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 revisit *Pretty Good Democracy*, a scheme for verifiable Internet voting from untrusted client machines. The original scheme was designed for first-past-the-post elections. Here, we show how Pretty Good Democracy 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 cover 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, we have in general no grounds to make such assumptions. 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* (PGD) [RT09], which provides a proof of correct tallying and guarantees of ballot privacy that do not depend on any assumptions about the device used to cast the vote.

PGD is a form of *code voting*, which uses a separate channel (such as the postal service) to deliver a *code sheet* to each voter. The voter casts their vote by selecting the appropriate code(s) from their sheet. They then check that they receive an acknowledgement code that matches the one on their sheet. This provides mutual authentication between the voter and the authorities. In PGD [RT09], we added a proof of correct tallying on a web bulletin board. Although obviously not secure against a coercer who observes both the code sheet and the voter's communications, PGD is receipt-free as long as *either* the code sheet *or* the voter's messages to the voting authorities remain secret. As far as we know, it is the only scheme to provide both receipt-freeness and verifiability on an untrusted voting client. The main disadvantage of PGD relative to other schemes is that the election's integrity depends on correct behaviour of a threshold of

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 405–423, 2010.

trustees. This is not as onerous a requirement as it may first seem, however: even in national governmental elections, this can be solved by giving threshold shares to each political party, along with various neutral organisations such as the UN, so that one can have confidence that a threshold set will not rig the election because there is no result that all its members would be content with. Nevertheless, this arguably makes PGD inappropriate for elections of significant political importance, though there are many other hard-fought elections in which privacy and evidence of correctness are important, such as elections in trade unions and professional societies.

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), also known as the Alternative Vote (AV), which is the single-vacancy version of STV, is used in some local elections in the USA, the UK, Australia, and many other countries, and it is widely expected that there will soon be a referendum on using it for UK parliamentary elections. 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, WB09]. We present three different protocols, creatively named A, B and C. Protocol C also encompasses Approval Voting or Range Voting, in which the voter gives a score to all candidates.

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 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 more detail in Section 1.2.

In Section 1.1 we review PGD 1.0 and explain why the obvious extension to more complex voting schemes fails, then in 1.2 we give an overview and comparison of the extensions to more expressive voting schemes. The next three sections each contain a different extension, with a discussion of pros and cons. In Section 2, the simplest method (Protocol A) is described, which is secure but suffers from the disadvantage that each preference requires a separate interaction with the authorities. Protocol B, in Section 3, has the simplest voting experience, but somewhat complicated acknowledgement checking. Protocol C, in Section 4,

is an approach based on a two-dimensional table, which allows votes that are ordered lists or approval or range votes.

1.1 Review of PGD 1.0

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. We assume that the vote codes are kept secret and revealed only to the voter; mechanisms and procedures to support this assumption are discussed in detail in [RT09]. An example of a Code Sheet is given in Figure 1.

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

Fig. 1. Example Vote Codes

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 and registered 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 PGD scheme is that, in order to access the correct ack code, the voter server must invoke the co-operation of a threshold set of Trustees. The revealing of the correct ack code is thus a side-effect of the correct registration on the bulletin board of a valid code.

Cryptographic tools. This protocol relies on two main cryptographic tools:

Verifiable re-encryption shuffles (for example, [FS01, Gro03, Nef01, JJR02]) allow an authority to permute encrypted values (votes) by a secret permutation, while providing a publicly verifiable proof on the bulletin board that some shuffle has been correctly performed.

Plaintext Equivalence Tests (PETs) ([JJ00, TH08]), allow a threshold set of authorities who share a public key to compare two values encrypted with that key, and determine whether the two values are equal or not, without learning any other information.

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 (BB) and reveal the ack codes. They have shares of the secret key corresponding to the threshold public key, PK_T. "Registration" means recording the vote on the BB. (In the extension protocols in Sections 2, 3 and 4, the Trustees will be split into several authorities with slightly different roles.)
- 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 shuffles and final decryption steps. The auditors are not privileged or trusted, and do not receive secret information—any member of the public can be an auditor.

