# Coercion-Resistant tallying for STV voting

*Vanessa Teague, Kim Ramchen and Lee Naish*
*Department of Computer Science and Software Engineering*
*The University of Melbourne*
{vteague, lee}@csse.unimelb.edu.au, kramchen@gmail.com

## 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. In this paper, we solve this problem for the popular STV 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.

## 1 Introduction

In elections of all kinds it is important that voters are free of coercion, votes are tallied correctly and this is seen to be the case. 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, electronic voting systems must take particular care to prevent a voter from being able to prove to a coercer how they voted. This property is known as "receipt freeness" or "coercion resistance". Without it, a coercer can either promise to reward a voter 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 first-past-the-post voting, the best voting schemes are those that 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 and Cambridge MA. 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 cannot be used to identify individual voters. However, with STV a vote can specify any permutation of the candidates; this has much greater information content. Hence the "short ballot assumption" [RS07] 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 are 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 co-

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

ercer's favourite candidate first. Although this method of coercion requires work for the coercer prior to the vote (so a mapping from voters to permutations can be established), it is feasible and the risk has been described in the voting literature [Ott03]. The problem has created a tradeoff between verifiability and coercion-resistance for paper-based STV elections,[2] but it applies even more strongly to electronic systems in which votes are printed on a bulletin board. Complete disclosure of all ballots exposes the voters to coercion, but incomplete exposure can make verification impossible without cryptography.

Our approach is to perform the entire tallying algorithm on encrypted votes. We use homomorphic encryption to tally the votes' first preferences. The main shortcoming of our scheme is that there is a single electoral authority who learns the contents of each vote, though it does not learn the correspondence between voters and their votes. This means that we are trusting the authority not to collude with the coercer in an Italian attack. The authority produces a transcript in which each step of the tallying process can be independently verified while revealing very little information, thus reducing the opportunities for coercion. We define a new notion of coercion-resistance for STV and prove that, under reasonable assumptions, the scheme can be parameterised to be coercion resistant. 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 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 4.3 for details.

This section gives an overview of STV and existing work on coercion-resistant STV tallying. Section 2 gives examples of coercion that can still occur even under (some of) these coercion-resistant schemes. Section 3 presents our main contribution, an algorithm for provably correct STV tallying that reveals very little unnecessary information. Analysis appears in Section 4, with our new definition in Section 4.1 and the sketch of a security proof in Section 4.2.

[2]Two elections ago, the Australian Electoral Commission revealed the exact numbers of voters who select the ticket (*i.e.* recommended vote) of a major party, but released only the first preference of those who choose their own permutation. In practice this was usually enough to verify 3 or 4 of the 6 seats. Following the last election, and under pressure from voters' organisations advocating transparency, the AEC released complete data.

## 1.1 Tallying using STV

There are many slightly different sets of rules for tallying according to this general method. We will describe the method in general terms and defer discussion of technicalities to Section 3.2.1.

Assume we have *cands* candidates, *seats* seats to be filled and $n$ votes. At any point during the tallying process some candidates have been declared *elected* and some have been *eliminated*; the other candidates are known as *continuing* candidates. Initially all candidates are continuing. The algorithm terminates when *seats* candidates have been elected or *cands* $-$ *seats* have been eliminated. A candidate is elected if they obtain at least a *quota* of votes. The quota is usually $\lfloor n/(\mathit{seats}+1)+1 \rfloor$. This is just large enough to ensure it is impossible to elect $\mathit{seats}+1$ candidates. Initially all votes are distributed according to their first preference and all have weight 1.

The algorithm consists of repeatedly (re)distributing votes according to the vote's highest preference which is for a continuing candidate. Each candidate's tally is the sum of the weights of the votes that are distributed to them.

- After a distribution, any candidate with a quota or more is elected. Candidates with strictly more than a quota have their *surplus* redistributed. Votes for a candidate who receives an excess of $x$ votes above quota $Q$ have their weights multiplied by the *transfer value*, $x/(x+Q)$ before they are redistributed to the next continuing candidate. For example, if a candidate obtains $1.5$ quotas, all votes for that candidate redistributed and given $\frac{0.5}{0.5+1} = \frac{1}{3}$ their earlier weight.
- If no candidate gets a quota, the candidate with the lowest total is declared eliminated. All their votes are redistributed without adjusting their weights.

There are many variations,[3] The details of our implementation are described in Section 3.2.1.

## 1.2 Prior work

There has been great progress in recent years on universally verifiable election schemes. Some schemes [CGS97, MN06, AR06] 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 [CRS05, Nef04, Cha]. Of these, Prêt à Voter [CRS05] easily incorporates preferential voting

