemes that can achieve this include Heather's version of Prêt à Voter [12] as well as [22], [23]. Variants of Neff's scheme [9] or the related Moran-Naor input mechanism [6] can also be used, giving a more secure but also more complicated voter interface. For simplicity of the presentation, we assume the candidates are numbered consecutively from 1 to m.

1) Data structures: The originally published ballots must contain all the information necessary to tally the votes, without revealing the voters' preferences. The general idea is to use the easiest ballot representation for each step of the tally process and change between representations using mixing. We use four different representations for a ballot:

- **Candidate-order ballot**. A list of encrypted preferences in candidate order (i.e., the ballot consists of an encryption of the ranking of candidate 1, followed by an encryption of the ranking of candidate 2, etc.). This is just an encrypted version of the usual STV paper ballot. An example is shown in Table I in section II-C.
- **Preference-order ballot**. A list of encrypted candidates in preference order (i.e., a ballot consists of an encryption of the index of the first-preference candidate, followed by an encryption of that of the second-preference candidate, etc). An example is shown in Table II in section II-C.
- **Candidate elimination ballot**. Just like the preference-order ballot, but with an encrypted flag "Eliminated" for each candidate, which is 1 if the candidate has been eliminated and 0 otherwise. (The list of preferences is also present, in order but encrypted.)
- **First-preference ballot**. A list of encrypted weights in candidate order, such that the voter's current first preference has the correct overall weight for this vote, and all other weights are zero. An example is shown in Table III in section II-C.

(We defer a fifth ballot representation, *table-of-comparisons form*, until Section III.)

In all cases we require the encryption scheme to be an additively homomorphic encryption scheme with verifiable threshold-decryption (e.g., the threshold version of the Damgård-Jurik scheme [24]). The trustees for the election will hold the secret-key shares to allow decryption — if a predetermined minimum number of trustees are honest, ballot secrecy will be guaranteed[6] (integrity is guaranteed even if all the trustees are corrupt). A more efficient version of the protocol can be used if there is only a single voting authority, which would then know the contents of all votes (this might be the case if the Shuffle-Sum protocol is used to provide additional verifiability to the currently used tally methods, which are performed by a single election authority).

a) Mixing Schemes: We use a mixing scheme as a subprotocol for verifiably shuffling tuples of encrypted values. For our security guarantee to hold, we require the mixing scheme to output a zero-knowledge argument that the decryption of the values at the output of the mix is a permutation of the decryptions of the values at the input to the mix. Efficient mixing schemes exist with this property for most homomorphic public-key encryption schemes (for example, the Groth shuffle [25]). It may be possible to implement the protocol using mixing schemes that do not have the zero-knowledge property (such as the extremely efficient randomized partial checking scheme [26]), however in this case the security claims and proofs for the tally protocol must be modified as well.

b) Notation: Homomorphic addition is denoted by $\oplus$, homomorphic subtraction by $\ominus$, and homomorphic multiplication (by an unencrypted constant) by $\otimes$.

2) Algorithm: Going over the steps in order:

1) **Compute first-preference tallies**. Since we are computing sums, we are naturally led to use the homomorphic property of the encryption scheme. However, neither the preference-order nor the candidate-order representations are easily used to perform the tally, so we first convert each ballot into first-preference form. An example of the first-preference tally computation appears in Figure I.1. Numbers surrounded by a black square represent encrypted values. From ballots in first-preference form, we can directly compute the encrypted first-preference tally for a candidate by homomorphically summing the encrypted values for that candidate in all the first-preference ballots
   The trustees then cooperate to decrypt the first-preference tallies.
2) **Reweight votes**. The transfer value for each candidate is publicly computed using the revealed first-preference tallies. Then, the weight of each first-preference vote for that candidate must be multiplied by the transfer value. Multiplying by a public constant can be performed using the homomorphic property of the encryption scheme. However, we do not want to reveal which votes are reweighted (since that would reveal the first preferences). The first-preference ballot representation turns out to be suited for this task as well: the ballot contains an encrypted weight for each candidate. We can multiply each of the weights by that candidate's transfer value, and homomorphically sum all the weights. Since only the weight corresponding to the first-preference candi-

[6]This number t can be chosen to be as small as one, but it may be desirable to choose $t > 1$ because the protocol allows recovery from up to $t-1$ failures.

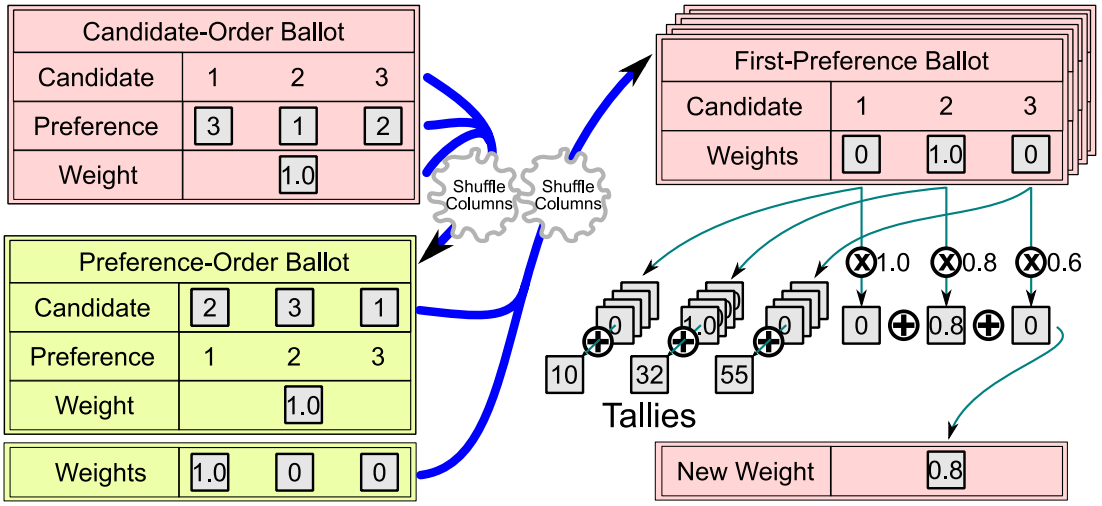

Fig. I.1: An Example of Tallying and Reweighting

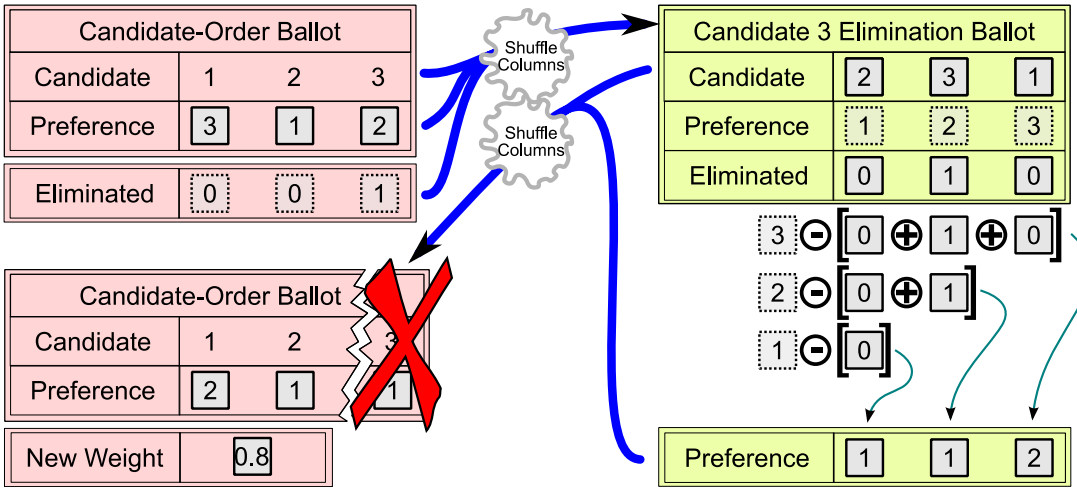

Fig. I.2: An Example of Candidate Elimination

date was nonzero, the resulting sum gives us the correct weight. Figure I.1 shows an example of reweighting a ballot. In the example, we are reweighting simultaneously for all three candidates, with weight 1.0 for candidate 1, weight 0.8 for candidate 2 (which is this voter's first-preference) and weight 0.6 for candidate 3.[7]

3) **Eliminate Candidates**. This is the trickiest part. We start out once again with the candidate-order ballot representation. From this we generate a representation that has the encrypted candidates in preference-order, but in addition has an encrypted indicator for each candidate that specifies which are the eliminated candidates (1 for the eliminated candidates and 0 for the others). We then encrypt the preference values (without shuffling). We call the resulting ballot form the *candidate elimination* representation. Using this representation, we can homomorphically subtract from the encrypted preference value of each candidate the number of eliminated candidates that appear before it in the ranking (the difference is the new preference value for the candidate). Now we return to the encrypted preferences in candidate-order representation, and can remove the eliminated candidate and preference value. Figure I.2 gives an example of a candidate elimination. In the example, candidate 3 is being eliminated. Light-colored numbers surrounded by dotted squares represent the encrypted values whose plaintexts are publicly known (we need them to be encrypted so we can homomorphically add them to an encrypted, unknown value).

