# Pretty Good Democracy

Peter Y A Ryan
Dept. Computer Science and Communications
University of Luxembourg
peter.ryan@uni.lu

Vanessa Teague
Dept. Computer Science and Software Engineering
University of Melbourne
vteague@csse.unimelb.edu.au

September 10, 2009

**Abstract**

*Code voting* seeks to address the issues of privacy and integrity for Remote Internet Voting. It sidesteps many of the inherent vulnerabilities of the Internet and client platforms but it does not provide *end-to-end* verification that votes are counted as cast. In this paper, we propose a simple technique to enhance the verifiability of code voting by ensuring that the Vote Server can only access the acknowledgement codes if the vote code is correctly registered by a threshold set of Trustees. The mechanism proposed here therefore adds an extra level of verifiability in registering and counting the vote. Voter-verification is simple and direct: the voters need only check that the acknowledgement code returned to them agrees with the value on their code sheet. To ensure receipt-freeness we propose the use of a single acknowledgement code per code sheet, rather than individual acknowledgement codes for each candidate with usual code voting.

## 1 Introduction

Internet Voting is highly controversial. The inherent insecurity and unreliability of the Internet as infrastructure, and of home or office PCs as voting platforms, present severe obstacles to its usage. The dangers of vote

buying and coercion are especially problematic for remote Internet voting. Nonetheless, there appear to be contexts in which such threats are mild enough to consider Internet voting as viable, such as elections for student bodies, professional societies, university officeholders [AdMPQ09] *etc.* This paper advances techniques for side-stepping many of the vulnerabilities of the Internet, though we would not claim our scheme is secure enough for use in national or regional elections.

Our scheme is based on *Code voting*, which aims to avoid requiring the voter to trust any computational device or digital signature in order to vote. Instead, code sheets are sent out to the voters via ordinary mail or other supposedly secure channel. These code sheets have random vote codes against each candidate. In effect, they serve as private code books to enable the voters to communicate securely with the voter server. Chaum's SureVote, [Cha01], appears as the first place that such an approach is presented. Typically these code sheets include acknowledgment codes (ack codes for short) that the server returns to the voter after receiving valid codes from the voter. These ack codes serve a dual role: to assure voters that their code was correctly received and to authenticate the voting system to the voter. However, receipt of the correct ack code does not give any assurance that the vote will subsequently be correctly included in the overall count. Such schemes are therefore not *end-to-end*, that is, there is no guarantee that votes make it all the way from being cast to being counted.

Our objective is to provide a mechanism to provide assurance that, if the correct ack code is returned to the voter during the voting session, then the voter can be confident that her vote will be accurately included in the election outcome. Like other *end-to-end* verifiable election schemes, we make use of a public *Bulletin Board*, which is an authenticated broadcast channel with memory. (This could be implemented as a website which provides some added assurance that everyone is served the same data.) We describe a mechanism whereby the voting Server can only recover the correct Acknowledgement Code with the cooperation of a threshold set of Trustees. In the absence of a large-scale collusion between the Trustees or the leaking of information about their codes, the fact that a voter gets the correct ack code back from the vote server ensures that their vote code is accurately registered on the Bulletin Board. Standard techniques such as the robust anonymising mixes described in [Rya08, Adi06] can then be used to ensure the codes recorded on the Bulletin Board are correctly translated into votes in the final tally.

In contrast to most verifiable schemes, that require the voter to subse-

quently visit the Bulletin Board and confirm their receipt is correctly posted, our scheme simplifies the procedure for the voter. All the voter needs to do is to verify the received ack code during the voting session. It is hoped that this simpler, more immediate check will encourage a higher proportion of voters to participate in the verification process. Optionally, voters, or proxies acting on their behalf, can also subsequently visit the Bulletin Board and confirm the registration of their vote.

Another problem of remote voting in general and *Code Voting* in particular is its vulnerability to vote selling, that is, a voter simply sells her code sheet to a coercer or vote buyer. The present scheme does not as it stands address this problem but it could be enhanced by adding, for example, multiple casting [VG06] or *tokens* [JCJ05]. The idea is allow the voter to appear to sell their code sheet to the coercer, while actually retaining the opportunity to vote.

Our scheme is however receipt-free due to the use of a single acknowledgement code for each ballot rather than a distinct ack code for each candidate as usual for code voting. Thus, the fact that the ack codes are publicly available does not allow an adversary to identify the vote cast. It does of course demonstrate that a vote was cast with that code sheet, so opening up the possibility of forced abstention attacks.

Although at first glance the single ack seems to reduce the voter's opportunity to verify that the *correct* code was received, we do not believe that this is true. With either single or multiple ack codes, the voter's guarantee of integrity depends upon the secrecy of their (vote and ack) codes. An adversary who could learn a voter's codes and intercept their communications could substitute the individual ack that the voter expected just as easily as it could send a substituted vote code to the server. One the other hand, it may be that receiving a distinct ack code corresponding to their choice of candidate may give the voter an additional sense of confidence. It might make sense to return distinct ack codes during the voting session but not post them on the bulletin board. We will return to this point in Section 9.3.

Related Work is described in following section, then in Section 3 we explain the security properties of our contribution. Section 4 describes Code Voting. Our enhanced code voting scheme, PGD, is presented in Sections 5 to 8, with some further enhancements in Section 9.

## 2 Related Work

Similarly in its goal, Peters in [Pet05] describes a scheme for a secure Web Bulletin Board that prevents the corruption of a few parties from endangering the security goals of the entire election. To do so he uses threshold signatures. If the user receives such a signature and is able to verify it, they get the assurance that their vote is included on the board. The drawback of this approach is that the voter needs to rely on some computational capabilities on their computer.

Oppliger *et al.* evaluate the pros and cons of using multiple casts, multiple code sheets, and voting credentials [OSH08]. As an alternative Helbach *et al.* mentions linkable group signatures in combination with multiple casts [HSS08]. While preventing vote selling attacks these mechanisms cannot prevent coercion except for allowing a voter to update an influenced vote.