Full details of PGD 1.0 can be found in [RT09]. Here, for completeness, we give a brief outline. First, a sufficiently large set of voting and ack codes of the appropriate form are generated. These are encrypted under the Teller's threshold public key and put through a sequence of re-encryption shuffles. These codes are then assembled into a table of the appropriate dimensions: each row will have n encrypted vote codes and an encrypted ack code and will correspond to a code sheet. This table is referred to as the P-table. This construction ensures that only certain sets of entities acting in collusion could compromise the secrecy of the codes. The Registrar decrypts the information of the P-table and prints the vote codes to the code sheets.

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.

In order to ensure receipt-freeness, the P-table must be transformed to conceal the correspondence between the position in the row of an (encrypted) code and the candidate it represents. Each row of the P-table is therefore subjected to a further sequence of re-encryption permutations to create the Q-table. The Q-table is used to register the votes.

When votes are cast, the encrypted Vote Code supplied by the Vote Server is matched by a threshold set of Trustees via plaintext equivalence tests against the encrypted codes in the appropriate row of the Q-table. If a match is found, the matching term is flagged and the threshold set of trustees decrypt the ack code. The fact that the terms of the Q-table are permuted ensures that flagging one does not reveal what candidate this represents. The information defining the permutation is stored in an encrypted *onion* in the usual Prêt à Voter fashion and this is later used during tabulation to interpret the flagged term to identify the chosen candidate.

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.

An obvious extension to preference voting that fails. 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.

1.2 New Protocol Comparison

Figure 2 contains a functional comparison of the three new protocols presented in this paper. "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 cannot 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.

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

Fig. 2. Comparison of protocol features

Security properties are compared in Figure 3, which also includes PGD 1.0. It shows under what circumstances a corrupt device or Voting Server can manipulate the vote. "No" is good and "yes" is bad. "Knowing" a code means knowing its meaning, *i.e.* its candidate or preference.

Cheating client or VS can undetectably:	PGD 1.0	A	B	C
Truncate the vote knowing only the Vote Codes	n/a	Yes	No	Yes
Truncate the vote knowing only the Ack Codes	n/a	Yes	Yes	Yes
Otherwise manipulate the vote knowing only the Vote Codes	Yes	Yes	No	Yes
Otherwise manipulate the vote knowing only the Ack Codes	No	Yes	Yes	No
Otherwise manipulate the vote knowing only the order of candidates on the code sheet	n/a	n/a	Y	n/a
Manipulate the vote knowing neither Vote nor Ack Codes	No	No	No	No

Fig. 3. Comparison of protocol security properties

2 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.

2.1 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.

3 Protocol B: Returning the Acknowledgement Codes in Ballot Order

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.

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, which is computed by the authorities without requiring any voter interaction. The voting protocol is thus a simple two-pass process: the voter sends her list of vote codes and then checks the sequence of preference codes. The main drawbacks with this are that it may be difficult for voters to understand how to check their preference codes, and that the integrity guarantee is not as strong as that offered by Protocol A.

3.1 Security Properties

The integrity guarantee for this protocol depends on the candidates being listed in a secret, random order. Our main security claim is:

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

Proof. See Section 3.4.

Furthermore, the protocol is receipt-free if the voter keeps secret either their code sheet or their communications with the Vote Server. This is stated more precisely in Section 3.4 and proved in Appendix A.

3.2 Voter Interface Details

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 4(a), 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 (though obviously a cheating client or VS must not know in advance that this voter will not check their returned preference codes, or they could rearrange the vote).

