Pretty Good Democracy for more expressive voting schemes

James Heather[1], Peter Y A Ryan[2], and Vanessa Teague[3]

[1] Department of Computing, University of Surrey, Guildford, Surrey GU2 7XH, UK
j.heather@surrey.ac.uk
[2] Dept. Computer Science and Communications, University of Luxembourg
peter.ryan@uni.lu
[3] Dept. Computer Science and Software Engineering, University of Melbourne
vteague@csse.unimelb.edu.au

Abstract. In this paper we reconsider *Pretty Good Democracy*, a scheme for verifiable Internet voting from untrusted client machines. The original scheme worked for first-past-the-post elections. Here we show how PGD can be extended to voting schemes in which the voter lists the candidates in their order of preference. Our scheme applies to elections using STV, IRV, Borda, or any other tallying scheme in which a vote is a list of candidates in preference order. We also describe an extension to Approval or Range voting.

1 Introduction

Secure Internet voting wouldn't be difficult at all, if only the authorities tallying the election were perfectly trustworthy, nobody ever attempted to influence another person's vote, and every home PC was perfectly secure. Unfortunately, all of these problems remain open. There are various schemes for Internet voting [JCJ05, Adi08], which use cryptography to weaken or eliminate (some of) these assumptions. Here we concentrate on *Pretty Good Democracy* [RT09], which has the great advantage over other schemes of providing a proof of correct tallying while placing no trust whatsoever in the device used to cast the vote. (It has the consequent disadvantage of weaker coercion-resistance and integrity guarantees than some other schemes—see [RT09].)

The first version of PGD [RT09] was designed for elections in which the voter chose a single favourite candidate. However, many countries and many other organisations use voting schemes requiring the voter to list several (or all) candidates in their order of preference. For example, the Single Transferable Vote scheme (STV) is used in national elections in Australia, Ireland, Malta and Scotland. Instant Runoff Voting (IRV), which is the single-vacancy version of STV, is used in some local elections

in the USA, the UK, Australia, and many other countries. The Borda Count is used in certain political elections in Slovenia, and also in many other organisations, such as the Eurovision Song Contest. In this paper we extend PGD to allow voters to express their vote as a list of candidates in preference order. Any method could then be used to tally the votes, including existing solutions for the secure tallying of Borda [CM05] or STV/IRV votes [BMN+09, Hea07, TRN08] . We present several different schemes, one of which also encompasses Approval Voting or Range Voting, in which the voter gives a score to all candidates.

In Section 2 we review PGD 1.0 and explain why the obvious extension to more complex voting schemes doesn't work. The next three sections each contain a different extension with a discussion of pros and cons. In Section 3 the simplest method (Protocol A) is described, which is simple and secure but suffers from the disadvantage that each preference requires a separate interaction with the authorities. Protocol B, in Section 4, has the simplest voting experience, but somewhat complicated acknowledgement checking. Protocol C, in Section 5, is an approach based on a two-dimensional table, which allows votes that are ordered lists or approval or range votes.

1.1 Protocol Comparison

Figure 1 contains a comparison of the three protocols. "Single-step voting" means that casting a vote requires only one interaction with the authorities. "Single ack" means that there is only one Ack code—this is important because it means that the protocol is receipt-free even against a coercer who observes the ack return directly. "Number of preferences hidden on BB" means that observers can't tell from the bulletin board how many preferences each voter cast. This is sometimes important, because different jurisdictions have very different rules about how many preferences may or must be cast. Being able to check via the bulletin board is a useful feature for demonstrating vote validity, though it may make voters vulnerable to being coerced into casting fewer (or more) peferences than they wished.

Security properties Like PGD 1.0, all the protocols in this paper are receipt-free but not coercion-resistant—a voter can sell her code sheet before voting, but cannot prove after voting what vote she cast. The protocols with a single ack (A and C) are receipt-free even against a coercer who directly observes the ack return. Protocol B is receipt-free

Protocol	A	B	C
Single-step voting	×	✓	✓
Single Ack	✓	×	✓
Number of preferences hidden on BB	✓	×	×
Approval or Range Votes	×	×	✓

Fig. 1. Comparison of protocol features

only if the voter has a chance to generate a fake ack code list before the coercer observes it.