The scheme presented here has some points of contact with Chaum *et al*'s Scantegrity II, [CCC+08]. While PGD and Scantegrity II are very different, in particular Scantegrity II is a supervised scheme while PGD is a remote scheme, there is some commonality in the use of random codes. Both schemes use such codes and in both it is important that certain players learn only one of the full set of codes per ballot, but for quite different reasons. In Scantegrity II, it is the voter who should only learn the selected code. This is done using invisible ink on optical scan type ballots to reveal only the code against the chosen candidate. Revealing more than one code per ballot invalidates the ballot. This is because the voter does not get a receipt but just notes the code revealed when making her selection. The fact that she should learn only the chosen code protects the voter against being accused of presenting fake codes. In PGD, it is the Vote Server that should only learn the selected code, to prevent it registering alternate codes. For both schemes of course it is important that codes are sufficiently hard to guess.

Civitas [CCM07], an implementation with extensions of [JCJ05], provides strong guarantees of integrity and coercion resistance, including resistance against a coercer who is physically present during voting, as long as that coercer leaves the voter unsupervised at least once. This is achieved by giving each voter one true voting credential, plus an algorithm for generating false credentials with which to fool the coercer. The security guarantees depend on trusting the voter's computer, which has to do cryptographic computations. Helios [Adi08] also provides very strong integrity guarantees, and uses a series of voter challenges [Ben06] to avoid having to trust the

computer. However, this scheme is not receipt free. Our work achieves (almost) the best of both worlds, offering receipt freeness without having to trust the computers used for voting. The main shortcoming compared to Civitas is that we do not defend against against a coercer who intercepts the voter before or during voting, though token style mechanisms could possibly be incorporated to counter such threats. Our integrity guarantee is slightly weaker than that of Helios, being based on trusting a threshold set of election trustees. Furthermore, the Vote Codes must be kept secret to preserve both privacy and integrity. This is described in detail below.

### 2.1 Registration

Secure voter registration and coercion-resistant distribution of voting credentials are probably the most difficult open problems in Internet voting. Krivoruchko [Kri07] devised a distributed registration protocol, in which several registration servers each send a share of a credential to a voter's computer. Civitas uses a similar mechanism. However, both these schemes require significant computation at the voter's end, and we do not know how to extend their ideas to a scheme that avoids trusting the voter's computer. This leaves us with the traditional "secure" channel of the postal system, in which case the best we can say is that vote stealing is no easier than it is for paper-based postal votes (not a strong claim). Since our scheme is not coercion-resistant anyway, this level of security may be sufficient in some cases. An alternative is to insist that voters attend a registration centre in person to receive the code sheet. Registration centres could be opened weeks before the election, so this could still be much more convenient than requiring people to actually vote in person.

## 3 Security Properties of PGD

The enhancements to conventional code voting proposed here improve the security properties by providing an end-to-end auditable voting system. We should note that the degree of verifiability provided here is not as strong as that provided by some other voter-verifiable schemes, e.g. Prêt à Voter, [Rya07]. For Prêt à Voter, the guarantees of integrity are not conditional on the absence of collusion: any attempts to corrupt votes can be detected. For the present scheme, a sufficient collusion of trustees or the leakage of codes could lead to corruption of votes in a way that would not be detected.

Our scheme does provide a far higher degree of verifiability than conventional code voting. We summarise the key properties as follows:

**Integrity**

**Cast as Intended** The received ack code allows the voter to verify that *some* ballot is included on her behalf on the Bulletin Board representing the batch of recorded votes. Assuming that no sufficiently large collusion of Trustees occurs and that code information is not leaked, the voter can be confident this encoded the candidate she intended. (She must also assume that the code sheets were properly audited, see Section 6.)

**Counted as Cast and Tallied Correctly** These can be verified from the Bulletin Board using standard techniques, which do not require trusting any of the authorities.

**Privacy**

**Receipt-freeness** Receipt-freeness is ensured by the use of a single ack code for each code sheet. This means that voters cannot afterwards prove how they voted, even if the coercer has access to the Bulletin Board and the voter's private information and code sheet.

**Coercion-resistance** Our scheme does *not* as it stands provide protection against coercion initiated before voting, such as code sheet buying or direct observation of the voting process. For this purpose we would need extra mechanisms like stronger voter authentication or tokens in the style of Juels *et al* [JCJ05].

# 4 Code Voting

*Code Voting* provides a way to sidestep much of the inherent insecurity of the Internet: a code sheet is sent via a supposedly secure channel such as conventional mail, to each eligible voter. Each code sheet carries a unique ballot ID number and the list of candidates. For a typical implementation there is a random voting code and an acknowledgement code against each candidate. For each code sheet, the codes will be pairwise distinct and the codes vary (pseudo-)randomly from sheet to sheet. In effect, each voter is given their own personal code book to communicate with the voting system.

In order to place a vote, the voter logs onto the election's Vote Server and provides the ballot ID along with the code corresponding to their choice of candidate. The server verifies the validity of the ballot ID and vote code and responds with the appropriate ack code for the code sheet. The voter should check the server's response against the value shown on her code sheet. This will serve a dual purpose of confirming receipt of the vote code and providing a degree of authentication of the server to the voter.

A code sheet that includes vote and acknowledgement codes is shown in Figure 1.

| Candidate | Vote Code | Ack code |
|---|---|---|
| Asterix | 3772 | 8872 |
| Idefix | 4909 | 4044 |
| Obelix | 9521 | 1098 |
| Panoramix | 7387 | 4309 |
| Ballot ID: 3884092844 | | |

Figure 1: Typical code sheet

After the election has closed, the submitted ID/codes will be translated back into the candidate choices. In a conventional code voting scheme, this latter step is typically not verifiable and requires significant trust to be vested in the tabulating authority.