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

[XSH+07], and most of the others could be modified to support it. While none of these schemes introduce a coercion problem, neither do they solve the preexisting coercion problem for STV. In particular, they all end with a step that decrypts all the votes and then publicly tallies them. The aim of this paper is to devise a tallying step for STV that can be used at the end of some other coercion-resistant electronic voting scheme such as these 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 a number of existing cryptographic schemes for verifiable STV tallying that limit the information revealed. Heather [Hea07] describes how to implement STV counting securely in Prêt à Voter. McMahon [McM] and Goh & Gollé [GG05] describe structures for secure STV tallying that are not attached to a particular electronic voting scheme. None of these allows the re-weighting necessary for the multiple-seat case, though a recent unpublished scheme by Benaloh and Moran [BM] does. More subtly, every one of these works reveals every candidate's running tally after every redistribution, which still leaves open the possibility of coercion. This is described in Sections 2.2 and 2.3. (Though [BM] could probably be modified to hide this information in the same way we do. Since the details of this scheme are unpublished as yet, we do not know how it compares in efficiency or security to ours.) A similar coercion problem was addressed for the Condorcet voting scheme in [CM05], 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.* [CMT06] consider the privacy violations of publically-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*, which means that a voter cannot prove to the coercer how she voted, even if she wants to. Often this is used interchangably with the term *coercion resistance*, introduced in [JCJ05], though sometimes a distinction is made between a coercer who can communicate with the voter before the election (requiring coercion resistance) and one who cannot (requiring receipt-freeness, which is then a weaker property). In this paper we concentrate on the stronger requirement, and call it coercion resistance. We assume the existence of an untappable channel between a voter and the authorities, implemented as a private voting booth at a polling place, so the coercer cannot communicate with the voter during the election. In this sense our definition is weaker than that of [JCJ05]. A definition by Moran and Naor [MN06] (which extends [CG96] by Canetti and Gennaro) uses an *ideal functionality*. They define a scheme to be coercion resistant if any successful coercion attack against it would also succeed against the ideal functionality. Since they deal only with first-past-the-post elections, they simply choose an ideal functionality that reveals 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 one 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. In Section 4.1, we provide a definition of coercion resistance that is complementary to Moran and Naor's, in that it deals directly with whether coercion is possible given that the coercer receives certain information. It is based on one by Okamoto [Oka97]. In order to prove that an end-to-end voting scheme were coercion resistant, one could first prove that it implemented a particular ideal functionality according to the coercion-resistance definition of [MN06], then prove that that functionality was coercion resistant in the sense we define here.

In Section 4 we argue informally that our algorithm only reveals certain information, then we prove that that functionality is coercion resistant according to our definition.

## 2 Examples of coercion

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.

## 2.1 Coercion when only the "important" preferences are revealed

Recent comments on the Irish election [Wic04] 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 $c_{\text{coercer}}$ first, followed by some particular permutation of unlikely candidates. If $c_{\text{coercer}}$ 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 $c_{\text{coercer}}$ *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 $c_{\text{coercer}}$ with full weight and has all previous preferences revealed.

## 2.2 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 $c_{\text{coercer}}$:

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 $c_{\text{coercer}}$ increases. This could catch $H$ voters with reasonable probability, if the hopeless candidates are eliminated before $c_{\text{coercer}}$ 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 catches 100 times as many voters, each with $1/100$ the probability.

3. Just like Example 1, but ask the voters to put the hopeless candidate *after* $c_{\text{coercer}}$. This could catch $H$ voters with reasonable probability, if the hopeless candidates are eliminated after $c_{\text{coercer}}$ is elected.

4. Demand, from each coerced voter, a different pair $(c_1, c_2)$ of hopeless candidates, to be followed by $c_{\text{coercer}}$. 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 only partially, submitting a vote with the correct prefix of hopeless candidates, but not following it with $c_{\text{coercer}}$.) Details of this analysis are omitted due to space constraints.

### 2.3 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.[4] 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.

## 3 The algorithm

### 3.1 The main idea

Our scheme is based on an encryption scheme with an additive homomorphism, that is, given two encrypted values $c_1$ and $c_2$ one can easily compute a ciphertext $c_1 \oplus c_2$ that decrypts to the sum of the two values. We define $\bigoplus$ to be to $\oplus$ what $\sum$ is to $+$.

Let *cands* be the number of candidates. A vote consists of a weight $w$ and a *cands* $\times$ *cands* 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

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

The vote can be summarised in a *vote summary* vector $s$ of which the $j$-th element $s_j$ is

$$s_j = -\bigoplus_{i \neq j} \mathbf{V}_{ij}$$

The vote summary is (an encrypted form of) the vote traditionally cast in STV, except that the counting starts from zero: the most preferred candidate gets 0, the next gets 1, and so on until the least-favoured candidate gets *cands* $-$ 1. The vote summary can be updated very efficiently upon each elimination or election by performing a homomorphic addition with the row of the candidate who was eliminated. That is, for all continuing candidates $j$, upon deleting candidate $i$, assign

$$s_j := s_j \oplus \mathbf{V}_{ij}$$

This means that redistributions can be performed without revealing which votes are being redistributed and without doing any mixing. An example is shown in Figure 1. (The values are shown in cleartext but would be encrypted). The authority then transforms the vote summary into a form in which each candidate's first preference vote can be tallied homomorphically, then declares and proves which candidate(s) should be elected or eliminated. The difficulty in the multiple-seat case is redistribution after an election, which requires multiplying the weights on votes by non-integer factors. The homomorphic nature of the encryption scheme gives us (reasonably) efficient multiplication of the encrypted value by an integer. Division is a problem, because in general it requires finding roots, a hard problem. We avoid division as follows. Approximate each redistribution factor by an integer $a$ divided by some fixed denominator $d$. For example, to approximate it to 3 decimal places, choose $d = 1000$. (This is the method recommended by the Proportional Representation Society of Australia.) Each vote begins with a weight of 1, but upon a candidate's election all votes for that candidate have their weights multiplied by $a$ and all other votes' weights are multiplied by $d$. The EC generates a fresh encryption of the new weight and proves it correct with standard (honest-verifier) zero knowledge proofs (Step 3). The algorithm is described compeletely in Section 3.

[4] Again the random sampling method is not susceptible to this particular problem.