In all cases, it takes either a leak of the printed code sheet or a collusion of a threshold number of trustees (which, by assumption, cannot occur) to derive an appropriate Ack Code without the vote being properly registered on the bulletin board. The assumptions behind integrity are described in Table 2, in which "no" is good and "yes" is bad.

Vote can be undetectably manipulated by:	A	B	C
Cheating client who doesn't know the codes	No	No	No
Cheating client who knows the codes	Yes	No	Yes
Cheating client who knows the codes and the order of the candidates on the code sheet	n/a	Yes	n/a

Fig. 2. Comparison of protocol security properties

2 Review of PGD

Like other forms of Code Voting [Cha01], PGD assumes that each voter receives a *Code Sheet*, which is a list of candidate names and corresponding Vote Codes. An example is given in Figure 3.

Each voter sends the Vote Code for their chosen candidate to the central Vote Server. They could use any networked device for the transmission, including a home PC or mobile phone. Even a corrupted device is unable to substitute a different choice of candidate, because it does not learn the other codes. After sending the Vote Code, the voter waits to receive an *acknowledgement code*. In the original Code Voting scheme [Cha01], the printed code sheet contained a separate Ack Code for each candidate. In PGD [RT09] we argued that one Ack code per code sheet sufficed. Either way, the purpose of the Ack is to demonstrate to the voter that they communicated with the correct server and that it received

Candidate	Vote Code
Red	3772
Green	4909
Chequered	9521
Fuzzy	7387
Cross	2531
Ack Code: 8243	
Ballot ID: 3884092844	

Fig. 3. Example Vote Codes

the correct Vote Code. In PGD there was then a computer-verifiable proof of correct tallying, which could be publicised on a bulletin board.

The key innovation of the PDG scheme is that, in order to access the correct ack code, the voter server must invoke the cooperation of a threshold set of Trustees. The revealing of the correct ack code is thus a side-effect of the correct registration of a valid code.

2.1 Overview of ballot construction

The roles of the authorities in PGD are:

- A Voting Authority VA who generates the requisite number of vote codes and ack codes encrypted under the Trustees' public key, PK_T.
- A set of Clerks, who generate encrypted Vote Codes for each ballot, one version for the Bulletin Board and one for the printed code sheets.
- A Registrar who decrypts the ballots provided by the Clerks and prints the code sheets.
- A Returning Officer who distributes the code sheets to the voters.
- A Voting Server, who receives the votes, then posts the ballot ID and the encrypted vote code on the Bulletin Board along with a Zero Knowledge proof of knowledge of the plaintext.
- A set of Trustees, who work with the Voting Server to register the votes on the Bulletin Board and reveal the ack codes. They have shares of the secret key corresponding to the threshold public key, PK_T.
- A set of Auditors responsible for performing various types of audit, on the initial set-up, on the information posted to the Bulletin Board, *e.g.* the zero knowledge proofs, and verifying the anonymising mixes.

All of this is done on the bulletin board, except obviously the Registrar's decryption of the ballots and the distribution of the code sheets to voters.

For details of ballot interpretation, tallying and audit, see [RT09]. Briefly, the Clerks generate the encrypted ballots by successive shuffling, the encrypted Vote Codes are matched via plaintext equivalence tests, and the rest of the tallying is similar to Prêt à Voter. In this paper we present two protocols with single ack codes that can be published on the bulletin board, and one protocol with an ordered list of ack codes that must be decrypted and returned to the voter secretly.

2.2 An obvious extension to preference voting that doesn't work

The simplest extension would be for the voter simply to list their vote codes in preference order, and wait for the (single) return Ack. However, this is insecure because a cheating client or VS could simply rearrange the codes undetectably.