# 5 Pretty Good Democracy

We now describe a simple enhancement to the basic code voting scheme described above that is designed to add an extra degree of assurance that votes are correctly tabulated. The key idea is to ensure that the Voting Server can only return the correct ack code to the voter with the cooperation of a threshold set of trustees who all participate in registering the vote on the Bulletin Board. Furthermore, rather than using a distinct ack code for each candidate, we use a single ack code per code sheet. This allows the posting of the ack codes on the Bulletin Board, allowing an extra level of verifiablity, while preserving receipt-freeness. Thus, if she wishes, a voter can additionally visit the Bulletin Board and confirm that her vote has indeed been registered. She cannot directly determine how her vote has been recorded. A typical code sheet for our scheme is shown in Figure 2.

| Candidate | Vote Code |
|---|---|
| Asterix | 3772 |
| Idefix | 4909 |
| Obelix | 9521 |
| Panoramix | 7387 |
| Ack Code: 8243 | |
| Ballot Id: 3884092844 | |

Figure 2: Typical PGD code sheet for the enhanced scheme

### 5.1 Notation

Throughout this paper $\{x\}_k$ denotes the randomised encryption of $x$ under the public key $k$. Public key is often abbreviated to PK.

Each code sheet will carry a unique serial number $i$. These serial numbers will be rather large, say twelve digits, so that, like credit card numbers, it will be hard to guess valid IDs. Let $VC_{i,j}$ denote the vote code for the $i$-th code sheet and a candidate indexed by $j$, with $j \in \{1, \ldots, c\}$. Similarly $Ack_i$ denotes the acknowledgement code on the ballot with ID $i$. The vote and ack codes might typically be 4 digits.

### 5.2 The Election Roles

Here we outline the key roles of the scheme.

- A Voting Authority VA who generates the requisite number of vote codes and ack codes encrypted under the Trustees' PK.
- A set of Clerks, involved in the setup phase.
- A Registrar who decrypts the table provided by the Clerks and prints the code sheets.
- A Returning Officer who distributes the code sheets to the voters.
- 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 Voting Server, who receives the votes, i.e. a serial number $i$ and vote code $VC_{i,j}$, from each voter, then encrypts $VC_{i,j}$, and posts $(i, \{VC_{i,j}\})$ on the Bulletin Board.
- 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 *etc.*, and verifying the anonymising mixes.

### 5.3 Generating the Code Sheets

Firstly, the Trustees collaborate to generate a joint, threshold key $PK_T$ with a distributed key generation protocol as originally described by Canetti *et al.* in [CGJ+99].

Suppose that we have $v$ voters and $c$ candidates. We need to generate sufficient code sheets, allowing for some random auditing, in such a way as to ensure that no single entity knows the codes associated with any ballot. The following construction achieves this in a simple and secure fashion. The Voting Authority generates $\lambda v(c+1)$ distinct codes of the appropriate form, say four to six digit codes[1], and then encrypts these under the Trustees' public key $PK_T$. This can be done in a fully open, auditable fashion on the Bulletin Board. The encryptions can be verified by simply revealing the randomisation factors (these will be re-randomised later). The initial encryptions could be performed with randomisations set equal to 1. The factor $\lambda > 1$ is just to allow for a sufficient proportion of the code sheets to be randomly audited and then discarded.

The Clerks are now responsible for putting the resulting batch of encrypted terms through a suitable number of re-encryption mixes. After this, the shuffled, re-encrypted terms are assembled into a table with $\lambda \cdot v$ rows and $c + 1$ columns. Each row will now constitute the codes and ack for a code sheet, the last column being the encryption of the Ack Code. We will refer to this as the *P table*, in analogy with the similar construct in Scantegrity II.

Note that all of these steps can be performed in the open on the Bulletin Board. The multiple secret shuffles ensure that nobody knows how the codes

[1] **V:** Obviously the codes must be long enough to allow $\lambda v(c+1)$ distinct values. The code length represents a tradeoff between usability and security, because short codes are easier for voters to type but also easier for attackers to guess. See Section 8 for a discussion of security threats related to guessing codes.

are grouped into code sheets. Note also that we need to ensure that each code sheet has distinct vote codes. This can be achieved by ensuring that the initial batch of codes are pairwise distinct. This may mean that the length of the codes needs to be increased, or if longer codes are regarded as undesirable, we could use several batches.

Notice that this construction ensures that all the terms have plaintext of the required form while at the same time ensuring that no single entity knows which codes are associated with which candidates or which codes appear on the same code sheet. This style of construction appears to be quite generic where it is required to generate a large number of encrypted terms all having plaintext of some constrained form. It will not be suitable for all contexts of course, for example, where the set of plaintexts generated at the outset from a strict subset all the possible valid plaintexts then the construction could reveal critical information. In the present case, this does not seem to be an issue.

Thus, each row of the table will correspond to a code sheet and has the from:

$$i, \{VC_{i,1}\}_{PK_T}, \{VC_{i,2}\}_{PK_T}, \ldots, \{VC_{i,c+1}\}_{PK_T}$$

Where $VC_{i,c+1}$ will serve as the Ack Code for the $i$th code sheet.

The entries in this table need to be decrypted and printed to the code sheets and distributed to the voters prior the start of the election. The decryption is performed by the Registrar by invoking the assistance of a threshold set of Trustees. The set of Trustees used here can be varied throughout the process, in particular, the Trustee who performs the final decryption step should be varied throughout in order to ensure that no single Trustee knows all the codes. Better yet, if we are using El Gamal, then each of a threshold number of Trustees can do a part of the decryption using their share, and the pieces can be assembled by the Registrar [CGS97]. That way, only the Registrar learns the decrypted values.

After decryption and (successful) auditing, the code sheets are distributed to the voters by the Returning Officer via the most secure feasible channel. (See the discussion in Section 2.1). It is assumed that exactly one code sheet is delivered to each eligible voter. We assume that the Returning Officer keeps no record of which sheets are distributed to which voters. We could for example require that the Registrar prints the code sheets in some secure fashion in privacy protected envelopes (using pressure printing?) or

scratch strips. Another possibility is to distribute the job among several printers using the techniques of [ECHA09], which uses invisible ink to allow the printing of a secret without any single printer learning what has been printed. The batch would then be (physically) shuffled before being passed to the Returning Officer. (The ideal would be to use the printing process itself to combine the Trustees' shares so that nobody, including the Registrar, learnt any Vote Codes, but it is not clear how to do this.)