Vote $\mathbf{V} =$

| | | | | |
|---|---|---|---|---|
| × | -1 | 0 | 0 | 0 |
| 0 | × | 0 | 0 | 0 |
| -1 | -1 | × | -1 | -1 |
| -1 | -1 | 0 | × | 0 |
| -1 | -1 | 0 | -1 | × |

Vote summary:

$s =$

| | | | | |
|---|---|---|---|---|
| 3 | 4 | 0 | 2 | 1 |

Eliminate 3. Update $s$ by adding row 3.
Vote summary:

$s =$

| | | | | |
|---|---|---|---|---|
| $3 \oplus -1$ | $4 \oplus -1$ | $0 \oplus 0$ | × | $1 \oplus 0$ |

$=$

| | | | | |
|---|---|---|---|---|
| 2 | 3 | 0 | × | 1 |

Figure 1: An example of a vote being redistributed. Candidates are numbered from 0 to 4. All values are encrypted. Squares marked × are ignored. This vote represents a first choice of candidate 2, then candidate 4, then 3, 0 and 1. We show what happens when candidate 3 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.

## 3.2 Technical background and notation

### 3.2.1 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 [Hil07] 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.[5]
- Another variable is the precision used to record tallies and votes' weights. As described in Section 2.3, this introduces its own coercion problems. Hence we leave this as a parameter in our scheme and show how to choose an appropriate value depending on what is known about voting patterns.
- 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[6].
- If two or more candidates have a quota, we elect them all and redistribute each vote to the next continuing candidate.
- We assume that all votes are complete permutations.

The last assumption is important—at several points the proof of correct tallying depends upon it. However, some jurisdictions do allow voters to stop before numbering all candidates. Heather [Hea07] suggests including a "STOP" candidate who is never elected or eliminated. If we were to do this, we would have to modify some of the tallying proofs and introduce an explicit count of how many votes had been exhausted, so that the quota could be appropriately reduced.

### 3.2.2 Cryptography background

We require a semantically secure encryption scheme for which there exist efficient operations $\oplus$ and $\ominus$ on ciphertexts so that decryption obeys the following homomorphisms:

$$\text{Decrypt}(c_1 \oplus c_2) = \text{Decrypt}(c_1) + \text{Decrypt}(c_2) \quad (1)$$

$$\text{Decrypt}(c_1 \ominus c_2) = \text{Decrypt}(c_1) - \text{Decrypt}(c_2) \quad (2)$$

This automatically allows multiplication by a constant, which we denote by $\otimes$.

$$\text{Decrypt}(exp \otimes c_1) = exp \times \text{Decrypt}(c_1) \quad (3)$$

**Notation** We define $\bigoplus$ to be to $\oplus$ what $\sum$ is to $+$.

We also need the following well-known zero knowledge proofs. In this paper, we require a trusted source of random challenges. This could be obtained from a

[5] 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.

[6] There are a variety of other common tiebreaking 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.

trusted beacon, or generated jointly by a number of participants. The easiest method is the Fiat-Shamir heuristic [FS87], hashing the input to the proof. We can then use proofs that are merely honest-verifier zero knowledge.

**Proof 1** The proof that a ciphertext encrypts a particular value, or that two ciphertexts encrypt the same (unrevealed) value [CP93, Sch91].

**Proof 2** The proof that an encrypted number lies in a certain range [Mao98]. The range must be from 0 to some power of 2.

**Proof 3** The proof that at least one of a list of vectors of encrypted numbers is the all-zero vector [CEvdG88, CDS94].

El Gamal encryption satisfies these requirements, and we have implemented our scheme using both standard and Elliptic Curve El Gamal. The latter is much more efficient, being feasible for Australian State elections and close to feasible for federal ones. See Section 4.3 for details. The Paillier encryption scheme also has the required homomorphism and efficient proofs, but we have not implemented it yet.

**exponential El Gamal** Let $p$ and $q$ be large primes such that $q|p-1$. (Here, "large" means about 1024 bits). Let $g$ be an element of order $q$ in $\mathbb{Z}_p^*$. All arithmetic is done in the group $\mathbb{Z}_p^*$. The private key is a secret $s$. The public key is $h = g^s$ and the parameters $p, q$ and $g$ are also public. To encrypt value $a$, first generate a random value $r \in \{1, \ldots, q\}$, then send the tuple $(g^r, g^a h^r)$. To decrypt a ciphertext $(x, y)$, the holder of the private key computes $\text{Decrypt}((x, y)) = \log_g(y/x^s)$.

This is just like ordinary El Gamal, except that each message is of the form $g^a$ and we take the decryption to be $a$, not $g^a$. Obviously, this requires solving the discrete log problem in general, so we ensure that $a$ is either known to the decryptor in advance or taken from some range that is small enough to enumerate efficiently.

With ciphertexts $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, and an integer *exp*, define the homomorphic operations by:

$$\begin{aligned} c_1 \oplus c_2 &= (x_1 x_2, y_1 y_2) \\ c_1 \ominus c_2 &= (x_1/x_2, y_1/y_2) \\ exp \otimes c_1 &= (x_1^{exp}, y_1^{exp}) \end{aligned}$$

This is easily seen to satisfy Equations 1, 2 and 3. Using this homomorphism to tally encrypted votes was first suggested in [CGS97].

**exponential Elliptic Curve El Gamal** An alternative is exponential El Gamal encryption over the Elliptic curve $E_{a,b}/\mathbb{F}_p$ where $p$ and $|E_{a,b}|$ are large primes (about 160 bits). This gives a semantically secure encryption scheme based on the hardness of the Decision Diffie-Hellman problem [Bon98].

Specifically, let $m$ be the order of the group and $G$ be a generator. The secret key is a number $s \in [0, m-1]$ and the public key is the point $H = sG$. To encrypt $a \in [0, m-1]$, choose a random $r \in [0, m-1]$ and form $(rG, aG + rH)$. Once again, to decrypt this we need $a$ to be either known to the decryptor or taken from a small range[7]. The homomorphic operators are simply the usual group operators. It is straightforward to replace ordinary El Gamal with EC-El Gamal in the zero knowledge proofs we use.