[7]Note that only a candidate that has been elected will carry a transfer value less than one.

II. Shuffle-Sum Protocol Description

A. Technicalities of our STV implementation

Although the basic idea of STV is simple, the details are complicated in practice. Indeed, intense debate rages over the best methods for breaking ties and deciding where to redistribute votes (see [27] for one example of debate over

the best method for counting computerised STV votes). Most variants are shortcuts which facilitate counting by hand. Even now, with computerised counting almost ubiquitous, outdated legislation often requires these shortcuts to be performed electronically. If necessary, our scheme could be modified to incorporate many of the common counting rules, but we would strongly advocate modifying the legislation instead. We have implemented the following variants:

- When a candidate is elected, we redistribute every vote and we take the vote's new weight to be the product of the transfer value and its previous weight.[8]
- If two or more candidates have a quota, we elect them all in the same step and redistribute each vote to the next continuing candidate. The weight of each vote is multiplied by the transfer value of the highest-ranked candidate that is elected in that step.
- We assume that all votes are complete permutations. (In Section II-F we show how to relax this requirement using an idea from [12].)

B. Updating weights

In the informal protocol description given above, we described multiplying weights by a non-integer transfer value, but technically this is infeasible because it (often) requires computing roots. Instead when the candidate's transfer value is announced, we produce a rational approximation a/d and multiply the weight of every vote for the elected candidate by a, and all other weights by d. We must then keep track of the product of all the denominators d that have been used, because the quota should be multiplied by that product in subsequent rounds to decide which candidates should be elected.

Of course, it is possible to run the computation with exact transfer values. As described in Section III-B1c, this introduces its own coercion problems. We therefore leave the choice of approximation as a parameter in our scheme. Any reasonable method of taking approximations can be accommodated, as long as it does not overflow the range of values that can be represented (and efficiently decrypted) in the encryption scheme.[9] For example, using exact values with one million voters and 5 seats could produce a denominator too large for feasible decryption with exponential El Gamal.

This reweighting process can easily be completed in parallel for more than one elected candidate. Let the elected candidates' transfer values be $s_1, \ldots, s_N$, with rational approximations $\frac{a_1}{d_1}, \ldots, \frac{a_N}{d_N}$ respectively. These are publicly computed using the revealed first-preference tallies. We can choose a common denominator $d' = \text{lcm}(d_1, \ldots, d_N)$, and compute a new representation for each approximation: $\frac{a'_i}{d'} = \frac{a_i}{d_i}$, where $a'_i = a_i \frac{d'}{d_i}$. The weight of each first-preference vote for an elected candidate i would be multiplied by a'_i, while the weight of each first-preference vote for a non-elected candidate will be multiplied by d'.

[8]Different rule sets vary in their treatment of surpluses. Some only redistribute votes obtained from the most recent distribution. Others take a random sample of the votes for the candidate being elected. In Australia it is common to redistribute all votes, giving all of them the same new weight, which is then calculated in a slightly different way.

[9]There is a good argument for rounding down, because rounding up could directly cause the election of candidates who do not really have a quota.

C. Notation and Ballot Representations

We denote voter v's preference permutation by σ_v; $\sigma_v(i)$ is the ranking of candidate i (conversely, the first-preference candidate would be $\sigma_v^{-1}(1)$). We denote $E : G_1 \mapsto G_2$ the homomorphic encryption function (this is a randomized function), where G_1 is an additive group and G_2 a multiplicative group. The encryption function satisfies, for every x and y: $E(x) \oplus E(y) = E(x + y)$. We denote the corresponding threshold decryption function D (for clarity, we ignore the randomness and keys in the presentation). We define $\bigoplus$ to be to $\oplus$ what $\sum$ is to $+$.

We regard the ballot representations as tables, with columns corresponding to candidates/preferences and rows corresponding to different things (depending on the representation). Table I describes the candidate-order representation. The Weight row in this table only contains a single value (the encrypted weight for the entire ballot). Table II describes the preference-order representation. The Weight row contains an encrypted weight for each candidate (column); the weight of the first-preference candidate is the ballot weight, and the weights for the other candidates are zero. Table III describes the first-preference representation. Table IV describes the candidate-i-elimination representaton (used to eliminate candidate i). In this representation, an extra "Eliminated" row is added that contains an encrypted "flag" for each candidate: this is an encryption of 1 for the candidates who are being eliminated, and an encryption of 0 for the other candidates. The Preference and Candidate rows in this representation are both encrypted (although the plaintext values for the Preference row are publicly known).

D. Tally Protocol

The tally protocol is an iterative protocol, with the same overall structure as a standard STV tally. The top-level description is given as Protocol 1a. The subprotocols for performing the different tasks are Protocol 1b for computing the first-preference tallies, Protocol 1c for reweighting votes and Protocol 1d for candidate elimination. In turn, these use "ballot conversion protocols" as subroutines to convert the ballot from one representation to another. The ballot conversion protocols are described in Section II-E.

E. Ballot Conversion Protocols

Protocols 2, 3, and 4 convert between different ballot representations. We denote the operation of encrypting (separately) each value in a row by "encrypting a row" (i.e., we replace the row with a new row, such that the i^{th} value in the new row is the encryption of the i^{th} value in the old row). In the same way, we refer to separately (threshold) decrypting each value in a table row as "(threshold) decrypting a row". In the protocols below, encrypting a row can always be performed deterministically (e.g., using publicly known randomness), so that public verification of encryption is easy.

TABLE I: "Candidate-Order" Ballot for Voter v

Candidate	1	2	$\cdots$	m
Preference	$P_{v,1} = E(\sigma_v(1))$	$P_{v,2} = E(\sigma_v(2))$	$\cdots$	$P_{v,m} = E(\sigma_v(m))$
Weight	$W_v = E(w_v)$			

TABLE II: "Preference-Order" Ballot for Voter v

Candidate	$E(\sigma_v^{-1}(1))$	$E(\sigma_v^{-1}(2))$	$\cdots$	$E(\sigma_v^{-1}(m))$
Preference	1	2	$\cdots$	m
Weight	$W_v = E(w_v)$			

TABLE III: "First-Preference" Ballot for Voter v. This voter's current first preference is candidate i. All other weights are 0.

Candidate	1	$\cdots$	i	$\cdots$	m
Preference	$P_{v,1} = E(\sigma_v(1))$	$\cdots$	$P_{v,i} = E(\sigma_v(i))$	$\cdots$	$P_{v,m} = E(\sigma_v(m))$
Weight	$W'_{v,1} = E(0)$	$\ldots$	$W'_{v,i} = W_v$	$\cdots$	$W'_{v,m} = E(0)$

TABLE IV: "Candidates C Elimination" Ballot for Voter v. The encrypted "Eliminated" flag is set for every candidate in C, zeroed for all other candidates.

Candidate	$C'_{v,1} = E(\sigma_v^{-1}(1))$	$\cdots$	$C'_{v,i} = E(\sigma_v^{-1}(i))$	$\cdots$	$C'_{v,m} = E(\sigma_v^{-1}(m))$
Preference	$P_{v,1} = E(1)$	$\cdots$	$P_{v.i} = E(i)$	$\cdots$	$P_{v.m} = E(m)$
Eliminated	$X_{v,1} = E(0)$	$\cdots$	$X_{v,i} = E(1)$	$\cdots$	$X_{v,m} = E(0)$
Weight	$W_v = E(w_v)$				

Protocol 1a STV Tally

$q \leftarrow \left\lfloor \frac{\#\text{valid votes}}{\#\text{seats}+1} \right\rfloor + 1$ // Compute q, the quota required for election.
while the number of remaining candidates is greater than the number of open seats **do**
  Compute First-Preference Tallies (cf. Protocol 1b).
  if at least one candidate was elected (had tally greater than q) **then**
    Let S be the set of candidates with tally above q.
    Announce that every candidate in S was elected.
    Reweight Votes for Elected Candidates S and quota q (cf. Protocol 1c).
    Eliminate all candidates in S from ballots (cf. Protocol 1d).
  else // No candidate was elected
    Let i be the candidate with the least tally
    Announce that i was eliminated.
    Eliminate $\{i\}$ from ballots (cf. Protocol 1d).
Announce that the remaining candidates were elected.