### 5.4 Setting up the Bulletin Board

We now need to transform the P table into a form that can used in the tallying process. For this we need to permute the encrypted vote codes terms within each row and store the information defining the permutation in encrypted form (in a so-called *onion*) added to each row. The ack codes are left untouched. Furthermore, we want to do this in a distributed fashion such that no single entity knows the permutations. We adapt some techniques from the Prêt à Voter design suite and have a number of options that we outline below.

One option is to use a decryption mix based approach along the lines presented in Prêt à Voter 2005, [RS06b]. This has the advantage that we can straightforwardly deal with full permutations of the candidates but is less flexible and robust than the re-encryption based techniques outlined later. For decryption mixes, we define a *seed* space $\Psi$, say 64 bit strings, and an unbiased function $\Gamma$ from $\Psi$ to the set of permutations on the candidate set. Now, for the $i$th row of the $P$ table, the first clerk chooses a seed $\rho_{i,1} \in \Psi$ at random and computes $\pi_{i,1} := \Gamma(\rho_i)$. The sequence of encrypted vote codes are re-encrypted and permuted according to $\pi_{i,1}$ and an initial onion is computed as $\{\rho_{i,1}\}_{PK_T}$. This is done independently for each row of the $P$ table and results in a $P_1$ table which is posted to the Bulletin Board. This table has rows of the form:

$$i, \{VC_{i,\pi_{i,1}(1)}\}'_{PK_T}, \ldots, \{VC_{i,\pi_{i,1}(c)}\}'_{PK_T}, \{VC_{i,c+1}\}_{PK_T}, \Theta_{i,1},$$

where the prime denotes re-encryption and:

$$\Theta_{i,1} := \{\rho_{i,1}\}_{PK_T}$$

The second Clerk now repeats this process on the $P_1$, *i.e.* for each row generates a fresh seed, computes the permutation from this seed, re-encrypts the vote code terms and shuffles them according to the permutation. The new seed is added as a further layer to the onion. Thus the $i$th row of the resulting $P_2$ table has the form:

$$i, \{VC_{i,\pi_{i,2}\circ\pi_{i,1}(1)}\}_{PK_T}, \ldots, \{VC_{i,\pi_{i,2}\circ\pi_{i,1}(c)}\}_{PK_T}, \{VC_{i,c+1}\}_{PK_T}, \Theta_{i,2},$$

where

$$\Theta_{i,2} := \{\rho_{i,2}, \Theta_{i,1}\}_{PK_T}$$

And so on for as many Clerks as are deemed appropriate. Tabulation will proceed as in [CRS05]. We omit the details here.

Alternatively, if we use a randomizing encryption such as ElGamal or Pallier we can use re-encryption mixes to construct the $Q$ table and to perform the tabulation. This is more flexible and robust but makes it harder to handle full permutations. A simple approach is to use just cyclic shifts of the vote codes, in the manner of [RS06a, Rya08]. Thus each Clerk takes the table output by the previous clerk and, for each row, generates a fresh seed $\phi \in Z_n$, re-encrypts the vote code terms and cyclically shifts them $\phi$ $(mod\ n)$ to the left. The new seed is folded into the onion exploiting the homomorphism of the encryption, which should be additive as in exponential ElGamal or Paillier. Thus, a row of the form:

$$i, \{VC_{i,j}\}_{PK_T}, \ldots, \{VC_{i,j}\}_{PK_T}, \{VC_{i,c+1}\}_{PK_T}, \Theta_{i,l},$$

is transformed to:

$$i, \{VC_{i,j+\phi\ (mod\ n)}\}'_{PK_T}, \ldots, \{VC_{i,j+\phi\ (mod\ n)}\}'_{PK_T}, \{VC_{i,c+1}\}_{PK_T}, \Theta_{i,2},$$

Where:

$$\Theta_{i,2} := \Theta_{i,1} \oplus \{\phi\}_{PK_T}$$

where $\oplus$ is the homomorphic operation on ciphertexts that adds the underlying plaintexts. The output of each step of the mix is posted to a Bulletin Board for subsequent audit.

Thus, the $\Theta_i$s are the usual Prêt à Voter style onions that encode the permutation $\pi_i$ of the vote codes within the $i$th row of the Q table with respect to the P table. For $c$ candidates the first $c$ cells contain a vote code encrypted under the (ElGamal or Paillier) Trustees' public key $PK_T$, permuted within the row according to $\pi_i$. The encrypted ack codes remain in their previous position in the $c+1$ column, followed by the $\Theta$ term.

Note that it is straightforward to introduce an extra onion that carried the definition of a full permutation in order to allow full permutation of the codes, in the manner of [Rya08]. In [Rya08], this resulted in the threat of "Italian" style attacks, in which an attacker could violate privacy by requiring the voter to record the candidate permutation and reveal it before the tabulation phase, so effectively identifying the ballot. In our context however, such attacks do not apply and hence it appears that this technique could be safely used.

### 5.5 Voting

In order to vote the voter logs in to the Voting Server, perhaps using some form of authentication. To cast a vote she simply enters the serial number of her code sheet and the vote code matching her candidate of choice. The information transmitted to the server consists of $i, VC_{i,j}$. The server knows no vote codes but might have a list of valid serial numbers. Votes with invalid serial numbers it will reject. For messages with valid serial numbers it computes $\{VC_{i,j}\}_{PK_T}$ and posts this to the $i$-th row of the Bulletin Board along with a zero knowledge proof ($ZKP$) of knowledge of the plaintext, [CP93, JJ00]. The proof is in order to avoid the server simply taking one of the encrypted codes and re-encrypting it before posting.