EC-ElGamal uses about $1/6$ the space of ElGamal over $\mathbb{Z}_p^*$. However, care must be taken to ensure that the greatest plaintext to be encrypted fits within the range $[0, m-1]$, which we assume to be about 160 bits. This is the case for all the STV elections we know of. (Of course a larger range can be chosen, but the savings are proportional to it).

## 3.3 Security model and assumptions

We begin with a public table of votes. Everyone agrees that this is the true list of all valid votes cast in the election. The encrypted votes are not linked to voter identities in any way. (A technical point: this means that the voters can't know the randomness used to encrypt their votes, or they are subject to coercion before we begin.) One way of verifiably reaching this point, based on Heather's modifications to Prêt à Voter [Hea07], 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.

The votes are encrypted with the public key of the Electoral Commission (EC hencefoward), the semi-trusted authority which carries out the tallying. The EC is trusted not to be a coercer, and not to reveal plaintext votes to the coercer or anyone else. It is *not* trusted to count correctly, and it does not learn the correspondence between votes and individual voters.

[7] By encrypting $aG$ we sidestep the problem of ensuring that every possible message corresponds to a point on the curve, which would be a problem if we were using standard EC-ElGamal and trying to encrypt a message $a$ as $(rG, A + rH)$ for some point $A$.

All votes must be valid and complete permutations. In our complete system described in Appendix A, this fact is proved by the Electoral Commission before counting commences. The system's security is based on the semantic security of (Elliptic Curve) El Gamal encryption.

## 3.4 Making a tallyable vote

Recall the main data structure, with the vote and vote summary, described in Section 3.1. Each vote has a weight, $w$, that is encrypted.

For counting, the vote summary is transformed into a *tallyable vote* $t$ in which the $j$-th element $t_j$ satisfies

$$\text{Decrypt}(t_j) = \begin{cases} w & \text{if candidate } j \text{ is the most-preferred} \\ & \text{continuing candidate.} \\ 0 & \text{otherwise.} \end{cases}$$

The EC produces this tallyable vote and then proves its correctness. It suffices to give (honest-verifier) zero knowledge proofs that

1. $\text{Decrypt}(\bigoplus_j t_j) = w$, and
2. for all $j$, either $\text{Decrypt}(t_j) = 0$ or $\text{Decrypt}(s_j) = 0$.

These are straightforward applications of Proof 1 and Proof 3 respectively.

## 3.5 The tallying algorithm

The current tallies are contained in the encrypted *tally vector* $\mathbf{T}$, with $\mathbf{T}_j$ being an encryption of candidate $j$'s tally, *i.e.* weighted total votes after reweighting and redistribution. [8] When $E$ candidates have been elected and their votes redistributed, all tallies are $d^E$ times the real tally (as in a traditional paper-based count). Obviously this means that the necessary quota is the real quota times $d^E$. Recall that $n$ is the number of votes. The maximum tally at any point is $nd^E$, and the next power of 2 is $2^{\lceil \log_2(nd^E) \rceil}$, which we denote by *MaxTally*$(E)$. Whenever we require a proof that some encrypted tally is non-negative we use Proof 2 and prove that the value is in the range $[0, \textit{MaxTally}(E)]$. [9]

The tallying algorithm is paramaterised by $d$. Although it is written as a series of computations for the EC, many of the steps can be performed by any observer—these are preceded by **[public]**. Obviously such computations do not reveal any information, even when performed by the EC.

Each step is either an election or an elimination, followed by a redistribution. The algorithm is as follows:

### 3.5.1 Tally(d)

1. **[public]** For all continuing candidates $c$, set $\mathbf{T}_c$ to be an encryption of the total first-preference vote for $c$. This can be computed from the tallyable votes by homomorphically adding: set the tally for each candidate $c$ to be

$$\mathbf{T}_c := \bigoplus_{t \text{ a tallyable vote}} t_c$$

   Recall that the EC knows the decryption of each value in $\mathbf{T}$ (without having to run the decryption algorithm).

2. **Elimination, if no candidate has a quota** If no one has a quota, the EC first proves this as follows: For every continuing candidate who will win a seat but hasn't yet[10], the EC 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.

   More formally, let $Q$ be the quota. Recall that $E$ is the number of candidates elected before this step. To show that candidate $c$ does not have a quota,

   (a) **[public]** the EC forms the "encryption" with zero randomness of the quota required. Denote it by $\mathbf{Q}^*$. (In ordinary El Gamal this would be the tuple $\mathbf{Q}^* = (1, g^{d^E Q})$; in EC-El Gamal it would be $(O, d^E QG)$).

   (b) **[public]** the EC forms the "encryption" with zero randomness of one. Call it $\mathbf{1}$.

[8] 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.

[9] The parameters must be chosen so that *MaxTally*$(E)$ 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 a federal Senate election, with about 4 million voters and 6 seats. Then with $d = 1000$ the maximum plaintext is $nd^{seats-1} \approx 4 * 10^6 * 1000^5 \approx 2^{72}$, which is well within range.

[10] 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 *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.