3 Protocol A: The simple solution

Another possibility is to use distinct Ack codes for each candidate, sent secretly to the voter in addition to the public one that is posted on the bulletin board (see Section 9.3 of [RT09]). The voter would have to send in each code in turn, then wait for the appropriate Ack to be received before sending in the next code, and so on.

3.1 Discussion

This is a secure and simple solution—it is impossible for a cheating client to switch vote codes or candidate acks undetectably, and it is easy for the voter to understand why. Its only shortcoming is that it could take some time for the authorities to generate and return the acks, during which time the voter has to wait. Furthermore, the security is undermined if a malicious client machine successfully persuades the voter to enter all their vote codes in one go without waiting for the intermediate acks, because the client could then apply the same rearrangement to both the vote codes and the ack codes.

4 Protocol B: Returning the Acknowledgement Codes in ballot order

In this protocol, the voter is provided with a set of preference codes as well as voting codes. Furthermore, the order of the candidates is randomised

on each code sheet in the manner of Prêt à Voter. Voting is a simple matter of sending in the vote codes in order of preference. The return acknowledgement should be a list of preference codes in the order the candidates appear on the code sheet. This is computed by the authorities without requiring any voter interaction. The main drawbacks with this are that it may be difficult for voters to understand how to check their acknowledgement codes, and that the integrity guarantee is not as strong as in the previous solution.

4.1 Security properties

For the scheme in Section 3, a cheating client or VS couldn't rearrange the vote without knowing the vote codes in advance. In that section, we could list the candidates on the code sheet in a canonical order. In this section each code sheet will have the candidates listed in a secret, random order. Our main security claim is:

Claim. A cheating client or VS (who doesn't know the codes) can swap two preferences undetectably only if it knows which two positions on the code sheet they correspond to.

A proof of this claim is contained in Section 4.4.

4.2 Voter Interface

The idea is to give each voter a code sheet with two lists of codes:

- a list of candidate codes in a random, secret order, and
- a list of preference codes in preference order.

The idea is that the voter submits their candidate codes in their order of preference, and receives as acknowledgement a list of preference codes in the order the candidates appear on their code sheet. For example, for the code sheet given in Figure 3, the voter might wish to vote "Chequered, Fuzzy, Green, Red, Cross", so they would enter codes $9521, 7387, 4909, 3772, 2531$ in sequence. At this point they have finished casting their vote, and if they are not interested in verifying their vote, they do not need to interact with the system any further.

They would then expect as acknowledgement a list of preference codes given in the order the candidates are printed on the code sheet. For the example preference codes in Figure 4, the first would be code W, (because Red is the first candidate in the order printed on the ballot paper, and the

Preference	Ack Code
1st	K
2nd	T
3rd	C
4th	W
5th	M
Ballot ID: 3884092844	

Fig. 4. Example Preference Codes

preference given to it was 4th), then C, K, T, M. Thus the voter should expect to receive the acknowledgement: $WCKTM$.

To assist the voter, we could provide a blank column alongside the candidate list. The voter writes the appropriate preference code for each candidate alongside the candidate. Then the acknowledgement code will be the sequence of letters read down the column.

4.3 Details of ballot construction, acknowledgement and tallying

Notation If σ and π are permutations on n items, then $\sigma \circ \pi$ is the permutation defined by $(\sigma \circ \pi)(i) = \sigma(\pi(i))$. If L is a (possibly encrypted) list, then L_i denotes the i-th element of L. Denote by $\pi(L)$ the idea of "applying" a permutation π to a list L, which means taking each element L_i in turn and copying it into position $\pi(i)$ in the new list. The result is $\pi(L) = L_{\pi^{-1}(1)}, \ldots, L_{\pi^{-1}(n)}$. It follows that the result of applying π and then σ to L is $(\sigma \circ \pi)(L) = L_{\pi^{-1}(\sigma^{-1}(1))}, \ldots, L_{\pi^{-1}(\sigma^{-1}(n))}$.

[x] means encrypted x. Actually, almost everything is encrypted, so the [] notation is just a reminder.