Protocol 1b Compute First-Preference Tallies

Input: All ballots in candidate-order form (cf. Table I)
for every voter $v \in \{1, \ldots, n\}$ **do**
  Convert ballot v to first-preference representation. // cf. Table III, Protocol 2
for every candidate $i \in \{1, \ldots, m\}$ **do**
  $T_i \leftarrow \bigoplus_{v=1}^{n} W'_{v,i}$ // Homomorphically add encrypted weights corresponding to candidate i.
  Threshold decrypt tally $t_i \leftarrow D(T_i)$.

Protocol 1c Reweight Votes for Elected Candidates in S with quota q

Input: All ballots in first-preference form (cf. Table III)
for every candidate $i \in S$ **do**
  $s_i \leftarrow 1 - \frac{q}{t_i}$ // Compute the transfer value for i.
  Choose $a_i/d_i \approx s_i$ // Approximate s_i.
Let $d' \leftarrow \text{lcm}(d_i : i \in S)$ // Compute the common denominator.
for every voter $v \in \{1, \ldots, n\}$ **do**
  for every candidate $i \in S$ **do**
    $W'_{v,i} \leftarrow a_i \frac{d'}{d_i} \otimes W'_{v,i}$ // Homomorphically multiply the i^{th} encrypted weight by the numerator of the corresponding scaling factor.
  for every candidate $j \notin S$ **do**
    $W'_{v,j} \leftarrow d' \otimes W'_{v,j}$ // Homomorphically multiply the j^{th} encrypted weight by the common denominator of the scaling factors for the elected candidates.
  $W_v \leftarrow \bigoplus_{i=1}^{m} W'_{v,i}$ // Homomorphically sum the encrypted weights for all candidates to get the new encrypted weight for vote v (note that at most one will be non-zero).

We assume we have a verifiable shuffle protocol that can shuffle arbitrary encrypted tuples; given a table consisting of encrypted values, the shuffle protocol outputs a new table created from the original by rerandomizing the encrypted values and randomly permuting the columns. We refer to this operation as "shuffling table columns".

F. Dealing with incomplete permutations

So far we have assumed that all votes are complete permutations. However, some jurisdictions do allow voters to stop before numbering all candidates. Following Heather [12], we could add a "STOP" candidate who is never elected

Protocol 1d Eliminate Candidate set C

Input: All ballots in candidate-order form (cf. Table I)

1: **for** every voter $v \in \{1, \ldots, n\}$ **do**
2: Convert v's ballot to Candidate C elimination form (cf. Table IV, Protocol 3).
3: **for** every candidate $\ell \in \{1, \ldots, m\}$ **do**
4: $P_{v,\ell} \leftarrow P_{v,\ell} - \bigoplus_{i=1}^{\ell} X_{v,i}$ // Homomorphically update the preference list to eliminate the tagged candidates ($X_{v,i} = E(1)$ for all $i \in C$).
5: Convert v's ballot to modified candidate-order form. (cf. Table I, Protocol 4). // it's modified because the preference list is no longer a permutation.
6: Remove every candidate in C (and the corresponding encrypted preference) from the ballot.

Protocol 2 Convert from Candidate-Order Ballot to First-Preference Ballot

1: Encrypt the candidate row.
2: Shuffle the table columns.
3: Threshold decrypt the preference row.
4: Sort columns in preference order
5: Add a "Weights" row, such that $W_{v,1} \leftarrow W_v$ and $\forall i > 1 : W_{v,i} \leftarrow E(0)$.
6: Encrypt the preference row.
7: Shuffle the table columns.
8: Threshold decrypt the candidate row.
9: Sort the columns in candidate order.

Protocol 3 Convert from Candidate-Order Ballot to Candidate C Elimination Ballot

Input: A ballot from voter v in candidate-order form (cf. Table I)

1: Add an "elimination-tag" row to the candidate-order ballot, so that the tag $X_{v,i}$ is $E(1)$ if $i \in C$, and otherwise $X_{v,i}$ is $E(0)$. // Note that the encrypted tags can use publicly known randomness.
2: Encrypt the candidate row.
3: Shuffle the table columns.
4: Threshold decrypt the preference row.
5: Sort the table columns by preference.
6: Encrypt the preference row.

Protocol 4 Convert from Candidate C Elimination Ballot to (modified) Candidate-Order Ballot

1: Shuffle the table columns.
2: Threshold decrypt the Candidate row.
3: Sort the table columns by candidate.

or eliminated, then place this candidate as the explicit next preference of every incomplete vote, and then pad out the rest of the vote (for privacy reasons) with the other candidates listed in some order. This effectively turns an incomplete vote into a complete permutation for the sake of tallying. This technique would work for Shuffle-Sum, with the obvious modifications to ensure that the STOP candidate was not elected or eliminated. We might also have to recompute the quota as the total number of votes diminishes—the exact way to do this varies from one STV jurisdiction to another, and would be as easy (or difficult) to do here as in any other STV setting, since we have a continuing tally of the total weight of exhausted votes.

III. Some variants

A. Faster eliminations: Table-Sum

The most expensive part of the scheme presented above is candidate elimination, because it requires converting from candidate-order form to candidate-elimination form and back again, each conversion using a few shuffles. In this section we present an alternative data structure which allows for very efficient construction of the candidate-order ballot after each elimination. The downside is that the initial data structure is large, $O(m^2)$ rather $O(m)$ in size. If almost all the candidates have to be eliminated (as they often do in practice), then this still takes less space and time in total for tallying. However, the efficiency of provably placing these votes on the bulletin board in the first place should be considered—depending on the scheme for doing so, this optimisation may or may not be an improvement overall.

One way of verifiably generating the right input, based on Heather's modifications to Prêt à Voter [12], is contained in Appendix A. Perhaps there is some alternative based on public checking of the input and encryption of paper ballots. This is clearly inferior from a security perspective, but it would be very simple to add on to the existing Australian electoral processes without rolling out an end-to-end verifiable voting scheme.

It remains crucial that all votes be valid and complete permutations. However, a proof of this fact is a side-effect of the conversion from candidate to preference order (or vice versa).

Recall that m is the number of candidates. The *Table-of-comparisons* ballot form consists of a weight W and an $m \times m$ matrix $\mathbf{V}$ with each cell separately encrypted. The diagonal of the matrix is unimportant and can be omitted. For non-diagonal values (with $i \neq j$), the interpretation of the matrix is that

$$\mathbf{V}_{ij} = \begin{cases} E(-1) & \text{if candidate } i \text{ is preferred to candidate } j, \\ E(0) & \text{otherwise.} \end{cases}$$

Candidates are eliminated by simply ignoring the corresponding row and column. The Candidate-order ballot form can be recovered efficiently by homomorphic addition (cf. Protocol 5), because the preference that should be given to candidate j is the sum over all continuing candidates i of $\mathbf{V}_{ij}$ (plus 1, so that preference counting starts from 1).

Protocol 5 Convert from Table-of-comparisons ballot to candidate-order ballot

Input: All ballots in Table-of-comparisons form

1: **for** every voter $v \in \{1, \ldots, n\}$ **do**
2: **for** every continuing candidate $j \in \{1, \ldots, m\}$ **do**
3: $P_{v,j} \leftarrow E(1) - \bigoplus_{i \neq j} \mathbf{V}_{ij}$ // Homomorphically add the column of comparisons to that candidate.

Protocol 6 Eliminate Candidate i using Table-of-comparisons ballot

Input: All ballots in Table-of-comparisons and candidate-order form (cf. Table I)

1: **for** every voter $v \in \{1, \ldots, n\}$ **do**
2: **for** every continuing candidate $j \in \{1, \ldots, m\}$ **do**
3: $P_{v,j} \leftarrow P_{v,j} \oplus \mathbf{V}_{ij}$ // Homomorphically update the preference list to eliminate candidate i.

This gives us a fast alternative to Protocol 1d, shown in Protocol 6.

In this way, redistributions can be performed without revealing which votes are being redistributed and without doing any mixing. An example is shown in Figure III.1. (The values are shown in cleartext but would be encrypted).

1) Performance comparison: A detailed evaluation of the efficiency of the table-of-comparisons form is contained in Appendix B. It allows faster tallying when almost all candidates are elected or eliminated, but the inefficiencies of our best-known input method outweigh those benefits.

2) Dealing with incomplete permutations in Table-Sum: Again Heather's "STOP" candidate idea works perfectly well here. Indeed, we can even omit the STOP candidate's row in the table, because it is never elected or eliminated.

Fig. III.1: An example of a vote being redistributed. Candidates are numbered from 1 to 5. All values are encrypted. Squares marked $\times$ are ignored. This vote represents a first choice of candidate 3, then candidate 5, then 4, 1 and 2. We show what happens when candidate 4 is eliminated. Note that the two least-preferred candidates are moved up in the ranks, while the first two preferences are unchanged. This process can be repeated for all elected or eliminated candidates.