(c) It then proves in Zero Knowledge using Proof 2 that

$$\text{Decrypt}(\mathbf{Q}^* \ominus \mathbf{1} \ominus \mathbf{T}_c) \in [0, \textit{MaxTally}(E)].$$

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

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 $c_{\text{min}}$, the EC proves that $c$ has a strictly higher tally as follows: it subtracts $c_{\text{min}}$'s (encrypted) tally from $c$'s minus 1 and proves that the result encrypts a non-negative number. More formally, to show that candidate $c_{\text{min}}$ has a strictly lower tally than anyone with higher index number, it proves in Zero Knowledge using Proof 2 for all candidates $c > c_{\text{min}}$, that

$$\text{Decrypt}(\mathbf{T}_c \ominus \mathbf{1} \ominus \mathbf{T}_{c_{\text{min}}}) \in [0, \textit{MaxTally}(E)].$$

Similarly, for each candidate with a lower index number than $c_{\text{min}}$, the EC proves that its tally is greater than or equal to that of $c_{\text{min}}$, *i.e.* that

$$\text{Decrypt}(\mathbf{T}_c \ominus \mathbf{T}_{c_{\text{min}}}) \in [0, \textit{MaxTally}(E)].$$

3. **Election, if at least one candidate has a quota**

   (a) For every continuing candidate who will win a seat but hasn't yet, the EC proves that they do not have a quota, using the same proof as in step 2. [11]

   (b) For each candidate that wins a seat, the weights of their votes must be updated before redistribution. Suppose candidate $c_{\text{win}}$ 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 that factor as some integer $a$ over $d$, rounding down. [12] The EC announces the correct value of $a$, then proves using Proof 2 that

$$\text{Decrypt}((d-a)\otimes\mathbf{T}_{cwin}) \geq \text{Decrypt}(d\otimes\mathbf{Q}^*)$$

and

$$\text{Decrypt}((d-a-1)\otimes\mathbf{T}_{cwin}) < \text{Decrypt}(d\otimes\mathbf{Q}^*).$$

This shows that the tally is within the range of values for which $a/d$ is the correct (rounded) transfer value.

   (c) For all votes, the EC will produce a new weight. If the vote had $c_{\text{win}}$ first, the new weight will be $a$ times the old; if not, it will be $d$ times the old. This is equivalent to re-weighting only those votes being redistributed, multiplying their weights by $a/d$. The trick is to do so without revealing which values are being changed. This works as follows. For each vote, with tallyable vote $t$,

      i. **[public]** the EC multiplies the encrypted weight $w_{\textit{old}}$ by $a$ by setting $w_{\textit{red}} = a \otimes w_{\textit{old}}$. This produces a weight that is equal to the correct new weight *if this vote is being redistributed.*

      ii. **[public]** the EC multiplies the encrypted weight $w_{\textit{old}}$ by $d$ by setting $w_{\textit{notRed}} = d \otimes w_{\textit{old}}$. This produces a weight that is equal to the correct new weight if this vote is *not* being redistributed.

      iii. The EC generates a new (encrypted) weight $w_{\textit{new}}$.

      iv. The EC proves in zero knowledge, using Proof 3 on $t$ and the vote summary $s$ that

$$\begin{aligned} (\text{Decrypt}(s_{c_{\text{win}}}) &= 0 \text{ and} \\ \text{Decrypt}(w_{\textit{new}}) &= \text{Decrypt}(w_{\textit{red}})) \\ \text{or} & \\ (\text{Decrypt}(t_{c_{\text{win}}}) &= 0 \text{ and} \\ \text{Decrypt}(w_{\textit{new}}) &= \text{Decrypt}(w_{\textit{notRed}})). \end{aligned}$$

This implies that either the vote is being redistributed and its weight was correctly changed, or it is not being redistributed and its weight effectively remained the same.

4. **Redistribution** The final step is for the EC to generate, for each vote, a new vote summary $s$ and a new tallyable vote $t$ as described in Sections 3.1 and 3.4, omitting the eliminated candidate.

5. Return to the tallying step, Step 1.

[11] Again, as in step 2, the proof is unnecessary for those who will not eventually win a seat.

[12] We have to round down - 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).

# 4 Analysis

## 4.1 Defining coercion-resistance

Having demonstrated that coercion can be effective even when only limited information is revealed, we now define coercion resistance more formally. We imagine an attack model in which the coercer communicates with the voter before and possibly after the election, and also receives all public information published during the election. In our case, this is at least the transcript of the public tallying process. The coercer is trying to make the voter cast a vote with some particular property, such as putting one party ahead of another, or putting a certain candidate first. Obvious special cases are that the coercer specifies the vote entirely, or tries to prevent someone from voting. Throughout the following discussion, we consider informal voting (including abstaining) to be a special kind of vote and incorporate it into our definition of coercion-resistance. The scheme is resistant to coercion if the voter can be confident that the coercer will be unsure of whether the voter obeyed their demand or not.

Even when the coercer is only attempting to coerce one voter, no voting scheme can be entirely secure against coercion because simply publicising the result could be enough to inform the coercer that the voter did not vote as required. For example, if the coercer demands a vote for candidate $c$ and the tally shows that nobody voted for them, then it is obvious that the voter disobeyed. This problem is exacerbated when the coercer knows some of the votes (because of fraud or because they are cast by political partisans). However, in a reasonably large election this sort of thing is unlikely to reveal disobedience decisively. Furthermore, the voter does not have to be absolutely certain that disobedience will be undetectable. She just has to consider the probability to be very high. Exactly how high depends on the voter's political preferences and the mode of coercion. For example, if she has a strong political preference against the coercer's party and the method of coercion is to offer her a small amount of money, then she may be willing to accept quite a high probability of being caught disobeying; if she is indifferent anyway and the coercer threatens to shoot her if she disobeys, she will require an extremely low probability of being caught.