Next the Trustees perform a check of this proof of knowledge of the plaintext, and, if valid, they perform a *Plaintext-Equivalence-Tests* ($PET$), of the $\{VC_{i,j}\}_{PK_T}$ term posted by the server against the entries in row $i$. (See [JJ00] and [TH08] for descriptions of $PET$ tests on El Gamal and Paillier ciphertexts, respectively.) In essence, in order to find the required match within the permuted encryptions the following test is performed: Two encryptions $(\alpha, \beta)$ and $(\alpha', \beta')$ are equivalent (encrypt the same plaintext) if the decryption of $(\alpha/\alpha', \beta/\beta')^r$ outputs 1. $r$ is a randomsing factor that all

the Trustees contribute to that serves to disguise the ratio of the underlying plaintexts, in the event that they are not equal. Where the Trustees find a match the relevant cell is thus flagged and they collectively decrypt the final Ack Code term. The Voting Server can now return the Ack code to the voter who can now check it against the value on their code sheet. The voter can of course later visit the Bulletin Board to confirm that their vote has been logged in the appropriate row. They cannot verify directly that their choice of candidate was correctly recorded due to the secret permutation of the positions of the codes posted on the Bulletin Board.

Note that the Server's proof of knowledge of the plaintext and the Trustees' proofs of plaintext equivalence are all posted alongside the appropriate row on the Bulletin Board. The Server's proof of plaintext knowledge counters ballot stuffing attacks in which a corrupt Voting Server simply posts a re-encryption of a randomly selected Vote Code from the appropriate row. (Though obviously it does not prevent a Voting Server who somehow learns valid Vote Codes from submitting them—see Section 8 for further discussion of this.) The Trustees' proofs of plaintext equivalence prove that the vote is being counted as it was cast, because it is being matched to the correct index on the ballot.

The full protocol sequence can be seen in Table 1.

The first five steps constitute the setup phase. In step five the Returning Officer distributes a code sheet to each voter via a supposedly secure channel such as conventional mail, denoted by $\Rightarrow$. Note that the only steps that involve the voter, aside from receiving the code sheet in step 5, are 7 and 10, and these both occur in the same session.

### 5.6 Tabulation

On the Bulletin Board, each row corresponding to a successfully cast ballot, there will be a flagged cell along with zero knowledge proofs and Plaintext Equivalence Tests. The index of the column in which this cell has been flagged is noted. Thus, for each row that corresponds to a successfully cast ballot, an onion and and an index value are extracted giving the usual Prêt à Voter style (index, onion) pair. The onion defines the ordered list of candidates from which the index was selected. These pairs can be tabulated with mixnets in the usual fashion. For full details we refer the reader to [CRS05, RS06a, Rya08].

| | | |
|---|---|---|
| 1. | Voting Authority → Bulletin Board | $\lambda v(c+1)$ encrypted codes |
| 2. | Clerks → Bulletin Board | shuffling of codes to produce the P table. |
| 3. | Trustees → Registrar | decrypted codes |
| 4. | Registrar → Returning officer | printed code sheets |
| 5. | Returning Officer ⇒ $Voter_i$: | $i$, $VC_{i,j}$ for $j = 1 \ldots c$, and $Ack_i$ |
| 7. | $Voter_i$ → Voting Server: | $i$, $VC_{i,j}$ for chosen candidate |
| 8. | Voting Server → $BulletinBoard$: | $i$, $\{VC_{i,j}\}_{PK_T}$, proof of plaintext knowledge |
| 9. | $Trustees$ → $BulletinBoard$: | flag Cell $(i,j)$, jointly decrypt $Ack_i$ |
| 10. | Voting Server → $Voter_i$: | $Ack_i$ |

Table 1: Protocol sequence

For completeness we outline the tabulation process for the cyclic shifts construction. Suppose that the cumulative cyclic shift applied to the $i$th row is $s$ steps to the left. Suppose that the $h$ cell is flagged. This means that the voter actually selected the $(s+h)$-th candidate in the base ordering. Thus, we transform the pair:

$$(h, \{s\}_{PK_T})$$

into:

$$\{h\}_{PK_T} \oplus \{s\}_{PK_T} = \{s+h\}_{PK_T}$$

The resulting cyphertext can now be put through a standard, robust, re-encryption mix and will decrypt to $h + s$, which taken *mod* $c$ gives the voter's original choice.

# 6 Auditing

In this section we outline the auditing procedures that serve to detect any malfunction or corruption during the setup and tabulation phases that could lead to an incorrect outcome.

### 6.1 Auditing the Election Setup

It is essential that the code sheets distributed to voters are consistent with the information posted to Q table on the Bulletin Board. More precisely, we need to check that the set of codes shown on any code sheet agrees with the codes buried in the encrypted terms on the corresponding row of the Q table posted to the Bulletin Board. Furthermore, the permutation of the codes on the Bulletin Board with respect to those on the code sheet should match the permutation encoded in the onion posted to that row. To address this we perform random checks on a suitable proportion of the code sheets. Prior to the election Auditors pick a random set of code sheets. For selected code sheets, a threshold set of Trustees are invoked to decrypt the vote code terms and the onion. The Trustees are required to provide proofs of these decryptions. With this information, the consistency of the selected code sheets and corresponding rows of the Q table can be verified.

If we randomly audit say half the code sheets and all of these pass the checks, then we can be confident that the remaining code sheets will also be consistent, assuming that there is no way to predict the audit selections. We must ensure that all audited code sheets be removed from the election. It must not be possible to cast a vote using an audited code sheet.

There are the usual chain of custody issues: we need to ensure that false code sheets cannot be substituted after auditing. Careful procedures can make this difficult. Also, we can randomly audit forms just prior to their being mailed out.

An alternative way to check consistency for a randomly selected set of code sheets is to require the Trustees to transform the code vote terms in each row according to the permutation in the onion on the Bulletin Board in a verifiable fashion and then decrypt the terms. These should agree with the sequence of vote codes on the code sheet.