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

(a) Vote Codes

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

(b) Preference Codes

Fig. 4. Example Code sheets

If the voter does take the trouble to verify the registration of her vote, she should 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(b), the first would be code W, (because Red is the first candidate in the order printed on the ballot paper, and the 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, as shown in Figure 5. Then the acknowledgement code will be the sequence of letters read down the column, thus:

Candidate	Vote Code	Pref code
Red	3772	W
Green	4909	C
Chequered	9521	K
Fuzzy	7387	T
Cross	2531	M
Ack Code: 8243		
Ballot ID: 3884092844		

Fig. 5. Example code sheet with preference codes filled in

Note that it would be possible to provide the voter with three sheets:

- A conventional code sheet showing the voting codes against the candidates with the candidates in canonical (*e.g.* lexical) order.
- The sheet showing the preference codes, as before.
- A sheet showing the candidates in the permuted order with the blank column alongside for the voter to fill in the preference codes.

Indeed, all three sheets could be generated and distributed via different processes and channels for added security. Whether this would help the voter and add sufficiently to the security to justify the additional costs would depend on the individual case.

3.3 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 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 $(\sigma \circ \pi)(L) = L_{\pi^{-1}(\sigma^{-1}(1))}, \ldots, L_{\pi^{-1}(\sigma^{-1}(n))}$.

In what follows, we will use '$[x]$' to denote the encryption of x. (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 [RT10], 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 five 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 *Output shuffle authorities* transform the votes into the correct order for tallying and the preference codes into the correct order for returning to voters.
5. the *decryption authorities* share the key for decrypting the candidate names in each vote.

Each row corresponds to one code sheet, *i.e.* one vote. To avoid cluttering the text we drop the indices that indicate the row, 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, a table similar to the P-table described above and displaying on the BB. Alongside each encrypted vote code we add the encryption of the corresponding candidate. We thus have a table in which each row comprises:

1. A list $\mathcal{VC}$ of encrypted (c_i, VC_i) pairs in a canonical order.

   Now, for each row, the pairs are subjected to a sequence of re-encryption shuffles to yield:
2. A re-encrypted version of $\mathcal{VC}$ with each row shuffled by a secret random order ρ.

$$\rho(\mathcal{VC}) = ([c_{\rho^{-1}(1)}], [VC_{\rho^{-1}(1)}]), ([c_{\rho^{-1}(2)}], [VC_{\rho^{-1}(2)}]), \ldots, ([c_{\rho^{-1}(n)}], [VC_{\rho^{-1}(n)}])$$

   This table we will again refer to as the Q-table.

   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:

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

The $\mathcal{VC}^*$-Table. Now we generate the S-table that will serve to register the votes. We need to introduce further permutations to the rows in order to ensure that the scheme is receipt free. First, in order to keep track of this permutation, we add to the ith pair an encryption of i. This each row is a list of triples of 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 [RT10] 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 (that is, it includes every candidate). 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 4.

The authorities construct the *tallyable vote* T and the *acknowledgement list* A on the BB as follows:

1. The PET authorities perform PET tests comparing the terms of the vote $\mathcal{V}$ with the list $\mathcal{VC}^*$ from the bulletin board, until they have found all possible matches.[1][2] 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. The following transformations can be perfomed (and verified) by anyone:

[1] 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.

[2] This could require n^2 PETs.

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

(b) **Ack code updating:** To construct the correct acknowledgement code, 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 in the authorities, or anyone, learning the σ shuffle. We can accomplish this as follows: the output shuffle authorities each perform a parallel shuffle on the sequence, preserving the pairings. Once this is done, a threshold set of the decryption authorities decrypt all the terms. The preference codes are now arranged into the order of the first terms. It is clear by construction that this puts the preference codes into the candidate order of the 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.

3.4 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. A malicious client can not undetectably cast a modified vote. Specifically, it cannot swap two candidates or preferences unless it knows the position of the corresponding candidates on

the code sheet, or the meanings of the relevant preference codes. Here we restate the claim and sketch a proof. Of course, if there are only two candidates then a swap can indeed be performed undetectably. The probability of successfully guessing the candidates positions is 1 in $\binom{n}{2}$.