Vote $\mathbf{V} =$

$\times$	-1	0	0	0
0	$\times$	0	0	0
-1	-1	$\times$	-1	-1
-1	-1	0	$\times$	0
-1	-1	0	-1	$\times$

Candidate order ballot. (Sum columns and add 1.)

$s =$

4	5	1	3	2

Eliminate candidate 4. Update s by adding row 4.
Candidate order ballot:

$s =$

$4 \oplus -1$	$5 \oplus -1$	$1 \oplus 0$	$\times$	$2 \oplus 0$

$=$

3	4	1	$\times$	2

B. Hiding the running tallies

This section describes how the authorties can prove which candidate deserves to be elected or eliminated at each step without revealing any of the tallies, except for demonstrating what the reweighting factor should be. This is considerably more computationally expensive than simply revealing the appropriate values, and in most circumstances it is probably unnecessary to hide this information. However, both the running tallies and the precise one-off tallies can reveal the absence of particular permutations, and hence provide an opportunity for coercion. Before we describe how to hide this information, we analyse the extent of the problem caused by revealing it.

1) Examples of coercion based on limited information: The following examples demonstrate that coercion is still possible even when a lot of information is hidden, if the coercer can still infer the absence of some permutations. This justifies our (rather computationally intensive) approach rather than the simpler alternatives mentioned above. In each case, the extent of the problem depends on the situation—there may be many environments in which the risks described here are acceptable and the advantages of a simple protocol overwhelming. The most important variable is the number of candidates. The more candidates there are, the more effective are the coercion strategies described here. Our motivating example is the Australian federal Senate elections, which often have more than 70 candidates in large states. There are usually about 50 candidates who are very unlikely to get a seat, and both the coercer and the voters know this, so the coercer can use them to encode a voter's identity. We call these $unlikely$ candidates.

a) Coercion when only the "important" preferences are revealed: Recent comments on the Irish election [28] have suggested revealing only those preferences that are actually used. This does not solve the coercion problem, because in multi-seat STV, many votes have many of their preferences used. The coercer could ask the voter to put the coercer's favourite candidate c_{coercer} first, followed by some particular permutation of unlikely candidates. If c_{coercer} wins a seat (and presumably they have a good chance of doing so, or coercing voters would be pointless) then the vote will be redistributed to a series of candidates that have been or will be eliminated. This means that most of the vote's preferences will be publicised. There are a very large number of possible votes of this form (about $50!$), so coercion is still a serious problem.

In some jurisdictions, including Ireland, surpluses are redistributed by randomly sampling some of the votes for the elected candidate. This makes this particular kind of coercion less effective, but it is still vulnerable to a closely related kind: the coercer demands that the voter put c_{coercer} *after* the list of unlikely candidates. This is a riskier strategy, but still likely to succeed even with many coerced voters (say about 1%). If none of the unlikely candidates are elected, then the coerced voter's vote passes to c_{coercer} with full weight and has all previous preferences revealed.

b) Coercion when partial tallies are revealed: Existing schemes for the secure counting of preferential votes all reveal

each candidate's tally after each (re)distribution. This can reveal the absence of a certain permutation: if the elimination of candidate c_1 does not increase the tally of candidate c_2, then the coercer can infer that there was no vote in which the highest continuing candidate was c_1 and the next was c_2.

This form of coercion only works if there are are reasonable number of *hopeless* candidates, that is, candidates which, when eliminated, are unlikely to effect the tally of many other candidates. In the last Australian federal election, 27 candidates for the Victorian Senate seats received fewer than 100 first-preference votes. When these were eliminated, it was common for many of the other tallies to remain constant, even after several candidates had been eliminated.

Let H be the number of hopeless candidates—we will assume $H > 20$. Here are some examples of how a coercer could make voters pass their preferences to candidate c_{coercer}:

1) Choose a hopeless candidate h for each coerced voter, and ask them to cast a vote that starts with $(h, c_{\text{coercer}}, \ldots)$. Then check, when h is eliminated, whether the tally of c_{coercer} increases. This could catch H voters with reasonable probability, if the hopeless candidates are eliminated before c_{coercer} is elected.
2) Just like Example 1, but coerce 100 times as many voters, randomly choose $H-1$ of them to be checked in the same way as Example 1 using $H-1$ of the hopeless candidates, and demand that the rest cast a vote of $(h_H, c_{\text{coercer}}, \ldots)$ (where h_H is the hopeless candidate who isn't being used to check the other voters). Compared to Example 1, this has the potential to catch 100 times as many voters, each with $1/100$ the probability. But this approach is less effective if coerced voters have the opportunity to collude and determine that they have been told to start their ballots with the same hopeless candidate.
3) Just like Example 1, but ask the voters to put the hopeless candidate *after* c_{coercer}. This could catch H voters with reasonable probability, if the hopeless candidates are eliminated after c_{coercer} is elected.
4) Demand, from each coerced voter, a different pair (c_1, c_2) of hopeless candidates, to be followed by c_{coercer}. When c_1 is eliminated, check that c_2's tally increases. Based on empirical analysis of the last Australian election, a coercer in a large state could coerce about 1000 voters and check nearly half of them. (Of course, the voter could deceive the coercer partially by submitting a vote with the correct prefix of hopeless candidates, but not following it with c_{coercer}.) Details of this analysis are in Appendix C-A.

Obviously many of these ideas could be combined.

c) Coercion when one tally is revealed with too much precision: Suppose we retain weights and tallies to many decimal places, and reveal final tallies (the ones when a candidate gets elected or eliminated) to many decimal places. Suppose that a coercer wants a voter to vote for candidate c_1 in first place, then candidate c_2. Suppose that c_1 is elected first and their votes redistributed, then c_2's tally is published. Suppose it happens to be an integer to 7 decimal places. Depending on the transfer value for c_1, this tally for c_2 may or may not reveal much useful information. For example, if the transfer value was $1/2$, then all the coercer can infer is that an even number of voters passed their preferences from c_1 to c_2. It is probably plausible that two did. However, for other transfer values, the coercer can be quite confident that no voter put c_1 first and then c_2. Extending the example, if the transfer value is $1/p$ for some prime p, then c_2's tally being an integer implies that the number of voters who put c_1 first and then c_2 is a multiple of p. If p is large, the only reasonably likely multiple could well be zero. Even if there is some small probability that p or $2p$ voters did so, it is far more likely that the voter disobeyed. A reasonable coercer could not be expected to pay up after that.

The extent of this problem depends on the probability distribution of all votes. Again the probabilities involved are small, but not negligible, and could possibly be used to coerce a small number of voters and discredit the election.[10] See Appendix C-B for detailed analysis. For a practical example, in the 2004 Australian Federal election, in the state of Victoria, 15 candidates' tallies did not increase when the first two elected candidates' votes were redistributed. Since the transfer values were 0.67533384 and 0.60324735, this fact would have been evident from their tallies alone, at least with some degree of confidence, even if running tallies were not revealed.

2) How to hide running tallies: The idea behind this section is very simple: to prove that a certain candidate deserves a seat, the authority need only prove that that candidate's total is more than a quota; to prove that a candidate deserves to be eliminated, they need only to prove that nobody has a quota and that that candidate has the lowest tally.

To simplify the description, we consider only a single authority; if we assume none of the authorities collude with a coercer, we can construct an efficient multiple-authority version by having the all the authorities learn the exact tally and then letting one of them perform the proof. There are several different ways for the (single) holder of a private key to prove that an encrypted value lies in a certain range [29], [30], [24].

We break ties deterministically, based on candidates' index number. Schemes based on randomised tiebreaking can easily be accommodated by choosing a random tiebreaking order before the beginning of tallying. Our main motivation is to avoid revealing that there was a tie.[11]

The first and most complicated protocol is for proving that a candidate deserves a seat. We have to prove that their tally is at least a quota, and we also want to prove what the transfer value should be without revealing any more precise information about the tally. Suppose candidate c_{win} won an excess of x votes over quota q. Then the candidate wins a seat and their preferences should be redistributed after having their weights multiplied by a factor of $x/(x+q)$. We wish to approximate

[10] Again the random sampling method used in Ireland is not susceptible to this particular problem.

[11] There are a variety of other common tie-breaking rules, which we do not accommodate. The PRSA rules (common in Australia) specify that the candidate who was most recently behind in the count gets eliminated. The Electoral Reform Society (UK) rules specify the candidate who was behind earliest in the count gets eliminated first.

that factor as some integer a over d, rounding down. [12] In Section II we allowed any way of (publicly) approximating the transfer value; now we will assume that the parameter d is set in advance, so the authorities have to prove only that they have chosen a correctly. This is described in Protocol 7. It proves that the tally is within the range of values for which a/d is the correct rounded transfer value. The protocol makes use of encrypted range proofs: these are zero-knowlege proofs that convince a verifier that an encrypted value is in a certain range. The proof protocols used depend on the encryption scheme; efficient range proofs exist for many of the common homomorphic encryption schemes (e.g., [30], [24]).