### 6.2 Auditing the Tabulation

All the Zero Knowledge proofs posted to the Bulletin Board during the registering of votes should be publicly audited. This serves to counter attempts to corrupt votes by shifting the flagged terms. It also serves to counter ballot stuffing, *i.e.* simply flagging terms in rows corresponding to code sheets that were never used to cast a vote. An adversary attempting such ballot

stuffing would have to construct corresponding zero knowledge proofs which could only be done with collusion with a threshold set of Trustees.

To detect any inaccuracy caused by the Talliers we employ standard mechanisms for auditing mixes and decryptions. The batch of index, onion pairs are put through anonymising mixes and then decrypted. All intermediate steps are posted to the Bulletin Board allowing verification, either via Random Partial Checking, [JJR02], or other techniques such as Neff's verifiable shuffles, [Nef01].

# 7 Error Recovery

## 7.1 Incorrect Ack codes

So far we have described the unfolding of the protocol and outlined techniques to detect errors or corruption. Clearly we need to set up procedures for the case in which all does not go according to plan. In particular, we need to consider the situation in which the $Ack_i$ returned by the server does not match with the ack code that appears on the voter's code sheet.

Certainly a number of voters who claim faulty ack codes raises doubts about the Vote Server. If the voting system relies on the availability of several servers (in fact, each Trustee could offer a voting service allowing the voters to choose) the discredited server is either defective or malicious and should be taken out of service. Additional information regarding recovery mechanisms for Prêt à Voter can be found in [BLRS06].

## 7.2 Multiple voting

We need to clarify what happens when the Vote Server receives a second submission that purports to be from the same voter as one already received. One reasonable approach would be to run the PET tests privately (*i.e.* without posting them on the bulletin board) and tabulate only those submitted codes that matched a correct one, then count only one. Nevertheless, multiple valid Vote Code submissions for the same ID will still occur. This indicates malicious behaviour by someone who knows the codes, either an adversary to whom the information was leaked, or a voter trying to discredit the process. Perhaps the most reasonable approach is to publish on the Bulletin Board the first two valid submissions from each ID, but to include only

the first one in the count. The Vote Server would be supposed to return the ack only to the first submitter. This does give an adversary a way of nullifying a person's vote, but only if she learns the Vote Codes beforehand, and it is detectable by the voter if either he looks at the Bulletin Board or he notices the absence of the return ack (which he might not if the adversary successfully sends it to him). It is unclear how to make this interact seamlessly with revoting techniques such as tokens.

# 8 A Threat Model

For the threat model we concentrate on malicious behaviour of the participating parties rather than outsiders, and on dangers that are distinct from other schemes. Principle dangers considered are undetected compromise of election integrity, compromise of ballot secrecy and denial of voting service.

## 8.1 Leaking of Codes

It appears that the most serious threat to the scheme is the possibility of information about vote codes leaking to the Vote Server. If the Vote Server gets to know alternative codes for a particular code sheet it can post an encryption of a code of its choice rather than the voter's choice to the Bulletin Board. The most likely scenario for such leakage is a malicious Registrar leaking codes to the Vote Server.

A number of counter-measures are possible, some of which we discuss in greater detail in section 9. One possibility is ensure that the Registrar does not know the association of the serial number with the set of codes. We might print a set of code sheets bearing only the candidate names and serial numbers and cover these with a scratch strip. Overprinted on the scratch strip would be another serial number that is related to the "real" serial number by a secret look-up table. Now, to print the codes, the Registrar would provide the overprinted code to a Custodian who would have access to the look-up table. The Custodian passes the "true" serial number to a threshold set of the Trustees who decrypt the codes on the appropriate row of the the P table and pass these back to the Registrar. Note that we can vary the set of Trustees used for each code and so ensure that no Trustee knows more than one code associated with a given serial number. To cast her vote, the voter needs to scratch of the strip and reveal the "true" serial number.

### 8.2 Vote Server

The Vote Server acting alone might try to cheat by trying to guess alternative, valid vote codes for a given vote represented by the identification value $i$. If successful, this might threaten the integrity of the vote. The code length can be adjusted to make this is rather difficult and alerts can be raised if a Vote Server repeatedly posts invalid codes. This has to be balanced against the discrediting attacks that voters might try to launch, see below.

### 8.3 Trustees - Vote Server

A colluding threshold set of Trustees can undermine the accuracy by, for example, decrypting alternative codes and then constructing the zero knowledge proofs for this alternative code rather than a code submitted by the voter, then returning the correct ack code to the server. As long the Trustees are drawn from a pool of mutual distrustful entities, we can ensure that such a collusion is unlikely.

Furthermore, we can raise the collusion threshold: we arrange for the set of trustees used for tabulation is disjoint from the trustees used to register vote codes, and these have different public keys. Now the onions are encrypted under the public key of the tabulation trustees. This means that the registration trustees alone will not know the meaning of any particular code. Thus, even if the are in collusion, the best they can do is launch a randomizing attack. For pure cyclic shifts of the codes such an attack could be quite effective, as observed in [RT]: ballots could be shifted from a popular candidate to a chosen candidate by an appropriate shift of the index value. In this case, using an additional onion to carry a full permutation, or the affine permutation techniques of [RT] will counter this.

### 8.4 Voters Undermining the Credibility of the Vote Server

A rather different style of threat is that of a group of voters attempting to undermine the credibility of the system by deliberately submitting fake codes. This would result in the Vote Server posting incorrect codes and so being suspected of attempting to guess alternative codes. Clearly, since the security of our scheme rests on the Vote Server not knowing the contents of code sheets we cannot simply counter this be enabling the Vote Server

to recognise codes that are not valid for any given code sheet. A possible counter-measure is to arrange for the entire space of codes to be rather sparse and give the Vote Server knowledge of the full set, but not of course any knowledge of how they are grouped into code sheets. This will allow the Vote Server to recognise attempts to submit fake codes. It does mean that the initial construction of the batch of codes and the P table cannot be done completely publicly on the Bulletin Board. This does not seem to matter however as the Q table will in any case be audited later

### 8.5 Summary

We have discussed a number of threats against the scheme, the most serious of which is the leakage of information about the codes. This threatens not only the privacy but also the integrity of the election. The fact that the integrity guarantees depend on secrecy guarantees does mean that the level of assurance provided by such a scheme is significantly less than that provided by other verifiable schemes.