Building Blocks Numerous protocols exist for proving a shuffle of a list of ciphertexts. In [Ram09], efficient protocols are given for proving that the *same* shuffle has been applied to several lists, even if they are encrypted under different public keys. We will call this protocol *Shuf-par*.

Ballot Construction: The Bulletin-Board part We use a distributed ballot construction similar to that of PGD. Obviously we need full permutations rather than cyclic shifts. For each vote ID we need to produce a printed code sheet as described above. There are four different authorities, each of which could be performed by a single (trustworthy) individual, or (preferably) distributed among several.

1. The *ballot-construction authorities* produce the codes and a randomly-arranged encrypted version of each code sheet, on the bulletin board.
2. the *code-sheet authority* randomly reshuffles and then prints the code sheets. (The shuffling and decrypting can be distributed using standard techniques, but the printing is more difficult to distribute.)
3. the *PET authorities* share the key with which the Vote Codes are encrypted. They perform distributed PET tests on the bulletin board to register each vote.
4. the *decryption authorities* share the key for decrypting the candidate names in each vote.

To avoid cluttering the text we drop the indices that indicate the row, and corresponding code sheet, and just describe the set up w.r.t. a typical row. Let c_i be the i-th candidate, and VC_i the i-th vote code. The *ballot-construction authorities* begin by constructing, for each vote, and displaying on the BB:

1. A list $\mathcal{VC}$ of encrypted (c_i, VC_i) pairs in a canonical order,
2. A re-encrypted version of $\mathcal{VC}$ with each row shuffled by a secret random order ρ.

$$\begin{aligned}\rho(\mathcal{VC}) = &([c_{\rho^{-1}(1)}], [VC_{\rho^{-1}(1)}]),\\ &\ldots,\\ &([c_{\rho^{-1}(n)}], [VC_{\rho^{-1}(n)}])\end{aligned}$$

   Each row of this table has to be decrypted and the information printed on a code sheet. Note that the candidates will be printed in the in the order given, i.e. according to the ρ permutation encoded in this sequence.
3. A table $\mathcal{PC}$ of encrypted preference codes in order is also posted to the Bulletin Board. Each row will correspond to a code sheet and will have the form:

$$\mathcal{PC} = [PC_1], \ldots, [PC_n]$$

Code Sheet construction: The secret part We now add encryptions of $\{1, ..., n\}$ in order to each of the rows of the $\mathcal{VC}$ table. Thus each element of the table is now a triple and each row has the form:

$$\mathcal{VC}' = ([1], [c_{\rho^{-1}(1)}], [VC_{\rho^{-1}(1)}]), \ldots, ([n], [c_{\rho^{-1}(n)}], [VC_{\rho^{-1}(n)}])$$

Another set of authorities called the *Code Sheet authorities* then perform further shuffles within each row of the Vote Codes, by another secret, parallel, random permutation σ_i, where i indexes the row in question. The protocol of [Ram09] is used here to ensure that the triples are preserved in these shuffles. The output of this is posted to the Bulletin Board.

The result of this will be a new table $\mathcal{VC}^*$ in which each row has the form:

$$\mathcal{VC}^* = ([\sigma^{-1}(1)], [c_{\rho^{-1}\circ\sigma^{-1}(1)}], [VC_{\rho^{-1}\circ\sigma^{-1}(1)}]), \ldots, ([\sigma^{-1}(n)], [c_{\rho^{-1}\circ\sigma^{-1}(n)}], [VC_{\rho^{-1}\circ\sigma^{-1}(n)}])$$

This table will be posted to the Bulletin Board and used to register the votes. Notice that the order in which the candidates, and the vote codes, appear is different to that that appears on the code sheets, in fact differs by the secret σ permutation. This is crucial to ensure that the scheme is receipt-free.

The authorities are also required to show their workings on the Bulletin Board to allow for auditing.