Protocol 7 Prove that candidate c's tally is at least a quota and what the transfer value should be.

Input: T_c, an encryption of the total first-preference vote for c, and also quota q and denominator d.

1: Announce the correct value of a which is the largest integer such that $a/d \leq x/(x+q)$.

2: Prove using a range proof that

$$\text{Decrypt}((d-a) \otimes T_c) \geq \text{Decrypt}(d \otimes E(q))$$

and

$$\text{Decrypt}((d-a-1) \otimes T_c) < \text{Decrypt}(d \otimes E(q)).$$

The current tallies are contained in the encrypted *tally vector* T, with T_j being an encryption of candidate j's tally, *i.e.* weighted total votes after reweighting and redistribution. [13] When L candidates have been elected and their votes redistributed, all tallies are d^L times the real tally (as in a traditional paper-based count). Obviously this means that the necessary quota is the real quota times d^L. Recall that n is the number of votes. The maximum tally at any point is nd^L, and the next power of 2 is $2^{\lceil \log_2(nd^L) \rceil}$, which we denote by *MaxTally*(L). Whenever we require a proof that some encrypted tally is nonnegative we use a range proof to show that the value is in the range $[0, \textit{MaxTally}(L)]$. [14]

When no candidate deserves a seat, the authorities must prove that no-one has a quota and then prove which candidate has the smallest tally. First we show how to prove that a certain candidate's tally is less than a quota. The authority proves that they do not have a quota, by subtracting that candidate's (encrypted) tally from a quota minus 1, then proving that the resulting encrypted value is non-negative.

It then identifies the candidate c_{min} that should be eliminated.

[12] We should round down since rounding up would increase the total value of all votes and risk an extra candidate gaining a quota (*i.e.*, electing more candidates than we should).

[13] In some literature, the word "tally" means a sheet containing lots of information; here, we use it only to mean one candidate's current total.

[14] The parameters must be chosen so that *MaxTally*(L) is always less than half the group size, otherwise this range proof is meaningless. The main problem occurs with too many seats. The worst case in Australia is the NSW Legislative Council election, with about 4 million voters and 21 seats. Then with $d = 1000$ the maximum plaintext is $nd^{seats-1} \approx 4 * 10^6 * 1000^{20} \approx 2^{221}$, which is well within range of 1024-bit Paillier (or Damgård-Jurik) encryption, but may be too large for 160-bit EC El-Gamal.

Protocol 8 Prove that candidate c's tally is less than a quota.

Input: T_c, an encryption of the total first-preference vote for c. Also quota q. Recall that L is the number of candidates elected before this step.

1: Prove in Zero Knowledge using a range proof that

$$\text{Decrypt}(E(q) \ominus E(1) \ominus T_c) \in [0, \textit{MaxTally}(L)].$$

Recall that we refer to candidates by an index number, and break ties by index number, so there are different facts to be proved for the other candidates' tallies, depending on whether the other candidate has a higher or lower index number. For each continuing candidate c with a higher index number than c_{min}, the EC proves that c has a strictly higher tally. Similarly, for each candidate with a lower index number than c_{min}, the EC proves that its tally is greater than or equal to that of c_{min}.

Protocol 9 Prove that candidate c_{min} has the lowest tally.

Input: T_c, an encryption of the total first-preference vote for c. Also quota q. Recall that L is the number of candidates elected before this step.

1: **for** each continuing candidate c with a higher index number than c_{min} **do**

2: Prove in Zero Knowledge using a range proof that

$$\text{Decrypt}(T_c \ominus E(1) \ominus T_{c_{\text{min}}}) \in [0, \textit{MaxTally}(L)].$$

3: **for** each continuing candidate c with a lower index number than c_{min} **do**

4: Prove that

$$\text{Decrypt}(T_c \ominus T_{c_{\text{min}}}) \in [0, \textit{MaxTally}(L)].$$

We can now give the complete protocols for proving that a certain candidate deserves to be eliminated or elected. An important part of the protocol is proving that other candidates do not have a quota. It suffices to prove this only for continuing candidates who will win a seat but haven't yet. At first glance a similar proof seems necessary for candidates who will not eventually win a seat, but it is not. If the candidate is eliminated, then at that point they will be proven to have the smallest tally, which must be less than a quota. If they are not eliminated, they will remain at the conclusion of the count when $\#\textit{seats}$ quotas have been subtracted from the total. Because of the careful definition of the quota, it is impossible for them to have a quota at that point. Since tallies do not decrease, either of these cases implies that the candidate could never have had a quota. In practice this is a significant saving because often the number of candidates is much greater than the number of seats.

The first protocol, Protocol 10, is for proving which candidates deserve to be elected and with what transfer values. This protocol should be considered as an alternative to step 5 in Protocol 1a.

The proof that a candidate deserves to be eliminated, Protocol 11, consists of proving that no one has a quota, then that c_{min} has the lowest tally. It should be considered as an

Protocol 10 Election: Prove which candidates should be elected and with what approximate transfer values.

1: For each continuing candidate c, let T_c be an encryption of the total first-preference vote for c. Let q be the quota. Recall that L is the number of candidates elected before this step.
2: **for** all continuing candidates c with at least a quota **do**
3: prove using Protocol 7 that c has a quota and what the appropriate transfer value approximation is.
4: **for** all candidates c' who will eventually win a seat but have not yet **do**
5: prove using Protocol 8 that c' does not have a quota.

alternative to step 10 in Protocol 1a.

Protocol 11 Elimination: Prove that candidate $c_{\min}$ should be eliminated.

1: For each continuing candidate c, let T_c be an encryption of the total first-preference vote for c. Let q be the quota. Recall that L is the number of candidates elected before this step.
2: **for** all continuing candidates c' who will eventually win a seat but have not yet **do**
3: prove using Protocol 8 that c' does not have a quota.
4: Prove, using Protocol 9, that $c_{\min}$ has the lowest tally.

3) Dealing with incomplete permutations while hiding tallies: This is the only section in which the STOP candidate modification doesn't automatically work. If the quota is held constant throughout the computation, then the proofs above are still valid. However, many STV jurisdictions apply (often very complex) rules for decreasing the quota as votes are exhausted. In some of these cases, the proofs above may not suffice—the details would depend on the exact rules being applied. It would of course be possible for the authority to prove the total weight of exhausted votes - this is just the total "vote" for the STOP candidate.

IV. Analysis

A. Security

Our main security claim is that the public transcript of the Shuffle-Sum tally scheme is a non-interactive zero-knowledge proof (in the random-oracle model) that the encrypted input votes correspond to the published result (consisting of the elected candidates, eliminated candidates, running tallies, and transfer values).

Below, we sketch a proof of this claim. We consider separately the soundness and zero-knowledge properties. Note that both the claim and the proof-sketch apply to the Table-Sum tally scheme as well.

a) Soundness: A proof system is said to be *sound* if a verifier will not accept an incorrect claim with more than negligible probability. In our case, the tally scheme is sound if corrupt authorities cannot produce an "incorrect" result that will pass verification with more than negligible probability (incorrect means that the result does not match the encrypted input votes).

The soundness of the tally scheme depends on the mixing scheme used and the zero-knowledge proofs of decryption. If the mixing scheme and decryption proofs are unconditionally sound, the entire tally is also unconditionally sound. If they are only computationally sound (i.e., sound only when the prover is computationally bounded), the scheme is actually a zero-knowledge argument rather than a proof.

We can consider, w.l.o.g, the case in which only a single, corrupt, election authority is the prover. Note that the only operations performed by the authority that cannot be reproduced directly from the public transcript are the decryptions and shuffles: encryption of constant values, homomorphic additions, and multiplications by a constant can all be performed using publicly available information. Thus, for the announced results to be incorrect, either the decryptions must be incorrect (in which case the decryption proof's soundness is broken), or the mixing must be incorrect (in which case the mixing scheme's soundness is broken).

To see that these are the only possible cases, consider the contrapositive: suppose all the decryptions are correct and all the mixing is performed correctly. Since the rest of the actions must be performed correctly (they are deterministic functions of the public transcript), proving that the entire tally is correct boils down to showing that the protocol is *complete*: that an *honest* authority following the protocol will output the correct tally. This can be done fairly easily by induction on the tally rounds.

b) Zero-Knowledge: A proof of some claim is said to be *zero-knowledge* if anything that can be learned from looking at a transcript of the proof can be learned just from the correctness of the claim. More formally, to show the protocol for tally scheme is zero-knowledge, we must prove there exists an efficient simulator that, given only the encrypted input votes and the final results, can output a transcript that is computationally indistinguishable from a real transcript with the same results. Note that our tally schemes are *computational* zero-knowledge proofs: the zero-knowledge property only holds when the verifier is computationally bounded. This restriction is reasonable in our case, since an unbounded verifier could completely violate voter privacy by breaking the encryption on the original ballots.