The following definition assumes a voting system whose outcome is a symmetrical function of the votes. It assumes that a voter can "cheat" the coercer only by submitting a different vote, not, for example, by modifying the algorithm of some Internet voting scheme. (This is an assumption about the vote-input scheme that precedes our tallying step. It would have to be proven to implement an ideal functionality as in [MN06].) We define the scheme to be secure if the coerced voter can undetectably submit whatever vote she chooses, regardless of what the coercer intended. We also assume that the voter and the coercer have a common prior probability distribution $\Pi$ on the set of other votes. This may include, for example, knowing for sure that a certain number of voters will vote in a particular way. Define the coercer's view $\text{VIEW}_{coercer}$ to be everything that the coercer generates, computes or observes before, during, or after the election. This is a randomised function of the input votes. Given the view, the coercer tries to determine whether the voter voted as requested or not. The exact definition of VIEW is dependent on the scheme. For example, in an old-fashioned paper-based scheme with secret ballot boxes, the coercer's view would consist only of conversations with the voters before and after the election, and the public tally results. In an electronic voting scheme that had been proven coercion-resistant in the sense of [MN06], *i.e.* one that securely implemented an ideal functionality IDEAL, the coercer's view could consist of the view that the ideal adversary obtains in an interaction with IDEAL.

We define the preimage of a coercer's view to be the set of vote profiles (*i.e.* lists of votes of all voters) that are consistent with that view.

**Definition 4.1** *Let $n$ be the number of voters. Define the* $\text{preimage}(v, \text{VIEW})$ *of a vote $v$ and a view* VIEW *to be the set of votes for all $n-1$ voters that produce a view computationally indistinguishable from* VIEW *when the $n$-th voter votes $v$.*

Consider what happens when a coercer demands a vote $v_{obey}$ but the voter instead casts $v_{cheat}$. Let $V$ be the profile of all the other votes cast. Then the coercer's view is that produced by $v_{cheat}$ and $V$, that is, $\text{VIEW}(V, v_{cheat})$. The coercer is trying to detect whether the voter cast $v_{obey}$ or some other vote. To do this, it will first estimate the probability of this view being produced by an obedient voter. This is the probability of the preimage of the view, *i.e.* $Pr_{\Pi}\left(preimage(v_{obey}, \text{VIEW}(V, v_{cheat}))\right)$. Obviously, if $preimage(v_{obey}, \text{VIEW}(V, v_{cheat})) = \emptyset$ or $Pr_{\Pi}\left(preimage(v_{obey}, \text{VIEW}(V, v_{cheat}))\right) = 0$, then the coercer knows that the voter has disobeyed. However, there are other situations in which the coercer might still be very suspicious of disobedience: if the voter produces a view that would be very unlikely with the obedient vote, but quite likely otherwise, then the coercer may

believe that the voter disobeyed. Recall the example of high-precision tallies from Section 2.3.

We define the *likelihood ratio* of the obedient vote $v_{obey}$ and the disobedient vote $v_{cheat}$ for a particular view VIEW to be the ratio of the probabilities that the view was produced with each vote[13].

**Definition 4.2** *Let $n$ be the number of voters, $v_{obey}$ and $v_{cheat}$ be votes ($v_{obey}$ the one the coercer would like the voter to pick, and $v_{cheat}$ what the voter casts instead), and* VIEW *the coercer's view. Define the* likelihood ratio $L(v_{obey}, v_{cheat}, \text{VIEW})$ *to be*

$$L(v_{obey}, v_{cheat}, \text{VIEW}) = \frac{Pr_{\Pi}\left(\text{preimage}(v_{obey}, \text{VIEW})\right)}{Pr_{\Pi}\left(\text{preimage}(v_{cheat}, \text{VIEW})\right)}.$$

We assume that the coercer has some "suspicion threshold" $t_{sus}$. The coercer, after getting a certain view, computes the likelihood ratio of the obedient vote and every possible disobedient vote, and punishes the voter when any value is below $t_{sus}$. The scheme is secure from coercion if there is a low probability of producing an outcome that falls below the suspicion threshold.

**Definition 4.3** *Let $n$ be the number of voters, and $\Pi$ be a joint distribution on $n-1$ votes (those of the voters other than the one being coerced). Let $Pr_{V \leftarrow \Pi}()$ denote the probability when $V$ is randomly chosen according to $\Pi$. Then a voting system is* $(t_{sus}, p_{caught})$-coercion resistant *if for all votes $v_{obey}$ (demanded by the coercer), for all votes $v_{cheat} \neq v_{obey}$, the probability*

$$Pr_{V \leftarrow \Pi}\left(L(v_{obey}, v_{cheat}, \text{VIEW}(V, v_{cheat})) \leq t_{sus}\right)$$

*is less than or equal to $p_{caught}$.*

There are several ways to weaken this definition. The most obvious one is to set the suspicion threshold $t_{sus}$ to zero, so the disobedience only fails if the coercer is certain that the voter disobeyed. However, this seems too weak because of the weighted votes example mentioned in Section 2.3. Another weakening is to allow the coerced voter to collude with other voters to deceive the coercer. Finally, we could consider a coercer trying to coerce a whole group of voters simultaneously. We do not consider these weakenings in this paper.

[13]There is no good reason to take the ratio—we just need a function that compares the probabilities, increases in the first argument and decreases in the second. Ratio was the simplest and is quite commonly used in similar likelihood calculations.