Ack computation and return When a vote $\mathcal{V}$ arrives with the Trustees (from the VS) it's an encrypted list of vote codes in preference order:

$$\mathcal{V} = ([VC_{\pi^{-1}(1)}], \ldots, [VC_{\pi^{-1}(n)}])$$

. For convenience we will assume throughout this section that each vote is a *complete* list of preferences, *i.e.* a list of all the candidates. However, partial lists could easily be accommodated, though the tallying would reveal how many preferences had been expressed. This issue is discussed further in Section 5.

The authorities construct the *tallyable vote* T and the *acknowledgement list* A on the BB as follows: For $j = 1$ to n:

1. Do PET tests comparing the vote $\mathcal{V}$ with the list $\mathcal{VC}^*$ from the bulletin board.[4] When $[\mathcal{V}_j]$ matches $[\mathcal{VC}^*]_i$, this means that $\pi^{-1}(j) = \rho^{-1} \circ \sigma^{-1}(i)$, so candidate $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ gets preference j.
   (a) **Vote Updating:** Put $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ into the vote T at preference j. (For example, T could just be a list of candidate names in order, in which case all we do is add $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ into the list T in the j-th

[4] Note that $\sigma \circ \rho$ is secret, i.e. not the permutation that's printed on the code sheets, so this does not reveal anything about the vote. If a party knows $\sigma \circ \rho$, or knows σ and has the code sheet, they can learn the vote from this step, which is a good reason to have ρ and σ generated by a series of shufflers.

place.) Since $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ is still encrypted, nobody knows which candidate actually got preference j.

(b) **Ack code updating:** To construct the correct acknowledgement code, the Trustees extract $[\sigma^{-1}(i)]$ from $[\mathcal{VC}^*{}_i]$ and append to A the term

$$([\sigma^{-1}(i)], [PC_j])$$

2. Once all the terms in the row have been registered and ranked we have a sequence of pairs of the form:

$$([\sigma^{-1}(i)], [PC_j]), \text{ for } i = 1, \ldots, n$$

in which $\pi^{-1}(j) = \rho^{-1} \circ \sigma^{-1}(i)$. Now the preference codes must be arranged in the correct order, corresponding to the order shown on the code sheet. We want to do this in a way that does not result on the Trustees, or anyone, learning the σ shuffle. We can accomplish this as follows: the Trustees each perform a secret parallel shuffle on the sequence, i.e. preserving the pairings. Once this is done, a threshold set of the Trustees decrypt all the terms. The preference codes are now arranged into the order of the first terms.

To see that this arranges the preference codes in the correct order, let τ be the composition of the Trustees' random shuffles in the last step. Then the list before τ is applied is

$$([\sigma^{-1}(i)], [PC_{\pi\circ\rho^{-1}\circ\sigma^{-1}(i)}]), \text{ for } i = 1, \ldots, n$$

Applying τ give us

$$([\sigma^{-1} \circ \tau^{-1}(i)], [PC_{\pi\circ\rho^{-1}\circ\sigma^{-1}\circ\tau^{-1}(i)}]), \text{ for } i = 1, \ldots, n.$$

Rearranging by first element gives us

$$([i], [PC_{\pi\circ\rho^{-1}(i)}]), \text{ for } i = 1, \ldots, n$$

which is exactly the result of rearranging the preference codes according to inverse of the vote, π^{-1}, then rearranging them again according to ρ, the order they are printed on their code sheet.

Tallying Since the votes are simply lists of encrypted candidate names in preference order, there are many possible tallying options depending on the voting scheme and on the degree of privacy required. Any of the secure tallying protocols for STV/IRV or Borda mentioned in the introduction could be implemented here.

4.4 Proofs of correctness

Basic proof of correctness It should already be clear, but is important to state, that when everyone follows the protocol the votes are cast and counted as the voter intended.

Lemma 1. *When all authorities follow the protocol correctly, the vote registered is the same as the permutation applied by the voter to the Vote Codes, which is also the same as the vote implied by the acknowledging preference codes.*

Proof. By construction. □

Proof of security against a cheating client We claimed at the beginning of this section that a malicious client could not undetectably cast a modified vote, except given extra information. Here we restate the claim and sketch a proof.