The tally protocol simulator, $\mathcal{S}$, works by using the simulators for the the mixing scheme and decryption proofs as subroutines: these sub-protocols are also zero-knowledge, hence must have their own simulators. The simulator for the decryption proof, given any ciphertext e and plaintext p, produces a transcript "proving" that p is a valid decryption of e (without actually knowing the decryption of e). This transcript is indistinguishable from that produced by a real prover, assuming the encryption scheme is semantically secure. In the same way, the simulator for the mixing scheme is given as input any two sets of ciphertexts, and produces a transcript "proving" one of the sets is a permutation of re-encryptions of the other.

The only decryption operations used in the tally protocols are decrypting an entire ballot's preference row, decrypting

a ballot's candidate row, and decrypting the first-preference tallies. Since each of the shuffles uses an independent, random permutation, the results of decrypting the preference or candidate row is always a random permutation of known values. Thus $\mathcal{S}$ can simulate the transcript by choosing random permutations itself, running the simulator for the mixing scheme to generate a fake shuffle proof, then running the simulator for the decryption proof to reveal the pre-chosen permutation. When the protocol calls for decrypting first-preference tallies, $\mathcal{S}$ uses the corresponding real first-preference tally from the results as input to the decryption simulator. From the zero-knowledge properties of the mixing and decryption proofs and the semantic security of the encryption scheme, the resulting transcript must be indistinguishable from a real transcript with the same encrypted input votes and results.

B. Computational requirements

We implemented the Table-Sum scheme (in the single-authority version), with the tally-hiding from Section III, using both standard and Elliptic Curve El Gamal. We then tested it on a subset of the votes from the last Victorian State election.[15] The Elliptic Curve version would be quite reasonable for verifying the whole of that election — extrapolating from our results, it would take a standard home computer about 10,000 hours to compute and produce a transcript of size about 400GB. Since the tally is highly parallelisable, the time is inversely proportional to the speed and number of processors performing the tally; five quad-core servers could perform the same computation in about 500 hours (this is acceptable for Australia because the data entry already takes weeks). Furthermore, verifying is also highly parallelisable (in particular, individual voters could download only a small part of the transcript — for example the votes for their own polling place — and verify that these ballots were all handled correctly).

Verifying an Australian federal election would require considerably more resources, because there are up to three times as many candidates and ten times as many voters, but would be quite feasible if a large number of computers were committed to the task.

V. Application to the hospitals and residents problem

In the United States, hospitals and candidate residents are matched annually by application of a stable matching algorithm. Each of the H hospitals provides a preferentially-ordered list of residents, and each of the R candidate residents provides a preferentially-ordered list of hospitals. Each resident will be assigned to one hospital, and each hospital has a quota of residents' places to be filled. An *assignment* is a function from residents to hospitals that sends each hospital the correct quota. (We are assuming that the sum of the quotas is equal to the number of residents.) The important correctness property of an assignment is *stability*, which means that no hospital can improve its situation by employing a resident who was assigned elsewhere but would prefer to have been assigned there.

Definition 1. An assignment A is *stable* if for all residents r, for all hospitals h that r preferred to $A(r)$, h employed only residents they preferred to r.

We will assume that the hospitals' lists are public (partly because this is good for transparency in many applications, and partly because our encrypted data structure is infeasible if we assume there are hundreds of items to be listed). However, there are good reasons to keep the residents' preferences secret, because residents may be unwilling to let their future employers learn that they were not their first choice. Indeed, residents may even lie about their preferences if they believe they will be publicised. Hence we will use the same table-of-comparisons data structure for residents' preferences that we used in section III-A for votes (omitting the weights), and say that for $i \neq j$,

$$\text{Decrypt}(\mathbf{V}_{ij}) = \begin{cases} -1 & \text{if hospital } i \text{ is preferred to hospital } j, \\ 0 & \text{otherwise.} \end{cases}$$

Now consider how the administrators can prove, without revealing any residents' preferences, that their assignment is stable. They can simply prove, for each resident, that the assignment satisfies the condition given above. Since the hospitals' lists are public, it will be immediately obvious that some hospitals employed only residents they preferred to r. Let H' be the other hospitals, *i.e.* those (other than $A(r)$) that employed residents they liked less than r. Then it suffices to show that for all $h \in H'$, r preferred $A(r)$ to h. This can be done with $|H'|$ (public) homomorphic additions and one (proven-correct) decryption: the administrator simply proves that

$$\bigoplus_{h \in H'} \mathbf{V}_{hA(r)} = 0$$

(where $\mathbf{V}$ is resident r's vote, *i.e.* list of preferences).

Our solution is much simpler and more efficient than previous work on private stable matching [31], [32], [33], for the simple reason that we have addressed a much easier problem by allowing one side's preferences to be public. Obviously this is not appropriate for all applications of private stable matching. However, in the particular example of real hospitals and residents there is a strong argument for transparency of hospitals' preferences, to show that they make their employment selections fairly.

VI. Conclusion and further work

This paper presents a way of tallying votes for multi-seat STV that protects voters against coercion if reasonable assumptions are made about the other voters' behaviour. It is intended to be added on as the final stage of an electronic voting scheme, though it could also be used after some other (paper-based) method of achieving an agreed-upon list of encrypted votes.

Although the scheme is specifically intended for Australian elections, it could also be appropriate for any electorate that

[15] The Victorian state elections have about 3 million voters divided into 8 regions, each of which elects 5 candidates. We used a 1.4GHz Pentium M with 512MB DDR.

uses STV with a reasonably small number of seats at a time (about a dozen at most). This includes Ireland, Malta, Scotland, Cambridge MA and a few others. The scheme would also work for single-seat STV, also called preferential voting and instant runoff voting. This is much more common than multi-seat STV, being used in Papua New Guinea, Fiji, and some local jurisdictions in the USA and the UK (including electing the mayor of London). The coercion problem we address here is caused by having a large number of candidates, which might happen even when electing only one person. The scheme could be considerably simplified for this case, because the complicated re-weighting step could be removed.

VII. Acknowledgements

Many thanks to Thea Peacock, Ron Rivest, Peter Ryan, Roland Wen, Andrew Conway and the anonymous reviewers for interesting discussions and helpful suggestions about this paper.

References

[1] V. Teague, K. Ramchen, and L. Naish, "Coercion-resistant tallying for STV voting," in *Proc. USENIXACCURATE Electronic Voting Technology Workshop*, 2008.

[2] R. Rivest and W. Smith, "Three voting protocols: Threeballot, VAV and twin," in *Proc. USENIX/ACCURATE electronic voting technology workshop (EVT 07)*, 2007.

[3] J. Otten, "Fuller disclosure than intended," *Voting Matters*, vol. 17, p. 8, October 2003, www.votingmatters.org.uk.

[4] J. C. Benaloh, "Verifiable secret-ballot elections," Ph.D. dissertation, Yale University, New Haven, CT, USA, 1987.

[5] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," *European transactions on Telecommunications*, vol. 8, no. 5, pp. 481–490, September-October 1997.

[6] T. Moran and M. Naor, "Receipt-free universally-verifiable voting with everlasting privacy," in *CRYPTO 2006*, ser. Lecture Notes in Computer Science, C. Dwork, Ed., vol. 4117. Springer-Verlag, August 2006, pp. 373–392.

[7] B. Adida and R. Rivest, "Scratch and vote," in *Proc. 5th ACM workshop on Privacy in the electronic society (WPES)*, 2006, pp. 29–40.

[8] D. Chaum, P. Y. A. Ryan, and S. A. Schneider, "A practical voter-verifiable election scheme," in *Proc. European Symposium on Research in Computer Security (ESORICS)*. Springer, 2005, pp. 118–139, lNCS 3679.

[9] C. A. Neff, "Practical high certainty intent verification for encrypted votes," October 2004, www.votehere.com/vhti/documentation/vsv-2.0.3638.pdf.

[10] D. Chaum, "Punchscan," `www.punchscan.org`.

[11] Z. Xia, S. Schneider, J. Heather, P. Ryan, D. Lundin, R. Peel, and P. Howard, "Prêt à Voter: all in one," in *Proc. Workshop on Trustworthy elections (WOTE 07)*, 2007, pp. 47–56.