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. 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 protocol is receipt-free. Obviously only computational privacy is achieved, because both the vote codes and the ordered candidate names are shown, encrypted, on the bulletin board (that, is it does not achieve "everlasting privacy"). Equally obviously, our protocol is not receipt-free against a coercer who can observe both the (properly authenticated) code sheet and also the voter's communications with the Vote Server. We show that the protocol is receipt free if either the code sheet or the voter's communications with the VS remain secret.

We prove receipt freeness according to the definition of Moran and Naor [MN06]. The basic idea is that the voter, when requested to vote in a certain way by the coercer, should have a "coercion resistance" strategy which allows them to vote in the way they wished while providing a view to the coercer that is indistinguishable from obedience.

The protocol presented here goes just up to the point of having a list of encrypted votes on the bulletin board, and hence is not supposed to reveal any information, so we can prove directly that the coercion resistance strategy produces a coercer view indistinguishable from obedience. (Moran and Naor's definition must be more complex to deal with the case that the coercer learns useful information from the public tally itself, which isn't relevant here.)

Whether the subsequent tallying step preserves receipt freeness is a separate question, which is outside the scope of this paper. As described in the Introduction, expressive voting schemes can be vulnerable to pattern-matching ("Italian") attacks, and the choice of tallying protocol should protect against this.

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:

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

We will refer to this list as the *Authority secrecy assumptions.*

The following theorem shows that an adversary who is ignorant of either the code sheet or the voter's communications with the VS learns nothing (more) about the vote from observing the bulletin board.

Theorem 1. *Given the authority secrecy assumptions above, Protocol B is* receipt-free *[MN06] against a coercer who either*

1. *does not observe the code sheet, or*
2. *does not observe the voter's communications with the Vote Server.*

Proof. The proof is in Appendix A. The main idea is that the voter can lie freely to the coercer about either ρ (if the coercer does not see the code sheet) or $\rho \circ \sigma$ (if the coercer does not observe communications with the VS) and hence produce a plausible claim to have cast any vote with the same number of candidates as the one they truly cast.

4 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 6, or they could be scores for Range or Approval voting, as shown in Figure 7. Compared with Protocol B, this has more complicated vote casting but much simpler Ack checking.

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.

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

4.1 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.[3]

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.

[3] 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 (typically) three 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 a privacy-preserving STV or IRV tallying protocol [BMN+09], [Hea07], [WB09], possibly with a preprocessing step to deal with votes that skip some preferences.

4.2 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 2. *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 3.4 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 fewer than a threshold number of decryption trustees collude.

Theorem 2. *Protocol C is* receipt-free *[MN06] against a coercer who either*

1. *does not observe the code sheet, or*
2. *does not observe the voter's communications with the Vote Server.*

Proof. Omitted, but very similar to that of Theorem 1.

5 Discussion

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 C), or "please enter all the numbers in both tables", (for Protocol B) would probably succeed with many 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 attacks of this kind 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] Adida, B.: Helios: Web-based Open-Audit Voting (2008)

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

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

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

[FS01] Furukawa, J., Sako, K.: An Efficient Scheme for Proving a Shuffle. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 368–387. Springer, Heidelberg (2001)

[Gro03] Groth, J.: A Verifiable Secret Shuffle of Homomorphic Encryptions. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 145–160. Springer, Heidelberg (2002), Later Version at `www.brics.dk/~jg/JournalShuffle2.ps`

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

[JCJ05] Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant Electronic Elections. In: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, vol. 11 (2005)

[JJ00] Jakobsson, M., Juels, A.: Mix and Match: Secure Function Evaluation via Ciphertexts. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, p. 162. Springer, Heidelberg (2000)

[JJR02] Jakobsson, M., Juels, A., Rivest, R.: Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In: USENIX Security Symposium, pp. 339–353 (2002)