Claim. A cheating client or VS (who doesn't know the codes) can swap two preferences undetectably only if it knows which two positions on the code sheet they correspond to.

Proof. Starting assumptions:

1. Each ballot ID gets only one registered vote and ack code list.
2. Each vote is a complete permutation.
3. The VS can derive no information from the Ack Codes.

In the worst case the client knows exactly what vote the voter wants to cast. We will assume this worst-case adversary and show that it can rearrange the preference acks correctly only if it knows the corresponding positions on the ballot.

Suppose the voter intends to cast vote V, a permutation of the candidate names. The cheating client swaps preferences i and j, which means swapping the i-th and j-th items in the list of Vote Codes (or candidate names), and submits the modified vote instead. It receives from the trustees a (cleartext) list of preference codes P arranged in the order the candidates appear on the code sheet. This list differs from what the voter is expecting only in that the codes for the i-th and j-th preferences must be swapped. Since the cheating VS knows which candidate names these correspond to, swapping them correctly implies knowing which (unordered) two locations on the code sheet they occupy. □

Proof of privacy We wish to show that the vote construction step on the bulletin board preserves vote privacy. Obviously only computational privacy is achieved, because both the vote codes and the ordered candidate names are shown, encrypted, on the bulletin board.

The weakest point for maintaining voter privacy is in the printing and distribution of the code sheets. If we assume that that phase doesn't leak information, the distributed ballot construction implies that ρ and σ remain secret if both:

1. At least one of the ballot construction authorities keeps their component permutation secret, and
2. Fewer than a threshold number of decryption trustees collude.

The following lemma shows that even an adversary who knows some information about this voter's preferences, and who can see the code sheet, learns nothing (more) from observing the bulletin board. The proof is in Appendix 6.1

Lemma 2. *The bulletin board proof can be simulated to produce a transcript computationally indistinguishable from the true one, even by an adversary who observes the code sheet.*

5 Protocol C: Two-dimensional tables

In this section each voter receives a two-dimensional table. Each row represents a candidate, each column a number. The numbers could be ranks for STV, Borda or IRV votes, as shown in Figure 5, or they could be scores for Range or Approval voting, as shown in Figure 6. Compared to Protocol B, this has more complicated vote casting but much simpler Ack checking.

Candidate	1st	2nd	3rd	4th	5th
Red	37	90	12	08	72
Green	14	46	88	49	09
Chequered	95	10	21	83	20
Fuzzy	33	99	21	73	87
Cross	39	25	31	11	92

Ack Code: 8243
Ballot ID: 3884092844

Fig. 5. Example of Candidate and Preference Codes

Candidate	Approve	Disapprove
Red	37	72
Green	49	09
Chequered	95	21
Fuzzy	73	87
Cross	25	31
Ack Code: 8243		
Ballot ID: 3884092844		

Fig. 6. Example of Candidate and Approval/Disapproval Codes

For each candidate, the voter selects the code in the appropriate column, which the client then sends to the vote server. As in PGD 1.0, each voter receives a single ack, and the security of the scheme is dependent upon the secrecy of the Vote Codes and Ack code.

5.1 Details of ballot construction, ack return and tallying

Ballot construction Ballot construction and ack return are much simpler than the corresponding construction in Protocol C. On the code sheets and on the Bulletin Board, the candidates can remain in canonical order throughout. For each ballot, for each candidate, the authorities post to the Bulletin Board

- an encrypted Ack Code, and
- for each canonically ordered candidate, a list of encrypted (Vote Code, number) pairs in a secret, random order.

There are two slightly different versions depending on the kind of voting.

- For Range or Approval Voting, each vote code list is shuffled independently. This makes it impossible to tell how many candidates received the same number.
- for STV, IRV, or Borda, the same shuffle is applied to the code list of every candidate on the same ballot. This makes it easy to check the validity of each vote: anything with at most one PET match in each column is valid, because it has no repeated preferences.[5]

In either case, the table should be printed on the code sheet in canonical order, while the order(s) on the bulletin board remain secret.