[12] J. Heather, "Implementing STV securely in Prêt à Voter," in *Proc. 20th IEEE Computer Security Foundations Symposium (CSF)*, 2007, pp. 157–169.

[13] M. McMahon, "Verification of preferential voting system elections without publishing plain-text ballots," michael@hexmedia.com.

[14] J. Keller and J. Kilian, "A linked-list approach to cryptographically secure elections using instant runoff voting," in *ASIACRYPT 2006*, ser. Lecture Notes in Computer Science, J. Pieprzyk, Ed., vol. 5350. Springer-Verlag, December 2008, pp. 198–215.

[15] E.-J. Goh and P. Gollé, "Event driven private counters," in *Proc. 9th International Conference on Financial Cryptography and Data Security, FC 2005*. Springer, LNCS 3570, 2005, pp. 313–327.

[16] R. Wen and R. Buckland, "Mix and test counting in preferential electoral systems," University of New South Wales, Tech. Rep. 0809, 2008, proc. WISSEC 2008, http://www.win.tue.nl/ berry/wissec2008/.

[17] M. Clarkson and A. Myers, "Coercion-resistant remote voting using decryption mixes," Frontiers in Electronic Elections, September 2005.

[18] C. Crutchfield, D. Molnar, and D. Turner, "Approximate measurement of voter privacy loss in an election with precinct reports," NIST/NSF Workshop on Threat analyses for voting system categories, June 2006, vote.cs.gwu.edu/vsrw2006/papers.html.

[19] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections," in *STOC '94*, 1994, pp. 544–553.

[20] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic voting," in *WPES 05*, 2005.

[21] R. Canetti and R. Gennaro, "Incoercible multiparty computation," in *Proc. FOCS 96*, 1996, pp. 504–513.

[22] J. Benaloh, "Ballot casting assurance via voter-initiated poll station auditing," in *EVT'07: Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*. Berkeley, CA, USA: USENIX Association, 2007, p. 14.

[23] J. Benaloh, "Administrative and public verifiability: can we have both?" in *EVT'08: Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–10.

[24] M. J. Jurik, "Extensions to the paillier cryptosystem with applications to cryptological protocols," Ph.D. dissertation, University of Aarhus, Denmark, August 2003, http://www.brics.dk/DS/03/9/index.html.

[25] J. Groth, "A verifiable secret shuffle of homomorphic encryptions," in *PKC '03: Proc. of the 6th International Workshop on Theory and Practice in Public Key Cryptography*. London, UK: Springer-Verlag, 2003, pp. 145–160.

[26] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *USENIX Security Symposium*, 2002, pp. 339–353.

[27] I. Hill, "Edited comments on robert newland's suggestions," *Voting Matters*, vol. 23, pp. 3–9, February 2007, www.votingmatters.org.uk.

[28] B. A. Wichmann, "A note on the use of preferences," *Voting Matters*, vol. 18, pp. 11–13, June 2004, www.votingmatters.org.uk.

[29] W. Mao, "Guaranteed correct sharing of integer factorization with off-line shareholders," in *Proc. Public Key Cryptography*, 1998, pp. 27–42.

[30] F. Boudot, "Efficient proofs that a committed number lies in an interval," in *Advances in Cryptology, proc. EUROCRYPT 2000; Lecture notes in Computer Science Vol. 1807*, B. Preneel, Ed. Springer, 2000.

[31] P. Gollé, "A private stable matching algorithm," in *Proc Financial Cryptography (FC 06)*, 2006.

[32] M. Franklin, M. Gondree, and P. Mohassel, "Multi-party indirect indexing and applications," in *Advances in cryptology—ASIACRYPT 07*, 2007.

[33] ——, "Improved efficiency for private stable matching," in *The cryptographers' track at the RSA conference (CT-RSA)*, 2007.

[34] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *CCS '01: Proc. of the 8th ACM conference on Computer and Communications Security*. New York, NY, USA: ACM, 2001, pp. 116–125.

Appendix A
Vote input

Our counting method was not designed with a specific front-end e-voting system in mind. All that we require is that the votes be printed on the bulletin board in the format we use, in such a way that everyone believes the set of published votes matches the set cast in the election, and votes remain encrypted. Heather [12] describes one appropriate way to produce preference-order ballots, based on Prêt à Voter. These could be converted to candidate-order ballots using a (very slight) variant of Protocol 4, and could then be used immediately as input to Protocol 1a.

The rest of this section shows how to produce instead the table-of-comparisons ballot form from section III. We take the ballot construction from the point at which every vote is in prefence-order form. Unlike [12], we do not allow incomplete permutations in the input phase. The algorithm is given in Protocol 12:

For example, suppose we have the (preference-order) ballot with encrypted unknown candidate names:

Protocol 12 Construct table-of-comparisons ballot form

Input: A ballot in preference-order form (cf. Table II)

1: **for** every pair of (encrypted) candidate numbers $E(\sigma_v^{-1}(i)), E(\sigma_v^{-1}(j)))$ with $i < j$ in the ballot **do**
2: Construct tuples $(E(-1), E(\sigma_v^{-1}(i)), E(\sigma_v^{-1}(j)))$ and $(E(0), E(\sigma_v^{-1}(j)), E(\sigma_v^{-1}(i)))$. // This is a public computation, *i.e.* can be done without the secret key.
3: Shuffle the tuples.
4: Threshold-decrypt the 2nd and 3rd elements of each tuple.
5: Place the (encrypted) first element of each tuple into the row and column indicated by the 2nd and 3rd elements. // This is a public operation.

$[E(c_3), E(c_1), E(c_4), E(c_2)]$. Then it is a public operation to produce the triples:

$(E_{ZR}(-1), E(c_3), E(c_1))$ $(E_{ZR}(-1), E(c_3), E(c_4))$
$(E_{ZR}(-1), E(c_3), E(c_2))$ $(E_{ZR}(-1), E(c_1), E(c_4))$
$(E_{ZR}(-1), E(c_1), E(c_2))$ $(E_{ZR}(-1), E(c_4), E(c_2))$
$(E_{ZR}(0), E(c_1), E(c_3))$ $(E_{ZR}(0), E(c_4), E(c_3))$
$(E_{ZR}(0), E(c_2), E(c_3))$ $(E_{ZR}(0), E(c_4), E(c_1))$
$(E_{ZR}(0), E(c_2), E(c_1))$ $(E_{ZR}(0), E(c_2), E(c_4))$

These are then mixed (within each vote) and the candidate names decrypted. Producing the table-of-comparisons ballot is then a simple, public matter of placing the correct encryption in the correct candidates' row and column.

This construction preserves the security assumptions that were made in the body of the paper: the trustees can decrypt each vote if more than the threshold collude, but do not learn which vote corresponds to which voter (unless the mix servers all collude too). They otherwise do not learn any information about the contents of any votes.

The following section compares the efficiency of the alternative data structures.

APPENDIX B
EVALUATING THE EFFICIENCY OF THE VARIANTS FROM SECTION III

In this section we evaluate the efficiency of the table-of-comparisons data structure from Section III. It has a greater setup cost (in both time and space) but allows for the tallying algorithm to run much more efficiently, particularly candidate eliminations. Overall, it is still less efficient than the original form of the protocol, although they are similar when almost all candidates are elected or eliminated. In practice this is often, but not always, the case.

The most important complexity measure is the size of the data that must be downloaded from the bulletin board—it makes a huge difference whether a verifier is expected to download 1Gb or 100Gb. The time complexity is also important. However, both the generation and the verification of the proofs are highly parallelisable. Partial verification is also possible, in which a verifier could choose some fraction of the proofs to check. Hence we concentrate on the size of the bulletin board data.

With either variant, the only operations that actually take space on the bulletin board are shuffles and threshold decryptions, as these are the ones that must be done by the authorities (and hence require proof to be output). Other operations, such as sorting columns, performing homomorphic additions, *etc.* can be performed by anyone during verification and do not have to be explicitly written on the bulletin board.

A. The Complexity of the Protocol

Let there be n votes, m candidates, t mixes and r rounds of tallying. Let $s(k, m, t)$ be the size of the proofs provided by t servers of shuffling m k-tuples, preserving the order of plaintexts within each tuple.

1) Complexity of the Tallying Step: This assumes we already have n Candidate-order ballots and n Table-of-comparisons ballots and compares the space complexity of the two tallying algorithms. We have:

- r rounds of tallying, common to both approaches: $r\times$ Protocol 1b, with μ, the number of continuing candidates, running from m to $m - r + 1$.
  - Protocol 1b is: $n\times$ Protocol 2 plus μ threshold decryptions
  - Protocol 2 is: $s(2, \mu, t)$ plus $s(3, \mu, t)$ plus 2μ threshold decryptions
- r rounds of 'elimination': without the table-of-comparisons structure, this means $r\times$ Protocol 1d, again with μ running from m to $m - r + 1$.
  - Protocol 1d is: $n\times$ Protocol 3 plus $n\times$ Protocol 4.
  - Protocol 3 is: $s(3, \mu, t)$ plus μ threshold decryptions.
  - Protocol 4 is: $s(3, \mu, t)$ plus μ threshold decryptions.
- r rounds of 'elimination': with the table-of-comparisons structure, this means $r\times$ Protocol 6.
  - Protocol 6 is: 0 shuffles plus 0 threshold decryptions.