[MN06] Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 373–392. Springer, Heidelberg (2006)

[Nef01] Andrew Neff, C.: A Verifiable Secret Shuffle and its Application to E-Voting. In: Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001), pp. 116–125. ACM Press, New York (2001)

[RT09] Ryan, P.Y.A., Teague, V.: Pretty Good Democracy. In: Proceedings of the 17th International Workshop on Security Protocols, Cambridge, UK, April 2009. LNCS. Springer, Heidelberg (2009)

[RT10] Ramchen, K., Teague, V.: Parallel shuffling and its application to Prêt à Voter. In: USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 2010), Washington, DC (August 2010)

[TH08] Ting, P.-Y., Huang, X.-W.: Distributed Paillier plaintext equivalence test. International Journal of Network Security 6(3), 258–264 (2008)

[WB09] Wen, R., Buckland, R.: Minimum disclosure counting for the alternative vote. In: Ryan, P.Y.A., Schoenmakers, B. (eds.) VOTE-ID 2009. LNCS, vol. 5767, pp. 122–140. Springer, Heidelberg (2009)

A Proof of Theorem 1

Theorem 1. *Given the authority secrecy assumptions defined in Section 3.4, the protocol is* receipt-free *[MN06] against a coercer who either*

1. *does not observe the order of candidates on the code sheet, or*
2. *does not observe the voter's communications with the Vote Server.*

Proof. This is really two separate results, one for each of conditions 1 and 2. The proofs are in the following two lemmas.

Lemma 3. *Suppose a voter wishes to cast vote v (a full permutation of the candidates) but the coercer instead demands π (also a full permutation). Suppose also that the coercer observes the voter's communication with the Vote Server, but no information about the order of candidates on the code sheet. Then there exists a coercion-resistance strategy CR for the voter such that the coercer's view when the voter obeys (i.e. votes π) is indistinguishable from the coercer's view when the voter disobeys (i.e. votes v) and runs CR.*

Proof. The coercer's view consists of any communication between herself and the voter before the vote, the complete list of messages between the voter and the VS, and the Bulletin Board transcript.

The voter's coercion resistance strategy is to vote v and tell the coercer that he voted π. Then the order of the preference acks, assuming vote π, is consistent with exactly one code sheet permutation ρ', which the voter can easily compute and claim to the coercer. Also the pattern of PET matches in the bulletin-board transcript is consistent with exactly one registration permutation σ'.

Distinguishing the claimed permutations ρ' and σ' from the true ρ and σ reduces to gaining information from the shuffles, which we are assuming to be impossible without sufficient authorities misbehaving. □

Lemma 4. *Suppose a voter wishes to cast vote v (a full permutation of the candidates) but the coercer instead demands π (also a full permutation). Suppose also that the coercer observes the voter's code sheet, but that the voter has an untappable channel to the VS. Then there exists a coercion-resistance strategy CR for the voter such that the coercer's view when the voter obeys (i.e. votes π) is indistinguishable from the coercer's view when the voter disobeys (i.e. votes v) and runs CR.*

Proof. The coercer's view consists of any communication between herself and the voter before the vote, all the information on the code sheet, and the Bulletin Board transcript.

The voter's coercion resistance strategy CR is to vote v and then tell the coercer that he voted π and received acks consistent with that.

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 coercer, having seen the code sheet, knows ρ. Given this information, the voter's claim to have voted π is consistent with exactly one registration permutation. (Specifically

$\sigma' = \sigma \circ \rho \circ \upsilon^{-1} \circ \pi \circ \rho^{-1}$, which the voter does not have to compute). Distinguishing voter obedience from the CR strategy hence reduces to distinguishing σ from σ' based on the shuffles in the construction phase, or distinguishing the true vote or preference return order from the suffled ones in the tally phase, which is infeasible without sufficient authorities misbehaving. □