## 4.2 Security Analysis

**Claim 4.4** *If the El Gamal cryptosystem is semantically secure, Tally(d) reveals only*

1. *The order of eliminations and elections, and*
2. *The range of the final tally of each candidate implied by the transfer value approximation in Step 3.*

**Proof:** A formal proof of this claim would consist of a reduction to an ideal functionality revealing only items 1 and 2. We do not present such a formal proof here, but we note that our techniques for hiding information are very standard. □

Let IDEAL TALLY(d) denote the ideal functionality described described in Claim 4.4. We prove coercion resistance for this ideal functionality. First we must make some assumptions about the probability distribution on everyone's votes, then we show that our scheme is coercion resistant.

Even with the ideal functionality, there are still opportunities for coercion if weights and tallies are reported with too much precision, as described in Section 2.3. Also, some low probability events still expose the absence of a particular permutation. For example, if an elected candidate has only slightly more than a quota, then the approximation $a/d$ reveals quite a lot of information about the tally. Fortunately, if the tallies are reported to relatively few decimal places, the coercer can catch a cheating voter only with low probability. Of course, we still need to make some assumptions about the coercer's uncertainty about everyone else's vote. A joint probability distribution $\Pi$ on $n-1$ votes induces a probability distribution on each partial tally (the tallies that would be obtained if only those $n-1$ votes were cast), and we make our assumptions based on that induced distribution. Informally, the idea is to define the distribution to be $(t_{sus}, d, \epsilon)$-*uncertain* if, given the information revealed according to Claim 4.4, for all votes that the coercer might demand, there is a probability of at most $\epsilon$ that the coercer's estimated likelihood that the voter cheated is greater than $t_{sus}$.

**Definition 4.5** *Define the transfer value approximation $approx(\mathbf{T}c)$ to be the rounded-down approximation from Step 3. For a given probability distribution $\Pi$ on $n-1$ votes, define the approximation $approx(\mathbf{T}c)$ for candidate $c$ elected during elimination and election order $O$ to be* $(t_{sus}, d)$-uninformative *if*

$$\frac{Pr_{\Pi}\left(approx(\mathbf{T}c) = \alpha\right)}{Pr_{\Pi}\left(approx(\mathbf{T}c) = \alpha + 1\right)} \geq t_{sus}$$

*and*

$$\frac{Pr_{\Pi}\left(approx(\mathbf{T}c) = \alpha + 1\right)}{Pr_{\Pi}\left(approx(\mathbf{T}c) = \alpha\right)} \geq t_{sus}$$

**Definition 4.6** *Define* $\Pi$ *to be* $(t_{sus}, d, \epsilon)$-uncertain *if the probability of getting at least one of*

1. *an elected candidate getting exactly a quota,*
2. *a tie for equal-lowest tally when nobody has a quota,*
3. *being within 1 of either of the first two occurrences,*
4. *an elimination and election order and at least one approximation that is not* $(t_{sus}, d)$*-uninformative, or*

*is at most* $\epsilon$*.*

**Theorem 4.7** *Suppose that the probability distribution* $\Pi$ *on others' votes is* $(t_{sus}, d, \epsilon)$*-uncertain. Then* IDEAL TALLY*(d) is* $(t_{sus}, \epsilon)$*-coercion resistant.*

**Proof:** The first 3 items of definition 4.6 imply that the probability of the coerced voter affecting the elimination or election order is at most $\epsilon$.

To see that no other votes are cheat-revealing, apply Claim 4.4. If the coerced voter's "cheat" does not affect the elimination or election order, then the coercer's only extra information is the approximation based on the final tally of each candidate who wins a seat. By Definition 4.5, this does not make the coercer suspicious (on the basis of the ratio of their probabilities with and without voter obedience). □

## 4.3 Computational requirements

We implemented the scheme using both standard and Elliptic Curve El Gamal, then tested it on a subset of the votes from the last Victorian State election. The Elliptic Curve version would be quite reasonable for verifying the whole of that election—it would take about 10,000 PC-hours to compute and produce a transcript of size about 400Gb. (This is acceptable because just the data entry already takes weeks.) Furthermore, both proving and verifying are highly parallelisable. 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.

The results for a standard PC running an election with 200 votes, 40 candidates, 5 seats and denominator $d = 1000$, over the elliptic curve defined by $E_{-3,383}/\mathbb{F}_p$ (where $p$ is a 160-bit prime) are as follows. The last three times include the time taken to read in the file of encrypted votes.

| | |
|---|---|
| Size of Ciphered Votes | 7.1 Mb |
| Transcript Size | 40.2 Mb |
| Time for EC to compute Election results | 12 mins |
| Time for EC to output Proof | 45.5 mins |
| Time to verify Transcript | 60.2 mins |

# 5 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.

It would of course be possible to reveal more information. For example, the initial (first-preference) tallies are used by the Australian Electoral Commission to determine public campaign funding. It would be easy to modify the scheme to reveal this, or to reveal partial information using range proofs, but we have not analysed the security implications of this.

The obvious next step is to try to distribute the secret key so that no single authority could decrypt ballots. A proof of correct decryption could be achieved without explicitly reconstructing the key, using the techniques of [CGS97]. The proof of equality of two encrypted values could also be adapted. The range proofs could be distributed using the techniques of [DFK+06] or [ST06] (which is based on Paillier encryption, to which our scheme could easily be adapted). However, we do not know how to do reweighting without all the authorities knowing which votes are being redistributed. This alone could be enough for successful coercion.

# 6 Acknowledgements

Many thanks to Thea Peacock, Tal Moran, Ron Rivest, Josh Benaloh, Peter Ryan and Andrew Conway for interesting discussions and helpful suggestions about this paper.

# References

[AR06] Ben Adida and Ronald Rivest. Scratch and vote. In *Proc. 5th ACM workshop on*

*Privacy in the electronic society (WPES)*, pages 29–40, 2006.