Balanced against this is the fact that vote casting and verification is significantly more convenient, both steps occurring during a single voting session. In effect the scheme is *vote, verify and go.*

## 9 Further Enhancements

The proposed mechanism for releasing the ack codes only if a threshold set of Trustees register a valid vote code gives a good level of assurance that, if the voter gets the correct ack code back then their vote will be correctly registered. Further mechanisms then ensure that correctly registered codes will be correctly decrypted and tabulated. Nonetheless the threat model has shown some weak points that require further discussion. In this section we discuss some possible enhancements that help counter this threat.

### 9.1 Encrypting messages to the Vote Server

At the cost of introducing the capability of performing encryption at the client side, we can strengthen the protocol as follows. Voters cast a ballot by sending a Vote Code as well as a Ballot ID, both encrypted under the Trustee key, to the Vote Server. The server passes $\{i\}_{PK_T}$ and $\{VC_{i,j}\}_{PK_T}$

to the Trustees. After a threshold decryption of $\{i\}_{PK_T}$ the Trustees again use a $PET$ to find the valid match for the received $\{VC_{i,j}\}_{PK_T}$ in row $i$ of the Bulletin Board. Now, even if the Vote Server has somehow learnt which Vote Codes correspond to which IDs, it cannot identify which values to substitute in which messages.

We have to assume here that the attacker can't link the packet to its ID via some other channel, such as IP address. To see why, suppose also that the attacker can intercept the voter's communications and knows the voter's codes, that we're using a homomorphic cipher and that the attacker can make a good guess as to how the voter will vote. (This last assumption is a strong one, but still fair in some circumstances, such as if the attacker lives in the same household.) Suppose the voter chose VC but the attacker wants VC'. Then the attacker can just use the homomorphism to add VC'-VC to the ciphertext as it relays it.

### 9.2 Adding salts to codes

Another possible mechanism to counter the fact that the Registrar is a single point of failure for the secrecy of the codes is to use code sheets that require the voter to execute an easy arithmetic operation in order to get a hold of the true vote code. This would work best if the encryption scheme provided a corresponding homomorphic operation. For example, we could ask voters to add the salt to their Vote Code, then the trustees could add the encrypted salt to the encrypted Vote Codes on the bulletin board with an additive-homomorphic encryption scheme. It might be effective, yet simple enough, to let the voter calculate the sum of a received code and a digit of the birthdate, e.g. a two digit representation of the month. The voter could use their computer to check (or do) the addition for them (and the extent to which this constitutes "trusting the computer" would then depend on the voter). The salt is supposed to be somewhat-private data about the voter. The idea is to increase the difficulty for an attacker who learns a voter's Vote Codes. Such an attacker will now have to guess the salt in order to launch an attack. We add another authority, the Salt Authority, who knows some somewhat-private data about everyone on the electoral roll. For example, this could be whichever authority is trusted to maintain the electoral roll, and they could use birthdates or house numbers as the salt.

In this section we assume that it is *not* a secret which IDs went to which voters.

The Salt Authority learns from the Registration Authority which IDs were sent to which voters. It then posts to the bulletin board an encrypted salt $\{S_i\}_{PK_T}$ for each $i$. At voting time, the voter adds their chosen Vote Code to their salt (which they are supposed to know in advance) and sends the sum to the Voting Server. The trustees use the homomorphism to add $\{S_i\}_{PK_T}$ to the codes in row $i$ before performing the PET.

We could even have several different Salt Authorities and allow voters to choose which salts they would like to add.

This scheme does not protect against a malicious Vote Server, or even an adversary who can intercept communications to the Vote Server, if that party knows the codes—such an adversary can perform a substitution attack very similar to the one described in Section 9.1. It does, however, make it harder for a third party who learns the codes to simply submit them to the Vote Server. For example, just stealing someone's code sheet from their mailbox does not allow an attacker to steal their vote.

### 9.3 Distinct Acknowledgement Codes Revisited

A further possibility is to re-introduce the the idea of distinct ack codes for each candidate, except that now we generate and distribute these codes independently from the vote codes. A simple way to achieve this is to construct a $P$ table much as before except that now each entry is a pair of encryptions of codes: one vote code one ack code. Note that the table does not now need a $(c+1)$th column for the ack codes. We transform this into a $Q$ table as before: for each row permuting the cells of the $P$ table and storing the permutation in an onion, but keeping the vote/ack codes paired.

Now we have a Vote Code Registrar and an Ack Code Registrar. The former will decrypt the vote codes (with the cooperation of a threshold set of tellers) and print the vote code sheets. The latter similarly will decode the ack codes and print the ack code sheets. Matching vote and ack code sheets are mailed to each voter. This does mean that a record needs to be kept of which sheets go to which voter. Here again, we could use the idea of a pseudo-serial number overprinted on a scratch strip to counter threats to ballot privacy that might arise as a result of recording this information.

Now, when a vote is registered, *i.e.* the PET performed by the Trustees finds a match, the cell of the $Q$ table is flagged as before. The Trustees decrypt the Ack code and pass this in secret to the Voter Server, who can

then relay it to the Voter. The ack code is not posted to the Bulletin Board as this would violate receipt freeness.

This idea would work best if the vote codes and ack codes could be distributed independently, in such a way that no single entity knew both of them.

## 10 Conclusions

We have presented a simple enhancement to code voting that adds a degree of end-to-end verification with the voter needing only to confirm that the ack code returned by the Voting Server agrees with that on their code sheet. The scheme is receipt-free, in the sense that the voter cannot prove to a coercer how they voted. As it stands it is not resistant to direct coercion, where the coercer is present at or before the time of casting. To counter such direct coercion would require additional mechanisms such as the mentioned *re-voting* or the use of a form of *voting token*.