[5] We are assuming here that votes are valid if they skip preferences, but not if they repeat a preference. If another rule were applied then an appropriate validity checking step would have to be added later.

Tallying Again Plaintext Equivalence tests are used to match each Voter's encrypted Vote Codes with those on the Bulletin Board. When the submitted Vote Code matches $(VC_{ij}, number_j)$, this implies that candidate i (who is known from the canonical order) "gets" number $number_j$ (which is still encrypted). The correct interpretation of this depends on the voting scheme.

Approval or Range Voting, or Borda Count For voting schemes that simply accumulate a score for each candidate, the tallying is simple. Using an encryption scheme with homomorphic addition, $number_j$ can simply be added to candidate i's total without being decrypted. Of course the scores have to be set up correctly in advance, with, for example, 1 and 0 for approval and disapproval respectively in AV, and $n - j$ for the j-th preference in Borda. This is straightforward.

Lists of preferences: STV or IRV If the straightforward PET matching is done on the bulletin board, it reveals how many preferences each voter expressed. This protects against a cheating client or VS who submits only a subset of the complete preference list, but unfortunately it also violates each voter's privacy to some extent. In many instances, this would be a serious problem because it could allow a coercer to demand that a voter restrict the number of preferences they expressed. However, in the case where everyone must list the same number of preferences, all valid votes would be indistinguishable. This is fairly common in Australia, where often a permuation has to be complete to be valid, and it also occurs in the United States, where IRV with about 3 compulsory preferences is sometimes used.

Tallying for IRV or STV is complex. So far, for each vote, we have produced a list of candidate names (in canonical order) with their corresponding (encrypted) rank. There are (at least) two possible options:

- Shuffle all the votes in this form and then decrypt them at the end. This would give the correct answer but possibly expose the voters to pattern-matching attacks (a.k.a. "Italian" Attacks) as described by Heather [Hea07] (and others).
- Apply the Shuffle-Sum protocol [BMN+09], possibly with a preprocessing step to deal with votes that skip some preferences.

5.2 Proofs of correctness for Protocol C

This protocol is considerably simpler than Protocol B, which is reflected in the relative simplicity of the assumptions and proofs.

Basic proof of correctness Again, when everyone follows the protocol the votes are cast and counted as the voter intended.

Lemma 3. *When all authorities follow Protocol C correctly, the vote registered corresponds to the rows and columns chosen on the code sheet.*

Proof. By construction. □

Proof of security against a cheating client We would like to argue that a cheating client or VS cannot alter a vote undetectably, but it is important to clarify "undetectably." So far in this paper the voter has been able to detect vote manipulation by the absence of the expected ack code(s). The same will be true here, unless the cheating client or VS submits a subset of the $(VC_{ij}, number_j)$ pairs, which is detectable only if the voter checks the bulletin board (presumably via an independent device). As explained above, this is not a problem in schemes in which the number of pairs is specified, such as AV with compulsory explicit approval or disapproval of each candidate, or IRV with exactly three preferences.

Claim. A cheating client or VS (who doesn't know the codes) cannot add valid (candidate, number) pairs.

Proof. Achieving a successful PET test requires either knowledge of the relevant code or collusion of a threshold number of decryption authorities. □

Claim. A cheating client or VS (who doesn't know the codes) cannot remove (candidate, number) pairs without this being observable on the bulletin board.

Proof. The bulletin board reveals how many pairs were registered for each vote. □

Proof of privacy As in Section 4.4 we wish to show that the data on the bulletin board preserve (computational) vote privacy. Again we assume that that code sheet printing phase doesn't leak information, that at least one of the ballot construction authorities keeps their component permutations secret, and that ewer than a threshold number of decryption trustees collude.

Lemma 4. *The bulletin board proof can be simulated to produce a transcript computationally indistinguishable from the true one, even by an adversary who observes the code sheet.*

Proof. Omitted, but very similar to that of Lemma 2.