[BM] J. Benaloh and T. Moran. Shuffle-sum: a practical protocol for receipt-free STV tallying.

[Bon98] Dan Boneh. The decision diffie-hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium, LNCS 1423*, pages 48–63. Springer-Verlag, 1998.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO '94, LNCS 839*, pages 174–187. Springer-Verlag, 1994.

[CEvdG88] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of a discrete logarithm and some generalisations. In *Proc. EUROCRYPT '87, LNCS 304*, pages 127–141. Springer-Verlag, 1988.

[CG96] R. Canetti and R. Gennaro. Incoercible multiparty computation. In *Proc. FOCS 96*, pages 504–513, 1996.

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 8(5):481–490, September-October 1997.

[Cha] D. Chaum. Punchscan. `www.punchscan.org`.

[CM05] M. Clarkson and A. Myers. Coercion-resistant remote voting using decryption mixes. Frontiers in Electronic Elections, September 2005.

[CMT06] Christopher Crutchfield, David Molnar, and David 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.

[CP93] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Proc. CRYPTO '92, LNCS 740*, pages 89–105. Springer-Verlag, 1993.

[CRS05] 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)*, pages 118–139. Springer, 2005. LNCS 3679.

[DFK+06] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T.Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proc. TCC 2006. LNCS 3876*, pages 285–304. Springer, 2006.

[FS87] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proc. CRYPTO '86, LNCS 263*, pages 186–194. Springer-Verlag, 1987.

[GG05] Eu-Jin Goh and Philippe Gollé. Event driven private counters. In *Proc. 9th International Conference on Financial Cryptography and Data Security, FC 2005*, pages 313–327. Springer, LNCS 3570, 2005.

[Hea07] J. Heather. Implementing STV securely in Prêt à Voter. In *Proc. 20th IEEE Computer Security Foundations Symposium (CSF)*, pages 157–169, 2007.

[Hil07] I.D. Hill. Edited comments on robert newland's suggestions. *Voting Matters*, 23:3–9, February 2007. www.votingmatters.org.uk.

[JCJ05] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic voting. In *WPES 05*, 2005.

[Mao98] W. Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Proc. Public Key Cryptography*, pages 27–42, 1998.

[McM] Michael McMahon. Verification of preferential voting system elections without publishing plain-text ballots. michael@hexmedia.com.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer-Verlag, August 2006.

[Nef04] C. A. Neff. Practical high certainty intent verification for encrypted votes, October 2004. www.votehere.com/vhti/documentation/vsv-2.0.3638.pdf.

[Oka97] T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Proc. 5th international workshop on security protocols, LNCS 1361*, pages 25–35. Springer-Verlag, 1997.

[Ott03] J. Otten. Fuller disclosure than intended. *Voting Matters*, 17:8, October 2003. www.votingmatters.org.uk.

[RS07] Ronald Rivest and Warren Smith. Three voting protocols: Threeballot, VAV and twin. In *Proc. USENIX/ACCURATE electronic voting technology workshop (EVT 07)*, 2007.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[ST06] B. Schoenmakers and P. Tuyls. Efficient binary conversion for paillier encrypted values. In *Proc. EUROCRYPT 2006, LNCS 4004*, pages 522–537. Springer, 2006.

[Wic04] B. A. Wichmann. A note on the use of preferences. *Voting Matters*, 18:11–13, June 2004. www.votingmatters.org.uk.

[XSH+07] 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)*, pages 47–56, 2007.

## 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, it is impossible to link individual votes with the corresponding voter, and votes remain encrypted. This section shows one way to achieve this. It is based on Prêt à Voter, as modified by Heather [Hea07]. We take the ballot construction from part way along Heather's decryption process, the point at which every vote is a (randomly ordered) list of pairs $(p, \text{Encrypt}(c))$, with $\text{Encrypt}(c)$ being an encrypted candidate name, and $p$ an (unencrypted) preference. We have to modify Heather's process slightly even before this, so that $\text{Encrypt}(c)$ is doubly-encrypted, first with the public key of the EC and then with the public key of the *vote-construction authorities*, who share the key so that some threshold number of them is required for decryption. Unless there is collusion between the EC and more than the threshold number of vote construction authorities, the latter learn nothing about the contents of the votes. The EC learns the decrypted votes but, unless all the mix-servers collude with it, does not learn the correspondence between votes and individual voters. Unlike [Hea07], we do not allow incomplete permutations in the input phase. The steps are:

1. **re-encryption mix(es)** A series of mix servers mixes the votes and, for each vote, randomly permutes the list of pairs and re-encrypts the candidate names. They prove correctness using one of the standard mixing proofs.

2. **vote-construction authorities** The vote-construction authorities perform a shared decryption of the first layer of encryption on the candidate names. This can be done with a proof of correctness and without explicitly reconstructing the key, using the techniques of [CGS97].

3. **vote reconstruction [public]** For each vote, arrange the list in preference order (first preference first, then second, etc.). These become the row and column labels for a matrix with "don't care" along the diagonal, zeros above, and "-1" everywhere below. Then encrypt the cells of the matrix with the public key of the EC and some standard randomness (such as 0).

4. **vote de- and en-cryption** The EC mixes votes and, for each vote, decrypts the list of candidate names, permutes them so that they are in the canonical order, then permutes the rows and columns of the matrix so that each cell has the same row and column lables as it did before the label permutation, then re-encrypts each cell. This is proven correct with one zero knowledge proof.

This construction preserves the security assumptions that were made in the body of the paper: the EC does learn each decrypted vote, but does not learn which vote corresponds to which voter (unless the mix servers all collude with it). No other entity learns any information about the contents of any votes.