Arguably, the verification performed by the voters is simpler and more direct than with previous voter-verifiable schemes: all the voter needs to do is check that the ack code returned to them agrees with that shown on their code sheet.

The scheme sacrifices a little in terms of verifiability but gains significantly in terms of usability. Most other cryptographic schemes provide a stricter form of verifiability in that any corruption in the processing of votes will be detectable with an appropriate probability. Typically the probability of detection will grow, ideally exponentially, with the scale of the corruption. In PGD, leakage of information about vote codes could undermine the integrity in a way that would be undetectable. We have proposed a number of counter-measures to make such leakage unlikely, but the guarantees of integrity are still dependant on certain trust assumptions. The question then is whether the trade-off of the high degree of convenience of the scheme, the voter casts and verifies in a single session, against the sacrifice of unconditional guarantees of integrity is valid and in what contexts. We suggest that for certain types of elections, for example of officers of a professional organisation, that the trade-off would be valid. We would not propose the use of PGD for binding, political elections.

PGD requires a threshold number of Trustees to be online during the voting period to decrypt and return the ack codes. Although this does

affect the system's reliability, it doesn't affect its integrity. Votes could still be posted on the Bulletin Board if fewer than the threshold were available, they just couldn't be decrypted or acknowledged until the necessary number of Trustees returned.

A topic for future research is how to distribute code sheet information on-line (or possibly in print, extending the techniques of [ECHA09]) so that only a large scale collusion could violate Vote Code secrecy. The goal is to ensure that the decrypted codes are only ever revealed to the intended voter. A further topic that requires further investigation is how to recover from erroneous or malicious behaviour by components of the system, including voters.

## References


[Adi06] B. Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, MIT Cambridge, July 2006.

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

[AdMPQ09] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean Jacques Quisquater. Electing a university president using open audit voting: Analysis of real world use of Helios. In *EVT'09: Proceedings of the electronic voting technology workshop*, 2009.

[Ben06] Josh Benaloh. Simple verifiable elections. In *Proc. 1st USENIX Accurate Electronic Voting Technology Workshop*, 2006.

[BLRS06] J.W. Bryans, B. Littlewood, P.Y.A. Ryan, and L. Strigini. E-voting: Dependability Requirements and Design for Dependability. Proceedings of the First International Conference on Availability, Reliability and Security (ARES06), 2006.

[CCC+08] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT'08: Proceedings of the electronic voting technology workshop*, pages 1–13, Berkeley, CA, USA, 2008. USENIX Association.

[CCM07] Michael Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: A secure remote voting system. Technical report, Cornell University Computing and Information Science Technology Report, May 2007.

[CGJ+99] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive Security for Threshold Cryptosystems. Advances in Cryptology - CRYPTO'99, 1999.

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology*, number 1233 in Lecture Notes in Computer Science, pages 103–118. Springer-Verlag, 1997.

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

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 89–105, London, UK, 1993. Springer-Verlag.

[CRS05] D. Chaum, P.Y.A. Ryan, and S. Schneider. A practical, voter-verifiable election scheme. In *European Symposium on Research in Computer Security*, number 3679 in Lecture Notes in Computer Science. Springer-Verlag, 2005.

[ECHA09] Aleks Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams. How to print a secret. In *Proc. 4th USENIX workshop on hot topics in security (HotSec 09)*, 2009. To appear.

[HSS08] J. Helbach, J. Schwenk, and S. Schräge. Code Voting with Linkable Group Signatures. Proceedings of Electronic Voting 2008, 2008.

[JCJ05] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant Electronic Elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61–70, 11 2005.

[JJ00] M. Jakobsson and A. Juels. Mix and Match: Secure Function Evaluation via Ciphertexts, 2000.

[JJR02] M. Jakobsson, A. Juels, and Ronald Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *USENIX Security Symposium*, pages 339–353, 2002.

[Kri07] Taisya Krivoruchko. Robust coercion-resistant registration for remote e-voting. In *Proc. IAVoSS Workshop on Trustworthy Elections (WOTE)*, 2007.

[Nef01] A. Neff. A verifiable secret shuffle and its application to e-voting. In *Conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[OSH08] R. Oppliger, J. Schwenk, and J. Helbach. Protecting Code Voting Against Vote Selling. Lecture Notes in Informatics - Sicherheit, Schutz und Zuverlässigkeit (Sicherheit 2008), 2008.

[Pet05] R.A. Peters. A Secure Bulletin Booard. Master's thesis, Technische Universiteit Eindhoven, 2005.

[RS06a] P.Y.A. Ryan and S. Schneider. Prêt à Voter with Re-encryption Mixes. In *European Symposium on Research in Computer Security*, number 4189 in Lecture Notes in Computer Science. Springer-Verlag, 2006.

[RS06b] P.Y.A. Ryan and S. A. Schneider. Prêt à voter with re-encryption mixes. Technical Report CS-TR-956, University of Newcastle upon Tyne, 2006.

[RT] P.Y.A. Ryan and V Teague. Ballot permutations in Prêt à Voter. In *Seventeenth International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer-Verlag. To appear.

[Rya07] P.Y.A. Ryan. The computer ate my vote. Technical Report CS-TR-988, University of Newcastle upon Tyne, 2007.

[Rya08] Peter Y. A. Ryan. Prêt à Voter with Paillier encryption. *Mathematical and Computer Modelling*, 48(9-10):1646–1662, November 2008.

[TH08] Pei-Yih Ting and Xiao-Wei Huang. Distributed Paillier plaintext equivalence test. *International Journal of Network Security*, 6(3):258–264, 2008. http://ijns.femto.com.tw/contents/ijns-v6-n3/ijns-v6-n3.html.

[VG06] M. Volkamer and R. Grimm. Multiple Casts in Online Voting: Analyzing Chances. Proceedings of Electronic Voting 2006, 2006.