6 Discussion

6.1 Social engineering attacks on voters

These protocols are designed so that even a completely corrupted device is unable to alter a voter's choices undetectably, *assuming that the voter follows the protocol perfectly.* Since the voter probably votes infrequently, and trusts the computer for voting instructions, the assumption of perfect voter behaviour might be easy to undermine. For example, a virus that presented an appealing window with instructions like, "please enter the candidate names and vote codes in the order they appear on your code sheet," (for Protocol 3), or "please enter all the numbers in both tables," (for Protocol 2) would probably succeed with most voters. Given that information it would then be able to cast whatever vote it chose and manipulate the returning acknowledgement codes correctly to avoid detection. Although these kinds of attacks also work on other versions of code voting, our protocols are considerably more complicated and have more subtle privacy assumptions than the others, and hence are probably more vulnerable.

References

[Adi08] B. Adida. Helios: Web-based Open-Audit Voting, 2008.

[BMN+09] Josh Benaloh, T. Moran, L. Naish, K. Ramchen, and Vanessa Teague. Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting. *IEEE Transactions on Information Forensics and Security*, 2009.

[Cha01] D. Chaum. SureVote: Technical Overview. Proceedings of the Workshop on Trustworthy Elections (WOTE '01), 2001.

[CM05] Michael R. Clarkson and Andrew C. Myers. Coercion-Resistant Remote Voting using Decryption Mixes. In *Workshop on Frontiers in Electronic Elections (FEE 2005)*, 2005.

[Hea07] James A. Heather. Implementing STV Securely in Prêt à Voter. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 157–169, Venice, Italy, July 2007.

[JCJ05] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant Electronic Elections. Proceedings of the 2005 ACM workshop on Privacy in the electronic society, 11 2005.

[Ram09] Kim Ramchen. Parallel shuffling and its application to Prêt à Voter. Honours Thesis, 2009.

[RT09] Peter Y. A. Ryan and Vanessa Teague. Pretty Good Democracy. In *Proceedings of the 17th International Workshop on Security Protocols*, Lecture Notes in Computer Science, Cambridge, UK, April 2009. Springer-Verlag.

[TRN08] Vanessa Teague, Kim Ramchen, and Lee Naish. Coercion-Resistant Tallying for STV Voting. In *USENIX/ACCURATE Electronic Voting Technology Workshop (EVT'08)*, San Jose, CA, July 2008.

A Proof of Lemma 2

Lemma 2. *The bulletin board proof from Protocol B can be simulated to produce a transcript computationally indistinguishable from the true one, even by an adversary who observes the code sheet.*

Proof. The new information revealed on the bulletin board is the pattern of which elements of $(\sigma \circ \rho)(\mathcal{VC})$ match which elements of the vote $\mathcal{V}$. The whole transcript for one voter can be simulated, given the PET matching sequence, as follows:

1. Generate a preference order π. This can be done according to any pre-existing information about the distribution of this voter's preferences.
2. Re-encrypt $\mathcal{VC}$ (the canonically-ordered Vote Codes) and rearrange them according to π. Call this $\mathcal{V}$.
3. Re-encrypt $\mathcal{VC}$ again and arrange it according to the order, ρ, shown on the code sheet.[6] Call the result $\rho(\mathcal{VC})$.
4. Re-encrypt $\rho(\mathcal{VC})$ again and rearrange it according to the sequence of matching PETs. Call the result $\sigma \circ \rho(\mathcal{VC})$.
5. For each matching PET, simulate the transcript.
6. Simulate the shuffles that produce $\rho(\mathcal{VC})$ and $\sigma \circ \rho(\mathcal{VC})$.
7. Perform vote and ack construction exactly as in the real protocol.

The transcript produced here is computationally indistinguishable from the true transcript, even in the presence of the code sheet. □

[6] To show privacy against an adversary who doesn't have the code sheet, ρ could be generated uniformly here.