The "Tally" column of Table V gives the totals.

For many useful shuffle proofs, including the Neff and Groth shuffle protocols [34], [25], as well as Randomised Partial Checking [26], $s(k, n, t) = \alpha nkt$ for some constant α. Table VI summarizes the results if we let $s(k, m, t) = \alpha mkt$ (*i.e.* assume the complexity is linear) and $r = m$ (*i.e.* assume almost all candidates are elected or eliminated).

This shows that the total space used by the tallying proofs when all candidates are elected or eliminated is approximately halved by using the table-of-comparisons data structure. Unfortunately, the setup cost is likely to be higher than the saving, as shown in the next section.

2) Complexity of the Vote Input Step: Since our scheme is designed without a particular front end in mind, this section is only approximate, based upon the example vote input method given in Appendix A. The setup costs are summarized in Tables V (for the general case) and VI (after setting reasonable parameters). The details follow:

We begin at the stage when each vote is a list of encrypted candidate names in preference order, *i.e.* a Preference-order ballot form (Table II), without the weight or the cleartext preference numbers.

TABLE V: Efficiency in the general case. (**S**: Standard Shuffle-Sum; **T**: Table-of-comparisons)

	Tally		Setup	
	shuffle size	# decryptions	shuffle size	# decryptions
S	$\sum_{\mu=m}^{m-r+1} n(s(2,\mu,t)+3s(3,\mu,t))$	$(4n+1)r(2m-r+1)/2$	$ns(2,m,t)$	mn
T	$\sum_{\mu=m}^{m-r+1} n(s(2,\mu,t)+s(3,\mu,t))$	$(2n+1)r(2m-r+1)/2$	$ns(3,m^2-m,t)$	$2nm(m-1)$

TABLE VI: Efficiency when $s(k,m,t)=\alpha mkt$, $r=m$ (**S**: Standard Shuffle-Sum; **T**: Table-of-comparisons).

	Tally		Setup		Total	
	shuffle size	# decryptions	shuffle size	# decryptions	shuffle size	# decryptions
S	$11nt\alpha m(m+1)/2$	$(4n+1)m(m+1)/2$	$2\alpha nmt$	nm	$(11m+15)\alpha nmt/2$	$2m^2n+3mn+m^2/2+m/2$
T	$5nt\alpha m(m+1)/2$	$(2n+1)m(m+1)/2$	$3(m-1)\alpha nmt$	$2nm(m-1)$	$(11m-1)\alpha nmt/2$	$3m^2n-mn+m^2/2+m/2$

a) Cost of Producing Candidate-order Ballots: From a preference-order ballot, we can produce a candidate-order ballot by encrypting the preference row, performing a shuffle of cost $s(2,m,t)$, then decrypting the candidate row (using m threshold decryptions). Hence the cost is $ns(2,m,t)$ ciphertexts plus mn decryptions.

b) Cost of Producing Table-of-comparisons Ballots: We calculate the cost of Protocol 12 from Appendix A. The shuffle proof has size $ns(3,m^2-m,t)$. This is followed by another $2nm(m-1)$ threshold decryptions (of candidate names).

B. Conclusions

The calculations above measure space complexity of the data that would be written on the bulletin board, counting the space taken by shuffles and the number of threshold decryptions. The bottom line is that using candidate-order ballots alone is more efficient. However it is possible that, for some methods of vote input, for elections with roughly as many rounds as candidates, it might be worthwhile to introduce the table-of-comparisons ballot data structure.

APPENDIX C
ANALYSIS OF COERCION STRATEGIES

A. Coercion when partial tallies are revealed (Section III-B1b)

Suppose there are about H candidates who all receive a very small number of first-preference votes. Call them hopeless candidates. For simplicity, we begin by assuming that these candidates receive *no* votes other than the ones influenced by the coercer. We then argue that this assumption can be removed. The coercer will watch the partial tallies announced during the count to check whether the coerced voters have obeyed or not. This is a very beneficial form of coercion because all these votes are very likely to be passed to c_{coercer} fairly early in the tallying and with weight 1.

Consider the predicament of a voter v asked to choose the sequence c_1, $c_2, c_{\text{coercer}}, \ldots$, (and then allowed to follow it with anything). She knows that, with high probability, candidates c_1 and c_2. Suppose she disregards the coercer's demands entirely and instead votes for some other candidate. Then she will be detected if, at any point in the eliminations, a tally that was supposed to increase instead remains constant. If candidates c_1 and c_2 are eliminated in order (not necessarily consecutively), then both of those eliminations gives the coercer a chance to check for v's vote. For example, if c_1 has already been eliminated then when c_2 is eliminated, the coercer can check to see that c_{coercer}'s tally increases.

Now for the argument that it really isn't necessary to assume that only coerced voters vote for hopeless candidates. All we really need to assume is that (coerced voters believe that) a reasonable number of eliminations will not cause c_{coercer}'s tally to increase. Based on available data (from the Australian Senate) this assumption seems reasonable. We analysed this empirically for the last federal Australian election, in the state of Victoria. If the coercer chose the 33 candidates who received the lowest first-preference vote and gave each of $33 \times 32 = 1056$ coerced voters one pair of these candidates, then they could have checked 441 of them. Furthermore, the coercer could make it more plausible by fielding some colluding candidates who agreed not to campaign at all. They would probably still get some extra votes, but not very many. This would cost a bond (which is $AU500 in Australia) per candidate, probably a small expenditure for an organisation seriously trying to steal an election.

Of course, the voter could deceive the coercer only partially, submitting a vote with the correct prefix of hopeless candidates, but not following it with c_{coercer}. Although this would not satisfy our definition of avoiding coercion, the voter could still affect the election outcome in almost any way they wished. The interesting social/psychological question of how many voters would understand STV well enough to realise they could do this is beyond the scope of this paper. (But note that at least 5% can't even manage to write a list of consecutive numbers.)

B. Coercion when one tally is revealed with too much precision (Section III-B1c)

This section analyses the extent of the problem described in Section III-B1c, where tallies are revealed to too many decimal places. Recall that the setup involved a coercer who wants a voter to vote for candidate 1 in first place, then candidate 2. Candidate 1 is elected first and their votes redistributed, then candidate 2's tally is published. Suppose it happens to be an integer to many decimal places, enough that the coercer is confident it truly is an integer. Remember that the transfer value for candidate 1 is public, being computed as

$1-q/Tc_1$. Let n_{12} be the number of voters who put candidate 1 first and candidate 2 later (with no continuing candidates between).Then the coercer can infer that n_{12} is a multiple of $N = Tc_1/\gcd(Tc_1 - q, Tc_1) = Tc_1/\gcd(q, Tc_1)$.

As already discussed, this only causes a problem if N is fairly large, at least a few hundred—the exact value depends on the coercer's suspicion threshold and their probability distribution on other votes. On the other hand this is actually quite a probable scenario. Write q as $p_1^{a_1}p_2^{a_2}, \ldots, p_k^{a_k}$ with $p_1 > p_2 > \ldots > p_k$ all prime. Very often $\gcd(q, Tc_1)$ will be small.

For example the proportion of Tc_1 in the range, say $[q, 2q-1]$, such that $\gcd(q, Tc_1) \leq p_1$ is at least

$$\frac{\varphi(q) + \varphi(q/p_1) + \ldots + \varphi(q/p_k)}{q} =$$

$$(1 - \frac{1}{p_1}) \ldots (1 - \frac{1}{p_k})(1 + \frac{1}{(p_1 - i_1)} + \ldots + \frac{1}{(p_k - i_k)})$$

where i_k is an indicator variable of whether $a_i = 1$ or not.

For comparison, in the worst case that $Q = 2^2.3^2.5.7.11.13$ (for $q < 10^6$) the proportion 0.68. (The approximation above gives 0.46). Thus typically there will be a substantial proportion of Tc_1, for which n_{12} must be a multiple of $N \geq \frac{Tc_1}{p_1}$.

Of course, neither q nor Tc_1 is really a randomly distributed variable, so this calculation gives only a rough lower bound on the extent of the problem. It is large enough to be considered seriously. (Another approximation can be obtained from the well-known result that the probability of two randomly chosen numbers being coprime is $(6/\pi^2) \approx 60\%$)